CONTENTS

TUTORIAL 1

Drawing Application

Introducing Computers, the Internet and C#

Solutions

Instructor's Manual Exercise Solutions Tutorial 1

MULTIPLE-CHOICE QUESTIONS

1.1 The HyperText Markup Language was developed _____.

a) by ARPA
b) at CERN by Tim Berners-Lee
c) before the Internet
d) as a replacement for the Internet

1.2 Microsoft's _____ initiative integrates the Internet and the Web into software development.

a) .NET
b) BASIC
c) Windows
d) W3C

1.3 `TextBoxes`, `Buttons` and `RadioButtons` are examples of _____.

a) platforms
b) high-level languages
c) IDEs
d) controls

1.4 _____ is an example of primary memory.

a) TCP
b) RAM
c) ALU
d) CD-ROM

1.5 C# is an example of a(n) _____ language, in which single program statements accomplish more substantial tasks.

a) machine
b) intermediate-level
c) high-level
d) assembly

1.6 Which protocol is primarily intended to create a "network of networks?"

a) TCP
b) IP
c) OOP
d) FCL

1.7 A major benefit of _____ programming is that it produces software that is more understandable and better organized than software produced with previously used techniques.

a) object-oriented
b) centralized
c) procedural
d) HTML

1.8 .NET's collection of prepackaged classes and methods is called the _____.

a) NCL
b) WCL
c) FCL
d) PPCM

1.9 The information-carrying capacity of communications lines is called _____.

a) networking
b) secondary storage
c) traffic
d) bandwidth

1.10 Which of these programming languages was specifically created for .NET?

a) C#
b) C++
c) BASIC
d) Visual Basic

Answers: 1.1) b. 1.2) a. 1.3) d. 1.4) b. 1.5) c. 1.6) b. 1.7) a. 1.8) c. 1.9) d. 1.10) a.

EXERCISES

1.11 Categorize each of the following items as either hardware or software:

a) CPU
b) Compiler
c) Input unit
d) A word-processor program
e) A C# program

Answers: a) hardware. b) software. c) hardware. d) software. e) software.

1.12 Translator programs, such as assemblers and compilers, convert programs from one language (referred to as the source language) to another language (referred to as the target language). Determine which of the following statements are *true* and which are *false*:

a) A compiler translates high-level-language programs into target-language programs.

b) An assembler translates source-language programs into machine-language programs.

c) A compiler translates source-language programs into target-language programs.

d) Machine languages are generally machine independent.

e) A machine-language program requires translation before it can be run on a computer.

Answers: a) True. b) True. c) True. d) False. Machine languages are generally machine dependent. e) False. A machine language program is native to that specific machine and can be run without translation.

1.13 Computers can be thought of as being divided into six units.

a) Which unit can be thought of as "the boss" of the other units?

b) Which unit is the high-capacity "warehouse" that retains information even when the computer is powered off?

c) Which unit might determine whether two items stored in memory are identical?

d) Which unit obtains information from devices like the keyboard and mouse?

Answers: a) CPU. b) Secondary storage unit. c) ALU. d) Input unit.

1.14 Expand each of the following acronyms:

a) W3C

b) TCP/IP

c) OOP

d) FCL

e) HTML

Answers: a) World Wide Web Consortium. b) Transmission Control Protocol/Internet Protocol. c) Object-oriented programming. d) Framework Class Library. e) HyperText Markup Language.

1.15 What are the advantages to using object-oriented programming techniques?

Answer: Programs that use object-oriented programming techniques are easier to understand, correct and modify. The key advantage with using object-oriented programming is that it tends to produce software that is more understandable because it is better organized and has fewer maintenance requirements than software produced with earlier methodologies. OOP helps the programmer build applications faster by reusing existing software components. OOP also helps programmers create new software components that can be reused on future software-development projects.

2 TUTORIAL

Welcome Application

Introducing the Visual Studio® .NET IDE

Solutions

Instructor's Manual Exercise Solutions Tutorial 2

MULTIPLE-CHOICE QUESTIONS

2.1 The _____ integrated development environment (IDE) is used for creating applications written in .NET programming languages such as C#.

a) **Solution Explorer** b) Gates
c) Visual Studio .NET d) Microsoft

2.2 The `.cs` filename extension indicates a _____.

a) C# file b) dynamic help file
c) help file d) cool solution file

2.3 The pictures on toolbar `Button`s are called _____.

a) prototypes b) icons
c) tool tips d) tabs

2.4 The _____ allows programmers to modify controls visually, without writing code.

a) **Properties** window b) **Solution Explorer**
c) menu bar d) **Toolbox**

2.5 The _____ hides the **Toolbox** when the mouse pointer is moved outside the **Toolbox**'s area.

a) component-selection feature b) Auto Hide feature
c) pinned command d) minimize command

2.6 A _____ appears when the mouse pointer is positioned over an IDE toolbar icon for a few seconds.

a) drop-down list b) menu
c) tool tip d) down arrow

2.7 The Visual Studio .NET IDE provides _____.

a) help documentation b) a toolbar
c) windows for accessing project files d) All of the above.

2.8 The _____ contains a list of helpful links, such as **Get Started** and **Online Community**.

a) **Solution Explorer** window b) **Properties** window
c) **Start Page** d) **Toolbox** link

2.9 The **Properties** window contains _____.

a) the component object box b) a **Solution Explorer**
c) menus d) a menu bar

2.10 A _____ can be enhanced by adding reusable controls such as `Button`s.

a) component b) `Form`
c) icon d) property

2.11 For Web browsing, Visual Studio .NET includes _____.

a) Web View b) Excel
c) a **Web** tab d) Internet Explorer

2.12 An application's GUI can include _____.

a) toolbars b) icons
c) menus d) All of the above.

2.13 The _____ does not contain a pin icon.

a) **Dynamic Help** window
b) **Solution Explorer** window
c) **Toolbox** window
d) active tab

2.14 When clicked, _____ in the **Solution Explorer** window will expand nodes and _____ will collapse nodes.

a) minus boxes; plus boxes
b) plus boxes; minus boxes
c) up arrows; down arrows
d) left arrows; right arrows

2.15 `Form` _____ specify attributes such as size and position.

a) nodes
b) inputs
c) properties
d) title bars

Answers: 2.1) c. 2.2) a. 2.3) b. 2.4) a. 2.5) b. 2.6) c. 2.7) d. 2.8) c. 2.9) a. 2.10) b. 2.11) d. 2.12) d. 2.13) d. 2.14) b. 2.15) c.

EXERCISES

2.16 (***Closing and Opening the Start Page***) In this exercise, you will learn how to close and reopen the **Start Page** (Fig. 2.30). To accomplish this task, perform the following steps:

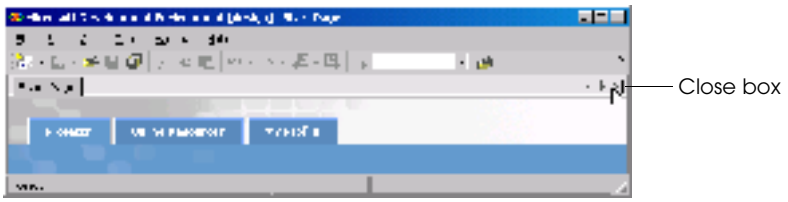


Figure 2.30 Showing the **Start Page**.

a) Close Visual Studio .NET if it is open by clicking its close box.
b) Start Visual Studio .NET.
c) Close the **Start Page** by clicking its close box (Fig. 2.30).
d) Select **Help > Show Start Page** to display the **Start Page**.

2.17 (***Enabling Auto Hide for the Solution Explorer Window***) In this exercise, you will learn how to use the **Solution Explorer** window's Auto Hide feature (Fig. 2.31) by performing the following steps:

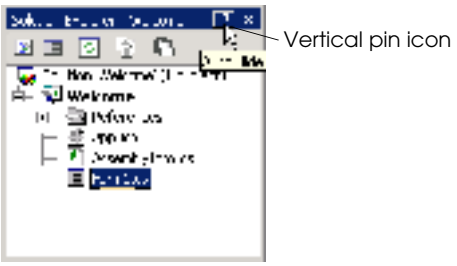


Figure 2.31 Enabling Auto Hide.

a) Open the **Start Page**.
b) In the **Projects** tab (displayed by default), click the **Open Project** `Button` to display the **Open Project** dialog. You can skip to *Step e* if the **Welcome** application is already open.
c) In the **Open Project** dialog, navigate to `C:\SimplyCSP\Welcome`, and click **Open**.
d) In the **Open Project** dialog, select `Welcome.sln`, and click **Open**.
e) Position the mouse pointer on the vertical pin icon in the **Solution Explorer** window's title bar. After a few seconds, a tool tip appears displaying the words **Auto Hide**.

f) Click the vertical pin icon. This action causes a **Solution Explorer** tab to appear on the right side of the IDE. The vertical pin icon changes to a horizontal pin icon (Fig. 2.32). Auto Hide has now been enabled for the **Solution Explorer** window.

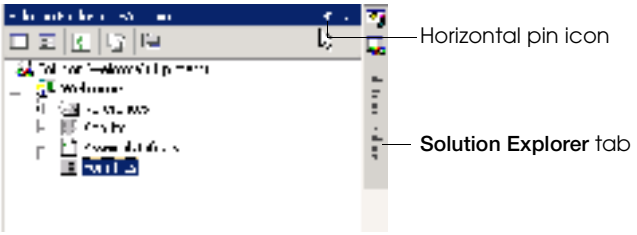


Figure 2.32 **Solution Explorer** window with Auto Hide enabled.

g) Position the mouse pointer outside the **Solution Explorer** window to hide the window.

h) Position the mouse pointer on the **Solution Explorer** tab to view the **Solution Explorer** window.

2.18 **(*Sorting Properties Alphabetically in the Properties Window*)** In this exercise, you will learn how to sort the **Properties** window's properties alphabetically (Fig. 2.33) by performing the following steps:

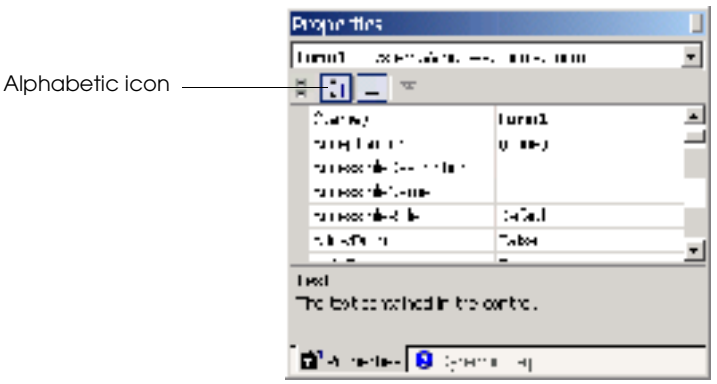


Figure 2.33 Sorting properties alphabetically.

a) Open the **Welcome** application by performing *Steps a–d* of Exercise 2.17. If the **Welcome** application is already open, you can skip this step.

b) Locate the **Properties** window. If it is not visible, select **View > Properties Window** to display the **Properties** window.

c) To sort properties alphabetically, click the **Properties** window's alphabetic icon. The properties will display in alphabetic order.

TUTORIAL 3

Welcome Application

Introduction to Visual Programming

Solutions

Instructor's Manual Exercise Solutions Tutorial 3

MULTIPLE-CHOICE QUESTIONS

3.1 The _____ property determines the `Form`'s background color.

a) `BackColor` b) `BackgroundColor`
c) `RGB` d) `Color`

3.2 To save all the solution's files, select _____.

a) **Save > Solution > Save Files** b) **File > Save**
c) **File > Save All** d) **File > Save As...**

3.3 When the ellipsis `Button` to the right of the **Font** property value is clicked, the _____ is displayed.

a) **Font Property** dialog b) **New Font** dialog
c) **Font Settings** dialog d) **Font** dialog

3.4 `PictureBox` property _____ contains a preview of the image displayed in the `PictureBox`.

a) `Picture` b) `ImageName`
c) `Image` d) `PictureName`

3.5 The _____ tab allows you to create your own color.

a) **Custom** b) **Web**
c) **System** d) **User**

3.6 The `PictureBox` class belongs to the _____ namespace.

a) `System.Windows.Forms` b) `System.Form.Form`
c) `System.Form.Font` d) `System.Form.Control`

3.7 A `Label` control displays the text specified by the _____ property.

a) `Caption` b) `Data`
c) `Text` d) `Name`

3.8 In _____ mode, the application is running (executing).

a) start b) run
c) execute d) design

3.9 The _____ command prevents programmers from accidentally altering the size and location of the `Form`'s controls.

a) **Lock Controls** b) **Anchor Controls**
c) **Lock** d) **Bind Controls**

3.10 Pixels are _____.

a) picture elements b) controls in the **Toolbox**
c) a set of fonts d) a set of colors on the **Web** tab

Answers: 3.1) a. 3.2) c. 3.3) d. 3.4) c. 3.5) a. 3.6) a. 3.7) c. 3.8) b. 3.9) a. 3.10) a.

EXERCISES

For Exercises 3.11–3.16, you are asked to create the GUI shown in each exercise. You will use the visual programming techniques presented in this tutorial to create a variety of GUIs. Because you are creating only GUIs, your applications will not be fully operational. For example, the **Calculator** *GUI in Exercise 3.11 will not behave like a calculator when its* `Buttons` *are clicked. You will learn how to make your applications fully operational in later tutorials. Create each application as a separate project.*

3.11 ***(Calculator GUI)*** Create the GUI for the calculator shown in Fig. 3.33.

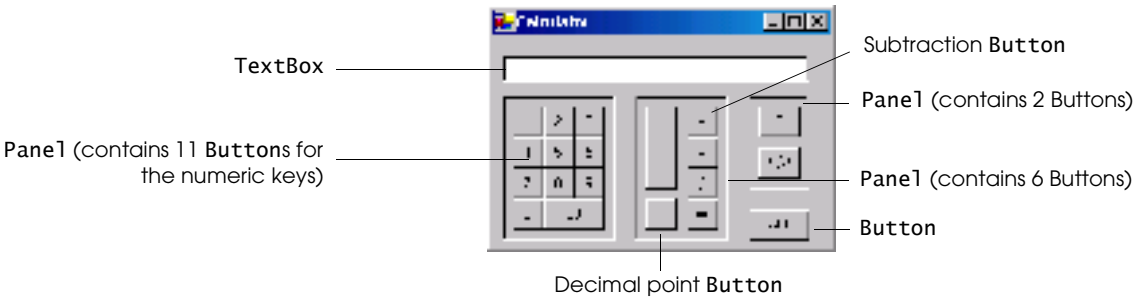
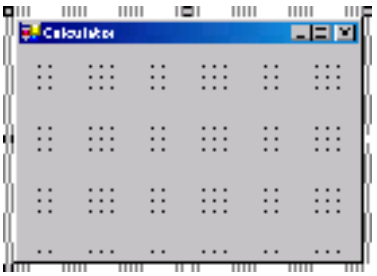


Figure 3.33 **Calculator** GUI.

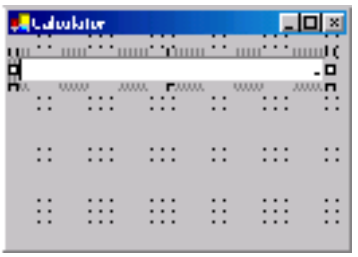
a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your `C:\SimplyCSP` directory named `Calculator`.

b) ***Renaming the Form file.*** Name the `Form` file `Calculator.cs`. Double click the file name to open the `Form` in design view.

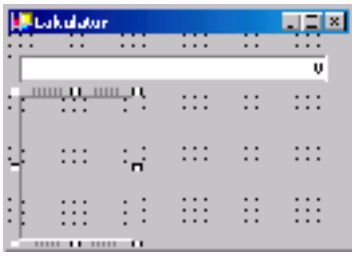
c) ***Manipulating the Form's properties.*** Change the `Size` property of the `Form` to `272, 192`. Change the `Text` property of the `Form` to `Calculator`. Change the `Font` property to **Tahoma**.



d) ***Adding a TextBox to the Form.*** Add a `TextBox` control by double clicking it in the **Toolbox**. A `TextBox` control is used to enter input into applications. Set the `TextBox`'s `Text` property in the **Properties** window to `0`. Change the `Size` property to `240, 21`. Set the `TextAlign` property to `Right`; this right aligns text displayed in the `TextBox`. Finally, set the `TextBox`'s `Location` property to `8, 16`.

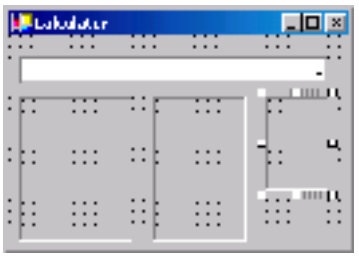


e) ***Adding the first Panel to the Form.*** `Panel` controls are used to group other controls. Double click the `Panel` icon ( ) in the **Toolbox** to add a `Panel` to the `Form`. Change the `Panel`'s `BorderStyle` property to `Fixed3D` to make the inside of the `Panel` appear recessed. Change the `Size` property to `88, 112`. Finally, set the `Location` property to `8, 48`. This `Panel` contains the calculator's numeric keys.

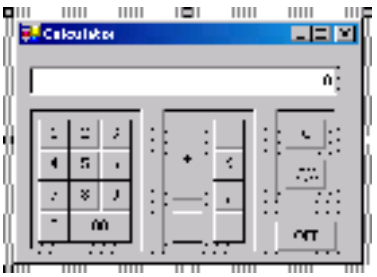


f) ***Adding the second Panel to the Form.*** Click the `Form`. Double click the `Panel` icon in the **Toolbox** to add another `Panel` to the `Form`. Change the `Panel`'s `BorderStyle` property to `Fixed3D`. Change the `Size` property to `72, 112`. Finally, set the `Location` property to `112, 48`. This `Panel` contains the calculator's operator keys.

g) ***Adding the third (and last) Panel to the Form.*** Click the `Form`. Double click the `Panel` icon in the **Toolbox** to add another `Panel` to the `Form`. Change the `Panel`'s `BorderStyle` property to `Fixed3D`. Change the `Size` property to `48, 72`. Finally, set the `Location` property to `200, 48`. This `Panel` contains the calculator's **C** (clear) and **C/A** (clear all) keys.



h) ***Adding Buttons to the Form.*** There are 20 `Button`s on the calculator. To add a `Button` to a `Panel`, double click the `Button` control ( ) in the **Toolbox**. Then add the `Button` to the `Panel` by dragging and dropping it on the `Panel`. Change the `Text` property of each `Button` to the calculator key it represents. The value you enter in the `Text` property will appear on the face of the `Button`. Finally, resize the `Button`s, using their `Size` properties. Each `Button` labeled `0`–`9`, `x`, `/`, `-`, `=` and `.` should have a size of `24, 24`. The **00** and **OFF** `Button`s have size `48, 24`. The **+** `Button` is sized `24, 64`. The **C** (clear) and **C/A** (clear all) `Button`s are sized `32, 24`.



i) ***Saving the project.*** Select **File > Save All** to save your changes.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

3.12 ***(Alarm Clock GUI)*** Create the GUI for the alarm clock in Fig. 3.34.

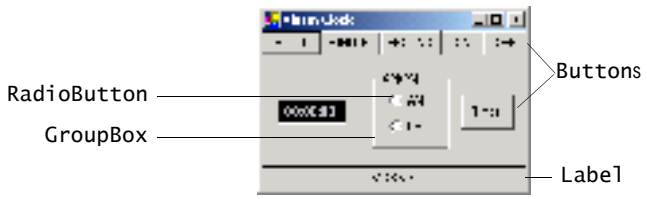
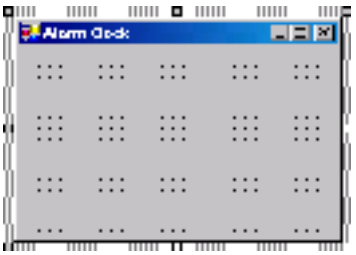


Figure 3.34 **Alarm Clock** GUI.

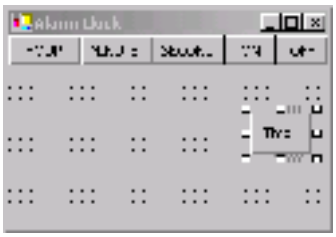
a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your `C:\SimplyCSP` directory named `AlarmClock`.

b) ***Renaming the Form file.*** Name the `Form` file `AlarmClock.cs`. Double click the file name to open the `Form` in design view.

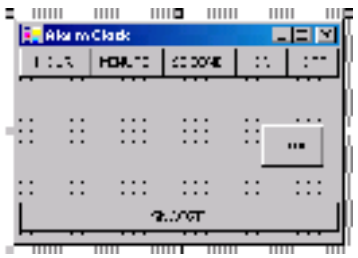
c) ***Manipulating the Form's properties.*** Change the `Size` property of the `Form` to `256, 176`. Change the `Text` property of the `Form` to `Alarm Clock`. Change the `Font` property to **Tahoma**.



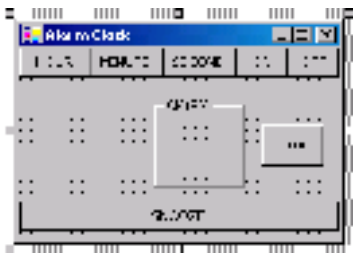
d) ***Adding Buttons to the Form.*** Add six `Button`s to the `Form`. Change the `Text` property of each `Button` to the appropriate text. Change the `Size` properties of the **Hour**, **Minute** and **Second** `Button`s to `56, 23`. The **ON** and **OFF** `Button`s get size `40, 23`. The **Timer** `Button` gets size `48, 32`. Align the `Button`s as shown in Fig. 3.34.



e) ***Adding a Label to the Form.*** Add a `Label` to the `Form`. Change the `Text` property to **Snooze**. Set its `Size` to `248, 23`. Set the `Label`'s `TextAlign` property to `MiddleCenter`. Finally, to draw a border around the edge of the **Snooze** `Label`, change the `BorderStyle` property of the **Snooze** `Label` to `FixedSingle`.

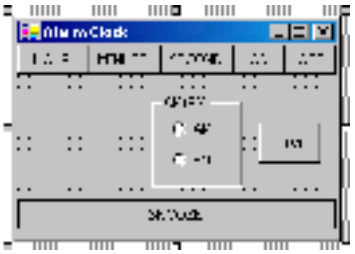


f) ***Adding a GroupBox to the Form.*** **GroupBox**es are like `Panel`s, except that `GroupBox`es can display a title. To add a `GroupBox` to the `Form`, double click the `GroupBox` control ( ) in the **Toolbox**. Change the `Text` property to **AM/PM**, and set the `Size` property to `72, 72`. To place the `GroupBox` in the correct location on the `Form`, set the `Location` property to `104, 38`.

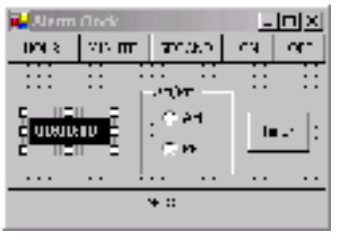


g) ***Adding AM/PM RadioButtons to the GroupBox.*** Add two `RadioButton`s to the `Form` by dragging the `RadioButton` control ( ) in the **Toolbox** and dropping it

onto the GroupBox twice. Change the Text property of one RadioButton to AM and the other to PM. Then place the RadioButtons as shown in Fig. 3.34 by setting the Location of the **AM** RadioButton to 16, 16 and that of the **PM** RadioButton to 16, 40. Set their Size properties to 48, 24.



h) ***Adding the time Label to the Form.*** Add a Label to the Form and change its Text property to 00:00:00. Change the BorderStyle property to Fixed3D and the BackColor to Black. Set the Size property to 64, 23. Use the Font property to make the time bold. Change the ForeColor to Silver (located in the **Web** tab) to make the time stand out against the black background. Set TextAlign to MiddleCenter to center the text in the Label. Position the Label as shown in Fig. 3.34.



i) ***Saving the project.*** Select **File > Save All** to save your changes.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

3.13 ***(Microwave Oven GUI)*** Create the GUI for the microwave oven shown in Fig. 3.35.

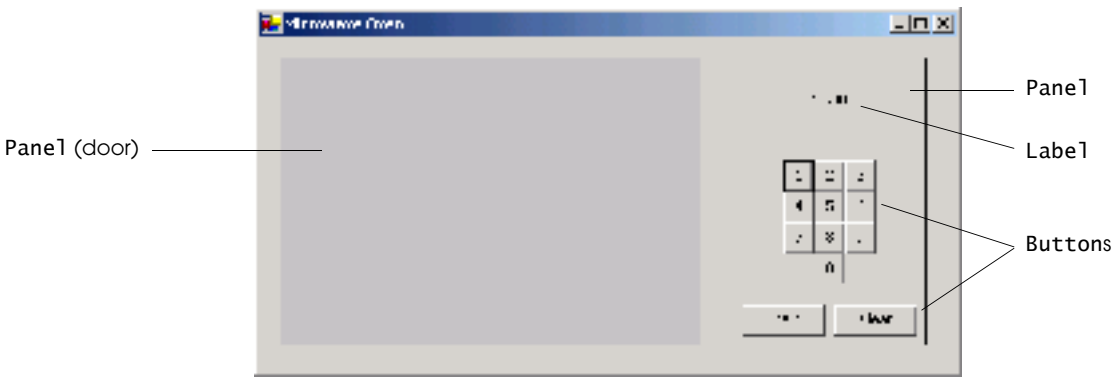
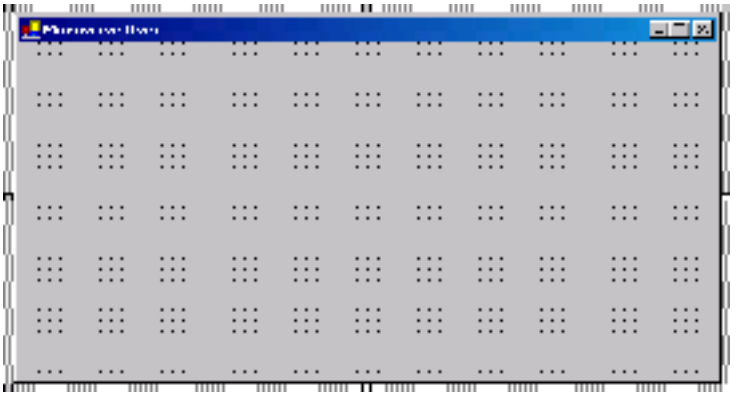


Figure 3.35 Microwave Oven GUI.

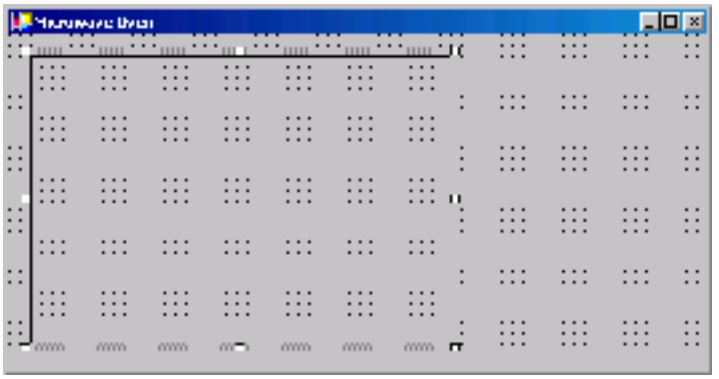
a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your C:\SimplyCSP directory named Microwave.

b) ***Renaming the Form file.*** Name the Form file Microwave.cs. Double click the file name to open the Form in design view.

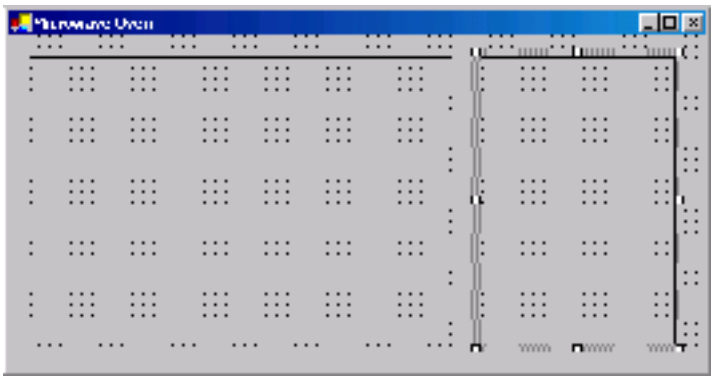
c) ***Manipulating the Form's properties.*** Change the Size property of the Form to 552, 288. Change the Text property of the Form to Microwave Oven. Change the Font property to **Tahoma**.



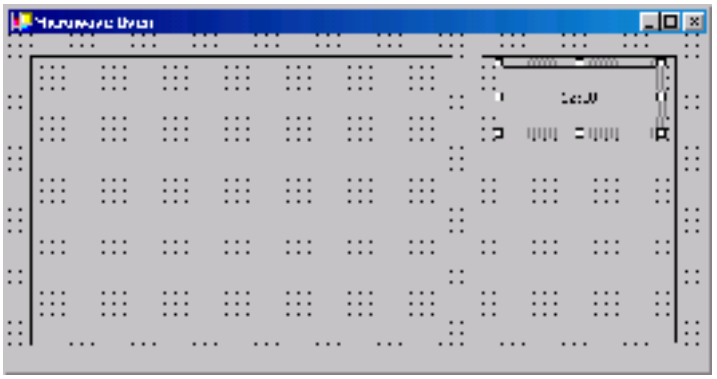
d) ***Adding the microwave oven door.*** Add a `Panel` to the `Form` by double clicking the `Panel` ( ) in the **Toolbox**. Select the `Panel` and change the `BackColor` property to `Silver` (located in the **Web** tab) in the **Properties** window. Then change the `Size` to `328, 224`. Next, change the `BorderStyle` property to `FixedSingle`.



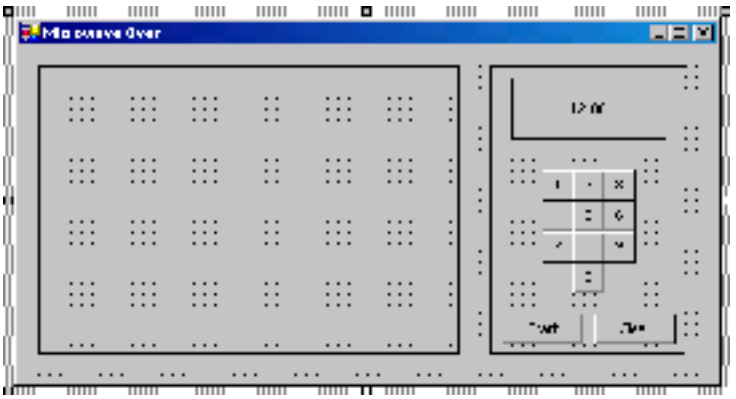
e) ***Adding another Panel.*** Add another `Panel` and change its `Size` to `152, 224` and its `BorderStyle` to `FixedSingle`. Place the `Panel` to the right of the door `Panel` as shown in Fig. 3.35.



f) ***Adding the microwave oven clock.*** Add a `Label` to the right `Panel` by clicking the `Label` in the **Toolbox** once, then clicking once inside the right `Panel`. Change the `Label`'s `Text` to `12:00`, `BorderStyle` to `FixedSingle` and `Size` to `120, 48`. Change `TextAlign` to `MiddleCenter`. Place the clock as shown in Fig. 3.35.



g) ***Adding a keypad to the microwave oven.*** Place a `Button` in the right `Panel` by clicking the `Button` control in the `Toolbox` once, then clicking inside the `Panel`. Change the `Text` to `1` and the `Size` to `24, 24`. Repeat this process for nine more `Button`s, changing the `Text` property in each to the next number in the keypad. Then add the **Start** and **Clear** `Button`s, each of `Size` `64, 24`. Do not forget to set the `Text` properties for each of these `Button`s. Finally, arrange the `Button`s as shown in Fig. 3.35. The **1** `Button` is located at `40, 80` and the **Start** `Button` is located at `8, 192`.



h) ***Saving the project.*** Select **File > Save All** to save your changes.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

3.14 ***(Cell Phone GUI)*** Create the GUI for the cell phone shown in Fig. 3.36.

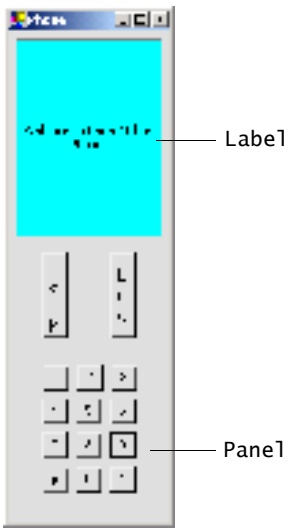
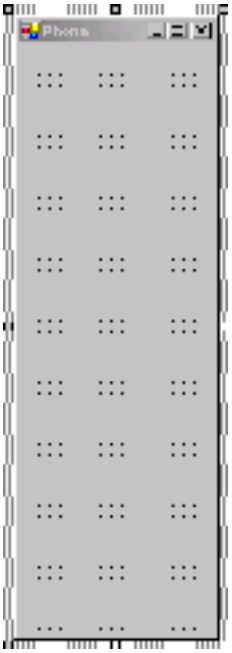


Figure 3.36 **Cell Phone** GUI.

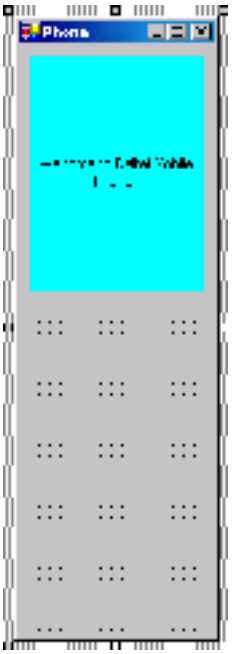
a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your `C:\SimplyCSP` directory named `Phone`.

b) ***Renaming the Form file.*** Name the `Form` file `Phone.cs`. Double click the file name to open the `Form` in design view.

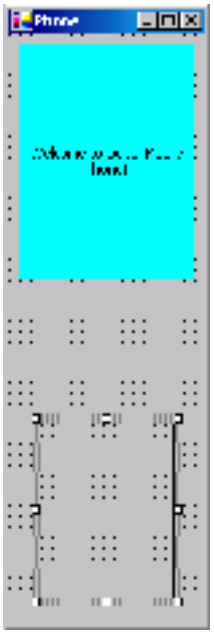
c) ***Manipulating the Form's properties.*** Change the `Form`'s `Text` property to `Phone` and the `Size` to `160, 488`. Change the `Font` property to **Tahoma**.



d) ***Adding the display Label.*** Add a `Label` to the `Form`. Change its `BackColor` to `Aqua` (in the **Web** tab palette), the `Text` to `Welcome to Deitel Mobile Phone!` and the `Size` to `136, 184`. Change the `TextAlign` property to `MiddleCenter`. Then place the `Label` as shown in Fig. 3.36.

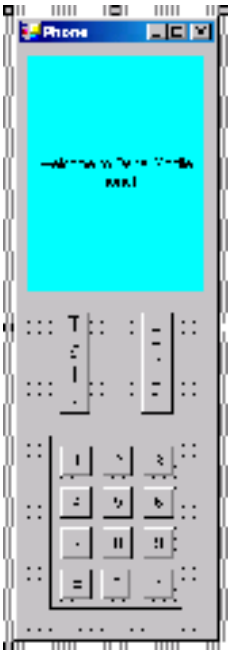


e) ***Adding the keypad Panel.*** Add a `Panel` to the `Form`. Change its `BorderStyle` property to `FixedSingle` and its `Size` to `104, 136`.



f) ***Adding the keypad Buttons.*** Add the keypad `Button`s to the `Form` (12 `Button`s in all). Each `Button` on the number pad should be of `Size` `24`, `24` and should be placed in the `Panel`. Change the `Text` property of each `Button` such that numbers `0`–`9`, the pound (`#`) and the star (`*`) keys are represented. Then add the final two `Button`s such that the `Text` property for one is `Talk` and the other is `End`. Change the `Size` of each `Button` to `24`, `80`, and notice how the small `Size` causes the `Text` to align vertically. Change the `Font` size of these two `Button`s to `12`.

g) ***Placing the controls.*** Arrange all the controls so that your GUI looks like Fig. 3.36.



h) ***Saving the project.*** Select **File > Save All** to save your changes.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

3.15 ***(Vending Machine GUI)*** Create the GUI for the vending machine in Fig. 3.37.

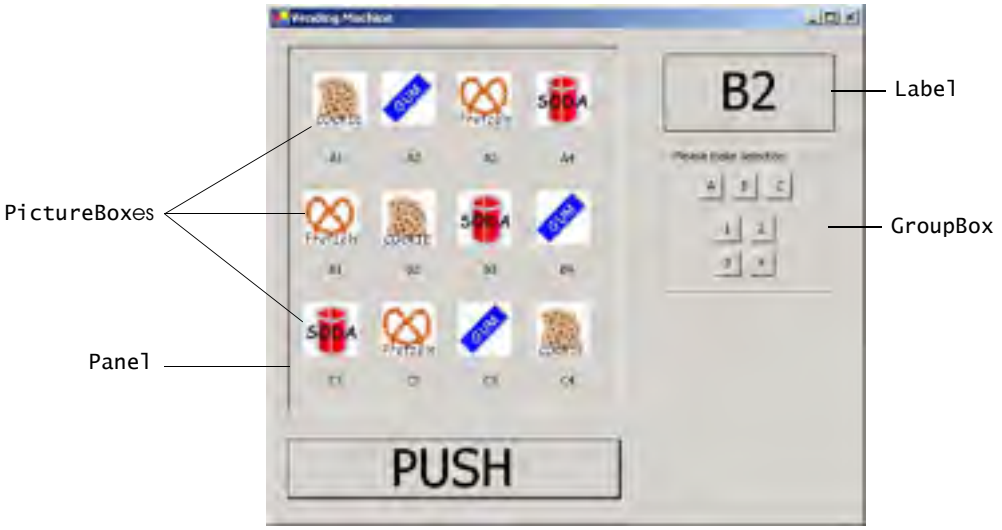
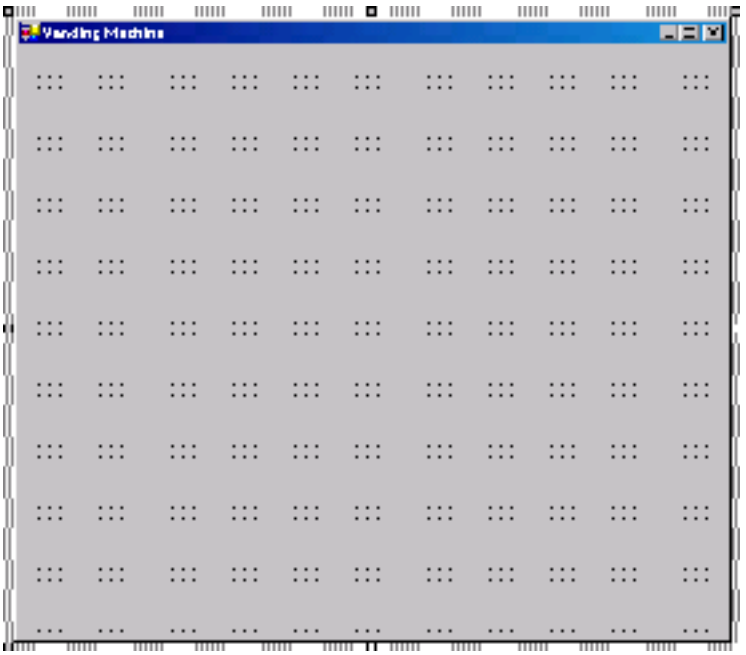


Figure 3.37 Vending Machine GUI.

a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your `C:\SimplyCSP` directory named `VendingMachine`.

b) ***Renaming the Form file.*** Name the `Form` file `VendingMachine.cs`. Double click the file name to open the `Form` in design view.

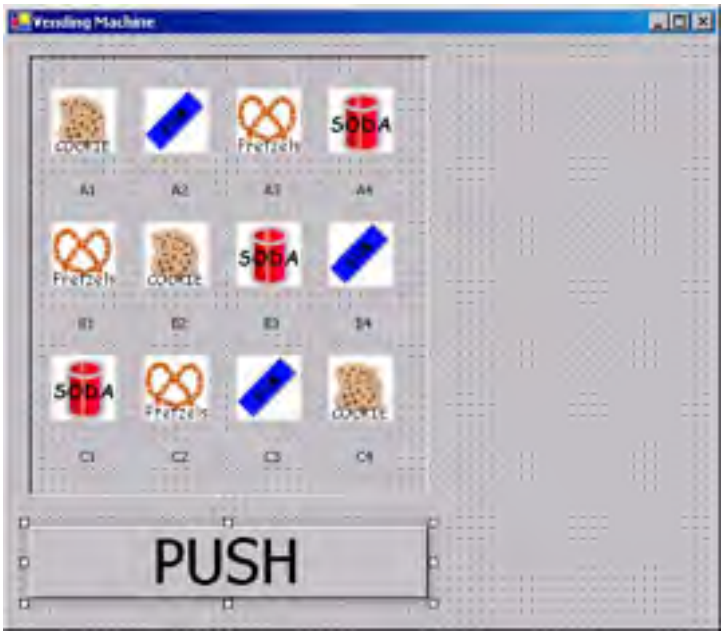
c) ***Manipulating the Form's properties.*** Set the `Text` property of the `Form` to `Vending Machine` and the `Size` to `560, 488`. Change the `Font` property to Tahoma.



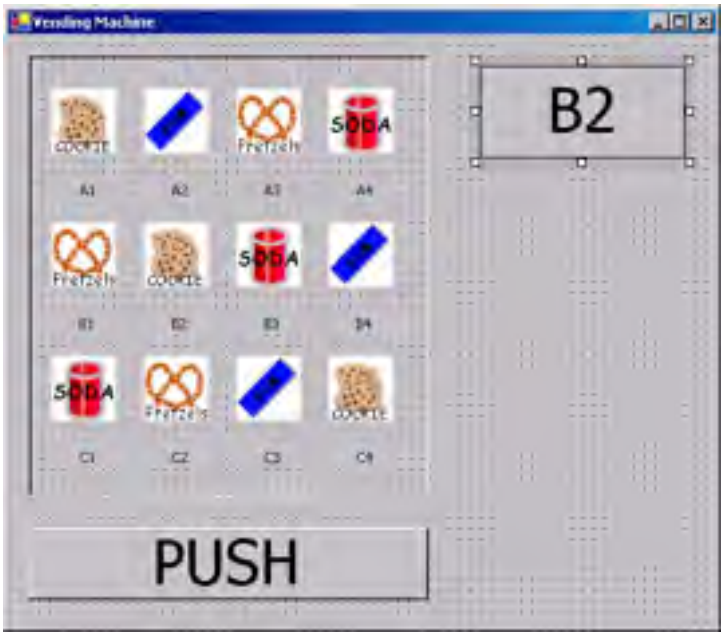
d) ***Adding the food selection Panel.*** Add a `Panel` to the `Form`, and change its `Size` to `312, 344` and `BorderStyle` to `Fixed3D`. Add a `PictureBox` to the `Panel`, and change its `Size` to `50, 50`. Then set the `Image` property by clicking the ellipsis `Button` and choosing a file from the `C:\Examples\Tutorial03\ExerciseImages\VendingMachine` directory. Repeat this process for 11 more `PictureBoxes`.

e) ***Adding Labels for each vending item.*** Add a `Label` under each `PictureBox`. Change the `Text` property of the `Label` to `A1`, the `TextAlign` property to `TopCenter` and the `Size` to `56, 16`. Place the `Label` so that it is located as in Fig. 3.37. Repeat this process for `A2` through `C4` (11 `Label`s).

f) ***Creating the vending machine door (as a Button).*** Add a `Button` to the `Form` by dragging the `Button` control in the **Toolbox** and dropping it below the `Panel`. Change the `Button`'s `Text` property to `PUSH`, its `Font Size` to `36` and its `Size` to `312, 56`. Then place the `Button` on the `Form` as shown in Fig. 3.37.

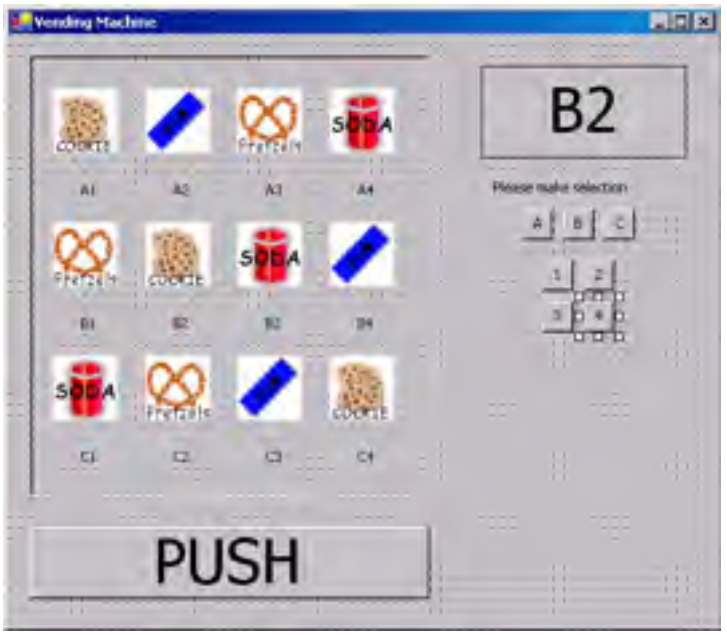


g) ***Adding the selection display Label.*** Add a `Label` to the `Form`, and change the `Text` property to `B2`, `BorderStyle` to `FixedSingle`, `Font Size` to `36`, `TextAlign` to `MiddleCenter` and `Size` to `160, 72`.



h) ***Grouping the input Buttons.*** Add a `GroupBox` below the `Label`, and change the `Text` property to `Please make selection` and the `Size` to `160, 136`.

i) ***Adding the input `Button`s.*** Finally, add `Button`s to the `GroupBox`. For the seven `Button`s, change the `Size` property to `24, 24`. Then change the `Text` property of the `Button`s such that each `Button` has one of the values `A`, `B`, `C`, `1`, `2`, `3` or `4`, as shown in Fig. 3.37. When you are done, move the controls on the `Form` so that they are aligned as shown in Fig. 3.37.



j) ***Saving the project.*** Select **File > Save All** to save your changes.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Programming Challenge ▶

3.16 ***(Radio GUI)*** Create the GUI for the radio in Fig. 3.38. [*Note:* All colors used in this exercises are from the **Web** palette.]

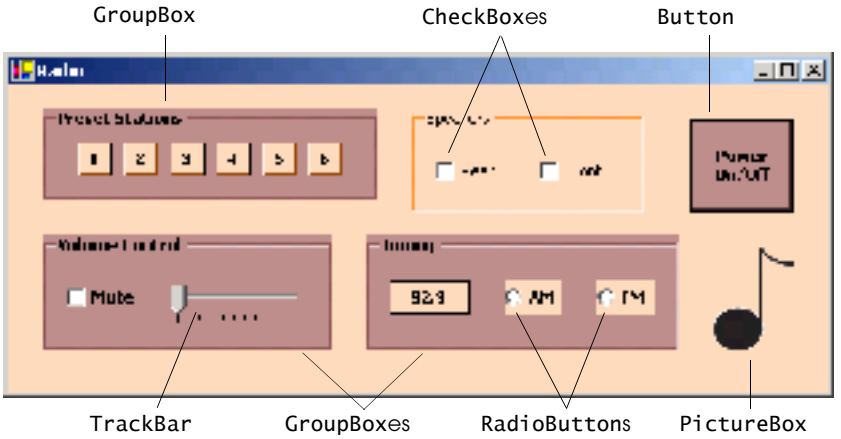


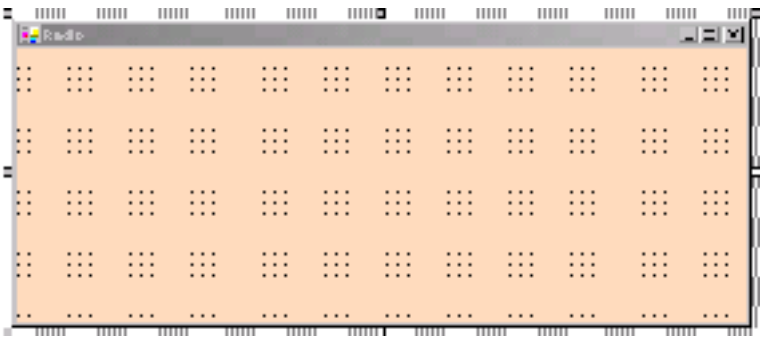
Figure 3.38 Radio GUI.

In this exercise, you will create this GUI on your own. Feel free to experiment with different control properties. For the image in the `PictureBox`, use the file (`MusicNote.gif`) found in the `C:\Examples\Tutorial03\ExerciseImages\Radio` directory.

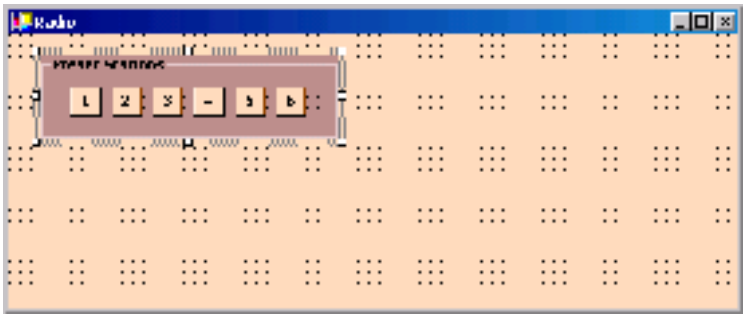
a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your `C:\SimplyCSP` directory named `Radio`.

b) ***Renaming the Form file.*** Name the `Form` file `Radio.cs`. Double click the file name to open the `Form` in design view.

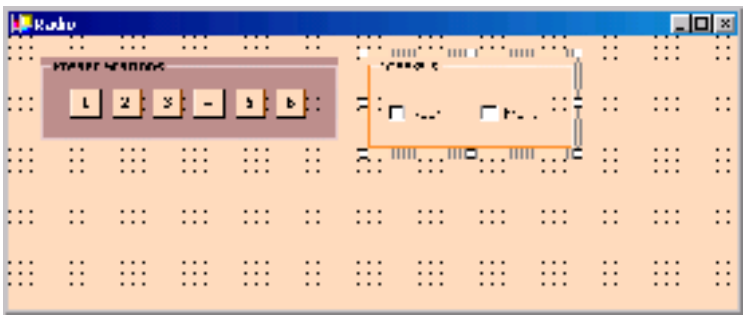
c) ***Manipulating the Form's properties.*** Change the `Form`'s `Text` property to `Radio` and the `Size` to `576, 240`. Change the `Font` property to **Tahoma**. Set `BackColor` to `PeachPuff`.



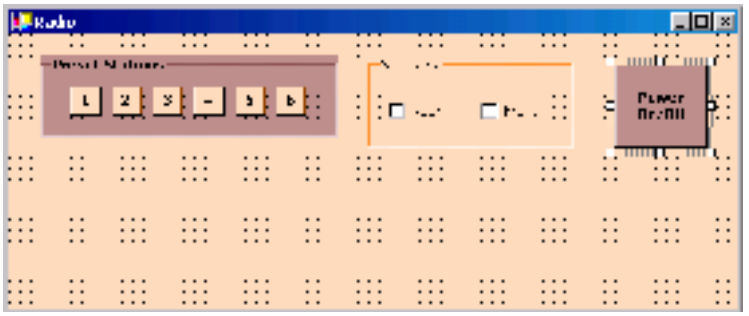
d) ***Adding the Preset Stations GroupBox and Buttons.*** Add a `GroupBox` to the `Form`. Set its `Size` to `232, 64`, its `Text` to `Preset Stations`, its `ForeColor` to `Black` and its `BackColor` to `RosyBrown`. Change its `Font` to `bold`. Finally, set its `Location` to `24, 16`. Add six `Buttons` to the `GroupBox`. Set each `BackColor` to `PeachPuff` and each `Size` to 24, 23. Change the `Buttons`' `Text` properties to `1`, `2`, `3`, `4`, `5`, and `6`, respectively.



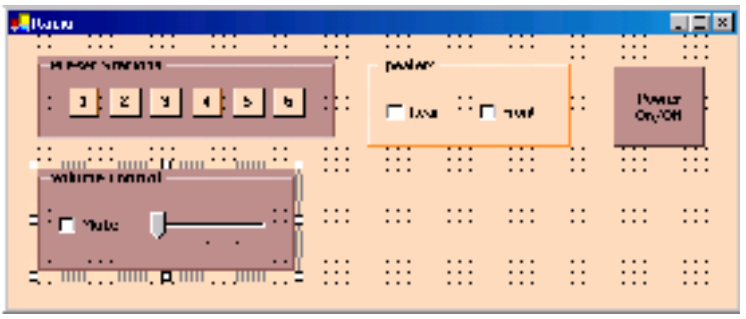
e) ***Adding the Speakers GroupBox and CheckBoxes.*** Add a `GroupBox` to the `Form`. Set its `Size` to `160, 72`, its `Text` to `Speakers` and its `ForeColor` to `Black`. Set its `Location` to `280, 16`. Add two `CheckBox`es to the `Form`. Set each `CheckBox`'s `Size` to `56, 24`. Set the `Text` properties for the `CheckBox`es to `Rear` and `Front`.



f) ***Adding the Power On/Off Button.*** Add a `Button` to the `Form`. Set its `Text` to `Power On/Off`, its `BackColor` to `RosyBrown`, its `ForeColor` to `Black` and its `Size` to `72, 64`. Change its `Font` style to `Bold`.

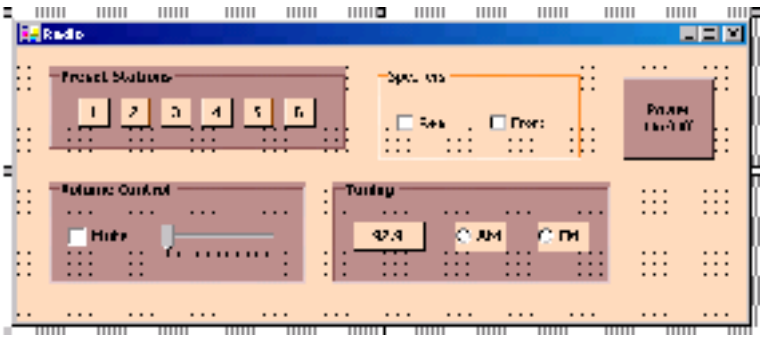


g) ***Adding the Volume Control GroupBox, the Mute CheckBox and the Volume TrackBar.*** Add a `GroupBox` to the `Form`. Set its `Text` to `Volume Control`, its `BackColor` to `RosyBrown`, its `ForeColor` to `Black` and its `Size` to `200, 80`. Set its `Font` style to `Bold`. Add a `CheckBox` to the `GroupBox`. Set its `Text` to `Mute` and its `Size` to `56, 24`. Add a `TrackBar` to the `GroupBox`.

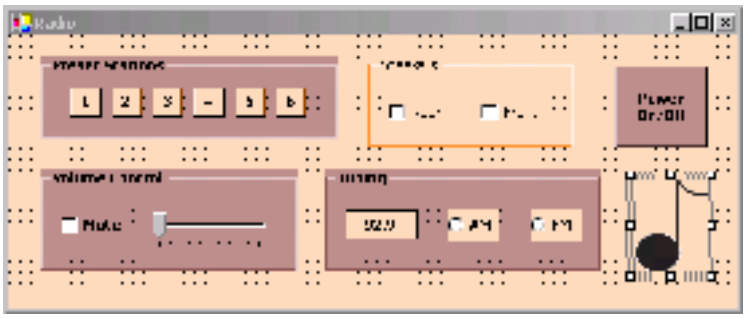


h) ***Adding the Tuning GroupBox, the radio station Label and the AM/FM RadioButtons.*** Add a `GroupBox` to the `Form`. Set its `Text` to `Tuning`, its `ForeColor` to `Black`

and its BackColor to RosyBrown. Set its Font style to Bold and its Size to 216, 80. Add a Label to the Form. Set its BackColor to PeachPuff, its ForeColor to Black, its BorderStyle to FixedSingle, its Font style to Bold, its TextAlign to MiddleCenter and its Size to 56, 23. Set its Text to 92.9. Place the Label as shown in the figure. Add two RadioButtons to the GroupBox. Change the BackColor to PeachPuff and change the Size to 40,24. Set one's Text to AM and the other's Text to FM.



i) ***Adding the image.*** Add a PictureBox to the Form. Set its BackColor to Transparent, its SizeMode to StretchImage and its Size to 56, 72. Set its Image property to C:\Examples\Tutorial03\ExerciseImages\Radio\MusicNote.gif.



j) ***Saving the project.*** Select **File > Save All** to save your changes.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

TUTORIAL 4

Designing the Inventory Application

Introducing `TextBoxes` and `Buttons`

Solutions

Instructor's Manual Exercise Solutions Tutorial 4

MULTIPLE-CHOICE QUESTIONS

4.1 A new Windows application is created by selecting _____ from the **File** menu.

a) **New > Program** b) **New > File...**
c) **New > Project...** d) **New > Application**

4.2 A `Label`'s `BorderStyle` property can be set to _____.

a) `Fixed3D` b) `Single`
c) `3D` d) All of the above.

4.3 When creating a `Label`, you can specify the _____ of that `Label`.

a) alignment of the text b) border style
c) size d) All of the above.

4.4 Changing the value stored in the _____ property will change the name of the `Form` file.

a) `Name` b) `File`
c) `File Name` d) `Full Path`

4.5 _____ should be used to prefix all `TextBox` names.

a) `txt` b) `tbx`
c) `Frm` d) `tbn`

4.6 A(n) _____ helps the user understand a control's purpose.

a) `Button` b) descriptive `Label`
c) output `Label` d) title bar

4.7 A _____ is a control in which the user can enter data from a keyboard.

a) `Button` b) `TextBox`
c) `Label` d) `PictureBox`

4.8 A descriptive `Label` uses _____.

a) sentence-style capitalization b) book-title capitalization
c) a colon at the end of its text d) Both a and c.

4.9 You should use the _____ font in your Windows applications.

a) Tahoma b) MS Sans Serif
c) Times d) Palatino

4.10 _____ should be used to prefix all `Button` names.

a) `but` b) `lbl`
c) `Frm` d) `btn`

Answers: 4.1) c. 4.2) a. 4.3) d. 4.4) c. 4.5) a. 4.6) b. 4.7) b. 4.8) d. 4.9) a. 4.10) d.

EXERCISES

At the end of each tutorial, you will find a summary of new GUI design tips listed in the GUI Design Guidelines section. A cumulative list of GUI design guidelines, organized by control appears in Appendix C. In these exercises, you will find C# `Forms` that do not follow the GUI design guidelines presented in this tutorial. For each exercise, you must modify control properties so that your end result is consistent with the guidelines presented in the tutorial. Note that these applications do not provide any functionality.

4.11 ***(Address Book GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for an address book (Fig. 4.22).

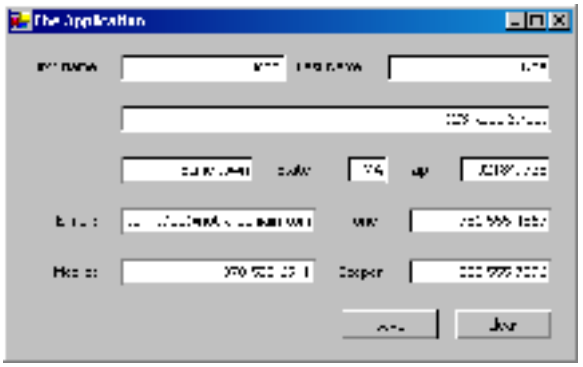


Figure 4.22 **Address Book** application without GUI design guidelines applied.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial04\Exercises\AddressBook` to your `C:\SimplyCSP` directory.

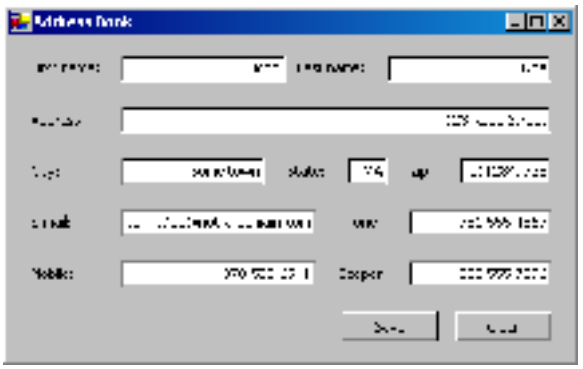
b) ***Opening the application's template file.*** Double click `AddressBook.sln` in the `AddressBook` directory to open the application. Double click `AddressBook.cs` in the **Solution Explorer** window to open the `Form` in design view.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify their properties so that the GUI conforms to the design guidelines you have learned. Add new controls as necessary.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

e) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:



1. Change the `Form`'s title (`Text` property).
2. All `TextBoxes` should have corresponding `Labels`.
3. `Labels` indicating control usage should use sentence-style capitalization.
4. `Buttons` should use book-title capitalization.
5. Each descriptive `Label` text should end with a colon.

4.12 ***(Mortgage Calculator GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for a mortgage calculator (Fig. 4.23).

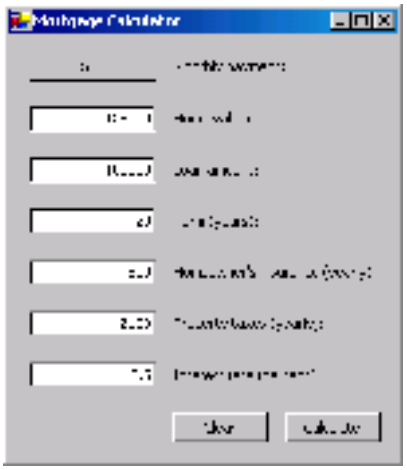


Figure 4.23 **Mortgage Calculator** application without GUI design guidelines applied.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial04\Exercises\MortgageCalculator` to your `C:\SimplyCSP` directory.

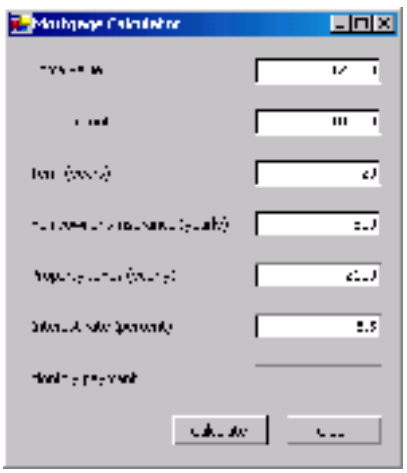
b) ***Opening the application's template file.*** Double click `MortgageCalculator.sln` in the `MortgageCalculator` directory to open the application. Double click `MortgageCalculator.cs` in the **Solution Explorer** window to open the `Form` in design view.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify their properties so that the GUI conforms to the design guidelines you have learned.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

e) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:



1. `Label` should be placed above or to the left of the control it is describing.
2. Output `Label`'s setting should be `BorderStyle` property `Fixed3D`.
3. Output `Label` initially should be blank.
4. Place an application's output below or to the right of the `Form`'s input control.

4.13 ***(Password GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for a password-protected message application (Fig. 4.24).

Figure 4.24 **Password** application without GUI design guidelines applied.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial04\Exercises\Password` to your `C:\SimplyCSP` directory.

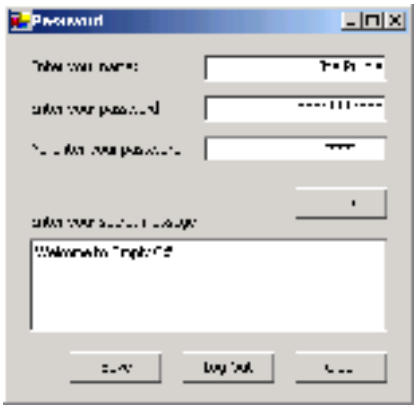
b) ***Opening the application's template file.*** Double click `Password.sln` in the `Password` directory to open the application. Double click `Password.cs` in the **Solution Explorer** window to open the `Form` in design view.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify their properties so that the GUI conforms to the design guidelines you have learned.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

e) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:



1. Keep the `Label` on the `Button`s as short and descriptive as possible.
2. Arrange groups of controls approximately 2 grid units apart on a `Form`.
3. Leave approximately 2 grid units of space between the edges of the `Form` and controls nearest the edge. Increase the `Form`'s width.
4. `Button`s use book-title capitalization.

Programming Challenge ▶

4.14 ***(Monitor Invoice GUI)*** In this exercise, you apply the GUI design guidelines you have learned to a graphical user interface for an invoice application (Fig. 4.25).

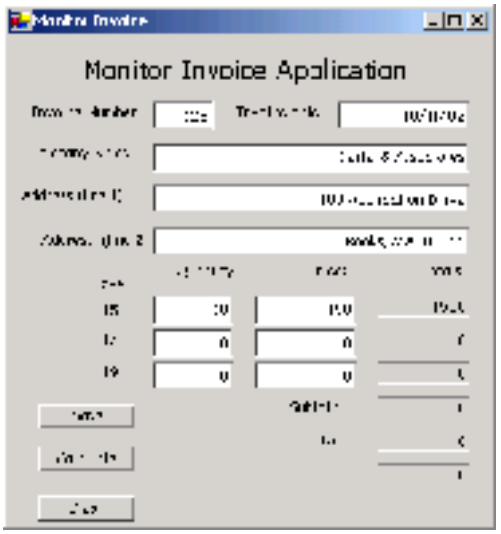


Figure 4.25 **Monitor Invoice** application without GUI design guidelines applied.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial04\Exercises\MonitorInvoice` to your `C:\SimplyCSP` directory.

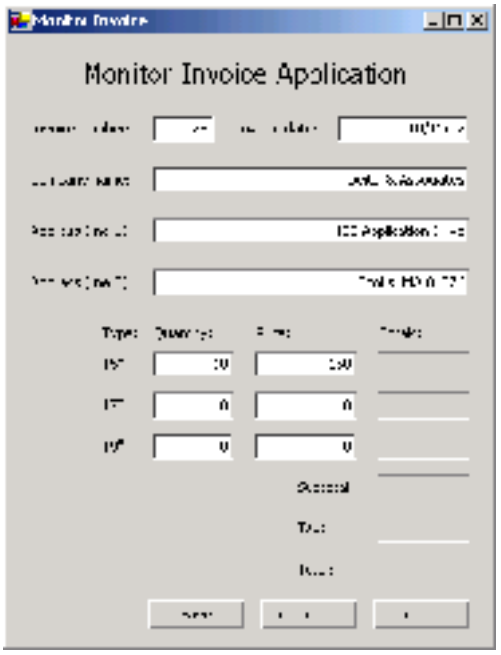
b) ***Opening the application's template file.*** Double click `MonitorInvoice.sln` in the `MonitorInvoice` directory to open the application. Double click `MonitorInvoice.cs` in the **Solution Explorer** window to open the `Form` in design view.

c) ***Applying GUI design guidelines.*** Rearrange the controls and modify their properties so that the GUI conforms to the design guidelines you have learned. Add new controls as necessary.

d) ***Saving the project.*** Select **File > Save All** to save your changes.

e) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:



1. Use **Tahoma** font.
2. `Labels` indicating control usage should end with colon.

3. The `Label` and the control it describes should be aligned on the left if arranged vertically.
4. `Label` should use sentence-style capitalization.
5. `Buttons` should be placed in the top right or bottom right of a `Form`.
6. Each output `Label` must have a label that describes it.
7. Output `Label`s arranged vertically and used to display numbers in a calculation should have the `TextAlign` property set to `MiddleRight`.
8. Descriptive `Label`s that are in the same column vertically should be left aligned.

TUTORIAL 5

Completing the Inventory Application

Introducing Programming

Solutions

Instructor's Manual Exercise Solutions Tutorial 5

MULTIPLE-CHOICE QUESTIONS

5.1 A(n) _____ represents a user action, such as clicking a `Button`.

a) statement
b) event
c) application
d) method

5.2 To switch to code view, select _____.

a) **Code > View**
b) **Design > Code**
c) **View > Code**
d) **View > File Code**

5.3 Code that performs the functionality of an application _____.

a) normally is provided by the programmer
b) can never be in the form of an event handler
c) always creates a graphical user interface
d) is always generated by the IDE

5.4 Comments _____.

a) help improve program readability
b) can be placed at the end of a line of code
c) are ignored by the compiler
d) All of the above.

5.5 A _____ typically ends a C# statement.

a) period
b) colon
c) semicolon
d) comma

5.6 A(n) _____ causes an application to produce erroneous results.

a) logic error
b) event
c) assignment statement
d) syntax error

5.7 A portion of code that performs a specific task and returns a value is known as a(n) _____.

a) variable
b) method
c) operand
d) identifier

5.8 C# keywords are _____.

a) identifiers
b) reserved words
c) case insensitive
d) properties

5.9 Visual Studio .NET allows you to organize code into _____, which you can expand or collapse to facilitate code editing.

a) statements
b) operators
c) regions
d) keywords

5.10 An example of a white-space character is a _____ character.

a) space
b) tab
c) newline
d) All of the above.

Answers: 5.1) b. 5.2) c. 5.3) a. 5.4) d. 5.5) c. 5.6) a. 5.7) b. 5.8) b. 5.9) c. 5.10) d.

EXERCISES

5.11 ***(Inventory Enhancement)*** Extend the **Inventory** application to include a `TextBox` in which the user can enter the number of shipments received in a week (Fig. 5.27). Assume

every shipment has the same number of cartons (each of which has the same number of items). Then, modify the code so that the **Inventory** application uses that value in its calculation.

Figure 5.27 Enhanced **Inventory** application GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial05\Exercises\InventoryEnhancement` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Inventory.sln` in the `InventoryEnhancement` directory to open the application.

c) ***Resizing the Form.*** Resize the `Form` you used in this tutorial by setting the `Size` property to `296, 144`. Move the `Button` toward the bottom of the `Form`, as shown in Fig. 5.27. Its new location should be `184, 78`.

d) ***Adding a `Label`.*** Add a `Label` to the `Form` and change the `Text` property to `Shipments this week:`. Set the `Location` property to `16, 80`. Resize the `Label` so that the entire text displays. Set the `Label`'s `Name` property to `lblShipments`.

e) ***Adding a `TextBox`.*** Add a `TextBox` to the right of the `Label`. Set the `Text` property to `0` and the `Location` property to `128, 80`. Set the `TextAlign` and `Size` properties to the same values as for the other `TextBoxes` in this tutorial's example. Set the `TextBox`'s `Name` property to `txtShipments`.

f) ***Modifying the code.*** Modify the **Calculate Total** `Click` event handler so that it multiplies the number of shipments per week with the product of the number of cartons in a shipment and the number of items in a carton.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter `2` in the **Cartons per shipment:** `TextBox`. Enter `3` in the **Items per carton:** `TextBox`. Enter `4` in the **Shipments this week:** `TextBox`. Click the **Calculate** `Button`. The **Inventory** `Form` in Fig. 5.27 shows the correct result after these values have been entered.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 5.11 Solution
2  // Inventory.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Inventory
12 {
13    /// <summary>
14    /// Summary description for FrmInventory.
15    /// </summary>
16    public class FrmInventory : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblCartons;
19       private System.Windows.Forms.Label lblItems;
20       private System.Windows.Forms.Label lblTotal;
21       private System.Windows.Forms.Label lblTotalResult;
```

```
22          private System.Windows.Forms.TextBox txtCartons;
23          private System.Windows.Forms.TextBox txtItems;
24          private System.Windows.Forms.Button btnCalculate;
25          private System.Windows.Forms.Label lblShipments;
26          private System.Windows.Forms.TextBox txtShipments;
27          /// <summary>
28          /// Required designer variable.
29          /// </summary>
30          private System.ComponentModel.Container components = null;
31
32          public FrmInventory()
33          {
34             //
35             // Required for Windows Form Designer support
36             //
37             InitializeComponent();
38
39             //
40             // TODO: Add any constructor code after InitializeComponent
41             // call
42             //
43          }
44
45          /// <summary>
46          /// Clean up any resources being used.
47          /// </summary>
48          protected override void Dispose( bool disposing )
49          {
50             if( disposing )
51             {
52                if (components != null)
53                {
54                   components.Dispose();
55                }
56             }
57             base.Dispose( disposing );
58          }
59
60          // Windows Form Designer generated code
61
62          /// <summary>
63          /// The main entry point for the application.
64          /// </summary>
65          [STAThread]
66          static void Main()
67          {
68             Application.Run( new FrmInventory() );
69          }
70
71          // handles Click event
72          private void btnCalculate_Click(
73             object sender, System.EventArgs e )
74          {
75             // multiply values input and display result in Label
76             lblTotalResult.Text = Convert.ToString(
77                Int32.Parse( txtCartons.Text ) *
78                Int32.Parse( txtItems.Text ) *
79                Int32.Parse( txtShipments.Text ) );
80
81          } // end method btnCalculate_Click
```

```
82
83     } // end class FrmInventory
84  }
```

5.12 ***(Counter Application)*** Create a counter application (Fig. 5.28). Your counter application will consist of a `Label` and `Button` on the `Form`. The `Label` initially displays `0`, but, each time a user clicks the `Button`, the value in the `Label` is increased by `1`. When incrementing the `Label`, you will need to write a statement such as

```
lblTotal.Text = Convert.ToString( Int32.Parse( lblTotal.Text ) + 1 );
```

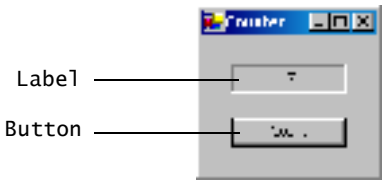


Figure 5.28 Counter GUI.

a) ***Creating a new project.*** Open Visual Studio .NET. Create a new project in your `C:\SimplyCSP` directory named `Counter`.

b) ***Modifying a new Form.*** Change the `Form`'s `Size` property to `168, 144`. Modify the `Form` so that the title reads **Counter**. Change the `Name` property to `FrmCounter`. Change `Form1` to `FrmCounter` in method `Main` and in the comments above the class declaration.

c) ***Adding a `Label`.*** Add a `Label` to the `Form`, and place it as shown in Fig. 5.28. Make sure that the `Label`'s `Text` property is set to `0` and that `TextAlign` property is set so that any text will appear in the middle (both horizontally and vertically) of the `Label`. This can be done by using the `MiddleCenter` `TextAlign` property. Also set the `BorderStyle` property to `Fixed3D`. Set the `Label`'s `Name` property to `lblCountTotal`.

d) ***Adding a `Button`.*** Add a `Button` to the `Form`. Set the `Button`'s `Text` property to contain the text **Count**. Set the `Button`'s `Name` property to `btnCount`.

e) ***Creating an event handler.*** Add an event handler to the **Count** `Button` such that the value in the `Label` increases by `1` each time the user clicks the **Count** `Button`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Count** `Button` repeatedly and watch the result.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 5.12 Solution
2  // Counter.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Counter
12 {
13    /// <summary>
14    /// Summary description for FrmCounter.
15    /// </summary>
16    public class FrmCounter : System.Windows.Forms.Form
17    {
```

```
18          private System.Windows.Forms.Label lblCountTotal;
19          private System.Windows.Forms.Button btnCalculate;
20          /// <summary>
21          /// Required designer variable.
22          /// </summary>
23          private System.ComponentModel.Container components = null;
24
25          public FrmCounter()
26          {
27             //
28             // Required for Windows Form Designer support
29             //
30             InitializeComponent();
31
32             //
33             // TODO: Add any constructor code after InitializeComponent
34             // call
35             //
36          }
37
38          /// <summary>
39          /// Clean up any resources being used.
40          /// </summary>
41          protected override void Dispose( bool disposing )
42          {
43             if( disposing )
44             {
45                if (components != null)
46                {
47                   components.Dispose();
48                }
49             }
50             base.Dispose( disposing );
51          }
52
53          // Windows Form Designer generated code
54
55          /// <summary>
56          /// The main entry point for the application.
57          /// </summary>
58          [STAThread]
59          static void Main()
60          {
61             Application.Run( new FrmCounter() );
62          }
63
64          // handles Click event
65          private void btnCount_Click(
66             object sender, System.EventArgs e )
67          {
68             // when Button is clicked add one to lblCountTotal
69             lblCountTotal.Text = Convert.ToString(
70                Int32.Parse( lblCountTotal.Text ) + 1 );
71
72          } // end method btnCount_Click
73
74       } // end class FrmCounter
75    }
```

5.13 ***(Account Information Application)*** Create an application that allows a user to input a name, account number and deposit amount (Fig. 5.29). The user then clicks the **Enter** `Button`, which causes the name and account number to be copied and displayed in two output `Label`s. The deposit amount entered will be added to the deposit amount displayed in another output `Label`. The result is displayed in the same output `Label`. Every time the **Enter** `Button` is clicked, the deposit amount entered is added to the deposit amount displayed in the output `Label`, keeping a cumulative total. When updating the `Label`, you will need to write a statement such as

```
lblBalance.Text = Convert.ToString(
   Int32.Parse( lblDeposits.Text ) + Int32.Parse( txtDepositAmount )
);
```

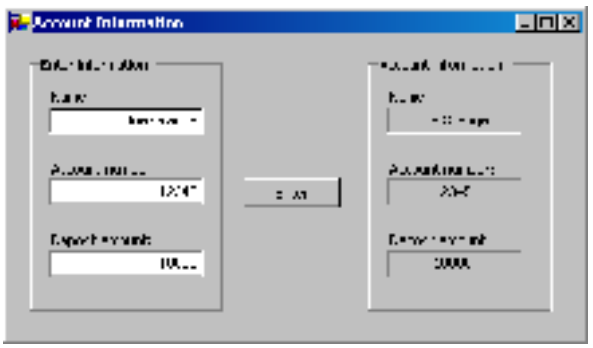


Figure 5.29 Account Information GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial05\Exercises\AccountInformation` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AccountInformation.sln` in the `AccountInformation` directory to open the application.

c) ***Creating an event handler.*** Add an event handler for the **Enter** `Button`'s `Click` event.

d) ***Coding the event handler.*** Code the event handler to copy information from the **Name:** and **Account number:** `TextBox`es to their corresponding output `Label`s. Then add the value in the **Deposit amount:** `TextBox` to the value in **Deposit amount:** output `Label`, and display the result in the **Deposit amount:** output `Label`.

e) ***Running the application.*** Select **Debug > Start** to run your application. Begin with the values in Fig. 5.29 when you test your application.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 5.13 Solution
2  // AccountInformation.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace AccountInformation
12 {
13    /// <summary>
14    /// Summary description for FrmAccountInformation.
15    /// </summary>
16    public class FrmAccountInformation : System.Windows.Forms.Form
```

```
17    {
18       private System.Windows.Forms.GroupBox fraInput;
19       private System.Windows.Forms.Label lblNameLabel1;
20       private System.Windows.Forms.TextBox txtName;
21       private System.Windows.Forms.Label lblAccountNumberLabel1;
22       private System.Windows.Forms.TextBox txtAccountNumber;
23       private System.Windows.Forms.Label lblDepositAmountLabel1;
24       private System.Windows.Forms.TextBox txtDepositAmount;
25       private System.Windows.Forms.Button btnEnter;
26       private System.Windows.Forms.GroupBox fraOutput;
27       private System.Windows.Forms.Label lblNameLabel2;
28       private System.Windows.Forms.Label lblCopiedName;
29       private System.Windows.Forms.Label lblAccountNumberLabel2;
30       private System.Windows.Forms.Label lblCopiedAccountNumber;
31       private System.Windows.Forms.Label lblDepositAmountLabel2;
32       private System.Windows.Forms.Label lblBalance;
33       /// <summary>
34       /// Required designer variable.
35       /// </summary>
36       private System.ComponentModel.Container components = null;
37
38       public FrmAccountInformation()
39       {
40          //
41          // Required for Windows Form Designer support
42          //
43          InitializeComponent();
44
45          //
46          // TODO: Add any constructor code after InitializeComponent
47          // call
48          //
49       }
50
51       /// <summary>
52       /// Clean up any resources being used.
53       /// </summary>
54       protected override void Dispose( bool disposing )
55       {
56          if( disposing )
57          {
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmAccountInformation() );
75       }
76
```

```
77          // handles Click event
78          private void btnEnter_Click(
79             object sender, System.EventArgs e )
80          {
81             // copy user input
82             lblCopiedName.Text = txtName.Text;
83             lblCopiedAccountNumber.Text = Convert.ToString(
84                Int32.Parse( txtAccountNumber.Text ) );
85             lblBalance.Text = Convert.ToString(
86                Int32.Parse( lblBalance.Text ) +
87                Int32.Parse( txtDepositAmount.Text ) );
88
89          } // end method btnEnter_Click
90
91       } // end class FrmAccountInformation
92    }
```

What does this code do? ▶

5.14 After entering 10 in the `txtPrice TextBox` and `1.05` in the `txtTax TextBox`, a user clicks the `Button` named `btnEnter`. What is the result of the click, given the following code? Assume that this application has an output `Label`, `lblOutput`. [*Note*: The `Double.Parse` method is similar to the `Int32.Parse` method, but can convert a string of characters into a number with a decimal point. You will learn more about the `Double.Parse` method in later tutorials.]

```
1  private void btnEnter_Click( object sender, System.EventArgs e )
2  {
3     lblOutput.Text = Convert.ToString(
4        Double.Parse( txtPrice.Text ) *
5        Double.Parse( txtTax.Text ) );
6
7  } // end method btnEnter_Click
```

Answer: This displays the number `10.5` in a `Label`. (This is the amount of the sale including the tax.) The complete code reads:

```
1  // Exercise 5.14 Solution
2  // TotalCost.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace TotalCost
12 {
13    /// <summary>
14    /// Summary description for FrmTotalCost.
15    /// </summary>
16    public class FrmTotalCost : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblCost;
19       private System.Windows.Forms.Label lblTax;
20       private System.Windows.Forms.Label lblBase;
21       private System.Windows.Forms.Label lblOutput;
22       private System.Windows.Forms.TextBox txtTax;
```

```
23          private System.Windows.Forms.TextBox txtPrice;
24          private System.Windows.Forms.Button btnEnter;
25          /// <summary>
26          /// Required designer variable.
27          /// </summary>
28          private System.ComponentModel.Container components = null;
29
30          public FrmTotalCost()
31          {
32             //
33             // Required for Windows Form Designer support
34             //
35             InitializeComponent();
36
37             //
38             // TODO: Add any constructor code after InitializeComponent
39             // call
40             //
41          }
42
43          /// <summary>
44          /// Clean up any resources being used.
45          /// </summary>
46          protected override void Dispose( bool disposing )
47          {
48             if ( disposing )
49             {
50                if ( components != null )
51                {
52                   components.Dispose();
53                }
54             }
55             base.Dispose( disposing );
56          }
57
58          // Windows Form Designer generated code
59
60          /// <summary>
61          /// The main entry point for the application.
62          /// </summary>
63          [STAThread]
64          static void Main()
65          {
66             Application.Run( new FrmTotalCost() );
67          }
68
69          // calculates the total cost of an item including tax
70          private void btnEnter_Click(
71             object sender, System.EventArgs e )
72          {
73             lblOutput.Text = Convert.ToString(
74                Double.Parse( txtPrice.Text ) *
75                Double.Parse( txtTax.Text ) );
76
77          } // end method btnEnter_Click
78
79       } // end class FrmTotalCost
80    }
```

What's wrong with this code? ▶

5.15 The following event handler should multiply two inputs when the user clicks a **Calculate** `Button`. Identify the error(s) in its code. Assume that this application has a `Label`, `lblResult`, and two `TextBoxes`, `txtFirst` and `txtSecond`. Also assume that the input entered into `txtFirst` and `txtSecond` are integers.

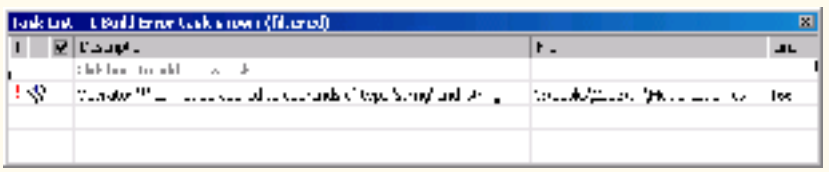
```
1  private void btnCalculate_Click( object sender, System.EventArgs e )
2  {
3     lblResult.Text = txtFirst.Text * txtSecond.Text;
4
5  } // end method btnCalculate_Click
```

Answer: The code should use the `Int32.Parse` method to convert the two inputs to numerical form, as well as the `Convert.ToString` method to put the answer back into string form for display in the `Label`. The complete incorrect code reads:

```
1  // Exercise 5.15 Solution
2  // Multiplication.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Multiplication
12 {
13    /// <summary>
14    /// Summary description for FrmMultiplication.
15    /// </summary>
16    public class FrmMultiplication : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblOutput;
19       private System.Windows.Forms.Label lblResult;
20       private System.Windows.Forms.Button btnCalculate;
21       private System.Windows.Forms.Label lblFirst;
22       private System.Windows.Forms.Label lblSecond;
23       private System.Windows.Forms.TextBox txtFirst;
24       private System.Windows.Forms.TextBox txtSecond;
25       /// <summary>
26       /// Required designer variable.
27       /// </summary>
28       private System.ComponentModel.Container components = null;
29
30       public FrmMultiplication()
31       {
32          //
33          // Required for Windows Form Designer support
```

```
34          //
35          InitializeComponent();
36
37          //
38          // TODO: Add any constructor code after InitializeComponent
39          // call
40          //
41       }
42
43       /// <summary>
44       /// Clean up any resources being used.
45       /// </summary>
46       protected override void Dispose( bool disposing )
47       {
48          if( disposing )
49          {
50             if (components != null)
51             {
52                components.Dispose();
53             }
54          }
55          base.Dispose( disposing );
56       }
57
58       // Windows Form Designer generated code
59
60       /// <summary>
61       /// The main entry point for the application.
62       /// </summary>
63       [STAThread]
64       static void Main()
65       {
66          Application.Run( new FrmMultiplication() );
67       }
68
69       // calculates the total cost of an item including tax
70       private void btnCalculate_Click(
71          object sender, System.EventArgs e )
72       {
73          lblResult.Text = txtFirst.Text * txtSecond.Text;
74
75       } // end method btnCalculate_Click
76
77    } // end class FrmMultiplication
78 }
```

Missing `Double.Parse` and `Convert.ToString`



Answer: The complete corrected code should read:

```
1  // Exercise 5.15 Solution
2  // Multiplication.cs (Correct)
3
```

```
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Multiplication
12 {
13    /// <summary>
14    /// Summary description for FrmMultiplication.
15    /// </summary>
16    public class FrmMultiplication : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblOutput;
19       private System.Windows.Forms.Label lblResult;
20       private System.Windows.Forms.Button btnCalculate;
21       private System.Windows.Forms.Label lblFirst;
22       private System.Windows.Forms.Label lblSecond;
23       private System.Windows.Forms.TextBox txtFirst;
24       private System.Windows.Forms.TextBox txtSecond;
25       /// <summary>
26       /// Required designer variable.
27       /// </summary>
28       private System.ComponentModel.Container components = null;
29
30       public FrmMultiplication()
31       {
32          //
33          // Required for Windows Form Designer support
34          //
35          InitializeComponent();
36
37          //
38          // TODO: Add any constructor code after InitializeComponent
39          // call
40          //
41       }
42
43       /// <summary>
44       /// Clean up any resources being used.
45       /// </summary>
46       protected override void Dispose( bool disposing )
47       {
48          if ( disposing )
49          {
50             if (components != null)
51             {
52                components.Dispose();
53             }
54          }
55          base.Dispose( disposing );
56       }
57
58       // Windows Form Designer generated code
59
60       /// <summary>
61       /// The main entry point for the application.
62       /// </summary>
63       [STAThread]
```

Added `Int32.Parse` and `Convert.ToString`

```
64         static void Main()
65         {
66            Application.Run( new FrmMultiplication() );
67         }
68
69         // calculates the total cost of an item including tax
70         private void btnCalculate_Click(
71            object sender, System.EventArgs e )
72         {
73            lblResult.Text = Convert.ToString(
74               Int32.Parse( txtFirst.Text ) *
75               Int32.Parse( txtSecond.Text ) );
76
77         } // end method btnCalculate_Click
78
79      } // end class FrmMultiplication
80
81   }
```

Using the Debugger ▶

5.16 ***(Account Information Debugging Exercise)*** Copy the directory `C:\Examples\Tutorial05\Exercises\Debugger\AccountInformation` to your `C:\SimplyCSP` directory, and run the **Account Information** application. Remove any syntax and compilation errors so that the application runs correctly.

Answer: Below is the code with syntax and compilation errors.

```
1   using System;
2   using System.Drawing;
3   using System.Collections;
4   using System.ComponentModel;
5   using System.Windows.Forms;
6   using System.Data;
7
8   namespace AccountInformation
9   {
10     /// <summary>
11     /// Summary description for FrmAccountInformation.
12     /// </summary>
13     public class FrmAccountInformation : System.Windows.Forms.Form
14     {
15        private System.Windows.Forms.GroupBox fraInput;
16        private System.Windows.Forms.Label lblNameLabel1;
17        private System.Windows.Forms.TextBox txtName;
18        private System.Windows.Forms.Label lblAccountNumberLabel1;
19        private System.Windows.Forms.TextBox txtAccountNumber;
20        private System.Windows.Forms.Label lblDepositAmountLabel1;
21        private System.Windows.Forms.TextBox txtDepositAmount;
22        private System.Windows.Forms.Button btnEnter;
23        private System.Windows.Forms.GroupBox fraOutput;
24        private System.Windows.Forms.Label lblNameLabel2;
25        private System.Windows.Forms.Label lblCopiedName;
```

```
26          private System.Windows.Forms.Label lblAccountNumberLabel2;
27          private System.Windows.Forms.Label lblCopiedAccountNumber;
28          private System.Windows.Forms.Label lblDepositAmountLabel2;
29          private System.Windows.Forms.Label lblBalance;
30          /// <summary>
31          /// Required designer variable.
32          /// </summary>
33          private System.ComponentModel.Container components = null;
34
35          public FrmAccountInformation()
36          {
37             //
38             // Required for Windows Form Designer support
39             //
40             InitializeComponent();
41
42             //
43             // TODO: Add any constructor code after InitializeComponent
44             // call
45             //
46          }
47
48          /// <summary>
49          /// Clean up any resources being used.
50          /// </summary>
51          protected override void Dispose( bool disposing )
52          {
53             if( disposing )
54             {
55                if (components != null)
56                {
57                   components.Dispose();
58                }
59             }
60             base.Dispose( disposing );
61          }
62
63          // Windows Form Designer generated code
64
65          /// <summary>
66          /// The main entry point for the application.
67          /// </summary>
68          [STAThread]
69          static void Main()
70          {
71             Application.Run( new FrmAccountInformation() );
72          }
73
74          // handles Click event
75          private void btnEnter_Click(
76             object sender, System.EventArgs e )
77          {
78             lblBalance.Text = Convert.ToString(
79                Int32.Parse( txtDepositAmount( Text ) )
80                - Int32.Parse( txtWithdrawalAmount.Text )
81                + Int32.Parse( lblBlance.Text ) )
82
83          } // end method btnEnter_Click
84
```

Property `Text` should be accessed with the member access operator, not parentheses

Misspelling of `lblBalance` and missing semicolon

```
85     } // end class FrmAccountInformation
86  }
```

Answer: Below is the code with all errors corrected.

```
1   // Exercise 5.16 Solution
2   // AccountInformation.cs (Debugger)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace AccountInformation
12  {
13     /// <summary>
14     /// Summary description for FrmAccountInformation.
15     /// </summary>
16     public class FrmAccountInformation : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.GroupBox fraInput;
19        private System.Windows.Forms.Label lblNameLabel1;
20        private System.Windows.Forms.TextBox txtName;
21        private System.Windows.Forms.Label lblAccountNumberLabel1;
22        private System.Windows.Forms.TextBox txtAccountNumber;
23        private System.Windows.Forms.Label lblDepositAmountLabel1;
24        private System.Windows.Forms.TextBox txtDepositAmount;
25        private System.Windows.Forms.Button btnEnter;
26        private System.Windows.Forms.GroupBox fraOutput;
27        private System.Windows.Forms.Label lblNameLabel2;
28        private System.Windows.Forms.Label lblCopiedName;
29        private System.Windows.Forms.Label lblAccountNumberLabel2;
30        private System.Windows.Forms.Label lblCopiedAccountNumber;
31        private System.Windows.Forms.Label lblDepositAmountLabel2;
32        private System.Windows.Forms.Label lblBalance;
33        /// <summary>
34        /// Required designer variable.
35        /// </summary>
36        private System.ComponentModel.Container components = null;
37
38        public FrmAccountInformation()
39        {
40           //
41           // Required for Windows Form Designer support
42           //
43           InitializeComponent();
44
45           //
46           // TODO: Add any constructor code after InitializeComponent
47           // call
48           //
49        }
50
51        /// <summary>
52        /// Clean up any resources being used.
53        /// </summary>
54        protected override void Dispose( bool disposing )
```

```
55         {
56            if( disposing )
57            {
58               if (components != null)
59               {
60                  components.Dispose();
61               }
62            }
63            base.Dispose( disposing );
64         }
65
66         // Windows Form Designer generated code
67
68         /// <summary>
69         /// The main entry point for the application.
70         /// </summary>
71         [STAThread]
72         static void Main()
73         {
74            Application.Run( new FrmAccountInformation() );
75         }
76
77         // handles Click event
78         private void btnEnter_Click(
79            object sender, System.EventArgs e )
80         {
81            lblBalance.Text = Convert.ToString(
82               Int32.Parse( txtDepositAmount.Text )
83               - Int32.Parse( txtWithdrawalAmount.Text )
84               + Int32.Parse( lblBalance.Text ) );
85
86         } // end method btnEnter_Click
87
88      } // end class FrmAccountInformation
89   }
```

Programming Challenge ▶

5.17 ***(Account Information Enhancement)*** Modify Exercise 5.13 so that it no longer asks for the user's name and account number, but rather asks the user for a withdrawal or deposit amount. The user can enter both a withdrawal and deposit amount at the same time. When the **Enter** `Button` is clicked, the balance is updated appropriately (Fig. 5.30).

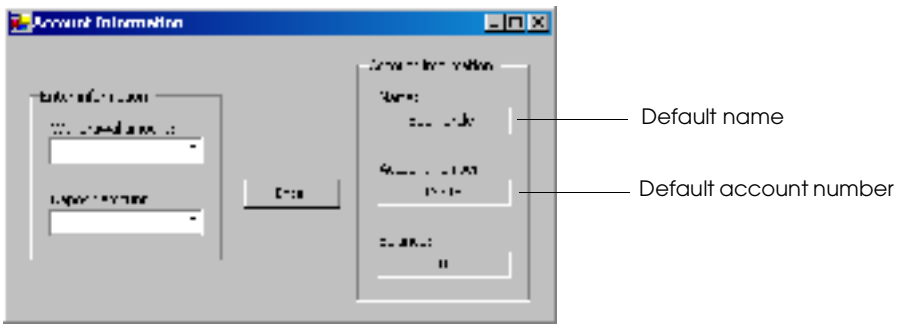


Figure 5.30 Enhanced **Account Information** GUI.

a) ***Copying the template to your working directory.*** If you have not already done so, copy the directory `C:\Examples\Tutorial05\Exercises\AccountInformation` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AccountInformation.sln` in the `AccountInformation` directory to open the application.

c) ***Modifying the GUI.*** Modify the GUI so that it appears as in Fig. 5.30.

d) ***Setting the default values.*** Set the default name and account number to the values shown in Fig. 5.30 using the **Properties** window.

e) ***Writing code to add functionality.*** Update the account balance for every withdrawal (which decreases the balance) and every deposit (which increases the balance). When the balance is updated, reset the `TextBox`es to 0.

f) ***Running the application.*** Select **Debug > Start** to run your application. Begin with the values in Fig. 5.30 when you test your application.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 5.17 Solution
2  // AccountInformation.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace AccountInformation
12 {
13    /// <summary>
14    /// Summary description for FrmAccountInformation.
15    /// </summary>
16    public class FrmAccountInformation : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.GroupBox fraInput;
19       private System.Windows.Forms.Label lblNameLabel1;
20       private System.Windows.Forms.TextBox txtName;
21       private System.Windows.Forms.Label lblAccountNumberLabel1;
22       private System.Windows.Forms.TextBox txtAccountNumber;
23       private System.Windows.Forms.Label lblDepositAmountLabel1;
24       private System.Windows.Forms.TextBox txtDepositAmount;
25       private System.Windows.Forms.Button btnEnter;
26       private System.Windows.Forms.GroupBox fraOutput;
27       private System.Windows.Forms.Label lblNameLabel2;
28       private System.Windows.Forms.Label lblCopiedName;
29       private System.Windows.Forms.Label lblAccountNumberLabel2;
30       private System.Windows.Forms.Label lblCopiedAccountNumber;
31       private System.Windows.Forms.Label lblDepositAmountLabel2;
32       private System.Windows.Forms.Label lblBalance;
33       /// <summary>
34       /// Required designer variable.
35       /// </summary>
36       private System.ComponentModel.Container components = null;
37
38       public FrmAccountInformation()
39       {
40          //
41          // Required for Windows Form Designer support
42          //
43          InitializeComponent();
44
```

```
45          //
46          // TODO: Add any constructor code after InitializeComponent
47          // call
48          //
49       }
50
51       /// <summary>
52       /// Clean up any resources being used.
53       /// </summary>
54       protected override void Dispose( bool disposing )
55       {
56          if( disposing )
57          {
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmAccountInformation() );
75       }
76
77       // handles Click event
78       private void btnEnter_Click(
79          object sender, System.EventArgs e )
80       {
81          lblBalance.Text = Convert.ToString(
82             Int32.Parse( txtDepositAmount.Text )
83             - Int32.Parse( txtWithdrawalAmount.Text )
84             + Int32.Parse( lblBalance.Text ) );
85
86          // reset TextBoxes
87          txtWithdrawalAmount.Text = "0";
88          txtDepositAmount.Text = "0";
89
90       } // end method btnEnter_Click
91
92    } // end class FrmAccountInformation
93 }
```

6

TUTORIAL

Enhancing the Inventory Application

Introducing Variables, Memory Concepts and Arithmetic

Solutions

Instructor's Manual Exercise Solutions Tutorial 6

MULTIPLE-CHOICE QUESTIONS

6.1 Parentheses that are added to an expression simply to make it easier to read are known as _____ parentheses.

a) necessary
b) redundant
c) embedded
d) nested

6.2 The _____ operator performs division.

a) /
b) +
c) \
d) ^

6.3 Every variable has a _____.

a) name
b) value
c) type
d) All of the above.

6.4 In C#, arithmetic expressions must be written in _____ form.

a) straight-line
b) top-bottom
c) left-right
d) right-left

6.5 Arithmetic expressions are evaluated _____.

a) from right to left
b) from left to right
c) according to the rules of operator precedence
d) from the lowest level of precedence to the highest level of precedence

6.6 Variable declarations in event handlers begin with their _____.

a) name
b) value
c) type
d) None of the above.

6.7 Entering a character in a `TextBox` raises the _____ event.

a) `TextAltered`
b) `ValueChanged`
c) `ValueEntered`
d) `TextChanged`

6.8 The _____ operator makes an explicit conversion from one type to another.

a) cast
b) changetype
c) convert
d) conversion

6.9 Variables used to store integer values should be declared with the _____ keyword.

a) `integer`
b) `int`
c) `intvariable`
d) None of the above.

6.10 The name of a variable in a variable declaration should come directly after its _____.

a) type
b) value
c) size
d) All of the above.

Answers: 6.1) b. 6.2) a. 6.3) d. 6.4) a. 6.5) c. 6.6) c. 6.7) d. 6.8) a. 6.9) b. 6.10) a.

EXERCISES

6.11 ***(Simple Encryption Application)*** This application uses a simple technique to encrypt a number. Encryption is the process of modifying data so that only those intended to receive it can undo the changes to view the original data. The user enters the data to be encrypted using a `TextBox`. The application then multiplies the number by `7` and adds `5`. The application displays the encrypted number in a `Label` as shown in Fig. 6.26.

Figure 6.26 Result of the completed **Simple Encryption** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial06\Exercises\SimpleEncryption` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SimpleEncryption.sln` in the `SimpleEncryption` directory to open the application.

c) ***Coding the `Click` event handler.*** Encrypt the number in the `Click` event handler by using the preceding technique. The user input should be stored in an `int` variable (`intNumber`) before it is encrypted. The event handler then should display the encrypted number.

d) ***Clearing the result.*** Add an event handler for the **Enter number to encrypt:** `TextBox`'s `TextChanged` event. This event handler should clear the **Encrypted number:** `TextBox` whenever the user enters new input.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the value `25` into the **Enter number to encrypt:** `TextBox` and click the **Encrypt** `Button`. Verify that the value `180` is displayed in the **Encrypted number:** output `Label`. Enter other values and click the **Encrypt** `Button` after each. Verify that the appropriate encrypted value is displayed each time.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 6.11 Solution
2  // SimpleEncryption.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace SimpleEncryption
12 {
13    /// <summary>
14    /// Summary description for FrmEncryption.
15    /// </summary>
16    public class FrmEncryption : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input number
19       private System.Windows.Forms.Label lblInput;
20       private System.Windows.Forms.TextBox txtInput;
21
22       // Labels to display encrypted number
23       private System.Windows.Forms.Label lblOutput;
24       private System.Windows.Forms.Label lblResult;
25
26       // Button to perform the encryption
27       private System.Windows.Forms.Button btnEncrypt;
28
29       /// <summary>
30       /// Required designer variable.
```

```
31          /// </summary>
32          private System.ComponentModel.Container components = null;
33
34          public FrmEncryption()
35          {
36             //
37             // Required for Windows Form Designer support
38             //
39             InitializeComponent();
40
41          }
42
43          /// <summary>
44          /// Clean up any resources being used.
45          /// </summary>
46          protected override void Dispose( bool disposing )
47          {
48             if( disposing )
49             {
50                if (components != null)
51                {
52                   components.Dispose();
53                }
54             }
55             base.Dispose( disposing );
56          }
57
58          // Windows Form Designer generated code
59
60          /// <summary>
61          /// The main entry point for the application.
62          /// </summary>
63          [STAThread]
64          static void Main()
65          {
66             Application.Run( new FrmEncryption() );
67          }
68
69          // handles Click event
70          private void btnEncrypt_Click(
71             object sender, System.EventArgs e )
72          {
73             int intNumber;
74
75             // obtain user input
76             intNumber = Int32.Parse( txtInput.Text );
77
78             // encrypt number
79             intNumber = intNumber * 7 + 5;
80
81             // display encrypted number
82             lblResult.Text = Convert.ToString( intNumber );
83
84          } // end method btnEncrypt_Click
85
86          // handles TextChanged event
87          private void txtInput_TextChanged(
88             object sender, System.EventArgs e )
89          {
90             lblResult.Text = "";
```

```
91
92          } // end method txtInput_TextChanged
93
94       } // end class FrmEncryption
95    }
```

6.12 ***(Temperature Converter Application)*** Write an application that converts a Celsius temperature, *C*, to its equivalent Fahrenheit temperature, *F*. Figure 6.27 displays the completed application. Use the following formula:

$$F = \frac{9}{5}C + 32$$

Figure 6.27 Completed **Temperature Converter**.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial06\Exercises\TemperatureConversion` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `TemperatureConversion.sln` in the `TemperatureConversion` directory to open the application.

c) ***Coding the `Click` event handler.*** Perform the conversion in the **Convert** `Button`'s `Click` event handler. Define `int` variables to store the user-input Celsius temperature and the result of the conversion. Display the Fahrenheit equivalent of the temperature conversion. Use the cast operator to convert between types. For the most accurate result, make sure you perform floating-point arithmetic and cast to an integer after all calculations have been performed.

d) ***Clearing user input.*** Clear the result in the **Enter a Celsius temperature:** `TextBox`'s `TextChanged` event.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the value `20` into the **Enter a Celsius temperature:** `TextBox` and click the **Convert** `Button`. Verify that the value `68` is displayed in the output `Label`. Enter other Celsius temperatures, click the **Convert** `Button` after each. Use the formula provided above to verify that the proper Fahrenheit equivalent is displayed each time.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 6.12 Solution
2  // TemperatureConversion.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace TemperatureConversion
12 {
13    /// <summary>
14    /// Summary description for FrmTemperatureConverter.
15    /// </summary>
```

```
16    public class FrmTemperatureConverter : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input Celsius temperature
19       private System.Windows.Forms.Label lblInput;
20       private System.Windows.Forms.TextBox txtInput;
21
22       // Labels to display Fahrenheit temperature
23       private System.Windows.Forms.Label lblOutput;
24       private System.Windows.Forms.Label lblResult;
25
26       // Button to perform the conversion
27       private System.Windows.Forms.Button btnConvert;
28
29       /// <summary>
30       /// Required designer variable.
31       /// </summary>
32       private System.ComponentModel.Container components = null;
33
34       public FrmTemperatureConverter()
35       {
36          //
37          // Required for Windows Form Designer support
38          //
39          InitializeComponent();
40
41       }
42
43       /// <summary>
44       /// Clean up any resources being used.
45       /// </summary>
46       protected override void Dispose( bool disposing )
47       {
48          if( disposing )
49          {
50             if (components != null)
51             {
52                components.Dispose();
53             }
54          }
55          base.Dispose( disposing );
56       }
57
58       // Windows Form Designer generated code
59
60       /// <summary>
61       /// The main entry point for the application.
62       /// </summary>
63       [STAThread]
64       static void Main()
65       {
66          Application.Run( new FrmTemperatureConverter() );
67       }
68
69       // handles Click event
70       private void btnConvert_Click(
71          object sender, System.EventArgs e )
72       {
73          // temperature variables
74          int intCelsius;
75          int intFahrenheit;
```

```
76
77          // obtain user input
78          intCelsius = Int32.Parse( txtInput.Text );
79
80          // perform conversion
81          intFahrenheit = ( int ) ( ( 9.0 / 5 ) * intCelsius + 32 );
82
83          lblResult.Text = Convert.ToString( intFahrenheit );
84
85       } // end method btnConvert_Click
86
87       // handles TextChanged event
88       private void txtInput_TextChanged(
89          object sender, System.EventArgs e )
90       {
91          lblResult.Text = "";
92
93       } // end method txtInput_TextChanged
94
95    } // end class FrmTemperatureConverter
96 }
```

6.13 ***(Simple Calculator Application)*** In this exercise, you will add functionality to a simple calculator application. The calculator will allow a user to enter two numbers in the `TextBoxes`. There will be four `Buttons`—labelled +, -, / and *. When the user clicks the `Button` labelled as addition, subtraction, multiplication or division, the application will perform that operation on the numbers in the `TextBoxes` and display the result. The calculator also should clear the calculation result when the user enters new input. Figure 6.28 displays the completed calculator.

Figure 6.28 Result of the **Simple Calculator** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial06\Exercises\SimpleCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SimpleCalculator.sln` in the `SimpleCalculator` directory to open the application.

c) ***Coding the addition `Click` event handler.*** This event handler should add the two numbers and display the result.

d) ***Coding the subtraction `Click` event handler.*** This event handler should subtract the second number from the first number and display the result.

e) ***Coding the multiplication `Click` event handler.*** This event handler should multiply the two numbers and display the result.

f) ***Coding the division `Click` event handler.*** This event handler should divide the first number by the second number and display the result.

g) ***Clearing the result.*** Write event handlers for the `TextBoxes`' `TextChanged` events. Write code to clear the result `Label` (`lblResult`) after the user enters new input into either `TextBox`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a first number and a second number, then verify that each of the `Buttons` works by clicking each, and viewing the output. Repeat this process with two new values and again verify that the proper output is displayed based on which `Button` is clicked.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 6.13 Solution
// SimpleCalculator.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SimpleCalculator
{
   /// <summary>
   /// Summary description for FrmCalculator.
   /// </summary>
   public class FrmCalculator : System.Windows.Forms.Form
   {
      // Label and TextBox to input the first number
      private System.Windows.Forms.Label lblFirstNumber;
      private System.Windows.Forms.TextBox txtFirstNumber;

      // Label and TextBox to input the second number
      private System.Windows.Forms.Label lblSecondNumber;
      private System.Windows.Forms.TextBox txtSecondNumber;

      // Labels to display the result
      private System.Windows.Forms.Label lblOutput;
      private System.Windows.Forms.Label lblResult;

      // Buttons to add, subtract, multiply and divide,
      // respectively
      private System.Windows.Forms.Button btnAdd;
      private System.Windows.Forms.Button btnSubtract;
      private System.Windows.Forms.Button btnMultiply;
      private System.Windows.Forms.Button btnDivide;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCalculator()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
```

```
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmCalculator() );
75       }
76
77       // handles addition Button's Click event
78       private void btnAdd_Click(
79          object sender, System.EventArgs e )
80       {
81          lblResult.Text = Convert.ToString(
82             Int32.Parse( txtFirstNumber.Text ) +
83             Int32.Parse( txtSecondNumber.Text ) );
84
85       } // end method btnAdd_Click
86
87       // handles subtraction Button's Click event
88       private void btnSubtract_Click(
89          object sender, System.EventArgs e )
90       {
91          lblResult.Text = Convert.ToString(
92             Int32.Parse( txtFirstNumber.Text ) -
93             Int32.Parse( txtSecondNumber.Text ) );
94
95       } // end method btnSubtract_Click
96
97       // handles multiplication Button's Click event
98       private void btnMultiply_Click(
99          object sender, System.EventArgs e )
100      {
101         lblResult.Text = Convert.ToString(
102            Int32.Parse( txtFirstNumber.Text ) *
103            Int32.Parse( txtSecondNumber.Text ) );
104
105      } // end method btnMultiply_Click
106
107      // handles division Button's Click event
108      private void btnDivide_Click(
109         object sender, System.EventArgs e )
110      {
111         lblResult.Text = Convert.ToString(
112            Int32.Parse( txtFirstNumber.Text ) /
113            Int32.Parse( txtSecondNumber.Text ) );
114
115      } // end method btnDivide_Click
116
117      private void txtFirstNumber_TextChanged(
```

```
118         object sender, System.EventArgs e )
119      {
120         lblResult.Text = "";
121
122      } // end method txtFirstNumber_TextChanged
123
124      private void txtSecondNumber_TextChanged(
125         object sender, System.EventArgs e )
126      {
127         lblResult.Text = "";
128
129      } // end method txtSecondNumber_TextChanged
130
131   } // end class FrmCalculator
132 }
```

What does this code do? ▶

6.14 This code modifies the values of `intNumber1`, `intNumber2` and `intResult`. What are the final values of these variables?

```
1  int intNumber1;
2  int intNumber2;
3  int intResult;
4
5  intNumber1 = 5 * ( 4 + 6 );
6  intNumber2 = 2 * 2;
7  intResult = intNumber1 / intNumber2;
```

Answer: `intNumber1` gets `50`, `intNumber2` gets `4`; `intResult` gets `12`. The complete code reads:

```
1  // Exercise 6.14 Solution
2  // MathExample.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace MathExample
12 {
13    /// <summary>
14    /// Summary description for FrmMathExample.
15    /// </summary>
16    public class FrmMathExample : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblResultOut;
19       private System.Windows.Forms.Label lblNumber2Out;
20       private System.Windows.Forms.Label lblNumber1Out;
21       private System.Windows.Forms.Label lblResult;
22       private System.Windows.Forms.Label lblNumber2;
23       private System.Windows.Forms.Label lblNumber1;
24       /// <summary>
25       /// Required designer variable.
26       /// </summary>
27       private System.ComponentModel.Container components = null;
```

```
28
29          public FrmMathExample()
30          {
31             //
32             // Required for Windows Form Designer support
33             //
34             InitializeComponent();
35
36          }
37
38          /// <summary>
39          /// Clean up any resources being used.
40          /// </summary>
41          protected override void Dispose( bool disposing )
42          {
43             if( disposing )
44             {
45                if (components != null)
46                {
47                   components.Dispose();
48                }
49             }
50             base.Dispose( disposing );
51          }
52
53          // Windows Form Designer generated code
54
55          /// <summary>
56          /// The main entry point for the application.
57          /// </summary>
58          [STAThread]
59          static void Main()
60          {
61             Application.Run( new FrmMathExample() );
62          }
63
64          // perform several calculations and display the results
65          private void FrmMathExample_Load(
66             object sender, System.EventArgs e )
67          {
68             int intNumber1;
69             int intNumber2;
70             int intResult;
71
72             intNumber1 = 5 * ( 4 + 6 );
73             intNumber2 = 2 * 2;
74             intResult = intNumber1 / intNumber2;
75
76             lblNumber1Out.Text = Convert.ToString( intNumber1 );
77             lblNumber2Out.Text = Convert.ToString( intNumber2 );
78             lblResultOut.Text = Convert.ToString( intResult );
79
80          } // end method FrmMathExample_Load
81
82       } // end class FrmMathExample
83    }
```

What's wrong with this code? ▶

6.15 Find the error(s) in the following code, which uses variables to perform a calculation.

```
int intNumber1;
int intNumber2;
int intResult;

intNumber1 = 4 * 6 / ( 10 % 4 - 2 );
intNumber2 = 16 / 3 * 6 + 1;
intResult = intNumber1 - intNumber2;
```

Answer: `intNumber1`'s assignment statement contains a division by zero, which will cause a syntax error to occur. The complete incorrect code reads:

```
// Exercise 6.15 Solution
// MathExample.cs (Incorrect)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace MathExample
{
   /// <summary>
   /// Summary description for FrmMathExample.
   /// </summary>
   public class FrmMathExample : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblResultOut;
      private System.Windows.Forms.Label lblNumber2Out;
      private System.Windows.Forms.Label lblNumber1Out;
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblNumber2;
      private System.Windows.Forms.Label lblNumber1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmMathExample()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

      }
```

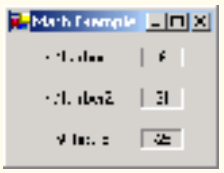
```
37
38          /// <summary>
39          /// Clean up any resources being used.
40          /// </summary>
41          protected override void Dispose( bool disposing )
42          {
43             if( disposing )
44             {
45                if (components != null)
46                {
47                   components.Dispose();
48                }
49             }
50             base.Dispose( disposing );
51          }
52
53          // Windows Form Designer generated code
54
55          /// <summary>
56          /// The main entry point for the application.
57          /// </summary>
58          [STAThread]
59          static void Main()
60          {
61             Application.Run( new FrmMathExample() );
62          }
63
64          // performs several calculations and displays the results
65          private void FrmMathExample_Load(
66              object sender, System.EventArgs e )
67          {
68             int intNumber1;
69             int intNumber2;
70             int intResult;
71
72             // conversions from variables that hold more data (double)
73             // into variables that hold less data (int) require an
74             // explicit cast
75             intNumber1 = 4 * 6 / ( 10 % 4 - 2 );
76             intNumber2 = 16 / 3 * 6 + 1;
77             intResult = intNumber1 - intNumber2;
78
79             lblNumber1Out.Text = Convert.ToString( intNumber1 );
80             lblNumber2Out.Text = Convert.ToString( intNumber2 );
81             lblResultOut.Text = Convert.ToString( intResult );
82
83          } // end method FrmMathExample_Load
84
85       } // end class FrmMathExample
86    }
```

10 % 4 – 2 evaluates to zero. Division by zero causes an error

Answer: There are several ways to correct this code that removes the division by zero. Below is just one solution:

```
// Exercise 6.15 Solution
// MathExample.cs (Correct)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace MathExample
{
   /// <summary>
   /// Summary description for FrmMathExample.
   /// </summary>
   public class FrmMathExample : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblResultOut;
      private System.Windows.Forms.Label lblNumber2Out;
      private System.Windows.Forms.Label lblNumber1Out;
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblNumber2;
      private System.Windows.Forms.Label lblNumber1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmMathExample()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
```

```
58          [STAThread]
59          static void Main()
60          {
61             Application.Run( new FrmMathExample() );
62          }
63
64          // performs several calculations and displays the results
65          private void FrmMathExample_Load(
66             object sender, System.EventArgs e )
67          {
68             int intNumber1;
69             int intNumber2;
70             int intResult;
71
72             // conversions from variables that hold more data (double)
73             // into variables that hold less data (int) require an
74             // explicit cast
75             intNumber1 = 4 * 6 / ( 10 % 4 + 2 );
76             intNumber2 = 16 / 3 * 6 + 1;
77             intResult = intNumber1 - intNumber2;
78
79             lblNumber1Out.Text = Convert.ToString( intNumber1 );
80             lblNumber2Out.Text = Convert.ToString( intNumber2 );
81             lblResultOut.Text = Convert.ToString( intResult );
82
83          } // end method FrmMathExample_Load
84
85       } // end class FrmMathExample
86    }
```



Using the Debugger ▶

6.16 ***(Average Three Numbers Application)*** You have just written an application that takes three numbers as input in `TextBoxes`, stores the three numbers in variables, then finds the average of the numbers. The output is displayed in a `Label` (Fig. 6.29, which displays the incorrect output). You soon realize, however, that the number displayed in the `Label` is not the average, but rather a number that does not make sense given the input. Use the debugger to help locate and remove this error.

Figure 6.29 **Average Three Numbers** application running incorrectly.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial06\Exercises\Debugger\AverageDebugging` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AverageDebugging.sln` in the `AverageDebugging` directory to open the application.

c) ***Setting breakpoints.*** Set a breakpoint in the `btnCalculate_Click` event handler. Run the application again, and use the debugger to help find the error(s).

d) ***Finding and correcting the error(s).*** Once you have found the error(s), modify the application so that it correctly calculates the average of three numbers.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the three values from Fig. 6.29 into the input `TextBoxes` provided and click the **Calculate** `Button`. Verify that the output now accurately reflects the average of these values, which is `8`.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer: The original code only divided the third number (`intNumber3`) by `3` when in fact the average ought to have been the sum of `intNumber1`, `intNumber2` and `intNumber3` divided by `3`. To correct the error, we included proper parentheses before `intNumber1` and after `intNumber3`.

```
1  // Exercise 6.16 Solution
2  // AverageDebugging.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace AverageDebugging
12 {
13    /// <summary>
14    /// Summary description for FrmAverageDebugging.
15    /// </summary>
16    public class FrmAverageDebugging : System.Windows.Forms.Form
17    {
18       // Labels and TextBoxes to input the three values
19       private System.Windows.Forms.Label lblFirst;
20       private System.Windows.Forms.TextBox txtFirst;
21
22       private System.Windows.Forms.Label lblSecond;
23       private System.Windows.Forms.TextBox txtSecond;
24
25       private System.Windows.Forms.Label lblThird;
26       private System.Windows.Forms.TextBox txtThird;
27
28       // Labels to display the average value
29       private System.Windows.Forms.Label lblAverage;
30       private System.Windows.Forms.Label lblOutput;
31
32       // Button to calculate the average value
33       private System.Windows.Forms.Button btnCalculate;
34
35       /// <summary>
36       /// Required designer variable.
37       /// </summary>
38       private System.ComponentModel.Container components = null;
```

```
39
40          public FrmAverageDebugging()
41          {
42             //
43             // Required for Windows Form Designer support
44             //
45             InitializeComponent();
46
47          }
48
49          /// <summary>
50          /// Clean up any resources being used.
51          /// </summary>
52          protected override void Dispose( bool disposing )
53          {
54             if( disposing )
55             {
56                if (components != null)
57                {
58                   components.Dispose();
59                }
60             }
61             base.Dispose( disposing );
62          }
63
64          // Windows Form Designer generated code
65
66          /// <summary>
67          /// The main entry point for the application.
68          /// </summary>
69          [STAThread]
70          static void Main()
71          {
72             Application.Run( new FrmAverageDebugging() );
73          }
74
75          // handles Click event
76          private void btnCalculate_Click(
77             object sender, System.EventArgs e )
78          {
79             // variables to store user inputs
80             int intNumber1;
81             int intNumber2;
82             int intNumber3;
83             int intAverage;
84
85             // obtain user inputs
86             intNumber1 = Int32.Parse( txtFirst.Text );
87             intNumber2 = Int32.Parse( txtSecond.Text );
88             intNumber3 = Int32.Parse( txtThird.Text );
89
90             // average numbers
91             intAverage = ( intNumber1 + intNumber2 + intNumber3 ) / 3;
92
93             // display result
94             lblOutput.Text = Convert.ToString( intAverage );
95
96          } // end method btnCalculate_Click
97
```

```
98      } // end class FrmAverageDebugging
99  }
```

Programming Challenge ▶

6.17 ***(Digit Extractor Application)*** Write an application that allows the user to enter a five-digit number into a `TextBox`. The application then separates the number into its individual digits and displays each digit in a `Label`. The application should look and behave similarly to Fig. 6.30. [*Hint:* You can use the `%` operator to extract the ones digit from a number. For instance, `12345 % 10` is `5`. You can use integer division (`/`) to "peel off" digits from a number. For instance, `12345 / 100` is `123`. This allows you to treat the `3` in `12345` as a ones digit. Now you can isolate the `3` by using the `%` operator. Apply this technique to the rest of the digits.]

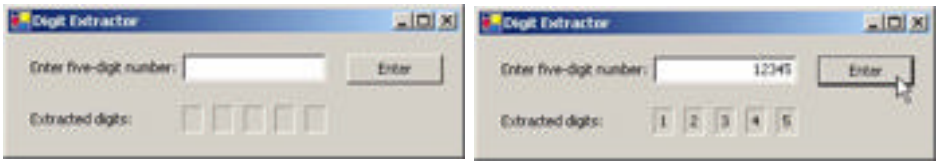


Figure 6.30 **Digit Extractor** application GUI.

a) ***Creating the application.*** Create a new project named `DigitExtractor`. Rename the `Form1.cs` file `DigitExtractor.cs`. Change the name of the `Form` to `FrmDigitExtractor`. Add `Labels`, a `TextBox` and a `Button` to the application's `Form`. Name the `TextBox txtInput` and name the `Button btnEnter`. Name the other controls logically based on the tips provided in earlier tutorials.

b) ***Adding an event handler for `btnEnter`'s `Click` event.*** In design view, double click `btnEnter` to create the `btnEnter_Click` event handler. In this event handler, create five variables of type `int`. Use the `%` operator to extract each digit. Store the digits in the five variables created.

c) ***Adding an event handler for `txtInput`'s `TextChanged` event.*** In design view, double click `txtInput` to create the `txtInput_TextChanged` event handler. In this event handler, clear the five `Labels` used to display each digit. This event handler clears the output whenever new input is entered.

d) ***Running the application.*** Select **Debug > Start** to run your application. Enter a five-digit number and click the **Enter** `Button`. Enter a new five-digit number and verify that the previous output is cleared.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1   // Exercise 6.17 Solution
2   // DigitExtractor.cs
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace DigitExtractor
12  {
13     /// <summary>
14     /// Summary description for FrmDigitExtractor.
15     /// </summary>
16     public class FrmDigitExtractor : System.Windows.Forms.Form
17     {
18        // Label and TextBox to input five-digit number
19        private System.Windows.Forms.Label lblInput;
```

```
20          private System.Windows.Forms.TextBox txtInput;
21
22          // Labels to display the five extracted digits
23          private System.Windows.Forms.Label lblOutput;
24          private System.Windows.Forms.Label lblFirstDigit;
25          private System.Windows.Forms.Label lblSecondDigit;
26          private System.Windows.Forms.Label lblThirdDigit;
27          private System.Windows.Forms.Label lblFourthDigit;
28          private System.Windows.Forms.Label lblFifthDigit;
29
30          // Button to enter the five-digit number
31          private System.Windows.Forms.Button btnEnter;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          public FrmDigitExtractor()
39          {
40             //
41             // Required for Windows Form Designer support
42             //
43             InitializeComponent();
44
45          }
46
47          /// <summary>
48          /// Clean up any resources being used.
49          /// </summary>
50          protected override void Dispose( bool disposing )
51          {
52             if( disposing )
53             {
54                if (components != null)
55                {
56                   components.Dispose();
57                }
58             }
59             base.Dispose( disposing );
60          }
61
62          // Windows Form Designer generated code
63
64          /// <summary>
65          /// The main entry point for the application.
66          /// </summary>
67          [STAThread]
68          static void Main()
69          {
70             Application.Run( new FrmDigitExtractor() );
71          }
72
73          // handles Click event
74          private void btnEnter_Click(
75             object sender, System.EventArgs e )
76          {
77             int intNumber; // five-digit number
78
79             // five variables for five digits
```

```
80            int intFirst;
81            int intSecond;
82            int intThird;
83            int intFourth;
84            int intFifth;
85
86            // obtain user input
87            intNumber = Int32.Parse( txtInput.Text );
88
89            // extract each digit
90            intFirst = intNumber / 10000;
91            intSecond = intNumber / 1000 % 10;
92            intThird = intNumber / 100 % 10;
93            intFourth = intNumber / 10 % 10;
94            intFifth = intNumber % 10;
95
96            // display extracted digits
97            lblFirstDigit.Text = Convert.ToString( intFirst );
98            lblSecondDigit.Text = Convert.ToString( intSecond );
99            lblThirdDigit.Text = Convert.ToString( intThird );
100           lblFourthDigit.Text = Convert.ToString( intFourth );
101           lblFifthDigit.Text = Convert.ToString( intFifth );
102
103        } // end method btnEnter_Click
104
105        // handles TextChanged event
106        private void txtInput_TextChanged(
107           object sender, System.EventArgs e )
108        {
109           // clear Labels
110           lblFirstDigit.Text = "";
111           lblSecondDigit.Text = "";
112           lblThirdDigit.Text = "";
113           lblFourthDigit.Text = "";
114           lblFifthDigit.Text = "";
115
116        } // end method txtInput_TextChanged
117
118     } // end class FrmDigitExtractor
119  }
```

TUTORIAL 7

Wage Calculator Application

Introducing Algorithms, Pseudocode and Program Control

Solutions

Instructor's Manual Exercise Solutions Tutorial 7

MULTIPLE-CHOICE QUESTIONS

7.1 The _____ operator returns `false` if the left operand is larger than the right operand.

a) `==` b) `<`
c) `<=` d) All of the above.

7.2 A _____ occurs when an executed statement does not directly follow the previously executed statement in the written application.

a) transition b) flow
c) logical error d) transfer of control

7.3 A variable or an expression that is added to the **Watch** window is known as a _____.

a) watched variable b) watched expression
c) watch d) watched value

7.4 The `if` statement is called a _____ statement because it selects or ignores one action or sequence of actions.

a) single-selection b) multiple-selection
c) double-selection d) repetition

7.5 The three types of control statements are the sequence statement, the selection statement and the _____ statement.

a) repeat b) looping
c) redo d) repetition

7.6 In an activity diagram, a rectangle with curved sides represents _____.

a) a complete algorithm b) a comment
c) an action d) the termination of the application

7.7 The `if...else` selection statement ends with a(n) _____.

a) right brace (`}`) b) `endif` statement
c) `endelse` statement d) double-selection statement

7.8 A variable of type `bool` can be assigned the _____ keyword or the _____ keyword.

a) `true`, `false` b) `off`, `on`
c) `true`, `notTrue` d) `yes`, `no`

7.9 A variable whose value cannot be changed after its initial declaration is called a _____.

a) `double` b) constant
c) standard d) `bool`

7.10 The _____ operator assigns the result of adding the left and right operands to the left operand.

a) `+` b) `=+`
c) `+=` d) `++`

Answers: 7.1) d. 7.2) d. 7.3) c. 7.4) a. 7.5) d. 7.6) c. 7.7) a. 7.8) a. 7.9) b. 7.10) c.

EXERCISES

7.11 ***(Currency Converter Application)*** Develop an application that functions as a currency converter (Fig. 7.31). Users must provide a number in the **Dollars:** `TextBox` and a currency name (as text) in the **Convert from Dollars to:** `TextBox`. Clicking the **Convert** `Button` will convert the specified amount into the indicated currency and display it in a `Label`. Limit

yourself to the following currencies as user input: Dollars, Euros, Yen and Pesos. Use the following exchange rates: 1 Dollar = 1.02 Euros, 120 Yen and 10 Pesos. Use the `M` suffix to convert `1.02` to a `decimal`.

Figure 7.31 Currency Converter GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial07\Exercises\CurrencyConverter` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CurrencyConverter.sln` in the `CurrencyConverter` directory to open the application.

c) ***Add an event handler for the Convert `Button`'s `Click` event.*** Double click the **Convert** `Button` to generate an empty event handler for the `Button`'s `Click` event. The code for *Steps d–f* belongs in this event handler.

d) ***Obtaining the user input.*** Use the `Decimal.Parse` method to convert the user input from the **Dollars:** `TextBox` to a `decimal`. Assign the `decimal` to a variable `decAmount`.

e) ***Performing the conversion.*** Use an `if`...`else` statement to determine which currency the user entered. Assign the result of the conversion to `decAmount`.

f) ***Displaying the result.*** Display the result using the `String.Format` method with `F` format specifier F.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter a value in dollars to convert and the currency you wish to convert to. Click the **Convert** `Button` and, using the exchange rates, verify that the correct output is displayed.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 7.11 Solution
2  // CurrencyConverter.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace CurrencyConverter
12 {
13    /// <summary>
14    /// Summary description for FrmCurrencyConverter.
15    /// </summary>
16    public class FrmCurrencyConverter : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input amount of dollars
19       private System.Windows.Forms.Label lblConvert;
20       private System.Windows.Forms.TextBox txtValue;
21
22       // Label and TextBox to input currency to convert into
23       private System.Windows.Forms.Label lblTo;
24       private System.Windows.Forms.TextBox txtCurrency;
```

```
25
26          // Labels to output converted value
27          private System.Windows.Forms.Label lblOutput;
28          private System.Windows.Forms.Label lblConvertedResult;
29
30          // Button to convert dollars into another currency
31          private System.Windows.Forms.Button btnConvert;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          public FrmCurrencyConverter()
39          {
40             //
41             // Required for Windows Form Designer support
42             //
43             InitializeComponent();
44
45          }
46
47          /// <summary>
48          /// Clean up any resources being used.
49          /// </summary>
50          protected override void Dispose( bool disposing )
51          {
52             if( disposing )
53             {
54                if (components != null)
55                {
56                   components.Dispose();
57                }
58             }
59             base.Dispose( disposing );
60          }
61
62          // Windows Form Designer generated code
63
64          /// <summary>
65          /// The main entry point for the application.
66          /// </summary>
67          [STAThread]
68          static void Main()
69          {
70             Application.Run( new FrmCurrencyConverter() );
71          }
72
73          // handles Click event
74          private void btnConvert_Click(
75             object sender, System.EventArgs e )
76          {
77             decimal decAmount;
78
79             // obtain dollar amount
80             decAmount = Decimal.Parse( txtValue.Text );
81
82             // perform currency conversion
83             if ( txtCurrency.Text == "Euros" )
84             {
```

```
85             decAmount *= 1.02M;
86          }
87          else if ( txtCurrency.Text == "Yen" )
88          {
89             decAmount *= 120;
90          }
91          else if ( txtCurrency.Text == "Pesos" )
92          {
93             decAmount *= 10;
94          }
95
96          lblConvertedResult.Text =
97             String.Format( "{0:F}", decAmount );
98
99       } // end class btnConvert_Click
100
101   } // end class FrmCurrencyConverter
102 }
```

7.12 ***(Expanded Wage Calculator that Performs Tax Calculations)*** Develop an application that calculates an employee's wages (Fig. 7.32). The user should provide the hourly wage and number of hours worked per week. When the **Calculate** `Button` is clicked, display the gross earnings in the **Gross earnings:** `TextBox`. The **Less FWT:** `TextBox` should display the amount deducted for Federal taxes and the **Net earnings:** `TextBox` should display the difference between the gross earnings and the Federal tax amount. Assume overtime wages are `1.5` times the hourly wage and Federal taxes are 15% of gross earnings. The **Clear** `Button` clears all fields.

Figure 7.32 **Expanded Wage Calculator** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial07\Exercises\ExpandedWageCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `WageCalculator.sln` in the `ExpandedWageCalculator` directory to open the application.

c) ***Modifying the* Calculate *`Button`'s `Click` event handler.*** Add the code for *Steps d–f* to `btnCalculate_Click`.

d) ***Adding a new variable.*** Declare `decFederalTaxes` to store the amount deducted for Federal taxes.

e) ***Calculating and displaying the Federal taxes deducted.*** Multiply the total earnings (`decEarnings`) by `0.15` (that is, 15%) to determine the amount to be removed for taxes. Use the `M` suffix to convert `0.15` to a `decimal`. Assign the result to `decFederalTaxes`. Display this value using the `String.Format` method with the `C` format specifier `C`.

f) ***Calculating and displaying the employee's net pay.*** Subtract `decFederalTaxes` from `decEarnings` to calculate the employee's net earnings. Display this value using the `String.Format` method with format specifier `C`.

g) ***Creating an event handler for the Clear Button.*** Double click the **Clear** `Button` to generate an empty event handler for the `Click` event. This event handler should clear user input from the two `TextBox`es and the results from the three `Label`s.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter an hourly wage and the number of hours worked. Click the **Calculate** `Button` and verify that the appropriate output is displayed for gross earnings, amount taken out for federal taxes and the net earnings. Click the **Clear** `Button` and check that all fields are cleared.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 7.12 Solution
// WageCalculator.cs (Expanded)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WageCalculator
{
   /// <summary>
   /// Summary description for FrmWageCalculator.
   /// </summary>
   public class FrmWageCalculator : System.Windows.Forms.Form
   {
      // Label and TextBox for hourly wage
      private System.Windows.Forms.Label lblWage;
      private System.Windows.Forms.TextBox txtWage;

      // Label and Textbox for weekly hours
      private System.Windows.Forms.Label lbHours;
      private System.Windows.Forms.TextBox txtHours;

      // Labels to display gross earnings
      private System.Windows.Forms.Label lblEarnings;
      private System.Windows.Forms.Label lblEarningsResult;

      // Labels to display amount deducted for taxes
      private System.Windows.Forms.Label lblFWT;
      private System.Windows.Forms.Label lblFWTNumber;

      // Labels to display total earnings after taxes
      private System.Windows.Forms.Label lblTotalEarnings;
      private System.Windows.Forms.Label lblTotal;

      // Button to calculate total earnings
      private System.Windows.Forms.Button btnCalculate;

      // Button to clear the Form
      private System.Windows.Forms.Button btnClear;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

```

```
49          public FrmWageCalculator()
50          {
51             //
52             // Required for Windows Form Designer support
53             //
54             InitializeComponent();
55
56          }
57
58          /// <summary>
59          /// Clean up any resources being used.
60          /// </summary>
61          protected override void Dispose( bool disposing )
62          {
63             if( disposing )
64             {
65                if (components != null)
66                {
67                   components.Dispose();
68                }
69             }
70             base.Dispose( disposing );
71          }
72
73          // Windows Form Designer generated code
74
75          /// <summary>
76          /// The main entry point for the application.
77          /// </summary>
78          [STAThread]
79          static void Main()
80          {
81             Application.Run( new FrmWageCalculator() );
82          }
83
84          // handles Click event
85          private void btnCalculate_Click(
86             object sender, System.EventArgs e )
87          {
88             // declare variables
89             double dblHours;
90             decimal decWage;
91             decimal decEarnings;
92             decimal decFederalTaxes;
93
94             const int intHOUR_LIMIT = 40; // declare constant
95
96             // assign values from user input
97             dblHours = Double.Parse( txtHours.Text );
98             decWage = Decimal.Parse( txtWage.Text );
99
100            // determine wage amount
101            if ( dblHours <= intHOUR_LIMIT )
102            {
103               // if under or equal to 40 hours, regular wages
104               decEarnings = ( decimal ) dblHours * decWage;
105            }
106            else
107            {
108               // if over 40 hours, regular wages for first 40
```

```
109             decEarnings = intHOUR_LIMIT * decWage;
110
111             // time and a half for the additional hours
112             decEarnings += ( decimal ) ( dblHours - intHOUR_LIMIT )
113                * ( 1.50M * decWage );
114          }
115
116          // assign the result to its corresponding Label
117          lblEarningsResult.Text =
118             String.Format( "{0:C}", decEarnings );
119
120          // assign federal taxes to the corresponding Label
121          decFederalTaxes = decEarnings * 0.15M;
122          lblFWTNumber.Text =
123             String.Format( "{0:C}", decFederalTaxes );
124
125          // assign net pay to the corresponding Label
126          lblTotal.Text =
127             String.Format( "{0:C}", decEarnings - decFederalTaxes );
128
129       } // end method btnCalculate_Click
130
131       private void btnClear_Click(
132          object sender, System.EventArgs e )
133       {
134          // clear each TextBox and output Label
135          txtWage.Text = "";
136          txtHours.Text = "";
137          lblEarningsResult.Text = "";
138          lblFWTNumber.Text = "";
139          lblTotal.Text = "";
140
141       } // end method btnClear_Click
142
143    } // end class FrmWageCalculator
144 }
```

7.13 ***(Credit Checker Application)*** Develop an application that determines whether a department-store customer has exceeded the credit limit on a charge account (Fig. 7.33). Each customer enters an account number (an `int`), a balance at the beginning of the month (a `decimal`), the total of all items charged for the month (a `decimal`), the total of all credits applied to the customer's account for the month (a `decimal`), and the customer's allowed credit limit (a `decimal`). The application should input each of these facts, calculate the new balance (*beginning balance – credits + charges*), display the new balance and determine whether the new balance exceeds the customer's credit limit. If the customer's credit limit is exceeded, the application should display a message (in a `Label` at the bottom of the `Form`) informing the customer of this fact.

Figure 7.33 Credit Checker GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial07\Exercises\CreditChecker` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CreditChecker.sln` in the `CreditChecker` directory to open the application.

c) ***Adding the Calculate `Button`'s `Click` event handler.*** Double click the **Calculate** `Button` to generate the empty event handler for the `Click` event. The code for *Steps d–g* is added to this event handler.

d) ***Declaring variables.*** Declare an `int` variable to store the account number. Declare four `decimal` variables to store the starting balance, charges, credits and credit limit. Declare a fifth `decimal` variable to store the new balance in the account after the credits and charges have been applied.

e) ***Obtaining user input.*** Obtain the user input from the `TextBox`es' `Text` properties.

f) ***Calculating and displaying the new balance.*** Calculate the new balance by subtracting the total credits from the starting balance and adding the charges. Assign the result to a variable. Display the result formatted as currency.

g) ***Determining if the credit limit has been exceeded.*** If the new balance exceeds the specified credit limit, a message should be displayed in `lblError`.

h) ***Handling the Account number: `TextBox`'s `TextChanged` event.*** Double click the **Account number:** `TextBox` to create its `TextChanged` event handler. This event handler should clear the other `TextBox`es, the error message `Label` and the result `Label`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter an account number, your starting balance, the amount charged to your account, the amount credited to your account and your credit limit. Click the **Calculate Balance** `Button` and verify that the new balance displayed is correct. Enter an amount charged that exceeds your credit limit. Click the **Calculate Balance** `Button` and ensure that a message is displayed in the lower `Label`.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 7.13 Solution
2  // CreditChecker.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
```

```
10
11 namespace CreditChecker
12 {
13    /// <summary>
14    /// Summary description for FrmCreditChecker.
15    /// </summary>
16    public class FrmCreditChecker : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input account number
19       private System.Windows.Forms.Label lblAccountNumber;
20       private System.Windows.Forms.TextBox txtAccountNumber;
21
22       // Label and TextBox to input starting balance
23       private System.Windows.Forms.Label lblStartBalance;
24       private System.Windows.Forms.TextBox txtStartBalance;
25
26       // Label and TextBox to input total charges
27       private System.Windows.Forms.Label lblTotalCharges;
28       private System.Windows.Forms.TextBox txtTotalCharges;
29
30       // Label and TextBox to input total credits
31       private System.Windows.Forms.Label lblTotalCredits;
32       private System.Windows.Forms.TextBox txtTotalCredits;
33
34       // Label and TextBox to input credit limit
35       private System.Windows.Forms.Label lblCreditLimit;
36       private System.Windows.Forms.TextBox txtCreditLimit;
37
38       // Labels to display new balance amount
39       private System.Windows.Forms.Label lblNewBalance;
40       private System.Windows.Forms.Label lblNewBalanceNumber;
41
42       // Label to display a message if the credit limit is
43       // exceeded
44       private System.Windows.Forms.Label lblError;
45
46       // Button to calculate the new balance
47       private System.Windows.Forms.Button btnCalculate;
48
49       /// <summary>
50       /// Required designer variable.
51       /// </summary>
52       private System.ComponentModel.Container components = null;
53
54       public FrmCreditChecker()
55       {
56          //
57          // Required for Windows Form Designer support
58          //
59          InitializeComponent();
60
61       }
62
63       /// <summary>
64       /// Clean up any resources being used.
65       /// </summary>
66       protected override void Dispose( bool disposing )
67       {
68          if( disposing )
69          {
```

```
70             if (components != null)
71             {
72                components.Dispose();
73             }
74          }
75          base.Dispose( disposing );
76       }
77
78       // Windows Form Designer generated code
79
80       /// <summary>
81       /// The main entry point for the application.
82       /// </summary>
83       [STAThread]
84       static void Main()
85       {
86          Application.Run( new FrmCreditChecker() );
87       }
88
89       // handles Calculate Button's Click event
90       private void btnCalculate_Click(
91          object sender, System.EventArgs e )
92       {
93          // clear output from previous calculations
94          lblError.Text = "";
95
96          // declare variables
97          int intAccountNumber;
98          decimal decStartBalance;
99          decimal decTotalCharges;
100         decimal decTotalCredits;
101         decimal decCreditLimit;
102         decimal decNewBalance;
103
104         // obtain user input
105         intAccountNumber = Int32.Parse( txtAccountNumber.Text );
106         decStartBalance = Decimal.Parse( txtStartBalance.Text );
107         decTotalCharges = Decimal.Parse( txtTotalCharges.Text );
108         decTotalCredits = Decimal.Parse( txtTotalCredits.Text );
109         decCreditLimit = Decimal.Parse( txtCreditLimit.Text );
110
111         // calculate balance after credits and charges
112         decNewBalance = decStartBalance -
113            decTotalCredits + decTotalCharges;
114
115         // display new balance in corresponding Label
116         lblNewBalanceNumber.Text =
117            String.Format( "{0:C}", decNewBalance );
118
119         // determine if credit limit has been exceeded
120         if ( decNewBalance > decCreditLimit )
121         {
122            // if credit limit exceeded, display error message
123            lblError.Text = "Credit Limit Exceeded!";
124         }
125
126      } // end method btnCalculate_Click
127
128      // handles TextChanged event
129      private void txtAccountNumber_TextChanged(
```

```
130         object sender, System.EventArgs e )
131      {
132         // clear all fields when account number is changed
133         txtStartBalance.Text = "";
134         txtTotalCharges.Text = "";
135         txtTotalCredits.Text = "";
136         txtCreditLimit.Text = "";
137         lblNewBalanceNumber.Text = "";
138         lblError.Text = "";
139
140      } // end method txtAccountNumber_TextChanged
141
142   } // end class FrmCreditChecker
143 }
```

What does this code do? ▶

7.14 Assume that `txtAge` is a `TextBox` control and that the user has entered the value `27` into this `TextBox`. Determine the action performed by the following code:

```
1  int intAge;
2
3  intAge = Int32.Parse( txtAge.Text );
4
5  if ( intAge < 0 )
6  {
7     txtAge.Text = "Enter a value greater than or equal to zero.";
8  }
9  else if ( intAge < 13 )
10 {
11    txtAge.Text = "Child";
12 }
13 else if ( intAge < 20 )
14 {
15    txtAge.Text = "Teenager";
16 }
17 else if ( intAge < 30 )
18 {
19    txtAge.Text = "Young adult";
20 }
21 else if ( intAge < 65 )
22 {
23    txtAge.Text = "Adult";
24 }
25 else
26 {
27    txtAge.Text = "Senior Citizen";
28 }
```

Answer: This code prints text when an age is inputted into the `txtAge` `TextBox`. In this case, the statement `txtAge.Text = "Young adult";` executes, because the value of `intAge` is less than `30`, but not less than `20`. The complete code reads:

```
1  // Exercise 7.14 Solution
2  // AgeGroup.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
```

```
 7  using System.ComponentModel;
 8  using System.Windows.Forms;
 9  using System.Data;
10
11  namespace AgeGroup
12  {
13     /// <summary>
14     /// Summary description for FrmAgeGroup.
15     /// </summary>
16     public class FrmAgeGroup : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Button btnAgeGroup;
19        private System.Windows.Forms.TextBox txtAge;
20        private System.Windows.Forms.Label lblAge;
21        /// <summary>
22        /// Required designer variable.
23        /// </summary>
24        private System.ComponentModel.Container components = null;
25
26        public FrmAgeGroup()
27        {
28           //
29           // Required for Windows Form Designer support
30           //
31           InitializeComponent();
32
33        }
34
35        /// <summary>
36        /// Clean up any resources being used.
37        /// </summary>
38        protected override void Dispose( bool disposing )
39        {
40           if( disposing )
41           {
42              if (components != null)
43              {
44                 components.Dispose();
45              }
46           }
47           base.Dispose( disposing );
48        }
49
50        // Windows Form Designer generated code
51
52        /// <summary>
53        /// The main entry point for the application.
54        /// </summary>
55        [STAThread]
56        static void Main()
57        {
58           Application.Run( new FrmAgeGroup() );
59        }
60
61        // determines which age group the entered age lies in
62        private void btnAgeGroup_Click(
63           object sender, System.EventArgs e )
64        {
65
66           int intAge;
```

```
67
68          intAge = Int32.Parse( txtAge.Text );
69
70          if ( intAge < 0 )
71          {
72             txtAge.Text = "Enter a value greater than or" +
73                "equal to zero.";
74          }
75          else if ( intAge < 13 )
76          {
77             txtAge.Text = "Child";
78          }
79          else if ( intAge < 20 )
80          {
81             txtAge.Text = "Teenager";
82          }
83          else if ( intAge < 30 )
84          {
85             txtAge.Text = "Young adult";
86          }
87          else if ( intAge < 65 )
88          {
89             txtAge.Text = "Adult";
90          }
91          else
92          {
93             txtAge.Text = "Senior Citizen";
94          }
95
96       } // end method btnAgeGroup_Click
97
98    } // end class FrmAgeGroup
99 }
```

What's wrong with this code? ▶

7.15 Assume that `lblAMPM` is a `Label` control. Find the error(s) in the following code.

```
1  int intHour;
2
3  intHour = 14;
4
5  if ( intHour < 0 )
6  {
7     lblAMPM.Text = "Time Error.";
8  }
9  else if ( intHour > 23 )
10 {
11    lblAMPM.Text = "Time Error.";
12 }
13 else
14 {
```

```
15       lblAMPM.Text = "PM";
16    }
17    else if ( intHour < 12 )
18    {
19       lblAMPM.Text = "AM";
20    }
```

Answer: An `else if` block (line 17) cannot follow an `else` block (line 13). An `else` block must be preceded by an `if` or `else if` block. The complete incorrect code reads:

```
1  // Exercise 7.15 Solution
2  // AMorPM.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace AMorPM
12 {
13    /// <summary>
14    /// Summary description for FrmAMorPM.
15    /// </summary>
16    public class FrmAMorPM : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblAMPM;
19       private System.Windows.Forms.Label lblAMPMText;
20       private System.Windows.Forms.Label lblHourOut;
21       private System.Windows.Forms.Label lblHour;
22       /// <summary>
23       /// Required designer variable.
24       /// </summary>
25       private System.ComponentModel.Container components = null;
26
27       public FrmAMorPM()
28       {
29          //
30          // Required for Windows Form Designer support
31          //
32          InitializeComponent();
33
34       }
35
36       /// <summary>
37       /// Clean up any resources being used.
38       /// </summary>
39       protected override void Dispose( bool disposing )
40       {
41          if( disposing )
42          {
43             if (components != null)
44             {
45                components.Dispose();
46             }
47          }
48          base.Dispose( disposing );
49       }
```

```
50
51          // Windows Form Designer generated code
52
53          /// <summary>
54          /// The main entry point for the application.
55          /// </summary>
56          [STAThread]
57          static void Main()
58          {
59             Application.Run( new FrmAMorPM() );
60          }
61
62          // determines if the specified time lies in the AM or PM
63          private void FrmAMorPM_Load(
64             object sender, System.EventArgs e )
65          {
66             int intHour;
67
68             intHour = 14;
69
70             if ( intHour < 0 )
71             {
72                lblAMPM.Text = "Time Error.";
73             }
74             else if ( intHour > 23 )
75             {
76                lblAMPM.Text = "Time Error.";
77             }
78             else
79             {
80                lblAMPM.Text = "PM";
81             }
82             else if ( intHour < 12 )
83             {
84                lblAMPM.Text = "AM";
85             }
86
87          } // end method FrmAmorPM_Load
88
89       } // end class FrmAMorPM
90    }
```

`else if` block cannot appear after an `else` block

Answer: The complete corrected code should read:

```
1  // Exercise 7.15 Solution
2  // AMorPM.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
```

```
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace AMorPM
{
   /// <summary>
   /// Summary description for FrmAMorPM.
   /// </summary>
   public class FrmAMorPM : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblAMPM;
      private System.Windows.Forms.Label lblAMPMText;
      private System.Windows.Forms.Label lblHourOut;
      private System.Windows.Forms.Label lblHour;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmAMorPM()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmAMorPM() );
      }

      // determines if the specified time lies in the AM or PM
      private void FrmAMorPM_Load(
         object sender, System.EventArgs e )
      {
         int intHour;
```

```
67
68          intHour = 14;
69
70          if ( intHour < 0 )
71          {
72             lblAMPM.Text = "Time Error.";
73          }
74          else if ( intHour > 23 )
75          {
76             lblAMPM.Text = "Time Error.";
77          }
78          else if ( intHour < 12 )
79          {
80             lblAMPM.Text = "AM";
81          }
82          else
83          {
84             lblAMPM.Text = "PM";
85          }
86
87       } // end method FrmAMorPM_Load
88
89    } // end class FrmAMorPM
90 }
```

Using the Debugger ▶

7.16 ***(Grade Calculator Application)*** Copy the directory `C:\Examples\Tutorial07\Exercises\Debugger\Grades` into your `C:\SimplyCSP` directory. This directory contains the `Grades` application, which takes a number from the user and displays the corresponding letter grade. For values in the range 90–100 it should display **A**, for 80–89, **B**, for 70–79, **C**, for 60–69, **D** and for anything lower, **F**. Run the application. Enter the value 85 in the `TextBox` and click **Calculate**. Notice that the application displays **D** when it ought to display **B**. Select **View > Code** to enter the code editor and set as many breakpoints as you feel necessary. Select **Debug > Start** to use the debugger to help you find the error(s). Figure 7.34 shows the incorrect output when the value 85 is input.

Figure 7.34 Incorrect output for **Grade Calculator** application.

Answer:

```
1  // Exercise 7.16 Solution
2  // Grades.cs
3
4  using System;
5  using System.Drawing;
```

```
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Grades
12 {
13    /// <summary>
14    /// Summary description for FrmGrade.
15    /// </summary>
16    public class FrmGrade : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input number grade
19       private System.Windows.Forms.Label lblInput;
20       private System.Windows.Forms.TextBox txtGrade;
21
22       // Labels to display letter grade
23       private System.Windows.Forms.Label lblOutput;
24       private System.Windows.Forms.Label lblDisplay;
25
26       // Button to calculate letter grade from number grade
27       private System.Windows.Forms.Button btnCalculate;
28
29       /// <summary>
30       /// Required designer variable.
31       /// </summary>
32       private System.ComponentModel.Container components = null;
33
34       public FrmGrade()
35       {
36          //
37          // Required for Windows Form Designer support
38          //
39          InitializeComponent();
40
41       }
42
43       /// <summary>
44       /// Clean up any resources being used.
45       /// </summary>
46       protected override void Dispose( bool disposing )
47       {
48          if( disposing )
49          {
50             if (components != null)
51             {
52                components.Dispose();
53             }
54          }
55          base.Dispose( disposing );
56       }
57
58       // Windows Form Designer generated code
59
60       /// <summary>
61       /// The main entry point for the application.
62       /// </summary>
63       [STAThread]
64       static void Main()
65       {
```

Individual `if` statements replaced with nested `if...else` statements

```
66             Application.Run( new FrmGrade() );
67          }
68
69          // handles Click event
70          private void btnCalculate_Click(
71             object sender, System.EventArgs e )
72          {
73             int intGrade;
74
75             intGrade = Int32.Parse( txtGrade.Text );
76
77             // display letter grade corresponding to number
78             if ( intGrade >= 90 )
79             {
80                lblDisplay.Text = "A";
81             }
82             else if ( intGrade >= 80 )
83             {
84                lblDisplay.Text = "B";
85             }
86             else if ( intGrade >= 70 )
87             {
88                lblDisplay.Text = "C";
89             }
90             else if ( intGrade >= 60 )
91             {
92                lblDisplay.Text = "D";
93             }
94             else
95             {
96                lblDisplay.Text = "F";
97             }
98
99          } // end method btnCalculate_Click
100
101       } // end class FrmGrade
102    }
```

Programming Challenge ▶

7.17 ***(Encryption Application)*** A company transmits data over the telephone, but it is concerned that its phones could be tapped. All its data is transmitted as four-digit `int`s. The company has asked you to write an application that encrypts its data so that it may be transmitted more securely. Encryption is the process of transforming data for security reasons. Create a `Form` similar to Fig. 7.35. Your application should read four digits entered by the user and encrypt the information as follows:

a) Replace each digit by *(the sum of that digit plus 7) modulo 10*. We use the term modulo to indicate you are to use the remainder (%) operator.

b) Swap the first digit with the third, and swap the second digit with the fourth.

Figure 7.35 **Encryption** application.

Answer:

```
1  // Exercise 7.17 Solution
2  // Encryption.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Encryption
12 {
13    /// <summary>
14    /// Summary description for FrmEncryption.
15    /// </summary>
16    public class FrmEncryption : System.Windows.Forms.Form
17    {
18       // Labels and TextBoxes to input the four numbers
19       // to be encrypted
20       private System.Windows.Forms.Label lblNumber1;
21       private System.Windows.Forms.TextBox txtNumber1;
22       private System.Windows.Forms.Label lblNumber2;
23       private System.Windows.Forms.TextBox txtNumber2;
24       private System.Windows.Forms.Label lblNumber3;
25       private System.Windows.Forms.TextBox txtNumber3;
26       private System.Windows.Forms.Label lblNumber4;
27       private System.Windows.Forms.TextBox txtNumber4;
28
29       // Button to perform the encryption
30       private System.Windows.Forms.Button btnEncyrpt;
31
32       // Labels to display the encrypted numbers
33       private System.Windows.Forms.Label lblEncrypted;
34       private System.Windows.Forms.Label lblEncryptedNumber1;
35       private System.Windows.Forms.Label lblEncryptedNumber2;
36       private System.Windows.Forms.Label lblEncryptedNumber3;
37       private System.Windows.Forms.Label lblEncryptedNumber4;
38
39       /// <summary>
40       /// Required designer variable.
41       /// </summary>
42       private System.ComponentModel.Container components = null;
43
44       public FrmEncryption()
45       {
46          //
47          // Required for Windows Form Designer support
48          //
49          InitializeComponent();
50
51       }
52
53       /// <summary>
54       /// Clean up any resources being used.
55       /// </summary>
56       protected override void Dispose( bool disposing )
57       {
58          if( disposing )
```

```
59          {
60             if (components != null)
61             {
62                components.Dispose();
63             }
64          }
65          base.Dispose( disposing );
66       }
67
68       // Windows Form Designer generated code
69
70       /// <summary>
71       /// The main entry point for the application.
72       /// </summary>
73       [STAThread]
74       static void Main()
75       {
76          Application.Run( new FrmEncryption() );
77       }
78
79       // handles Click event
80       private void btnEncyrpt_Click(
81          object sender, System.EventArgs e )
82       {
83          // clear previous output
84          lblEncryptedNumber1.Text = "";
85          lblEncryptedNumber2.Text = "";
86          lblEncryptedNumber3.Text = "";
87          lblEncryptedNumber4.Text = "";
88
89          int intNumber1;
90          int intNumber2;
91          int intNumber3;
92          int intNumber4;
93
94          // retrieve numbers from TextBoxes
95          intNumber1 = Int32.Parse( txtNumber1.Text );
96          intNumber2 = Int32.Parse( txtNumber2.Text );
97          intNumber3 = Int32.Parse( txtNumber3.Text );
98          intNumber4 = Int32.Parse( txtNumber4.Text );
99
100         // convert to 1-digit numbers
101         if ( intNumber1 > 9 )
102         {
103            intNumber1 %= 10;
104         }
105         else if ( intNumber2 > 9 )
106         {
107            intNumber2 %= 10;
108         }
109         else if ( intNumber3 > 9 )
110         {
111            intNumber3 %= 10;
112         }
113         else if ( intNumber4 > 9 )
114         {
115            intNumber4 %= 10;
116         }
117
118         // display using the following:
```

```
119          // 1st number and third number are swapped
120          // 2nd number and 4th number are swapped
121          lblEncryptedNumber1.Text =
122             Convert.ToString( ( intNumber3 + 7 ) % 10 );
123          lblEncryptedNumber2.Text =
124             Convert.ToString( ( intNumber4 + 7 ) % 10 );
125          lblEncryptedNumber3.Text =
126             Convert.ToString( ( intNumber1 + 7 ) % 10 );
127          lblEncryptedNumber4.Text =
128             Convert.ToString( ( intNumber2 + 7 ) % 10 );
129
130       } // end method btnEncrypt_Click
131
132    } // end class FrmEncryption
133 }
```

8

TUTORIAL

Dental Payment Application

Introducing CheckBoxes and Message Dialogs

Solutions

Instructor's Manual Exercise Solutions Tutorial 8

MULTIPLE-CHOICE QUESTIONS

8.1 How many `CheckBox`es in a GUI can be selected at once?

a) `0` b) `1`
c) `4` d) any number

8.2 The text that appears alongside a `CheckBox` is referred to as the _____.

a) `CheckBox` label b) `CheckBox` name
c) `CheckBox` value d) `CheckBox` data

8.3 The first argument passed to the `MessageBox.Show` method is _____.

a) the text displayed in the dialog's title bar
b) a constant representing the `Button`s displayed in the dialog
c) the text displayed inside the dialog
d) a constant representing the icon that appears in the dialog

8.4 You can specify the `Button`(s) and icon to be displayed in a message dialog by using the `MessageBoxButtons` and _____ constants.

a) `MessageIcon` b) `MessageBoxImages`
c) `MessageBoxPicture` d) `MessageBoxIcon`

8.5 _____ are used to create complex conditions.

a) Assignment operators b) Activity diagrams
c) Logical operators d) Formatting codes

8.6 The `&&` operator _____.

a) performs short-circuit evaluation
b) is a keyword
c) is a comparison operator
d) evaluates to `false` if both operands are `true`

8.7 A `CheckBox` is selected when its `Checked` property is set to _____.

a) `on` b) `true`
c) `selected` d) `checked`

8.8 The condition *expression1* `&&` *expression2* evaluates to `true` when _____.

a) *expression1* is `true` and *expression2* is `false`
b) *expression1* is `false` and *expression2* is `true`
c) both *expression1* and *expression2* are `true`
d) both *expression1* and *expression2* are `false`

8.9 The condition *expression1* `||` *expression2* evaluates to `false` when _____.

a) *expression1* is `true` and *expression2* is `false`
b) *expression1* is `false` and *expression2* is `true`
c) both *expression1* and *expression2* are `true`
d) both *expression1* and *expression2* are `false`

8.10 The condition *expression1* `^` *expression2* evaluates to `true` when _____.

a) *expression1* is `true` and *expression2* is `false`
b) *expression1* is `false` and *expression2* is `true`
c) both *expression1* and *expression2* are `true`
d) Both a and b.

Answers: 8.1) d. 8.2) a. 8.3) c. 8.4) d. 8.5) c. 8.6) a. 8.7) b. 8.8) c. 8.9) d. 8.10) d.

EXERCISES

8.11 (***Enhanced Dental Payment Application***) Modify the **Dental Payment** application from this tutorial to include additional services, as shown in Fig. 8.21. Add the proper functionality (using `if` statements) to determine whether any of the new `CheckBox`es are selected and, if so, add the price of the service to the total bill.

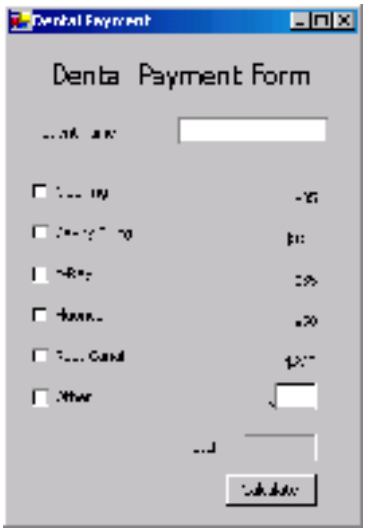


Figure 8.21 Enhanced **Dental Payment** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial08\Exercises\DentalPaymentEnhanced` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `DentalPayment.sln` in the `DentalPaymentEnhanced` directory to open the application.

c) ***Adding CheckBoxes and Labels and a TextBox.*** Add two `CheckBox`es and two `Label`s to the `Form`. The new `CheckBox`es should be labelled **Fluoride** and **Root Canal**, respectively. Add these `CheckBox`es and `Label`s beneath the `X-Ray` `CheckBox` and its price `Label`. The price for a Fluoride treatment is $50; the price for a root canal is $225. Add a `CheckBox` labelled **Other** and a `Label` containing a dollar sign (**$**) to the `Form`, as shown in Fig. 8.21. Then add a `TextBox` to the right of the **$** `Label` in which the user can enter the cost of the service performed. Rearrange and comment the new control declarations appropriately.

d) ***Modifying the `Click` event handler code.*** Add code to the `btnCalculate_Click` event handler to determine whether the new `CheckBox`es have been selected. This can be done using `if` statements that are similar to the ones already in the event handler. Use the `if` statements to update the bill amount.

e) ***Running the application.*** Select **Debug > Start** to run your application. Test your application by checking one or more of the new services. Click the **Calculate** `Button` and verify that the proper total is displayed. Test the application again by checking some of the services, then checking the **Other** `CheckBox` and entering a dollar value for this service. Click the **Calculate** `Button` and verify that the proper total is displayed, and that it includes the price for the "other" service.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 8.11 Solution
2  // DentalPayment.cs (Enhanced)
3
```

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace DentalPayment
{
   /// <summary>
   /// Summary description for FrmDentalPayment.
   /// </summary>
   public class FrmDentalPayment : System.Windows.Forms.Form
   {
      // Label displaying title
      private System.Windows.Forms.Label lblTitle;

      // Label and TextBox for patient name
      private System.Windows.Forms.Label lblName;
      private System.Windows.Forms.TextBox txtName;

      // CheckBox and Label for a cleaning
      private System.Windows.Forms.CheckBox chkClean;
      private System.Windows.Forms.Label lblCleanCost;

      // CheckBox and Label for a filling
      private System.Windows.Forms.CheckBox chkCavity;
      private System.Windows.Forms.Label lblFillingCost;

      // CheckBox and Label for an X-Ray
      private System.Windows.Forms.CheckBox chkXRay;
      private System.Windows.Forms.Label lblXRayCost;

      // CheckBox and Label for a fluoride treatment
      private System.Windows.Forms.CheckBox chkFluoride;
      private System.Windows.Forms.Label lblFluorideCost;

      // CheckBox and Label for a root canal
      private System.Windows.Forms.CheckBox chkRootCanal;
      private System.Windows.Forms.Label lblRootCanalCost;

      // CheckBox, Label and TextBox for a user-defined
      // "other" cost
      private System.Windows.Forms.CheckBox chkOther;
      private System.Windows.Forms.Label lblOther;
      private System.Windows.Forms.TextBox txtOtherCost;

      // Labels to display total cost
      private System.Windows.Forms.Label lblTotal;
      private System.Windows.Forms.Label lblTotalResult;

      // Button to calculate total cost
      private System.Windows.Forms.Button btnCalculate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmDentalPayment()
```

```
64          {
65             //
66             // Required for Windows Form Designer support
67             //
68             InitializeComponent();
69
70             //
71             // TODO: Add any constructor code after InitializeComponent
72             // call
73             //
74          }
75
76          /// <summary>
77          /// Clean up any resources being used.
78          /// </summary>
79          protected override void Dispose( bool disposing )
80          {
81             if( disposing )
82             {
83                if (components != null)
84                {
85                   components.Dispose();
86                }
87             }
88             base.Dispose( disposing );
89          }
90
91          // Windows Form Designer generated code
92
93          /// <summary>
94          /// The main entry point for the application.
95          /// </summary>
96          [STAThread]
97          static void Main()
98          {
99             Application.Run( new FrmDentalPayment() );
100         }
101
102         // handles Calculate Button's Click event
103         private void btnCalculate_Click(
104            object sender, System.EventArgs e )
105         {
106            // clear text displayed in Label
107            lblTotalResult.Text = "";
108
109            // if no name entered and no CheckBox checked,
110            // display message
111            if ( txtName.Text == "" ||
112               ( chkClean.Checked == false &&
113               chkXRay.Checked == false &&
114               chkCavity.Checked == false &&
115               chkFluoride.Checked == false &&
116               chkRootCanal.Checked == false &&
117               chkOther.Checked == false ) )
118            {
119               // display message in dialog
120               MessageBox.Show(
121                  "Please enter a name and check at least one item",
122                  "Missing Information", MessageBoxButtons.OK,
123                  MessageBoxIcon.Exclamation );
```

```
124          }
125          else // add prices
126          {
127             // intTotal contains amount to bill patient
128             int intTotal = 0;
129
130             // if patient had a cleaning
131             if ( chkClean.Checked == true)
132             {
133                intTotal += 35;
134             }
135
136             // if patient had cavity filled
137             if ( chkCavity.Checked == true )
138             {
139                intTotal += 150;
140             }
141
142             // if patient had X-Ray taken
143             if ( chkXRay.Checked == true )
144             {
145                intTotal += 85;
146             }
147
148             // if patient had fluoride treatment
149             if ( chkFluoride.Checked == true )
150             {
151                intTotal += 50;
152             }
153
154             // if patient had a root canal
155             if ( chkRootCanal.Checked == true )
156             {
157                intTotal += 225;
158             }
159
160             // if patient had some other service performed
161             if ( chkOther.Checked == true )
162             {
163                if ( txtOtherCost.Text == "" )
164                {
165                   MessageBox.Show( "Please enter cost of service",
166                      "No Cost Entered", MessageBoxButtons.OK,
167                      MessageBoxIcon.Exclamation );
168                }
169                else
170                {
171                   // add cost entered
172                   intTotal += Int32.Parse( txtOtherCost.Text );
173                }
174             }
175
176             // display total
177             lblTotalResult.Text =
178                String.Format( "{0:C}", intTotal );
179
180          } // end else
181
182       } // end method btnCalculate_Click
183
```

```
184    } // end class FrmDentalPayment
185 }
```

8.12 (*Fuzzy Dice Order Form Application*) Write an application that allows users to process orders for fuzzy dice, as shown in Fig. 8.22. The application should calculate the total price of the order, including tax and shipping. `TextBoxes` for inputting the order number, the customer name and the shipping address are provided. Initially, these fields contain text that describes their purpose. Provide `CheckBoxes` for selecting the fuzzy-dice color and `TextBoxes` for inputting the quantities of fuzzy dice to order. The application should also contain a `Button` that, when clicked, calculates the subtotals for each type of fuzzy dice ordered and the total of the entire order (including tax and shipping). Use 5% for the tax rate. Shipping charges are $1.50 for up to 20 pairs of dice. If more than 20 pairs of dice are ordered, shipping is free. For the total to be calculated, the user must enter an order number, a name and a shipping address. If they have not done so, a message should be displayed in a dialog.

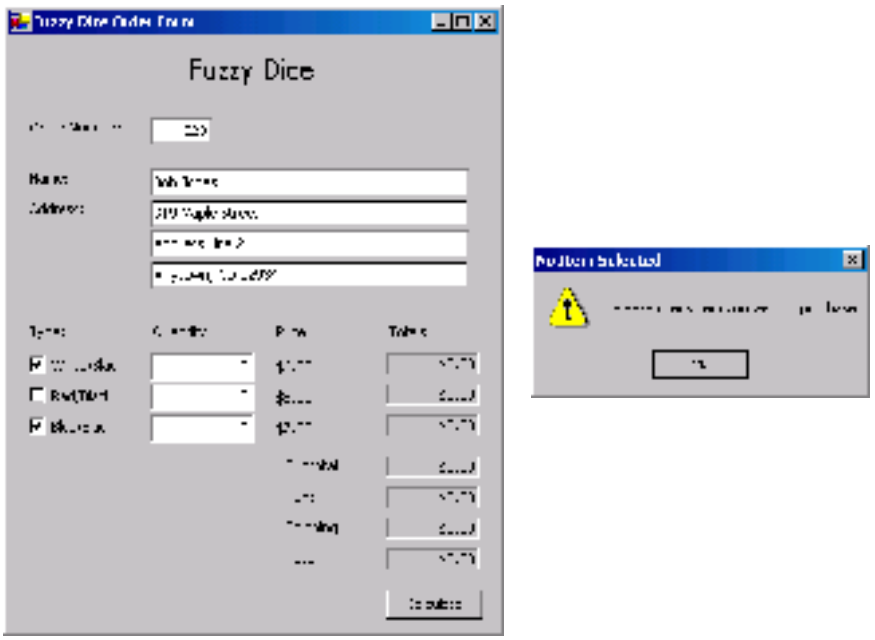


Figure 8.22 **Fuzzy Dice Order Form** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial08\Exercises\FuzzyDiceOrderForm` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `FuzzyDiceOrderForm.sln` in the `FuzzyDiceOrderForm` directory to open the application.

c) ***Adding CheckBoxes to the Form.*** Add three `CheckBoxes` to the `Form`. Label the first `CheckBox` **White/Black**, the second one **Red/Black** and the third **Blue/Black**. Rearrange and comment the new control declarations appropriately.

d) ***Adding a `Click` event handler and its code.*** Create the `Click` event handler for the **Calculate** `Button`. For this application, users should not be allowed to specify an item's quantity unless the item's corresponding `CheckBox` is checked. For the total to be calculated, the user must enter an order number, a name and a shipping address. Use logical operators to ensure that these terms are met. If they are not, display a message in a dialog.

e) ***Calculating the total cost.*** Calculate the subtotal, tax, shipping and total, and display the results in their corresponding `Labels`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Test the application by providing quantities for checked items. For instance, ensure that your application is calculating 5% sales tax. If more than 20 pairs of dice are ordered, verify that shipping is free. Also, determine whether your code containing the logical operators works correctly by specifying a quantity for an item that is not checked. For

instance, in Fig. 8.22, a quantity is specified for **Red/Black** dice, but the corresponding `CheckBox` is not selected. This should cause the message dialog in Fig. 8.22 to appear.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 8.12 Solution
// FuzzyDiceOrderForm.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace FuzzyDiceOrderForm
{
   /// <summary>
   /// Summary description for FrmFuzzyDiceOrderForm.
   /// </summary>
   public class FrmFuzzyDiceOrderForm : System.Windows.Forms.Form
   {
      // Label to display the Form's title
      private System.Windows.Forms.Label lblTitle;

      // Label and TextBox to input order number
      private System.Windows.Forms.Label lblOrderNumber;
      private System.Windows.Forms.TextBox txtOrderNumber;

      // Label and TextBox to input name
      private System.Windows.Forms.Label lblCompanyName;
      private System.Windows.Forms.TextBox txtName;

      // Label and TextBoxes to input address city, state and zip
      private System.Windows.Forms.Label lblAddress;
      private System.Windows.Forms.TextBox txtAddressLine1;
      private System.Windows.Forms.TextBox txtAddressLine2;
      private System.Windows.Forms.TextBox txtCityStateZip;

      // Labels that serve as column headings for type, quantity,
      // price and total cost
      private System.Windows.Forms.Label lblType;
      private System.Windows.Forms.Label lblQuantity;
      private System.Windows.Forms.Label lblPrice;
      private System.Windows.Forms.Label lblTotals;

      // CheckBox, TextBox and Labels for white and black fuzzy dice
      private System.Windows.Forms.CheckBox chkWhiteBlack;
      private System.Windows.Forms.TextBox txtWhiteBlackQuantity;
      private System.Windows.Forms.Label lblWhiteBlackPrice;
      private System.Windows.Forms.Label lblWhiteBlackTotals;

      // CheckBox, TextBox and Labels for red and black fuzzy dice
      private System.Windows.Forms.CheckBox chkRedBlack;
      private System.Windows.Forms.TextBox txtRedBlackQuantity;
      private System.Windows.Forms.Label lblRedBlackPrice;
      private System.Windows.Forms.Label lblRedBlackTotals;

```

```
54        // CheckBox, TextBox and Labels for blue and black fuzzy dice
55        private System.Windows.Forms.CheckBox chkBlueBlack;
56        private System.Windows.Forms.TextBox txtBlueBlackQuantity;
57        private System.Windows.Forms.Label lblBlueBlackPrice;
58        private System.Windows.Forms.Label lblBlueBlackTotals;
59
60        // Labels to display subtotal
61        private System.Windows.Forms.Label lblSubtotal;
62        private System.Windows.Forms.Label lblSubtotalResult;
63
64        // Labels to display tax
65        private System.Windows.Forms.Label lblTax;
66        private System.Windows.Forms.Label lblTaxResult;
67
68        // Labels to display shipping cost
69        private System.Windows.Forms.Label lblShipping;
70        private System.Windows.Forms.Label lblShippingResult;
71
72        // Labels to display total cost after tax and shipping
73        private System.Windows.Forms.Label lblTotal;
74        private System.Windows.Forms.Label lblTotalResult;
75
76        // Button to calculate costs
77        private System.Windows.Forms.Button btnCalculate;
78
79        /// <summary>
80        /// Required designer variable.
81        /// </summary>
82        private System.ComponentModel.Container components = null;
83
84        public FrmFuzzyDiceOrderForm()
85        {
86           //
87           // Required for Windows Form Designer support
88           //
89           InitializeComponent();
90
91           //
92           // TODO: Add any constructor code after InitializeComponent
93           // call
94           //
95        }
96
97        /// <summary>
98        /// Clean up any resources being used.
99        /// </summary>
100       protected override void Dispose( bool disposing )
101       {
102          if( disposing )
103          {
104             if (components != null)
105             {
106                components.Dispose();
107             }
108          }
109          base.Dispose( disposing );
110       }
111
112       // Windows Form Designer generated code
113
```

```
114         /// <summary>
115         /// The main entry point for the application.
116         /// </summary>
117         [STAThread]
118         static void Main()
119         {
120            Application.Run( new FrmFuzzyDiceOrderForm() );
121         }
122
123         // check validity of order before calculating totals
124         private void btnCalculate_Click(
125            object sender, System.EventArgs e )
126         {
127            // display message if user does not check box
128            if ( ( Int32.Parse( txtWhiteBlackQuantity.Text ) > 0 &&
129               chkWhiteBlack.Checked == false ) ||
130               ( Int32.Parse( txtRedBlackQuantity.Text ) > 0 &&
131               chkRedBlack.Checked == false ) ||
132               ( Int32.Parse( txtBlueBlackQuantity.Text ) > 0 &&
133               chkBlueBlack.Checked == false ) )
134            {
135               // display message in dialog
136               MessageBox.Show(
137                  "Please check item you wish to purchase",
138                  "No Item Selected", MessageBoxButtons.OK,
139                  MessageBoxIcon.Exclamation );
140            }
141            // display message if order number, name or address fields
142            // are empty or contain default values
143            else if ( txtOrderNumber.Text == ""
144               || txtOrderNumber.Text == "0" || txtName.Text == ""
145               || txtName.Text == "Enter a name here"
146               || txtAddressLine1.Text == ""
147               || txtAddressLine1.Text == "Address Line 1"
148               || txtCityStateZip.Text == ""
149               || txtCityStateZip.Text == "City, State, zip" )
150            {
151               // display message in dialog
152               MessageBox.Show(
153                  "Please fill out all information fields",
154                  "Empty Fields", MessageBoxButtons.OK,
155                  MessageBoxIcon.Exclamation );
156            }
157            else // calculate totals
158            {
159               // individual totals
160               // total of white/black dice ordered
161               decimal decWhiteBlackTotals =
162                  Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
163
164               // total of red/black dice ordered
165               decimal decRedBlackTotals =
166                  Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
167
168               // total of blue/black dice ordered
169               decimal decBlueBlackTotals =
170                  Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
171
172               // display totals of dice ordered
173               lblWhiteBlackTotals.Text =
```

```
            String.Format( "{0:C}", decWhiteBlackTotals );
         lblRedBlackTotals.Text =
            String.Format( "{0:C}", decRedBlackTotals );
         lblBlueBlackTotals.Text =
            String.Format( "{0:C}", decBlueBlackTotals );

         // calculate and display subtotal
         decimal decSubtotal = decWhiteBlackTotals +
            decRedBlackTotals + decBlueBlackTotals;

         lblSubtotalResult.Text =
            String.Format( "{0:C}", decSubtotal );

         // calculate and display tax
         decimal decTax = decSubtotal * 0.05M;

         lblTaxResult.Text = String.Format( "{0:C}", decTax );

         // shipping
         // $1.50 for up to 20 items
         // free after 20 items
         int intNumberOfItems =
            ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
            Int32.Parse( txtRedBlackQuantity.Text ) +
            Int32.Parse( txtBlueBlackQuantity.Text ) );

         decimal decShippingCost = 0.0M;

         // shipping is $1.50 if less than 20 pairs ordered
         if ( intNumberOfItems <= 20 )
         {
            decShippingCost = 1.5M;
         }

         // display shipping cost
         lblShippingResult.Text =
            String.Format( "{0:C}", decShippingCost );

         // calculate and display total
         decimal decTotalCharge = decSubtotal +
            decTax + decShippingCost;

         lblTotalResult.Text =
            String.Format( "{0:C}", decTotalCharge );

      } // end else

   } // end method btnCalculate_Click

} // end class FrmFuzzyDiceOrderForm
}
```

8.13 (***Modified Fuzzy Dice Order Form Application***) Modify the **Fuzzy Dice Order Form** application from Exercise 8.12 to determine whether customers should receive a 7% discount off their purchase. Customers ordering more than $500 (before tax and shipping) in fuzzy dice are eligible for this discount (Fig. 8.23).

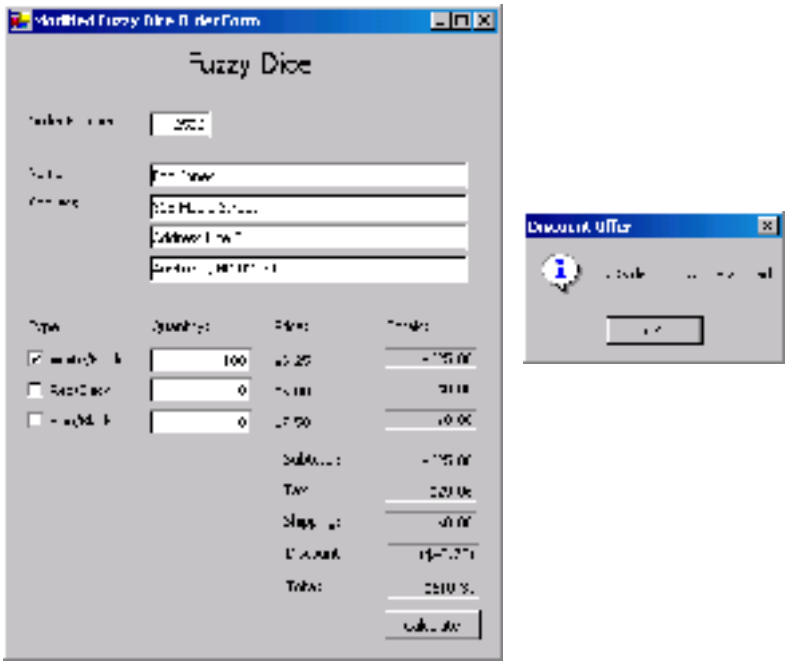


Figure 8.23 Modified **Fuzzy Dice Order Form** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial08\Exercises\FuzzyDiceOrderFormModified` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `FuzzyDiceOrderForm.sln` in the `FuzzyDiceOrderFormModified` directory to open the application.

c) ***Determining whether the total cost is over $500***. Use an `if` statement to determine if the amount ordered is greater than $500.

d) ***Displaying the discount and subtracting the discount from the total***. If a customer orders more than $500, display a message dialog as shown in Fig. 8.23 that informs the user that the customer is entitled to a 7% discount. The message dialog should contain an `Information` icon and an **OK** `Button`. Calculate 7% of the total amount, and display the discount amount in the **Discount:** field. Subtract this amount from the total, and update the **Total:** field.

e) ***Running the application.*** Select **Debug > Start** to run your application. Test your application to ensure that it runs correctly and calculates and displays the discount properly, as shown in Fig. 8.23.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 8.13 Solution
2  // FuzzyDiceOrderForm.cs (Modified)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace FuzzyDiceOrderForm
12 {
13    /// <summary>
```

```
14    /// Summary description for FrmFuzzyDiceOrderForm.
15    /// </summary>
16    public class FrmFuzzyDiceOrderForm : System.Windows.Forms.Form
17    {
18       // Label to display the Form's title
19       private System.Windows.Forms.Label lblTitle;
20
21       // Label and TextBox to input order number
22       private System.Windows.Forms.Label lblOrderNumber;
23       private System.Windows.Forms.TextBox txtOrderNumber;
24
25       // Label and TextBox to input name
26       private System.Windows.Forms.Label lblCompanyName;
27       private System.Windows.Forms.TextBox txtName;
28
29       // Label and TextBoxes to input address, city, state and zip
30       private System.Windows.Forms.Label lblAddress;
31       private System.Windows.Forms.TextBox txtAddressLine1;
32       private System.Windows.Forms.TextBox txtAddressLine2;
33       private System.Windows.Forms.TextBox txtCityStateZip;
34
35       // Labels that serve as column headings for type, quantity,
36       // price and total cost
37       private System.Windows.Forms.Label lblType;
38       private System.Windows.Forms.Label lblQuantity;
39       private System.Windows.Forms.Label lblPrice;
40       private System.Windows.Forms.Label lblTotals;
41
42       // CheckBox, TextBox and Labels for white and black fuzzy dice
43       private System.Windows.Forms.CheckBox chkWhiteBlack;
44       private System.Windows.Forms.TextBox txtWhiteBlackQuantity;
45       private System.Windows.Forms.Label lblWhiteBlackPrice;
46       private System.Windows.Forms.Label lblWhiteBlackTotals;
47
48       // CheckBox, TextBox and Labels for red and black fuzzy dice
49       private System.Windows.Forms.CheckBox chkRedBlack;
50       private System.Windows.Forms.TextBox txtRedBlackQuantity;
51       private System.Windows.Forms.Label lblRedBlackPrice;
52       private System.Windows.Forms.Label lblRedBlackTotals;
53
54       // CheckBox, TextBox and Labels for blue and black fuzzy dice
55       private System.Windows.Forms.CheckBox chkBlueBlack;
56       private System.Windows.Forms.TextBox txtBlueBlackQuantity;
57       private System.Windows.Forms.Label lblBlueBlackPrice;
58       private System.Windows.Forms.Label lblBlueBlackTotals;
59
60       // Labels to display subtotal
61       private System.Windows.Forms.Label lblSubtotal;
62       private System.Windows.Forms.Label lblSubtotalResult;
63
64       // Labels to display tax
65       private System.Windows.Forms.Label lblTax;
66       private System.Windows.Forms.Label lblTaxResult;
67
68       // Labels to display shipping cost
69       private System.Windows.Forms.Label lblShipping;
70       private System.Windows.Forms.Label lblShippingResult;
71
72       // Labels to display discount
73       private System.Windows.Forms.Label lblDiscountLabel;
```

```
74          private System.Windows.Forms.Label lblDiscount;
75
76          // Labels to display total cost after tax and shipping
77          private System.Windows.Forms.Label lblTotal;
78          private System.Windows.Forms.Label lblTotalResult;
79
80          // Button to calculate costs
81          private System.Windows.Forms.Button btnCalculate;
82
83          /// <summary>
84          /// Required designer variable.
85          /// </summary>
86          private System.ComponentModel.Container components = null;
87
88          public FrmFuzzyDiceOrderForm()
89          {
90             //
91             // Required for Windows Form Designer support
92             //
93             InitializeComponent();
94
95             //
96             // TODO: Add any constructor code after InitializeComponent
97             // call
98             //
99          }
100
101         /// <summary>
102         /// Clean up any resources being used.
103         /// </summary>
104         protected override void Dispose( bool disposing )
105         {
106            if( disposing )
107            {
108               if (components != null)
109               {
110                  components.Dispose();
111               }
112            }
113            base.Dispose( disposing );
114         }
115
116         // Windows Form Designer generated code
117
118         /// <summary>
119         /// The main entry point for the application.
120         /// </summary>
121         [STAThread]
122         static void Main()
123         {
124            Application.Run( new FrmFuzzyDiceOrderForm() );
125         }
126
127         // check validity of order before calculating totals
128         private void btnCalculate_Click(
129            object sender, System.EventArgs e )
130         {
131            // display message if user does not check box
132            if ( ( Int32.Parse( txtWhiteBlackQuantity.Text ) > 0 &&
133               chkWhiteBlack.Checked == false ) ||
```

```
134            ( Int32.Parse( txtRedBlackQuantity.Text) > 0 &&
135            chkRedBlack.Checked == false ) ||
136            ( Int32.Parse( txtBlueBlackQuantity.Text) > 0 &&
137            chkBlueBlack.Checked == false ) )
138         {
139            // display message in dialog
140            MessageBox.Show(
141               "Please check item you wish to purchase",
142               "No Item Selected", MessageBoxButtons.OK,
143               MessageBoxIcon.Exclamation );
144         }
145         // display message if order number, name or address fields
146         // are empty or contain default values
147         else if ( txtOrderNumber.Text == ""
148            || txtOrderNumber.Text == "0" || txtName.Text == ""
149            || txtName.Text == "Enter a name here"
150            || txtAddressLine1.Text == ""
151            || txtAddressLine1.Text == "Address Line 1"
152            || txtCityStateZip.Text == ""
153            || txtCityStateZip.Text == "City, State, zip" )
154         {
155            // display message in dialog
156            MessageBox.Show(
157               "Please fill out all information fields",
158               "Empty Fields", MessageBoxButtons.OK,
159               MessageBoxIcon.Exclamation );
160         }
161         else // calculate totals
162         {
163            // individual totals
164            // total of white/black dice ordered
165            decimal decWhiteBlackTotals =
166               Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
167
168            // total of red/black dice ordered
169            decimal decRedBlackTotals =
170               Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
171
172            // total of blue/black dice ordered
173            decimal decBlueBlackTotals =
174               Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
175
176            // display totals of dice ordered
177            lblWhiteBlackTotals.Text =
178               String.Format( "{0:C}", decWhiteBlackTotals );
179            lblRedBlackTotals.Text =
180               String.Format( "{0:C}", decRedBlackTotals );
181            lblBlueBlackTotals.Text =
182               String.Format( "{0:C}", decBlueBlackTotals );
183
184            // calculate and display subtotal
185            decimal decSubtotal = decWhiteBlackTotals +
186               decRedBlackTotals + decBlueBlackTotals;
187
188            lblSubtotalResult.Text =
189               String.Format( "{0:C}", decSubtotal );
190
191            // if decTotalCharge is greater than $500
192            // display message box and give 7% discount
193            if ( decSubtotal > 500 )
```

```
194             {
195                MessageBox.Show(
196                   "7% discount will be applied", "Discount Offer",
197                   MessageBoxButtons.OK, MessageBoxIcon.Information );
198
199                // calculate and display new decTotalCharge
200                // with discount
201                decimal decDiscount = decSubtotal * 0.07M;
202
203                decSubtotal -= decDiscount;
204
205                // decDiscount is negative to reflect that it is
206                // being subtracted from the subtotal during display
207                lblDiscount.Text =
208                   String.Format( "{0:C}", -decDiscount );
209             }
210
211             // calculate and display tax
212             decimal decTax = decSubtotal * 0.05M;
213
214             lblTaxResult.Text = String.Format( "{0:C}", decTax );
215
216             // shipping
217             // $1.50 for up to 20 items
218             // free after 20 items
219             int intNumberOfItems =
220                ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
221                Int32.Parse( txtRedBlackQuantity.Text ) +
222                Int32.Parse( txtBlueBlackQuantity.Text ) );
223
224             decimal decShippingCost = 0.0M;
225
226             // shipping is $1.50 if less than 20 pairs ordered
227             if ( intNumberOfItems <= 20 )
228             {
229                decShippingCost = 1.5M;
230             }
231
232             // display shipping cost
233             lblShippingResult.Text =
234                String.Format( "{0:C}", decShippingCost );
235
236             // calculate and display total
237             decimal decTotalCharge = decSubtotal +
238                decTax + decShippingCost;
239
240             lblTotalResult.Text =
241                String.Format( "{0:C}", decTotalCharge );
242
243          } // end else
244
245       } // end method btnCalculate_Click
246
247    } // end class FrmFuzzyDiceOrderForm
248 }
```

What does this code do? ▶

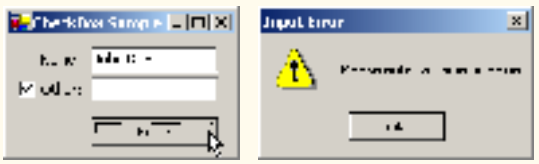
8.14 Assume that `txtName` is a `TextBox` and that `chkOther` is a `CheckBox` next to which is a `TextBox`, `txtOther`, in which the user should specify a value. What does this code segment do?

```
if ( txtName.Text == "" ||
   ( ( chkOther.Checked == true ) &&
   ( txtOther.Text == "" ) ) )
{
   MessageBox.Show( "Please enter a name or value",
      "Input Error", MessageBoxButtons.OK,
      MessageBoxIcon.Exclamation );
}
```

Answer: This code displays a message dialog only if `txtName.Text` is empty or `CheckBox chkOther` is selected and its corresponding `TextBox` is left blank. The complete code reads:

```
// Exercise 8.14 Solution
// CheckBoxSample.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CheckBoxSample
{
   /// <summary>
   /// Summary description for FrmCheckBoxSample.
   /// </summary>
   public class FrmCheckBoxSample : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Button btnEnter;
      private System.Windows.Forms.CheckBox chkOther;
      private System.Windows.Forms.Label lblName;
      private System.Windows.Forms.TextBox txtOther;
      private System.Windows.Forms.TextBox txtName;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCheckBoxSample()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
```

```
48              if (components != null)
49              {
50                 components.Dispose();
51              }
52           }
53           base.Dispose( disposing );
54        }
55
56        // Windows Form Designer generated code
57
58        /// <summary>
59        /// The main entry point for the application.
60        /// </summary>
61        [STAThread]
62        static void Main()
63        {
64           Application.Run( new FrmCheckBoxSample() );
65        }
66
67        // checks to make sure the user entered "other" text
68        private void btnEnter_Click(
69           object sender, System.EventArgs e )
70        {
71           if ( txtName.Text == "" ||
72              ( ( chkOther.Checked == true ) &&
73              ( txtOther.Text == "" ) ) )
74           {
75              MessageBox.Show( "Please enter a name or value",
76                 "Input Error", MessageBoxButtons.OK,
77                 MessageBoxIcon.Exclamation );
78           }
79
80        } // end method btnEnter_Click
81
82     } // end class FrmCheckBoxSample
83  }
```



What's wrong with this code?

8.15 Assume that `txtName` is a `TextBox`. Find the error(s) in the following code:

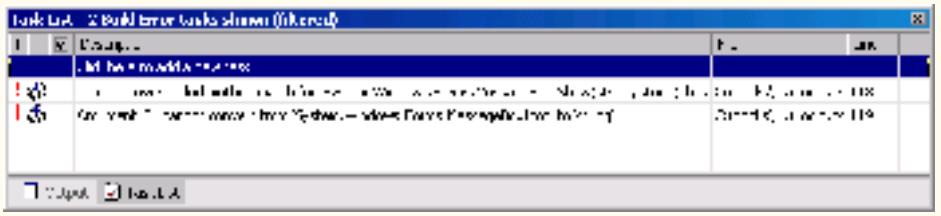
```
1  if ( txtName.Text == "John Doe" )
2  {
3     MessageBox.Show( "Welcome, John!",
4        MessageBoxIcon.Exclamation );
5  }
```

Answer: The call to method `MessageBox.Show` is missing arguments. Also, the nature of the message indicates that `MessageBoxIcon.Information` should be used instead of `MessageBoxIcon.Exclamation`. The complete incorrect code reads:

```
1  // Exercise 8.15 Solution
2  // Login.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Login
12 {
13    /// <summary>
14    /// Summary description for FrmLogin.
15    /// </summary>
16    public class FrmLogin : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Button btnLogin;
19       private System.Windows.Forms.TextBox txtName;
20       private System.Windows.Forms.Label lblUserName;
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.Container components = null;
25
26       public FrmLogin()
27       {
28          //
29          // Required for Windows Form Designer support
30          //
31          InitializeComponent();
32
33          //
34          // TODO: Add any constructor code after InitializeComponent
35          // call
36          //
37       }
38
39       /// <summary>
40       /// Clean up any resources being used.
41       /// </summary>
42       protected override void Dispose( bool disposing )
43       {
44          if( disposing )
45          {
46             if (components != null)
47             {
48                components.Dispose();
49             }
50          }
51          base.Dispose( disposing );
52       }
53
54       // Windows Form Designer generated code
55
56       /// <summary>
57       /// The main entry point for the application.
58       /// </summary>
59       [STAThread]
60       static void Main()
```

```
61          {
62             Application.Run( new FrmLogin() );
63          }
64
65          // checks to see if the user entered "John Doe"
66          private void btnLogin_Click(
67             object sender, System.EventArgs e )
68          {
69             if ( txtName.Text == "John Doe" )
70             {
71                MessageBox.Show( "Welcome, John!",
72                   MessageBoxIcon.Exclamation );
73             }
74
75          } // end method btnLogin_Click
76
77       } // end class FrmLogin
78    }
```

Missing arguments and using incorrect icon



Answer: The complete corrected code should read:

```
1   // Exercise 8.15 Solution
2   // Login.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace Login
12  {
13     /// <summary>
14     /// Summary description for FrmLogin.
15     /// </summary>
16     public class FrmLogin : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Button btnLogin;
19        private System.Windows.Forms.TextBox txtName;
20        private System.Windows.Forms.Label lblUserName;
21        /// <summary>
22        /// Required designer variable.
23        /// </summary>
24        private System.ComponentModel.Container components = null;
25
26        public FrmLogin()
27        {
28           //
29           // Required for Windows Form Designer support
```

```
30            //
31            InitializeComponent();
32
33            //
34            // TODO: Add any constructor code after InitializeComponent
35            // call
36            //
37         }
38
39         /// <summary>
40         /// Clean up any resources being used.
41         /// </summary>
42         protected override void Dispose( bool disposing )
43         {
44            if( disposing )
45            {
46               if (components != null)
47               {
48                  components.Dispose();
49               }
50            }
51            base.Dispose( disposing );
52         }
53
54         // Windows Form Designer generated code
55
56         /// <summary>
57         /// The main entry point for the application.
58         /// </summary>
59         [STAThread]
60         static void Main()
61         {
62            Application.Run (new FrmLogin() );
63         }
64
65         // checks to see if the user entered "John Doe"
66         private void btnLogin_Click(
67            object sender, System.EventArgs e )
68         {
69            if ( txtName.Text == "John Doe" )
70            {
71               MessageBox.Show( "Welcome, John!", "Welcome",
72                  MessageBoxButtons.OK, MessageBoxIcon.Information );
73            }
74
75         } // end method btnLogin_Click
76
77      } // end class FrmLogin
78   }
```

Using the Debugger

8.16 (*Sibling Survey Application*) The **Sibling Survey** application displays the siblings selected by the user in a dialog. If the user checks either the **Brother(s)** or **Sister(s)** `CheckBox`, and the **No Siblings** `CheckBox`, the user is asked to verify the selection. Otherwise, the user's selection is displayed in a `MessageBox`. While testing this application, you noticed that it does not execute properly. Use the debugger to find and correct the logic error(s) in the code. This exercise is located in the `C:\Examples\Tutorial08\Exercises\Debugger\SiblingSurvey` directory. Figure 8.24 shows the correct output for the application.

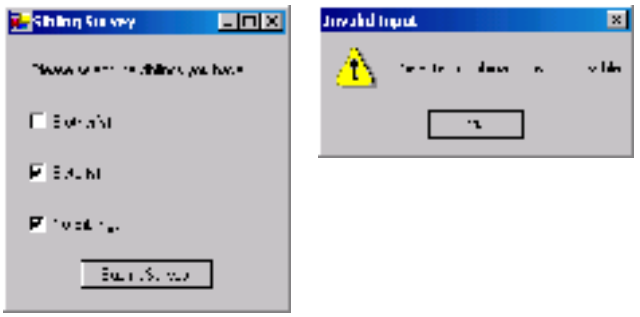


Figure 8.24 Correct output for the **Sibling Survey** application.

Answer:

```
// Exercise 8.16 Solution
// SiblingSurvey.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SiblingSurvey
{
   /// <summary>
   /// Summary description for FrmSiblingSurvey.
   /// </summary>
   public class FrmSiblingSurvey : System.Windows.Forms.Form
   {
      // Label that gives directions for the survey
      private System.Windows.Forms.Label lblDirections;

      // CheckBoxes to choose siblings
      private System.Windows.Forms.CheckBox chkBrother;
      private System.Windows.Forms.CheckBox chkSister;
      private System.Windows.Forms.CheckBox chkNone;

      // Button to submit information
      private System.Windows.Forms.Button btnSubmit;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmSiblingSurvey()
      {
         //
         // Required for Windows Form Designer support
```

```
38          //
39          InitializeComponent();
40
41          //
42          // TODO: Add any constructor code after InitializeComponent
43          // call
44          //
45       }
46
47       /// <summary>
48       /// Clean up any resources being used.
49       /// </summary>
50       protected override void Dispose( bool disposing )
51       {
52          if( disposing )
53          {
54             if (components != null)
55             {
56                components.Dispose();
57             }
58          }
59          base.Dispose( disposing );
60       }
61
62       // Windows Form Designer generated code
63
64       /// <summary>
65       /// The main entry point for the application.
66       /// </summary>
67       [STAThread]
68       static void Main()
69       {
70          Application.Run( new FrmSiblingSurvey() );
71       }
72
73       // handles Click event
74       private void btnSubmit_Click(
75          object sender, System.EventArgs e )
76       {
77          // check if user selects brothers or sisters
78          // and no siblings
79          if ( chkNone.Checked == true &&
80             ( chkBrother.Checked == true ||
81             chkSister.Checked == true ) )
82          {
83             MessageBox.Show( "Selected combination is not possible",
84                "Invalid Input", MessageBoxButtons.OK,
85                MessageBoxIcon.Exclamation );
86          }
87          // check if user selects CheckBox
88          else if ( chkNone.Checked == false &&
89             chkBrother.Checked == false &&
90             chkSister.Checked == false )
91          {
92             MessageBox.Show( "Please check at least one CheckBox",
93                "Invalid Input", MessageBoxButtons.OK,
94                MessageBoxIcon.Exclamation );
95          }
96          // check if user has brothers and sisters
97          else if ( chkBrother.Checked == true &&
```

Replaced && with ||

Replaced || with &&

```
98             chkSister.Checked == true )
99          {
100            MessageBox.Show( "You have brothers and sisters",
101               "Siblings", MessageBoxButtons.OK,
102               MessageBoxIcon.Information );
103         }
104         // check if user has brothers
105         else if ( chkBrother.Checked == true )
106         {
107            MessageBox.Show( "You have at least one brother",
108               "Siblings", MessageBoxButtons.OK,
109               MessageBoxIcon.Information );
110         }
111         // check if user has sisters
112         else if ( chkSister.Checked == true )
113         {
114            MessageBox.Show( "You have at least one sister",
115               "Siblings", MessageBoxButtons.OK,
116               MessageBoxIcon.Information );
117         }
118         // user has no siblings
119         else
120         {
121            MessageBox.Show( "You have no siblings",
122               "Siblings", MessageBoxButtons.OK,
123               MessageBoxIcon.Information );
124         }
125
126      } // end method btnSubmit_Click
127
128   } // end class FrmSiblingSurvey
129 }
```

Programming Challenge

8.17 (***Enhanced Fuzzy Dice Order Form Application***) Copy the directory `C:\Examples\Tutorial08\Exercises\FuzzyDiceOrderFormEnhanced` to your `C:\SimplyCSP` directory. Double click `FuzzyDiceOrderForm.sln` in the `FuzzyDiceOrderFormEnhanced` directory to open the application. Enhance the **Fuzzy Dice Order Form** application from Exercise 8.12 by replacing the **Calculate** `Button` with a **Clear** `Button`. The application should update the total cost, tax and shipping when the user changes any one of the three **Quantity:** field's values (Fig. 8.25). The **Clear** `Button` should return all fields to their original values. [*Hint*: You will need to use the `CheckBox` **CheckedChanged** event for each `CheckBox`. This event is raised when the state of a `CheckBox` changes. Double click a `CheckBox` in design view to create an event handler for that `CheckBox`'s `CheckedChanged` event. You also will need to assign `bool` values to the `CheckBox`es' `Checked` properties to control their states.]

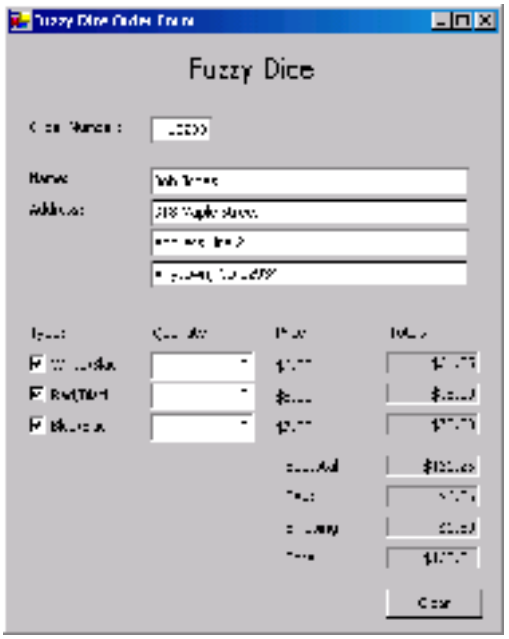


Figure 8.25 Enhanced **Fuzzy Dice Order Form** application

Answer:

```
// Exercise 8.17 Solution
// FuzzyDiceOrderForm.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace FuzzyDiceOrderForm
{
   /// <summary>
   /// Summary description for FrmFuzzyDiceOrderForm.
   /// </summary>
   public class FrmFuzzyDiceOrderForm : System.Windows.Forms.Form
   {
      // Label to display the Form's title
      private System.Windows.Forms.Label lblTitle;

      // Label and TextBox to input order number
      private System.Windows.Forms.Label lblOrderNumber;
      private System.Windows.Forms.TextBox txtOrderNumber;

      // Label and TextBox to input name
      private System.Windows.Forms.Label lblCompanyName;
      private System.Windows.Forms.TextBox txtName;

      // Label and TextBoxes to input address, city, state and zip
      private System.Windows.Forms.Label lblAddress;
      private System.Windows.Forms.TextBox txtAddressLine1;
      private System.Windows.Forms.TextBox txtAddressLine2;
      private System.Windows.Forms.TextBox txtCityStateZip;

      // Labels that serve as column headings for type, quantity,
```

```
36          // price and total cost
37          private System.Windows.Forms.Label lblType;
38          private System.Windows.Forms.Label lblQuantity;
39          private System.Windows.Forms.Label lblPrice;
40          private System.Windows.Forms.Label lblTotals;
41
42          // CheckBox, TextBox and Labels for white and black fuzzy dice
43          private System.Windows.Forms.CheckBox chkWhiteBlack;
44          private System.Windows.Forms.TextBox txtWhiteBlackQuantity;
45          private System.Windows.Forms.Label lblWhiteBlackPrice;
46          private System.Windows.Forms.Label lblWhiteBlackTotals;
47
48          // CheckBox, TextBox and Labels for red and black fuzzy dice
49          private System.Windows.Forms.CheckBox chkRedBlack;
50          private System.Windows.Forms.TextBox txtRedBlackQuantity;
51          private System.Windows.Forms.Label lblRedBlackPrice;
52          private System.Windows.Forms.Label lblRedBlackTotals;
53
54          // CheckBox, TextBox and Labels for blue and black fuzzy dice
55          private System.Windows.Forms.CheckBox chkBlueBlack;
56          private System.Windows.Forms.TextBox txtBlueBlackQuantity;
57          private System.Windows.Forms.Label lblBlueBlackPrice;
58          private System.Windows.Forms.Label lblBlueBlackTotals;
59
60          // Labels to display subtotal
61          private System.Windows.Forms.Label lblSubtotal;
62          private System.Windows.Forms.Label lblSubtotalResult;
63
64          // Labels to display tax
65          private System.Windows.Forms.Label lblTax;
66          private System.Windows.Forms.Label lblTaxResult;
67
68          // Labels to display shipping cost
69          private System.Windows.Forms.Label lblShipping;
70          private System.Windows.Forms.Label lblShippingResult;
71
72          // Labels to display total cost after tax and shipping
73          private System.Windows.Forms.Label lblTotal;
74          private System.Windows.Forms.Label lblTotalResult;
75
76          // Button to clear the Form
77          private System.Windows.Forms.Button btnClear;
78
79          /// <summary>
80          /// Required designer variable.
81          /// </summary>
82          private System.ComponentModel.Container components = null;
83
84          public FrmFuzzyDiceOrderForm()
85          {
86             //
87             // Required for Windows Form Designer support
88             //
89             InitializeComponent();
90
91             //
92             // TODO: Add any constructor code after InitializeComponent
93             // call
94             //
95          }
```

```
96
97          /// <summary>
98          /// Clean up any resources being used.
99          /// </summary>
100         protected override void Dispose( bool disposing )
101         {
102            if( disposing )
103            {
104               if (components != null)
105               {
106                  components.Dispose();
107               }
108            }
109            base.Dispose( disposing );
110         }
111
112         // Windows Form Designer generated code
113
114         /// <summary>
115         /// The main entry point for the application.
116         /// </summary>
117         [STAThread]
118         static void Main()
119         {
120            Application.Run( new FrmFuzzyDiceOrderForm() );
121         }
122
123         // handles TextChanged event
124         private void txtWhiteBlackQuantity_TextChanged(
125            object sender, System.EventArgs e )
126         {
127            // store quantity entered as int
128            int intNumberOfWhiteBlack =
129               Int32.Parse( txtWhiteBlackQuantity.Text );
130
131            // display message if user tries to enter a value
132            // without selecting CheckBox
133            if ( intNumberOfWhiteBlack != 0 &&
134               chkWhiteBlack.Checked == false )
135            {
136               // keep white/black quantity at 0
137               txtWhiteBlackQuantity.Text = "0";
138
139               // display message in dialog
140               MessageBox.Show(
141                  "Please check item you wish to purchase",
142                  "No Item Selected", MessageBoxButtons.OK,
143                  MessageBoxIcon.Exclamation );
144            }
145
146            // display message if order number, name or address fields
147            // are empty or contain default values
148            else if ( txtOrderNumber.Text == ""
149               || txtOrderNumber.Text == "0" || txtName.Text == ""
150               || txtName.Text == "Enter a name here"
151               || txtAddressLine1.Text == ""
152               || txtAddressLine1.Text == "Address Line 1"
153               || txtCityStateZip.Text == ""
154               || txtCityStateZip.Text == "City, State, zip" )
155            {
```

```
156             // display message in dialog
157             MessageBox.Show(
158                "Please fill out all information fields.",
159                "Empty Fields", MessageBoxButtons.OK,
160                MessageBoxIcon.Exclamation );
161          }
162          // display message if negative number entered
163          else if ( intNumberOfWhiteBlack < 0 )
164          {
165             txtWhiteBlackQuantity.Text = "0";
166             MessageBox.Show(
167                "Please enter a positive quantity",
168                "Bad Input", MessageBoxButtons.OK,
169                MessageBoxIcon.Exclamation );
170          }
171          else // calculate totals
172          {
173             // individual totals
174             // total of white/black dice
175             decimal decWhiteBlackTotals =
176                Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
177
178             // total of red/black dice
179             decimal decRedBlackTotals =
180                Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
181
182             // total of blue/black dice
183             decimal decBlueBlackTotals =
184                Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
185
186             // display individual totals
187             lblWhiteBlackTotals.Text =
188                String.Format( "{0:C}", decWhiteBlackTotals );
189             lblRedBlackTotals.Text =
190                String.Format( "{0:C}", decRedBlackTotals );
191             lblBlueBlackTotals.Text =
192                String.Format( "{0:C}", decBlueBlackTotals );
193
194             // subtotal, before tax and shipping
195             decimal decSubtotal = decWhiteBlackTotals
196                + decRedBlackTotals + decBlueBlackTotals;
197
198             lblSubtotalResult.Text =
199                String.Format( "{0:C}", decSubtotal );
200
201             // calculate and display tax
202             decimal decTax = decSubtotal * 0.05M;
203
204             lblTaxResult.Text = String.Format( "{0:C}", decTax );
205
206             // shipping
207             // $1.50 for up to 20 items
208             // free after 20 items
209             int intNumberOfItems =
210                ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
211                Int32.Parse( txtRedBlackQuantity.Text ) +
212                Int32.Parse( txtBlueBlackQuantity.Text ) );
213
214             decimal decShippingCost = 0.0M;
215
```

```
            // shipping is $1.50 if under 20 items ordered
            if ( intNumberOfItems <= 20 &&
               intNumberOfItems > 0 )
            {
               decShippingCost = 1.5M;
            }

            // display shipping cost
            lblShippingResult.Text =
               String.Format( "{0:C}", decShippingCost );

            // calculate and display total charge
            decimal decTotalCharge = decSubtotal + decTax +
               decShippingCost;

            lblTotalResult.Text =
               String.Format( "{0:C}", decTotalCharge );

         } // end else

      } // end method txtWhiteBlackQuantity_TextChanged

      // handles TextChanged event
      private void txtRedBlackQuantity_TextChanged(
         object sender, System.EventArgs e )
      {
         // store quantity entered as int
         int intNumberOfRedBlack =
            Int32.Parse( txtRedBlackQuantity.Text );

         // display message if user tries to enter a value
         // without selecting CheckBox
         if ( intNumberOfRedBlack != 0 &&
            chkRedBlack.Checked == false )
         {
            // keep red/black quantity at 0
            txtRedBlackQuantity.Text = "0";

            // display message in dialog
            MessageBox.Show(
               "Please check item you wish to purchase",
               "No Item Selected", MessageBoxButtons.OK,
               MessageBoxIcon.Exclamation );
         }

         // display message if order number, name or address fields
         // are empty or contain default values
         else if ( txtOrderNumber.Text == ""
            || txtOrderNumber.Text == "0" || txtName.Text == ""
            || txtName.Text == "Enter a name here"
            || txtAddressLine1.Text == ""
            || txtAddressLine1.Text == "Address Line 1"
            || txtCityStateZip.Text == ""
            || txtCityStateZip.Text == "City, State, zip" )
         {
            // display message in dialog
            MessageBox.Show(
               "Please fill out all information fields.",
               "Empty Fields", MessageBoxButtons.OK,
               MessageBoxIcon.Exclamation );
```

```
276          }
277          // display message if negative number entered
278          else if ( intNumberOfRedBlack < 0 )
279          {
280             txtRedBlackQuantity.Text = "0";
281             MessageBox.Show(
282                "Please enter a positive quantity",
283                "Bad Input", MessageBoxButtons.OK,
284                MessageBoxIcon.Exclamation );
285          }
286          else // calculate totals
287          {
288             // individual totals
289             // total of white/black dice
290             decimal decWhiteBlackTotals =
291                Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
292
293             // total of red/black dice
294             decimal decRedBlackTotals =
295                Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
296
297             // total of blue/black dice
298             decimal decBlueBlackTotals =
299                Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
300
301             // display individual totals
302             lblWhiteBlackTotals.Text =
303                String.Format( "{0:C}", decWhiteBlackTotals );
304             lblRedBlackTotals.Text =
305                String.Format( "{0:C}", decRedBlackTotals );
306             lblBlueBlackTotals.Text =
307                String.Format( "{0:C}", decBlueBlackTotals );
308
309             // subtotal, before tax and shipping
310             decimal decSubtotal = decWhiteBlackTotals
311                + decRedBlackTotals + decBlueBlackTotals;
312
313             lblSubtotalResult.Text =
314                String.Format( "{0:C}", decSubtotal );
315
316             // calculate and display tax
317             decimal decTax = decSubtotal * 0.05M;
318
319             lblTaxResult.Text = String.Format( "{0:C}", decTax );
320
321             // shipping
322             // $1.50 for up to 20 items
323             // free after 20 items
324             int intNumberOfItems =
325                ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
326                Int32.Parse( txtRedBlackQuantity.Text ) +
327                Int32.Parse( txtBlueBlackQuantity.Text ) );
328
329             decimal decShippingCost = 0.0M;
330
331             // shipping is $1.50 if under 20 items ordered
332             if ( intNumberOfItems <= 20 &&
333                intNumberOfItems > 0 )
334             {
335                decShippingCost = 1.5M;
```

```
336             }
337
338             // display shipping cost
339             lblShippingResult.Text =
340                String.Format( "{0:C}", decShippingCost );
341
342             // calculate and display total charge
343             decimal decTotalCharge = decSubtotal + decTax +
344                decShippingCost;
345
346             lblTotalResult.Text =
347                String.Format( "{0:C}", decTotalCharge );
348
349          } // end else
350
351       } // end method txtRedBlackQuantity_TextChanged
352
353       // handles TextChanged event
354       private void txtBlueBlackQuantity_TextChanged(
355          object sender, System.EventArgs e )
356       {
357          // store quantity entered as int
358          int intNumberOfBlueBlack =
359             Int32.Parse( txtWhiteBlueQuantity.Text );
360
361          // display message if user tries to enter a value
362          // without selecting CheckBox
363          if ( intNumberOfBlueBlack != 0 &&
364             chkWhiteBlack.Checked == false )
365          {
366             // keep blue/black quantity at 0
367             txtWhiteBlueQuantity.Text = "0";
368
369             // display message in dialog
370             MessageBox.Show(
371                "Please check item you wish to purchase",
372                "No Item Selected", MessageBoxButtons.OK,
373                MessageBoxIcon.Exclamation );
374          }
375
376          // display message if order number, name or address fields
377          // are empty or contain default values
378          else if ( txtOrderNumber.Text == ""
379             || txtOrderNumber.Text == "0" || txtName.Text == ""
380             || txtName.Text == "Enter a name here"
381             || txtAddressLine1.Text == ""
382             || txtAddressLine1.Text == "Address Line 1"
383             || txtCityStateZip.Text == ""
384             || txtCityStateZip.Text == "City, State, zip" )
385          {
386             // display message in dialog
387             MessageBox.Show(
388                "Please fill out all information fields.",
389                "Empty Fields", MessageBoxButtons.OK,
390                MessageBoxIcon.Exclamation );
391          }
392          // display message if negative number entered
393          else if ( intNumberOfBlueBlack < 0 )
394          {
395             txtBlueBlackQuantity.Text = "0";
```

```
396             MessageBox.Show(
397                "Please enter a positive quantity",
398                "Bad Input", MessageBoxButtons.OK,
399                MessageBoxIcon.Exclamation );
400          }
401          else // calculate totals
402          {
403             // individual totals
404             // total of white/black dice
405             decimal decWhiteBlackTotals =
406                Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
407
408             // total of red/black dice
409             decimal decRedBlackTotals =
410                Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
411
412             // total of blue/black dice
413             decimal decBlueBlackTotals =
414                Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
415
416             // display individual totals
417             lblWhiteBlackTotals.Text =
418                String.Format( "{0:C}", decWhiteBlackTotals );
419             lblRedBlackTotals.Text =
420                String.Format( "{0:C}", decRedBlackTotals );
421             lblBlueBlackTotals.Text =
422                String.Format( "{0:C}", decBlueBlackTotals );
423
424             // subtotal, before tax and shipping
425             decimal decSubtotal = decWhiteBlackTotals
426                + decRedBlackTotals + decBlueBlackTotals;
427
428             lblSubtotalResult.Text =
429                String.Format( "{0:C}", decSubtotal );
430
431             // calculate and display tax
432             decimal decTax = decSubtotal * 0.05M;
433
434             lblTaxResult.Text = String.Format( "{0:C}", decTax );
435
436             // shipping
437             // $1.50 for up to 20 items
438             // free after 20 items
439             int intNumberOfItems =
440                ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
441                Int32.Parse( txtRedBlackQuantity.Text ) +
442                Int32.Parse( txtBlueBlackQuantity.Text ) );
443
444             decimal decShippingCost = 0.0M;
445
446             // shipping is $1.50 if under 20 items ordered
447             if ( intNumberOfItems <= 20 &&
448                intNumberOfItems > 0 )
449             {
450                decShippingCost = 1.5M;
451             }
452
453             // display shipping cost
454             lblShippingResult.Text =
455                String.Format( "{0:C}", decShippingCost );
```

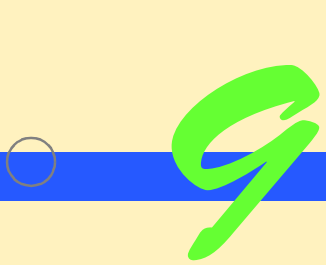
```
456
457             // calculate and display total charge
458             decimal decTotalCharge = decSubtotal + decTax +
459                decShippingCost;
460
461             lblTotalResult.Text =
462                String.Format( "{0:C}", decTotalCharge );
463
464          } // end else
465
466       } // end method txtBlueBlackQuantity_TextChanged
467
468       // clear all fields
469       private void btnClear_Click(
470          object sender, System.EventArgs e )
471       {
472          // set all fields to their original values
473          txtOrderNumber.Text = "0";
474          txtName.Text = "Enter name here";
475          txtAddressLine1.Text = "Address Line 1";
476          txtAddressLine2.Text = "Address Line 2";
477          txtCityStateZip.Text = "City, State, zip";
478          txtWhiteBlackQuantity.Text = "0";
479          txtRedBlackQuantity.Text = "0";
480          txtBlueBlackQuantity.Text = "0";
481          lblWhiteBlackTotals.Text = "$0.00";
482          lblRedBlackTotals.Text = "$0.00";
483          lblBlueBlackTotals.Text = "$0.00";
484          lblSubtotalResult.Text = "$0.00";
485          lblTaxResult.Text = "$0.00";
486          lblShippingResult.Text = "$0.00";
487          lblTotalResult.Text = "$0.00";
488          chkWhiteBlack.Checked = false;
489          chkRedBlack.Checked = false;
490          chkBlueBlack.Checked = false;
491
492       } // end method btnClear_Click
493
494       // handles CheckedChanged event
495       private void chkWhiteBlack_CheckedChanged(
496          object sender, System.EventArgs e )
497       {
498          txtWhiteBlackQuantity.Text = "0";
499          lblWhiteBlackTotals.Text = "0";
500
501          // individual totals
502          // total of white/black dice
503          decimal decWhiteBlackTotals =
504             Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
505
506          // total of red/black dice
507          decimal decRedBlackTotals =
508             Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
509
510          // total of blue/black dice
511          decimal decBlueBlackTotals =
512             Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
513
514          // display individual totals
515          lblWhiteBlackTotals.Text =
```

```
516            String.Format( "{0:C}", decWhiteBlackTotals );
517         lblRedBlackTotals.Text =
518            String.Format( "{0:C}", decRedBlackTotals );
519         lblBlueBlackTotals.Text =
520            String.Format( "{0:C}", decBlueBlackTotals );
521
522         // subtotal, before tax and shipping
523         decimal decSubtotal = decWhiteBlackTotals
524            + decRedBlackTotals + decBlueBlackTotals;
525
526         lblSubtotalResult.Text =
527            String.Format( "{0:C}", decSubtotal );
528
529         // calculate and display tax
530         decimal decTax = decSubtotal * 0.05M;
531
532         lblTaxResult.Text = String.Format( "{0:C}", decTax );
533
534         // shipping
535         // $1.50 for up to 20 items
536         // free after 20 items
537         int intNumberOfItems =
538            ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
539            Int32.Parse( txtRedBlackQuantity.Text ) +
540            Int32.Parse( txtBlueBlackQuantity.Text ) );
541
542         decimal decShippingCost = 0.0M;
543
544         // shipping is $1.50 if under 20 items ordered
545         if ( intNumberOfItems <= 20 &&
546            intNumberOfItems > 0 )
547         {
548            decShippingCost = 1.5M;
549         }
550
551         // display shipping cost
552         lblShippingResult.Text =
553            String.Format( "{0:C}", decShippingCost );
554
555         // calculate and display total charge
556         decimal decTotalCharge = decSubtotal + decTax +
557            decShippingCost;
558
559         lblTotalResult.Text =
560            String.Format( "{0:C}", decTotalCharge );
561
562      } // end method chkWhiteBlack_CheckedChanged
563
564      // handles CheckedChanged event
565      private void chkRedBlack_CheckedChanged(
566         object sender, System.EventArgs e )
567      {
568         txtRedBlackQuantity.Text = "0";
569         lblRedBlackTotals.Text = "0";
570
571         // individual totals
572         // total of white/black dice
573         decimal decWhiteBlackTotals =
574            Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
575
```

```
576          // total of red/black dice
577          decimal decRedBlackTotals =
578             Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
579
580          // total of blue/black dice
581          decimal decBlueBlackTotals =
582             Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
583
584          // display individual totals
585          lblWhiteBlackTotals.Text =
586             String.Format( "{0:C}", decWhiteBlackTotals );
587          lblRedBlackTotals.Text =
588             String.Format( "{0:C}", decRedBlackTotals );
589          lblBlueBlackTotals.Text =
590             String.Format( "{0:C}", decBlueBlackTotals );
591
592          // subtotal, before tax and shipping
593          decimal decSubtotal = decWhiteBlackTotals
594             + decRedBlackTotals + decBlueBlackTotals;
595
596          lblSubtotalResult.Text =
597             String.Format( "{0:C}", decSubtotal );
598
599          // calculate and display tax
600          decimal decTax = decSubtotal * 0.05M;
601
602          lblTaxResult.Text = String.Format( "{0:C}", decTax );
603
604          // shipping
605          // $1.50 for up to 20 items
606          // free after 20 items
607          int intNumberOfItems =
608             ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
609             Int32.Parse( txtRedBlackQuantity.Text ) +
610             Int32.Parse( txtBlueBlackQuantity.Text ) );
611
612          decimal decShippingCost = 0.0M;
613
614          // shipping is $1.50 if under 20 items ordered
615          if ( intNumberOfItems <= 20 &&
616             intNumberOfItems > 0 )
617          {
618             decShippingCost = 1.5M;
619          }
620
621          // display shipping cost
622          lblShippingResult.Text =
623             String.Format( "{0:C}", decShippingCost );
624
625          // calculate and display total charge
626          decimal decTotalCharge = decSubtotal + decTax +
627             decShippingCost;
628
629          lblTotalResult.Text =
630             String.Format( "{0:C}", decTotalCharge );
631
632       } // end method chkRedBlack_CheckedChanged
633
634       // handles CheckedChanged event
635       private void chkBlueBlack_CheckedChanged(
```

```
636         object sender, System.EventArgs e )
637      {
638         txtBlueBlackQuantity.Text = "0";
639         lblBlueBlackTotals.Text = "0";
640
641         // individual totals
642         // total of white/black dice
643         decimal decWhiteBlackTotals =
644            Int32.Parse( txtWhiteBlackQuantity.Text ) * 6.25M;
645
646         // total of red/black dice
647         decimal decRedBlackTotals =
648            Int32.Parse( txtRedBlackQuantity.Text ) * 5.00M;
649
650         // total of blue/black dice
651         decimal decBlueBlackTotals =
652            Int32.Parse( txtBlueBlackQuantity.Text ) * 7.50M;
653
654         // display individual totals
655         lblWhiteBlackTotals.Text =
656            String.Format( "{0:C}", decWhiteBlackTotals );
657         lblRedBlackTotals.Text =
658            String.Format( "{0:C}", decRedBlackTotals );
659         lblBlueBlackTotals.Text =
660            String.Format( "{0:C}", decBlueBlackTotals );
661
662         // subtotal, before tax and shipping
663         decimal decSubtotal = decWhiteBlackTotals
664            + decRedBlackTotals + decBlueBlackTotals;
665
666         lblSubtotalResult.Text =
667            String.Format( "{0:C}", decSubtotal );
668
669         // calculate and display tax
670         decimal decTax = decSubtotal * 0.05M;
671
672         lblTaxResult.Text = String.Format( "{0:C}", decTax );
673
674         // shipping
675         // $1.50 for up to 20 items
676         // free after 20 items
677         int intNumberOfItems =
678            ( Int32.Parse( txtWhiteBlackQuantity.Text ) +
679            Int32.Parse( txtRedBlackQuantity.Text ) +
680            Int32.Parse( txtBlueBlackQuantity.Text ) );
681
682         decimal decShippingCost = 0.0M;
683
684         // shipping is $1.50 if under 20 items ordered
685         if ( intNumberOfItems <= 20 &&
686            intNumberOfItems > 0 )
687         {
688            decShippingCost = 1.5M;
689         }
690
691         // display shipping cost
692         lblShippingResult.Text =
693            String.Format( "{0:C}", decShippingCost );
694
695         // calculate and display total charge
```

```
696          decimal decTotalCharge = decSubtotal + decTax +
697             decShippingCost;
698
699          lblTotalResult.Text =
700             String.Format( "{0:C}", decTotalCharge );
701
702       } // end method chkBlueBlack_CheckedChanged
703
704    } // end class FrmFuzzyDiceOrderForm
705 }
```



TUTORIAL

Car Payment Calculator Application

Introducing the `while` Repetition Statement

Solutions

Instructor's Manual Exercise Solutions Tutorial 9

MULTIPLE-CHOICE QUESTIONS

9.1 If a `while` statement's loop-continuation condition is initially false, how many times will the body execute?

a) 0
b) 1
c) infinite
d) There is no way to know in advance.

9.2 The _____ statement executes until its loop-continuation condition becomes `false`.

a) `while`
b) `until`
c) `loop`
d) `whileTrue`

9.3 A(n) _____ loop occurs when a condition in a `while` loop never becomes `false`.

a) infinite
b) undefined
c) nested
d) indefinite

9.4 A _____ is a variable that helps control the number of times that a set of statements will execute.

a) repeater
b) counter
c) loop
d) repetition control statement

9.5 The _____ control allows users to add and view items in a list.

a) `ListItems`
b) `SelectBox`
c) `ListBox`
d) `ViewBox`

9.6 In a UML activity diagram, a(n) _____ symbol joins two flows of activity into one flow of activity.

a) merge
b) combine
c) action state
d) decision

9.7 The _____ property returns an object containing all the values in a `ListBox`.

a) `All`
b) `List`
c) `ListItemValues`
d) `Items`

9.8 The _____ method deletes all the values in a `ListBox`.

a) `Remove`
b) `Delete`
c) `Clear`
d) `Del`

9.9 The _____ method of property `Items` adds an item to a `ListBox`.

a) `Include`
b) `Append`
c) `Add`
d) `Insert`

9.10 The _____ method raises a number to an exponent.

a) `Math.Exp`
b) `Math.Exponent`
c) `Math.Power`
d) `Math.Pow`

Answers: 9.1) a. 9.2) a. 9.3) a. 9.4) b. 9.5) c. 9.6) a. 9.7) d. 9.8) c. 9.9) c. 9.10) d.

EXERCISES

9.11 ***(Table of Powers Application)*** Write an application that displays a table of numbers from 1 to an upper limit, along with each number's squared value (for example, the number *n* to the power 2, or *n* ^ 2) and cubed value (the number *n* to the power 3, or *n* ^ 3). The users should specify the upper limit, and the results should be displayed in a `ListBox`, as in Fig. 9.21.

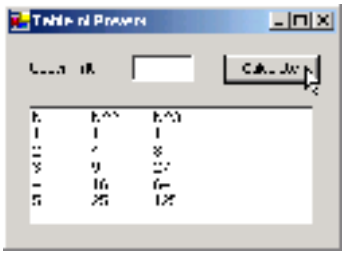


Figure 9.21 Table of Powers application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial09\Exercises\TableOfPowers` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `TableOfPowers.sln` in the `TableOfPowers` directory to open the application.

c) ***Adding a `ListBox`.*** Add a `ListBox` to the application, as shown in Fig. 9.21. Name the `ListBox` `lstResults`. Rearrange and comment the new control declaration appropriately.

d) ***Adding the Upper limit: `TextBox` event handler.*** Double click the **Upper limit:** `TextBox` to generate an event handler for this `TextBox`'s `TextChanged` event. In this event handler, clear the `ListBox`.

e) ***Adding the Calculate `Button` event handler.*** Double click the **Calculate** `Button` to generate the empty event handler `btnCalculate_Click`. Add the code specified by the remaining steps to this event handler.

f) ***Clearing the `ListBox`.*** Use the `Clear` method on the `Items` property to clear the `ListBox` from any previous data.

g) ***Obtaining the upper limit supplied by the user.*** Assign the value entered by the user in the **Upper limit:** `TextBox` to a variable. Note that the `TextBox`'s `Name` property is set to `txtInput`.

h) ***Adding a header.*** Use the `Add` method on the `Items` property to insert a header in the `ListBox`. The header should label three columns—`N`, `N^2` and `N^3`. Column headings should be separated by tab characters.

i) ***Calculating the powers from 1 to the specified upper limit.*** Use a `while` statement to calculate the squared value and the cubed value of each number from `1` to the upper limit, inclusive. Add an item to the `ListBox` containing the current number being analyzed, its squared value and its cubed value. Use the `Math.Pow` method to perform the exponentiations.

j) ***Incrementing the counter.*** Remember to increment the counter appropriately each time through the loop.

k) ***Running the application.*** Select **Debug > Start** to run your application. Enter an upper limit and click the **Calculate** `Button`. Verify that the table of powers displayed contains the correct values.

l) ***Closing the application.*** Close your running application by clicking its close box.

m) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 9.11 Solution
2  // TableOfPowers.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace TableOfPowers
```

```
12  {
13     /// <summary>
14     /// Summary description for FrmTableOfPowers.
15     /// </summary>
16     public class FrmTableOfPowers : System.Windows.Forms.Form
17     {
18        // Label and TextBox to input upper limit of N
19        private System.Windows.Forms.Label lblUpperLimit;
20        private System.Windows.Forms.TextBox txtInput;
21
22        // Button to compute powers
23        private System.Windows.Forms.Button btnCalculate;
24
25        // ListBox to list the powers
26        private System.Windows.Forms.ListBox lstResults;
27
28        /// <summary>
29        /// Required designer variable.
30        /// </summary>
31        private System.ComponentModel.Container components = null;
32
33        public FrmTableOfPowers()
34        {
35           //
36           // Required for Windows Form Designer support
37           //
38           InitializeComponent();
39           //
40           // TODO: Add any constructor code after InitializeComponent
41           // call
42           //
43        }
44
45        /// <summary>
46        /// Clean up any resources being used.
47        /// </summary>
48        protected override void Dispose( bool disposing )
49        {
50           if( disposing )
51           {
52              if (components != null)
53              {
54                 components.Dispose();
55              }
56           }
57           base.Dispose( disposing );
58        }
59
60        // Windows Form Designer generated code
61
62        /// <summary>
63        /// The main entry point for the application.
64        /// </summary>
65        [STAThread]
66        static void Main()
67        {
68           Application.Run( new FrmTableOfPowers() );
69        }
70
71        // handles Click event
```

```
72          private void btnCalculate_Click(
73             object sender, System.EventArgs e )
74          {
75             int intLimit = 0;   // upper limit set by user
76             int intCounter = 1; // counter begins at 1
77
78             // clear ListBox
79             lstResults.Items.Clear();
80
81             // retrieve user input
82             intLimit = Int32.Parse( txtInput.Text );
83
84             // add header
85             lstResults.Items.Add( "N\tN^2\tN^3" );
86
87             // calculate and display square and cube of 1 to intLimit
88             while ( intCounter <= intLimit )
89             {
90                lstResults.Items.Add( intCounter + "\t" +
91                   Math.Pow( intCounter, 2 ) + "\t" +
92                   Math.Pow( intCounter, 3 ) );
93
94                // increment counter
95                intCounter++;
96             }
97
98          } // end method btnCalculate_Click
99
100         // handles TextChanged event
101         private void txtInput_TextChanged(
102            object sender, System.EventArgs e )
103         {
104            lstResults.Items.Clear();
105
106         } // end method txtInput_TextChanged
107
108      } // end class FrmTableOfPowers
109   }
```

9.12 ***(Mortgage Calculator Application)*** A bank offers mortgages that can be repaid in 5, 10, 15, 20, 25 or 30 years. Write an application that allows a user to enter the price of a house (the amount of the mortgage) and the annual interest rate. When the user clicks a `Button`, the application displays a table of the mortgage length in years together with the monthly payment, as shown in Fig. 9.22.

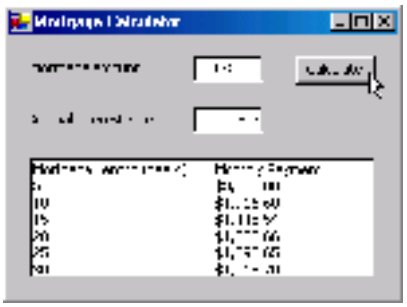


Figure 9.22 **Mortgage Calculator** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial09\Exercises\MortgageCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `MortgageCalculator.sln` in the `MortgageCalculator` directory to open the application.

c) ***Adding a `ListBox` to display the results.*** Add a `ListBox` as shown in Fig. 9.22. Name the `ListBox` `lstResults`. Rearrange and comment the new control declaration appropriately.

d) ***Adding a Calculate `Button` event handler.*** Double click the **Calculate** `Button` to generate the empty event handler `btnCalculate_Click`. Add the code specified in the remaining steps to your event handler.

e) ***Converting the annual interest rate to the monthly interest rate.*** To convert the annual interest rate from a percent value into its `double` equivalent, divide the annual rate by `100`. Then, divide the `double` annual rate by `12` to obtain the monthly rate.

f) ***Clearing the `ListBox`.*** Use the `Clear` method on the `Items` property to clear the `ListBox` from any previous data.

g) ***Displaying a header.*** Use the `Add` method to display a header in the `ListBox`. The header should be the column headers "`Mortgage Length (Years)`" and "`Monthly Payment`", separated by a tab character.

h) ***Using a repetition statement.*** Add a `while` repetition statement to calculate six monthly payment options for the user's mortgage. Each option has a different number of years that the mortgage can last. For this exercise, use the following numbers of years: `5`, `10`, `15`, `20`, `25` and `30`.

i) ***Converting the length of the mortgage from years to months.*** Convert the number of years to months.

j) ***Calculating the monthly payments for six different mortgages.*** Use the formula in lines 218–222 of Fig. 9.17 to compute the monthly payments, given the monthly interest rate, the number of months in the mortgage and the mortgage amount.

k) ***Displaying the results.*** Use the `Add` method of the `Items` property to display the length of the mortgage in years and the monthly payments in the `ListBox`. You will need to use three tab characters to ensure that the monthly payments appear in the second column.

l) ***Running the application.*** Select **Debug > Start** to run your application. Enter a mortgage amount and annual interest rate, then click the **Calculate** `Button`. Verify that the monthly payments displayed contain the correct values.

m) ***Closing the application.*** Close your running application by clicking its close box.

n) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 9.12 Solution
2  // MortgageCalculator.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace MortgageCalculator
12 {
13    /// <summary>
14    /// Summary description for FrmMortgageCalculator.
15    /// </summary>
16    public class FrmMortgageCalculator : System.Windows.Forms.Form
17    {
18       // Label and TextBox to enter mortgage amount
19       private System.Windows.Forms.Label lblMortgageAmount;
20       private System.Windows.Forms.TextBox txtMortgageAmount;
21
```

```
22          // Label and TextBox to enter interest rate
23          private System.Windows.Forms.Label lblInterestRate;
24          private System.Windows.Forms.TextBox txtRate;
25
26          // Button to calculate mortgage payments
27          private System.Windows.Forms.Button btnCalculate;
28
29          // ListBox to display mortgage payments
30          private System.Windows.Forms.ListBox lstResults;
31
32          /// <summary>
33          /// Required designer variable.
34          /// </summary>
35          private System.ComponentModel.Container components = null;
36
37          public FrmMortgageCalculator()
38          {
39             //
40             // Required for Windows Form Designer support
41             //
42             InitializeComponent();
43             //
44             // TODO: Add any constructor code after InitializeComponent
45             // call
46             //
47          }
48
49          /// <summary>
50          /// Clean up any resources being used.
51          /// </summary>
52          protected override void Dispose( bool disposing )
53          {
54             if( disposing )
55             {
56                if (components != null)
57                {
58                   components.Dispose();
59                }
60             }
61             base.Dispose( disposing );
62          }
63
64          // Windows Form Designer generated code
65
66          /// <summary>
67          /// The main entry point for the application.
68          /// </summary>
69          [STAThread]
70          static void Main()
71          {
72             Application.Run( new FrmMortgageCalculator() );
73          }
74
75          // handles Click event
76          private void btnCalculate_Click(
77             object sender, System.EventArgs e )
78          {
79             int intMortgageAmount = 0;     // house price
80             double dblAnnualRate = 0;      // annual interest rate
81             double dblMonthlyRate = 0;     // monthly interest rate
```

```
82          decimal decPayment = 0;        // monthly payment amount
83          int intYears = 5;              // years in mortgage
84          int intMonths = 0;             // months in mortgage
85
86          // obtain user input
87          intMortgageAmount = Int32.Parse( txtMortgageAmount.Text );
88          dblAnnualRate = Double.Parse( txtRate.Text ) / 100;
89
90          // calculate monthly interest rate
91          dblMonthlyRate = dblAnnualRate / 12;
92
93          // clear previous results from ListBox
94          lstResults.Items.Clear();
95
96          // add header to ListBox
97          lstResults.Items.Add(
98             "Mortgage Length (Years)\tMonthly Payment" );
99
100         // perform payment calculation and display result for
101         // 5, 10, 15, 20, 25 and 30 years
102         while ( intYears <= 30 )
103         {
104            // convert years to months for the calculation
105            intMonths = intYears * 12;
106
107            // perform calculation
108            decPayment = ( decimal )
109               ( intMortgageAmount * dblMonthlyRate *
110               Math.Pow( 1 + dblMonthlyRate, intMonths ) /
111               ( Math.Pow( 1 + dblMonthlyRate, intMonths )
112               - 1 ) );
113
114            // display result
115            lstResults.Items.Add( intYears + "\t\t\t" +
116               String.Format( "{0:C}", decPayment ) );
117
118            // increment counter
119            intYears += 5;
120
121         } // end while
122
123      } // end method btnCalculate_Click
124
125   } // end class FrmMortgageCalculator
126 }
```

9.13 ***(Office Supplies Application)*** Create an application that allows a user to make a list of office supplies to buy, as shown in Fig. 9.23. The user should enter the supply in a `TextBox` and click the **Buy** `Button` to add it to the `ListBox`. The **Clear** `Button` removes all the items from the `ListBox`.

Figure 9.23 Office Supplies application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial09\Exercises\OfficeSupplies` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `OfficeSupplies.sln` in the `OfficeSupplies` directory to open the application.

c) ***Adding a `ListBox`.*** Add a `ListBox` to the `Form`. Name the `ListBox` `lstSupplies`. Place and size it as shown in Fig. 9.23. Rearrange and comment the new control declaration appropriately.

d) ***Adding an event handler for the Buy `Button`.*** Double click the **Buy** `Button` to generate the `btnBuy_Click` event handler. The event handler should obtain the user input from the `TextBox`. The user input is then added as an item into the `ListBox`. After the input is added to the `ListBox`, clear the **Supply:** `TextBox`.

e) ***Adding an event handler for the Clear `Button`.*** Double click the **Clear** `Button` to generate the `btnClear_Click` event handler. The event handler should use the `Clear` method on the `Items` property to clear the `ListBox`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter several items into the **Supply:** `TextBox` and click the **Buy** `Button` after entering each item. Verify that each item is added to the `ListBox`. Click the **Clear** `Button` and verify that all items are removed from the `ListBox`.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 9.13 Solution
2  // OfficeSupplies.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace OfficeSupplies
12 {
13    /// <summary>
14    /// Summary description for FrmOfficeSupplies.
15    /// </summary>
16    public class FrmOfficeSupplies : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input an office supply
19       private System.Windows.Forms.Label lblOfficeSupply;
20       private System.Windows.Forms.TextBox txtOfficeSupply;
21
22       // ListBox to list the supplies
23       private System.Windows.Forms.ListBox lstSupplies;
24
```

```
25          // Button to buy a supply
26          private System.Windows.Forms.Button btnBuy;
27
28          // Button to clear the list of supplies
29          private System.Windows.Forms.Button btnClear;
30
31          /// <summary>
32          /// Required designer variable.
33          /// </summary>
34          private System.ComponentModel.Container components = null;
35
36          public FrmOfficeSupplies()
37          {
38             //
39             // Required for Windows Form Designer support
40             //
41             InitializeComponent();
42             //
43             // TODO: Add any constructor code after InitializeComponent
44             // call
45             //
46          }
47
48          /// <summary>
49          /// Clean up any resources being used.
50          /// </summary>
51          protected override void Dispose( bool disposing )
52          {
53             if( disposing )
54             {
55                if (components != null)
56                {
57                   components.Dispose();
58                }
59             }
60             base.Dispose( disposing );
61          }
62
63          // Windows Form Designer generated code
64
65          /// <summary>
66          /// The main entry point for the application.
67          /// </summary>
68          [STAThread]
69          static void Main()
70          {
71             Application.Run( new FrmOfficeSupplies() );
72          }
73
74          // handles Buy Button's Click event
75          private void btnBuy_Click(
76             object sender, System.EventArgs e )
77          {
78             // add supply item to ListBox, clear input TextBox
79             lstSupplies.Items.Add( txtOfficeSupply.Text );
80             txtOfficeSupply.Text = "";
81
82          } // end method btnBuy_Click
83
84          // handles Clear Button's Click event
```

```
85          private void btnClear_Click(
86             object sender, System.EventArgs e )
87          {
88             lstSupplies.Items.Clear();
89
90          } // end method btnClear_Click
91
92       } // end class FrmOfficeSupplies
93    }
```

What does this code do?

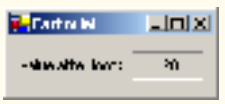
9.14 What value does `intMysteryValue` contain when the following code is completed?

```
1  int intX = 1;
2  int intMysteryValue = 1;
3
4  while ( intX < 6 )
5  {
6     intMysteryValue *= intX;
7     intX++;
8  }
```

Answer: `intMysteryValue` = 120. The complete code reads:

```
1  // Exercise 9.14 Solution
2  // CalcValue.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace CalcValue
12 {
13    /// <summary>
14    /// Summary description for FrmCalcValue.
15    /// </summary>
16    public class FrmCalcValue : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.Label lblValue;
20       /// <summary>
21       /// Required designer variable.
22       /// </summary>
23       private System.ComponentModel.Container components = null;
24
25       public FrmCalcValue()
26       {
27          //
28          // Required for Windows Form Designer support
29          //
30          InitializeComponent();
31          //
32          // TODO: Add any constructor code after InitializeComponent
33          // call
34          //
```

```
35         }
36
37         /// <summary>
38         /// Clean up any resources being used.
39         /// </summary>
40         protected override void Dispose( bool disposing )
41         {
42            if( disposing )
43            {
44               if (components != null)
45               {
46                  components.Dispose();
47               }
48            }
49            base.Dispose( disposing );
50         }
51
52         // Windows Form Designer generated code
53
54         /// <summary>
55         /// The main entry point for the application.
56         /// </summary>
57         [STAThread]
58         static void Main()
59         {
60            Application.Run( new FrmCalcValue() );
61         }
62
63         //  performs several calculations using loops
64         private void FrmCalcValue_Load(
65            object sender, System.EventArgs e )
66         {
67            int intX = 1;
68            int intMysteryValue = 1;
69
70            while ( intX < 6 )
71            {
72               intMysteryValue *= intX;
73               intX++;
74            }
75
76            lblDisplay.Text = Convert.ToString( intMysteryValue );
77
78         } // end method FrmCalcValue_Load
79
80      } // end class FrmCalcValue
81   }
```



What's wrong with this code?

9.15 Find the error(s) in the following code:

a) Assume that the `intX` variable is declared and initialized to 1. The loop should total the numbers from 1 to 10.

```
while ( ! ( intX <= 10 ) )
{
   intTotal += intX;
   intX++;
}
```

Answer: This loop will never execute, as `intX` is already less than or equal to `10`. The `!` (not) symbol should be removed from the loop.

b) Assume that the `intCounter` variable is declared and initialized to `1`. The loop should sum the numbers from `1` to `100`.

```
int intCounter = 1;

while ( intCounter <= 100 )
{
   intTotal += intCounter;
}

intCounter++;
```

Answer: This is an infinite loop, as `intCounter` will never be greater than `100`. The statement that increments `intCounter` must be placed within the `while` statement.

c) Assume that the `intCounter` variable is declared and initialized to `1000`. The loop should iterate from `1000` to `1`.

```
intCounter = 1000;

while ( intCounter > 0 )
{
   lblDisplay.Text = Convert.ToString( intCounter );
   intCounter++;
}
```

Answer: The values must decrease. The value `1` should be subtracted from, rather than added to, `intCounter`.

d) Assume that the `intCounter` variable is declared and initialized to `1`. The loop should execute five times, adding the numbers `1`–`5` to a `ListBox`.

```
intCounter = 1;

while ( intCounter < 5 )
{
   lstNumbers.Items.Add( intCounter );
   intCounter++;
}
```

Answer: This loop will execute only four times. To fix the application, the loop-continuation condition should use the `<=` operator, rather than the `<` operator. The complete incorrect code is shown below. [*Note*: Because part *b* is an infinite loop, the `Form` never appears. As a result, we do not show any output at the end of this code listing.]:

```
// Exercise 9.15 Solution
// LoopTest.cs (Incorrect)

using System;
```

```
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace LoopTest
12 {
13    /// <summary>
14    /// Summary description for FrmLoopTest.
15    /// </summary>
16    public class FrmLoopTest : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.Label lblLoop2Out;
20       private System.Windows.Forms.Label lblLoop1Out;
21       private System.Windows.Forms.ListBox lstNumbers;
22       private System.Windows.Forms.Label lblLoop4;
23       private System.Windows.Forms.Label lblLoop3;
24       private System.Windows.Forms.Label lblLoop2;
25       private System.Windows.Forms.Label lblLoop1;
26       /// <summary>
27       /// Required designer variable.
28       /// </summary>
29       private System.ComponentModel.Container components = null;
30
31       public FrmLoopTest()
32       {
33          //
34          // Required for Windows Form Designer support
35          //
36          InitializeComponent();
37          //
38          // TODO: Add any constructor code after InitializeComponent
39          // call
40          //
41       }
42
43       /// <summary>
44       /// Clean up any resources being used.
45       /// </summary>
46       protected override void Dispose( bool disposing )
47       {
48          if( disposing )
49          {
50             if (components != null)
51             {
52                components.Dispose();
53             }
54          }
55          base.Dispose( disposing );
56       }
57
58       // Windows Form Designer generated code
59
60       /// <summary>
61       /// The main entry point for the application.
62       /// </summary>
63       [STAThread]
64       static void Main()
```

```
65          {
66             Application.Run( new FrmLoopTest() );
67          }
68
69          //  performs several calculations using loops
70          private void FrmLoopTest_Load(
71             object sender, System.EventArgs e)
72          {
73             // code for loop 1
74             int intTotal = 0;
75             int intX = 1;
76
77             while ( !( intX <= 10 ) )
78             {
79                intTotal += intX;
80                intX++;
81             }
82
83             // display the result
84             lblLoop1Out.Text = Convert.ToString( intTotal );
85
86             // code for loop 2
87             int intCounter = 1;
88
89             while ( intCounter <= 100 )
90             {
91                intTotal += intCounter;
92             }
93
94             intCounter++;
95
96             // display the result
97             lblLoop2Out.Text = Convert.ToString( intTotal );
98
99             // code for loop 3
100            intCounter = 1000;
101
102            while ( intCounter > 0 )
103            {
104               lblDisplay.Text = Convert.ToString( intCounter );
105               intCounter++;
106            }
107
108            // code for loop 4
109            intCounter = 1;
110
111            while ( intCounter < 5 )
112            {
113               lstNumbers.Items.Add( intCounter );
114               intCounter++;
115            }
116
117         } // end method FrmLoopTest_Load
118
119      } // end class FrmLoopTest
120   }
```

The not (!) operator will prevent this loop from executing

The counter never increments because it is outside the loop

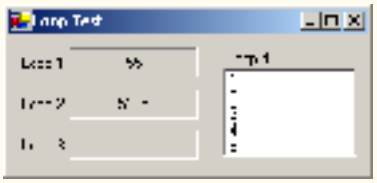
The counter should decrease, not increase

This should use <= rather than <

Answer: The complete corrected code should read:

```
1  // Exercise 9.15 Solution
2  // LoopTest.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace LoopTest
12 {
13    /// <summary>
14    /// Summary description for FrmLoopTest.
15    /// </summary>
16    public class FrmLoopTest : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.Label lblLoop2Out;
20       private System.Windows.Forms.Label lblLoop1Out;
21       private System.Windows.Forms.ListBox lstNumbers;
22       private System.Windows.Forms.Label lblLoop4;
23       private System.Windows.Forms.Label lblLoop3;
24       private System.Windows.Forms.Label lblLoop2;
25       private System.Windows.Forms.Label lblLoop1;
26       /// <summary>
27       /// Required designer variable.
28       /// </summary>
29       private System.ComponentModel.Container components = null;
30
31       public FrmLoopTest()
32       {
33          //
34          // Required for Windows Form Designer support
35          //
36          InitializeComponent();
37          //
38          // TODO: Add any constructor code after InitializeComponent
39          // call
40          //
41       }
42
43       /// <summary>
44       /// Clean up any resources being used.
45       /// </summary>
46       protected override void Dispose( bool disposing )
47       {
48          if( disposing )
49          {
50             if (components != null)
51             {
52                components.Dispose();
53             }
54          }
55          base.Dispose( disposing );
56       }
57
58       // Windows Form Designer generated code
59
60       /// <summary>
```

```
61          /// The main entry point for the application.
62          /// </summary>
63          [STAThread]
64          static void Main()
65          {
66             Application.Run(new FrmLoopTest());
67          }
68
69          //  performs several calculations using loops
70          private void FrmLoopTest_Load(
71             object sender, System.EventArgs e)
72          {
73             // code for loop 1
74             int intTotal = 0;
75             int intX = 1;
76
77             while ( intX <= 10 )
78             {
79                intTotal += intX;
80                intX++;
81             }
82
83             // display the result
84             lblLoop1Out.Text = Convert.ToString( intTotal );
85
86             // code for loop 2
87             int intCounter = 1;
88
89             while ( intCounter <= 100 )
90             {
91                intTotal += intCounter;
92                intCounter++;
93             }
94
95             // display the result
96             lblLoop2Out.Text = Convert.ToString( intTotal );
97
98             // code for loop 3
99             intCounter = 1000;
100
101            while ( intCounter > 0 )
102            {
103               lblDisplay.Text = Convert.ToString( intCounter );
104               intCounter--;
105            }
106
107            // code for loop 4
108            intCounter = 1;
109
110            while ( intCounter <= 5 )
111            {
112               lstNumbers.Items.Add( intCounter );
113               intCounter++;
114            }
115
116         } // end method FrmLoopTest_Load
117
118      } // end class FrmLoopTest
119   }
```



Using the Debugger

9.16 (*Odd Numbers Application*) The **Odd Numbers** application should display all of the odd integers between one and the number input by the user. Copy the directory `C:\Examples\Tutorial09\Exercises\Debugger\OddNumbers` to your `C:\SimplyCSP` directory. Run the application. Notice that, after you enter a value into the **Upper limit:** `TextBox` and click the **View** `Button`, an infinite loop occurs (your application will not respond). Use the debugger to find and fix the error(s) in the application. Figure 9.24 displays the correct output for the application.

Figure 9.24 Correct output for the **Odd Numbers** application.

Answer:

```
1  // Exercise 9.16 Solution
2  // OddNumbers.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace OddNumbers
12 {
13    /// <summary>
14    /// Summary description for FrmOddNumbers.
15    /// </summary>
16    public class FrmOddNumbers : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input upper limit of odd numbers
19       private System.Windows.Forms.Label lblLimit;
20       private System.Windows.Forms.TextBox txtLimit;
21
22       // Button to view the list of odd numbers
23       private System.Windows.Forms.Button btnView;
24
25       // ListBox to list the odd numbers
26       private System.Windows.Forms.ListBox lstResults;
27
28       /// <summary>
```

```
29          /// Required designer variable.
30          /// </summary>
31          private System.ComponentModel.Container components = null;
32
33          public FrmOddNumbers()
34          {
35             //
36             // Required for Windows Form Designer support
37             //
38             InitializeComponent();
39             //
40             // TODO: Add any constructor code after InitializeComponent
41             // call
42             //
43          }
44
45          /// <summary>
46          /// Clean up any resources being used.
47          /// </summary>
48          protected override void Dispose( bool disposing )
49          {
50             if( disposing )
51             {
52                if (components != null)
53                {
54                   components.Dispose();
55                }
56             }
57             base.Dispose( disposing );
58          }
59
60          // Windows Form Designer generated code
61
62          /// <summary>
63          /// The main entry point for the application.
64          /// </summary>
65          [STAThread]
66          static void Main()
67          {
68             Application.Run( new FrmOddNumbers() );
69          }
70
71          // handles Click event
72          private void btnView_Click(
73             object sender, System.EventArgs e )
74          {
75             int intLimit = 0; // upper limit set by user
76             int intCounter = 1; // counter begins at 1
77
78             lstResults.Items.Clear(); // clear ListBox
79             intLimit =
80                Int32.Parse( txtLimit.Text ); // retrieve upper limit
81             lstResults.Items.Add( "Odd numbers:" ); // display header
82
83             while ( intCounter < intLimit )
84             {
85                // determine and display odd numbers
86                if ( intCounter % 2 != 0 )
87                {
88                   lstResults.Items.Add( intCounter );
```

Incorrect code given to students incremented `intLimit` instead of `intCounter`

```
89              }
90
91              intCounter++; // increment counter
92           }
93
94        } // end method btnView_Click
95
96     } // end class FrmOddNumbers
97  }
```

Programming Challenge

9.17 ***(To Do List Application)*** Use a `ListBox` as a to do list. Enter each item in a `TextBox`, and add it to the `ListBox` by clicking a `Button`. The item should be displayed in a numbered list as in Fig. 9.25. To do this, we introduce the `Count` method, which returns the number of items in a `ListBox`'s `Items` property. The following is a sample call to assign the number of items displayed in the `lstSample ListBox` to an `int` variable:

```
intCount = lstSample.Items.Count();
```

Figure 9.25 To Do List application.

Answer:

```
1  // Exercise 9.17 Solution
2  // ToDoList.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ToDoList
12 {
13    /// <summary>
14    /// Summary description for FrmToDoList.
15    /// </summary>
16    public class FrmToDoList : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input an activity
19       private System.Windows.Forms.Label lblInput;
20       private System.Windows.Forms.TextBox txtInput;
21
22       // Button to add an activity to the to do list
23       private System.Windows.Forms.Button btnAdd;
24
25       // ListBox to list the to do list
26       private System.Windows.Forms.ListBox lstOutput;
27
28       /// <summary>
```

```
29          /// Required designer variable.
30          /// </summary>
31          private System.ComponentModel.Container components = null;
32
33          public FrmToDoList()
34          {
35             //
36             // Required for Windows Form Designer support
37             //
38             InitializeComponent();
39             //
40             // TODO: Add any constructor code after InitializeComponent
41             // call
42             //
43          }
44
45          /// <summary>
46          /// Clean up any resources being used.
47          /// </summary>
48          protected override void Dispose( bool disposing )
49          {
50             if( disposing )
51             {
52                if (components != null)
53                {
54                   components.Dispose();
55                }
56             }
57             base.Dispose( disposing );
58          }
59
60          // Windows Form Designer generated code
61
62          /// <summary>
63          /// The main entry point for the application.
64          /// </summary>
65          [STAThread]
66          static void Main()
67          {
68             Application.Run( new FrmToDoList() );
69          }
70
71          // handles Click event
72          private void btnAdd_Click(
73             object sender, System.EventArgs e )
74          {
75             int intItemNumber;
76
77             // set number of item
78             intItemNumber = lstOutput.Items.Count + 1;
79
80             // add input with number to ListBox
81             lstOutput.Items.Add( intItemNumber + ". " +
82                txtInput.Text );
83
84             // clear TextBox
85             txtInput.Text = "";
86
87          } // end method btnAdd_Click
88
```

```
89      } // end class FrmToDoList
90  }
```

10 TUTORIAL

Class Average Application

Introducing the do...while Repetition Statement

Solutions

Instructor's Manual Exercise Solutions Tutorial 10

MULTIPLE-CHOICE QUESTIONS

10.1 A(n) _____ occurs when a loop-continuation condition in a `while` loop never becomes `false`.

a) infinite loop
b) counter-controlled loop
c) control statement
d) nested control statement

10.2 Set the _____ property to `true` to enable a `Button`.

a) `Disabled`
b) `Focus`
c) `Enabled`
d) `ButtonEnabled`

10.3 The _____ statement executes at least once and continues executing until its loop-continuation condition becomes `false`.

a) `do...loop`
b) `while`
c) `do...while`
d) `loop...while`

10.4 Which of these loop conditions describes the loop "repeat until `i` is greater than five"?

a) `do...until ( i > 5 )`
b) `while ( i <= 5 )`
c) `while ( i < 5 )`
d) `do...while ( i == 5 )`

10.5 The _____ method transfers the focus to a control.

a) `GetFocus`
b) `Focus`
c) `Transfer`
d) `Activate`

10.6 A _____ contains the sum of a series of values.

a) total
b) counter
c) condition
d) loop

10.7 The _____ property of _____ property contains the number of items in a `ListBox`.

a) `Count`, `ListBox`
b) `ListCount`, `Items`
c) `ListCount`, `ListBox`
d) `Count`, `Items`

10.8 A(n) _____ occurs when a loop executes for one more or one less iteration than is necessary.

a) infinite loop
b) counter-controlled loop
c) off-by-one error
d) nested control statement

10.9 When a GUI element (such as a `Button`) should no longer be accessible by the user, it should be _____.

a) deleted
b) hidden
c) disabled
d) All of the above.

10.10 If its loop-continuation condition is initially `false`, a `do...while` repetition statement _____.

a) never executes
b) executes while the condition is `false`
c) executes until the condition becomes `true`
d) executes only once

Answers: 10.1) a. 10.2) c. 10.3) c. 10.4) b. 10.5) b. 10.6) a. 10.7) d. 10.8) c. 10.9) c. 10.10) d.

EXERCISES

10.11 ***(Modified Class Average Application)*** Modify the **Class Average** application, as in Fig. 10.17, so that **Average** `Button` is disabled until 10 grades have been entered.

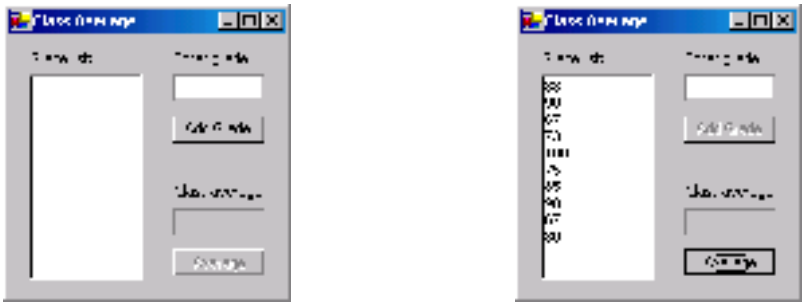


Figure 10.17 Modified **Class Average** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial10\Exercises\ModifiedClassAverage` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ClassAverage.sln` in the `ModifiedClassAverage` directory to open the application.

c) ***Initially disabling the Average `Button`.*** Use the **Properties** window to modify the **Average** `Button` in the `Form` so that it is disabled when the application first runs by setting its `Enabled` property to `false`.

d) ***Enabling the Average `Button` after 10 grades have been entered.*** Add code to the `btnAdd_Click` event handler so that the **Average** `Button` becomes enabled when 10 grades have been entered.

e) ***Disabling the Average `Button` after the calculation has been performed.*** Add code to the `btnAverage_Click` event handler so that the **Average** `Button` is disabled once the calculation result has been displayed.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter 10 grades and ensure that the **Average** `Button` is disabled until all 10 grades are entered. Verify that the **Add Grade** `Button` is disabled after 10 grades are entered. Once the **Average** `Button` is enabled, click it and verify that the average displayed is correct. The **Average** `Button` should then become disabled again, and the **Add Grade** `Button` should be enabled.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 10.11 Solution
2  // ClassAverage.cs (Modified)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ClassAverage
12 {
13    /// <summary>
14    /// Summary description for FrmClassAverage.
15    /// </summary>
16    public class FrmClassAverage : System.Windows.Forms.Form
17    {
18       // Label and ListBox to display grades
19       private System.Windows.Forms.Label lblGradeList;
20       private System.Windows.Forms.ListBox lstGrades;
21
22       // Label, TextBox and Button to add a grade
23       private System.Windows.Forms.Label lblPrompt;
```

```
24          private System.Windows.Forms.TextBox txtInput;
25          private System.Windows.Forms.Button btnAdd;
26
27          // Labels to display the average grade
28          private System.Windows.Forms.Label lblDescribeOutput;
29          private System.Windows.Forms.Label lblOutput;
30
31          // Button to compute the average grade
32          private System.Windows.Forms.Button btnAverage;
33
34          /// <summary>
35          /// Required designer variable.
36          /// </summary>
37          private System.ComponentModel.Container components = null;
38
39          public FrmClassAverage()
40          {
41             //
42             // Required for Windows Form Designer support
43             //
44             InitializeComponent();
45             //
46             // TODO: Add any constructor code after InitializeComponent
47             // call
48             //
49          }
50
51          /// <summary>
52          /// Clean up any resources being used.
53          /// </summary>
54          protected override void Dispose( bool disposing )
55          {
56             if( disposing )
57             {
58                if (components != null)
59                {
60                   components.Dispose();
61                }
62             }
63             base.Dispose( disposing );
64          }
65
66          // Windows Form Designer generated code
67
68          /// <summary>
69          /// The main entry point for the application.
70          /// </summary>
71          [STAThread]
72          static void Main()
73          {
74             Application.Run( new FrmClassAverage() );
75          }
76
77          // handles Add Grade Button's Click event
78          private void btnAdd_Click(
79             object sender, System.EventArgs e )
80          {
81             // clear previous grades and calculation result
82             if ( lblOutput.Text != "" )
83             {
```

```
84             lblOutput.Text = "";
85             lstGrades.Items.Clear();
86          }
87
88          // display grade in ListBox
89          lstGrades.Items.Add( Int32.Parse( txtInput.Text ) );
90          txtInput.Clear(); // clear grade from TextBox
91          txtInput.Focus(); // transfer focus to TextBox
92
93          // prohibit users from entering more than 10 grades
94          if ( lstGrades.Items.Count == 10 )
95          {
96             btnAdd.Enabled = false; // disable Add Grade Button
97             btnAverage.Enabled = true; // enabled Average Button
98             btnAverage.Focus(); // transfer focus to Average Button
99          }
100
101       } // end method btnAdd_Click
102
103       // handles Average Button's Click event
104       private void btnAverage_Click(
105          object sender, System.EventArgs e )
106       {
107          // initialization phase
108          int intTotal = 0;
109          int intGradeCounter = 0;
110          int intGrade = 0;
111          double dblAverage = 0;
112
113          // sum grades in ListBox
114          do
115          {
116             // read grade from ListBox
117             intGrade = ( int )
118                lstGrades.Items[ intGradeCounter ];
119             intTotal += intGrade; // add grade to total
120             intGradeCounter++; // increment counter
121          } while ( intGradeCounter < 10 );
122
123          dblAverage = intTotal / 10.0; // calculate average
124          lblOutput.Text = String.Format( "{0:F}", dblAverage );
125          btnAdd.Enabled = true; // enable Add Grade Button
126          btnAverage.Enabled = false; // disable Average Button
127          txtInput.Focus(); // reset focus to Enter grade: TextBox
128
129       } // end method btnAverage_Click
130
131    } // end class FrmClassAverage
132 }
```

10.12 ***(Undetermined Class Average Application That Handles Any Number of Grades)*** Rewrite the **Class Average** application to handle any number of grades, as in Fig. 10.18. Note that, because the application does not know how many grades the user will enter, the `Button`s must be enabled at all times.

Figure 10.18 Undetermined **Class Average** application handling an unspecified number of grades.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial10\Exercises\UndeterminedClassAverage` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ClassAverage.sln` in the `UndeterminedClassAverage` directory to open the application.

c) ***Never disabling the Add Grade `Button`.*** Remove code from the `btnAdd_Click` event handler so that the **Add Grade** `Button` is not disabled after entering 10 grades.

d) ***Summing the grades in the `ListBox`.*** Modify code in the `btnAverage_Click` event handler so that `intGradeCounter` is incremented until it is equal to the number of grades entered. Use `lstGrades.Items.Count` to determine the number of items in the `ListBox`. The number returned by the `Count` property will be zero if there are no grades entered. Use an `if` selection statement to avoid division by zero and display a message dialog to the user if there are no grades entered when the user clicks the **Average** `Button`.

e) ***Calculating the class average.*** Modify the code in the `btnAverage_Click` event handler so that `dblAverage` is computed by using `intGradeCounter` rather than the value 10.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter 10 grades and click the Average Button. Verify that the average displayed is correct. Follow the same actions but this time for 15 grades, then for 5 grades. Each time, verify that the appropriate average is displayed.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 10.12 Solution
2  // ClassAverage.cs (Undetermined)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ClassAverage
12 {
13    /// <summary>
14    /// Summary description for FrmClassAverage.
15    /// </summary>
16    public class FrmClassAverage : System.Windows.Forms.Form
17    {
18       // Label and ListBox to display grades
19       private System.Windows.Forms.Label lblGradeList;
20       private System.Windows.Forms.ListBox lstGrades;
```

```
21
22          // Label, TextBox, and Button to add a grade
23          private System.Windows.Forms.Label lblPrompt;
24          private System.Windows.Forms.TextBox txtInput;
25          private System.Windows.Forms.Button btnAdd;
26
27          // Labels to display the average grade
28          private System.Windows.Forms.Label lblDescribeOutput;
29          private System.Windows.Forms.Label lblOutput;
30
31          // Button to compute the average grade
32          private System.Windows.Forms.Button btnAverage;
33
34          /// <summary>
35          /// Required designer variable.
36          /// </summary>
37          private System.ComponentModel.Container components = null;
38
39          public FrmClassAverage()
40          {
41             //
42             // Required for Windows Form Designer support
43             //
44             InitializeComponent();
45             //
46             // TODO: Add any constructor code after InitializeComponent
47             // call
48             //
49          }
50
51          /// <summary>
52          /// Clean up any resources being used.
53          /// </summary>
54          protected override void Dispose( bool disposing )
55          {
56             if( disposing )
57             {
58                if (components != null)
59                {
60                   components.Dispose();
61                }
62             }
63             base.Dispose( disposing );
64          }
65
66          // Windows Form Designer generated code
67
68          /// <summary>
69          /// The main entry point for the application.
70          /// </summary>
71          [STAThread]
72          static void Main()
73          {
74             Application.Run( new FrmClassAverage() );
75          }
76
77          // handles Add Grade Button's Click event
78          private void btnAdd_Click(
79             object sender, System.EventArgs e )
80          {
```

```
81          // clear previous grades and calculation result
82          if ( lblOutput.Text != "" )
83          {
84             lblOutput.Text = "";
85             lstGrades.Items.Clear();
86          }
87
88          // display grade in ListBox
89          lstGrades.Items.Add( Int32.Parse( txtInput.Text ) );
90          txtInput.Clear(); // clear grade from TextBox
91          txtInput.Focus(); // transfer focus to TextBox
92
93       } // end method btnAdd_Click
94
95       // handles Average Button's Click event
96       private void btnAverage_Click(
97          object sender, System.EventArgs e )
98       {
99          // initialization phase
100         int intTotal = 0;
101         int intGradeCounter = 0;
102         int intGrade = 0;
103         double dblAverage = 0;
104
105         // no grades entered
106         if ( lstGrades.Items.Count == 0 )
107         {
108            MessageBox.Show( "Please enter at least one grade",
109               "Enter Grade", MessageBoxButtons.OK,
110               MessageBoxIcon.Exclamation );
111         }
112         else
113         {
114            // sum grades in ListBox
115            do
116            {
117               // read grade from ListBox
118               intGrade = ( int )
119                  lstGrades.Items[ intGradeCounter ];
120               intTotal += intGrade; // add grade to total
121               intGradeCounter++; // increment counter
122            } while ( intGradeCounter < lstGrades.Items.Count );
123
124            // calculate average
125            dblAverage = ( double ) intTotal / intGradeCounter;
126            lblOutput.Text = String.Format( "{0:F}", dblAverage );
127            txtInput.Focus(); // reset focus to Enter grade: TextBox
128         }
129
130      } // end method btnAverage_Click
131
132   } // end class FrmClassAverage
133 }
```

10.13 ***(Arithmetic Calculator Application)*** Write an application that allows users to enter a series of numbers and manipulate them. The application should provide users with the option of adding or multiplying the numbers. Users should enter each number in a `TextBox`. After entering each number, users should click a `Button`, then the number should be inserted in a `ListBox`. The GUI should appear as shown in Fig. 10.19.

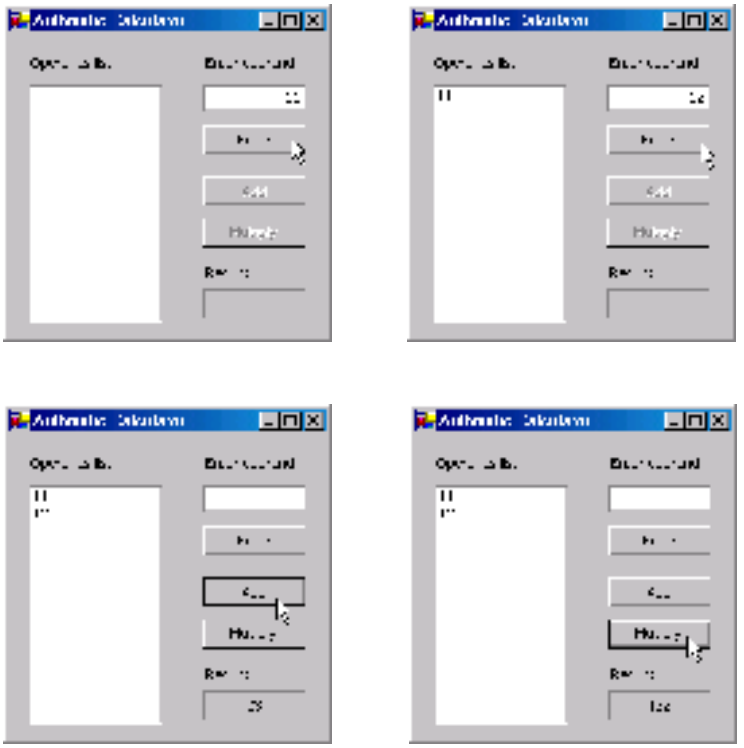


Figure 10.19 Arithmetic Calculator application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial10\Exercises\ArithmeticCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ArithmeticCalculator.sln` in the `ArithmeticCalculator` directory to open the application.

c) ***Add a `ListBox` to display the entered numbers.*** Add a `ListBox`. Place and size it as in Fig. 10.19. Rearrange and comment the control declarations appropriately.

d) ***Creating an event handler for the Enter `Button`.*** Create the `Click` event handler for the **Enter** `Button`. If the result of a previous calculation is displayed, this event handler should clear the result and disable the addition and multiplication `Buttons`. It should then insert the current number in the **Operands list:** `ListBox`. When the `ListBox` contains at least two numbers, the event handler should then enable the addition and multiplication `Buttons`.

e) ***Summing the grades in the `ListBox`.*** Define the `Click` event handler for the **Add** `Button`. This event handler should compute the sum of all of the values in the **Operands list:** `ListBox` and display the result in a `Label`, `lblResult`. Use the square bracket notation to access each item in the `ListBox`. Use the cast operator to convert the `ListBox` items into `doubles`.

f) ***Define the `Click` event handler for the Multiply `Button`.*** This event handler should compute the product of all of the values in the **Operands list:** `ListBox` and display the result in the `lblResult` `Label`. Use the square bracket notation to access each item in the `ListBox`. Use the cast operator to convert the `ListBox` items into `doubles`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter two values, then click the **Add** and **Multiply** `Buttons`. Verify that the results displayed are correct. Also, make sure that the **Add** and **Multiply** `Buttons` are not enabled until two values have been entered.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 10.13 Solution
2  // ArithmeticCalculator.cs
```

```
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ArithmeticCalculator
12 {
13    /// <summary>
14    /// Summary description for FrmArithmeticCalculator.
15    /// </summary>
16    public class FrmArithmeticCalculator : System.Windows.Forms.Form
17    {
18       // Label and ListBox to display list of operands
19       private System.Windows.Forms.Label lblOperands;
20       private System.Windows.Forms.ListBox lstNumbers;
21
22       // Label and TextBox to input an operand
23       private System.Windows.Forms.Label lblInput;
24       private System.Windows.Forms.TextBox txtInput;
25
26       // Button to enter an operand to the list
27       private System.Windows.Forms.Button btnEnter;
28
29       // Buttons to add or multiply the operands together
30       private System.Windows.Forms.Button btnAdd;
31       private System.Windows.Forms.Button btnMultiply;
32
33       // Labels to display result of calculation
34       private System.Windows.Forms.Label lblOutput;
35       private System.Windows.Forms.Label lblResult;
36
37       /// <summary>
38       /// Required designer variable.
39       /// </summary>
40       private System.ComponentModel.Container components = null;
41
42       public FrmArithmeticCalculator()
43       {
44          //
45          // Required for Windows Form Designer support
46          //
47          InitializeComponent();
48          //
49          // TODO: Add any constructor code after InitializeComponent
50          // call
51          //
52       }
53
54       /// <summary>
55       /// Clean up any resources being used.
56       /// </summary>
57       protected override void Dispose( bool disposing )
58       {
59          if( disposing )
60          {
61             if (components != null)
62             {
```

```
63                components.Dispose();
64             }
65          }
66          base.Dispose( disposing );
67       }
68
69       // Windows Form Designer generated code
70
71       /// <summary>
72       /// The main entry point for the application.
73       /// </summary>
74       [STAThread]
75       static void Main()
76       {
77          Application.Run( new FrmArithmeticCalculator() );
78       }
79
80       // handles Enter Button's Click event
81       private void btnEnter_Click(
82          object sender, System.EventArgs e )
83       {
84          // clear ListBox and lblResult if necessary
85          if ( lblResult.Text != "" )
86          {
87             lblResult.Text = "";
88             lstNumbers.Items.Clear();
89             btnAdd.Enabled = false; // disable operation Buttons
90             btnMultiply.Enabled = false;
91          }
92
93          // add number to ListBox
94          lstNumbers.Items.Add( Double.Parse( txtInput.Text ) );
95
96          // enable binary operation Buttons when
97          // user has entered two numbers
98          if ( lstNumbers.Items.Count == 2 )
99          {
100            btnAdd.Enabled = true;
101            btnMultiply.Enabled = true;
102         }
103
104         txtInput.Clear(); // clear TextBox
105         txtInput.Focus(); // set focus to TextBox
106
107      } // end method btnEnter_Click
108
109      // handles Add Button's Click event
110      private void btnAdd_Click(
111         object sender, System.EventArgs e )
112      {
113         // initialize total and counter
114         double dblTotal = 0;
115         int intCounter = 0;
116
117         // sum numbers in ListBox
118         do
119         {
120            dblTotal += ( double )
121               lstNumbers.Items[ intCounter ];
122            intCounter++;
```

```
123          } while ( intCounter < lstNumbers.Items.Count );
124
125          lblResult.Text = Convert.ToString( dblTotal );
126
127       } // end method btnAdd_Click
128
129       // handles Multiply Button's Click event
130       private void btnMultiply_Click(
131          object sender, System.EventArgs e )
132       {
133          // initialize total and counter
134          double dblTotal = 1;
135          int intCounter = 0;
136
137          // sum numbers in ListBox
138          do
139          {
140             dblTotal *= ( double )
141                lstNumbers.Items[ intCounter ];
142             intCounter++;
143          } while ( intCounter < lstNumbers.Items.Count );
144
145          lblResult.Text = Convert.ToString( dblTotal );
146
147       } // end method btnMultiply_Click
148
149    } // end class FrmArithmeticCalculator
150 }
```

What does this code do?

10.14 What is the result of the following code?

```
1  int intY;
2  int intX;
3  int intMysteryValue;
4
5  intX = 1;
6  intMysteryValue = 0;
7
8  do
9  {
10    intY = ( int ) Math.Pow( intX, 2 );
11    lstDisplay.Items.Add( intY );
12    intMysteryValue++;
13    intX++;
14
15 } while ( intX <= 10 );
16
17 lblResult.Text = Convert.ToString( intMysteryValue );
```

Answer: The value displayed in `lblResult` is 10. The code also displays the squares of the first ten integers in `lstDisplay`. The complete code reads:

```
1  // Exercise 10.14 Solution
2  // Calculate.cs
3
4  using System;
```

```
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Calculate
{
   /// <summary>
   /// Summary description for FrmCalculate.
   /// </summary>
   public class FrmCalculate : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblValues;
      private System.Windows.Forms.ListBox lstDisplay;
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label Label1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCalculate()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmCalculate() );
      }

```

```
65        // calculates the squares of the first 10 positive integers
66        private void FrmCalculate_Load(
67           object sender, System.EventArgs e)
68        {
69           int intY;
70           int intX;
71           int intMysteryValue;
72
73           intX = 1;
74           intMysteryValue = 0;
75
76           do
77           {
78              intY = ( int ) Math.Pow( intX, 2 );
79              lstDisplay.Items.Add( intY );
80              intMysteryValue++;
81              intX++;
82
83           } while ( intX <= 10 );
84
85           lblResult.Text = Convert.ToString( intMysteryValue );
86
87        } // end method FrmCalculate_Load
88
89     } // end class FrmCalculate
90  }
```

What's wrong with this code? ▶

10.15 Find the error(s) in the following code. This code should add 10 to the value in `intY` and store it in `intZ`. It then should reduce the value of `intY` by one and repeat until `intY` is less than 10. The output `Label` `lblResult` should display the final value of `intZ`.

```
1  int intY = 10;
2  int intZ = 2;
3
4  do
5  {
6     intZ = intY + 10;
7
8  } while ( !( intY < 10 ) );
9
10 intY--;
11
12 lblResult.Text = Convert.ToString( intZ );
```

Answer: This code will loop infinitely because the statement that decrements `intY` (line 10) is not inside the repetition statement. The complete incorrect code reads:

```
1  // Exercise 10.15 Solution
2  // LoopTest.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace LoopTest
12 {
13    /// <summary>
14    /// Summary description for FrmLoopTest.
15    /// </summary>
16    public class FrmLoopTest : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblResult;
19       private System.Windows.Forms.Label lblResultText;
20       /// <summary>
21       /// Required designer variable.
22       /// </summary>
23       private System.ComponentModel.Container components = null;
24
25       public FrmLoopTest()
26       {
27          //
28          // Required for Windows Form Designer support
29          //
30          InitializeComponent();
31          //
32          // TODO: Add any constructor code after InitializeComponent
33          // call
34          //
35       }
36
37       /// <summary>
38       /// Clean up any resources being used.
39       /// </summary>
40       protected override void Dispose( bool disposing )
41       {
42          if( disposing )
43          {
44             if (components != null)
45             {
46                components.Dispose();
47             }
48          }
49          base.Dispose( disposing );
50       }
51
52       // Windows Form Designer generated code
53
54       /// <summary>
55       /// The main entry point for the application.
56       /// </summary>
57       [STAThread]
58       static void Main()
59       {
```

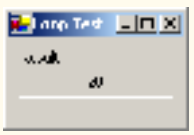
```
60             Application.Run( new FrmLoopTest() );
61          }
62
63          // performs several calculations using loops
64          private void FrmLoopTest_Load(
65             object sender, System.EventArgs e )
66          {
67             int intY = 10;
68             int intZ = 2;
69
70             do
71             {
72                intZ = intY + 10;
73
74             } while ( !( intY < 10 ) );
75
76             intY--;
77
78             lblResult.Text = Convert.ToString( intZ );
79
80          } // end method FrmLoopTest_Load
81
82       } // end class FrmLoopTest
83    }
```

intY must be decremented inside the loop statement

Answer: The complete correct code should read:

```
1   // Exercise 10.15 Solution
2   // LoopTest.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace LoopTest
12  {
13     /// <summary>
14     /// Summary description for FrmLoopTest.
15     /// </summary>
16     public class FrmLoopTest : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Label lblResult;
19        private System.Windows.Forms.Label lblResultText;
20        /// <summary>
21        /// Required designer variable.
22        /// </summary>
23        private System.ComponentModel.Container components = null;
24
25        public FrmLoopTest()
26        {
27           //
28           // Required for Windows Form Designer support
29           //
30           InitializeComponent();
31           //
32           // TODO: Add any constructor code after InitializeComponent
```

```
33            // call
34            //
35         }
36
37         /// <summary>
38         /// Clean up any resources being used.
39         /// </summary>
40         protected override void Dispose( bool disposing )
41         {
42            if( disposing )
43            {
44               if (components != null)
45               {
46                  components.Dispose();
47               }
48            }
49            base.Dispose( disposing );
50         }
51
52         // Windows Form Designer generated code
53
54         /// <summary>
55         /// The main entry point for the application.
56         /// </summary>
57         [STAThread]
58         static void Main()
59         {
60            Application.Run( new FrmLoopTest() );
61         }
62
63         // performs several calculations in a loop
64         private void FrmLoopTest_Load(
65            object sender, System.EventArgs e )
66         {
67            int intY = 10;
68            int intZ = 2;
69
70            do
71            {
72               intZ = intY + 10;
73               intY--;
74
75            } while ( !( intY < 10 ) );
76
77            lblResult.Text = Convert.ToString( intZ );
78
79         } // end method FrmLoopTest_Load
80
81      } // end class FrmLoopTest
82   }
```



Using the Debugger ▶

10.16 (***Factorial Application***) The **Factorial** application calculates the factorial of an integer input by the user. The factorial of an integer is the product of the integers from one to that number. For example, the factorial of 3 is 6 ($1 \times 2 \times 3$). Copy the directory `C:\Examples\Tutorial10\Debugger\Factorial` to your `C:\SimplyCSP` directory. Run the application. While testing the application you noticed that it does not execute correctly. Use the debugger to find and correct the logic error(s) in the application. Figure 10.20 displays the correct output for the **Factorial** application.

Figure 10.20 Correct output for the **Factorial** application.

Answer:

```
// Exercise 10.16 Solution
// Factorial.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Factorial
{
   /// <summary>
   /// Summary description for FrmFactorial.
   /// </summary>
   public class FrmFactorial : System.Windows.Forms.Form
   {
      // Label and TextBox to enter the number
      private System.Windows.Forms.Label lblNumber;
      private System.Windows.Forms.TextBox txtNumber;

      // Labels to display the factorial of the number
      private System.Windows.Forms.Label lblFactorial;
      private System.Windows.Forms.Label lblResult;

      // Button to calculate the factorial
      private System.Windows.Forms.Button btnCalculate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmFactorial()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
```

```
42            // call
43            //
44         }
45
46         /// <summary>
47         /// Clean up any resources being used.
48         /// </summary>
49         protected override void Dispose( bool disposing )
50         {
51            if( disposing )
52            {
53               if (components != null)
54               {
55                  components.Dispose();
56               }
57            }
58            base.Dispose( disposing );
59         }
60
61         // Windows Form Designer generated code
62
63         /// <summary>
64         /// The main entry point for the application.
65         /// </summary>
66         [STAThread]
67         static void Main()
68         {
69            Application.Run( new FrmFactorial() );
70         }
71
72         // handles Click event
73         private void btnCalculate_Click(
74            object sender, System.EventArgs e )
75         {
76            int intInput; // user input
77            int intFactorial = 1; // holds factorial
78
79            intInput = Int32.Parse( txtNumber.Text ); // get user input
80
81            // loop until intInput equals zero
82            do
83            {
84               intFactorial *= intInput; // calculate factorial
85               intInput--; // decrement counter
86            } while ( intInput > 1 ); // test guard condition
87
88            // display factorial
89            lblResult.Text = Convert.ToString( intFactorial );
90
91         } // end method btnCalculate_Click
92
93      } // end class FrmFactorial
94   }
```

Replaced <= with >

Programming Challenge

10.17 (***Restaurant Bill Application***) Develop an application that calculates a restaurant bill. The user should be able to enter the item ordered, the quantity of the item ordered and the price per item. When the user clicks the **Add Item** `Button`, your application should display the number ordered, the item ordered and the price per unit in three `ListBox`es as shown in Fig. 10.21. When the user clicks the **Total Bill** `Button`, the application should calculate the

total cost. For each entry in the ListBox, multiply the cost of each item by the number of items ordered. Use the square bracket notation to access each item in a ListBox.

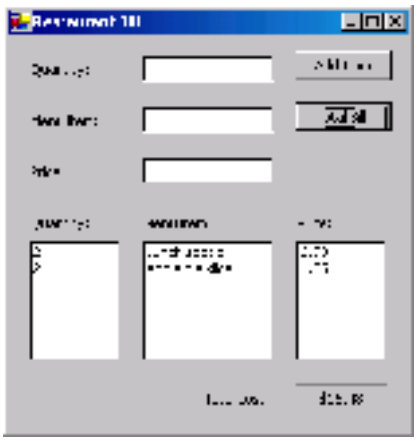


Figure 10.21 **Restaurant Bill** application.

Answer:

```
1  // Exercise 10.17 Solution
2  // RestaurantBill.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace RestaurantBill
12 {
13    /// <summary>
14    /// Summary description for FrmRestaurantBill.
15    /// </summary>
16    public class FrmRestaurantBill : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input quantity of an item
19       private System.Windows.Forms.Label lblQuantity;
20       private System.Windows.Forms.TextBox txtQuantity;
21
22       // Label and TextBox to input an item
23       private System.Windows.Forms.Label lblItem;
24       private System.Windows.Forms.TextBox txtItem;
25
26       // Label and TextBox to input price of an item
27       private System.Windows.Forms.Label lblPrice;
28       private System.Windows.Forms.TextBox txtPrice;
29
30       // Button to add an item to the bill
31       private System.Windows.Forms.Button btnAddItem;
32
33       // Button to compute the total bill
34       private System.Windows.Forms.Button btnTotal;
35
36       // Label and ListBox to display the list of quantities
37       // of items
38       private System.Windows.Forms.Label lblQuantityList;
39       private System.Windows.Forms.ListBox lstQuantity;
```

```
40
41          // Label and ListBox to display the list of items
42          private System.Windows.Forms.Label lblMenuItemList;
43          private System.Windows.Forms.ListBox lstItem;
44
45          // Label and ListBox to display the list of prices
46          // of items
47          private System.Windows.Forms.Label lblPriceList;
48          private System.Windows.Forms.ListBox lstPrice;
49
50          // Labels to display the total bill
51          private System.Windows.Forms.Label lblTotal;
52          private System.Windows.Forms.Label lblTotalCost;
53
54          /// <summary>
55          /// Required designer variable.
56          /// </summary>
57          private System.ComponentModel.Container components = null;
58
59          public FrmRestaurantBill()
60          {
61             //
62             // Required for Windows Form Designer support
63             //
64             InitializeComponent();
65             //
66             // TODO: Add any constructor code after InitializeComponent
67             // call
68             //
69          }
70
71          /// <summary>
72          /// Clean up any resources being used.
73          /// </summary>
74          protected override void Dispose( bool disposing )
75          {
76             if( disposing )
77             {
78                if (components != null)
79                {
80                   components.Dispose();
81                }
82             }
83             base.Dispose( disposing );
84          }
85
86          // Windows Form Designer generated code
87
88          /// <summary>
89          /// The main entry point for the application.
90          /// </summary>
91          [STAThread]
92          static void Main()
93          {
94             Application.Run( new FrmRestaurantBill() );
95          }
96
97          // handles Add Item Button's Click event
98          private void btnAddItem_Click(
99             object sender, System.EventArgs e )
```

```
100       {
101          // display user input in ListBoxes
102          lstQuantity.Items.Add( Int32.Parse( txtQuantity.Text ) );
103          lstItem.Items.Add( txtItem.Text );
104          lstPrice.Items.Add( Decimal.Parse( txtPrice.Text ) );
105
106          // clear TextBoxes
107          txtItem.Clear();
108          txtQuantity.Clear();
109          txtPrice.Clear();
110
111          txtQuantity.Focus(); // set the focus to Quantity: TextBox
112
113       } // end method btnAddItem_Click
114
115       // handles Total Bill Button's Click event
116       private void btnTotal_Click(
117          object sender, System.EventArgs e )
118       {
119          int intCounter = 0;
120          decimal decCost = 0;
121
122          // calculate bill
123          do
124          {
125             decCost += ( decimal ) lstPrice.Items[ intCounter ]
126                * ( int ) lstQuantity.Items[ intCounter ];
127             intCounter++;
128          } while ( intCounter < lstPrice.Items.Count );
129
130          // display result
131          lblTotalCost.Text = String.Format( "{0:C}", decCost );
132
133       } // end method btnTotal_Click
134
135    } // end class FrmRestaurantBill
136 }
```

TUTORIAL 11

Interest Calculator Application

Introducing the for Repetition Statement

Solutions

Instructor's Manual Exercises Solutions Tutorial 11

MULTIPLE-CHOICE QUESTIONS

11.1 "Hello" has a _____ type.

a) `string`
b) `stringliteral`
c) `char`
d) `stringtext`

11.2 A _____ provides the ability to enter or display multiple lines of text in the same control.

a) `TextBox`
b) `NumericUpDown`
c) multiline `TextBox`
d) multiline `NumericUpDown`

11.3 The `NumericUpDown` control allows you to specify _____.

a) a maximum value the user can select
b) a minimum value the user can select
c) an increment for the values presented to the user
d) All of the above.

11.4 The _____ is often omitted from a `for` header when the control variable has already been assigned a value.

a) semicolon
b) initial value of the control variable
c) `for` keyword
d) final value of the control variable

11.5 Setting `TextBox` property `ScrollBars` to _____ creates a vertical scrollbar.

a) `true`
b) `Vertical`
c) `Up`
d) `Down`

11.6 _____ is used to determine whether a `for` loop continues to iterate.

a) The initial value of the control variable
b) The `for` keyword
c) The increment value
d) The loop-continuation condition

11.7 In a `for` loop, the control variable is incremented (or decremented) _____.

a) after the body of the loop executes
b) the first time through only
c) while the loop-continuation condition is `false`
d) while the body of the loop executes

11.8 Setting a `NumericUpDown` control's _____ property to `true` ensures that the user cannot enter invalid values in the control.

a) `Increment`
b) `ScrollBars`
c) `ReadOnly`
d) `Invalid`

11.9 The _____ and _____ properties limit the values users can select in the `NumericUpDown` control.

a) `Maximum`, `Minimum`
b) `Top`, `Bottom`
c) `High`, `Low`
d) `Max`, `Min`

11.10 The `for` header _____ can be used to vary the control variable over the odd numbers between 1 and 10.

a) `for ( i = 1 ; i <= 10; i++ )`
b) `for ( i = 1 ; i <= 10; i += 2 )`
c) `for ( i = 1 ; i <= 10; i-- )`
d) `for ( i = 1 ; i <= 10; i -= 2 )`

Answers: 11.1) a. 11.2) c. 11.3) d. 11.4) b. 11.5) b. 11.6) d. 11.7) a. 11.8) c. 11.9) a. 11.10) b.

EXERCISES

11.11 ***(Present Value Calculator Application)*** A bank wants to show its customers how much they would need to invest to achieve a specified financial goal (future value) in 5, 10, 15, 20, 25 or 30 years. Users must provide their financial goal (the amount of money desired after the specified number of years has elapsed), an interest rate and the length of the investment in years. Create an application that calculates and displays the principal (initial amount to invest) needed to achieve the user's financial goal. Your application should allow the user to invest money for 5, 10, 15, 20, 25 or 30 years. For example, if a customer wants to reach the financial goal of $15,000 over a period of five years when the interest rate is 6.6%, the customer would need to invest $10,896.96 as shown in Fig. 11.17.

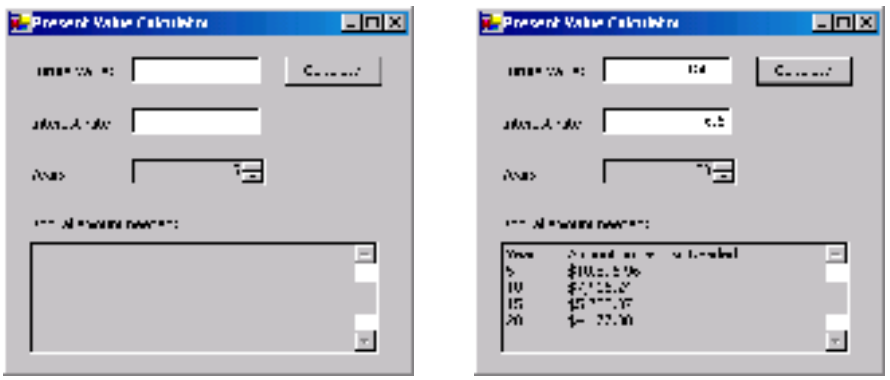


Figure 11.17 Present Value Calculator GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial11\Exercises\PresentValue` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `PresentValue.sln` in the `PresentValue` directory to open the application.

c) ***Adding the `NumericUpDown` control.*** Place and size the `NumericUpDown` so that it follows the GUI Design Guidelines. Set the `NumericUpDown` control's `Name` property to `updYear`. Set the `NumericUpDown` control to allow only multiples of five for the number of years. Also, allow the user to select only a duration that is in the specified range of values.

d) ***Adding a multiline `TextBox`.*** Add a `TextBox` to the `Form` below the `NumericUpDown` control. Change the size to `272, 88` and position the `TextBox` on the `Form` so that it follows the GUI Design Guidelines. Then, set that `TextBox` to display multiple lines and a scrollbar. Also ensure that the user cannot modify the text in the `TextBox`. Rearrange and comment the new control declarations appropriately.

e) ***Creating a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, add code to the application such that, when the **Calculate** `Button` is clicked, the multiline `TextBox` displays the necessary principal for each five-year interval. Use the following version of the present-value calculation formula:

$$p = a / (1 + r)^{n}$$

where
p is the amount needed to achieve the future value
r is the annual interest rate (for example, .05 is equivalent to 5%)
n is the number of years
a is the future-value amount.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter amounts for the future value, interest rate and number of years. Click the **Calculate** `Button` and verify that the year intervals and the amount on deposit needed for each is correct. Test the application again, this time entering `30` for the number of years. Verify that the vertical scrollbar appears to display all of the output.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 11.11 Solution
// PresentValue.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace PresentValue
{
   /// <summary>
   /// Summary description for FrmPresentValue.
   /// </summary>
   public class FrmPresentValue : System.Windows.Forms.Form
   {
      // Label and TextBox to input financial goal amount
      private System.Windows.Forms.Label lblPrincipal;
      private System.Windows.Forms.TextBox txtFutureValue;

      // Label and TextBox to input interest rate
      private System.Windows.Forms.Label lblRate;
      private System.Windows.Forms.TextBox txtRate;

      // Label and NumericUpDown to input number of years
      private System.Windows.Forms.Label lblYears;
      private System.Windows.Forms.NumericUpDown updYear;

      // Label and TextBox to display annual amount needed
      // to achieve goal
      private System.Windows.Forms.Label lblAnnualAmount;
      private System.Windows.Forms.TextBox txtResult;

      // Button to calculate annual amount needed to achieve goal
      private System.Windows.Forms.Button btnCalculate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmPresentValue()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
```

```
59          {
60             if( disposing )
61             {
62                if (components != null)
63                {
64                   components.Dispose();
65                }
66             }
67             base.Dispose( disposing );
68          }
69
70          // Windows Form Designer generated code
71
72          /// <summary>
73          /// The main entry point for the application.
74          /// </summary>
75          [STAThread]
76          static void Main()
77          {
78             Application.Run( new FrmPresentValue() );
79          }
80
81          // handles Calculate Button's Click event
82          private void btnCalculate_Click(
83             object sender, System.EventArgs e )
84          {
85
86             // clear txtResult from previous results
87             txtResult.Clear();
88
89             // declare variables
90             decimal decFutureValue;
91             double dblRate;
92             int intYears;
93             decimal decPresentValue;
94
95             // retrieve values from user input
96             decFutureValue = Decimal.Parse( txtFutureValue.Text );
97             dblRate = Double.Parse( txtRate.Text );
98             intYears = Int32.Parse( updYear.Text );
99
100            // set initial output line
101            txtResult.Text = "Year\tAmount on Deposit Needed\r\n";
102
103            // calculate principal and display result
104            for ( int intCounter = 5; intCounter <= intYears;
105               intCounter += 5 )
106            {
107               decPresentValue = decFutureValue / ( decimal )
108                  Math.Pow( 1 + ( dblRate / 100 ), intCounter );
109
110               // append result to txtresult's text property
111               txtResult.Text += intCounter + "\t" +
112                  String.Format( "{0:C}", decPresentValue) + "\r\n";
113            }
114
115         } // end method btnCalculate_Click
116
117      } // end class FrmPresentValue
118   }
```

11.12 ***(Comparing Rates Application)*** Write an application that calculates the amount of money in an account after 10 years for interest rate amounts of 5%–10% (Fig. 11.18). For this application, users must provide the initial principal.

Figure 11.18 Comparing Rates GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial11\Exercises\ComparingRates` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ComparingRates.sln` in the `ComparingRates` directory to open the application.

c) ***Adding a multiline `TextBox`.*** Add a `TextBox` to the `Form` below the **Result:** `Label` control. Change the size to `256, 104`, and position the `TextBox` on the `Form` so that it follows the GUI Design Guidelines (Fig. 11.18). Then, set that `TextBox` to display multiple lines. Also ensure that the user cannot modify the text in the `TextBox`. Rearrange and comment the new control declaration appropriately.

d) ***Creating a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, add code to the application such that, when the **Calculate** `Button` is clicked, the multiline `TextBox` displays the amount in the account after 10 years for interest rates of 5, 6, 7, 8, 9 and 10 percent. Use the following version of the interest-calculation formula:

$$a = p\ (1 + r)^{n}$$

where
p is the original amount invested (the principal)
r is the annual interest rate (for example, .05 is equivalent to 5%)
n is the number of years
a is the investment's value at the end of the *n*th year.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter the principal amount for an account and click the **Calculate** `Button`. Verify that the correct amounts after 10 years are then displayed, based on interest rate amounts of 5–10%.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 11.12 Solution
2  // ComparingRates.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ComparingRates
12 {
13    /// <summary>
14    /// Summary description for FrmComparingRates.
```

```
15    /// </summary>
16    public class FrmComparingRates : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input principal
19       private System.Windows.Forms.Label lblPrincipal;
20       private System.Windows.Forms.TextBox txtPrincipal;
21
22       // Label and TextBox to display results with different rates
23       private System.Windows.Forms.Label lblResult;
24       private System.Windows.Forms.TextBox txtResult;
25
26       // Button to calculate results with different rates
27       private System.Windows.Forms.Button btnCalculate;
28
29       /// <summary>
30       /// Required designer variable.
31       /// </summary>
32       private System.ComponentModel.Container components = null;
33
34       public FrmComparingRates()
35       {
36          //
37          // Required for Windows Form Designer support
38          //
39          InitializeComponent();
40          //
41          // TODO: Add any constructor code after InitializeComponent
42          // call
43          //
44       }
45
46       /// <summary>
47       /// Clean up any resources being used.
48       /// </summary>
49       protected override void Dispose( bool disposing )
50       {
51          if( disposing )
52          {
53             if (components != null)
54             {
55                components.Dispose();
56             }
57          }
58          base.Dispose( disposing );
59       }
60
61       // Windows Form Designer generated code
62
63       /// <summary>
64       /// The main entry point for the application.
65       /// </summary>
66       [STAThread]
67       static void Main()
68       {
69          Application.Run( new FrmComparingRates() );
70       }
71
72       // invoke when Calculate Button is pressed
73       private void btnCalculate_Click(
74          object sender, System.EventArgs e )
```

```
75         {
76            // declare local variables
77            string strOutput;
78            int intRateCounter;
79            decimal decPrincipal = Decimal.Parse( txtPrincipal.Text );
80            decimal decAmount = 0;
81
82            // set output header
83            strOutput = "Rate(%)\t\tAmount after 10 years\r\n";
84
85            // calculate amount for each rate and append to string
86            for ( intRateCounter = 5; intRateCounter <= 10;
87               intRateCounter++ )
88            {
89               decAmount = decPrincipal * ( decimal )
90                  Math.Pow( 1 + intRateCounter / 100.0, 10 );
91
92               strOutput += Convert.ToString( intRateCounter ) +
93                  "\t\t" + String.Format( "{0:C}", decAmount ) +
94                  "\r\n";
95            }
96
97            txtResult.Text = strOutput; // display result
98
99         } // end method btnCalculate_Click
100
101      } // end class FrmComparingRates
102   }
```

11.13 ***(Enhanced Interest Calculator Application to Validate Input)*** Enhance the **Interest Calculator** application with error checking. Test for whether the user has entered valid values for the principal and interest rate. If the user enters an invalid value, display a message in the multiline `TextBox`. Figure 11.19 demonstrates the application handling an invalid input.

Figure 11.19 **Enhanced Interest Calculator** application with error checking.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial11\Exercises\InterestCalculatorEnhancement` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `InterestCalculator.sln` in the `InterestCalculatorEnhancement` directory to open the application.

c) ***Adding code to the `Click` event handler.*** Modify the `Click` event handler for the **Calculate** `Button` so that it validates the input. The principal should be a positive amount greater than 0. Also, the interest rate should be greater than 0, but less than 100.

d) ***Displaying the error message.*** Display the text "`The information was not within the correct range of values.`" in `txtResult` if the values are not valid.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter invalid data for the principal and interest rate. The invalid data can include negative numbers and letters. Verify that entering invalid data and clicking the **Calculate** `Button` results in the error message displayed in Fig. 11.19.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 11.13 Solution
// InterestCalculator.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace InterestCalculator
{
   /// <summary>
   /// Summary description for FrmInterestCalculator.
   /// </summary>
   public class FrmInterestCalculator : System.Windows.Forms.Form
   {
      // Label and TextBox to input principal
      private System.Windows.Forms.Label lblPrincipal;
      private System.Windows.Forms.TextBox txtPrincipal;

      // Label and TextBox to input interest rate
      private System.Windows.Forms.Label lblRate;
      private System.Windows.Forms.TextBox txtRate;

      // Label and NumericUpDown to input number of years
      private System.Windows.Forms.Label lblYears;
      private System.Windows.Forms.NumericUpDown updYear;

      // Button to calculate yearly account balance
      private System.Windows.Forms.Button btnCalculate;

      // Label and TextBox to display yearly account balance
      private System.Windows.Forms.Label lblYearlyAccount;
      private System.Windows.Forms.TextBox txtResult;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmInterestCalculator()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
```

```
51          //
52       }
53
54       /// <summary>
55       /// Clean up any resources being used.
56       /// </summary>
57       protected override void Dispose( bool disposing )
58       {
59          if( disposing )
60          {
61             if (components != null)
62             {
63                components.Dispose();
64             }
65          }
66          base.Dispose( disposing );
67       }
68
69       // Windows Form Designer generated code
70
71       /// <summary>
72       /// The main entry point for the application.
73       /// </summary>
74       [STAThread]
75       static void Main()
76       {
77          Application.Run( new FrmInterestCalculator() );
78       }
79
80       // handles Calculate Button's Click event
81       private void btnCalculate_Click(
82          object sender, System.EventArgs e )
83       {
84          // declare variables to store user input
85          decimal decPrincipal;  //store principal
86          double dblRate;  //store interest rate
87          int intYear;  // store number of years
88          int intYearCounter;  // store count
89          decimal decAmount;  //store each calculation
90          string strOutput;  // store output
91
92          // retrieve user input
93          decPrincipal = Decimal.Parse( txtPrincipal.Text );
94          dblRate = Double.Parse( txtRate.Text );
95          intYear = Int32.Parse( updYear.Text );
96
97          if ( decPrincipal > 0 && dblRate > 0 && dblRate < 100 )
98          {
99             // set output header
100            strOutput = "Year\tAmount on Deposit\r\n";
101
102            // calculate amount after each year and append to string
103            for ( intYearCounter = 1; intYearCounter <= intYear;
104               intYearCounter++ )
105            {
106               decAmount = decPrincipal * ( decimal )
107                  Math.Pow( 1 + dblRate / 100, intYearCounter );
108
109               strOutput += ( intYearCounter + "\t" +
110                  String.Format( "{0:C}", decAmount ) + "\r\n" );
```

```
            }
         }
         else
         {
            strOutput = "The information input was not within the"
               + " correct range of values.";
         }

         txtResult.Text = strOutput; // display result

      } // end method btnCalculate_Click

   } // end class FrmInterestCalculator
}
```

What does this code do? ▶

11.14 What is the value of `intResult` after the following code executes? Assume that `intPower`, `intI`, `intResult` and `intNumber` are all declared as `int`s.

```
intPower = 5;
intNumber = 10;
intResult = intNumber;

for ( intI = 1; intI <= ( intPower - 1 ); intI++ )
{
   intResult *= intNumber;
}
```

Answer: This code segment raises `intNumber` to the `intPower` power. In this case, `intResult` gets 10^5 (100000). The complete code reads:

```
// Exercise 11.14 Solution
// Exponents.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Exponents
{
   /// <summary>
   /// Summary description for FrmExponents.
   /// </summary>
   public class FrmExponents : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblResultOut;
      private System.Windows.Forms.Label lblPowerOut;
      private System.Windows.Forms.Label lblBaseOut;
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblPower;
      private System.Windows.Forms.Label lblBase;
      /// <summary>
      /// Required designer variable.
      /// </summary>
```

```
27          private System.ComponentModel.Container components = null;
28
29          public FrmExponents()
30          {
31             //
32             // Required for Windows Form Designer support
33             //
34             InitializeComponent();
35             //
36             // TODO: Add any constructor code after InitializeComponent
37             // call
38             //
39          }
40
41          /// <summary>
42          /// Clean up any resources being used.
43          /// </summary>
44          protected override void Dispose( bool disposing )
45          {
46             if( disposing )
47             {
48                if (components != null)
49                {
50                   components.Dispose();
51                }
52             }
53             base.Dispose( disposing );
54          }
55
56          // Windows Form Designer generated code
57
58          /// <summary>
59          /// The main entry point for the application.
60          /// </summary>
61          [STAThread]
62          static void Main()
63          {
64             Application.Run( new FrmExponents() );
65          }
66
67          // calculates intNumber to the intPower power
68          private void FrmExponents_Load(
69              object sender, System.EventArgs e )
70          {
71             int intPower;
72             int intNumber;
73             int intResult;
74             int intI;
75
76             intPower = 5;
77             intNumber = 10;
78             intResult = intNumber;
79
80             for ( intI = 1; intI <= ( intPower - 1 ); intI++ )
81             {
82                intResult *= intNumber;
83             }
84
85             lblResultOut.Text = Convert.ToString( intResult.ToString );
86
```

```
87         } // end method FrmExponents_Load
88
89      } // end class FrmExponents
90   }
```

What's wrong with this code?

11.15 Assume that the `intCounter` variable is declared as an `int` for both a and b. Identify and correct the error(s) in each of the following:

a) This statement should display in a `ListBox` all numbers from 100 to 1 in decreasing order.

```
1  for ( intCounter = 100; intCounter >= 1 )
2  {
3     lstDisplay.Items.Add( intCounter );
4  }
```

Answer: The code needs `intCounter--` at the end of the `for` header.

b) The following code should display in a `ListBox` the odd `int`s from 19 to 1 in decreasing order.

```
1  for ( intCounter = 19; intCounter >= 1; intCounter-- )
2  {
3     lstDisplay.Items.Add( intCounter );
4  }
```

Answer: `intCounter--` should be `intCounter -= 2`. The complete incorrect code reads:

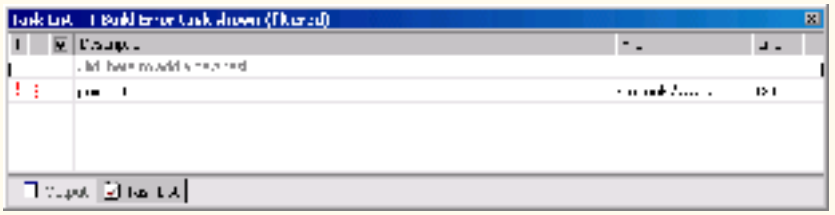
```
1  // Exercise 11.15 Solution
2  // Countdown.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Countdown
12 {
13    /// <summary>
14    /// Summary description for FrmCountdown.
15    /// </summary>
16    public class FrmCountdown : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.ListBox lstDisplay;
20       private System.Windows.Forms.Button btnLoop2;
21       private System.Windows.Forms.Button btnLoop1;
22       /// <summary>
23       /// Required designer variable.
24       /// </summary>
```

```
25          private System.ComponentModel.Container components = null;
26
27          public FrmCountdown()
28          {
29             //
30             // Required for Windows Form Designer support
31             //
32             InitializeComponent();
33             //
34             // TODO: Add any constructor code after InitializeComponent
35             // call
36             //
37          }
38
39          /// <summary>
40          /// Clean up any resources being used.
41          /// </summary>
42          protected override void Dispose( bool disposing )
43          {
44             if( disposing )
45             {
46                if (components != null)
47                {
48                   components.Dispose();
49                }
50             }
51             base.Dispose( disposing );
52          }
53
54          // Windows Form Designer generated code
55
56          /// <summary>
57          /// The main entry point for the application.
58          /// </summary>
59          [STAThread]
60          static void Main()
61          {
62             Application.Run( new FrmCountdown() );
63          }
64
65          // counts down from 100 to 1 in increments of 1
66          private void btnLoop1_Click(
67             object sender, System.EventArgs e )
68          {
69             int intCounter;
70
71             // empty the ListBox
72             lstDisplay.Items.Clear();
73
74             for ( intCounter = 100; intCounter >= 1 )
75             {
76                lstDisplay.Items.Add( intCounter );
77             }
78
79          } // btnLoop1_Click
80
81          // counts down from 19 to 1 in increments of 2
82          private void btnLoop2_Click(
83             object sender, System.EventArgs e )
84          {
```

Missing `; intCounter--` at the end of the `for` statement

```
85          int intCounter;
86
87          // empty the ListBox
88          lstDisplay.Items.Clear();
89
90          for ( intCounter = 19; intCounter >= 1; intCounter-- )
91          {
92             lstDisplay.Items.Add( intCounter );
93          }
94
95       } // end method btnLoop2_Click
96
97    } // end class FrmCountdown
98 }
```

intCounter-- should be intCounter -= 2



Answer: The complete corrected code should read:

```
1  // Exercise 11.15 Solution
2  // Countdown.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Countdown
12 {
13    /// <summary>
14    /// Summary description for FrmCountdown.
15    /// </summary>
16    public class FrmCountdown : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.ListBox lstDisplay;
20       private System.Windows.Forms.Button btnLoop2;
21       private System.Windows.Forms.Button btnLoop1;
22       /// <summary>
23       /// Required designer variable.
24       /// </summary>
25       private System.ComponentModel.Container components = null;
26
27       public FrmCountdown()
28       {
29          //
30          // Required for Windows Form Designer support
31          //
32          InitializeComponent();
33          //
```

```
34          // TODO: Add any constructor code after InitializeComponent
35          // call
36          //
37       }
38
39       /// <summary>
40       /// Clean up any resources being used.
41       /// </summary>
42       protected override void Dispose( bool disposing )
43       {
44          if( disposing )
45          {
46             if (components != null)
47             {
48                components.Dispose();
49             }
50          }
51          base.Dispose( disposing );
52       }
53
54       // Windows Form Designer generated code
55
56       /// <summary>
57       /// The main entry point for the application.
58       /// </summary>
59       [STAThread]
60       static void Main()
61       {
62          Application.Run( new FrmCountdown() );
63       }
64
65       // counts down from 100 to 1 in increments of 1
66       private void btnLoop1_Click(
67          object sender, System.EventArgs e)
68       {
69          int intCounter;
70
71          // empty the ListBox
72          lstDisplay.Items.Clear();
73
74          for ( intCounter = 100; intCounter >= 1; intCounter-- )
75          {
76             lstDisplay.Items.Add( intCounter );
77          }
78
79       } // end method btnLoop1_Click
80
81       // counts down from 19 to 1 in increments of 2
82       private void btnLoop2_Click(
83          object sender, System.EventArgs e)
84       {
85          int intCounter;
86
87          // empty the ListBox
88          lstDisplay.Items.Clear();
89
90          for ( intCounter = 19; intCounter >= 1; intCounter -= 2 )
91          {
92             lstDisplay.Items.Add( intCounter );
93          }
```

```
94
95          } // end method btnLoop2_Click
96
97       } // end class FrmCountdown
98    }
```

Using the Debugger

11.16 (*Savings Calculator Application*) The **Savings Calculator** application calculates the amount that the user will have on deposit after one year. The application gets the initial amount on deposit from the user, and assumes that the user will add $100 to the account every month for the entire year. No interest is added to the account. Copy the directory `C:\Examples\Tutorial11\Debugger\SavingsCalculator` to your `C:\SimplyCSP` directory. Run the application. While testing the application, you noticed that the amount calculated by the application was incorrect. Use the debugger to locate and correct any logic error(s). Figure 11.20 displays the correct output for this application.

Figure 11.20 Correct output for the **Savings Calculator** application.

Answer:

```
1  // Exercise 11.16 Solution
2  // SavingsCalculator.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace SavingsCalculator
12 {
13    /// <summary>
14    /// Summary description for FrmSavingsCalculator.
15    /// </summary>
16    public class FrmSavingsCalculator : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input starting amount
19       private System.Windows.Forms.Label lblStartingAmount;
20       private System.Windows.Forms.TextBox txtStartAmount;
21
22       // Labels display savings after one year
```

```
23          private System.Windows.Forms.Label lblResult;
24          private System.Windows.Forms.Label lblTotal;
25
26          // Button calculates savings after one year
27          private System.Windows.Forms.Button btnCalculate;
28
29          /// <summary>
30          /// Required designer variable.
31          /// </summary>
32          private System.ComponentModel.Container components = null;
33
34          public FrmSavingsCalculator()
35          {
36             //
37             // Required for Windows Form Designer support
38             //
39             InitializeComponent();
40             //
41             // TODO: Add any constructor code after InitializeComponent
42             // call
43             //
44          }
45
46          /// <summary>
47          /// Clean up any resources being used.
48          /// </summary>
49          protected override void Dispose( bool disposing )
50          {
51             if( disposing )
52             {
53                if (components != null)
54                {
55                   components.Dispose();
56                }
57             }
58             base.Dispose( disposing );
59          }
60
61          // Windows Form Designer generated code
62
63          /// <summary>
64          /// The main entry point for the application.
65          /// </summary>
66          [STAThread]
67          static void Main()
68          {
69             Application.Run( new FrmSavingsCalculator() );
70          }
71
72          // calculate amount in account after one year
73          private void btnCalculate_Click(
74             object sender, System.EventArgs e )
75          {
76             int intTotal = 0; // amount on deposit
77             int intCounter = 1; // counter starts at 1
78
79             // get amount on deposit
80             intTotal = Int32.Parse( txtStartAmount.Text );
81
82             // add $100 a month for one year
```

Incorrect code given to students looped 13 times (0-12), instead of 12 (1-12)

```
83          for ( intCounter = 1; intCounter <= 12;
84             intCounter++ )
85          {
86             intTotal += 100; // add money
87          }
88
89          // display total after 12 months
90          lblTotal.Text = Convert.ToString( intTotal );
91
92       } // end method btnCalculate_Click
93
94    } // end class FrmSavingsCalculator
95 }
```

Programming Challenge

11.17 ***(Pay Raise Calculator Application)*** Develop an application that computes the amount of money an employee makes each year over a user-specified number of years (Fig. 11.21). The employee receives an hourly wage and a pay raise once every year. The user specifies the hourly wage and the amount of the raise (in percentages per year) in the application.

Figure 11.21 **Pay Raise** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial11\Exercises\PayRaise` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `PayRaise.sln` in the `PayRaise` directory to open the application.

c) ***Adding controls to the Form.*** Add two `NumericUpDown` controls to the `Form`. The first `NumericUpDown` control should be provided to allow the user to specify the pay raise percentage. The user should only be able to specify percentages in the range 3–8 percent. Create the second `NumericUpDown` control for users to select the number of years in the range 1–50. Then add a multiline `TextBox` control to the application. Ensure that the user cannot modify the text in the `NumericUpDown` and `TextBox` controls. Resize and move the controls you created so that they follow the GUI Design Guidelines as in Fig. 11.21. Rearrange and comment the new control declarations appropriately.

d) ***Creating a `Click` event handler and adding code.*** Add a `Click` event handler for the **Calculate** `Button`. Once in code view, add code to use the `for` statement to compute the yearly salary amounts, based on the yearly pay raise.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter a starting wage per hour, the size of the yearly raise and the number of years worked. Click the **Calculate** `Button` and verify that the correct amount after each year is displayed in the **Yearly amount earned:** `TextBox`.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 11.17 Solution
// PayRaise.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace PayRaise
{
   /// <summary>
   /// Summary description for FrmPayRaise.
   /// </summary>
   public class FrmPayRaise : System.Windows.Forms.Form
   {
      // Label and TextBox to input wage per hour
      private System.Windows.Forms.Label lblWage;
      private System.Windows.Forms.TextBox txtWage;

      // Label and NumericUpDown to input amount of yearly
      // raise in percent
      private System.Windows.Forms.Label lblRaise;
      private System.Windows.Forms.NumericUpDown updRaise;

      // Label and NumericUpDown to input number of years
      private System.Windows.Forms.Label lblYears;
      private System.Windows.Forms.NumericUpDown updYears;

      // Label and TextBox to display future salaries
      private System.Windows.Forms.Label lblYearlyAmount;
      private System.Windows.Forms.TextBox txtResult;

      // Button to calculate future salaries
      private System.Windows.Forms.Button btnCalculate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmPayRaise()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
```

```
60          if( disposing )
61          {
62             if (components != null)
63             {
64                components.Dispose();
65             }
66          }
67          base.Dispose( disposing );
68       }
69
70       // Windows Form Designer generated code
71
72       /// <summary>
73       /// The main entry point for the application.
74       /// </summary>
75       [STAThread]
76       static void Main()
77       {
78          Application.Run( new FrmPayRaise() );
79       }
80
81       // invoked when Calculate Button is clicked
82       private void btnCalculate_Click(
83          object sender, System.EventArgs e )
84       {
85          int intYears = Int32.Parse( updYears.Text );
86          int intCounter;
87          decimal decWage = Decimal.Parse( txtWage.Text );
88          int intCurrentYear = 0;
89          decimal decTotal = 0;
90
91          // create headers to display in txtResult
92          txtResult.Text = "Year\tAmount\r\n";
93
94          // calculate first year's total
95          decTotal += ( decWage * 40 * 52 );
96
97          // display wages per year with raise
98          for ( intCounter = 1; intCounter <= intYears;
99             intCounter++ )
100         {
101            // determine if raise should be applied
102            if ( intCounter != 1 )
103            {
104               // calculate total with raise amount
105               decTotal *= ( decimal )
106                  ( 1 + ( Double.Parse( updRaise.Text ) / 100.0 ) );
107            }
108
109            intCurrentYear++; // increment count
110
111            // append amounts to string displayed in
112            // txtResult TextBox
113            txtResult.Text += intCurrentYear + "\t" +
114               String.Format( "{0:C}", decTotal ) + "\r\n";
115         }
116
117      } // end method btnCalculate_Click
118
```

```
119    } // end class FrmPayRaise
120 }
```

12 TUTORIAL

Security Panel Application

Introducing the `switch` Multiple-Selection Statement

Solutions

Instructor's Manual Exercise Solutions Tutorial 12

MULTIPLE-CHOICE QUESTIONS

12.1 The _____ symbol signifies the end of a `switch` statement.

a) right brace (`}`)
b) right square bracket (`]`)
c) newline (`\n`)
d) backslash (`\`)

12.2 The _____ expression returns the current system time and date.

a) `DateTime.DateTime`
b) `DateTime.SystemDateTime`
c) `DateTime.Now`
d) `DateTime.SystemTimeDate`

12.3 You can hide information entered into a `TextBox` by setting that `TextBox`'s _____ property to a character—that character will be displayed for every character entered by the user.

a) `PrivateChar`
b) `Mask`
c) `MaskingChar`
d) `PasswordChar`

12.4 Which of the following is a syntax error?

a) Having duplicate `case`s in the same `switch` statement.
b) Using an `int` variable in the test expression of a `switch` statement.
c) Having a `default case` in a `switch` statement.
d) Using a `string` variable in the test expression of a `switch` statement.

12.5 The _____ technique is used to specify multiple values for a `case`.

a) multiple selection
b) nested `switch`
c) fall-through
d) `default`

12.6 The _____ keyword is required at the end of each non-empty `case`.

a) `endcase`
b) `default`
c) `switch`
d) `break`

12.7 An empty `case` of a `switch` statement _____.

a) contains only the `break` statement
b) does not contain any statements
c) never matches the controlling expression
d) None of the above.

12.8 Forgetting the `break` keyword in a non-empty `case` of a `switch` statement results in _____.

a) a logic error
b) a syntax error
c) an infinite loop
d) the next `case` executing

12.9 The expression following the `switch` keyword is called a _____.

a) guard condition
b) controlling expression
c) selection expression
d) `case` expression

12.10 To prevent a user from modifying text in a `TextBox`, set its _____ property to `false`.

a) `Enabled`
b) `Text`
c) `TextChange`
d) `Editable`

Answers: 12.1) a. 12.2) c. 12.3) d. 12.4) a. 12.5) c. 12.6) d. 12.7) b. 12.8) b. 12.9) b. 12.10) a.

EXERCISES

12.11 ***(Sales Commission Calculator Application)*** Develop an application that calculates a salesperson's commission from the number of items sold (Fig. 12.17). Assume that all items have a fixed price of 100 dollars per unit. Use a `switch` statement to implement the following sales commission schedule:

Fewer than 10 items sold = 1% commission
Between 10 and 39 items sold = 2% commission
Between 40 and 99 items sold = 3% commission
More than 99 items sold = 4% commission

Figure 12.17 Sales Commission Calculator application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial12\Exercises\SalesCommissionCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SalesCommissionCalculator.sln` in the `SalesCommissionCalculator` directory to open the application.

c) ***Defining an event handler for the `Button`'s `Click` event.*** Create an event handler for the **Calculate** `Button`'s `Click` event.

d) ***Calculate the gross sales.*** In your new event handler, convert the user's input to an `int` value and assigns the value to the `intItems` local variable. Then insert a statement that multiplies the number of items that the salesperson has sold by the cost per item and assigns the result to `double` variable `dblSales`.

e) ***Determine the salesperson's commission percentage.*** Declare `int` local variable `intCommission` to store the commission percentage. Next, insert a `switch` statement to determine the salesperson's commission percentage from the number of items sold. In this `switch` statement, assign the commission percentage as a whole number to the `intCommission` variable. For example, if the commission percentage is 2%, assign 2 to `commission`. The controlling expression should be `intItems / 10`. Because this is integer arithmetic, any number of items in the range 0–9 will result in 0, any number of items in the range 10–19 will result in 1, etc. Inside the `switch` statement, provide `case`s that enable the `switch` to test for values in the ranges specified by the problem statement.

f) ***Calculate the salesperson's earnings.*** Insert a statement that multiplies `intSales` by `intCommission` (divided by `100.0`) and assigns the result to `double` local variable `dblEarnings`.

g) ***Display the gross sales, the commission percentage and the salesperson's earnings.*** Display the values of `dblSales`, `intCommission` and `dblEarnings` local variables in their corresponding `TextBoxes`. For the sales and earnings values, format the values as dollar amounts.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a value for the number of items sold and click the **Calculate** `Button`. Verify that the gross sales displayed is correct, that the percentage of commission is correct and that the earnings displayed is correct based on the commission assigned.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 12.11 Solution
// SalesCommissionCalculator.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SalesCommissionCalculator
{
   /// <summary>
   /// Summary description for FrmSalesCommission.
   /// </summary>
   public class FrmSalesCommission : System.Windows.Forms.Form
   {
      // Label and TextBox to input number of items sold
      private System.Windows.Forms.Label lblItemsSold;
      private System.Windows.Forms.TextBox txtItemsSold;

      // Labels to display gross sales
      private System.Windows.Forms.Label lblGrossSales;
      private System.Windows.Forms.Label lblGrossSalesResult;

      // Labels to display commission percentage
      private System.Windows.Forms.Label lblCommissionPercentage;
      private System.Windows.Forms.Label
         lblCommissionPercentageResult;

      // Labels to display earnings
      private System.Windows.Forms.Label lblEarnings;
      private System.Windows.Forms.Label lblEarningsResult;

      // Button to perform calculations
      private System.Windows.Forms.Button btnCalculate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmSalesCommission()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
```

```
60          if( disposing )
61          {
62             if (components != null)
63             {
64                components.Dispose();
65             }
66          }
67          base.Dispose( disposing );
68       }
69
70       // Windows Form Designer generated code
71
72       /// <summary>
73       /// The main entry point for the application.
74       /// </summary>
75       [STAThread]
76       static void Main()
77       {
78          Application.Run( new FrmSalesCommission() );
79       }
80
81       // invoked when Calculate Button is clicked
82       private void btnCalculate_Click(
83          object sender, System.EventArgs e )
84       {
85          // number of items sold
86          int intItems = Int32.Parse( txtItemsSold.Text );
87
88          double dblSales = intItems * 100.0; // gross sales
89          int intCommission; // commission percentage
90
91          // determine commission percentage
92          switch ( intItems / 10 )
93          {
94             // number of items sold 0-9
95             case 0:
96                intCommission = 1; // 1% commission
97                break;
98
99             // number of items sold 10-39
100            case 1:
101            case 2:
102            case 3:
103               intCommission = 2; // 2% commission
104               break;
105
106            // number of items sold 40-99
107            case 4:
108            case 5:
109            case 6:
110            case 7:
111            case 8:
112            case 9:
113               intCommission = 3; // 3% commission
114               break;
115
116            // number of items sold over 99
117            default:
118               intCommission = 4; // 4% commission
119               break;
```

```
120
121          } // end switch statement
122
123          // calculate earnings
124          double dblEarnings = dblSales * ( intCommission / 100.0 );
125
126          // display results
127          lblGrossSalesResult.Text =
128             String.Format( "{0:C}", dblSales );
129          lblCommissionPercentageResult.Text =
130             Convert.ToString( intCommission );
131          lblEarningsResult.Text =
132             String.Format( "{0:C}", dblEarnings );
133
134       } // end method btnCalculate_Click
135
136    } // end class FrmSalesCommission
```

12.12 ***(Cash Register Application)*** Use the numeric keypad from the **Security Panel** application to build a **Cash Register** application (Fig. 12.18). In addition to numbers, the cash register should include a decimal point `Button`. Apart from this numeric operation, there should be **Enter**, **Delete**, **Clear** and **Total** `Buttons`. Sales tax should be calculated on the amount purchased. Use a `switch` statement to compute sales tax. Add the tax amount to the subtotal to calculate the total. Display the tax and total for the user. Use the following sales-tax percentages, which are based on the amount of money spent:

Amounts under $100 = 5% (.05) sales tax
Amounts between $100 and $499 = 7.5% (.075) sales tax
Amounts above $499 = 10% (.10) sales tax

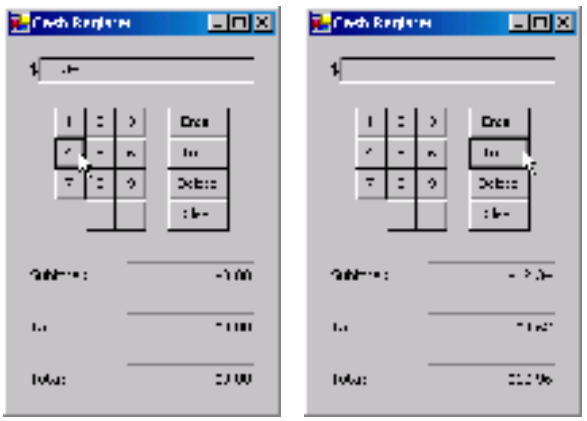


Figure 12.18 **Cash Register** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial12\Exercises\CashRegister` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CashRegister.sln` in the `CashRegister` directory to open the application.

c) ***Define event handlers for the numeric `Button`s and decimal point in the keypad.*** Create event handlers for the `Click` events each of these `Button`'s. Have each event handler concatenate the proper value to the `TextBox` at the top of the `Form`.

d) ***Define an event handler for the Enter `Button`'s `Click` event.*** Create an event handler for this `Button`'s `Click` event. Have this event handler add the current amount to the subtotal and display the new subtotal. Use the expression `Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) )` to skip over the dollar sign and extract the subtotal from its `Label`. [*Note*: You will learn more about `String` method `Substring` in later tutorials.]

e) ***Define an event handler for the Total Button's Click event.*** Create an event handler for this Button's Click event. Have this event handler use the value in the **Subtotal** Label to compute the tax amount. Extract the subtotal value as you did in *Step d*, and assign it to decimal variable decSubTotal. Insert a switch statement that uses the controlling expression (int) decSubTotal / 100, which converts the subtotal to an int, then divides the subtotal by 100. The value of this expression will be 0 for subtotals less than $100 and 1–4 for subtotals in the range $100–499. All other values should be handled by the default case in this exercise. Calculate the tax amount and total amount based on the subtotal and tax rate. Display these values in their corresponding Labels.

f) ***Define an event handler for the Clear Button's Click event.*** Create an event handler for this Button's Click event. Have this event handler clear the user input and display the value $0.00 for the subtotal, sales tax and total.

g) ***Define an event handler for the Delete Button's Click event.*** Create an event handler for this Button's Click event. Have this event handler clear only the data in the TextBox.

h) ***Running the application.*** Select **Debug > Start** to run your application. Use the keypad to enter various dollar amounts, clicking the **Enter** Button after each. After several amounts have been entered, click the **Total** Button and verify that the appropriate sales tax and total are displayed. Enter several values again and click the **Delete** Button to clear the current input. Click the **Clear** Button to clear all the output values.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 12.12 Solution
// CashRegister.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CashRegister
{
   /// <summary>
   /// Summary description for FrmCashRegister.
   /// </summary>
   public class FrmCashRegister : System.Windows.Forms.Form
   {
      // Label and TextBox to display current price
      private System.Windows.Forms.Label lblDollarSign;
      private System.Windows.Forms.TextBox txtCurrentPrice;

      // Buttons to input prices
      private System.Windows.Forms.Button btnOne;
      private System.Windows.Forms.Button btnTwo;
      private System.Windows.Forms.Button btnThree;
      private System.Windows.Forms.Button btnFour;
      private System.Windows.Forms.Button btnFive;
      private System.Windows.Forms.Button btnSix;
      private System.Windows.Forms.Button btnSeven;
      private System.Windows.Forms.Button btnEight;
      private System.Windows.Forms.Button btnNine;
      private System.Windows.Forms.Button btnZero;
      private System.Windows.Forms.Button btnPoint;
```

```
34
35          // Buttons to enter a price, calculate the total,
36          // clear the price in the TextBox and clear all input,
37          // respectively.
38          private System.Windows.Forms.Button btnEnter;
39          private System.Windows.Forms.Button btnTotal;
40          private System.Windows.Forms.Button btnDelete;
41          private System.Windows.Forms.Button btnClear;
42
43          // Labels to display the subtotal
44          private System.Windows.Forms.Label lblSubtotal;
45          private System.Windows.Forms.Label lblSubTotalValue;
46
47          // Labels to display the tax
48          private System.Windows.Forms.Label lblTax;
49          private System.Windows.Forms.Label lblTaxValue;
50
51          // Labels to display the total price
52          private System.Windows.Forms.Label lblTotal;
53          private System.Windows.Forms.Label lblTotalValue;
54
55          /// <summary>
56          /// Required designer variable.
57          /// </summary>
58          private System.ComponentModel.Container components = null;
59
60          public FrmCashRegister()
61          {
62             //
63             // Required for Windows Form Designer support
64             //
65             InitializeComponent();
66             //
67             // TODO: Add any constructor code after InitializeComponent
68             // call
69             //
70          }
71
72          /// <summary>
73          /// Clean up any resources being used.
74          /// </summary>
75          protected override void Dispose( bool disposing )
76          {
77             if( disposing )
78             {
79                if (components != null)
80                {
81                   components.Dispose();
82                }
83             }
84             base.Dispose( disposing );
85          }
86
87          // Windows Form Designer generated code
88
89          /// <summary>
90          /// The main entry point for the application.
91          /// </summary>
92          [STAThread]
93          static void Main()
```

```
94          {
95             Application.Run( new FrmCashRegister() );
96          }
97
98          // invoked when Button 1 is clicked
99          private void btnOne_Click(
100            object sender, System.EventArgs e )
101         {
102            txtCurrentPrice.Text += "1";
103
104         } // end method btnOne_Click
105
106         // invoked when Button 2 is clicked
107         private void btnTwo_Click(
108            object sender, System.EventArgs e )
109         {
110            txtCurrentPrice.Text += "2";
111
112         } // end method btnTwo_Click
113
114         // invoked when Button 3 is clicked
115         private void btnThree_Click(
116            object sender, System.EventArgs e )
117         {
118            txtCurrentPrice.Text += "3";
119
120         } // end method btnThree_Click
121
122         // invoked when Button 4 is clicked
123         private void btnFour_Click(
124            object sender, System.EventArgs e )
125         {
126            txtCurrentPrice.Text += "4";
127
128         } // end method btnFour_Click
129
130         // invoked when Button 5 is clicked
131         private void btnFive_Click(
132            object sender, System.EventArgs e )
133         {
134            txtCurrentPrice.Text += "5";
135
136         } // end method btnFive_Click
137
138         // invoked when Button 6 is clicked
139         private void btnSix_Click(
140            object sender, System.EventArgs e )
141         {
142            txtCurrentPrice.Text += "6";
143
144         } // end method btnSix_Click
145
146         // invoked when Button 7 is clicked
147         private void btnSeven_Click(
148            object sender, System.EventArgs e )
149         {
150            txtCurrentPrice.Text += "7";
151
152         } // end method btnSeven_Click
153
```

```
154         // invoked when Button 8 is clicked
155         private void btnEight_Click(
156            object sender, System.EventArgs e )
157         {
158            txtCurrentPrice.Text += "8";
159
160         } // end method btnEight_Click
161
162         // invoked when Button 9 is clicked
163         private void btnNine_Click(
164            object sender, System.EventArgs e )
165         {
166            txtCurrentPrice.Text += "9";
167
168         } // end method btnNine_Click
169
170         // invoked when Button 0 is clicked
171         private void btnZero_Click(
172            object sender, System.EventArgs e )
173         {
174            txtCurrentPrice.Text += "0";
175
176         } // end method btnZero_Click
177
178         // invoked when Button . is clicked
179         private void btnPoint_Click(
180            object sender, System.EventArgs e )
181         {
182            txtCurrentPrice.Text += ".";
183
184         } // end method btnPoint_Click
185
186         // invoke when Enter Button is clicked
187         private void btnEnter_Click(
188            object sender, System.EventArgs e )
189         {
190            // variable to store new value
191            decimal decAmount;
192
193            // store value in txtCurrentPrice to decAmount
194            decAmount = Decimal.Parse( txtCurrentPrice.Text );
195
196            // add input amount to dblTotal and clear TextBox
197            lblSubTotalValue.Text = String.Format( "{0:C}", decAmount +
198               Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) ) );
199
200            txtCurrentPrice.Clear(); // clear the TextBox
201
202         } // end method btnEnter_Click
203
204         // invoke when Total Button is clicked
205         private void btnTotal_Click(
206            object sender, System.EventArgs e )
207         {
208            double dblTaxRate;
209            decimal decSubTotal =
210               Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) );
211
212            // determines tax rate based on subtotal
213            switch ( ( int ) decSubTotal / 100 )
```

```
214          {
215             // amounts in the range $0-99
216             case 0:
217                dblTaxRate = 0.05;
218                break;
219
220             // amounts in the range $100-499
221             case 1:
222             case 2:
223             case 3:
224             case 4:
225                dblTaxRate = 0.075;
226                break;
227
228             // amounts $500 and above
229             default:
230                dblTaxRate = 0.1;
231                break;
232
233          } // end switch
234
235          // display tax and total
236          lblTaxValue.Text = String.Format( "{0:C}",
237             decSubTotal * ( decimal ) dblTaxRate );
238
239          lblTotalValue.Text = String.Format( "{0:C}",
240             decSubTotal * ( decimal ) ( 1 + dblTaxRate ) );
241
242       } // end method btnTotal_Click
243
244       // invoke when Delete Button is clicked
245       private void btnDelete_Click(
246          object sender, System.EventArgs e )
247       {
248          txtCurrentPrice.Clear(); // clear the TextBox
249
250       } // end method btnDelete_Click
251
252       // invoke when Clear Button is clicked
253       private void btnClear_Click(
254          object sender, System.EventArgs e )
255       {
256          // clear txtCurrentPrice and set remaining Labels to $0.00
257          txtCurrentPrice.Clear();
258          lblSubTotalValue.Text = "$0.00";
259          lblTaxValue.Text = "$0.00";
260          lblTotalValue.Text = "$0.00";
261
262       } // end method btnClear_Click
263
264    } // end class FrmCashRegister
265 }
```

12.13 ***(Income Tax Calculator Application)*** Create an application that computes the amount of income tax that a person must pay, depending upon that person's salary. Your application should perform as shown in Fig. 12.19. Use the following income ranges and corresponding tax rates:

Under $25,000 = 2% income tax
$25,000 – 49,999 = 5% income tax

$50,000 – 74,999 = 10% income tax
$75,000 – 99,999 = 15% income tax
$100,000 and over = 20% income tax

Figure 12.19 Income Tax Calculator application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial12\Exercises\IncomeTaxCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `IncomeTaxCalculator.sln` in the `IncomeTaxCalculator` directory to open the application.

c) ***Define an event handler for the Calculate `Button`'s `Click` event.*** Use the designer to create an event handler for this `Button`'s `Click` event. Have this event handler convert the user's input to an `int` value and assign the value to the local variable `intSalary`. Then use a `switch` statement to determine the user's income-tax percentage. Use the controlling expression `intSalary / 25000` to determine the tax rate. If the salary is less than $25,000, the controlling expression's value will be `0`. For salaries in the range $25,000–$49,999, the controlling expression's value will be `1`. For salaries in the range $50,000–$74,999, the controlling expression's value will be `2`. For salaries in the range $75,000–$99,999, the controlling expression's value will be `3`. For all other salaries, use the `default case`. The tax rate should then be multiplied by the user's salary and displayed in the output `Label`. Clear the input `TextBox` when the output is displayed.

d) ***Running the application.*** Select **Debug > Start** to run your application. Enter a yearly salary and click the **Calculate** `Button`. Verify that the appropriate income tax is displayed, based on the ranges listed in the exercise description.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 12.13 Solution
2  // IncomeTax.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace IncomeTax
12 {
13    /// <summary>
14    /// Summary description for FrmIncomeTax.
15    /// </summary>
16    public class FrmIncomeTax : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input yearly salary
19       private System.Windows.Forms.Label lblSalary;
20       private System.Windows.Forms.TextBox txtSalary;
21
22       // Labels to display income tax
```

```
23          private System.Windows.Forms.Label lblIncomeTax;
24          private System.Windows.Forms.Label lblResult;
25
26          // Button to calculate income tax
27          private System.Windows.Forms.Button btnCalculate;
28
29          /// <summary>
30          /// Required designer variable.
31          /// </summary>
32          private System.ComponentModel.Container components = null;
33
34          public FrmIncomeTax()
35          {
36             //
37             // Required for Windows Form Designer support
38             //
39             InitializeComponent();
40             //
41             // TODO: Add any constructor code after InitializeComponent
42             // call
43             //
44          }
45
46          /// <summary>
47          /// Clean up any resources being used.
48          /// </summary>
49          protected override void Dispose( bool disposing )
50          {
51             if( disposing )
52             {
53                if (components != null)
54                {
55                   components.Dispose();
56                }
57             }
58             base.Dispose( disposing );
59          }
60
61          // Windows Form Designer generated code
62
63          /// <summary>
64          /// The main entry point for the application.
65          /// </summary>
66          [STAThread]
67          static void Main()
68          {
69             Application.Run( new FrmIncomeTax() );
70          }
71
72          // invoked when Calculate Button is clicked
73          private void btnCalculate_Click(
74             object sender, System.EventArgs e )
75          {
76             double dblPercent;
77             int intSalary = Int32.Parse( txtSalary.Text );
78
79             // determine income percentage
80             switch ( intSalary / 25000 )
81             {
82                case 0:
```

```
83                dblPercent = 0.02;
84                break;
85
86             case 1:
87                dblPercent = 0.05;
88                break;
89
90             case 2:
91                dblPercent = 0.1;
92                break;
93
94             case 3:
95                dblPercent = 0.15;
96                break;
97
98             default:
99                dblPercent = 0.2;
100               break;
101         }
102
103         // display result in currency format
104         lblResult.Text = String.Format( "{0:C}",
105            intSalary * dblPercent );
106
107         txtSalary.Clear(); // clear the TextBox
108
109      } // end method btnCalculate_Click
110
111   } // end class FrmIncomeTax
112 }
```

What does this code do?

12.14 What is output by the following code? Assume that `btnDonation` is a `Button`, `txtDonation` is a `TextBox` and `lblMessage` is an output `Label`.

```
1  private void btnDonation_Click(
2     object sender, System.EventArgs e )
3  {
4     if ( Int32.Parse( txtDonationAmount.Text ) <= 0 )
5     {
6        lblMessage.Text = "Please enter a donation.";
7     }
8     else
9     {
10       switch ( Int32.Parse( txtDonationAmount.Text ) / 100 )
11       {
12          case 0:
13             lblMessage.Text = "Thank you for your donation.";
14             break;
15
16          case 1:
17             lblMessage.Text =
18                "Thank you very much for your donation!";
19             break;
20
21          default:
22             lblMessage.Text =
23                "Wow! Thank you for your generosity!";
24             break;
```

```
25          }
26       }
27
28    } // end method btnDonation_Click
```

Answer: The output `Label lblMessage` displays "`Thank you for your donation`" if the user enters a value under 100, "`Thank you very much for your donation`" if the user enters a value between 100 and 199 and "`Wow! Thank you for your generosity!`" if the user enters more than 199. The complete correct code reads:

```
1  // Exercise 12.14 Solution
2  // Donation.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Donation
12 {
13    /// <summary>
14    /// Summary description for FrmDonation.
15    /// </summary>
16    public class FrmDonation : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblMessagetext;
19       private System.Windows.Forms.Label lblMessage;
20       private System.Windows.Forms.Button btnDonation;
21       private System.Windows.Forms.TextBox txtDonationAmount;
22       private System.Windows.Forms.Label lblAmount;
23       /// <summary>
24       /// Required designer variable.
25       /// </summary>
26       private System.ComponentModel.Container components = null;
27
28       public FrmDonation()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34          //
35          // TODO: Add any constructor code after InitializeComponent
36          // call
37          //
38       }
39
40       /// <summary>
41       /// Clean up any resources being used.
42       /// </summary>
43       protected override void Dispose( bool disposing )
44       {
45          if( disposing )
46          {
47             if (components != null)
48             {
49                components.Dispose();
```

```
50             }
51          }
52          base.Dispose( disposing );
53       }
54
55       // Windows Form Designer generated code
56
57       /// <summary>
58       /// The main entry point for the application.
59       /// </summary>
60       [STAThread]
61       static void Main()
62       {
63          Application.Run( new FrmDonation() );
64       }
65
66       private void btnDonation_Click(
67          object sender, System.EventArgs e )
68       {
69          if ( Int32.Parse( txtDonationAmount.Text ) <= 0 )
70          {
71             lblMessage.Text = "Please enter a donation.";
72          }
73          else
74          {
75             switch ( Int32.Parse( txtDonationAmount.Text ) / 100 )
76             {
77                case 0:
78                   lblMessage.Text = "Thank you for your donation.";
79                   break;
80
81                case 1:
82                   lblMessage.Text =
83                      "Thank you very much for your donation!";
84                   break;
85
86                default:
87                   lblMessage.Text =
88                      "Wow! Thank you for your generosity!";
89                   break;
90             }
91          }
92
93       } // end method btnDonation_Click
94
95    } // end class FrmDonation
96 }
```

What's wrong with this code?

12.15 This `switch` statement should determine whether the `int` variable `intValue` is even or odd. Assume that `txtInput` is a `TextBox` and `lblOutput` is an output `Label`. Find the error(s) in the following code:

```
1  intValue = Int32.Parse( txtInput.Text );
2
3  switch ( intValue % 2 )
4  {
5     case 0 :
6        lblOutput.Text = "Odd Integer";
7        break;
8
9     case 1 :
10       lblOutput.Text = "Even Integer";
11       break;
12 }
```

Answer: Line 6 and line 10 should be swapped. The complete incorrect code reads:

```
1  // Exercise 12.15 Solution
2  // EvenOdd.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace EvenOdd
12 {
13    /// <summary>
14    /// Summary description for FrmEvenOdd.
15    /// </summary>
16    public class FrmEvenOdd : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Button btnTest;
19       private System.Windows.Forms.TextBox txtInput;
20       private System.Windows.Forms.Label lblOutput;
21       private System.Windows.Forms.Label lblEvenOdd;
22       private System.Windows.Forms.Label lblNumber;
23       /// <summary>
24       /// Required designer variable.
25       /// </summary>
26       private System.ComponentModel.Container components = null;
27
28       public FrmEvenOdd()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34          //
35          // TODO: Add any constructor code after InitializeComponent
36          // call
37          //
38       }
39
```

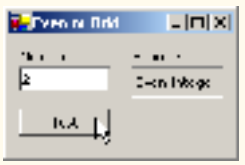
```
40          /// <summary>
41          /// Clean up any resources being used.
42          /// </summary>
43          protected override void Dispose( bool disposing )
44          {
45             if( disposing )
46             {
47                if (components != null)
48                {
49                   components.Dispose();
50                }
51             }
52             base.Dispose( disposing );
53          }
54
55          // Windows Form Designer generated code
56
57          /// <summary>
58          /// The main entry point for the application.
59          /// </summary>
60          [STAThread]
61          static void Main()
62          {
63             Application.Run( new FrmEvenOdd() );
64          }
65
66          // tests whether a number is even or odd
67          private void btnTest_Click(
68             object sender, System.EventArgs e )
69          {
70             int intValue;
71
72             intValue = Int32.Parse( txtInput.Text );
73
74             switch ( intValue % 2 )
75             {
76                case 0:
77                   lblOutput.Text = "Odd Integer";
78                   break;
79
80                case 1:
81                   lblOutput.Text = "Even Integer";
82                   break;
83             }
84
85          } // end method btnTest_Click
86
87       } // end class FrmEvenOdd
88    }
```

Even and Odd should be swapped

Answer: The complete corrected code should read:

```
1  // Exercise 12.15 Solution
2  // EvenOdd.cs (Correct)
3
4  using System;
5  using System.Drawing;private
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace EvenOdd
12 {
13    /// <summary>
14    /// Summary description for FrmEvenOdd.
15    /// </summary>
16    public class FrmEvenOdd : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Button btnTest;
19       private System.Windows.Forms.TextBox txtInput;
20       private System.Windows.Forms.Label lblOutput;
21       private System.Windows.Forms.Label lblEvenOdd;
22       private System.Windows.Forms.Label lblNumber;
23       /// <summary>
24       /// Required designer variable.
25       /// </summary>
26       private System.ComponentModel.Container components = null;
27
28       public FrmEvenOdd()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34          //
35          // TODO: Add any constructor code after InitializeComponent
36          // call
37          //
38       }
39
40       /// <summary>
41       /// Clean up any resources being used.
42       /// </summary>
43       protected override void Dispose( bool disposing )
44       {
45          if( disposing )
46          {
47             if (components != null)
48             {
49                components.Dispose();
50             }
51          }
52          base.Dispose( disposing );
53       }
54
55       // Windows Form Designer generated code
56
57       /// <summary>
58       /// The main entry point for the application.
59       /// </summary>
60       [STAThread]
```

```
61         static void Main()
62         {
63            Application.Run( new FrmEvenOdd() );
64         }
65
66         // tests whether a number is even or odd
67         private void btnTest_Click(
68            object sender, System.EventArgs e )
69         {
70            int intValue;
71
72            intValue = Int32.Parse( txtInput.Text );
73
74            switch ( intValue % 2 )
75            {
76               case 0:
77                  lblOutput.Text = "Even Integer;
78                  break;
79
80               case 1: lblOutput.Text = "Odd Integer";
81                  break;
82
83            }
84
85         } // end method btnTest_Click
86
87      } // end class FrmEvenOdd
88   }
```



Using the Debugger

12.16 (*Discount Calculator Application*) The **Discount Calculator** application determines the discount the user will receive, based on how much money the user spends. A 15% discount is received for purchases of $150 or more, a 10% discount is received for purchases from $100–$149 and a 5% discount is received for purchases from $50–$99. Purchases less than $50 do not receive a discount. While testing your application, you notice that the application is not calculating the discount properly for some values. Use the debugger to find and fix the logic error(s) in the application. Figure 12.20 displays the correct output for the application.

Figure 12.20 Correct output for the **Discount Calculator** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial12\Exercises\Debugger\DiscountCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `DiscountCalculator.sln` in the `DiscountCalculator` directory to open the application.

c) ***Finding and correcting the error(s).*** Use the debugging skills learned in previous tutorials to determine where the application's logic errors exist.

d) ***Running the application.*** Select **Debug > Start** to run your application. Test your application with at least one value in each range.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

```
// Exercise 12.16 Solution
// DiscountCalculator.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace DiscountCalculator
{
   /// <summary>
   /// Summary description for FrmDiscountCalculator.
   /// </summary>
   public class FrmDiscountCalculator : System.Windows.Forms.Form
   {
      // Label and TextBox to input amount spent
      private System.Windows.Forms.Label lblAmount;
      private System.Windows.Forms.TextBox txtAmount;

      // Button to view discount
      private System.Windows.Forms.Button btnView;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmDiscountCalculator()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
```

```
53          }
54          base.Dispose( disposing );
55       }
56
57       // Windows Form Designer generated code
58
59       /// <summary>
60       /// The main entry point for the application.
61       /// </summary>
62       [STAThread]
63       static void Main()
64       {
65          Application.Run( new FrmDiscountCalculator() );
66       }
67
68       // handles View Button's Click event
69       private void btnView_Click(
70          object sender, System.EventArgs e )
71       {
72          int intTotal; // amount spent
73          string strDiscount; // discount rate
74
75          intTotal = Int32.Parse( txtAmount.Text ); // get total
76
77          switch ( intTotal / 50 )
78          {
79             // values in the range $0-49
80             case 0:
81                strDiscount = "0";
82                break;
83
84             // values in the range $50-99
85             case 1:
86                strDiscount = "5";
87                break;
88
89             // values in the range $100-149
90             case 2:
91                strDiscount = "10";
92                break;
93
94             // values $150 or greater
95             default:
96                strDiscount = "15";
97                break;
98
99          } // end switch
100
101         // display discount to user
102         MessageBox.Show( "Your discount is: " + strDiscount + "%",
103            "Discount", MessageBoxButtons.OK,
104            MessageBoxIcon.Information );
105
106      } // end method btnView_Click
107
108   } // end class FrmDiscountCalculator
109 }
```

Incorrect code given to students omitted this `case`

Programming Challenge

12.17 ***(Enhanced Cash Register Application)*** Modify the **Cash Register** application (Exercise 12.12) to include the addition, subtraction and multiplication operations. Remove the **Enter** `Button`, and replace it with the addition (**+**), subtraction (**–**) and multiplication (*) `Button`s. These `Button`s should take the value displayed in the **Subtotal:** field and the value displayed in the upper `Label` and perform the operation of the clicked `Button`. The result should be displayed in the **Subtotal:** field. Figure 12.21 displays the enhanced **Cash Register** application. [*Note*: You need only deal with positive values. To prevent negative subtotals, set the subtotal to zero if a subtraction would otherwise result in a negative value.]

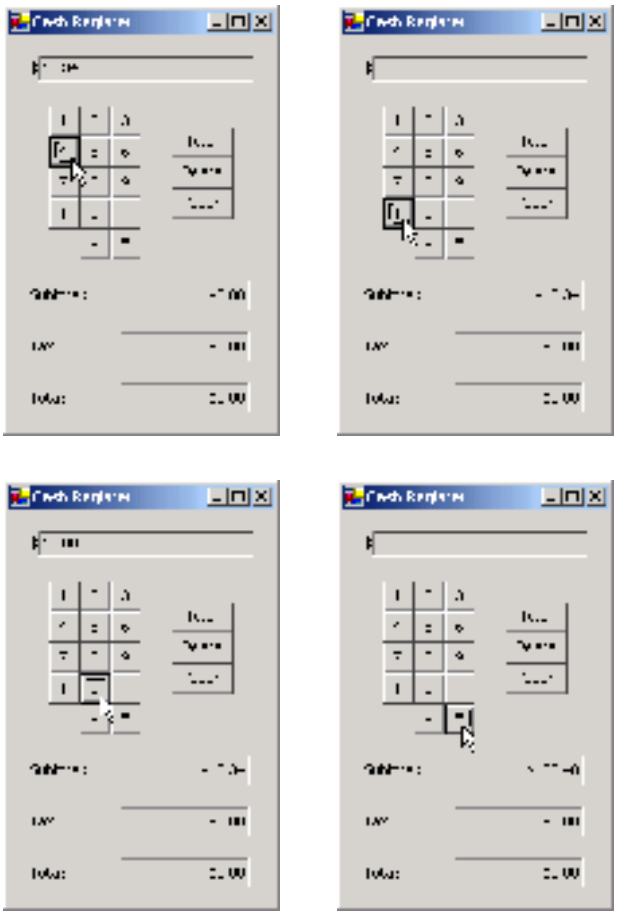


Figure 12.21 **Enhanced Cash Register** application.

Answer:

```
1  // Exercise 12.17 Solution
2  // CashRegister.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace CashRegister
12 {
13    /// <summary>
14    /// Summary description for FrmCashRegister.
15    /// </summary>
```

```
16    public class FrmCashRegister : System.Windows.Forms.Form
17    {
18       // Label and TextBox to display current price
19       private System.Windows.Forms.Label lblDollarSign;
20       private System.Windows.Forms.TextBox txtCurrentPrice;
21
22       // Buttons to input prices
23       private System.Windows.Forms.Button btnOne;
24       private System.Windows.Forms.Button btnTwo;
25       private System.Windows.Forms.Button btnThree;
26       private System.Windows.Forms.Button btnFour;
27       private System.Windows.Forms.Button btnFive;
28       private System.Windows.Forms.Button btnSix;
29       private System.Windows.Forms.Button btnSeven;
30       private System.Windows.Forms.Button btnEight;
31       private System.Windows.Forms.Button btnNine;
32       private System.Windows.Forms.Button btnZero;
33       private System.Windows.Forms.Button btnPoint;
34
35       // Buttons to add, subtract or multiply the current
36       // value to, from or by the subtotal, respectively
37       private System.Windows.Forms.Button btnPlus;
38       private System.Windows.Forms.Button btnMinus;
39       private System.Windows.Forms.Button btnMultiply;
40
41       // Buttons to calculate the total, clear the price
42       // in the TextBox and clear all input, respectively
43       private System.Windows.Forms.Button btnEnter;
44       private System.Windows.Forms.Button btnTotal;
45       private System.Windows.Forms.Button btnDelete;
46       private System.Windows.Forms.Button btnClear;
47
48       // Labels to display the subtotal
49       private System.Windows.Forms.Label lblSubtotal;
50       private System.Windows.Forms.Label lblSubTotalValue;
51
52       // Labels to display the tax
53       private System.Windows.Forms.Label lblTax;
54       private System.Windows.Forms.Label lblTaxValue;
55
56       // Labels to display the total price
57       private System.Windows.Forms.Label lblTotal;
58       private System.Windows.Forms.Label lblTotalValue;
59
60       /// <summary>
61       /// Required designer variable.
62       /// </summary>
63       private System.ComponentModel.Container components = null;
64
65       public FrmCashRegister()
66       {
67          //
68          // Required for Windows Form Designer support
69          //
70          InitializeComponent();
71          //
72          // TODO: Add any constructor code after InitializeComponent
73          // call
74          //
75       }
```

```
76
77          /// <summary>
78          /// Clean up any resources being used.
79          /// </summary>
80          protected override void Dispose( bool disposing )
81          {
82             if( disposing )
83             {
84                if (components != null)
85                {
86                   components.Dispose();
87                }
88             }
89             base.Dispose( disposing );
90          }
91
92          // Windows Form Designer generated code
93
94          /// <summary>
95          /// The main entry point for the application.
96          /// </summary>
97          [STAThread]
98          static void Main()
99          {
100            Application.Run( new FrmCashRegister() );
101         }
102
103         // invoked when Button 1 is clicked
104         private void btnOne_Click(
105            object sender, System.EventArgs e )
106         {
107            txtCurrentPrice.Text += "1";
108
109         } // end method btnOne_Click
110
111         // invoked when Button 2 is clicked
112         private void btnTwo_Click(
113            object sender, System.EventArgs e )
114         {
115            txtCurrentPrice.Text += "2";
116
117         } // end method btnTwo_Click
118
119         // invoked when Button 3 is clicked
120         private void btnThree_Click(
121            object sender, System.EventArgs e )
122         {
123            txtCurrentPrice.Text += "3";
124
125         } // end method btnThree_Click
126
127         // invoked when Button 4 is clicked
128         private void btnFour_Click(
129            object sender, System.EventArgs e )
130         {
131            txtCurrentPrice.Text += "4";
132
133         } // end method btnFour_Click
134
135         // invoked when Button 5 is clicked
```

```
136         private void btnFive_Click(
137            object sender, System.EventArgs e )
138         {
139            txtCurrentPrice.Text += "5";
140
141         } // end method btnFive_Click
142
143         // invoked when Button 6 is clicked
144         private void btnSix_Click(
145            object sender, System.EventArgs e )
146         {
147            txtCurrentPrice.Text += "6";
148
149         } // end method btnSix_Click
150
151         // invoked when Button 7 is clicked
152         private void btnSeven_Click(
153            object sender, System.EventArgs e )
154         {
155            txtCurrentPrice.Text += "7";
156
157         } // end method btnSeven_Click
158
159         // invoked when Button 8 is clicked
160         private void btnEight_Click(
161            object sender, System.EventArgs e )
162         {
163            txtCurrentPrice.Text += "8";
164
165         } // end method btnEight_Click
166
167         // invoked when Button 9 is clicked
168         private void btnNine_Click(
169            object sender, System.EventArgs e )
170         {
171            txtCurrentPrice.Text += "9";
172
173         } // end method btnNine_Click
174
175         // invoked when Button 0 is clicked
176         private void btnZero_Click(
177            object sender, System.EventArgs e )
178         {
179            txtCurrentPrice.Text += "0";
180
181         } // end method btnZero_Click
182
183         // invoked when Button . is clicked
184         private void btnPoint_Click(
185            object sender, System.EventArgs e )
186         {
187            txtCurrentPrice.Text += ".";
188
189         } // end method btnPoint_Click
190
191         // invoke when Total Button is clicked
192         private void btnTotal_Click(
193            object sender, System.EventArgs e )
194         {
195            double dblTaxRate;
```

```
196          decimal decSubTotal =
197             Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) );
198
199          // determines tax rate based on subtotal
200          switch ( ( int ) decSubTotal / 100 )
201          {
202             // amounts in the range $0-99
203             case 0:
204                dblTaxRate = 0.05;
205                break;
206
207             // amounts in the range $100-499
208             case 1:
209             case 2:
210             case 3:
211             case 4:
212                dblTaxRate = 0.075;
213                break;
214
215             // amounts $500 and above
216             default:
217                dblTaxRate = 0.1;
218                break;
219
220          } // end switch
221
222          // display tax and total
223          lblTaxValue.Text = String.Format( "{0:C}",
224             decSubTotal * ( decimal ) dblTaxRate );
225
226          lblTotalValue.Text = String.Format( "{0:C}",
227             decSubTotal * ( decimal ) ( 1 + dblTaxRate ) );
228
229       } // end method btnTotal_Click
230
231       // invoke when Delete Button is clicked
232       private void btnDelete_Click(
233          object sender, System.EventArgs e )
234       {
235          txtCurrentPrice.Clear(); // clear the TextBox
236
237       } // end method btnDelete_Click
238
239       // invoke when Clear Button is clicked
240       private void btnClear_Click(
241          object sender, System.EventArgs e )
242       {
243          // clear txtCurrentPrice and set remaining Labels to $0.00
244          txtCurrentPrice.Clear();
245          lblSubTotalValue.Text = "$0.00";
246          lblTaxValue.Text = "$0.00";
247          lblTotalValue.Text = "$0.00";
248
249       } // end method btnClear_Click
250
251       // invoked when + Button is invoked
252       private void btnPlus_Click(
253          object sender, System.EventArgs e )
254       {
255          decimal decAmount;
```

```
256          decimal decSubTotal;
257
258          // store txtCurrentPrice value for adding
259          decAmount = Decimal.Parse( txtCurrentPrice.Text );
260
261          // store lblSubTotalValue for adding
262          decSubTotal =
263             Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) );
264
265          // add decAmount to decSubTotal
266          lblSubTotalValue.Text = String.Format( "{0:C}",
267             decSubTotal + decAmount );
268
269          txtCurrentPrice.Clear(); // clear the TextBox
270
271       } // end method btnPlus_Click
272
273       // invoked when - Button is invoked
274       private void btnMinus_Click(
275          object sender, System.EventArgs e )
276       {
277          decimal decAmount;
278          decimal decSubTotal;
279
280          // store txtCurrentPrice value for subtracting
281          decAmount = Decimal.Parse( txtCurrentPrice.Text );
282
283          // store lblSubTotalValue for subtracting
284          decSubTotal =
285             Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) );
286
287          // check if subtotal would be less than zero
288          if ( decAmount > decSubTotal )
289          {
290             lblSubTotalValue.Text = String.Format( "{0:C}", 0 );
291          }
292          else
293          {
294             // subtract decAmount from lblSubTotalValue
295             lblSubTotalValue.Text = String.Format( "{0:C}",
296                decSubTotal - decAmount );
297          }
298
299          txtCurrentPrice.Clear(); // clear the TextBox
300
301       } // end method btnMinus_Click
302
303       // invoked when * Button is invoked
304       private void btnMultiply_Click(
305          object sender, System.EventArgs e )
306       {
307          decimal decAmount;
308          decimal decSubTotal;
309
310          // store txtCurrentPrice value for multiplying
311          decAmount = Decimal.Parse( txtCurrentPrice.Text );
312
313          // store lblSubTotalValue for multiplying
314          decSubTotal =
315             Decimal.Parse( lblSubTotalValue.Text.Substring( 1 ) );
```

```
316
317          // multiply decAmount by decSubTotal
318          lblSubTotalValue.Text = String.Format( "{0:C}",
319             decSubTotal * decAmount );
320
321          txtCurrentPrice.Clear(); // clear the TextBox
322
323       } // end method btnMultiply_Click
324
325    } // end class FrmCashRegister
326 }
```

13 TUTORIAL

Enhancing the Wage Calculator Application

Introducing Methods

Solutions

Instructor's Manual Exercise Solutions Tutorial 13

MULTIPLE-CHOICE QUESTIONS

13.1 A method defined with keyword `void` _____.

a) must specify a return type
b) does not accept arguments
c) returns a value
d) does not return a value

13.2 The technique of developing large applications from small, manageable pieces is known as _____.

a) divide and conquer
b) returning a value
c) click and mortar
d) a building-block algorithm

13.3 What is the difference between `void` and non-`void` methods?

a) `void` methods return values, non-`void` methods do not.
b) non-`void` methods return values, `void` methods do not.
c) `void` methods accept parameters, non-`void` methods do not.
d) non-`void` methods accept parameters, `void` methods do not.

13.4 What occurs after a method call is made?

a) Control is given to the called method. After the method is run, the application continues execution at the point where the method call was made.
b) Control is given to the called method. After the method is run, the application continues execution with the statement after the called method's declaration.
c) The statement before the method call is executed.
d) The application terminates.

13.5 Methods can return _____ value(s).

a) zero or one
b) exactly one
c) one or more
d) any number of

13.6 Which of the following must be true when making a method call?

a) The number of arguments in the method call must match the number of parameters in the method header.
b) The argument types must be compatible with their corresponding parameter types.
c) Both a and b.
d) None of the above.

13.7 Which of the following statements correctly returns the variable `intValue` from a method?

a) `return int intValue;`
b) `return void intValue;`
c) `intValue return;`
d) `return intValue;`

13.8 The Step Into and _____ `Buttons` execute the next statement in the application.

a) Step Onto
b) Step Out
c) Step Over
d) Steps

13.9 The first line of a method (including the return type, the method name and the parameter list) is known as the method _____.

a) body
b) title
c) caller
d) header

13.10 `Math` method _____ calculates the square root of the value passed as an argument.

a) `SquareRoot`
b) `Root`
c) `Sqrt`
d) `Square`

Answers: 13.1) d. 13.2) a. 13.3) b. 13.4) a. 13.5) a. 13.6) c. 13.7) d. 13.8) c. 13.9) d. 13.10) c.

EXERCISES

13.11 (*Temperature Conversion Application*) Write an application that performs temperature conversions (Fig. 13.27). The application should be capable of converting from degrees Fahrenheit to degrees Celsius and degrees Celsius to degrees Fahrenheit.

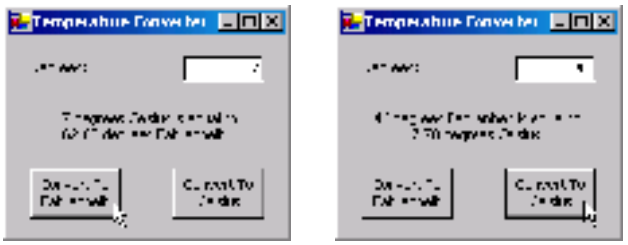


Figure 13.27 Temperature Conversion GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial13\Exercises\TemperatureConversion` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `TemperatureConversion.sln` in the `TemperatureConversion` directory to open the application.

c) ***Convert Fahrenheit to Celsius.*** To convert degrees Fahrenheit to degrees Celsius, use this formula:

```
dblCelsius = ( 5.0 / 9 ) * ( dblFahrenheit - 32 );
```

d) ***Convert Celsius to Fahrenheit.*** To convert degrees Celsius to degrees Fahrenheit, use this formula:

```
dblFahrenheit = ( 9.0 / 5 ) * dblCelsius + 32;
```

e) ***Adding event handlers to your application.*** Double click each `Button` to add the proper event handlers to your application. These event handlers will call methods (that you will define in the next step) to convert the degrees entered to either Fahrenheit or Celsius. Each event handler will display the result in the application's output `Label`.

f) ***Adding methods to your application.*** Create methods to perform each conversion, using the formulas above. The user should provide the temperature to convert.

g) ***Formatting the temperature output.*** To format the temperature information, use the `String.Format` method. Use `F` as the formatting code to limit the temperature to two decimal places.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a temperature value. Click the **Convert to Fahrenheit** `Button` and verify that correct output is displayed based on the formula given. Click the **Convert to Celsius** `Button` and again verify that the output is correct.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 13.11 Solution
2  // TemperatureConversion.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace TemperatureConversion
```

```
{
   /// <summary>
   /// Summary description for FrmTemperatureConversion.
   /// </summary>
   public class FrmTemperatureConversion : System.Windows.Forms.Form
   {
      // Label and TextBox to input degrees
      private System.Windows.Forms.Label lblDegrees;
      private System.Windows.Forms.TextBox txtDegrees;

      // Label to output converted degrees
      private System.Windows.Forms.Label lblOutput;

      // Button to convert to degrees Fahrenheit
      private System.Windows.Forms.Button btnConvertFahrenheit;

      // Button to convert to degrees Celsius
      private System.Windows.Forms.Button btnConvertCelsius;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmTemperatureConversion()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmTemperatureConversion() );
```

```
72          }
73
74          // converts degrees to Fahrenheit
75          private void btnConvertFahrenheit_Click(
76             object sender, System.EventArgs e )
77          {
78             double dblDegree = Double.Parse( txtDegrees.Text );
79
80             lblOutput.Text = dblDegree +
81                " degrees Celsius is equal to \r\n" +
82                String.Format( "{0:F}",
83                ConvertToFahrenheit( dblDegree ) ) +
84                " degrees Fahrenheit.";
85
86          } // end method btnConvertFahrenheit_Click
87
88          // converts degrees to Celsius
89          private void btnConvertCelsius_Click(
90             object sender, System.EventArgs e )
91          {
92             double dblDegree = Double.Parse( txtDegrees.Text );
93
94             lblOutput.Text = dblDegree +
95                " degrees Fahrenheit is equal to \r\n" +
96                String.Format( "{0:F}",
97                ConvertToCelsius( dblDegree ) ) +
98                " degrees Celsius.";
99
100         } // end method btnConvertCelsius_Click
101
102         // convert degree to Fahrenheit
103         double ConvertToFahrenheit( double dblDegree )
104         {
105            return ( 9.0 / 5 ) * dblDegree + 32;
106
107         } // end method ConvertToFahrenheit
108
109         // convert degree to Celsius
110         double ConvertToCelsius( double dblDegree )
111         {
112            return ( 5.0 / 9 ) * ( dblDegree - 32 );
113
114         } // end method ConvertToCelsius
115
116      } // end class FrmTemperatureConversion
117   }
```

13.12 ***(Display Square Application)*** Write an application that displays a solid square composed of a character input by the user (Fig. 13.28). The user also should input the size.

Figure 13.28 **Display Square** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial13\Exercises\DisplaySquare` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `DisplaySquare.sln` in the `DisplaySquare` directory to open the application.

c) ***Adding a method.*** Write a method `DisplaySquare` of `void` return type to display the solid square. The size should be specified by the `int` parameter `intSize`. The character that fills the square should be specified by the `string` parameter `strFillCharacter`. You should use a `for` statement nested within another `for` statement to create the square. The outer `for` specifies what row is currently being displayed. The inner `for` appends all the characters that form the row to a display `string`.

d) ***Adding an event handler for your `Button`'s `Click` event.*** Double click the **Display Square** `Button` to create the event handler. Program the event handler to call method `DisplaySquare`.

e) ***Displaying the output.*** Use the multiline `TextBox` provided to display the square. For example, if `intSize` is `8` and `strFillCharacter` is `#`, the application should look similar to Fig. 13.28.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter a size for the square (the length of each side) and a fill character. Click the **Display Square** `Button`. A square should be displayed of the size you specified, using the character you specified.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 13.12 Solution
2  // DisplaySquare.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace DisplaySquare
12 {
13    /// <summary>
14    /// Summary description for FrmDisplaySquare.
15    /// </summary>
16    public class FrmDisplaySquare : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input size of side
19       private System.Windows.Forms.Label lblSideSize;
20       private System.Windows.Forms.TextBox txtSideSize;
```

```
21
22          // Label and TextBox to input character to fill square with
23          private System.Windows.Forms.Label lblFillCharacter;
24          private System.Windows.Forms.TextBox txtFillCharacter;
25
26          // Button to display the square
27          private System.Windows.Forms.Button btnDisplaySquare;
28
29          // TextBox to display the square
30          private System.Windows.Forms.TextBox txtOutput;
31
32          /// <summary>
33          /// Required designer variable.
34          /// </summary>
35          private System.ComponentModel.Container components = null;
36
37          public FrmDisplaySquare()
38          {
39             //
40             // Required for Windows Form Designer support
41             //
42             InitializeComponent();
43             //
44             // TODO: Add any constructor code after InitializeComponent
45             // call
46             //
47          }
48
49          /// <summary>
50          /// Clean up any resources being used.
51          /// </summary>
52          protected override void Dispose( bool disposing )
53          {
54             if( disposing )
55             {
56                if (components != null)
57                {
58                   components.Dispose();
59                }
60             }
61             base.Dispose( disposing );
62          }
63
64          // Windows Form Designer generated code
65
66          /// <summary>
67          /// The main entry point for the application.
68          /// </summary>
69          [STAThread]
70          static void Main()
71          {
72             Application.Run( new FrmDisplaySquare() );
73          }
74
75          // display square in TextBox
76          void DisplaySquare(
77             int intSize, string strFillCharacter )
78          {
79             // declare loop variables
80             int intRow;             // number of rows counter
```

```
81          int intColumn;          // number columns counter
82          string strOutput = "";  // output string
83
84          // loop until intRow reaches value of
85          // first argument (intSize)
86          for ( intRow = 1; intRow <= intSize; intRow++ )
87          {
88             // loop until intColumn reaches value of intSize
89             for ( intColumn = 1; intColumn <= intSize; intColumn++ )
90             {
91                strOutput += strFillCharacter;
92             }
93
94             strOutput += "\r\n"; // add line to output
95          }
96
97          txtOutput.Text = strOutput; // display square in output area
98
99       } // end method DisplaySquare
100
101      // handles Display Square Button's Click event
102      private void btnDisplaySquare_Click(
103         object sender, System.EventArgs e )
104      {
105         // if valid input is entered
106         if ( txtSideSize.Text != "" &&
107            txtFillCharacter.Text != "" )
108         {
109            DisplaySquare( Int32.Parse( txtSideSize.Text ),
110               txtFillCharacter.Text );
111         }
112         else
113         {
114            MessageBox.Show( "Square size and fill character needed",
115               "Incorrect Input", MessageBoxButtons.OK,
116               MessageBoxIcon.Exclamation );
117         }
118
119      } // end method btnDisplaySquare_Click
120
121   } // end class FrmDisplaySquare
122 }
```

13.13 (*Miles Per Gallon Application*) Drivers often want to know the miles per gallon their cars get so they can estimate gasoline costs. Develop an application that allows the user to input the numbers of miles driven and the number of gallons used for a tank of gas (Fig. 13.29).

Figure 13.29 **Miles Per Gallon** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial13\Exercises\MilesPerGallon` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `MilesPerGallon.sln` in the `MilesPerGallon` directory to open the application.

c) ***Calculating the miles per gallon***. Write a method `MilesPerGallon` that takes the number of miles driven and gallons used (entered by the user), calculates the amount of miles per gallon and returns the miles per gallon for a tankful of gas.

d) ***Displaying the result***. Create a `Click` event handler for the **Calculate MPG** `Button` that invokes the method `MilesPerGallon` and displays the result returned from the method.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter a value for the number of miles driven and the amount of gallons used. Click the **Calculate MPG** `Button` and verify that the correct output is displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 13.13 Solution
// MilesPerGallon.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace MilesPerGallon
{
   /// <summary>
   /// Summary description for FrmMilesPerGallon.
   /// </summary>
   public class FrmMilesPerGallon : System.Windows.Forms.Form
   {
      // Label and TextBox to input miles driven
      private System.Windows.Forms.Label lblMilesDriven;
      private System.Windows.Forms.TextBox txtMilesDriven;

      // Label and TextBox to input gallons used
      private System.Windows.Forms.Label lblGallonsUsed;
      private System.Windows.Forms.TextBox txtGallonsUsed;

      // Labels to output miles per gallon
      private System.Windows.Forms.Label lblMilesPerGallon;
      private System.Windows.Forms.Label lblOutputValue;

      // Button to calculate miles per gallon
      private System.Windows.Forms.Button btnCalculateMPG;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmMilesPerGallon()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
```

```
46          // call
47          //
48       }
49
50       /// <summary>
51       /// Clean up any resources being used.
52       /// </summary>
53       protected override void Dispose( bool disposing )
54       {
55          if( disposing )
56          {
57             if (components != null)
58             {
59                components.Dispose();
60             }
61          }
62          base.Dispose( disposing );
63       }
64
65       // Windows Form Designer generated code
66
67       /// <summary>
68       /// The main entry point for the application.
69       /// </summary>
70       [STAThread]
71       static void Main()
72       {
73          Application.Run( new FrmMilesPerGallon() );
74       }
75
76       // calculate and return miles per gallon
77       double MilesPerGallon(
78          double dblMilesDriven, double dblGallonsUsed )
79       {
80          return dblMilesDriven / dblGallonsUsed;
81
82       } // end method MilesPerGallon
83
84       // handles Calculate Button's Click event
85       private void btnCalculateMPG_Click(
86          object sender, System.EventArgs e )
87       {
88          // display miles per gallon
89          lblOutputValue.Text = String.Format( "{0:F}",
90             MilesPerGallon( Double.Parse( txtMilesDriven.Text ),
91             Double.Parse( txtGallonsUsed.Text ) ) );
92
93       } // end method btnCalculateMPG_Click
94
95    } // end class FrmMilesPerGallon
96 }
```

What does this code do?

13.14 What does the following code do? Assume this method is invoked by using `Mystery ( 70, 80 )`.

```
1  void Mystery( int intNumber1, int intNumber2 )
2  {
3     int intX;
```

```
   double dblY;

   intX = intNumber1 + intNumber2;
   dblY = intX / 2;

   if ( dblY <= 60 )
   {
      lblResult.Text = "<= 60 ";
   }
   else
   {
      lblResult.Text = " Result is " + dblY;
   }

} // end method Mystery
```

Answer: This code calculates the average of two numbers. The user is informed of the average, but only if the average is above 60. In this case the two numbers entered are 70 and 80, which results in an average of 75, which will be displayed. The complete code reads:

```
// Exercise 13.14 Solution
// Average.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Average
{
   /// <summary>
   /// Summary description for FrmAverage.
   /// </summary>
   public class FrmAverage : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblNumber2Out;
      private System.Windows.Forms.Label lblNumber1Out;
      private System.Windows.Forms.Label lblResultText;
      private System.Windows.Forms.Label lblNumber2;
      private System.Windows.Forms.Label lblNumber1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmAverage()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();
         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }
```

```
40
41          /// <summary>
42          /// Clean up any resources being used.
43          /// </summary>
44          protected override void Dispose( bool disposing )
45          {
46             if( disposing )
47             {
48                if (components != null)
49                {
50                   components.Dispose();
51                }
52             }
53             base.Dispose( disposing );
54          }
55
56          // Windows Form Designer generated code
57
58          /// <summary>
59          /// The main entry point for the application.
60          /// </summary>
61          [STAThread]
62          static void Main()
63          {
64             Application.Run( new FrmAverage() );
65          }
66
67          // calls the Mystery method to perform a calculation
68          private void FrmAverage_Load(
69             object sender, System.EventArgs e )
70          {
71             Mystery( 70, 80 );
72
73          } // end method FrmAverage_Load
74
75          // tests whether the average of the inputs is <= 60
76          void Mystery( int intNumber1, int intNumber2 )
77          {
78             int intX;
79             double dblY;
80
81             intX = intNumber1 + intNumber2;
82             dblY = intX / 2;
83
84             if ( dblY <= 60 )
85             {
86                lblResult.Text = "<= 60 ";
87             }
88             else
89             {
90                lblResult.Text = " Result is " + dblY;
91             }
92
93          } // end method Mystery
94
95       } // end class FrmAverage
96    }
```

What's wrong with this code?

13.15 Find the error(s) in the following code, which should take an `int` value as an argument and `return` the value of that argument multiplied by two.

```
1  int TimesTwo( int intNumber )
2  {
3     int intResult;
4
5     intResult = intNumber * 2;
6
7  } // end method TimesTwo
```

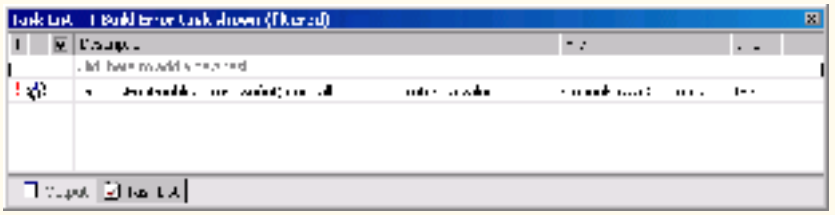
Answer: A non-`void` method should `return` a value. If you do not include a `return` statement, it is a syntax error. The complete incorrect code reads:

```
1  // Exercise 13.15 Solution
2  // Double.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Double
12 {
13    /// <summary>
14    /// Summary description for FrmDouble.
15    /// </summary>
16    public class FrmDouble : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Button btnDouble;
19       private System.Windows.Forms.TextBox txtInput;
20       private System.Windows.Forms.Label lblDoubled;
21       private System.Windows.Forms.Label lblDoubledText;
22       private System.Windows.Forms.Label lblInputText;
23       /// <summary>
24       /// Required designer variable.
25       /// </summary>
26       private System.ComponentModel.Container components = null;
27
28       public FrmDouble()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34          //
35          // TODO: Add any constructor code after InitializeComponent
```

```
36            // call
37            //
38         }
39
40         /// <summary>
41         /// Clean up any resources being used.
42         /// </summary>
43         protected override void Dispose( bool disposing )
44         {
45            if( disposing )
46            {
47               if (components != null)
48               {
49                  components.Dispose();
50               }
51            }
52            base.Dispose( disposing );
53         }
54
55         // Windows Form Designer generated code
56
57         /// <summary>
58         /// The main entry point for the application.
59         /// </summary>
60         [STAThread]
61         static void Main()
62         {
63            Application.Run( new FrmDouble() );
64         }
65
66         // prints the result of helper method TimesTwo
67         private void btnDouble_Click(
68            object sender, System.EventArgs e )
69         {
70            int intInputNumber;
71
72            intInputNumber = Int32.Parse( txtInput.Text );
73
74            lblDoubled.Text =
75               Convert.ToString( TimesTwo( intInputNumber ) );
76
77         } // end method btnDouble_Click
78
79         // returns the input multiplied by 2
80         int TimesTwo( int intNumber )
81         {
82            int intResult;
83
84            intResult = intNumber * 2;
85
86         } // end method TimesTwo
87
88      } // end class FrmDouble
89   }
```

`TimesTwo` is declared as returning an `int` value

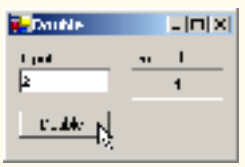
There is no `return` statement in the method



Answer: The complete corrected application should read:

```
1  // Exercise 13.15 Solution
2  // Double.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Double
12 {
13    /// <summary>
14    /// Summary description for FrmDouble.
15    /// </summary>
16    public class FrmDouble : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Button btnDouble;
19       private System.Windows.Forms.TextBox txtInput;
20       private System.Windows.Forms.Label lblDoubled;
21       private System.Windows.Forms.Label lblDoubledText;
22       private System.Windows.Forms.Label lblInputText;
23       /// <summary>
24       /// Required designer variable.
25       /// </summary>
26       private System.ComponentModel.Container components = null;
27
28       public FrmDouble()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34          //
35          // TODO: Add any constructor code after InitializeComponent
36          // call
37          //
38       }
39
40       /// <summary>
41       /// Clean up any resources being used.
42       /// </summary>
43       protected override void Dispose( bool disposing )
44       {
45          if( disposing )
```

```
46          {
47             if (components != null)
48             {
49                components.Dispose();
50             }
51          }
52          base.Dispose( disposing );
53       }
54
55       // Windows Form Designer generated code
56
57       /// <summary>
58       /// The main entry point for the application.
59       /// </summary>
60       [STAThread]
61       static void Main()
62       {
63          Application.Run( new FrmDouble() );
64       }
65
66       // prints the result of helper method TimesTwo
67       private void btnDouble_Click(
68          object sender, System.EventArgs e)
69       {
70          int intInputNumber;
71
72          intInputNumber = Int32.Parse( txtInput.Text ) ;
73
74          lblDoubled.Text =
75             Convert.ToString( TimesTwo( intInputNumber ) );
76
77       } // end method btnDouble_Click
78
79       // returns the input multiplied by 2
80       int TimesTwo( int intNumber )
81       {
82          int intResult;
83
84          intResult = intNumber * 2;
85
86          return intResult;
87
88       } // end method TimesTwo
89
90    } // end class FrmDouble
91 }
```



Using the Debugger ▶

13.16 (*Gas Pump Application*) The **Gas Pump** application calculates the cost of gas at a local gas station (Fig. 13.30). This gas station charges $1.41 per gallon for **Regular** grade gas, $1.47 per gallon for **Special** grade gas and $1.57 per gallon for **Super+** grade gas. The user enters the number of gallons to purchase and clicks the desired grade. The application calls a

method to compute the total cost from the number of gallons entered and the selected grade. While testing your application, you noticed that one of your totals was incorrect, given the input.

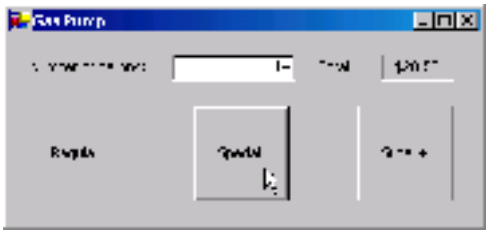


Figure 13.30 **Gas Pump** application running correctly.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial13\Exercises\Debugger\GasPump` to your `C:\SimplyCSP` directory.

b) ***Opening the incorrect application.*** Double click `GasPump.sln` in the `GasPump` directory to open the application.

c) ***Running the application***. Run the application and determine which total is incorrect.

d) ***Setting a breakpoint***. Set a breakpoint at the beginning of the event handler that is providing incorrect output. For instance, if the **Regular** `Button` is providing incorrect output when clicked, add a breakpoint at the beginning of that `Button`'s `Click` event handler. Use the debugger to help find any logic error(s) in the application.

e) ***Modifying the application***. Once you have located the error(s), modify the application so that it behaves correctly.

f) ***Running the corrected application.*** Select **Debug > Start** to run your application. Enter a number of gallons and click the **Regular**, **Special** and **Super+** `Button`s. After each `Button` is clicked, verify that the total displayed is correct based on the prices given in this exercise's description.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 13.16 Solution
// GasPump.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace GasPump
{
   /// <summary>
   /// Summary description for FrmGasPump.
   /// </summary>
   public class FrmGasPump : System.Windows.Forms.Form
   {
      // Label and TextBox to input number of gallons
      private System.Windows.Forms.Label lblNumberGallons;
      private System.Windows.Forms.TextBox txtNumberGallons;

      // Labels to display total cost
      private System.Windows.Forms.Label lblTotal;
      private System.Windows.Forms.Label lblTotalResult;

      // Buttons to choose regular, special or super+ gas
```

```
27          private System.Windows.Forms.Button btnRegular;
28          private System.Windows.Forms.Button btnSpecial;
29          private System.Windows.Forms.Button btnSuper;
30
31          /// <summary>
32          /// Required designer variable.
33          /// </summary>
34          private System.ComponentModel.Container components = null;
35
36          public FrmGasPump()
37          {
38             //
39             // Required for Windows Form Designer support
40             //
41             InitializeComponent();
42             //
43             // TODO: Add any constructor code after InitializeComponent
44             // call
45             //
46          }
47
48          /// <summary>
49          /// Clean up any resources being used.
50          /// </summary>
51          protected override void Dispose( bool disposing )
52          {
53             if( disposing )
54             {
55                if (components != null)
56                {
57                   components.Dispose();
58                }
59             }
60             base.Dispose( disposing );
61          }
62
63          // Windows Form Designer generated code
64
65          /// <summary>
66          /// The main entry point for the application.
67          /// </summary>
68          [STAThread]
69          static void Main()
70          {
71             Application.Run( new FrmGasPump() );
72          }
73
74          // handles Regular Button's Click Event
75          private void btnRegular_Click(
76             object sender, System.EventArgs e )
77          {
78             int intGallons; // number of gallons
79
80             intGallons = Int32.Parse( txtNumberGallons.Text );
81
82             // call method to determine total
83             Total( btnRegular.Text, intGallons );
84
85          } // end method btnRegular_Click
86
```

```
87          // handles Special Button's Click Event
88          private void btnSpecial_Click(
89             object sender, System.EventArgs e )
90          {
91             int intGallons; // number of gallons
92
93             intGallons = Int32.Parse( txtNumberGallons.Text );
94
95             // call method to determine total
96             Total( btnSpecial.Text, intGallons );
97
98          } // end method btnSpecial_Click
99
100         // handles Super+ Button's Click Event
101         private void btnSuper_Click(
102            object sender, System.EventArgs e )
103         {
104            int intGallons; // number of gallons
105
106            intGallons = Int32.Parse( txtNumberGallons.Text );
107
108            // call method to determine total
109            Total( btnSuper.Text, intGallons );
110
111         } // end method btnSuper_Click
112
113         void Total( string strGrade, int intGallons )
114         {
115            // determine grade selected and output total
116            switch ( strGrade )
117            {
118               case "Regular":
119                  lblTotalResult.Text =
120                     String.Format( "{0:C}", 1.41 * intGallons );
121                  break;
122
123               case "Special":
124                  lblTotalResult.Text =
125                     String.Format( "{0:C}", 1.47 * intGallons );
126                  break;
127
128               case "Super+":
129                  lblTotalResult.Text =
130                     String.Format( "{0:C}", 1.57 * intGallons );
131                  break;
132            }
133
134         } // end method Total
135
136      } // end class FrmGasPump
137   }
```

Code provided to student had value `1.91` in place of `1.47`

Programming Challenge

13.17 ***(Prime Numbers Application)*** An `int` greater than 1 is said to be prime if it is divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime numbers, but 4, 6, 8 and 9 are not. Write an application that takes two numbers (representing a lower bound and an upper bound) and determines all the prime numbers within the specified bounds, inclusive (Fig. 13.31).

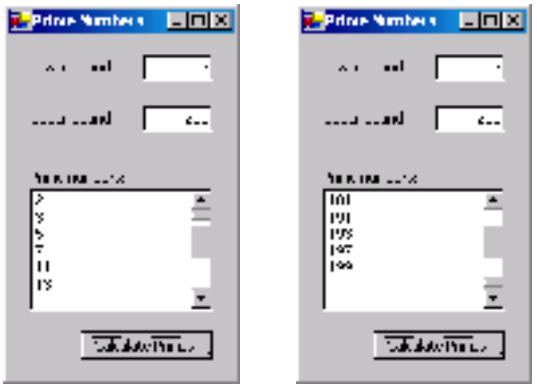


Figure 13.31 Prime Numbers application.

a) ***Creating the application***. Create an application named `PrimeNumbers` in your `C:\SimplyCSP` directory and have its GUI appear as shown in Fig. 13.31. Rearrange and comment the control declarations appropriately. Add an event handler for the **Calculate Primes** `Button`'s `Click` event.

b) ***Checking for prime numbers***. Write a method `Prime` that returns `true` if a number is prime, `false` otherwise.

c) ***Limiting user input***. Allow users to enter a lower bound (`intLower`) and an upper bound (`intUpper`). Prevent the user from entering bounds less than `2` (the smallest prime), or an upper bound that is smaller than the lower bound.

d) ***Displaying the prime numbers***. Call method `Prime` from your event handler to determine which numbers between the lower and upper bounds are prime. Then have the event handler display the prime numbers in a multiline, scrollable `TextBox`, as in Fig. 13.31.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter a lower bound and an upper bound that is smaller than the lower bound. Click the **Calculate Primes** `Button`. You should receive an error message. Enter negative bounds and click the **Calculate Primes** `Button`. Again, you should receive an error message. Enter valid bounds and click the **Calculate Primes** Button. This time, the primes within that range should be displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 13.17 Solution
2  // PrimeNumbers.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace PrimeNumbers
12 {
13    /// <summary>
14    /// Summary description for FrmPrimeNumbers.
15    /// </summary>
16    public class FrmPrimeNumbers : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input lower bound for primes
19       private System.Windows.Forms.Label lblLowerBound;
20       private System.Windows.Forms.TextBox txtLowerBound;
21
```

```
22          // Label and TextBox to input upper bound for primes
23          private System.Windows.Forms.Label lblUpperBound;
24          private System.Windows.Forms.TextBox txtUpperBound;
25
26          // Label and TextBox to display the prime numbers
27          private System.Windows.Forms.Label lblPrimeNumbers;
28          private System.Windows.Forms.TextBox txtPrimeNumbers;
29
30          // Button to generate the prime numbers
31          private System.Windows.Forms.Button btnCalculatePrimes;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          public FrmPrimeNumbers()
39          {
40             //
41             // Required for Windows Form Designer support
42             //
43             InitializeComponent();
44             //
45             // TODO: Add any constructor code after InitializeComponent
46             // call
47             //
48          }
49
50          /// <summary>
51          /// Clean up any resources being used.
52          /// </summary>
53          protected override void Dispose( bool disposing )
54          {
55             if( disposing )
56             {
57                if (components != null)
58                {
59                   components.Dispose();
60                }
61             }
62             base.Dispose( disposing );
63          }
64
65          // Windows Form Designer generated code
66
67          /// <summary>
68          /// The main entry point for the application.
69          /// </summary>
70          [STAThread]
71          static void Main()
72          {
73             Application.Run( new FrmPrimeNumbers() );
74          }
75
76          // determine if number is prime
77          bool Prime( int intNumber )
78          {
79             int intCount; // declare counter
80
81             // set square root of intNumber as limit
```

```
82          int intLimit = ( int ) Math.Sqrt( intNumber );
83
84          // loop until intCount reaches square root of intNumber
85          for ( intCount = 2; intCount <= intLimit; intCount++ )
86          {
87             if ( intNumber % intCount == 0 )
88             {
89                return false; // number is not prime
90             }
91          }
92
93          return true; // number is prime
94
95       } // end method Prime
96
97       // handles Calculate Prime Button's Click event
98       private void btnCalculatePrimes_Click(
99          object sender, System.EventArgs e )
100      {
101         // declare variables
102         int intLowerBound = Int32.Parse( txtLowerBound.Text );
103         int intUpperBound = Int32.Parse( txtUpperBound.Text );
104         int intCounter;
105         string strOutput = "";
106
107         if ( intLowerBound <= 1 || intUpperBound <= 1 )
108         {
109            MessageBox.Show( "Bounds must be greater than 1",
110               "Invalid Bounds", MessageBoxButtons.OK,
111               MessageBoxIcon.Exclamation );
112         }
113         else if ( intUpperBound < intLowerBound )
114         {
115            MessageBox.Show( "Upper bound cannot be less than " +
116               "lower bound", "Invalid Bounds", MessageBoxButtons.OK,
117               MessageBoxIcon.Exclamation );
118         }
119         else
120         {
121            // loop from lower bound to upper bound
122            for ( intCounter = intLowerBound;
123               intCounter <= intUpperBound; intCounter++ )
124            {
125               // if prime number, display in TextBox
126               if ( Prime( intCounter ) == true )
127               {
128                  strOutput += ( intCounter + "\r\n" );
129               }
130            }
131         }
132
133         txtPrimeNumbers.Text = strOutput;
134
135      } // end method btnCalculatePrimes_Click
136
137   } // end class FrmPrimeNumbers
138 }
```

14 TUTORIAL

Shipping Time Application

Using DateTimes and Timers

Solutions

Instructor's Manual Exercises Solutions Tutorial 14

MULTIPLE-CHOICE QUESTIONS

14.1 The _____ allows you to store and manipulate date information easily.

a) `DateTime` structure
b) `DatePicker` control
c) `GroupBox` control
d) `Now` property

14.2 You can _____ to a `DateTime` variable.

a) add hours
b) add days
c) subtract hours
d) All of the above.

14.3 To subtract one day from `DateTime` variable `dtmDay`'s value, assign the value returned by _____ to `dtmDay`.

a) `dtmDay.AddHours( -24 );`
b) `dtmDay.SubtractDays( 1 );`
c) `dtmDay.AddDays( -1 );`
d) Both a and c.

14.4 The time 3:45 and 35 seconds in the afternoon would be formatted as 03:45:35 PM according to the format string _____.

a) `"hh:mm:ss"`
b) `"hh:mm:ss tt"`
c) `"hh:mm:ss am:pm"`
d) `"h:m:s tt"`

14.5 A(n) _____ event occurs before the `Form` is displayed.

a) `LoadForm`
b) `InitializeForm`
c) `Load`
d) `FormLoad`

14.6 `Timer` property `Interval` sets the rate at which `Tick` events occur in _____.

a) nanoseconds
b) microseconds
c) milliseconds
d) seconds

14.7 To set `DateTime` `dtmNow`'s time five hours earlier, use _____.

a) `dtmNow = dtmNow.SubHours( 5 );`
b) `dtmNow = dtmNow.AddHours( -5 );`
c) `dtmNow = dtmNow.AddHours( 5 );`
d) `dtmNow.AddHours( -5 );`

14.8 A _____ is a container.

a) `GroupBox`
b) `Form`
c) `Timer`
d) Both a and b.

14.9 A `DateTime` variable stores hour values in the range _____.

a) `1` to `12`
b) `0` to `12`
c) `0` to `24`
d) `0` to `23`

14.10 A `DateTimePicker`'s _____ property specifies the format string with which to display the date.

a) `CustomFormat`
b) `FormatString`
c) `Format`
d) `Text`

Answers: 14.1) a. 14.2) d. 14.3) d. 14.4) b. 14.5) c. 14.6) c. 14.7) b. 14.8) d. 14.9) d. 14.10) a.

EXERCISES

14.11 (***World Clock Application***) Create an application that displays the current time in Los Angeles, Atlanta, London and Tokyo. Use a `Timer` to update the clock every second. Assume that your local time is the time in Atlanta. Atlanta is three hours later than Los Angeles. London is five hours later than Atlanta. Tokyo is eight hours later than London. The application should look similar to Fig. 14.21.

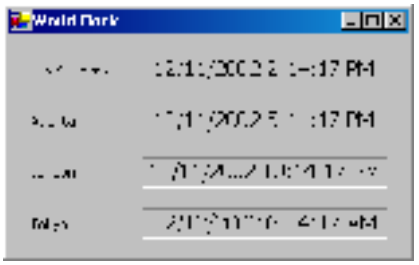


Figure 14.21 World Clock GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial14\Exercises\WorldClock` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `WorldClock.sln` in the `WorldClock` directory to open the application.

c) ***Adding a `Timer` to the `Form`.*** Add a `Timer` control to the **World Clock** application. Set the `Timer` control's name property to `tmrClock`. Rearrange and comment the new control declaration appropriately.

d) ***Adding a `Tick` event handler for `tmrClock`.*** Add a `Tick` event handler for `Timer tmrClock`. The event handler should calculate and display the current times for Los Angeles, Atlanta, London and Tokyo. Use the `DateTime` variable's `ToShortDateString` and `ToLongTimeString` methods to create the display text.

e) ***Running the application.*** Select **Debug > Start** to run your application. Look at the clock on your machine to verify that the time for Los Angeles is three hours earlier, the time in Atlanta is the same as what your clock says, the time in London is five hours later, and the time in Tokyo is 13 hours later (eight hours later than London).

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 14.11 Solution
// WorldClock.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WorldClock
{
   /// <summary>
   /// Summary description for FrmWorldClock.
   /// </summary>
   public class FrmWorldClock : System.Windows.Forms.Form
   {
      // Labels to display the time in Los Angeles
      private System.Windows.Forms.Label lblLA;
      private System.Windows.Forms.Label lblLATime;

      // Labels to display the time in Atlanta
      private System.Windows.Forms.Label lblAtlanta;
      private System.Windows.Forms.Label lblAtlantaTime;

      // Labels to display the time in London
      private System.Windows.Forms.Label lblLondon;
      private System.Windows.Forms.Label lblLondonTime;

```

```
30          // Labels to display the time in Tokyo
31          private System.Windows.Forms.Label lblTokyo;
32          private System.Windows.Forms.Label lblTokyoTime;
33
34          // Timer to keep track of current time
35          private System.Windows.Forms.Timer tmrClock;
36
37          /// <summary>
38          /// Required designer variable.
39          /// </summary>
40          private System.ComponentModel.IContainer components;
41
42          public FrmWorldClock()
43          {
44             //
45             // Required for Windows Form Designer support
46             //
47             InitializeComponent();
48
49             //
50             // TODO: Add any constructor code after InitializeComponent
51             // call
52             //
53          }
54
55          /// <summary>
56          /// Clean up any resources being used.
57          /// </summary>
58          protected override void Dispose( bool disposing )
59          {
60             if( disposing )
61             {
62                if (components != null)
63                {
64                   components.Dispose();
65                }
66             }
67             base.Dispose( disposing );
68          }
69
70          // Windows Form Designer generated code
71
72          /// <summary>
73          /// The main entry point for the application.
74          /// </summary>
75          [STAThread]
76          static void Main()
77          {
78             Application.Run( new FrmWorldClock() );
79          }
80
81          // update times
82          private void tmrClock_Tick(
83             object sender, System.EventArgs e )
84          {
85             // retrieve current time
86             DateTime dtmNow = DateTime.Now;
87
88             // display Los Angeles time
89             lblLATime.Text =
```

```
            dtmNow.AddHours( -3 ).ToShortDateString() +
            " " + dtmNow.AddHours( -3 ).ToLongTimeString();

         // display Atlanta time
         lblAtlantaTime.Text =
            dtmNow.ToShortDateString() +
            " " + dtmNow.ToLongTimeString();

         // display London time
         lblLondonTime.Text =
            dtmNow.AddHours( 5 ).ToShortDateString() +
            " " + dtmNow.AddHours( 5 ).ToLongTimeString();

         // display Tokyo time
         lblTokyoTime.Text =
            dtmNow.AddHours( 13 ).ToShortDateString() +
            " " + dtmNow.AddHours( 13 ).ToLongTimeString();

      } // end method tmrClock_Tick

   } // end class FrmWorldClock
}
```

14.12 (*Shipping Time Application Enhancement*) During the winter, a distribution center in Denver, Colorado needs to receive seafood shipments to supply the local ski resorts. Enhance the **Shipping Time** application by adding Denver, Colorado as another shipping destination (Fig. 14.22). Denver is two time zones west of Portland, meaning time is two hours earlier than Portland, Maine. There are no direct flights to Denver, so shipments from Portland will take 8 hours.

Figure 14.22 Enhanced Shipping Time GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial14\Exercises\ShippingTimeEnhanced` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ShippingTime.sln` in the `ShippingTimeEnhanced` directory to open the application.

c) ***Inserting a GroupBox.*** Resize the `Form` to fit the **Express Shipping to Denver** `GroupBox` as shown in Fig. 14.22. Add a `GroupBox` to the `Form`. Change the `Text` property of the `GroupBox` to indicate that it will contain the delivery time in Denver. Resize and move the `GroupBox` so that it resembles the GUI shown in Fig. 14.22.

d) ***Inserting Labels.*** In the `GroupBox` you just created, add an output `Label` to display the delivery time for a seafood shipment to Denver and a corresponding descriptive `Label`. Rearrange and comment the new control declarations appropriately.

e) ***Inserting code to the `DisplayDeliveryTime` method.*** Add code to `DisplayDeliveryTime` method to compute and display the delivery time in Denver.

f) ***Running the application.*** Select **Debug > Start** to run your application. Select various drop-off times and ensure the delivery times are correct for Las Vegas and Denver.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 14.12 Solution
2  // ShippingTime.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ShippingTime
12 {
13    /// <summary>
14    /// Summary description for FrmShippingTime.
15    /// </summary>
16    public class FrmShippingTime : System.Windows.Forms.Form
17    {
18       // Labels and Timer to display current time
19       private System.Windows.Forms.Label lblCurrentTimeIs;
20       private System.Windows.Forms.Label lblCurrentTime;
21       private System.Windows.Forms.Timer tmrClock;
22
23       // GroupBox for drop-off time
24       private System.Windows.Forms.GroupBox fraDropOff;
25
26       // Label and DateTimePicker to choose drop-off time
27       private System.Windows.Forms.Label lblDropOff;
28       private System.Windows.Forms.DateTimePicker dtpDropOff;
29
30       // GroupBox for delivery time to Las Vegas
31       private System.Windows.Forms.GroupBox fraDeliveryTime;
32
33       // Labels to display delivery time to Las Vegas
34       private System.Windows.Forms.Label lblDeliveryTime;
35       private System.Windows.Forms.Label lblLasVegasTime;
36
37       // GroupBox for delivery time to Denver
38       private System.Windows.Forms.GroupBox fraDenverTime;
39
40       // Labels to display delivery time to Denver
41       private System.Windows.Forms.Label lblDenverTime;
42       private System.Windows.Forms.Label lblDenverDelivery;
43
44       /// <summary>
45       /// Required designer variable.
46       /// </summary>
47       private System.ComponentModel.IContainer components;
48
49       public FrmShippingTime()
50       {
51          //
52          // Required for Windows Form Designer support
```

```
53          //
54          InitializeComponent();
55
56          //
57          // TODO: Add any constructor code after InitializeComponent
58          // call
59          //
60       }
61
62       /// <summary>
63       /// Clean up any resources being used.
64       /// </summary>
65       protected override void Dispose( bool disposing )
66       {
67          if( disposing )
68          {
69             if (components != null)
70             {
71                components.Dispose();
72             }
73          }
74          base.Dispose( disposing );
75       }
76
77       // Windows Form Designer generated code
78
79       /// <summary>
80       /// The main entry point for the application.
81       /// </summary>
82       [STAThread]
83       static void Main()
84       {
85          Application.Run( new FrmShippingTime() );
86       }
87
88       // update current time every second
89       private void tmrClock_Tick(
90          object sender, System.EventArgs e )
91       {
92          // print current time
93          lblCurrentTime.Text =
94             String.Format( "{0:hh:mm:ss tt}", DateTime.Now );
95
96       } // end method tmrClock_Tick
97
98       // initialize DateTimePicker status when Form loads
99       private void FrmShippingTime_Load(
100         object sender, System.EventArgs e )
101      {
102         // store current time
103         DateTime dtmCurrentTime = DateTime.Now;
104
105         // set range of possible drop-off times
106         dtpDropOff.MinDate = new DateTime( dtmCurrentTime.Year,
107            dtmCurrentTime.Month, dtmCurrentTime.Day, 0, 0, 0 );
108
109         dtpDropOff.MaxDate = dtpDropOff.MinDate.AddDays( 1 );
110
111         // display the delivery time
112         DisplayDeliveryTime();
```

```
113
114       } // end method FrmShippingTime_Load
115
116       // update ship time on change of drop-off time
117       void dtpDropOff_ValueChanged(
118          object sender, System.EventArgs e )
119       {
120          // display the delivery time
121          DisplayDeliveryTime();
122
123       } // end method dtpDropOff_ValueChanged
124
125       // calculates and displays the delivery time
126       private void DisplayDeliveryTime()
127       {
128          // print initial delivery time
129          DateTime dtmDelivery = DepartureTime();
130
131          // add 3 hours to departure and display result
132          dtmDelivery = dtmDelivery.AddHours( 3 );
133          lblLasVegasTime.Text = dtmDelivery.ToLongDateString()
134             + " at " + dtmDelivery.ToShortTimeString();
135
136          // add 6 hours to departure and display result
137          dtmDelivery = dtmDelivery.AddHours( 6 );
138          lblDenverTime.Text = dtmDelivery.ToLongDateString()
139             + " at " + dtmDelivery.ToShortTimeString();
140
141       } // end method DisplayDeliveryTime
142
143       // returns flight departure time for selected drop-off time
144       DateTime DepartureTime()
145       {
146          // store current date and departure time
147          DateTime dtmCurrentDate = DateTime.Now;
148          DateTime dtmDepartureTime;
149
150          // determine which flight the shipment takes
151          switch ( dtpDropOff.Value.Hour )
152          {
153             // seafood will be on the noon flight
154             case 1: case 2: case 3: case 4: case 5:
155             case 6: case 7: case 8: case 9: case 10:
156                dtmDepartureTime = new DateTime(
157                   dtmCurrentDate.Year, dtmCurrentDate.Month,
158                   dtmCurrentDate.Day, 12, 0, 0 );
159                break;
160
161             // seafood will be on tomorrow's noon flight
162             case 23:
163                dtmCurrentDate = dtmCurrentDate.AddDays( 1 );
164                dtmDepartureTime = new DateTime(
165                   dtmCurrentDate.Year, dtmCurrentDate.Month,
166                   dtmCurrentDate.Day, 12, 0, 0 );
167                break;
168
169             // seafood will be on midnight flight
170             default:
171                dtmCurrentDate = dtmCurrentDate.AddDays( 1 );
172                dtmDepartureTime = new DateTime(
```

```
                    dtmCurrentDate.Year, dtmCurrentDate.Month,
                    dtmCurrentDate.Day, 0, 0, 0 );
                break;

            } // end switch

            // return the flight's departure time
            return dtmDepartureTime;

        } // end method DepartureTime

    } // end class FrmShippingTime
}
```

14.13 (*Pop-Up Reminder Application*) Create an application that allows the user to set an pop-up reminder (Fig. 14.23). The application should allow the user to set the exact time of that a pop-up message should appear by using a `DateTimePicker`. While the time for the reminder is set, the user should not be able to modify the `DateTimePicker`. If the time for the reminder is set and the current time matches the time in the `DateTimePicker`, display a `MessageBox` as a reminder for the user. The user should be able to cancel a reminder by using a **Reset** `Button`. This `Button` is disabled when the application starts.

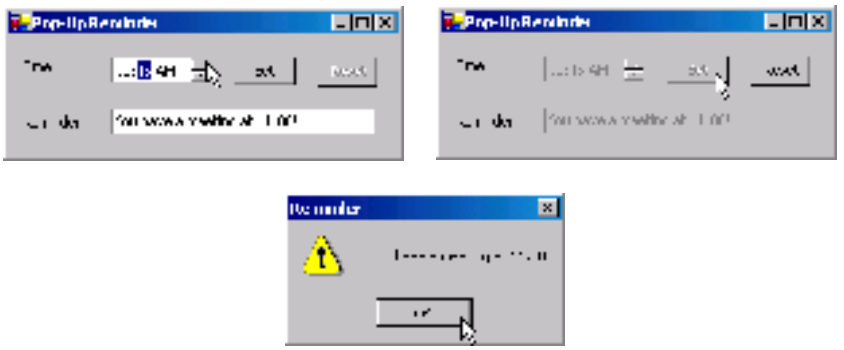


Figure 14.23 **Pop-Up Reminder** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial14\Exercises\Reminder` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Reminder.sln` in the `Reminder` directory to open the application.

c) ***Inserting a `DateTimePicker`.*** Add a `DateTimePicker` control to the `Form`. Set the `DateTimePicker` to display only the time, as is shown in Fig. 14.23. Set the `DateTimePicker` control's `Size` property to `80, 20`, and move the control so that it appears as it does in Fig. 14.23. Rearrange and comment the new control declaration appropriately.

d) ***Coding the Set `Button`'s `Click` event handler.*** Add a `Click` event handler for the **Set** `Button`. This event handler should disable the **Set** `Button`, the **Reminder:** `TextBox` and the `DateTimePicker` and enable the **Reset** `Button`.

e) ***Coding the `Timer`'s `Tick` event handler.*** Define the `Tick` event handler for the `Timer`. A `Tick` event should occur every 1000 milliseconds (one second). If the reminder time is set and the current time matches the time in the `DateTimePicker` (with a seconds value of `0`), display the text from the **Reminder:** `TextBox` in a `MessageBox`.

f) ***Coding the Reset `Button`'s `Click` event handler.*** Define the `Click` event handler for the **Reset** `Button`. When the **Reset** `Button` is clicked, the GUI should be set back to its original state.

g) ***Running the application.*** Select **Debug > Start** to run your application. Use the `DateTimePicker` and the **Set** `Button` to set a time for the reminder to display. Wait

for that time to verify that the `MessageBox` appears. Click the **Reset** `Button` to set a new time for the reminder to display.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 14.13 Solution
// Reminder.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Reminder
{
   /// <summary>
   /// Summary description for FrmReminder.
   /// </summary>
   public class FrmReminder : System.Windows.Forms.Form
   {
      // Label and DateTimePicker to choose the time
      private System.Windows.Forms.Label lblTime;
      private System.Windows.Forms.DateTimePicker dtpReminderTime;

      // Label and TextBox for reminder's text
      private System.Windows.Forms.Label lblReminder;
      private System.Windows.Forms.Label txtReminder;

      // Button to set the reminder for the displayed time
      private System.Windows.Forms.Button btnSetTime;

      // Button to cancel the reminder
      private System.Windows.Forms.Button btnReset;

      // Timer to keep track of reminder time
      private System.Windows.Forms.Timer tmrReminder;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.IContainer components;

      public FrmReminder()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
```

```
55          /// </summary>
56          protected override void Dispose( bool disposing )
57          {
58             if( disposing )
59             {
60                if (components != null)
61                {
62                   components.Dispose();
63                }
64             }
65             base.Dispose( disposing );
66          }
67
68          // Windows Form Designer generated code
69
70          /// <summary>
71          /// The main entry point for the application.
72          /// </summary>
73          [STAThread]
74          static void Main()
75          {
76             Application.Run( new FrmReminder() );
77          }
78
79          // set time for reminder to be displayed
80          private void btnSetTime_Click(
81             object sender, System.EventArgs e )
82          {
83             // disable user input
84             btnSetTime.Enabled = false;
85             dtpReminderTime.Enabled = false;
86             txtReminder.Enabled = false;
87
88             btnReset.Enabled = true; // enable reset
89
90          } // end method btnSetTime_Click
91
92          // timer ticks once every second
93          private void tmrReminder_Tick(
94             object sender, System.EventArgs e )
95          {
96             // display the reminder
97             if ( dtpAlarmTime.Value.Hour == DateTime.Now.Hour &&
98                dtpAlarmTime.Value.Minute == DateTime.Now.Minute &&
99                DateTime.Now.Second == 0 && btnReset.Enabled == true )
100            {
101               // display reminder text
102               MessageBox.Show( textReminder.Text, "Reminder",
103                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation );
104            }
105
106         } // end method tmrReminder_Tick
107
108         // return to initial state
109         private void btnReset_Click(
110            object sender, System.EventArgs e )
111         {
112            // return all GUI controls to initial state
113            btnSetTime.Enabled = true;
114            txtReminder.Enabled = true;
```

```
115          btnReset.Enabled = false;
116          dtpReminderTime.Enabled = true;
117
118       } // end method btnReset_Click
119
120    } // end class FrmAlarmClock
121 }
```

What does this code do?

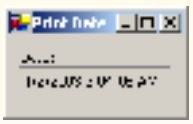
14.14 This code creates a `DateTime` variable. What date does this variable contain?

```
DateTime dtmTime = new DateTime( 2003, 1, 2, 3, 4, 5 );
```

Answer: This variable contains the date January 2, 2003 at 3:04:05 A.M. The complete code reads:

```
1  // Exercise 14.14 Solution
2  // PrintDate.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace PrintDate
12 {
13    /// <summary>
14    /// Summary description for FrmPrintDate.
15    /// </summary>
16    public class FrmPrintDate : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDate;
19       private System.Windows.Forms.Label lblDateText;
20       /// <summary>
21       /// Required designer variable.
22       /// </summary>
23       private System.ComponentModel.Container components = null;
24
25       public FrmPrintDate()
26       {
27          //
28          // Required for Windows Form Designer support
29          //
30          InitializeComponent();
31
32          //
33          // TODO: Add any constructor code after InitializeComponent
34          // call
35          //
36       }
37
38       /// <summary>
39       /// Clean up any resources being used.
40       /// </summary>
41       protected override void Dispose( bool disposing )
42       {
43          if( disposing )
44          {
```

```
45              if (components != null)
46              {
47                 components.Dispose();
48              }
49           }
50           base.Dispose( disposing );
51        }
52
53        // Windows Form Designer generated code
54
55        /// <summary>
56        /// The main entry point for the application.
57        /// </summary>
58        [STAThread]
59        static void Main()
60        {
61           Application.Run( new FrmPrintDate() );
62        }
63
64        // prints the value of a date variable
65        private void FrmPrintDate_Load(
66           object sender, System.EventArgs e)
67        {
68           DateTime dtmTime = new DateTime( 2003, 1, 2, 3, 4, 5 );
69
70           lblDate.Text = Convert.ToString( dtmTime );
71
72        } // end method FrmPrintDate_Load
73
74     } // end class FrmPrintDate
75  }
```



What's wrong with this code?

14.15 The following lines of code are supposed to create a `DateTime` variable and increment its hour value by two. Find the error(s) in the code.

```
DateTime dtmNow = DateTime.Now;
dtmNow.AddHours( 2 );
```

Answer: Method `AddHours` does not actually increment the `DateTime` variable, but instead returns a new `DateTime` variable with the updated value. Thus, the preceding code will not successfully add two hours to `dtmNow`. The complete incorrect code reads:

```
1  // Exercise 14.15 Solution
2  // ChangeTime.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ChangeTime
```

```
{
   /// <summary>
   /// Summary description for FrmChangeTime.
   /// </summary>
   public class FrmChangeTime : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblTwoHours;
      private System.Windows.Forms.Label lblCurrentTime;
      private System.Windows.Forms.Label lblTwoHoursText;
      private System.Windows.Forms.Label lblCurrentText;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmChangeTime()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmChangeTime() );
      }

      // adds two hours to the current time and shows the result
      private void FrmChangeTime_Load(
         object sender, System.EventArgs e )
      {
         DateTime dtmNow = DateTime.Now;
         dtmNow.AddHours( 2 );
```

Should assign the result of this method call to `dtmNow`

```
72
73             lblCurrentTime.Text = Convert.ToString( DateTime.Now );
74             lblTwoHours.Text = Convert.ToString( dtmNow );
75
76          } // end method FrmChangeTime_Load
77
78       } // end class FrmChangeTime
79    }
```

Answer: The complete correct code should read:

```
1   // Exercise 14.15 Solution
2   // ChangeTime.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace ChangeTime
12  {
13     /// <summary>
14     /// Summary description for FrmChangeTime.
15     /// </summary>
16     public class FrmChangeTime : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Label lblTwoHours;
19        private System.Windows.Forms.Label lblCurrentTime;
20        private System.Windows.Forms.Label lblTwoHoursText;
21        private System.Windows.Forms.Label lblCurrentText;
22        /// <summary>
23        /// Required designer variable.
24        /// </summary>
25        private System.ComponentModel.Container components = null;
26
27        public FrmChangeTime()
28        {
29           //
30           // Required for Windows Form Designer support
31           //
32           InitializeComponent();
33
34           //
35           // TODO: Add any constructor code after InitializeComponent
36           // call
37           //
38        }
39
40        /// <summary>
41        /// Clean up any resources being used.
42        /// </summary>
```

```
43         protected override void Dispose( bool disposing )
44         {
45            if( disposing )
46            {
47               if (components != null)
48               {
49                  components.Dispose();
50               }
51            }
52            base.Dispose( disposing );
53         }
54
55         // Windows Form Designer generated code
56
57         /// <summary>
58         /// The main entry point for the application.
59         /// </summary>
60         [STAThread]
61         static void Main()
62         {
63            Application.Run( new FrmChangeTime() );
64         }
65
66         // adds two hours to the current time and shows the result
67         private void FrmChangeTime_Load(
68             object sender, System.EventArgs e)
69         {
70            DateTime dtmNow = DateTime.Now;
71            dtmNow = dtmNow.AddHours( 2 );
72
73            lblCurrentTime.Text = Convert.ToString( DateTime.Now );
74            lblTwoHours.Text = Convert.ToString( dtmNow );
75
76         } // end method FrmChangeTime_Load
77
78      } // end class FrmChangeTime
79   }
```

Programming Challenge

14.16 (*Parking Garage Fee Calculator Application*) Create an application that computes the fee for parking a car in a parking garage (Fig. 14.24). The user should provide the **Time In:** and **Time Out:** values by using `DateTimePickers`. The application should calculate the cost of parking in the garage for the specified amount of time. Assume that parking costs three dollars an hour. When calculating the total time spent in the garage, you can ignore the seconds value, but treat the minutes value as a fraction of an hour (1 minute is 1/60 of an hour). For simplicity, assume that no overnight parking is allowed, so each car leaves the garage on the same day in which it arrives.

Figure 14.24 Parking Garage Fee Calculator GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial14\Exercises\ParkingGarageFeeCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ParkingGarageFeeCalculator.sln` in the `ParkingGarageFeeCalculator` directory to open the application.

c) ***Inserting the `DateTimePicker` controls.*** Add two `DateTimePicker` controls to the `Form`. Set the `DateTimePickers` so that they show the time only. Set the `Size` property of each `DateTimePicker` control to `80, 20`, and move the `DateTimePickers` so that they are positioned as in Fig. 14.24. Rearrange and comment the new control declarations appropriately.

d) ***Writing the method `Fee`.*** Define a method `Fee` that accepts four `int`s as parameters—the hour value of the **Time In:**, the hour value of the **Time Out:**, the minute value of the **Time In:** and the minute value of the **Time Out:**. Using this information, method `Fee` should calculate the fee for parking in the garage. The method should then return this value as a `decimal`.

e) ***Coding the Calculate `Button`'s `Click` event handler.*** Add the `Click` event handler for the **Calculate** `Button`. If the time out is earlier than the time in, disallow the input and display a `MessageBox` to the user. Otherwise, this event handler should call `Fee` to obtain the amount due. It should then display the amount (formatted as currency) in a `Label`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Use the `DateTimePickers`' up and down arrows to select a time the car was placed in the garage and the time the car was taken out of the garage. Click the **Calculate** `Button` and verify that the correct fee is displayed.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 14.16 Solution
2  // ParkingGarageFeeCalculator.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ParkingGarageFeeCalculator
12 {
13    /// <summary>
14    /// Summary description for FrmParkingGarageFeeCalculator.
15    /// </summary>
16    public class FrmParkingGarageFeeCalculator :
17       System.Windows.Forms.Form
18    {
19       // Label and DateTimePicker to choose time that
20       // parking garage was entered
```

```
21          private System.Windows.Forms.Label lblTimeIn;
22          private System.Windows.Forms.DateTimePicker dtpTimeIn;
23
24          // Label and DateTimePicker to choose time that
25          // parking garage was exited
26          private System.Windows.Forms.Label lblTimeOut;
27          private System.Windows.Forms.DateTimePicker dtpTimeOut;
28
29          // Labels to display parking fee
30          private System.Windows.Forms.Label lblFee;
31          private System.Windows.Forms.Label lblFeeResult;
32
33          // Button to calculate parking fee
34          private System.Windows.Forms.Button btnCalculate;
35
36          /// <summary>
37          /// Required designer variable.
38          /// </summary>
39          private System.ComponentModel.Container components = null;
40
41          public FrmParkingGarageFeeCalculator()
42          {
43             //
44             // Required for Windows Form Designer support
45             //
46             InitializeComponent();
47
48             //
49             // TODO: Add any constructor code after InitializeComponent
50             // call
51             //
52          }
53
54          /// <summary>
55          /// Clean up any resources being used.
56          /// </summary>
57          protected override void Dispose( bool disposing )
58          {
59             if( disposing )
60             {
61                if (components != null)
62                {
63                   components.Dispose();
64                }
65             }
66             base.Dispose( disposing );
67          }
68
69          // Windows Form Designer generated code
70
71          /// <summary>
72          /// The main entry point for the application.
73          /// </summary>
74          [STAThread]
75          static void Main()
76          {
77             Application.Run( new FrmParkingGarageFeeCalculator() );
78          }
79
80          // calculates cost of parking in garage
```

```
81          decimal Fee(
82             int intTimeOutHour, int intTimeInHour,
83             int intTimeOutMinute, int intTimeInMinute )
84          {
85             // determines number of elapsed hours
86             int intHours = intTimeOutHour - intTimeInHour;
87
88             // determines number of elapsed minutes
89             int intMinutes = intTimeOutMinute - intTimeInMinute;
90
91             if ( intMinutes < 0 )
92             {
93                intHours--;
94                intMinutes = intTimeOutMinute + ( 60 - intTimeInMinute );
95             }
96
97             return ( intHours + ( intMinutes / 60.0M ) ) * 3;
98
99          } // end method Fee
100
101         // invoked when Calculate Button is clicked
102         private void btnCalculate_Click(
103            object sender, System.EventArgs e )
104         {
105            decimal decFee = 0;
106
107            // ensure that time out is later than time in
108            if ( dtpTimeOut.Value < dtpTimeIn.Value )
109            {
110               MessageBox.Show( "Time out must be later than time in.",
111                  "Incorrect Input", MessageBoxButtons.OK,
112                  MessageBoxIcon.Exclamation );
113            }
114            else
115            {
116               // calls method Fee
117               decFee = Fee(
118                  dtpTimeOut.Value.Hour, dtpTimeIn.Value.Hour,
119                  dtpTimeOut.Value.Minute, dtpTimeIn.Value.Minute );
120
121               // output fee as currency
122               lblFeeResult.Text = String.Format( "{0:C}", decFee );
123            }
124
125         } // end method btnCalculate_Click
126
127      } // end class FrmParkingGarageFeeCalculator
128   }
```

TUTORIAL 15

Fund Raiser Application

Introducing Scope and Pass-by-Reference

Solutions

Instructor's Manual Exercise Solutions Tutorial 15

MULTIPLE-CHOICE QUESTIONS

15.1 Trying to convert from a `double` to an `int` without any special code causes _____.

a) a syntax error
b) nothing unusual
c) a logic error
d) data loss

15.2 To carry out some conversions, C# requires that variables _____.

a) are passed by value
b) are passed by reference
c) be explicitly converted to a different type to avoid unintended data loss
d) are used only within the block in which the variables are declared

15.3 A variable declared inside a class, but outside a method, is called a(n) _____.

a) local variable
b) hidden variable
c) instance variable
d) constant variable

15.4 C# provides methods in class _____ to convert from one type to another.

a) `ChangeTo`
b) `Convert`
c) `ConvertTo`
d) `ChangeType`

15.5 Referencing a variable with block scope outside the block in which the programmer declared it is _____.

a) a logic error
b) allowed, but not recommended
c) a syntax error
d) None of the above.

15.6 When using pass-by-reference, keyword _____ should be used before parameters that are initialized before the method call.

a) `out`
b) `ref`
c) `val`
d) `byref`

15.7 With _____, changes made to parameter variables' values do not affect the value of the variables in the calling method.

a) explicit conversions
b) pass-by-value
c) pass-by-reference
d) None of the above.

15.8 Instance variables _____.

a) are members of a class
b) should be prefixed by `m_`
c) can be accessed by a method in the same class
d) All of the above.

15.9 Assigning a "smaller" type to a "larger" type is a _____ conversion.

a) narrowing
b) shortening
c) widening
d) lengthening

15.10 A value of type `bool` can be implicitly converted to _____.

a) `int`
b) `string`
c) `object`
d) None of the above.

Answers: 15.1) a. 15.2) c. 15.3) c. 15.4) b. 15.5) c. 15.6) b. 15.7) b. 15.8) d. 15.9) c. 15.10) c.

EXERCISES

15.11 (***Task List Application***) Create an application that allows users to add items to a daily task list (Fig. 15.24). The application should also display the number of tasks to be performed. Use method `Convert.ToString` to display the number of tasks in a `Label`.

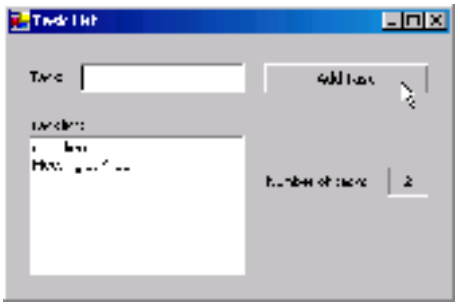


Figure 15.24 **Task List** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial15\Exercises\TaskList` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `TaskList.sln` in the `TaskList` directory to open the application.

c) ***Declaring an instance variable.*** Declare instance variable `intCounter` of type `int` and initialize its value to `0`.

d) ***Adding the Add Task `Button`'s `Click` event handler.*** Double click the **Add Task** `Button` to generate the empty event handler `btnAdd_Click`. This event handler should display the user input in the `ListBox` and clear the user input from the `TextBox`. The event handler should Increment the instance variable you created in the previous step and update the `Label` that displays the number of tasks. Use method `Convert.ToString` to display the number of tasks in the `Label`. Finally, the event handler should transfer the focus to the `TextBox`.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter several tasks, click the **Add Task** `Button` after each. Verify that each task is added to the **Task list:** `ListBox`, and that the number of tasks is incremented with each new task.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 15.11 Solution
// TaskList.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace TaskList
{
   /// <summary>
   /// Summary description for FrmTaskList.
   /// </summary>
   public class FrmTaskList : System.Windows.Forms.Form
   {
      // Label and TextBox to input a task
      private System.Windows.Forms.Label lblTask;
      private System.Windows.Forms.TextBox txtTask;

      // Button to add a task to the list
      private System.Windows.Forms.Button btnAdd;

      // Label and ListBox to display the task list
      private System.Windows.Forms.Label lblDescription;
      private System.Windows.Forms.ListBox lstTasks;
```

```
28
29          // Labels to display the number of tasks
30          private System.Windows.Forms.Label lblDescribeOutput;
31          private System.Windows.Forms.Label lblOutput;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          // declare instance variable to store number of tasks
39          int intCounter = 0;
40
41          public FrmTaskList()
42          {
43             //
44             // Required for Windows Form Designer support
45             //
46             InitializeComponent();
47
48             //
49             // TODO: Add any constructor code after InitializeComponent
50             // call
51             //
52          }
53
54          /// <summary>
55          /// Clean up any resources being used.
56          /// </summary>
57          protected override void Dispose( bool disposing )
58          {
59             if( disposing )
60             {
61                if (components != null)
62                {
63                   components.Dispose();
64                }
65             }
66             base.Dispose( disposing );
67          }
68
69          // Windows Form Designer generated code
70
71          /// <summary>
72          /// The main entry point for the application.
73          /// </summary>
74          [STAThread]
75          static void Main()
76          {
77             Application.Run( new FrmTaskList() );
78          }
79
80          // handles Add Task Button's Click event
81          private void btnAdd_Click(
82             object sender, System.EventArgs e )
83          {
84             // insert task into ListBox
85             lstTasks.Items.Add( txtTask.Text );
86
87             intCounter++; // increment task number
```

```
88
89          txtTask.Clear(); // clear TextBox of user input
90
91          // convert int to string to display number of tasks
92          lblOutput.Text = Convert.ToString( intCounter );
93
94          txtTask.Focus(); // tranfer focus to the TextBox
95
96       } // end method btnAdd_Click
97
98    } // end class FrmTaskList
99 }
```

15.12 (*Quiz Average Application*) Develop an application that computes a student's average quiz score for all of the quiz scores entered. The application should look like the GUI in Fig. 15.25. Use method `Convert.ToInt32` to convert the user input to an `int`. Use instance variables with class scope to keep track of the sum of all the quiz scores entered and the number of quiz scores entered.

Figure 15.25 Quiz Average application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial15\Exercises\QuizAverage` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `QuizAverage.sln` in the `QuizAverage` directory to open the application.

c) ***Adding instance variables.*** Add two instance variables—`m_intTotalScore`, which keeps track of the sum of all the quiz scores entered, and `m_intTaken`, which keeps track of the number of quiz scores entered.

d) ***Adding the Grade Quiz `Button`'s event handler.*** Double click the **Submit Score** `Button` to generate the empty event handler `btnCalculate_Click`. The code required in *Steps e–j* should be placed in this event handler.

e) ***Obtaining user input.*** Use method `Convert.ToInt32` to convert the user input from the `TextBox` to an `int`.

f) ***Updating the number of quiz scores entered.*** Increment the number of quiz scores entered.

g) ***Updating the sum of all the quiz scores entered.*** Add the current quiz score to the current total to update the sum of all the quiz scores entered.

h) ***Calculating the average score.*** Divide the sum of all the quiz scores entered by the number of quiz scores entered to calculate the average score.

i) ***Displaying the average score.*** Use method `Convert.ToString` to display the average quiz grade in the **Average:** field.

j) ***Displaying the number of quizzes taken.*** Use method `Convert.ToString` to display the number of quiz scores entered in the **Number taken:** field.

k) ***Running the application.*** Select **Debug > Start** to run your application. Enter several quiz scores, clicking the **Submit Score** `Button` after each. With each new score, verify that the **Number taken:** field is incremented and that the average is updated correctly.

l) ***Closing the application.*** Close your running application by clicking its close box.

m) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 15.12 Solution
// QuizAverage.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace QuizAverage
{
   /// <summary>
   /// Summary description for FrmQuizAverage.
   /// </summary>
   public class FrmQuizAverage : System.Windows.Forms.Form
   {
      // Label and TextBox to input a quiz score
      private System.Windows.Forms.Label lblScore;
      private System.Windows.Forms.TextBox txtScore;

      // Labels to display number of quizzes taken
      private System.Windows.Forms.Label lblNumber;
      private System.Windows.Forms.Label lblTaken;

      // Labels to display average quiz score
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblAverage;

      // Button to submit a quiz score and recalculate the average
      private System.Windows.Forms.Button btnCalculate;

      // instance variables store total score and
      // number quizzes taken
      int m_intTotalScore = 0;
      int m_intTaken = 0;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmQuizAverage()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
```

```
60          {
61             if( disposing )
62             {
63                if (components != null)
64                {
65                   components.Dispose();
66                }
67             }
68             base.Dispose( disposing );
69          }
70
71          // Windows Form Designer generated code
72
73          /// <summary>
74          /// The main entry point for the application.
75          /// </summary>
76          [STAThread]
77          static void Main()
78          {
79             Application.Run( new FrmQuizAverage() );
80          }
81
82          // handles Submit Score Button's Click event
83          private void btnCalculate_Click(
84             object sender, System.EventArgs e )
85          {
86             int intScore;
87             int intAverage;
88
89             // obtain and convert user input
90             intScore = Convert.ToInt32( txtScore.Text );
91
92             // update number of quizzes taken
93             m_intTaken++;
94
95             // update total score
96             m_intTotalScore += intScore;
97
98             // calculate average score
99             intAverage = m_intTotalScore / m_intTaken;
100
101            // display average score
102            lblAverage.Text = Convert.ToString( intAverage );
103
104            // display number of quizzes taken
105            lblTaken.Text = Convert.ToString( m_intTaken );
106
107         } // end method btnCalculate_Click
108
109      } // end class FrmQuizAverage
110   }
```

15.13 (*Maximum Application*) Modify the **Maximum** application from Tutorial 13 (Fig. 15.26) to use keyword `out` to pass a fourth argument to method `Maximum` by reference. Also, use methods from class `Convert` to perform any necessary type conversions.

Figure 15.26 Maximum application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial15\Exercises\Maximum` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Maximum.sln` in the `Maximum` directory to open the application.

c) ***Adding a local variable.*** Add local variable `dblMaximum` of type `double` to event handler `btnMaximum_Click`. The code required in *Steps d–e* should be placed in this event handler.

d) ***Passing four arguments to method `Maximum`.*** Use method `Double.Parse` to convert the user input from the `TextBoxes` to `doubles`. Pass these three values as the first three arguments to method `Maximum`. Pass local variable `dblMaximum` as the fourth argument to method `Maximum`.

e) ***Displaying the maximum value.*** Use method `Convert.ToString` to display local variable `dblMaximum` in the **Maximum:** field.

f) ***Changing the return type of `Maximum`.*** Change the return type of `Maximum` to `void`. Make sure that method `Maximum` no longer returns a value and does not specify a return type.

g) ***Adding a fourth parameter to method `Maximum`.*** Add a fourth parameter `dblFinalMaximum` of type `double` to `Maximum`'s method header. Use keyword `out` to specify that this argument will be passed by reference uninitialized. Remove the declaration of variable `dblFinalMaximum` from the body of method `Maximum`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter three different values into the input fields and click the **Maximum** `Button`. Verify that the largest value is displayed in the **Maximum:** field.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 15.13 Solution
2  // Maximum.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Maximum
12 {
13    /// <summary>
14    /// Summary description for FrmMaximum.
15    /// </summary>
16    public class FrmMaximum : System.Windows.Forms.Form
17    {
18       // Labels and TextBoxes to input the three values
```

```
19          private System.Windows.Forms.Label lblFirst;
20          private System.Windows.Forms.TextBox txtFirst;
21
22          private System.Windows.Forms.Label lblSecond;
23          private System.Windows.Forms.TextBox txtSecond;
24
25          private System.Windows.Forms.Label lblThird;
26          private System.Windows.Forms.TextBox txtThird;
27
28          // Labels to display the maximum value
29          private System.Windows.Forms.Label lblMaximum;
30          private System.Windows.Forms.Label lblOutput;
31
32          // Button to calculate the maximum value
33          private System.Windows.Forms.Button btnMaximum;
34
35          /// <summary>
36          /// Required designer variable.
37          /// </summary>
38          private System.ComponentModel.Container components = null;
39
40          public FrmMaximum()
41          {
42             //
43             // Required for Windows Form Designer support
44             //
45             InitializeComponent();
46
47             //
48             // TODO: Add any constructor code after InitializeComponent
49             // call
50             //
51          }
52
53          /// <summary>
54          /// Clean up any resources being used.
55          /// </summary>
56          protected override void Dispose( bool disposing )
57          {
58             if( disposing )
59             {
60                if (components != null)
61                {
62                   components.Dispose();
63                }
64             }
65             base.Dispose( disposing );
66          }
67
68          // Windows Form Designer generated code
69
70          /// <summary>
71          /// The main entry point for the application.
72          /// </summary>
73          [STAThread]
74          static void Main()
75          {
76             Application.Run( new FrmMaximum() );
77          }
78
```

```
79          // obtain values in each TextBox, call method Maximum
80          private void btnMaximum_Click(
81             object sender, System.EventArgs e )
82          {
83             double dblMaximum;
84
85             Maximum( Double.Parse( txtFirst.Text ),
86                Double.Parse( txtSecond.Text ),
87                Double.Parse( txtThird.Text ), out dblMaximum );
88
89             lblOutput.Text =  Convert.ToString( dblMaximum );
90
91          } // end method btnMaximum_Click
92
93          // find maximum of three parameter values
94          void Maximum(
95             double dblOne, double dblTwo,
96             double dblThree, out double dblFinalMaximum )
97          {
98             double dblTemporaryMaximum;
99
100            dblTemporaryMaximum = Math.Max( dblOne, dblTwo );
101            dblFinalMaximum = Math.Max( dblTemporaryMaximum, dblThree );
102
103         } // end method Maximum
104
105      } // end class FrmMaximum
106   }
```

What does this code do? ▶

15.14 What is displayed in `Label lblDisplay` when the following code is executed? Assume these variables and methods are declared inside the class `FrmScopeTest`.

```
1   int intValue2 = 5;
2
3   private void btnEnter_Click(
4      object sender, System.EventArgs e )
5   {
6      int intValue1 = 10;
7      int intValue2 = 3;
8
9      Test( ref intValue1 );
10
11     lblDisplay.Text = Convert.ToString( intValue1 );
12
13  } // end method btnEnter_Click
14
15  void Test( ref int intValue1 )
16  {
17     intValue1 *= intValue2;
18
19  } // end method Test
```

Answer: `Label lblDisplay` displays the value of variable `intValue1` (50). When the code invokes the method `Test`, it passes `intValue1` pass-by-reference. Any changes made to `intValue1` in the method `Test` are reflected in `btnEnter_Click`'s local variable `intValue1`. When method `Test` multiplies `intValue1` by `intValue2`, `intValue2` is the class instance variable, whose value is 5. Method `Test` does not have access to `btnEnter_Click`'s local variable `intValue2`. The complete code reads:

```
1  // Exercise 15.14 Solution
2  // ScopeTest.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ScopeTest
12 {
13    /// <summary>
14    /// Summary description for FrmScopeTest.
15    /// </summary>
16    public class FrmScopeTest : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Button btnEnter;
19       private System.Windows.Forms.Label lblDisplay;
20       private System.Windows.Forms.Label lblResult;
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.Container components = null;
25
26       int intValue2 = 5;
27
28       public FrmScopeTest()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34
35          //
36          // TODO: Add any constructor code after InitializeComponent
37          // call
38          //
39       }
40
41       /// <summary>
42       /// Clean up any resources being used.
43       /// </summary>
44       protected override void Dispose( bool disposing )
45       {
46          if( disposing )
47          {
48             if (components != null)
49             {
50                components.Dispose();
51             }
52          }
53          base.Dispose( disposing );
54       }
55
56       // Windows Form Designer generated code
57
58       /// <summary>
59       /// The main entry point for the application.
60       /// </summary>
```

```
61         [STAThread]
62         static void Main()
63         {
64            Application.Run( new FrmScopeTest() );
65         }
66
67         // performs calculations with variables of different scopes
68         private void btnEnter_Click(
69            object sender, System.EventArgs e )
70         {
71            int intValue1 = 10;
72            int intValue2 = 3;
73
74            Test( ref intValue1 );
75
76            lblDisplay.Text = Convert.ToString( intValue1 );
77
78         } // end method btnEnter_Click
79
80         // multiplies two variables
81         void Test( ref int intValue1 )
82         {
83            intValue1 *= intValue2;
84
85         } // end method Test
86
87      } // end class FrmScopeTest
88   }
```

What's wrong with this code?

15.15 Find the error(s) in the following code (the method should assign the value 14 to variable `intResult`).

```
1   int Sum()
2   {
3      string strNumber = "4";
4      int intNumber = 10;
5      int intResult;
6
7      intResult = strNumber + intNumber;
8
9      return intResult;
10
11  } // end method Sum
```

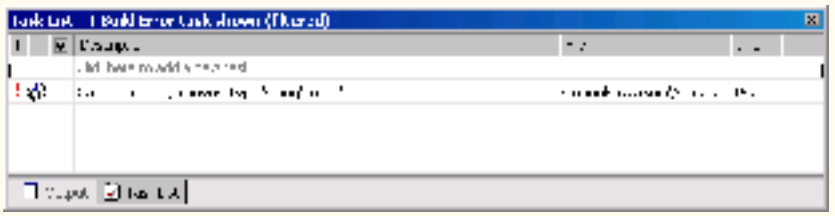
Answer: The code must explicitly convert `strNumber` to `int`. The complete incorrect code reads:

```
1   // Exercise 15.15 Solution
2   // Sum.cs (Incorrect)
3
4   using System;
5   using System.Drawing;
```

```
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Sum
{
   /// <summary>
   /// Summary description for FrmSum.
   /// </summary>
   public class FrmSum : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblSumOut;
      private System.Windows.Forms.Label lblNumber2Out;
      private System.Windows.Forms.Label lblNumber1Out;
      private System.Windows.Forms.Label lblSum;
      private System.Windows.Forms.Label lblNumber2;
      private System.Windows.Forms.Label lblNumber1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmSum()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmSum() );
```

```
66          }
67
68          // prints the result of the Sum method
69          private void FrmSum_Load( object sender, System.EventArgs e )
70          {
71             lblSumOut.Text = Convert.ToString( Sum() );
72
73          } // end method FrmSum_Load
74
75          // takes the sum of a string and an int
76          int Sum()
77          {
78             string strNumber = "4";
79             int intNumber = 10;
80             int intResult;
81
82             intResult = strNumber + intNumber;
83
84             return intResult;
85
86          } // end method Sum
87
88       } // end class FrmSum
89    }
```

Should use `Int32.Parse` to convert `strNumber` to an `int`



Answer: The complete corrected code should read:

```
1  // Exercise 15.15 Solution
2  // Sum.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Sum
12 {
13    /// <summary>
14    /// Summary description for FrmSum.
15    /// </summary>
16    public class FrmSum : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblSumOut;
19       private System.Windows.Forms.Label lblNumber2Out;
20       private System.Windows.Forms.Label lblNumber1Out;
21       private System.Windows.Forms.Label lblSum;
22       private System.Windows.Forms.Label lblNumber2;
23       private System.Windows.Forms.Label lblNumber1;
```

```
24          /// <summary>
25          /// Required designer variable.
26          /// </summary>
27          private System.ComponentModel.Container components = null;
28
29          public FrmSum()
30          {
31             //
32             // Required for Windows Form Designer support
33             //
34             InitializeComponent();
35
36             //
37             // TODO: Add any constructor code after InitializeComponent
38             // call
39             //
40          }
41
42          /// <summary>
43          /// Clean up any resources being used.
44          /// </summary>
45          protected override void Dispose( bool disposing )
46          {
47             if( disposing )
48             {
49                if (components != null)
50                {
51                   components.Dispose();
52                }
53             }
54             base.Dispose( disposing );
55          }
56
57          // Windows Form Designer generated code
58
59          /// <summary>
60          /// The main entry point for the application.
61          /// </summary>
62          [STAThread]
63          static void Main()
64          {
65             Application.Run( new FrmSum() );
66          }
67
68          // prints the result of the Sum method
69          private void FrmSum_Load( object sender, System.EventArgs e )
70          {
71             lblSumOut.Text = Convert.ToString( Sum() );
72
73          } // end method FrmSum_Load
74
75          // takes the sum of a string and an integer
76          int Sum()
77          {
78             string strNumber = "4";
79             int intNumber = 10;
80             int intResult;
81
82             intResult = Int32.Parse( strNumber ) + intNumber;
83
```

```
84            return intResult;
85
86         } // end method Sum
87
88      } // end class FrmSum
89   }
```

Programming Challenge

15.16 (***Schedule Book Application***) Develop an application that allows a user to enter a schedule of appointments and their respective times. Create the `Form` in Fig. 15.27 and name the application **Schedule Book**. Add a method called `TimeTaken` that returns a `bool` value. Each time a user enters a new appointment, method `TimeTaken` determines if the user has scheduled more than one appointment at the same time. If `TimeTaken` returns `true`, the user will be notified via a message dialog. Otherwise, the appointment should be added to the `ListBoxes`. Use methods from class `Convert` as necessary. Use the `Items` property of the `ListBoxes` along with the square bracket (`[]`) notation from Tutorial 10 to access the members of the `ListBoxes`.

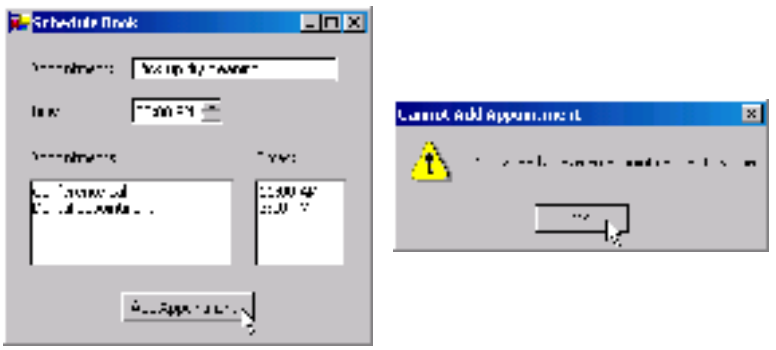


Figure 15.27 **Schedule Book** application.

Answer:

```
1  // Exercise 15.16 Solution
2  // ScheduleBook.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ScheduleBook
12 {
13    /// <summary>
14    /// Summary description for FrmScheduleBook.
15    /// </summary>
16    public class FrmScheduleBook : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input an appointment
19       private System.Windows.Forms.Label lblAppointment;
20       private System.Windows.Forms.TextBox txtAppointment;
21
```

```
22          // Label and DateTimePicker to choose an appointment time
23          private System.Windows.Forms.Label lblTime;
24          private System.Windows.Forms.DateTimePicker dtpTime;
25
26          // Label and ListBox to display list of appointments
27          private System.Windows.Forms.Label lblAppointmentList;
28          private System.Windows.Forms.ListBox lstAppointments;
29
30          // Label and ListBox to display list of appointment times
31          private System.Windows.Forms.Label lblTimeList;
32          private System.Windows.Forms.ListBox lstTimes;
33
34          // Button to add an appointment to the list
35          private System.Windows.Forms.Button btnAdd;
36
37          /// <summary>
38          /// Required designer variable.
39          /// </summary>
40          private System.ComponentModel.Container components = null;
41
42          public FrmScheduleBook()
43          {
44             //
45             // Required for Windows Form Designer support
46             //
47             InitializeComponent();
48
49             //
50             // TODO: Add any constructor code after InitializeComponent
51             // call
52             //
53          }
54
55          /// <summary>
56          /// Clean up any resources being used.
57          /// </summary>
58          protected override void Dispose( bool disposing )
59          {
60             if( disposing )
61             {
62                if (components != null)
63                {
64                   components.Dispose();
65                }
66             }
67             base.Dispose( disposing );
68          }
69
70          // Windows Form Designer generated code
71
72          /// <summary>
73          /// The main entry point for the application.
74          /// </summary>
75          [STAThread]
76          static void Main()
77          {
78             Application.Run( new FrmScheduleBook() );
79          }
80
81          // handles Add Appointment Button's Click event
```

```
82          private void btnAdd_Click(
83             object sender, System.EventArgs e )
84          {
85             // appointment scheduled for given time
86             bool blnTimeTaken = TimeTaken();
87
88             // display message if appointment conflict
89             if ( blnTimeTaken == true )
90             {
91                MessageBox.Show( "You already have an appointment " +
92                   "at this time", "Cannot Add Appointment",
93                   MessageBoxButtons.OK, MessageBoxIcon.Exclamation );
94             }
95             // otherwise add appointment and time to ListBoxes
96             else
97             {
98                lstAppointments.Items.Add( txtAppointment.Text );
99                lstTimes.Items.Add( dtpTime.Value.ToShortTimeString() );
100            }
101
102            // clear user input from TextBoxes
103            txtAppointment.Clear();
104
105         } // end method btnAdd_Click
106
107         // determines if an appointment already exists
108         // at specified time
109         bool TimeTaken()
110         {
111            // determine number of appointments
112            int intItems = lstTimes.Items.Count;
113
114            // determines if items are in time ListBox
115            if ( intItems != 0 )
116            {
117               int intCounter;
118
119               // search ListBox to determine if an appointment
120               // has been made for that time
121               for ( intCounter = 0; intCounter < intItems;
122                  intCounter++ )
123               {
124                  // compare times in ListBox with user entry
125                  if ( Convert.ToString( lstTimes.Items[ intCounter ] )
126                     == dtpTime.Value.ToShortTimeString() )
127                  {
128                     return true;
129                  }
130               }
131            }
132
133            return false;
134
135         } // end method TimeTaken
136
137      } // end class FrmScheduleBook
138   }
```

TUTORIAL 16

Craps Game Application

Introducing Random-Number Generation

Solutions

Instructor's Manual Exercise Solutions Tutorial 16

MULTIPLE-CHOICE QUESTIONS

16.1 A `Random` object can generate pseudorandom numbers of type _____.

a) `int` b) `decimal`
c) `double` d) Both a and c.

16.2 A _____ is a group of related classes in the Framework Class Library.

a) `classpace` b) `directory`
c) `namespace` d) `cluster`

16.3 Object variable names should be prefixed with _____.

a) `var` b) `obj`
c) `ran` d) `ojt`

16.4 The `Next` method of class `Random` can be called using _____.

a) one argument b) no arguments
c) two arguments d) All of the above.

16.5 The statement _____ assigns `intValue` a random number in the range from 5 to 20.

a) `intValue = objRandom.Next(5, 21);` b) `intValue = objRandom.Next(4, 20);`
c) `intValue = objRandom.Next(5, 20);` d) `intValue = objRandom.Next(4, 21);`

16.6 The _____ method specifies the file from which an image is loaded.

a) `Random.Next` b) `Image.FromFile`
c) `Directory.GetCurrentDirectory` d) None of the above.

16.7 The `System.IO` namespace contains classes and methods to _____.

a) access files and directories b) display graphics in an application
c) insert multimedia into an application d) All of the above.

16.8 The values returned by the _____ method of class `Random` are actually pseudorandom numbers.

a) `NextRandom` b) `Pseudorandom`
c) `Next` d) `Pseudo`

16.9 When creating random numbers, the second argument passed to the `Next` method is _____.

a) equal to the maximum value you wish to be generated
b) equal to one more than the maximum value you wish to be generated
c) equal to one less than the maximum value you wish to be generated
d) equal to the minimum value you wish to be generated

16.10 A(n) _____ is a group of related, named constants.

a) namespace b) variable
c) enumeration d) None of the above.

Answers: 16.1) d. 16.2) c. 16.3) b. 16.4) d. 16.5) a. 16.6) b. 16.7) a. 16.8) c. 16.9) b. 16.10) c.

EXERCISES

16.11 (***Guess the Number Application***) Develop an application that generates a random number and prompts the user to guess the number (Fig. 16.21). When the user clicks the **New Game** `Button`, the application chooses a number in the range 1 to 100 at random. The user enters guesses into the **Guess:** `TextBox` and clicks the **Enter** `Button`. If the guess is correct,

the game ends, and the user can start a new game. If the guess is not correct, the application should indicate if the guess is higher or lower than the correct number.

Figure 16.21 Guess the Number application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial16\Exercises\GuessNumber` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `GuessNumber.sln` in the `GuessNumber` directory to open the application.

c) ***Creating a `Random` object.*** Create two instance variables. The first variable should store a `Random` object and the second variable should store a random-generated number.

d) ***Adding a `Load` event handler for the `Form`.*** Add a `Load` event handler for the `Form` that initializes the second instance variable to a random-generated number.

e) ***Adding a `Click` event handler for the Enter `Button`.*** Add a `Click` event handler for the **Enter** `Button` that retrieves the value entered by the user and compares that value to the random-generated number. If the guess is correct, display `"Correct!"` in the output `Label`. Then, disable the **Enter** `Button` and enable the **New Game** `Button`. If the user's guess is higher than the correct answer, display `"Too high..."` in the output `Label`. If the user's guess is lower than the correct answer, display `"Too low..."` in the output `Label`.

f) ***Adding a `Click` event handler for the New Game `Button`.*** Add a `Click` event handler for the **New Game** `Button` that generates a new random number for the instance variable. The event handler should then disable the **New Game** `Button`, enable the **Enter** `Button` and clear the **Result:** `TextBox`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Enter guesses (clicking the **Enter** `Button` after each) until you have successfully determined the answer. Click the **New Game** `Button` and test the application again.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 16.11 Solution
2  // GuessNumber.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace GuessNumber
12 {
13    /// <summary>
14    /// Summary description for FrmGuessNumber.
15    /// </summary>
16    public class FrmGuessNumber : System.Windows.Forms.Form
17    {
18       // Label to display directions
19       private System.Windows.Forms.Label lblQuestion;
20
```

```
21          // Label and TextBox to input a number as a guess
22          private System.Windows.Forms.Label lblGuessNumber;
23          private System.Windows.Forms.TextBox txtGuessNumber;
24
25          // Labels to display result of guess
26          private System.Windows.Forms.Label lblResult;
27          private System.Windows.Forms.Label lblResultLabel;
28
29          // Button to enter a guess
30          private System.Windows.Forms.Button btnEnter;
31
32          // Button to begin a new game
33          private System.Windows.Forms.Button btnNewGame;
34
35          /// <summary>
36          /// Required designer variable.
37          /// </summary>
38          private System.ComponentModel.Container components = null;
39
40          Random m_objRandom = new Random();
41          int m_intNumber;
42
43          public FrmGuessNumber()
44          {
45             //
46             // Required for Windows Form Designer support
47             //
48             InitializeComponent();
49
50             //
51             // TODO: Add any constructor code after InitializeComponent
52             // call
53             //
54          }
55
56          /// <summary>
57          /// Clean up any resources being used.
58          /// </summary>
59          protected override void Dispose( bool disposing )
60          {
61             if( disposing )
62             {
63                if (components != null)
64                {
65                   components.Dispose();
66                }
67             }
68             base.Dispose( disposing );
69          }
70
71          // Windows Form Designer generated code
72
73          /// <summary>
74          /// The main entry point for the application.
75          /// </summary>
76          [STAThread]
77          static void Main()
78          {
79             Application.Run( new FrmGuessNumber() );
80          }
```

```
81
82          // Load Form event
83          private void FrmGuessNumber_Load(
84             object sender, System.EventArgs e )
85          {
86             m_intNumber = m_objRandom.Next( 1, 101 );
87
88          } // end FrmGuessNumber_Load
89
90          // handles Enter Button's Click event
91          private void btnEnter_Click(
92             object sender, System.EventArgs e )
93          {
94             // check answer
95             if ( Int32.Parse( txtGuessNumber.Text ) == m_intNumber )
96             {
97                lblResult.Text = "Correct!";
98                btnEnter.Enabled = false;
99                btnNewGame.Enabled = true;
100            }
101            else if ( Int32.Parse( txtGuessNumber.Text )
102               > m_intNumber )
103            {
104               lblResult.Text = "Too high...";
105            }
106            else
107            {
108               lblResult.Text = "Too low...";
109            }
110
111            txtGuessNumber.Clear();
112            txtGuessNumber.Focus();
113
114         } // end method btnEnter_Click
115
116         // handles New Game Button's Click event
117         private void btnNewGame_Click(
118            object sender, System.EventArgs e )
119         {
120            // generate new random number
121            m_intNumber = m_objRandom.Next( 1, 101 );
122            btnEnter.Enabled = true;
123            btnNewGame.Enabled = false;
124            lblResult.Text = ""; // clear result;
125
126         } // end method btnNewGame_Click
127
128      } // end class FrmGuessNumber
129   }
```

16.12 (***Dice Simulator Application***) Develop an application that simulates rolling two six-sided dice. Your application should have a **Roll** `Button` that, when clicked, displays two dice images corresponding to random numbers. It should also display the number of times each face has appeared. Your application should appear similar to Fig. 16.22.

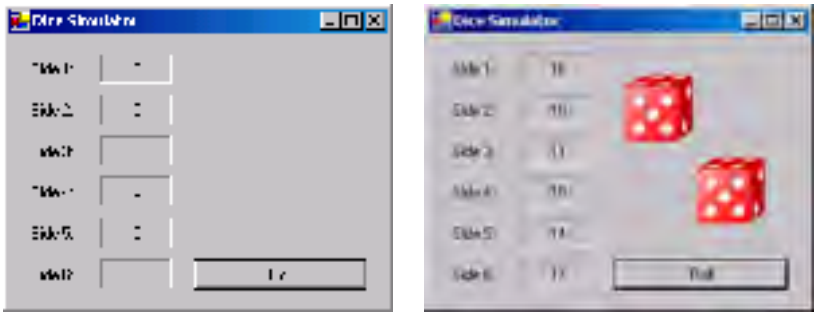


Figure 16.22 Dice Simulator application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial16\Exercises\DiceSimulator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `DiceSimulator.sln` in the `DiceSimulator` directory to open the application.

c) ***Displaying the die image.*** Create a method named `DisplayDie` that takes a `PictureBox` control as an argument. This method should generate a random number to simulate a die roll. Then, display the die image in the corresponding `PictureBox` control on the `Form`. The die image should correspond to the random number that was generated. To set the image, refer to the code presented in Fig. 16.20.

d) ***Adding a `Click` event handler for the Roll `Button`.*** Add a `Click` event handler for the **Roll** `Button`. Call method `DisplayDie` in this event handler to display the images for both dice.

e) ***Displaying the frequency.*** Add a method called `DisplayFrequency` that uses a `switch` statement to update the number of times each face has appeared. Create an enumeration for the dice faces which will be used in the `switch` statement.

f) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Roll** `Button` several times. Each time, two die faces should be displayed at random. Verify after each roll that the appropriate face values on the left are incremented.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 16.12 Solution
2  // DiceSimulator.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.IO;
11
12 namespace DiceSimulator
13 {
14    /// <summary>
15    /// Summary description for FrmDiceSimulator.
16    /// </summary>
17    public class FrmDiceSimulator : System.Windows.Forms.Form
18    {
19       // Labels to display the number of times each face
20       // has appeared
21       private System.Windows.Forms.Label lblSide1;
22       private System.Windows.Forms.Label lblOutput1;
23       private System.Windows.Forms.Label lblSide2;
24       private System.Windows.Forms.Label lblOutput2;
```

```
25          private System.Windows.Forms.Label lblSide3;
26          private System.Windows.Forms.Label lblOutput3;
27          private System.Windows.Forms.Label lblSide4;
28          private System.Windows.Forms.Label lblOutput4;
29          private System.Windows.Forms.Label lblSide5;
30          private System.Windows.Forms.Label lblOutput5;
31          private System.Windows.Forms.Label lblSide6;
32          private System.Windows.Forms.Label lblOutput6;
33
34          // PictureBoxes to show the two dice
35          private System.Windows.Forms.PictureBox picDie1;
36          private System.Windows.Forms.PictureBox picDie2;
37
38          // Button to roll the dice
39          private System.Windows.Forms.Button btnRoll;
40
41          /// <summary>
42          /// Required designer variable.
43          /// </summary>
44          private System.ComponentModel.Container components = null;
45
46          // die face constants
47          enum FaceNames
48          {
49             ONE = 1,
50             TWO = 2,
51             THREE = 3,
52             FOUR = 4,
53             FIVE = 5,
54             SIX = 6,
55          }
56
57          // declare Random object reference
58          Random m_objRandomNumber = new Random();
59
60          public FrmDiceSimulator()
61          {
62             //
63             // Required for Windows Form Designer support
64             //
65             InitializeComponent();
66
67             //
68             // TODO: Add any constructor code after InitializeComponent
69             // call
70             //
71          }
72
73          /// <summary>
74          /// Clean up any resources being used.
75          /// </summary>
76          protected override void Dispose( bool disposing )
77          {
78             if( disposing )
79             {
80                if (components != null)
81                {
82                   components.Dispose();
83                }
84             }
```

```
85             base.Dispose( disposing );
86          }
87
88          // Windows Form Designer generated code
89
90          /// <summary>
91          /// The main entry point for the application.
92          /// </summary>
93          [STAThread]
94          static void Main()
95          {
96             Application.Run( new FrmDiceSimulator() );
97          }
98
99          // handless Roll Button's Click event
100         private void btnRoll_Click(
101            object sender, System.EventArgs e )
102         {
103            // method randomly assigns a face to each die
104            DisplayDie( picDie1 );
105            DisplayDie( picDie2 );
106
107         } // end method btnRoll_Click
108
109         // get a random die image
110         void DisplayDie( PictureBox picDie )
111         {
112            // generate random integer in range 1 to 6
113            int intFace = m_objRandomNumber.Next( 1, 7 );
114
115            // load corresponding image
116            picDie.Image = Image.FromFile(
117               Directory.GetCurrentDirectory() + "/Images/die" +
118               intFace + ".png" );
119
120            // method displays rolls frequency
121            DisplayFrequency( intFace );
122
123         } // end method DisplayDie
124
125         // display the rolls frequency
126         void DisplayFrequency( int intFace )
127         {
128            // increment and display frequency values
129            switch ( intFace )
130            {
131               case ( int ) FaceNames.ONE:
132                  lblOutput1.Text = Convert.ToString(
133                     Int32.Parse( lblOutput1.Text ) + 1 );
134                  break;
135
136               case ( int ) FaceNames.TWO:
137                  lblOutput2.Text = Convert.ToString(
138                     Int32.Parse( lblOutput2.Text ) + 1 );
139                  break;
140
141               case ( int ) FaceNames.THREE:
142                  lblOutput3.Text = Convert.ToString(
143                     Int32.Parse( lblOutput3.Text ) + 1 );
144                  break;
```

```
145
146             case ( int ) FaceNames.FOUR:
147                lblOutput4.Text = Convert.ToString(
148                   Int32.Parse( lblOutput4.Text ) + 1 );
149                break;
150
151             case ( int ) FaceNames.FIVE:
152                lblOutput5.Text = Convert.ToString(
153                   Int32.Parse( lblOutput5.Text ) + 1 );
154                break;
155
156             case ( int ) FaceNames.SIX:
157                lblOutput6.Text = Convert.ToString(
158                   Int32.Parse( lblOutput6.Text ) + 1 );
159                break;
160
161          } // end switch
162
163       } // end method DisplayFrequency
164
165    } // end class FrmDiceSimulator
166 }
```

16.13 (***Lottery Picker Application***) A lottery commission offers four different lottery games to play: Three-number, Four-number, Five-number and Five-number + 1 lotteries. Each game has independent numbers. Develop an application that randomly picks numbers for all four games and displays the generated numbers in a GUI (Fig. 16.23). The games are played as follows:

- Three-number lotteries require players to choose three numbers in the range of 0–9.
- Four-number lotteries require players to choose four numbers, in the range of 0–9.
- Five-number lotteries require players to choose five numbers in the range of 1–39.
- Five-number + 1 lotteries require players to choose five numbers in the range of 1–49 and an additional number in the range of 1–42.

Figure 16.23 **Lottery Picker** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial16\Exercises\LotteryPicker` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `LotteryPicker.sln` in the `LotteryPicker` directory to open the application.

c) ***Generating random numbers.*** Create a method that will generate the random numbers for all four games.

d) ***Drawing numbers for the games.*** Add code into your application to generate numbers for all four games. To make the applications simple, allow repetition of numbers.

e) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Generate** `Button` multiple times. Make sure the values displayed are within the ranges described in the exercise description.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 16.13 Solution
// LotteryPicker.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace LotteryPicker
{
   /// <summary>
   /// Summary description for FrmLotteryPicker.
   /// </summary>
   public class FrmLotteryPicker : System.Windows.Forms.Form
   {
      // Labels to display the three-number lottery numbers
      private System.Windows.Forms.Label lblThree;
      private System.Windows.Forms.Label lblOutput3;

      // Labels to display the four-number lottery numbers
      private System.Windows.Forms.Label lblFour;
      private System.Windows.Forms.Label lblOutput4;

      // Labels to display the five-number lottery numbers
      private System.Windows.Forms.Label lblFive;
      private System.Windows.Forms.Label lblOutput5;

      // Labels to display the five-number-plus-1 lottery numbers
      private System.Windows.Forms.Label lblFivePlusOne;
      private System.Windows.Forms.Label lblOutput5Plus1;
      private System.Windows.Forms.Label lblOutputExtra1;

      // Button to generate a set of lottery numbers
      private System.Windows.Forms.Button btnGenerate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      Random m_objRandom = new Random();

      public FrmLotteryPicker()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
```

```
54             // call
55             //
56          }
57
58          /// <summary>
59          /// Clean up any resources being used.
60          /// </summary>
61          protected override void Dispose( bool disposing )
62          {
63             if( disposing )
64             {
65                if (components != null)
66                {
67                   components.Dispose();
68                }
69             }
70             base.Dispose( disposing );
71          }
72
73          // Windows Form Designer generated code
74
75          /// <summary>
76          /// The main entry point for the application.
77          /// </summary>
78          [STAThread]
79          static void Main()
80          {
81             Application.Run( new FrmLotteryPicker() );
82          }
83
84          // display random lottery numbers
85          private void btnGenerate_Click(
86             object sender, System.EventArgs e )
87          {
88             // generate three numbers
89             lblOutput3.Text = Generate( 0, 10 ) + " " +
90                Generate( 0, 10 ) + " " + Generate( 0, 10 );
91
92             // generate four numbers
93             lblOutput4.Text = Generate( 0, 10 ) + " " +
94                Generate( 0, 10 ) + " " + Generate( 0, 10 ) + " "
95                + Generate( 0, 10 );
96
97             // generate five numbers
98             lblOutput5.Text = Generate( 1, 40 ) + " " +
99                Generate( 1, 40 ) + " " + Generate( 1, 40 ) +
100               " " + Generate( 1, 40 ) + " " + Generate( 1, 40 );
101
102            // generate five plus one numbers
103            lblOutput5Plus1.Text = Generate( 1, 50 ) + " " +
104               Generate( 1, 50 ) + " " + Generate( 1, 50 ) + " " +
105               Generate( 1, 50 ) + " " + Generate( 1, 50 );
106
107            // generate extra number
108            lblOutputExtra1.Text = Generate( 1, 43 );
109
110         } // end method btnGenerate_Click
111
112         // generate random numbers
113         string Generate( int intLow, int intHigh )
```

```
114       {
115          return String.Format( "{0:D2}",
116             m_objRandom.Next( intLow, intHigh ) );
117
118       } // end method Generate
119
120    } // end class FrmLotteryPicker
121 }
```

What does this code do?

16.14 What does this code do?

```
1  void PickRandomNumbers()
2  {
3     int intNumber1;
4     double dblNumber;
5     int intNumber2;
6     Random objRandom = new Random();
7
8     intNumber1 = objRandom.Next();
9     dblNumber = 5 * objRandom.NextDouble();
10    intNumber2 = objRandom.Next( 1, 10 );
11
12    lblInteger1.Text = Convert.ToString( intNumber1 );
13    lblDouble1.Text = Convert.ToString( dblNumber );
14    lblInteger2.Text = Convert.ToString( intNumber2 );
15
16 } // end method PickRandomNumbers
```

Answer: `intNumber1` gets a positive integer (between `0` and `Int32.MaxValue`), `dblNumber` gets a floating-point number between `0` and `5` (not including `5`) and `intNumber2` gets an integer between `1` and `10` (not including `10`). The complete code reads:

```
1  // Exercise 16.14 Solution
2  // RandomNumber.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace RandomNumber
12 {
13    /// <summary>
14    /// Summary description for FrmRandomNumber.
15    /// </summary>
16    public class FrmRandomNumber : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblInteger2;
19       private System.Windows.Forms.Label lblDouble1;
20       private System.Windows.Forms.Label lblInteger1;
21       private System.Windows.Forms.Label lblRandomInt2;
22       private System.Windows.Forms.Label lblRandomDouble;
23       private System.Windows.Forms.Label lblRandomInt;
24       /// <summary>
25       /// Required designer variable.
```

```
26          /// </summary>
27          private System.ComponentModel.Container components = null;
28
29          public FrmRandomNumber()
30          {
31             //
32             // Required for Windows Form Designer support
33             //
34             InitializeComponent();
35
36             //
37             // TODO: Add any constructor code after InitializeComponent
38             // call
39             //
40          }
41
42          /// <summary>
43          /// Clean up any resources being used.
44          /// </summary>
45          protected override void Dispose( bool disposing )
46          {
47             if( disposing )
48             {
49                if (components != null)
50                {
51                   components.Dispose();
52                }
53             }
54             base.Dispose( disposing );
55          }
56
57          // Windows Form Designer generated code
58
59          /// <summary>
60          /// The main entry point for the application.
61          /// </summary>
62          [STAThread]
63          static void Main()
64          {
65             Application.Run( new FrmRandomNumber() );
66          }
67
68          // calls the PickRandomNumbers helper method
69          private void FrmRandomNumber_Load(
70             object sender, System.EventArgs e )
71          {
72             PickRandomNumbers();
73
74          } // end method FrmRandomNumber_Load
75
76          // chooses and displays several random numbers of various types
77          void PickRandomNumbers()
78          {
79             int intNumber1;
80             double dblNumber;
81             int intNumber2;
82             Random objRandom = new Random();
83
84             intNumber1 = objRandom.Next();
85             dblNumber = 5 * objRandom.NextDouble();
```

```
86          intNumber2 = objRandom.Next( 1, 10 );
87
88          lblInteger1.Text = Convert.ToString( intNumber1 );
89          lblDouble1.Text = Convert.ToString( dblNumber );
90          lblInteger2.Text = Convert.ToString( intNumber2 );
91
92       } // end method PickRandomNumbers
93
94    } // end class FrmRandomNumber
95 }
```

What's wrong with this code?

16.15 This method should assign a random `decimal` number (in the range `0` to `Int32.MaxValue`) to `decimal decNumber`. Find the error(s) in the following code.

```
1  void RandomDecimal()
2  {
3     decimal decNumber;
4     Random objRandom = new Random();
5
6     decNumber = objRandom.NextDouble();
7     lblDisplay.Text = Convert.ToString( decNumber );
8
9  } // end method RandomDecimal
```

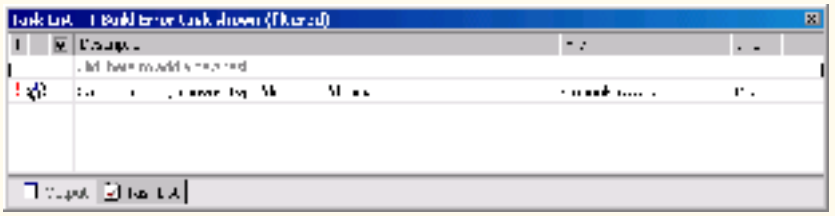
Answer: `Random` objects can produce `int`s and `double`s only; this will yield only an `int` random number. [*Note*: There is no way to have the `Random` class generate a `decimal` random value, so the solution is a bit contrived. The closest approximation is to use `NextDouble`.] The result also must be multiplied by `Int32.MaxValue` to achieve the correct range of values. The complete incorrect code reads:

```
1  // Exercise 16.15 Solution
2  // RandomNumber.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace RandomNumber
12 {
13    /// <summary>
14    /// Summary description for FrmRandomNumber.
15    /// </summary>
16    public class FrmRandomNumber : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.Label lblRandom;
20       /// <summary>
21       /// Required designer variable.
```

```
22          /// </summary>
23          private System.ComponentModel.Container components = null;
24
25          public FrmRandomNumber()
26          {
27             //
28             // Required for Windows Form Designer support
29             //
30             InitializeComponent();
31
32             //
33             // TODO: Add any constructor code after InitializeComponent
34             // call
35             //
36          }
37
38          /// <summary>
39          /// Clean up any resources being used.
40          /// </summary>
41          protected override void Dispose( bool disposing )
42          {
43             if( disposing )
44             {
45                if (components != null)
46                {
47                   components.Dispose();
48                }
49             }
50             base.Dispose( disposing );
51          }
52
53          // Windows Form Designer generated code
54
55          /// <summary>
56          /// The main entry point for the application.
57          /// </summary>
58          [STAThread]
59          static void Main()
60          {
61             Application.Run( new FrmRandomNumber() );
62          }
63
64          // calls the RandomDecimal helper method
65          private void FrmRandomNumber_Load(
66             object sender, System.EventArgs e )
67          {
68             RandomDecimal();
69
70          } // end method FrmRandomNumber_Load
71
72          // generates a random decimal which it then displays
73          void RandomDecimal()
74          {
75             decimal decNumber;
76             Random objRandom = new Random();
77
78             decNumber = objRandom.NextDouble();
79             lblDisplay.Text = Convert.ToString( decNumber );
80
81          } // end method RandomDecimal
```

Should call `Convert.ToDecimal` and multiply the result by `Int32.MaxValue`

```
82
83     } // end class FrmRandomNumber
84  }
```



Answer: The complete corrected code should read:

```
1   // Exercise 16.15 Solution
2   // RandomNumber.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace RandomNumber
12  {
13     /// <summary>
14     /// Summary description for FrmRandomNumber.
15     /// </summary>
16     public class FrmRandomNumber : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Label lblDisplay;
19        private System.Windows.Forms.Label lblRandom;
20        /// <summary>
21        /// Required designer variable.
22        /// </summary>
23        private System.ComponentModel.Container components = null;
24
25        public FrmRandomNumber()
26        {
27           //
28           // Required for Windows Form Designer support
29           //
30           InitializeComponent();
31
32           //
33           // TODO: Add any constructor code after InitializeComponent
34           // call
35           //
36        }
37
38        /// <summary>
39        /// Clean up any resources being used.
40        /// </summary>
41        protected override void Dispose( bool disposing )
42        {
43           if( disposing )
44           {
```

```
45             if (components != null)
46             {
47                components.Dispose();
48             }
49          }
50          base.Dispose( disposing );
51       }
52
53       // Windows Form Designer generated code
54
55       /// <summary>
56       /// The main entry point for the application.
57       /// </summary>
58       [STAThread]
59       static void Main()
60       {
61          Application.Run( new FrmRandomNumber() );
62       }
63
64       // calls the RandomDecimal helper method
65       private void FrmRandomNumber_Load(
66          object sender, System.EventArgs e )
67       {
68          RandomDecimal();
69
70       } // end method FrmRandomNumber_Load
71
72       // generates a random decimal which it then displays
73       void RandomDecimal()
74       {
75          decimal decNumber;
76          Random objRandom = new Random();
77
78          decNumber = ( Int32.MaxValue *
79             Convert.ToDecimal( objRandom.NextDouble() ) );
80          lblDisplay.Text = Convert.ToString( decNumber );
81
82       } // end method RandomDecimal
83
84    } // end class FrmRandomNumber
85 }
```

Programming Challenge

16.16 (*Multiplication Teacher Application*) Develop an application that helps children learn multiplication (Fig. 16.24). Use random-number generation to produce two positive one-digit integers that display in a question, such as "How much is 6 times 7?" The student should type the answer into a `TextBox`. If the answer is correct, then the application randomly displays one of three messages: `"Very Good!"`, `"Excellent!"` or `"Great Job!"` in a `Label` and displays the next question. If the student is wrong, the `Label` displays the message `"No. Please try again"`.

Figure 16.24 Multiplication Teacher application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial16\Exercises\MultiplicationTeacher` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `MultiplicationTeacher.sln` in the `MultiplicationTeacher` directory to open the application.

c) ***Generating the questions***. Add a method into your application to generate each new question.

d) ***Determining whether the right answer was entered***. Add code into your application to call the method created in the previous step. After this method has been called, determine whether the student answered the question correctly, and display the appropriate message.

e) ***Displaying a random message.*** Add a method `GenerateOutput` that displays a random message congratulating the student for answering correctly. This method should be called if the student answered the question correctly.

f) ***Running the application.*** Select **Debug > Start** to run your application. Enter several correct answers and at least one incorrect answer. Verify that `"No. Please try again"` is displayed when you are incorrect, and one of the other responses is displayed at random when you are correct.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 16.16 Solution
2  // MultiplicationTeacher.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace MultiplicationTeacher
12 {
13    /// <summary>
14    /// Summary description for FrmMultiplicationTeacher.
15    /// </summary>
16    public class FrmMultiplicationTeacher : System.Windows.Forms.Form
17    {
18       // Label to display multiplication problem
19       private System.Windows.Forms.Label lblQuestion;
20
21       // TextBox to input answer
22       private System.Windows.Forms.TextBox txtAnswer;
23
24       // Button to submit answer
25       private System.Windows.Forms.Button btnSubmit;
26
27       // Labels to display result
28       private System.Windows.Forms.Label lblResult;
```

```
29          private System.Windows.Forms.Label lblResponse;
30
31          /// <summary>
32          /// Required designer variable.
33          /// </summary>
34          private System.ComponentModel.Container components = null;
35
36          // create new random object
37          Random m_objRandomObject = new Random();
38
39          // random numbers for questions
40          int m_intRandomNumber1;
41          int m_intRandomNumber2;
42          string m_strQuestion; // string for the question being asked
43          int m_intCorrectAnswer; // correct answer to question
44
45          public FrmMultiplicationTeacher()
46          {
47             //
48             // Required for Windows Form Designer support
49             //
50             InitializeComponent();
51
52             //
53             // TODO: Add any constructor code after InitializeComponent
54             // call
55             //
56          }
57
58          /// <summary>
59          /// Clean up any resources being used.
60          /// </summary>
61          protected override void Dispose( bool disposing )
62          {
63             if( disposing )
64             {
65                if (components != null)
66                {
67                   components.Dispose();
68                }
69             }
70             base.Dispose( disposing );
71          }
72
73          // Windows Form Designer generated code
74
75          /// <summary>
76          /// The main entry point for the application.
77          /// </summary>
78          [STAThread]
79          static void Main()
80          {
81             Application.Run( new FrmMultiplicationTeacher() );
82          }
83
84          // handles Load event for FrmMultiplicationTeacher
85          private void FrmMultiplicationTeacher_Load(
86             object sender, System.EventArgs e )
87          {
88             GenerateQuestion(); // generate a question
```

```
89
90          } // end method FrmMultiplicationTeacher_Load
91
92          // handles Submit Button's Click event
93          private void btnSubmit_Click(
94             object sender, System.EventArgs e )
95          {
96             // retrieve user's answer
97             int intAnswer = Int32.Parse( txtAnswer.Text );
98
99             txtAnswer.Clear(); // clear the TextBox
100
101            // check if user answer is correct
102            if ( intAnswer == m_intCorrectAnswer )
103            {
104               GenerateOutput(); // display correct message
105               GenerateQuestion(); // create another question
106            }
107            else // answer was wrong, try again
108            {
109               lblResponse.Text = "No. Please try again.";
110            }
111
112         } // end method btnSubmit_Click
113
114         // generates a new question
115         void GenerateQuestion()
116         {
117            // create two random numbers
118            m_intRandomNumber1 = m_objRandomObject.Next( 0, 10 );
119            m_intRandomNumber2 = m_objRandomObject.Next( 0, 10 );
120
121            // record the correct answer
122            m_intCorrectAnswer = m_intRandomNumber1 *
123               m_intRandomNumber2;
124
125            // construct the question
126            m_strQuestion = "How much is " +
127               Convert.ToString( m_intRandomNumber1 ) + " times " +
128               Convert.ToString( m_intRandomNumber2 ) + "?";
129
130            // display the question
131            lblQuestion.Text = m_strQuestion;
132
133         } // end method GenerateQuestion
134
135         // generates correct message
136         void GenerateOutput()
137         {
138            int intNumber = m_objRandomObject.Next( 0, 3 );
139
140            // show random message
141            switch ( intNumber )
142            {
143               case 0:
144                  lblResponse.Text = "Very Good!";
145                  break;
146
147               case 1:
148                  lblResponse.Text = "Excellent!";
```

```
149                break;
150
151             case 2:
152                lblResponse.Text = "Great Job!";
153                break;
154          }
155
156       } // end method GenerateOutput
157
158    } // end class FrmMultiplicationTeacher
159 }
```

17 TUTORIAL

Flag Quiz Application

Introducing One-Dimensional Arrays and ComboBoxes

Solutions

Instructor's Manual Exercise Solutions Tutorial 17

MULTIPLE-CHOICE QUESTIONS

17.1 Arrays can be declared to hold values of _____.

a) type `double`
b) type `int`
c) type `string`
d) any type

17.2 The elements of an array are related by the fact that they have the same _____.

a) constant value
b) subscript
c) type
d) value

17.3 Method _____ returns the largest index in the array.

a) `GetUpperBound`
b) `GetUpperLimit`
c) `GetLargestIndex`
d) `GetUpperSubscript`

17.4 The first element in every array is the _____.

a) subscript
b) zeroth element
c) length of the array
d) smallest value in the array

17.5 Arrays _____.

a) are controls
b) always have one dimension
c) keep data in sorted order at all times
d) are objects

17.6 The initializer list can _____.

a) be used to determine the size of the array
b) contain a comma-separated list of initial values for the array elements
c) be empty
d) All of the above.

17.7 Which method call sorts array `strWords` in ascending order?

a) `Array.Sort( strWords )`
b) `strWords.SortArray()`
c) `Array.Sort( strWords, 1 )`
d) `Sort( strWords )`

17.8 The `ComboBox` control combines a `TextBox` with a _____ control.

a) `DateTimePicker`
b) `ListBox`
c) `NumericUpDown`
d) `Label`

17.9 To search for a period (`.`) in a `string` called `strTest`, call method _____.

a) `String.Search( strTest, "." )`
b) `String.IndexOf( strTest, "." )`
c) `strTest.IndexOf( "." )`
d) `strTest.Search( "." )`

17.10 Property _____ contains the size of an array.

a) `Elements`
b) `ArraySize`
c) `Length`
d) `Size`

Answers: 17.1) d. 17.2) c. 17.3) a. 17.4) b. 17.5) d. 17.6) d. 17.7) a. 17.8) b. 17.9) c. 17.10) c.

EXERCISES

17.11 (***Enhanced Flag Quiz Application***) Enhance the **Flag Quiz** application by counting the number of questions that were answered correctly. After all the questions have been answered, display a message in a `Label` that describes how well the user performed, as in Fig. 17.32. The following table shows which messages to display:

Number of correct answers	Message
5	`Excellent!`

4	Very good
3	Good
2	Poor
1 or 0	Fail

Figure 17.32 Enhanced **Flag Quiz** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial17\Exercises\FlagQuiz2` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `FlagQuiz.sln` in the `FlagQuiz2` directory to open the application.

c) ***Adding a variable to count the number of correct answers.*** Add an instance variable `m_intNumberCorrect`, and initialize it to `0`. You will use this variable to count the number of correct answers submitted by the user.

d) ***Counting the correct answers.*** Increment `m_intNumberCorrect` in the **Submit** `Button`'s event handler whenever the submitted answer is correct.

e) ***Displaying the message.*** Write a method `DisplayMessage` that displays a message in `lblScore` depending on the value of `m_intNumberCorrect`. Call this method from the **Submit** `Button`'s event handler when the quiz is completed.

f) ***Running the application.*** Select **Debug > Start** to run your application. The finished application should behave as in Fig. 17.32. Run the application a few times and enter a different number of correct answers each time to verify that the correct feedback is displayed.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 17.11 Solution
// FlagQuiz.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace FlagQuiz
{
   /// <summary>
   /// Summary description for FrmFlagQuiz.
   /// </summary>
   public class FrmFlagQuiz : System.Windows.Forms.Form
   {
      // GroupBox with PictureBox inside to display a flag
      private System.Windows.Forms.GroupBox fraFlagGroupBox;
      private System.Windows.Forms.PictureBox picFlag;

      // Label and ComboBox to choose a country name
      private System.Windows.Forms.Label lblChoose;
      private System.Windows.Forms.ComboBox cboOptions;
```

```
25
26          // Label to display result
27          private System.Windows.Forms.Label lblFeedback;
28
29          // Buttons to submit an answer and move to the next flag
30          private System.Windows.Forms.Button btnSubmit;
31          private System.Windows.Forms.Button btnNext;
32
33          // Label to describe how well user performed
34          private System.Windows.Forms.Label lblScore;
35
36          /// <summary>
37          /// Required designer variable.
38          /// </summary>
39          private System.ComponentModel.Container components = null;
40
41          // string array stores country names
42          string[] m_strOptions = {
43             "Russia", "China", "United States", "Italy",
44             "Australia", "South Africa", "Brazil", "Spain" };
45
46          // boolean array tracks displayed flags
47          bool[] m_blnUsed;
48
49          // number of flags shown
50          int m_intCount = 1;
51          string m_strCountry; // current flag's country
52
53          int m_intNumberCorrect = 0;
54
55          public FrmFlagQuiz()
56          {
57             //
58             // Required for Windows Form Designer support
59             //
60             InitializeComponent();
61
62             //
63             // TODO: Add any constructor code after InitializeComponent
64             // call
65             //
66          }
67
68          /// <summary>
69          /// Clean up any resources being used.
70          /// </summary>
71          protected override void Dispose( bool disposing )
72          {
73             if( disposing )
74             {
75                if (components != null)
76                {
77                   components.Dispose();
78                }
79             }
80             base.Dispose( disposing );
81          }
82
83          // Windows Form Designer generated code
84
```

```
85          /// <summary>
86          /// The main entry point for the application.
87          /// </summary>
88          [STAThread]
89          static void Main()
90          {
91             Application.Run( new FrmFlagQuiz() );
92          }
93
94          // handles Flag Quiz Form's Load event
95          private void FrmFlagQuiz_Load(
96             object sender, System.EventArgs e )
97          {
98             // initialize the boolean array
99             m_blnUsed = new bool[ m_strOptions.GetUpperBound( 0 ) ];
100
101            Array.Sort( m_strOptions ); // alphabetize country names
102
103            // display country names in ComboBox
104            cboOptions.DataSource = m_strOptions;
105
106            DisplayFlag(); // display first flag in PictureBox
107
108         } // end method FrmFlagQuiz_Load
109
110         // return full path name of image file as a string
111         string BuildPathName()
112         {
113            // begin with country name
114            string strOutput = m_strCountry;
115
116            // locate space character if there is one
117            int intSpace = strOutput.IndexOf( " " );
118
119            // remove space from country name if there is one
120            if ( intSpace > 0 )
121            {
122               strOutput = strOutput.Remove( intSpace, 1 );
123            }
124
125            strOutput = strOutput.ToLower(); // make chars lowercase
126            strOutput += ".png"; // add file extension
127
128            // add path name
129            strOutput = strOutput.Insert( 0,
130               System.Environment.CurrentDirectory + "\\images\\" );
131
132            return strOutput;  // return full path name
133
134         } // end method BuildPathName
135
136         // return an unused random number
137         int GetUniqueRandomNumber()
138         {
139            Random objRandom = new Random();
140            int intRandom;
141
142            // generate random numbers until unused flag is found
143            do
144            {
```

```
145             intRandom = objRandom.Next( 0, m_blnUsed.Length );
146          } while ( m_blnUsed[ intRandom ] == true );
147
148          // indicate that flag has been used
149          m_blnUsed[ intRandom ] = true;
150
151          return intRandom; // return index for new flag
152
153       } // end method GetUniqueRandomNumber
154
155       // display random flag in PictureBox
156       void DisplayFlag()
157       {
158          // unique index ensures that a flag is used
159          // no more than once
160          int intRandom = GetUniqueRandomNumber();
161
162          // retrieve country name from array m_strOptions
163          m_strCountry = m_strOptions[ intRandom ];
164
165          // get image's full path name
166          string strPath = BuildPathName();
167          picFlag.Image = Image.FromFile( strPath ); // display image
168
169       } // end method DisplayFlag
170
171       // handles Submit Button's Click event
172       private void btnSubmit_Click(
173          object sender, System.EventArgs e )
174       {
175          // retrieve answer from ComboBox
176          string strResponse =
177             Convert.ToString( cboOptions.SelectedValue );
178
179          // verify answer
180          if ( strResponse == m_strCountry )
181          {
182             lblFeedback.Text = "Correct!";
183             m_intNumberCorrect++; // update correct answers counter
184          }
185          else
186          {
187             lblFeedback.Text = "Sorry, incorrect.";
188          }
189
190          // inform user if quiz is over
191          if ( m_intCount >= 5 ) // quiz is over
192          {
193             lblFeedback.Text += "   Done!";
194             btnNext.Enabled = false;
195             btnSubmit.Enabled = false;
196             cboOptions.Enabled = false;
197
198             DisplayMessage();
199          }
200          else // quiz is not over
201          {
202             btnSubmit.Enabled = false;
203             btnNext.Enabled = true;
204          }
```

```
205
206        } // end method btnSubmit_Click
207
208        // handles Next Button's Click event
209        private void btnNext_Click(
210           object sender, System.EventArgs e )
211        {
212           DisplayFlag(); // display next flag
213           lblFeedback.Text = ""; // clear output
214
215           // change selected country to first in ComboBox
216           cboOptions.SelectedIndex = 0;
217
218           m_intCount++; // update number of flags shown
219
220           btnSubmit.Enabled = true;
221           btnNext.Enabled = false;
222
223        } // end method btnNext_Click
224
225        // displays quiz score to user
226        void DisplayMessage()
227        {
228           switch ( m_intNumberCorrect )
229           {
230              case 5:
231                 lblScore.Text = "Excellent";
232                 break;
233
234              case 4:
235                 lblScore.Text = "Very good";
236                 break;
237
238              case 3:
239                 lblScore.Text = "Good";
240                 break;
241
242              case 2:
243                 lblScore.Text = "Poor";
244                 break;
245
246              default:
247                 lblScore.Text = "Fail";
248                 break;
249           }
250
251        } // end method DisplayMessage
252
253     } // end class FrmFlagQuiz
254  }
```

17.12 (***Salary Survey Application***) Use a one-dimensional array to solve the following problem: A company pays its salespeople on a commission basis. The salespeople receive $200 per week, plus 9% of their gross sales for that week. For example, a salesperson who grosses $5000 in sales in a week receives $200 plus 9% of $5000, a total of $650. Write an application (using an array of counters) that determines how many of the salespeople earned salaries in each of the following ranges (assuming that each salesperson's salary is truncated to an integer amount): $200–299, $300–399, $400–499, $500–599, $600–699, $700–799, $800–899, $900–999 and over $999.

Allow the user to enter the sales for each employee in a `TextBox`. The user should click the **Calculate** `Button` to calculate that salesperson's salary. When the user is done entering this information, clicking the **Show Totals** `Button` should display how many of the salespeople earned salaries in each of the above ranges. The finished application should behave like Fig. 17.33.

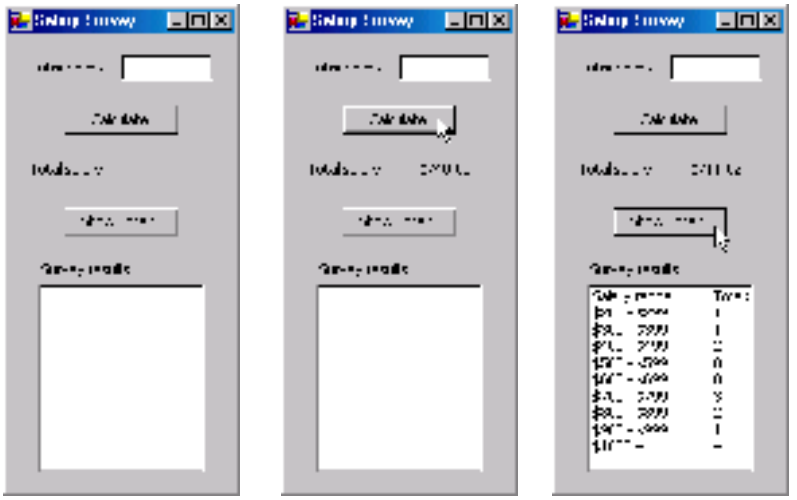


Figure 17.33 **Salary Survey** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial17\Exercises\SalarySurvey` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SalarySurvey.sln` in the `SalarySurvey` directory to open the application.

c) ***Creating an array of salary ranges.*** Create a `string` array, and initialize it to contain the salary ranges (the `string`s displayed in the `ListBox`'s first column).

d) ***Create an array that represents the number of salaries in each range.*** Create an empty `int` array to store the number of employees who earn salaries in each range.

e) ***Creating a `Load` event handler for the `Form`.*** Write event handler `FrmSalarySurvey_Load`. Initialize the `int` array to contain default values.

f) ***Creating an event handler for the Calculate `Button`.*** Write event handler `btnCalculate_Click`. Obtain the user input from the **Enter sales:** `TextBox`. If the user enters a negative value, display a `MessageBox` asking for non-negative input and exit the event handler. Otherwise, calculate the commission due to the employee and add that amount to the base salary. Increment the element in array `decSalaries` that corresponds to the employee's salary range. Use a series of `if`...`else` statements to determine which element should be incremented. This event handler should also display the employee's salary in the **Total salary:** `Label`.

g) ***Writing an event handler for the Show Totals `Button`.*** Create event handler `btnShowTotals_Click` to display the salary distribution in the `ListBox`. Use a `for` statement to display the range (an element in `strSalaryRanges`) and the number of employees whose salary falls in that range (an element in `decSalaries`).

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter several sales amounts using the **Calculate** `Button`. Click the **Show Totals** `Button` and verify that the proper amounts are displayed for each salary range, based on the salaries calculate from your input.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 17.12 Solution
2  // SalarySurvey.cs
3
4  using System;
```

```
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace SalarySurvey
12 {
13    /// <summary>
14    /// Summary description for FrmSalarySurvey.
15    /// </summary>
16    public class FrmSalarySurvey : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input sales for an employee
19       private System.Windows.Forms.Label lblInputSales;
20       private System.Windows.Forms.TextBox txtInputSales;
21
22       // Button to calculate an employee's salary
23       private System.Windows.Forms.Button btnCalculate;
24
25       // Labels to display an employee's salary
26       private System.Windows.Forms.Label lblTotalSalaryLabel;
27       private System.Windows.Forms.Label lblTotalSalary;
28
29       // Button to display number of employees who have
30       // salaries in various ranges
31       private System.Windows.Forms.Button btnShowTotals;
32
33       // Label and ListBox to display number of employees who have
34       // salaries in various ranges
35       private System.Windows.Forms.Label lblSurveyResults;
36       private System.Windows.Forms.ListBox lstSalaryTotals;
37
38       /// <summary>
39       /// Required designer variable.
40       /// </summary>
41       private System.ComponentModel.Container components = null;
42
43       // salary ranges
44       string[] m_strSalaryRanges = {
45          "$200 - $299", "$300 - $399", "$400 - $499",
46          "$500 - $599", "$600 - $699", "$700 - $799",
47          "$800 - $899", "$900 - $999", "$1000 +  " };
48
49       // number of employees in each salary range
50       int[] m_intSalaries;
51
52       public FrmSalarySurvey()
53       {
54          //
55          // Required for Windows Form Designer support
56          //
57          InitializeComponent();
58
59          //
60          // TODO: Add any constructor code after InitializeComponent
61          // call
62          //
63       }
64
```

```
65          /// <summary>
66          /// Clean up any resources being used.
67          /// </summary>
68          protected override void Dispose( bool disposing )
69          {
70             if( disposing )
71             {
72                if (components != null)
73                {
74                   components.Dispose();
75                }
76             }
77             base.Dispose( disposing );
78          }
79
80          // Windows Form Designer generated code
81
82          /// <summary>
83          /// The main entry point for the application.
84          /// </summary>
85          [STAThread]
86          static void Main()
87          {
88             Application.Run( new FrmSalarySurvey() );
89          }
90
91          // handles Form Load event
92          private void FrmSalarySurvey_Load(
93             object sender, System.EventArgs e )
94          {
95             m_intSalaries =
96                new int[ m_strSalaryRanges.GetUpperBound( 0 ) + 1 ];
97
98          } // end method FrmSalarySurvey_Load
99
100         // handles Calculate Button's Click event
101         private void btnCalculate_Click(
102            object sender, System.EventArgs e )
103         {
104            // obtain total sales
105            decimal decSales = Decimal.Parse( txtInputSales.Text );
106
107            // check for negative input
108            if ( decSales < 0M )
109            {
110               MessageBox.Show( "Please enter a positive sales value.",
111                  "Invalid Input", MessageBoxButtons.OK,
112                  MessageBoxIcon.Exclamation );
113            }
114            else
115            {
116               // employee's base salary
117               decimal decTotalSalary = 200M;
118
119               // add commission to total salary
120               decTotalSalary += decSales * 0.09M;
121
122               // display salary in a Label
123               lblTotalSalary.Text =
124                  String.Format( "{0:C}", decTotalSalary );
```

```
125
126             // increment the correct counter in array intSalaries
127             if ( decTotalSalary < 300M )
128             {
129                m_intSalaries[ 0 ]++;
130             }
131             else if ( decTotalSalary < 400M )
132             {
133                m_intSalaries[ 1 ]++;
134             }
135             else if ( decTotalSalary < 500M )
136             {
137                m_intSalaries[ 2 ]++;
138             }
139             else if ( decTotalSalary < 600M )
140             {
141                m_intSalaries[ 3 ]++;
142             else if ( decTotalSalary < 700M )
143             {
144                m_intSalaries[ 4 ]++;
145             }
146             else if ( decTotalSalary < 800M )
147             {
148                m_intSalaries[ 5 ]++;
149             }
150             else if ( decTotalSalary < 900M )
151             {
152                m_intSalaries[ 6 ]++;
153             }
154             else if ( decTotalSalary < 1000M )
155             {
156                m_intSalaries[ 7 ]++;
157             }
158             else if ( decTotalSalary >= 1000M )
159             {
160                m_intSalaries[ 8 ]++;
161             }
162
163             txtInputSales.Clear(); // clear TextBox
164
165          } // end else
166
167       } // end method btnCalculate_Click
168
169       // handles Show Total Button's Click event
170       private void btnShowTotals_Click(
171          object sender, System.EventArgs e )
172       {
173          int intIndex = 0; // counter
174
175          // clear all items in the ListBox
176          lstSalaryTotals.Items.Clear();
177
178          // add header to ListBox
179          lstSalaryTotals.Items.Add( "Salary range:\tTotal" );
180
181          // displays total for each salary range
182          for ( intIndex = 0; intIndex <=
183             m_strSalaryRanges.GetUpperBound( 0 ); intIndex++ )
184          {
```

```
185             lstSalaryTotals.Items.Add( m_strSalaryRanges[ intIndex ]
186                + "\t" + m_intSalaries[ intIndex ] );
187          }
188
189       } // end method btnShowTotals_Click
190
191    } // end class FrmSalarySurvey
192 }
```

17.13 (*Cafeteria Survey Application*) Twenty students were asked to rate, on the scale from 1 to 10, the quality of the food in the student cafeteria, with 1 being "awful" and 10 being "excellent." Allow the user input to be entered using a `ComboBox`. Place the 20 responses in an `int` array, and determine the frequency of each rating. Display the frequencies as a histogram in a multiline, scrollable `TextBox`. Figure 17.34 demonstrates the completed application.

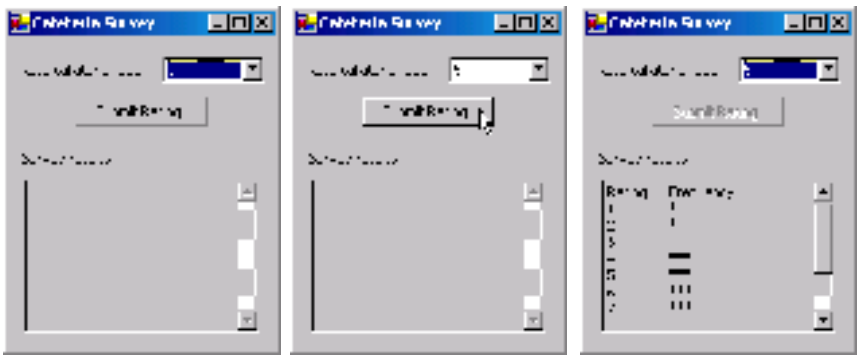


Figure 17.34 Cafeteria Survey GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial17\Exercises\CafeteriaSurvey` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CafeteriaSurvey.sln` in the `CafeteriaSurvey` directory to open the application.

c) ***Creating an array of the possible ratings.*** Create an array of 10 consecutive integers, called `m_intChoices` to contain the integers in the range 1–10, inclusive.

d) ***Adding a ComboBox.*** Add a `ComboBox` to the GUI as in Fig. 17.34. The `ComboBox` will display the possible ratings. Set property `DropDownStyle` to `DropDownList`. Rearrange and comment the new control declaration appropriately.

e) ***Displaying the possible ratings when the application starts.*** Write the event handler for the `Load` event so that the `DataSource` of the `ComboBox` is set to `intChoices` when the application starts.

f) ***Creating an array to store the responses.*** Create an `int` array of length `11` named `m_intResponses`. This will be used to store the number of responses in each of the 10 categories (element `0` will not be used).

g) ***Counting the number of responses.*** Create an `int` variable named `m_intResponseCounter` to keep track of how many responses have been input.

h) ***Storing the responses.*** Write the event handler `btnSubmit_Click` to increment `m_intResponseCounter`. Store the response in array `m_intResponses`. Call method `DisplayHistogram` to display the results.

i) ***Creating method `DisplayHistogram`.*** Add a header to the `TextBox`. Use nested `for` loops to display the ratings in the first column. The second column uses asterisks to indicate how many students surveyed submitted the corresponding rating.

j) ***Running the application.*** Select **Debug > Start** to run your application. Enter 20 responses using the **Submit Rating** `Button`. Verify that the resulting histogram displays the responses entered.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 17.13 Solution
// CafeteriaSurvey.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CafeteriaSurvey
{
   /// <summary>
   /// Summary description for FrmCafeteriaSurvey.
   /// </summary>
   public class FrmCafeteriaSurvey : System.Windows.Forms.Form
   {
      // Label and ComboBox to input cafeteria food rating
      private System.Windows.Forms.Label lblInput;
      private System.Windows.Forms.ComboBox cboInput;

      // Button to submit rating
      private System.Windows.Forms.Button btnSubmit;

      // Label and TextBox to display survey results
      private System.Windows.Forms.Label lblOutput;
      private System.Windows.Forms.TextBox txtOutput;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // possible answers
      int[] m_intChoices = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

      // counter for number of responses
      int m_intResponseCounter = 0;

      // array to keep track of all responses
      int[] m_intResponses = new int[ 11 ];

      public FrmCafeteriaSurvey()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
```

```
59          protected override void Dispose( bool disposing )
60          {
61             if( disposing )
62             {
63                if (components != null)
64                {
65                   components.Dispose();
66                }
67             }
68             base.Dispose( disposing );
69          }
70
71          // Windows Form Designer generated code
72
73          /// <summary>
74          /// The main entry point for the application.
75          /// </summary>
76          [STAThread]
77          static void Main()
78          {
79             Application.Run( new FrmCafeteriaSurvey() );
80          }
81
82          // displays histogram
83          void DisplayHistogram()
84          {
85             // construct output string with the frequencies
86             // as a histogram
87             string strOutput = "Rating\tFrequency\r\n";
88
89             int intRatings;
90             int intCounter;
91
92             // add entry to TextBox for each rating
93             for ( intRatings = 1; intRatings <= 10; intRatings++ )
94             {
95                strOutput += ( Convert.ToString( intRatings ) + "\t" );
96
97                for ( intCounter = 1; intCounter <=
98                   m_intResponses[ intRatings ]; intCounter++ )
99                {
100                  strOutput += "*";
101               }
102
103               strOutput += "\r\n";
104            }
105
106            txtOutput.Text = strOutput; // display results in TextBox
107
108         } // end method DisplayHistogram
109
110         // handles Submit Button's Click event
111         private void btnSubmit_Click(
112            object sender, System.EventArgs e )
113         {
114            // retrieve user input
115            int intResponse =
116               Convert.ToInt32( cboInput.SelectedItem );
117
118            if ( m_intResponseCounter < 20 )
```

```
119          {
120             m_intResponses[ intResponse ]++;
121          }
122
123          m_intResponseCounter++;
124
125          if ( m_intResponseCounter == 20 )
126          {
127             // disable btnSubmit Button so no more
128             // responses can be entered
129             btnSubmit.Enabled = false;
130
131             DisplayHistogram();
132          }
133
134       } // end class btnSubmit_Click
135
136       // handles Form Load event
137       private void FrmCafeteriaSurvey_Load(
138          object sender, System.EventArgs e )
139       {
140          cboInput.DataSource = m_intChoices;
141
142       } // end class FrmCafeteriaSurvey_Load
143
144    } // end class FrmCafeteriaSurvey
145 }
```

What does this code do?

17.14 Assume that `intNumbers` is an `int` array. What does `intTempArray` contain after the execution of the `for` statement?

```
1  int intLength = intNumbers.Length;
2  int[] intTempArray = new int[ intLength ];
3
4  intLength--;
5
6  for ( int intI = intLength; intI >= 0; intI-- )
7  {
8     intTempArray[ intI ] = intNumbers[ intLength - intI ];
9  }
```

Answer: This code takes the contents of the parameter `intNumbers` and reverses the order of its contents, which are then stored in `intTempArray`. The complete code reads:

```
1  // Exercise 17.14 Solution
2  // ReverseArray.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ReverseArray
12 {
13    /// <summary>
```

```
14    /// Summary description for FrmReverse.
15    /// </summary>
16    public class FrmReverse : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.ListBox lstReversed;
19       private System.Windows.Forms.Label lblReversed;
20       private System.Windows.Forms.ListBox lstOriginal;
21       private System.Windows.Forms.Label lblOriginal;
22       /// <summary>
23       /// Required designer variable.
24       /// </summary>
25       private System.ComponentModel.Container components = null;
26
27       public FrmReverse()
28       {
29          //
30          // Required for Windows Form Designer support
31          //
32          InitializeComponent();
33
34          //
35          // TODO: Add any constructor code after InitializeComponent
36          // call
37          //
38       }
39
40       /// <summary>
41       /// Clean up any resources being used.
42       /// </summary>
43       protected override void Dispose( bool disposing )
44       {
45          if( disposing )
46          {
47             if (components != null)
48             {
49                components.Dispose();
50             }
51          }
52          base.Dispose( disposing );
53       }
54
55       // Windows Form Designer generated code
56
57       /// <summary>
58       /// The main entry point for the application.
59       /// </summary>
60       [STAThread]
61       static void Main()
62       {
63          Application.Run( new FrmReverse() );
64       }
65
66       // calls the Mystery method
67       private void FrmReverse_Load(
68          object sender, System.EventArgs e )
69       {
70          int[] intNumbers = { 1, 2, 3, 4, 5 };
71
72          int intLength = intNumbers.Length;
73          int[] intTempArray = new int[ intLength ];
```

```
74
75          intLength--;
76
77          for ( int intI = intLength; intI >= 0; intI-- )
78          {
79             intTempArray[ intI ] = intNumbers[ intLength - intI ];
80          }
81
82          // instructs the ListBox to fill itself with this array
83          lstOriginal.DataSource = intNumbers;
84
85          // calls Mystery, then instructs the ListBox to fill itself
86          // with the new array
87          lstReversed.DataSource = intTempArray;
88
89       } // end method FrmReverse_Load
90
91    } // end class FrmReverse
92 }
```

What's wrong with this code?

17.15 The code that follows uses a `for` loop to sum the elements in an array. Find the error(s) in the following code:

```
1  int SumArray()
2  {
3     int intSum = 0;
4     int[] intNumbers = { 1, 2, 3, 4, 5, 6, 7, 8 };
5
6     for ( int intCounter = 0; intCounter <= intNumbers.Length;
7        intCounter++ )
8     {
9        intSum += intNumbers[ intCounter ];
10    }
11
12    return intSum;
13
14 } // end method SumArray
```

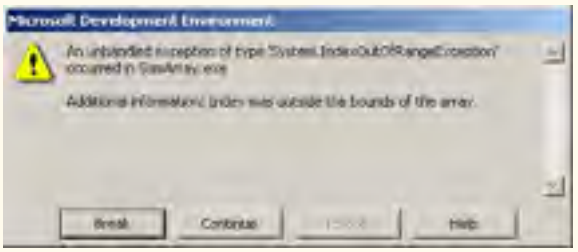
Answer: Array `intNumbers` does have `intNumbers.Length` number of elements, but the indices are zero through `intNumbers.Length - 1`. The `for` loop increments `intCounter` beyond the highest index in the array which results in a run-time error. To correct the error, change the loop-continuation condition to use < rather than <=. The complete incorrect code reads:

```
1  // Exercise 17.15 Solution
2  // SumArray.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
```

```
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SumArray
{
   /// <summary>
   /// Summary description for FrmSumArray.
   /// </summary>
   public class FrmSumArray : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblSumOutput;
      private System.Windows.Forms.Label lblSum;
      private System.Windows.Forms.Label lblArrayContents;
      private System.Windows.Forms.Label lblArray;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmSumArray()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmSumArray() );
      }

```

```
66          // prints the result of the SumArray method
67          private void FrmSumArray_Load(
68              object sender, System.EventArgs e )
69          {
70             lblSumOutput.Text = Convert.ToString( SumArray() );
71
72          } // end method FrmSumArray_Load
73
74          // returns the sum of all of an array's elements
75          int SumArray()
76          {
77             int intSum = 0;
78             int[] intNumbers = { 1, 2, 3, 4, 5, 6, 7, 8 };
79
80             for ( int intCounter = 0; intCounter <= intNumbers.Length;
81                intCounter++ )
82             {
83                intSum += intNumbers[ intCounter ];
84             }
85
86             return intSum;
87
88          } // end method SumArray
89
90       } // end class FrmSumArray
91    }
```

Should use < rather than <=



Answer: The complete corrected code should read:

```
1   // Exercise 17.15 Solution
2   // SumArray.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace SumArray
12  {
13     /// <summary>
14     /// Summary description for FrmSumArray.
15     /// </summary>
16     public class FrmSumArray : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Label lblSumOutput;
19        private System.Windows.Forms.Label lblSum;
20        private System.Windows.Forms.Label lblArrayContents;
```

```
21          private System.Windows.Forms.Label lblArray;
22          /// <summary>
23          /// Required designer variable.
24          /// </summary>
25          private System.ComponentModel.Container components = null;
26
27          public FrmSumArray()
28          {
29             //
30             // Required for Windows Form Designer support
31             //
32             InitializeComponent();
33
34             //
35             // TODO: Add any constructor code after InitializeComponent
36             // call
37             //
38          }
39
40          /// <summary>
41          /// Clean up any resources being used.
42          /// </summary>
43          protected override void Dispose( bool disposing )
44          {
45             if( disposing )
46             {
47                if (components != null)
48                {
49                   components.Dispose();
50                }
51             }
52             base.Dispose( disposing );
53          }
54
55          // Windows Form Designer generated code
56
57          /// <summary>
58          /// The main entry point for the application.
59          /// </summary>
60          [STAThread]
61          static void Main()
62          {
63             Application.Run( new FrmSumArray() );
64          }
65
66          // prints the result of the SumArray method
67          private void FrmSumArray_Load(
68             object sender, System.EventArgs e )
69          {
70             lblSumOutput.Text = Convert.ToString( SumArray() );
71
72          } // end method FrmSumArray_Load
73
74          // returns the sum of all of an array's elements
75          int SumArray()
76          {
77             int intSum = 0;
78             int[] intNumbers = { 1, 2, 3, 4, 5, 6, 7, 8 } ;
79
80             for ( int intCounter = 0; intCounter < intNumbers.Length;
```

```
            intCounter++ )
         {
            intSum += intNumbers[ intCounter ];
         }

         return intSum;

      } // end method SumArray

   } // end class FrmSumArray
}
```

Programming Challenge

17.16 (***Road Sign Test Application***) Write an application that will test the user's knowledge of road signs. Your application should display a random sign image and ask the user to select the sign name from a `ComboBox`. This application should look like Fig. 17.35. [*Hint*: The application is similar to the **Flag Quiz** application.] You can find the images in `C:\Examples\Tutorial17\Exercises\images`.

Figure 17.35 Road Sign Test GUI.

Answer:

```
// Exercise 17.16 Solution
// RoadSignTest.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace RoadSignTest
{
   /// <summary>
   /// Summary description for FrmRoadSignTest.
   /// </summary>
   public class FrmRoadSignTest : System.Windows.Forms.Form
   {
      // GroupBox with PictureBox inside displaying road sign
      private System.Windows.Forms.GroupBox fraRoadSign;
      private System.Windows.Forms.PictureBox picSign;

      // Label and ComboBox to choose answer
      private System.Windows.Forms.Label lblChoose;
      private System.Windows.Forms.ComboBox cboOptions;
```

```
25
26          // Label to display result of answer
27          private System.Windows.Forms.Label lblFeedback;
28
29          // Button to submit an answer
30          private System.Windows.Forms.Button btnSubmit;
31
32          // Button to move to the next sign
33          private System.Windows.Forms.Button btnNext;
34
35          /// <summary>
36          /// Required designer variable.
37          /// </summary>
38          private System.ComponentModel.Container components = null;
39
40          // string array stores sign names
41          string[] m_strOptions = {
42             "Do Not Enter", "Narrow bridge", "No bicycles",
43             "No left turn", "No Pedestrians", "No U-turn",
44             "Road Narrows", "Stop", "Stop sign ahead",
45             "Traffic signals ahead", "Winding road ahead", "Yield" };
46
47          // bool array tracks displayed signs
48          bool[] m_blnUsed;
49
50          int m_intCount = 1; // number of signs shown
51          int m_intCorrectAnswer; // index of current sign
52
53          public FrmRoadSignTest()
54          {
55             //
56             // Required for Windows Form Designer support
57             //
58             InitializeComponent();
59
60             //
61             // TODO: Add any constructor code after InitializeComponent
62             // call
63             //
64          }
65
66          /// <summary>
67          /// Clean up any resources being used.
68          /// </summary>
69          protected override void Dispose( bool disposing )
70          {
71             if( disposing )
72             {
73                if (components != null)
74                {
75                   components.Dispose();
76                }
77             }
78             base.Dispose( disposing );
79          }
80
81          // Windows Form Designer generated code
82
83          /// <summary>
84          /// The main entry point for the application.
```

```
85          /// </summary>
86          [STAThread]
87          static void Main()
88          {
89             Application.Run( new FrmRoadSignTest() );
90          }
91
92          // handles Form Load event
93          private void FrmRoadSignTest_Load(
94             object sender, System.EventArgs e )
95          {
96             m_blnUsed = new bool[ m_strOptions.GetUpperBound( 0 ) ];
97
98             Array.Sort( m_strOptions ); // alphabetize sign names
99
100            // display sign names in ComboBox
101            cboOptions.DataSource = m_strOptions;
102
103            DisplaySign(); // display first sign in PictureBox
104
105         } // end method FrmRoadSignTest_Load
106
107         // return full path name of image file as a string
108         string BuildPathName()
109         {
110            // return full path name
111            return System.Environment.CurrentDirectory +
112               "\\images\\sign" + m_intCorrectAnswer + ".png";
113
114         } // end method BuildPathName
115
116         // return an unused random number
117         int GetUniqueRandomNumber()
118         {
119            Random objRandom = new Random();
120            int intRandom;
121
122            // generate random numbers until unused sign is found
123            do
124            {
125               intRandom = objRandom.Next( 0, m_blnUsed.Length );
126            } while ( m_blnUsed[ intRandom ] == true );
127
128            // indicate that flag has been used
129            m_blnUsed[ intRandom ] = true;
130
131            return intRandom; // return index for new sign
132
133         } // end method GetUniqueRandomNumber
134
135         // display random sign in PictureBox
136         void DisplaySign()
137         {
138            // unique index ensures that a sign is used
139            // no more than once
140            m_intCorrectAnswer = GetUniqueRandomNumber();
141
142            // get image's full path name
143            string strPath = BuildPathName();
144
```

```
145          picSign.Image = Image.FromFile( strPath ); // display image
146
147       } // end method DisplayFlag
148
149       // handles Submit Button's Click event
150       private void btnSubmit_Click(
151          object sender, System.EventArgs e )
152       {
153          // retrieve answer from ComboBox
154          string strResponse =
155             Convert.ToString( cboOptions.SelectedValue );
156
157          // verify answer
158          if ( strResponse == m_strOptions[ m_intCorrectAnswer ] )
159          {
160             lblFeedback.Text = "Correct!";
161          }
162          else
163          {
164             lblFeedback.Text = "Incorrect.";
165          }
166
167          // inform user if test is over
168          if ( m_intCount >= 5 ) // test is over
169          {
170             lblFeedback.Text += "   Done!";
171             btnNext.Enabled = false;
172             btnSubmit.Enabled = false;
173          }
174          else // test is not over
175          {
176             btnSubmit.Enabled = false;
177             btnNext.Enabled = true;
178          }
179
180       } // end method btnSubmit_Click
181
182       // handles Next Button's Click event
183       private void btnNext_Click(
184          object sender, System.EventArgs e )
185       {
186          DisplaySign(); // display next sign
187          lblFeedback.Text = ""; // clear output
188
189          // change selected sign to first in ComboBox
190          cboOptions.SelectedIndex = 0;
191
192          m_intCount++; // update number of signs shown
193
194          btnSubmit.Enabled = true;
195          btnNext.Enabled = false;
196
197       } // end method btnNext_Click
198
199    } // end class FrmRoadSignTest
200 }
```

18 TUTORIAL

Student Grades Application

Introducing Two-Dimensional Arrays and `RadioButtons`

Solutions

Instructor's Manual Exercise Solutions Tutorial 18

MULTIPLE-CHOICE QUESTIONS

18.1 `RadioButton` controls should be prefixed with _____.

a) `rad` b) `rbn`
c) `btn` d) `radbtn`

18.2 A two-dimensional array in which each row contains the same number of columns is called a _____ array.

a) data b) rectangular
c) tabular d) All of the above.

18.3 In an *m*-by-*n* array, the *m* stands for _____.

a) the number of columns in the array b) the total number of array elements
c) the number of rows in the array d) the number of elements in each row

18.4 The _____ statement assigns an array of three columns and five rows to two-dimensional `int` array `intArray`.

a) `intArray = new int[ 5, 3 ];` b) `intArray = new int[ 4, 2 ];`
c) `intArray = new int[ 4, 3 ];` d) `intArray = new int[ 5, 2 ];`

18.5 A `RadioButton` is a type of _____ button.

a) check b) change
c) state d) action

18.6 Use a _____ to group `RadioButtons` on the `Form`.

a) `GroupBox` control b) `ComboBox` control
c) `ListBox` control d) None of the above.

18.7 The _____ event handler is invoked when selecting a `RadioButton` control.

a) `Selected` b) `CheckedChanged`
c) `ButtonChanged` d) `CheckSelected`

18.8 The _____ property is set to `true` when a `RadioButton` is selected.

a) `Selected` b) `Chosen`
c) `On` d) `Checked`

18.9 Two-dimensional arrays are often used to represent _____.

a) a pie chart b) distances
c) lines d) tables

18.10 The statement _____ assigns an array of three columns and three rows to two-dimensional `int` array `intArray`.

a) `int[][] intArray = { { 1 2 3 } { 4 5 6 } { 7 8 9 } };`
b) `int[,] intArray = new int( { { 1 2 3 } { 4 5 6 } { 7 8 9 } };`
c) `int[,] intArray = { { 1 2 3 } { 4 5 6 } { 7 8 9 } };`
d) `int[] intArray = { { 1 2 3 } { 4 5 6 } { 7 8 9 } } );`

Answers: 18.1) a. 18.2) b. 18.3) c. 18.4) a. 18.5) c. 18.6) a. 18.7) b. 18.8) d. 18.9) d. 18.10) c.

EXERCISES

18.11 ***(Food Survey Application)*** A school cafeteria is giving an electronic survey to its students to improve their lunch menu. Create an application that will use a two-dimensional array to hold counters for the survey (Fig. 18.18). You will also provide `RadioButtons` for the students to indicate whether they like or dislike a particular food.

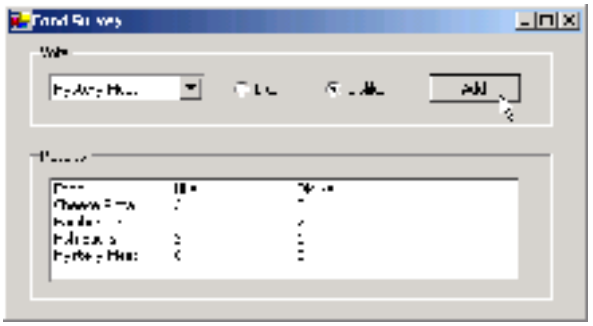


Figure 18.18 **Food Survey** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial18\Exercises\FoodSurvey` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `FoodSurvey.sln` in the `FoodSurvey` directory to open the application.

c) ***Adding RadioButtons to the Vote GroupBox.*** Add two `RadioButtons` to the **Vote** `GroupBox`. Name one `radLike` and the other `radDislike`. Change their `Text` properties to `Like` and `Dislike`, respectively. Set the `Checked` property of `radLike` to `true`. Rearrange and comment the control declarations appropriately.

d) ***Declaring a two-dimensional int array.*** Declare a two-dimensional `int` array named `m_intDisplay`, with 4 rows and 2 columns.

e) ***Creating event handler btnAdd_Click.*** Generate the `Click` event handler for the **Add** `Button`. In the event handler, clear the `ListBox`, then add the header `"Food\t\tLike\t\tDislike"`. Create a local `int` variable `intIndex`. This variable should contain the index of the selected item in `cboFoods`.

f) ***Using a for loop to display the data.*** Insert a `for` statement into the event handler to loop through each row in the `m_strFoods` array (rows 0–3). In the body of the loop, insert an `if` statement that checks if the `radLike RadioButton` is selected and if the variable `intIndex` is equal to the counter of your `for` statement. If both conditions are `true`, increment the counter in column 0 in the `m_intDisplay` array. Insert an `else if` statement that determines whether the `radDislike RadioButton` is selected and whether the variable `intIndex` is equal to the counter of your `for` statement. If both conditions are `true`, increment the counter in column 1 of the `m_intDisplay` array.

g) ***Adding the current row to the ListBox.*** Inside the `for` statement, add the current row to the `ListBox`. The counter variable of the `for` statement will be used as the index of the `m_strFoods` array. Use the `"\t"` escape sequence to align the results.

h) ***Running the application.*** Select **Debug > Start** to run your application. Choose either the **Like** or **Dislike** `RadioButton`. Click the **Add** `Button` and check to make sure all `strings` and numbers in the **Results** `GroupBox` are correct. Add several other selections to the **Food Survey** and make sure that the numbers are correct.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 18.11 Solution
2  // FoodSurvey.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace FoodSurvey
```

```
{
   /// <summary>
   /// Summary description for FrmFoodSurvey.
   /// </summary>
   public class FrmFoodSurvey : System.Windows.Forms.Form
   {
      // GroupBox for voting controls
      private System.Windows.Forms.GroupBox fraVote;

      // ComboBox to give choice of foods
      private System.Windows.Forms.ComboBox cboFoods;

      // RadioButtons to select opinion of food
      private System.Windows.Forms.RadioButton radLike;
      private System.Windows.Forms.RadioButton radDislike;

      // Button to add vote to results
      private System.Windows.Forms.Button btnAdd;

      // GroupBox with ListBox to display results
      private System.Windows.Forms.GroupBox fraResults;
      private System.Windows.Forms.ListBox lstResults;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // string array for food choices
      string[] m_strFoods = { "Cheese Pizza", "Hamburger",
                              "Fish Sticks", "Mystery Meat" };

      // two-dimensional array to store survey results
      int[,] m_intResults = new int[ 4, 2 ];

      public FrmFoodSurvey()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
```

```
72             base.Dispose( disposing );
73          }
74
75          // Windows Form Designer generated code
76
77          /// <summary>
78          /// The main entry point for the application.
79          /// </summary>
80          [STAThread]
81          static void Main()
82          {
83             Application.Run( new FrmFoodSurvey() );
84          }
85
86          // handles Form's Load event
87          private void FrmFoodSurvey_Load(
88             object sender, System.EventArgs e )
89          {
90             // display foods in ComboBox
91             cboFoods.DataSource = m_strFoods;
92
93          } // end method FrmFoodSurvey_Load
94
95          // handles Add Button's Click event
96          private void btnAdd_Click(
97             object sender, System.EventArgs e )
98          {
99             // clear ListBox and output header
100            lstResults.Items.Clear();
101            lstResults.Items.Add( "Food\t\tLike\t\tDislike" );
102
103            // index of selected food
104            int intIndex = cboFoods.SelectedIndex;
105
106            // for loop to display 2-dimensional array
107            for ( int intRow = 0; intRow < 4; intRow++ )
108            {
109               // increment appropriate counter
110               if ( radLike.Checked == true && intIndex == intRow )
111               {
112                  m_intResults[ intRow, 0 ]++;
113               }
114               else if ( radDislike.Checked == true
115                  && intIndex == intRow )
116               {
117                  m_intResults[ intRow, 1 ]++;
118               }
119
120               // add food to output
121               lstResults.Items.Add( m_strFoods[ intRow ] + "\t"
122                  + m_intResults[ intRow, 0 ] + "\t\t"
123                  + m_intResults[ intRow, 1 ] );
124            }
125
126         } // end method btnAdd_Click
127
128      } // end class FrmFoodSurvey
129   }
```

18.12 ***(Sales Report Application)*** A clothing manufacturer has asked you to create an application that will calculate the total sales of that manufacturer in a week. Sales values should be input separately for each clothing item, but the amount of sales for all five weekdays should be input at once. The application should calculate the total amount of sales for each item in the week and also calculate the total sales for the manufacturer for all the items in the week. Because the manufacturer is a small company, it will produce at most ten items in any week. The application is shown in Fig. 18.19.

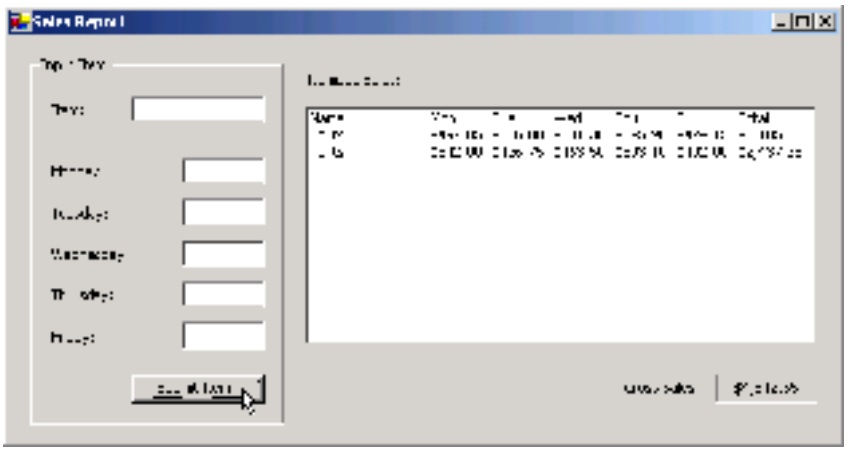


Figure 18.19 Sales Report application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial18\Exercises\SalesReport` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SalesReport.sln` in the `SalesReport` directory to open the application.

c) ***Declaring a two-dimensional `int` array.*** Declare a two-dimensional `decimal` array named `m_decItemSales`, with 10 rows and 5 columns.

d) ***Inputting data from the user.*** Add code to the beginning of the `btnSubmit_Click` event handler to input the data from the user. Assign the item name to the one-dimensional `m_intItemNames` array, indexed with `m_intItemCount` (which stores the number of items added). Assign the daily sales data to the two-dimensional `m_decItemSales` array. The first index to this array should be `m_intItemCount`, and the second will range from `0` to `4`. Finally, increment variable `m_intItemCount` to record that another item's sales data has been added.

e) ***Iterating over all the items added.*** Inside the `DisplaySales` method, after the `strOutput` variable has been declared, add code to begin a `for` statement. This `for` statement should iterate from `0` to `m_intItemCount - 1`. Declare the `intItem` variable as the `for` statement's counter. Insert code in this `for` statement to set `strOutput` to the item's name. Remember that the items' names are stored in `string` array `m_strItemNames`. Append two tab characters to format the output properly.

f) ***Iterating over the days in the week.*** Add code to initialize the `decWeekTotal` variable to `0`. This variable keeps track of the total sales for each item over the course of the week. Add code to start a `for` statement. This `for` statement will iterate from `0` to `4`, which is one less than the number of days in the work week. This is an example of a nested `for` statement, which is comparable to a nested `if` statement.

g) ***Appending the daily sales and summing sales for the week.*** Add code in this `for` statement to append the daily sales to `strOutput`. These sales are stored in the `m_decItemSales` array. This array must be accessed with the current item and the day of the week. The output is money, so use the `String.Format` method to format the value. Also append a tab character to format the output properly.

h) ***Calculating the weekly sales.*** Insert code to add the amount of the daily sales to the `decWeekTotal` variable. This variable stores the weekly sales for each item. Add a right brace to end the `for` statement started in *Step e.*

i) ***Calculating the total sales and outputting an item's sales.*** Insert code to add the weekly sales to the `decSalesTotal` variable. This variable keeps track of the total

sales for all the items for the week. Add code to append the weekly sales to `strOutput`. The weekly sales are also stored as money, so use the `String.Format` method again. Then add `strOutput` to the `ListBox` using the `Add` method of `ListBox` property `Items`. Insert a right brace to end the `for` statement started in *Step e*.

j) ***Running the application.*** Select **Debug > Start** to run your application. Test your application to ensure that it runs correctly as in Fig. 18.19.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 18.12 Solution
// SalesReport.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SalesReport
{
   /// <summary>
   /// Summary description for FrmSalesReport.
   /// </summary>
   public class FrmSalesReport : System.Windows.Forms.Form
   {
      // GroupBox to input and item's sales
      private System.Windows.Forms.GroupBox fraInput;

      // Label and TextBox to input item
      private System.Windows.Forms.Label lblItem;
      private System.Windows.Forms.TextBox txtItem;

      // Labels and TextBoxes to input daily sales
      private System.Windows.Forms.Label lblMonday;
      private System.Windows.Forms.TextBox txtMonday;
      private System.Windows.Forms.Label lblTuesday;
      private System.Windows.Forms.TextBox txtTuesday;
      private System.Windows.Forms.Label lblWednesday;
      private System.Windows.Forms.TextBox txtWednesday;
      private System.Windows.Forms.Label lblThursday;
      private System.Windows.Forms.TextBox txtThursday;
      private System.Windows.Forms.Label lblFriday;
      private System.Windows.Forms.TextBox txtFriday;

      // Button to submit an item's daily sales
      private System.Windows.Forms.Button btnSubmit;

      // Label and ListBox to display itemized sales
      private System.Windows.Forms.Label lblItemizedSales;
      private System.Windows.Forms.ListBox lstItemizedSales;

      // Labels to display total sales
      private System.Windows.Forms.Label lblGrossSales;
      private System.Windows.Forms.Label lblOutput;

      /// <summary>
      /// Required designer variable.
      /// </summary>
```

```
51          private System.ComponentModel.Container components = null;
52
53          // initialize number of items to zero
54          int m_intItemCount = 0;
55
56          // one-dimensional array to store names of items
57          string[] m_strItemNames = new string[ 10 ];
58
59          // two-dimensional array to store item sales
60          decimal[,] m_decItemSales = new decimal[ 10, 5 ];
61
62          public FrmSalesReport()
63          {
64             //
65             // Required for Windows Form Designer support
66             //
67             InitializeComponent();
68
69             //
70             // TODO: Add any constructor code after InitializeComponent
71             // call
72             //
73          }
74
75          /// <summary>
76          /// Clean up any resources being used.
77          /// </summary>
78          protected override void Dispose( bool disposing )
79          {
80             if( disposing )
81             {
82                if (components != null)
83                {
84                   components.Dispose();
85                }
86             }
87             base.Dispose( disposing );
88          }
89
90          // Windows Form Designer generated code
91
92          /// <summary>
93          /// The main entry point for the application.
94          /// </summary>
95          [STAThread]
96          static void Main()
97          {
98             Application.Run( new FrmSalesReport() );
99          }
100
101         // handles Submit Item Button's Click event
102         private void btnSubmit_Click(
103            object sender, System.EventArgs e )
104         {
105            // add name to array
106            m_strItemNames[ m_intItemCount ] = txtItem.Text;
107
108            // add gains to array
109            m_decItemSales[ m_intItemCount, 0 ] =
110               Decimal.Parse( txtMonday.Text );
```

```
111          m_decItemSales[ m_intItemCount, 1 ] =
112             Decimal.Parse( txtTuesday.Text );
113          m_decItemSales[ m_intItemCount, 2 ] =
114             Decimal.Parse( txtWednesday.Text );
115          m_decItemSales[ m_intItemCount, 3 ] =
116             Decimal.Parse( txtThursday.Text );
117          m_decItemSales[ m_intItemCount, 4 ] =
118             Decimal.Parse( txtFriday.Text );
119
120          m_intItemCount++; // increment the number of items
121
122          DisplaySales(); // display itemized sales
123
124          // clear TextBoxes for new data
125          txtItem.Text = "";
126          txtMonday.Text = "";
127          txtTuesday.Text = "";
128          txtWednesday.Text = "";
129          txtThursday.Text = "";
130          txtFriday.Text = "";
131
132          // if 10 or more items
133          if ( m_intItemCount == 10 )
134          {
135             // disable btnSubmit
136             btnSubmit.Enabled = false;
137          }
138
139       } // end method btnSubmit_Click
140
141       // display sales by type and day of week
142       void DisplaySales()
143       {
144          // clear the ListBox
145          lstItemizedSales.Items.Clear();
146
147          // add a header to the ListBox
148          lstItemizedSales.Items.Add( "Name\t\tMon.\t" +
149             "Tue.\tWed.\tThu.\tFri.\tTotal" );
150
151          decimal decWeekTotal = 0;  // initialize weekly total
152          decimal decSalesTotal = 0; // initialize total sales
153          string strOutput;          // displays sales in ListBox
154
155          // calculate items
156          for ( int intItem = 0; intItem < m_intItemCount; intItem++ )
157          {
158             // set the output string to the item name
159             strOutput = m_strItemNames[ intItem ] + "\t\t";
160
161             decWeekTotal = 0; // initialize weekly total
162
163             for ( int intDay = 0; intDay < 5; intDay++ )
164             {
165                // append the day's gain to the output string
166                strOutput += String.Format( "{0:C}",
167                   m_decItemSales[ intItem, intDay ] ) + "\t";
168
169                // add day's gain to weekly total
170                decWeekTotal += m_decItemSales[ intItem, intDay ];
```

```
171             }
172
173             // append weekly total to the output string
174             strOutput += String.Format( "{0:C}", decWeekTotal );
175
176             // add weekly total to sales total
177             decSalesTotal += decWeekTotal;
178
179             // add the output string to the ListBox
180             lstItemizedSales.Items.Add( strOutput );
181
182          } // end outer for loop
183
184          // output the total sales
185          lblOutput.Text = String.Format( "{0:C}", decSalesTotal );
186
187       } // end method DisplaySales
188
189    } // end class FrmSalesReport
190 }
```

18.13 ***(Profit Report Application)*** The clothing manufacturer was so impressed with the **Sales Report** application you created for them (Exercise 18.12) they want you to create a **Profit Report** application as well. This application will be similar to the **Sales Report** application, but it will allow the user to input information as gains or losses. It should provide `RadioButtons` to allow the user to select whether a certain item is a gain or a loss (Fig. 18.20).

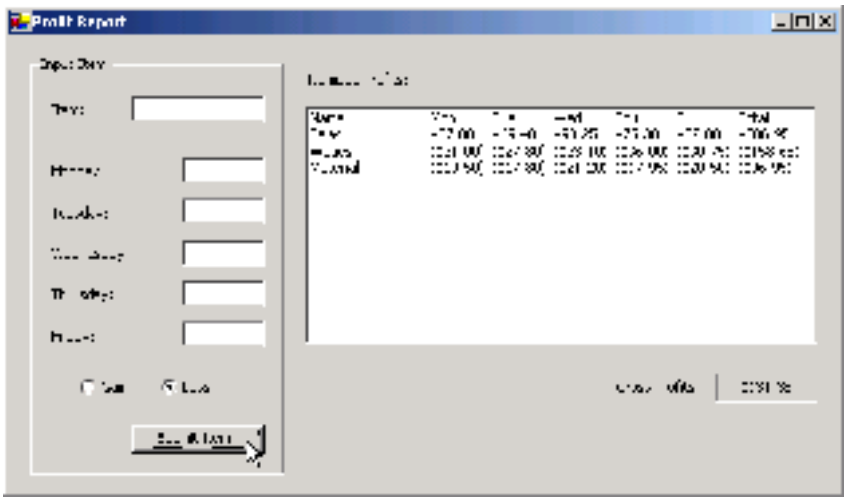


Figure 18.20 **Profit Report** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial18\Exercises\ProfitReport` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ProfitReport.sln` in the `ProfitReport` directory to open the application.

c) ***Modifying the template application.*** Modify the template as you did in Exercise 18.12.

d) ***Adding the Gain `RadioButton`.*** Add a `RadioButton` to the **Input Item** `GroupBox`. Set its `Name` property to `radGain` and its `Text` property to `Gain`. Set its `Size` and `Location` properties so that the control appears as in Fig. 18.20. Set the `RadioButton` to be selected when the application starts (the default).

e) ***Adding the Loss `RadioButton`.*** Add a second `RadioButton` to the **Input Item** `GroupBox`. Set its `Name` property to `radLoss` and its `Text` property to `Loss`. Set its `Size` and `Location` properties so that the control appears as in Fig. 18.20. Rearrange and comment the control declarations appropriately.

f) ***Testing which RadioButton was selected.*** Add code to the `btnSubmit_Click` event handler to test which `RadioButton` was selected. If the **Gain** `RadioButton` was selected, add the input values to the array normally. If the **Loss** `RadioButton` was selected, add the input value to the array as negative values.

g) ***Running the application.*** Select **Debug > Start** to run your application. Test your application to ensure that it runs correctly as in Fig. 18.20.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 18.13 Solution
// ProfitReport.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ProfitReport
{
   /// <summary>
   /// Summary description for FrmProfitReport.
   /// </summary>
   public class FrmProfitReport : System.Windows.Forms.Form
   {
      // GroupBox to input and item's profits
      private System.Windows.Forms.GroupBox fraInput;

      // Label and TextBox to input item
      private System.Windows.Forms.Label lblItem;
      private System.Windows.Forms.TextBox txtItem;

      // Labels and TextBoxes to input daily profits
      private System.Windows.Forms.Label lblMonday;
      private System.Windows.Forms.TextBox txtMonday;
      private System.Windows.Forms.Label lblTuesday;
      private System.Windows.Forms.TextBox txtTuesday;
      private System.Windows.Forms.Label lblWednesday;
      private System.Windows.Forms.TextBox txtWednesday;
      private System.Windows.Forms.Label lblThursday;
      private System.Windows.Forms.TextBox txtThursday;
      private System.Windows.Forms.Label lblFriday;
      private System.Windows.Forms.TextBox txtFriday;

      // RadioButtons to select gain or loss
      private System.Windows.Forms.RadioButton radGain;
      private System.Windows.Forms.RadioButton radLoss;

      // Button to submit an item's daily profits
      private System.Windows.Forms.Button btnSubmit;

      // Label and ListBox to display itemized profits
      private System.Windows.Forms.Label lblItemizedProfits;
      private System.Windows.Forms.ListBox lstItemizedProfits;

      // Labels to display total profits
      private System.Windows.Forms.Label lblGrossProfits;
      private System.Windows.Forms.Label lblOutput;
```

```
51
52          /// <summary>
53          /// Required designer variable.
54          /// </summary>
55          private System.ComponentModel.Container components = null;
56
57          // initialize number of items to zero
58          int m_intItemCount = 0;
59
60          // one-dimensional array to store names of items
61          string[] m_strItemNames = new string[ 10 ];
62
63          // two-dimensional array to store item profits
64          decimal[,] m_decItemProfits = new decimal[ 10, 5 ];
65
66          public FrmProfitReport()
67          {
68             //
69             // Required for Windows Form Designer support
70             //
71             InitializeComponent();
72
73             //
74             // TODO: Add any constructor code after InitializeComponent
75             // call
76             //
77          }
78
79          /// <summary>
80          /// Clean up any resources being used.
81          /// </summary>
82          protected override void Dispose( bool disposing )
83          {
84             if( disposing )
85             {
86                if (components != null)
87                {
88                   components.Dispose();
89                }
90             }
91             base.Dispose( disposing );
92          }
93
94          // Windows Form Designer generated code
95
96          /// <summary>
97          /// The main entry point for the application.
98          /// </summary>
99          [STAThread]
100         static void Main()
101         {
102            Application.Run( new FrmProfitReport() );
103         }
104
105         // handles Submit Item Button's Click event
106         private void btnSubmit_Click(
107            object sender, System.EventArgs e )
108         {
109            // add name to array
110            m_strItemNames[ m_intItemCount ] = txtItem.Text;
```

```
111
112          if ( radGain.Checked == true )
113          {
114             // add gains to array
115             m_decItemProfits[ m_intItemCount, 0 ] =
116                Decimal.Parse( txtMonday.Text );
117             m_decItemProfits[ m_intItemCount, 1 ] =
118                Decimal.Parse( txtTuesday.Text );
119             m_decItemProfits[ m_intItemCount, 2 ] =
120                Decimal.Parse( txtWednesday.Text );
121             m_decItemProfits[ m_intItemCount, 3 ] =
122                Decimal.Parse( txtThursday.Text );
123             m_decItemProfits[ m_intItemCount, 4 ] =
124                Decimal.Parse( txtFriday.Text );
125          }
126          else
127          {
128             // add losses to array
129             m_decItemProfits[ m_intItemCount, 0 ] =
130                -Decimal.Parse( txtMonday.Text );
131             m_decItemProfits[ m_intItemCount, 1 ] =
132                -Decimal.Parse( txtTuesday.Text );
133             m_decItemProfits[ m_intItemCount, 2 ] =
134                -Decimal.Parse( txtWednesday.Text );
135             m_decItemProfits[ m_intItemCount, 3 ] =
136                -Decimal.Parse( txtThursday.Text );
137             m_decItemProfits[ m_intItemCount, 4 ] =
138                -Decimal.Parse( txtFriday.Text );
139          }
140
141          m_intItemCount++; // increment the number of items
142
143          DisplayProfits(); // display itemized profits
144
145          // clear TextBoxes for new data
146          txtItem.Text = "";
147          txtMonday.Text = "";
148          txtTuesday.Text = "";
149          txtWednesday.Text = "";
150          txtThursday.Text = "";
151          txtFriday.Text = "";
152
153          // if 10 or more items
154          if ( m_intItemCount == 10 )
155          {
156             // disable btnSubmit
157             btnSubmit.Enabled = false;
158          }
159
160       } // end method btnSubmit_Click
161
162       // display profits by type and day of week
163       void DisplayProfits()
164       {
165          // clear the ListBox
166          lstItemizedProfits.Items.Clear();
167
168          // add a header to the ListBox
169          lstItemizedProfits.Items.Add( "Name\t\tMon.\t" +
170             "Tue.\tWed.\tThu.\tFri.\tTotal" );
```

```
171
172          decimal decWeekTotal = 0;  // initialize weekly total
173          decimal decProfitsTotal = 0; // initialize total profits
174          string strOutput;          // displays profits in ListBox
175
176          // calculate items
177          for ( int intItem = 0; intItem < m_intItemCount; intItem++ )
178          {
179             // set the output string to the item name
180             strOutput = m_strItemNames[ intItem ] + "\t\t";
181
182             decWeekTotal = 0; // initialize weekly total
183
184             for ( int intDay = 0; intDay < 5; intDay++ )
185             {
186                // append the day's gain to the output string
187                strOutput += String.Format( "{0:C}",
188                   m_decItemProfits[ intItem, intDay ] ) + "\t";
189
190                // add day's gain to weekly total
191                decWeekTotal += m_decItemProfits[ intItem, intDay ];
192             }
193
194             // add weekly total to profits total
195             decProfitsTotal += decWeekTotal;
196
197             // append weekly total to the output string
198             strOutput += String.Format( "{0:C}", decWeekTotal );
199
200             // add the output string to the ListBox
201             lstItemizedProfits.Items.Add( strOutput );
202
203          } // end outer for loop
204
205          // output the total profits
206          lblOutput.Text = String.Format( "{0:C}", decProfitsTotal );
207
208       } // end method DisplayProfits
209
210    } // end class FrmProfitReport
211 }
```

What does this code do?

18.14 What is returned by the following code? Assume that `GetStockPrices` is a method that returns a 2-by-31 array, with the first row containing the stock price at the beginning of the day and the last row containing the stock price at the end of the day, for each day of the month.

```
1  int[] Mystery()
2  {
3     int[,] intPrices = new int[ 2, 31 ];
4
5     intPrices = GetStockPrices();
6
7     int[] intResult = new int[ 31 ];
8
9     for ( int intI = 0; intI < intResult.Length; intI++ )
10    {
11       intResult[ intI ] = intPrices[ 1, intI ] -
```

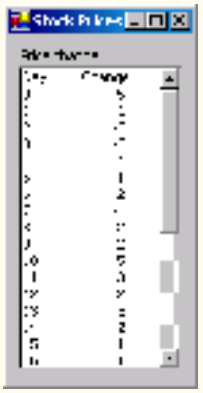
```
            intPrices[ 0, intI ];
      }

      return intResult;

   } // end method Mystery
```

Answer: The method returns a one-dimensional array containing the daily stock price change for each day of the month. The complete code reads:

```
// Exercise 18.14 Solution
// StockPrices.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace StockPrices
{
   /// <summary>
   /// Summary description for FrmStockPrices.
   /// </summary>
   public class FrmStockPrices : System.Windows.Forms.Form
   {
      private System.Windows.Forms.ListBox lstPriceChange;
      private System.Windows.Forms.Label lblChange;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmStockPrices()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
```

```
50             base.Dispose( disposing );
51          }
52
53          // Windows Form Designer generated code
54
55          /// <summary>
56          /// The main entry point for the application.
57          /// </summary>
58          [STAThread]
59          static void Main()
60          {
61             Application.Run( new FrmStockPrices() );
62          }
63
64          // displays the daily change in stock price over 1 month
65          private void FrmStockPrices_Load(
66             object sender, System.EventArgs e )
67          {
68             int[] intPrices;
69             int intLength;
70
71             intPrices = Mystery();
72
73             intLength = intPrices.Length;
74
75             // add a header to indicate the columns meaning
76             lstPriceChange.Items.Add( "Day\tChange" );
77
78             // print the array with proper formatting
79             for ( int intI = 0; intI < intLength; intI++ )
80             {
81                lstPriceChange.Items.Add( intI + "\t" +
82                   Convert.ToString( intPrices[ intI ] ).PadLeft( 10,
83                   Convert.ToChar( " " ) ) );
84             }
85
86          } // end method FrmStockPrices_Load
87
88          // calculates the daily change in stock price for one month
89          int[] Mystery()
90          {
91             int[,] intPrices = new int[ 2, 31 ];
92
93             intPrices = GetStockPrices();
94
95             int[] intResult = new int[ 31 ];
96
97             for ( int intI = 0; intI < intResult.Length; intI++ )
98             {
99                intResult[ intI ] = intPrices[ 1, intI ] -
100                  intPrices[ 0, intI ];
101            }
102
103            return intResult;
104
105         } // end method Mystery
106
107         // returns a 2D array of stock prices
108         int[,] GetStockPrices()
109         {
```

```
110          int[,] intStocks = {
111             { 13, 18, 17, 15, 10, 9, 10, 12, 11, 13, 15, 20, 23,
112               25, 20, 18, 19, 20, 14, 12, 10, 9, 10, 12, 13, 14,
113               15, 16, 13, 14, 20 },
114             { 18, 17, 15, 10, 9, 10, 12, 11, 13, 15, 20, 23, 25,
115               20, 18, 19, 20, 14, 12, 10, 9, 10, 12, 13, 14, 15,
116               16, 13, 14, 20, 22 }
117             };
118
119          return intStocks;
120
121       } // end method GetStockPrices
122
123    } // end class FrmStockPrices
124 }
125
```



What's wrong with this code?

18.15 Find the error(s) in the following code. The `TwoDArrays` method should create a two-dimensional array and initialize all its values to one.

```
1  int[,] TwoDArrays()
2  {
3     int[,] intArray = new int[ 4, 4 ];
4
5     // Assign 1 to all cell values
6     for ( int intI = 0; intI < 4; intI++ )
7     {
8        intArray[ intI, intI ] = 1;
9     }
10
11    return intArray;
12
13 } // end method TwoDArrays
```

Answer: To assign each element in a two-dimensional array, nested `for` loops should be used. The complete incorrect code reads:

```
1  // Exercise 18.15 Solution
2  // Initialize.cs (Incorrect)
3
4  using System;
```

```
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Initialize
{
   /// <summary>
   /// Summary description for FrmInitialize.
   /// </summary>
   public class FrmInitialize : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblArray;
      private System.Windows.Forms.ListBox lstArray;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmInitialize()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmInitialize() );
      }

      // calls TwoDArrays and prints the resulting array
```

```
65          private void FrmInitialize_Load(
66             object sender, System.EventArgs e )
67          {
68             int[,] intTempArray = TwoDArrays();
69             int intUpperBound = intTempArray.GetUpperBound( 0 );
70
71             // print each row of the array
72             for ( int intI = 0; intI <= intUpperBound; intI++ )
73             {
74                lstArray.Items.Add(intTempArray[ intI, 0 ] +
75                   "\t" + intTempArray[ intI, 1 ] +
76                   "\t" + intTempArray[ intI, 2 ] +
77                   "\t" + intTempArray[ intI, 3 ] );
78             }
79
80          } // end method FrmInitialize_Load
81
82          // initializes a 2D array to contain all 1's
83          int[,] TwoDArrays()
84          {
85             int[,] intArray = new int[ 4, 4 ];
86
87             // assign 1 to all cell values
88             for ( int intI = 0; intI < 4; intI++ )
89             {
90                intArray[ intI, intI ] = 1;
91             }
92
93             return intArray;
94
95          } // end method TwoDArrays
96
97       } // end class FrmInitialize
98    }
```

Needs nested `for` loops to access the entire array

Answer: The complete corrected code should read:

```
1  // Exercise 18.15 Solution
2  // Initialize.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Initialize
12 {
13    /// <summary>
14    /// Summary description for FrmInitialize.
15    /// </summary>
```

```
16    public class FrmInitialize : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblArray;
19       private System.Windows.Forms.ListBox lstArray;
20       /// <summary>
21       /// Required designer variable.
22       /// </summary>
23       private System.ComponentModel.Container components = null;
24
25       public FrmInitialize()
26       {
27          //
28          // Required for Windows Form Designer support
29          //
30          InitializeComponent();
31
32          //
33          // TODO: Add any constructor code after InitializeComponent
34          // call
35          //
36       }
37
38       /// <summary>
39       /// Clean up any resources being used.
40       /// </summary>
41       protected override void Dispose( bool disposing )
42       {
43          if( disposing )
44          {
45             if (components != null)
46             {
47                components.Dispose();
48             }
49          }
50          base.Dispose( disposing );
51       }
52
53       // Windows Form Designer generated code
54
55       /// <summary>
56       /// The main entry point for the application.
57       /// </summary>
58       [STAThread]
59       static void Main()
60       {
61          Application.Run( new FrmInitialize() );
62       }
63
64       // calls TwoDArrays and prints the resulting array
65       private void FrmInitialize_Load(
66           object sender, System.EventArgs e )
67       {
68          int[,] intTempArray = TwoDArrays();
69          int intUpperBound = intTempArray.GetUpperBound( 0 );
70
71          // print each row of the array
72          for ( int intI = 0; intI <= intUpperBound; intI++ )
73          {
74             lstArray.Items.Add( intTempArray[ intI, 0 ] +
75                "\t" + intTempArray[ intI, 1 ] +
```

```
                "\t" + intTempArray[ intI, 2 ] +
                "\t" + intTempArray[ intI, 3 ] );
            }

         } // end method FrmInitialize_Load

         // initializes a 2D array to contain all 1's
         int[,] TwoDArrays()
         {
            int[,] intArray = new int[ 4, 4 ];

            // assign 1 to all cell values
            for ( int intI = 0; intI < 4; intI++ )
            {
               for ( int intJ = 0; intJ < 4; intJ++ )
               {
                  intArray[ intI, intJ ] = 1;
               }
            }

            return intArray;

         } // end method TwoDArrays

      } // end class FrmInitialize
   }
```

Programming Challenge

18.16 ***(Enhanced Lottery Picker)*** A lottery commission offers four different lottery games to play: three-number, four-number, five-number and five-number + 1 lotteries. In Tutorial 16, your **Lottery Picker** application could select duplicate numbers for each lottery. In this exercise, you enhance the **Lottery Picker** to prevent duplicate numbers for the five-number and five-number + 1 lotteries (Fig. 18.21). According to this new requirement the games are now played as follows:

- Three-number lotteries require players to choose three numbers in the range 0–9.
- Four-number lotteries require players to choose four numbers, in the range 0–9.
- Five-number lotteries require players to choose five unique numbers in the range 1–39.
- Five-number + 1 lotteries require players to choose five unique numbers in the range 1–49 and an additional unique number in the range 1–42.

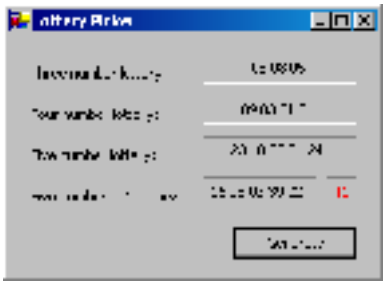


Figure 18.21 Enhanced **Lottery Picker** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial18\Exercises\EnhancedLotteryPicker` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `LotteryPicker.sln` in the `EnhancedLotteryPicker` directory to open the application.

c) ***Declaring a two-dimensional array to maintain unique random numbers.*** Declare the variable `m_blnNumbers` instance that stores a 2-by-50 `bool` array. You will use this array later in this exercise to test whether a lottery number has already been chosen.

d) ***Initializing the array.*** Each time the user clicks the **Generate** `Button`, the application should initialize the array by declaring its rows and setting the initial values. Write a `ClearArray` method that uses a `for` statement to assign each value in the `m_blnNumbers` array to `false`. Call the `ClearArray` method at the beginning of the **Generate** `Button`'s `Click` event handler.

e) ***Modifying the `Generate` method.*** You will modify the `Generate` method to use the `bool` array to pick unique random numbers. Begin by writing a statement that generates a random number and assigns its value to an `int` variable `intNumber`.

f) ***Determining whether the random number has already been selected.*** Use an `if` statement to determine whether the maximum lottery number is equal to `40`. (This happens when the upper limit on the random number equals `40`.) In this case, you will examine the first row of the array. To maintain unique numbers, you will set the value of the element in that row whose index equals the random number to `true` (indicating that it has been picked). For example, if the random number `34` has been picked, `m_blnNumbers[ 0, 34 ]` would contain the value `true`. To test whether a number has been picked, use a `while` statement inside the `if` statement to access that element of the array. If the array element's value is `true`, use the body of the loop to assign a new random number to `intNumber`. If the value in the array is `false`, use the condition in the `while` header to ignore the body of the loop. Just outside the `while`, include a statement that modifies the array to indicate that the number has now been picked.

g) ***Completing the application.*** Use a second `if` statement to determine whether the maximum lottery number is greater than `40`. In this case, you will examine the second row of the array. Repeat the process in the previous step. Remember to return the value stored in `intNumber` at the end of the `Generate` method.

h) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Generate** `Button` and check to make sure all numbers in both five number lotteries are unique. Do this several more times to make sure.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 18.16 Solution
2  // LotteryPicker.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
```

```
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace LotteryPicker
{
   /// <summary>
   /// Summary description for FrmLotteryPicker.
   /// </summary>
   public class FrmLotteryPicker : System.Windows.Forms.Form
   {
      // Labels to display the three-number lottery numbers
      private System.Windows.Forms.Label lblThree;
      private System.Windows.Forms.Label lblOutput3;

      // Labels to display the four-number lottery numbers
      private System.Windows.Forms.Label lblFour;
      private System.Windows.Forms.Label lblOutput4;

      // Labels to display the five-number lottery numbers
      private System.Windows.Forms.Label lblFive;
      private System.Windows.Forms.Label lblOutput5;

      // Labels to display the five-number-plus-1 lottery numbers
      private System.Windows.Forms.Label lblFivePlusOne;
      private System.Windows.Forms.Label lblOutput5Plus1;
      private System.Windows.Forms.Label lblOutputExtra1;

      // Button to generate a set of lottery numbers
      private System.Windows.Forms.Button btnGenerate;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      Random m_objRandom = new Random();

      // declare new array to store chosen lottery numbers
      bool[,] m_blnNumbers = new bool[ 2, 50 ];

      public FrmLotteryPicker()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
```

```
67          {
68             if (components != null)
69             {
70                components.Dispose();
71             }
72          }
73          base.Dispose( disposing );
74       }
75
76       // Windows Form Designer generated code
77
78       /// <summary>
79       /// The main entry point for the application.
80       /// </summary>
81       [STAThread]
82       static void Main()
83       {
84          Application.Run( new FrmLotteryPicker() );
85       }
86
87       // display random lottery numbers
88       private void btnGenerate_Click(
89          object sender, System.EventArgs e )
90       {
91          // generate random number with given boundaries
92          int intNumber = m_objRandom.Next( intLow, intHigh );
93
94          // generate three numbers
95          lblOutput3.Text = Generate( 0, 10 ) + " " +
96             Generate( 0, 10 ) + " " + Generate( 0, 10 );
97
98          // generate four numbers
99          lblOutput4.Text = Generate( 0, 10 ) + " " +
100            Generate( 0, 10 ) + " " + Generate( 0, 10 ) + " "
101            + Generate( 0, 10 );
102
103         // generate five numbers
104         lblOutput5.Text = Generate( 1, 40 ) + " " +
105            Generate( 1, 40 ) + " " + Generate( 1, 40 ) +
106            " " + Generate( 1, 40 ) + " " + Generate( 1, 40 );
107
108         // generate five plus one numbers
109         lblOutput5Plus1.Text = Generate( 1, 50 ) + " " +
110            Generate( 1, 50 ) + " " + Generate( 1, 50 ) + " " +
111            Generate( 1, 50 ) + " " + Generate( 1, 50 );
112
113         // generate extra number
114         lblOutputExtra1.Text = Generate( 1, 43 );
115
116      } // end method btnGenerate_Click
117
118      // generate random numbers
119      string Generate( int intLow, int intHigh )
120      {
121         // generate random number with given boundaries
122         int intNumber = m_objRandom.Next( intLow, intHigh );
123
124         // use first row for five-number lottery
125         if ( intHigh == 40 )
126         {
```

```
127             // select new random number
128             while ( m_blnNumbers[ 0, intNumber ] == true )
129             {
130                intNumber = m_objRandom.Next( intLow, intHigh );
131             }
132
133             // mark number as used
134             m_blnNumbers[ 0, intNumber ] = true;
135          }
136
137          // use second row for five-number + 1 lottery
138          if ( intHigh > 40 )
139          {
140             // select another random number that is not used
141             while ( m_blnNumbers[ 1, intNumber ] == true )
142             {
143                intNumber = m_objRandom.Next( intLow, intHigh );
144             }
145
146             // mark number as used
147             m_blnNumbers[ 1, intNumber ] = true;
148          }
149
150          return String.Format( "{0:D2}", intNumber );
151
152       } // end method Generate
153
154       // assigns all values in array to false
155       void ClearArray()
156       {
157          // loop through rows
158          for ( int intI = 0; intI < 2; intI++ )
159          {
160             // loop through columns
161             for ( int intJ = 0; intJ < 50; intJ++ )
162             {
163                m_blnNumbers[ intI, intJ ] = false;
164             }
165          }
166
167       } // end method ClearArray
168
169    } // end class FrmLotteryPicker
170 }
```

TUTORIAL 19

Microwave Oven Application

Building Your Own Classes and Objects

Solutions

Instructor's Manual Exercise Solutions Tutorial 19

MULTIPLE-CHOICE QUESTIONS

19.1 A `Button` appears flat if its _____ property is set to `Flat`.

a) `BorderStyle` b) `FlatStyle`
c) `Style` d) `BackStyle`

19.2 The _____ keyword introduces a class declaration.

a) `newclass` b) `classdef`
c) `csclass` d) `class`

19.3 The _____ operator is used to create an object.

a) `createobject` b) `instantiate`
c) `create` d) `new`

19.4 A `string` character is of the _____ type.

a) `char` b) `stringcharacter`
c) `character` d) `strcharacter`

19.5 The _____ is used to retrieve the value of an instance variable.

a) `get` accessor of a property b) `retrieve` method of a class
c) `client` method of a class d) `set` accessor of a property

19.6 When you add a new class to a project, the default namespace that contains the class in Visual Studio .NET is the name of the current _____.

a) file b) project
c) method d) variable

19.7 An important difference between constructors and other methods is that _____.

a) constructors cannot specify a return type
b) constructors cannot specify any parameters
c) other methods are implemented as `void` methods
d) constructors can assign values to instance variables

19.8 A class can yield many _____, just as a built-in type can yield many variables.

a) names b) objects
c) values d) types

19.9 The `set` accessor enables you to _____.

a) provide range checking b) modify data
c) provide data validation d) All of the above.

19.10 Instance variables declared `private` are not accessible _____.

a) outside the class b) by other methods of the same class
c) by other members of the same class d) inside the same class

Answers: 19.1) b. 19.2) d. 19.3) d. 19.4) a. 19.5) a. 19.6) b. 19.7) a. 19.8) b. 19.9) d. 19.10) a.

EXERCISES

19.11 (***Triangle Creator Application***) Create an application that allows the user to enter the lengths for the three sides of a triangle as `int`s. The application should then determine whether the triangle is a right triangle (two sides of the triangle form a 90-degree angle), an equilateral triangle (all sides of equal length) or neither. The application's GUI is completed for you (Fig. 19.50). You must create a class to represent a triangle object and define the event handler for the **Create** `Button`.

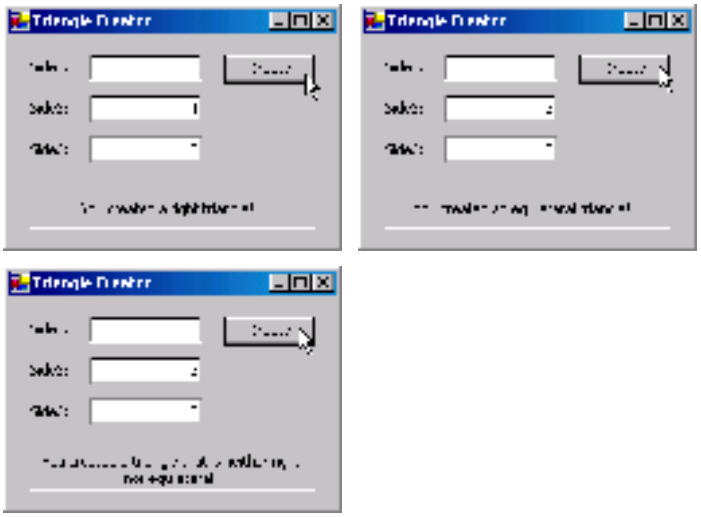


Figure 19.50 **Triangle Creator** application with all possible outputs.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial19\Exercises\Triangle` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Triangle.sln` in the `Triangle` directory to open the application.

c) ***Creating the `Triangle` class.*** Add a class to the project, and name it `Triangle`. This is where you will define the properties of the `Triangle` class.

d) ***Defining the necessary properties.*** Define a constructor that will take the lengths of the three sides of the triangle as arguments. Create three properties that enable clients to access and modify the lengths of the three sides. If the user enters a negative value, that side should be assigned the value zero.

e) ***Adding additional features.*** Create two more properties in the `Triangle` class: One determines whether the sides form a right triangle, the other an equilateral triangle. These properties are considered read-only, because you would naturally define only the `get` accessor. There is no simple `set` accessor that can make a triangle a right triangle or an equilateral triangle without first modifying the lengths of the triangle's sides. Therefore, simply omit the `set` accessor for this property.

f) ***Adding code to the event handler.*** Now that you have created your `Triangle` class, you can use it to create objects in your application. Double click the **Create** `Button` in **Design View** to generate the event handler. Create new variables to store the three lengths from the `TextBoxes`; then, use those values to create a new `Triangle` object.

g) ***Displaying the result.*** Use an `if...else` statement to determine if the triangle is a right triangle, an equilateral triangle or neither. Display the result in a `Label`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Create various inputs until you have created an equilateral triangle, a right triangle and a triangle that is neither right nor equilateral. Verify that the proper output is displayed for each.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer: [*Note*: Because the `Triangle` class is created from scratch, we have not highlighted any lines of code in `Triangle.cs`.]

```
1  // Exercise 19.11 Solution
2  // Triangle.cs
3  // Represent a triangle.
4
5  using System;
6
7  namespace Triangle
8  {
9     /// <summary>
```

```
10    /// Summary description for Triangle.
11    /// </summary>
12    public class Triangle
13    {
14       // declare three private side values
15       private int m_intSide1;
16       private int m_intSide2;
17       private int m_intSide3;
18
19       // Triangle constructor ( side1, side2 and side3 )
20       public Triangle(
21          int side1Value, int side2Value, int side3Value )
22       {
23          Side1 = side1Value;
24          Side2 = side2Value;
25          Side3 = side3Value;
26
27       } // end constructor Triangle
28
29       // get and set Side1
30       public int Side1
31       {
32          // return m_intSide1 value
33          get
34          {
35             return m_intSide1; // return length of side1
36
37          } // end of get accessor
38
39          // set side value
40          set
41          {
42             // make sure value is non-negative
43             if ( value > 0 )
44             {
45                m_intSide1 = value;
46             }
47             else
48             {
49                m_intSide1 = 0; // set to zero
50             }
51
52          } // end of set accessor
53
54       } // end property Side1
55
56       // get and set Side2
57       public int Side2
58       {
59          // return m_intSide2 value
60          get
61          {
62             return m_intSide2; // return length of side2
63
64          } // end of get accessor
65
66          // set side value
67          set
68          {
69             // make sure value is non-negative
```

```
70              if ( value > 0 )
71              {
72                 m_intSide2 = value;
73              }
74              else
75              {
76                 m_intSide2 = 0; // set to zero
77              }
78
79           } // end of set accessor
80
81        } // end property Side2
82
83        // get and set Side3
84        public int Side3
85        {
86           // return m_intSide3 value
87           get
88           {
89              return m_intSide3; // return length of side3
90
91           } // end of get accessor
92
93           // set side value
94           set
95           {
96              // make sure value is non-negative
97              if ( value > 0 )
98              {
99                 m_intSide3 = value;
100             }
101             else
102             {
103                m_intSide3 = 0; // set to zero
104             }
105
106          } // end of set accessor
107
108       } // end property Side3
109
110
111       // test if triangle is equilateral
112       public bool Equilateral
113       {
114          // check sides, return true or false
115          get
116          {
117             // test if sides are equal
118             if ( m_intSide1 == m_intSide2 &&
119                m_intSide1 == m_intSide3 )
120             {
121                return true; // indicate that sides are equal
122             }
123             else
124             {
125                return false; // indicate triangle is not equal sided
126             }
127
128          } // end of get accessor
129
```

```
130       } // end property Equilateral
131
132       // check if sides create right triangle
133       public bool RightTriangle
134       {
135          // check sides, return true or false
136          get
137          {
138             // check length
139             if ( m_intSide1 * m_intSide1 +
140                m_intSide2 * m_intSide2 ==
141                m_intSide3 * m_intSide3 )
142             {
143                return true; // it is a right triangle
144             }
145             // check another length combination
146             else if ( m_intSide1 * m_intSide1 +
147                m_intSide3 * m_intSide3 ==
148                m_intSide2 * m_intSide2 )
149             {
150                return true; // it is a right triangle
151             }
152             // check last length combination
153             else if ( m_intSide2 * m_intSide2 +
154                m_intSide3 * m_intSide3 ==
155                m_intSide1 * m_intSide1 )
156             {
157                return true; // it is a right triangle
158             }
159             else
160             {
161                return false; // it is not a right triangle
162             }
163
164          } // end of get accessor
165
166       } // end property RightTriangle
167
168    } // end class Triangle
169 }
```

```
1  // Exercise 19.11 Solution
2  // TriangleCreator.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Triangle
12 {
13    /// <summary>
14    /// Summary description for FrmTriangleCreator.
15    /// </summary>
16    public class FrmTriangleCreator : System.Windows.Forms.Form
17    {
18       // Labels and TextBoxes to input the side lengths
```

```
19          private System.Windows.Forms.Label lblSide1;
20          private System.Windows.Forms.TextBox txtSide1;
21          private System.Windows.Forms.Label lblSide2;
22          private System.Windows.Forms.TextBox txtSide2;
23          private System.Windows.Forms.Label lblSide3;
24          private System.Windows.Forms.TextBox txtSide3;
25
26          // Button to create a triangle
27          private System.Windows.Forms.Button btnCreate;
28
29          // Label to display triangle information
30          private System.Windows.Forms.Label lblDisplay;
31
32          /// <summary>
33          /// Required designer variable.
34          /// </summary>
35          private System.ComponentModel.Container components = null;
36
37          public FrmTriangleCreator()
38          {
39             //
40             // Required for Windows Form Designer support
41             //
42             InitializeComponent();
43
44             //
45             // TODO: Add any constructor code after InitializeComponent
46             // call
47             //
48          }
49
50          /// <summary>
51          /// Clean up any resources being used.
52          /// </summary>
53          protected override void Dispose( bool disposing )
54          {
55             if( disposing )
56             {
57                if (components != null)
58                {
59                   components.Dispose();
60                }
61             }
62             base.Dispose( disposing );
63          }
64
65          // Windows Form Designer generated code
66
67          /// <summary>
68          /// The main entry point for the application.
69          /// </summary>
70          [STAThread]
71          static void Main()
72          {
73             Application.Run( new FrmTriangleCreator() );
74          }
75
76          // create and test triangle properties
77          private void btnCreate_Click(
78             object sender, System.EventArgs e )
```

```
79          {
80             Triangle objTriangle; // create Triangle reference
81
82             // values for three sides
83             int intSide1 = Int32.Parse( txtSide1.Text );
84             int intSide2 = Int32.Parse( txtSide2.Text );
85             int intSide3 = Int32.Parse( txtSide3.Text );
86
87             lblDisplay.Text = ""; // clear display Label
88
89             // create triangle object
90             objTriangle = new Triangle( intSide1, intSide2, intSide3 );
91
92             // test for right triangle
93             if ( objTriangle.RightTriangle == true )
94             {
95               lblDisplay.Text = "You created a right triangle!";
96             }
97             // test for equilateral triangle
98             else if ( objTriangle.Equilateral == true )
99             {
100               lblDisplay.Text =
101                  "You created an equilateral triangle!";
102            }
103            // triangle is neither right nor equilateral
104            else
105            {
106               lblDisplay.Text = ( "You created a triangle that is " +
107                 "neither right nor equilateral!" );
108            }
109
110         } // end method btnCreate_Click
111
112      } // end class FrmTriangleCreator
113   }
```

19.12 (***Modified Microwave Oven Application***) Modify the tutorial's **Microwave Oven** application to include an additional digit, which would represent the hour. Allow the user to enter up to 9 hours, 59 minutes and 59 seconds (Fig. 19.51).

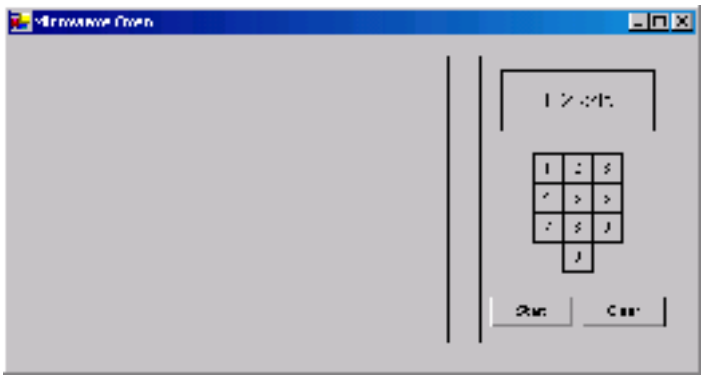


Figure 19.51 **Microwave Oven** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial19\Exercises\MicrowaveOven2` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `MicrowaveOven.sln` in the `MicrowaveOven2` directory to open the application.

c) ***Adding the hour variable.*** To allow the cooking time to include the hour digit, you will need to modify the `Time` class. Define a new `private` instance variable to represent the hour. Change the `Time` constructor to take as its first argument (now `Time` should have three arguments) the hour amount. You will also have to modify the **Start** `Button` event handler and the `DisplayTime` method to include an hour variable.

d) ***Adding the `Hour` property.*** Use the `Minute` and `Second` properties as your template to create the property for the hour. Remember, we are allowing an additional digit to represent the hour (`hour < 10`).

e) ***Changing the padding amount.*** Change the calls to the `PadLeft` method to be consistent with the new time format.

f) ***Extracting the hour.*** Add a call to the `Substring` method so that hour gets the first digit in the `m_strTime string`. Also, change the calls to the `Substring` method for minute and second so that they extract the proper digits from the `m_strTime string`.

g) ***Accessing the first five digits.*** Change the `if` statement from the `DisplayTime` method to take and display the first five digits entered by the user.

h) ***Edit the `Timer` object.*** Edit the `tmrClock_Tick` event handler to provide changes to hours and its corresponding minutes and seconds.

i) ***Displaying the time.*** Edit the format `string` so that the display `Label` includes the hour.

j) ***Running the application.*** Select **Debug > Start** to run your application. To ensure that your application works correctly, enter `1:00:10`, then click the **Start** `Button`. Watch as the microwave's timer counts down under one hour.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 19.12 Solution
// Time.cs
// Represents time data and contains properties.

using System;

namespace MicrowaveOven
{
   /// <summary>
   /// Summary description for Time.
   /// </summary>
   public class Time
   {
      // declare ints for hour, minute and second
      private int m_intHour;
      private int m_intMinute;
      private int m_intSecond;

      // Time constructor with hour, minute and second supplied
      public Time( int hourValue, int minuteValue, int secondValue )
      {
         Hour = hourValue;     // invokes Hour set accessor
         Minute = minuteValue; // invokes Minute set accessor
         Second = secondValue; // invokes Second set accessor

      } // end constructor Time

      // property Hour
      public int Hour
      {
         // return m_intHour value
         get
         {
```

```
34             return m_intHour;
35
36          } // end of get accessor
37
38          // set m_intHour value
39          set
40          {
41             // if minute value entered is valid
42             if ( value < 10 )
43             {
44                m_intHour = value;
45             }
46             else
47             {
48                m_intHour = 0; // set invalid input to 0
49             }
50
51          } // end of set accessor
52
53       } // end property Hour
54
55       // property Minute
56       public int Minute
57       {
58          // return m_intMinute value
59          get
60          {
61             return m_intMinute;
62
63          } // end of get accessor
64
65          // set m_intMinute value
66          set
67          {
68             // if minute value entered is valid
69             if ( value < 60 )
70             {
71                m_intMinute = value;
72             }
73             else
74             {
75                m_intMinute = 0; // set invalid input to 0
76             }
77
78          } // end of set accessor
79
80       } // end property Minute
81
82       // property Second
83       public int Second
84       {
85          // return m_intSecond value
86          get
87          {
88             return m_intSecond;
89
90          } // end of get accessor
91
92          // set m_intSecond value
93          set
```

```
94          {
95             // if minute value entered is valid
96             if ( value < 60 )
97             {
98                m_intSecond = value;
99             }
100            else
101            {
102               m_intSecond = 0; // set invalid input to 0
103            }
104
105         } // end of set accessor
106
107      } // end property Second
108
109   } // end class Time
110 }
```

```
1  // Exercise 19.12 Solution
2  // MicrowaveOven.cs (Modified)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace MicrowaveOven
12 {
13    /// <summary>
14    /// Summary description for FrmMicrowaveOven.
15    /// </summary>
16    public class FrmMicrowaveOven : System.Windows.Forms.Form
17    {
18       // Panel for the microwave's window
19       private System.Windows.Forms.Panel pnlWindow;
20
21       // Panel for the microwave's buttons
22       private System.Windows.Forms.Panel pnlControl;
23
24       // Label and Timer for the microwave's timer display
25       private System.Windows.Forms.Label lblDisplay;
26       private System.Windows.Forms.Timer tmrClock;
27
28       // Buttons of the microwave
29       private System.Windows.Forms.Button btnOne;
30       private System.Windows.Forms.Button btnTwo;
31       private System.Windows.Forms.Button btnThree;
32       private System.Windows.Forms.Button btnFour;
33       private System.Windows.Forms.Button btnFive;
34       private System.Windows.Forms.Button btnSix;
35       private System.Windows.Forms.Button btnSeven;
36       private System.Windows.Forms.Button btnEight;
37       private System.Windows.Forms.Button btnNine;
38       private System.Windows.Forms.Button btnZero;
39       private System.Windows.Forms.Button btnStart;
40       private System.Windows.Forms.Button btnClear;
41
```

```
42          /// <summary>
43          /// Required designer variable.
44          /// </summary>
45          private System.ComponentModel.IContainer components;
46
47          // contains time entered as a string
48          private string m_strTime = "";
49
50          // contains time entered
51          private Time m_objTime;
52
53          public FrmMicrowaveOven()
54          {
55             //
56             // Required for Windows Form Designer support
57             //
58             InitializeComponent();
59
60             //
61             // TODO: Add any constructor code after InitializeComponent
62             // call
63             //
64          }
65
66          /// <summary>
67          /// Clean up any resources being used.
68          /// </summary>
69          protected override void Dispose( bool disposing )
70          {
71             if( disposing )
72             {
73                if (components != null)
74                {
75                   components.Dispose();
76                }
77             }
78             base.Dispose( disposing );
79          }
80
81          // Windows Form Designer generated code
82
83          /// <summary>
84          /// The main entry point for the application.
85          /// </summary>
86          [STAThread]
87          static void Main()
88          {
89             Application.Run( new FrmMicrowaveOven() );
90          }
91
92          // event handler appends 1 to time string
93          private void btnOne_Click(
94             object sender, System.EventArgs e )
95          {
96             m_strTime += "1";   // append digit to time input
97             DisplayTime();      // display time input properly
98
99          } // end method btnOne_Click
100
101         // event handler appends 2 to time string
```

```
102         private void btnTwo_Click(
103            object sender, System.EventArgs e )
104         {
105            m_strTime += "2";   // append digit to time input
106            DisplayTime();      // display time input properly
107
108         } // end method btnTwo_Click
109
110         // event handler appends 3 to time string
111         private void btnThree_Click(
112            object sender, System.EventArgs e )
113         {
114            m_strTime += "3";   // append digit to time input
115            DisplayTime();      // display time input properly
116
117         } // end method btnThree_Click
118
119         // event handler appends 4 to time string
120         private void btnFour_Click(
121            object sender, System.EventArgs e )
122         {
123            m_strTime += "4";   // append digit to time input
124            DisplayTime();      // display time input properly
125
126         } // end method btnFour_Click
127
128         // event handler appends 5 to time string
129         private void btnFive_Click(
130            object sender, System.EventArgs e )
131         {
132            m_strTime += "5";   // append digit to time input
133            DisplayTime();      // display time input properly
134
135         } // end method btnFive_Click
136
137         // event handler appends 6 to time string
138         private void btnSix_Click(
139            object sender, System.EventArgs e )
140         {
141            m_strTime += "6";   // append digit to time input
142            DisplayTime();      // display time input properly
143
144         } // end method btnSix_Click
145
146         // event handler appends 7 to time string
147         private void btnSeven_Click(
148            object sender, System.EventArgs e )
149         {
150            m_strTime += "7";   // append digit to time input
151            DisplayTime();      // display time input properly
152
153         } // end method btnSeven_Click
154
155         // event handler appends 8 to time string
156         private void btnEight_Click(
157            object sender, System.EventArgs e )
158         {
159            m_strTime += "8";   // append digit to time input
160            DisplayTime();      // display time input properly
161
```

```
162         } // end method btnEight_Click
163
164         // event handler appends 9 to time string
165         private void btnNine_Click(
166            object sender, System.EventArgs e )
167         {
168            m_strTime += "9";   // append digit to time input
169            DisplayTime();      // display time input properly
170
171         } // end method btnNine_Click
172
173         // event handler appends 0 to time string
174         private void btnZero_Click(
175            object sender, System.EventArgs e )
176         {
177            m_strTime += "0";   // append digit to time input
178            DisplayTime();      // display time input properly
179
180         } // end method btnZero_Click
181
182         // event handler starts the microwave oven's cooking process
183         private void btnStart_Click(
184            object sender, System.EventArgs e )
185         {
186            int intSecond;
187            int intMinute;
188            int intHour;
189
190            // ensure that m_strTime has 5 characters
191            m_strTime = m_strTime.PadLeft( 5, Convert.ToChar( "0" ) );
192
193            // extract seconds, minutes, and hours
194            intSecond = Int32.Parse( m_strTime.Substring( 3 ) );
195            intMinute = Int32.Parse( m_strTime.Substring( 1, 2 ) );
196            intHour = Int32.Parse( m_strTime.Substring( 0, 1 ) );
197
198            // create Time object to contain time entered by user
199            m_objTime = new Time( intHour, intMinute, intSecond );
200
201            // display number of hours, minutes, and seconds
202            lblDisplay.Text = String.Format( "{0:D2}:{1:D2}:{2:D2}",
203               m_objTime.Hour, m_objTime.Minute, m_objTime.Second );
204
205            m_strTime = ""; // clear m_strTime for future input
206
207            tmrClock.Enabled = true; // start timer
208
209            pnlWindow.BackColor = Color.Yellow; // turn "light" on
210
211         } // end method btnStart_Click
212
213         // event handler to clear input
214         private void btnClear_Click(
215            object sender, System.EventArgs e )
216         {
217            // reset each property or variable to its initial setting
218            lblDisplay.Text = "Microwave Oven";
219            m_strTime = "";
220            m_objTime = new Time( 0, 0, 0 );
221            tmrClock.Enabled = false;
```

```
222          pnlWindow.BackColor = SystemColors.Control;
223
224       } // end method btnClear_Click
225
226       // method to display formatted time in timer window
227       private void DisplayTime()
228       {
229          int intSecond;
230          int intMinute;
231          int intHour;
232
233          string strDisplay; // string displays current input
234
235          // if too much input entered
236          if ( m_strTime.Length > 5 )
237          {
238             m_strTime = m_strTime.Substring( 0, 5 );
239          }
240
241          strDisplay = m_strTime.PadLeft( 5, Convert.ToChar( "0" ) );
242
243          // extract seconds, minutes, and hours
244          intSecond = Int32.Parse( strDisplay.Substring( 3 ) );
245          intMinute = Int32.Parse( strDisplay.Substring( 1, 2 ) );
246          intHour = Int32.Parse( strDisplay.Substring( 0, 1 ) );
247
248          // display number of hours, minutes, and seconds
249          lblDisplay.Text = String.Format( "{0:D2}:{1:D2}:{2:D2}",
250             intHour, intMinute, intSecond );
251
252       } // end method DisplayTime
253
254       // event handler displays new time each second
255       private void tmrClock_Tick(
256          object sender, System.EventArgs e )
257       {
258          // perform countdown, subtract one second
259          if  ( m_objTime.Second > 0 )
260          {
261             m_objTime.Second--;
262          }
263          else if ( m_objTime.Minute > 0 )
264          {
265             m_objTime.Minute--; // subtract one minute
266             m_objTime.Second = 59; // reset seconds for new minute
267          }
268          else if ( m_objTime.Hour > 0 )
269          {
270             m_objTime.Hour--; // subtract one hour
271             m_objTime.Minute = 59; // reset minutes for new hour
272             m_objTime.Second = 59; // reset seconds for new minute
273          }
274          else // no more seconds
275          {
276             tmrClock.Enabled = false; // stop timer
277             lblDisplay.Text = "Done!"; // tell user time is finished
278             pnlWindow.BackColor = SystemColors.Control;
279             return;
280          }
281
```

```
282          lblDisplay.Text = String.Format( "{0:D2}:{1:D2}:{2:D2}",
283             m_objTime.Hour, m_objTime.Minute, m_objTime.Second );
284
285       } // end method tmrClock_Tick
286
287    } // end class FrmMicrowaveOven
288 }
```

19.13 (*Account Information Application*) The local bank wants you to create an application that will allow them to view their clients' information. The interface is created for you; you need to implement the class (Fig. 19.52). Once the application is completed, the bank manager should be able to click the **Next** or **Previous** `Button` to run through each client's information. The information is stored in four arrays containing first names, last names, account numbers and account balances.

Figure 19.52 **Account Information** application GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial19\Exercises\AccountInformation` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AccountInformation.sln` in the `AccountInformation` directory to open the application.

c) ***Determining variables for the class.*** Examine the code from `AccountInformation.cs`, including all the properties that the `Client` object uses to retrieve the information.

d) ***Creating the `Client` class.*** Create a new class, called `Client`. Add this class to the project. Define four `private` instance variables to represent each property value, to ensure that each `Client` object contains all the required information about each client. Use those variables to define a constructor.

e) ***Defining each property.*** Each `private` variable should have a corresponding property, allowing the user to set or get each `private` variable's value.

f) ***Adding more information.*** In the `FrmAccountInformation_Load` event handler, add two more accounts. Include names, account numbers and balances for each corresponding array.

g) ***Running the application.*** Select **Debug > Start** to run your application. Ensure that your application works correctly by using the **Next** and **Previous** `Buttons` to scroll through the different accounts.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer: [*Note*: Because the `Client` class is created from scratch, we have not highlighted any lines of code in `Client.cs`. Also, the highlighted portions of `AccountInformation.cs` will vary from student to student.]

```
1  // Exercise 19.13 Solution
2  // Client.cs
3  // represent Client balance information
4  using System;
5
6  namespace AccountInformation
```

```
{
   /// <summary>
   /// Summary description for Client.
   /// </summary>
   public class Client
   {
      private string m_strFirstName; // first name
      private string m_strLastName; //last name
      private int m_intAccount; // account number
      private decimal m_intbalance; // account balance

      // Client constructor, first and last names,
      // account number and account balance supplied
      public Client(
         string strFirstName, string strLastName,
         int intAccount, decimal decBalance )
      {
         First = strFirstName;
         Last = strLastName;
         Account = intAccount;
         Balance = decBalance;

      } // end constructor Client

      // property First
      public string First
      {
         // return m_strFirstName
         get
         {
            return m_strFirstName; // return first name

         } // end get accessor

         // set first name
         set
         {
            m_strFirstName = value;

         } // end set accessor

      } // end property First

      // property Last
      public string Last
      {
         // return m_strLastName
         get
         {
            return m_strLastName; // return last name

         } // end get accessor

         // set last name
         set
         {
            m_strLastName = value;

         } // end set accessor

```

```
67         } // end property Last
68
69         // account number
70         public int Account
71         {
72            // return m_intAccount
73            get
74            {
75               return m_intAccount; // return account number
76
77            } // end get accessor
78
79            // set account number
80            set
81            {
82               // account number cannot be negative
83               if ( value > -1 )
84               {
85                  m_intAccount = value;
86               }
87               else
88               {
89                  m_intAccount = 0; // default to zero
90               }
91
92            } // end set accessor
93
94         } // end property Account
95
96         // account balance
97         public decimal Balance
98         {
99            // return m_intbalance
100           get
101           {
102              return m_intbalance; // return account balance
103
104           } // end get accessor
105
106           // set the account balance
107           set
108           {
109              m_intbalance = value;
110
111           } // end set accessor
112
113        } // end property Balance
114
115     } // end class Client
116  }
```

```
1  // Exercise 19.13 Solution
2  // AccountInformation.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
```

```
using System.Data;

namespace AccountInformation
{
   /// <summary>
   /// Summary description for FrmAccountInformation.
   /// </summary>
   public class FrmAccountInformation : System.Windows.Forms.Form
   {
      // Label and TextBox to display client's first name
      private System.Windows.Forms.Label lblFirst;
      private System.Windows.Forms.TextBox txtFirst;

      // Label and TextBox to display client's last name
      private System.Windows.Forms.Label lblLast;
      private System.Windows.Forms.TextBox txtLast;

      // Label and TextBox to display account number
      private System.Windows.Forms.Label lblAccountNumber;
      private System.Windows.Forms.TextBox txtAccount;

      // Label and TextBox to display account balance
      private System.Windows.Forms.Label lblBalance;
      private System.Windows.Forms.TextBox txtBalance;

      // Button to move to the previous client's information
      private System.Windows.Forms.Button btnPrevious;

      // Button to move to the next client's information
      private System.Windows.Forms.Button btnNext;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // Client object
      private Client[] m_objName = new Client[ 10 ];
      private int m_intPosition = 0; // current account

      public FrmAccountInformation()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
```

```
69             if (components != null)
70             {
71                components.Dispose();
72             }
73          }
74          base.Dispose( disposing );
75       }
76
77       // Windows Form Designer generated code
78
79       /// <summary>
80       /// The main entry point for the application.
81       /// </summary>
82       [STAThread]
83       static void Main()
84       {
85          Application.Run( new FrmAccountInformation() );
86       }
87
88       // create array of Client objects
89       private void FrmAccountInformation_Load(
90          object sender, System.EventArgs e )
91       {
92          int intCount; // counter variable
93
94          // array of first names
95          string[] strFirstName =
96             { "John", "Sarah", "Jack", "Adam", "Diane",
97             "David", "Kristin", "Jennifer", "Andrew", "Betsy" };
98
99          // array of last names
100         string[] strLastName =
101            { "Blue", "White", "Red", "Brown", "Yellow",
102            "Black", "Green", "Orange", "Purple", "Pink" };
103
104         // array of account numbers
105         int[] intAccount =
106            { 1234652, 1234666, 1234678, 1234681, 1234690,
107            1234770, 1234787, 1234887, 1234895, 1234989 };
108
109         // array of account balances
110         decimal[] decBalance =
111            { 1000.78M, 2056.24M, 978.65M, 990.0M, 432.75M,
112            780.78M, 4590.63M, 7910.11M, 230.1M, 903.2M };
113
114         // loop and create 10 Client objects
115         for ( intCount = 0; intCount <=
116            m_objName.GetUpperBound( 0 ); intCount++ )
117         {
118            // create new object and store into Client array
119            m_objName[ intCount ] = new Client(
120               strFirstName[ intCount ], strLastName[ intCount ],
121               intAccount[ intCount ], decBalance[ intCount ] );
122         }
123
124      } // end method FrmAccountInformation_Load
125
126      // display next object
127      private void btnNext_Click(
128         object sender, System.EventArgs e )
```

```
129       {
130          m_intPosition++; // increment position
131
132          // if position is last (top) object
133          if ( m_intPosition > m_objName.GetUpperBound( 0 ) )
134          {
135             m_intPosition = 0;    // set to first position in array
136             DisplayInformation(); // display information
137          }
138          else
139          {
140             DisplayInformation();
141          }
142
143       } // end method btnNext_Click
144
145       // display previous object
146       private void btnPrevious_Click(
147          object sender, System.EventArgs e )
148       {
149          m_intPosition--; // decrement position
150
151          // if position is last (bottom) object
152          if ( m_intPosition < 0 )
153          {
154             // set to last position in array
155             m_intPosition = m_objName.GetUpperBound( 0 );
156             DisplayInformation(); // display information
157          }
158          else
159          {
160             DisplayInformation();
161          }
162
163       } // end method btnPrevious_Click
164
165       // display information
166       private void DisplayInformation()
167       {
168          // use m_intPosition as index for each object
169          txtFirst.Text = m_objName[ m_intPosition ].First;
170          txtLast.Text = m_objName[ m_intPosition ].Last;
171          txtAccount.Text =
172             Convert.ToString( m_objName[ m_intPosition ].Account );
173
174          // format as currency
175          txtBalance.Text = String.Format( "{0:C}",
176             m_objName[ m_intPosition ].Balance );
177
178       } // end method DisplayInformation
179
180    } // end class FrmAccountInformation
181 }
```

What does this code do? ▶

19.14 What does the following code do? The first code listing contains the declaration of the `Shape` class. Each `Shape` object represents a closed shape with a number of sides. The second code listing contains a method (`Mystery`) created by a client of the `Shape` class. What does this method do?

```
1  public class Shape
2  {
3     private int m_intSides;
4
5     // constructor with number of sides
6     public Shape( int intSides )
7     {
8        Side = intSides;
9
10    } // end constructor Shape
11
12    // set and get side value
13    public int Side
14    {
15       // return m_intSides
16       get
17       {
18          return m_intSides;
19
20       } // end of get accessor
21
22       // set m_intSides
23       set
24       {
25          if ( value > 0 )
26          {
27             m_intSides = value;
28          }
29          else
30          {
31             m_intSides = 0;
32          }
33
34       } // end of set accessor
35
36    } // end property Side
37
38 } // end class Shape
```

```
1  public string Mystery( Shape objShape )
2  {
3     string strShape;
4
5     switch ( objShape.Side )
6     {
7        case 0:
8        case 1:
9        case 2: strShape = "Not a Shape";
10          break;
11
12       case 3: strShape = "Triangle";
13          break;
14
15       case 4: strShape = "Square";
16          break;
17
18       default: strShape = "Polygon";
19          break;
20    }
```

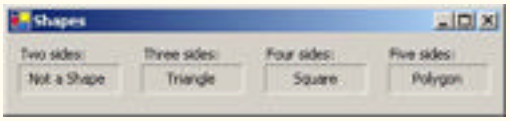
```
21
22       return strShape;
23
24    } // end method Mystery
```

Answer: The `Shape` class defines a shape with a given number of sides. Method `Mystery` determines the shape of its `Shape` and returns the name of the shape. Method `Mystery` takes a `Shape` object as an argument. The complete code reads:

```
1  // Exercise 19.14 Solution
2  // Shape.cs
3
4  using System;
5
6  namespace Shapes
7  {
8     /// <summary>
9     /// Summary description for Shape.
10    /// </summary>
11    public class Shape
12    {
13       private int m_intSides;
14
15       // constructor with number of sides
16       public Shape( int intSides )
17       {
18          Side = intSides;
19
20       } // end constructor Shape
21
22       // set and get side value
23       public int Side
24       {
25          // return m_intSides
26          get
27          {
28             return m_intSides;
29
30          } // end of get accessor
31
32          // set m_intSides
33          set
34          {
35             if ( value > 0 )
36             {
37                m_intSides = value;
38             }
39             else
40             {
41                m_intSides = 0;
42             }
43
44          } // end of set accessor
45
46       } // end property Side
47
48    } // end class Shape
49 }
```

```
// Exercise 19.14 Solution
// Shapes.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Shapes
{
   /// <summary>
   /// Summary description for FrmShapes.
   /// </summary>
   public class FrmShapes : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblFiveSidesOut;
      private System.Windows.Forms.Label lblFourSidesOut;
      private System.Windows.Forms.Label lblThreeSidesOut;
      private System.Windows.Forms.Label lblTwoSidesOut;
      private System.Windows.Forms.Label lblFiveSides;
      private System.Windows.Forms.Label lblFourSides;
      private System.Windows.Forms.Label lblThreeSides;
      private System.Windows.Forms.Label lblTwoSides;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmShapes()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

```

```
61          /// <summary>
62          /// The main entry point for the application.
63          /// </summary>
64          [STAThread]
65          static void Main()
66          {
67             Application.Run( new FrmShapes() );
68          }
69
70          // calls Mystery to and prints the results
71          private void FrmShapes_Load(
72              object sender, System.EventArgs e )
73          {
74             Shape objLine = new Shape( 2 );
75             Shape objTriangle = new Shape( 3 );
76             Shape objSquare = new Shape( 4 );
77             Shape objPoly = new Shape( 5 );
78
79             lblTwoSidesOut.Text = Mystery( objLine );
80             lblThreeSidesOut.Text = Mystery( objTriangle );
81             lblFourSidesOut.Text = Mystery( objSquare );
82             lblFiveSidesOut.Text = Mystery( objPoly );
83
84          } // end method FrmShapes_Load
85
86          // returns the name of a shape based on its number of sides
87          public string Mystery( Shape objShape )
88          {
89             string strShape;
90
91             switch ( objShape.Side )
92             {
93                case 0:
94                case 1:
95                case 2: strShape = "Not a Shape";
96                   break;
97
98                case 3: strShape = "Triangle";
99                   break;
100
101               case 4: strShape = "Square";
102                  break;
103
104               default: strShape = "Polygon";
105                  break;
106            }
107
108            return strShape;
109
110         } // end method Mystery
111
112      } // end class FrmShapes
113   }
```



What's wrong with this code?

19.15 Find the error(s) in the following code. The following method should create a new `Shape` object with `intNumberSides` sides. Assume the `Shape` class from Exercise 19.14 exists.

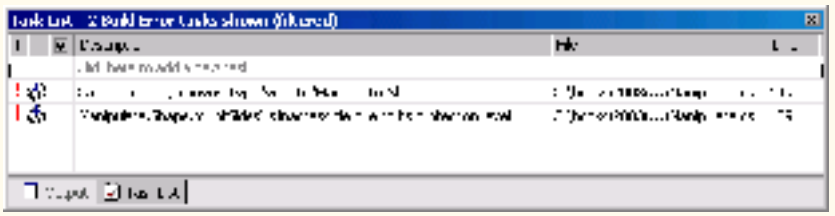
```
1  private void ManipulateShape( int intNumberSides )
2  {
3     Shape objShape = new Shape( 3 );
4     Shape.m_intSides = intNumberSides;
5
6  } // end method ManipulateShape
```

Answer: The method should create a `Shape` object with `intNumberSides` sides—not a `Shape` object with 3 sides. Also, a `private` variable (`m_intSides`) cannot be accessed from outside the class. The complete incorrect code reads:

```
1  // Exercise 19.15 Solution
2  // Manipulate.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using Shapes;
11
12 namespace Manipulate
13 {
14    /// <summary>
15    /// Summary description for FrmManipulate.
16    /// </summary>
17    public class FrmManipulate : System.Windows.Forms.Form
18    {
19       private System.Windows.Forms.Button btnChange;
20       private System.Windows.Forms.TextBox txtInput;
21       private System.Windows.Forms.Label lblChangeTo;
22       private System.Windows.Forms.Label lblSidesOut;
23       private System.Windows.Forms.Label lblSides;
24       /// <summary>
25       /// Required designer variable.
26       /// </summary>
27       private System.ComponentModel.Container components = null;
28
29       public FrmManipulate()
30       {
31          //
32          // Required for Windows Form Designer support
33          //
34          InitializeComponent();
35
36          //
37          // TODO: Add any constructor code after InitializeComponent
38          // call
39          //
40       }
41
42       /// <summary>
43       /// Clean up any resources being used.
44       /// </summary>
```

```
45          protected override void Dispose( bool disposing )
46          {
47             if( disposing )
48             {
49                if (components != null)
50                {
51                   components.Dispose();
52                }
53             }
54             base.Dispose( disposing );
55          }
56
57          // Windows Form Designer generated code
58
59          /// <summary>
60          /// The main entry point for the application.
61          /// </summary>
62          [STAThread]
63          static void Main()
64          {
65             Application.Run( new FrmManipulate() );
66          }
67
68          // updates the number of sides in the shape
69          private void btnChange_Click(
70             object sender, System.EventArgs e )
71          {
72             int intNewSides;
73             Shape objNewShape;
74
75             intNewSides = Int32.Parse( txtInput.Text );
76
77             objNewShape = ManipulateShape( intNewSides );
78
79             lblSidesOut.Text = Convert.ToString( objNewShape.Side );
80
81          } // end method btnChange_Click
82
83          // creates a new shape object with intNumberSides sides
84          private void ManipulateShape( int intNumberSides )
85          {
86             Shape objShape = new Shape( 3 );
87             Shape.m_intSides = intNumberSides;
88
89          } // end method ManipulateShape
90
91       } // end class FrmManipulate
92    }
```

Should use `intNumberSides`, `m_intSides` is declared `private`



Answer: The complete corrected code should read:

```
// Exercise 19.15 Solution
// Manipulate.cs (Correct)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using Shapes;

namespace Manipulate
{
   /// <summary>
   /// Summary description for FrmManipulate.
   /// </summary>
   public class FrmManipulate : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Button btnChange;
      private System.Windows.Forms.TextBox txtInput;
      private System.Windows.Forms.Label lblChangeTo;
      private System.Windows.Forms.Label lblSidesOut;
      private System.Windows.Forms.Label lblSides;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmManipulate()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
```

```
61          /// </summary>
62          [STAThread]
63          static void Main()
64          {
65             Application.Run( new FrmManipulate() );
66          }
67
68          // updates the number of sides in the shape
69          private void btnChange_Click(
70             object sender, System.EventArgs e )
71          {
72             int intNewSides;
73             Shape objNewShape;
74
75             intNewSides = Int32.Parse( txtInput.Text );
76
77             objNewShape = ManipulateShape( intNewSides );
78
79             lblSidesOut.Text = Convert.ToString( objNewShape.Side );
80
81          } // end method btnChange_Click
82
83          // creates a new shape object with intNumberSides sides
84          private Shape ManipulateShape( int intNumberSides )
85          {
86             Shape objShape = new Shape( intNumberSides );
87
88             return objShape;
89
90          } // end method ManipulateShape
91
92       } // end class FrmManipulate
93    }
```

Using the Debugger

19.16 (***View Name Application***) The **View Name** application allows the user to enter the user's first and last name. When the user clicks the **View Name** `Button`, a `MessageBox` that displays the user's first and last name appears. The application creates an instance of the `Name` class. This class uses its property declarations to set the first-name and last-name instance variables. Copy the directory `C:\Examples\Tutorial19\Exercises\Debugger\ViewName` to your `C:\SimplyCSP` directory. Open and run the application. While testing your application, you noticed that the `MessageBox` did not display the correct output. Use the debugger to find the logic error(s) in the application. The application with the correct output is displayed in Fig. 19.53.

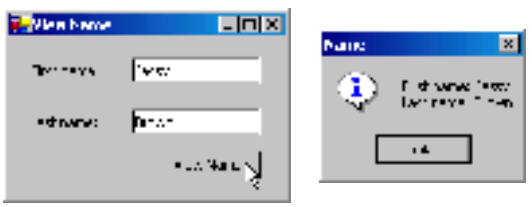


Figure 19.53 **View Name** application with correct output.

Answer:

```
1  // Exercise 19.16 Solution
2  // Name.cs
3  // represents a name
4
5  using System;
6
7  namespace ViewName
8  {
9     /// <summary>
10    /// Summary description for Name.
11    /// </summary>
12    public class Name
13    {
14       private string m_strFirstName;
15       private string m_strLastName;
16
17       // Name constructor, first and last names supplied
18       public Name( string strFirstName, string strLastName )
19       {
20          First = strFirstName;
21          Last = strLastName;
22
23       } // end constructor Name
24
25       // property First
26       public string First
27       {
28          // return first name
29          get
30          {
31             return m_strFirstName;
32
33          } // end get accessor
34
35          // set first name
36          set
37          {
38             m_strFirstName = value;
39
40          } // end set accessor
41
42       } // end property First
43
44       // property Last
45       public string Last
46       {
47          // return last name
48          get
49          {
50             return m_strLastName;
51
52          } // end get accessor
53
54          // set last name
55          set
56          {
57             m_strLastName = value;
58
59          } // end set accessor
```

Original line assigned `m_strFirstName` to `value`

Original line assigned data to `m_strFirstName`, rather than to `m_strLastName`

```
60
61          } // end property Last
62
63       } // end class Name
64    }
```

```
1  // Exercise 19.16 Solution
2  // ViewName.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ViewName
12 {
13    /// <summary>
14    /// Summary description for FrmViewName.
15    /// </summary>
16    public class FrmViewName : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input first name
19       private System.Windows.Forms.Label lblFirst;
20       private System.Windows.Forms.TextBox txtFirst;
21
22       // Label and TextBox to input last name
23       private System.Windows.Forms.Label lblLast;
24       private System.Windows.Forms.TextBox txtLast;
25
26       // Button to view name in MessageBox
27       private System.Windows.Forms.Button btnView;
28
29       /// <summary>
30       /// Required designer variable.
31       /// </summary>
32       private System.ComponentModel.Container components = null;
33
34       private Name m_objName; // Name object
35
36       public FrmViewName()
37       {
38          //
39          // Required for Windows Form Designer support
40          //
41          InitializeComponent();
42
43          //
44          // TODO: Add any constructor code after InitializeComponent
45          // call
46          //
47       }
48
49       /// <summary>
50       /// Clean up any resources being used.
51       /// </summary>
52       protected override void Dispose( bool disposing )
53       {
```

```
54          if( disposing )
55          {
56             if (components != null)
57             {
58                components.Dispose();
59             }
60          }
61          base.Dispose( disposing );
62       }
63
64       // Windows Form Designer generated code
65
66       /// <summary>
67       /// The main entry point for the application.
68       /// </summary>
69       [STAThread]
70       static void Main()
71       {
72          Application.Run( new FrmViewName() );
73       }
74
75       // handles View Name Button's Click event
76       private void btnView_Click(
77          object sender, System.EventArgs e )
78       {
79          string strOutput; // holds first name and last name
80
81          // create new Name
82          m_objName = new Name( txtFirst.Text, txtLast.Text );
83
84          // assign user's name to strOutput
85          strOutput = "First Name: " + m_objName.First +
86             "\r\nLast Name: " + m_objName.Last;
87
88          // output name to string
89          MessageBox.Show( strOutput, "Name",
90             MessageBoxButtons.OK, MessageBoxIcon.Information );
91
92       } // end method btnView_Click
93
94    } // end class FrmViewName
95 }
```

Programming Challenge

19.17 (***DVD Burner Application***) Create an application that simulates a DVD burner. Users create a DVD with their choice of title and bonus materials. The GUI is provided for you (Fig. 19.54). You will create a class (`DVDObject`) to represent the DVD object and another class (`Bonus`) to represent bonus materials for a DVD object.

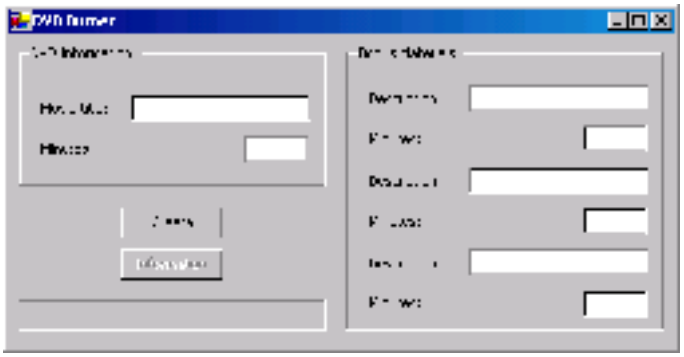


Figure 19.54 **DVD Burner** application's GUI.

a) ***Copying the template to your working directory.*** Copy the `C:\Examples\Tutorial19\Exercises\DVDBurner` directory to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `DVDBurner.sln` in the `DVDBurner` directory to open the application.

c) ***Creating the bonus material class.*** Create a class named `Bonus`. The class's objects will each represent one bonus-material item on the DVD. Each `Bonus` object should have a name (description) and a length (in minutes). Use this tutorial's `Time` class as your guide in creating the properties for the name and length of each bonus material.

d) ***Creating the DVD class.*** Create a class, and name it `DVDObject`. This class contains the movie title and the length of the movie. The class should also include an array of three `Bonus` items.

e) ***Creating the necessary variables.*** Before you define the **Create** `Button`'s event handler, create an `DVDObject` class instance variable. Inside the **Create** `Button`'s event handler, create the necessary variables to store the information from the `TextBoxes` on the GUI. Also, this is where you need to create the array of `Bonus` objects to store the bonus materials.

f) ***Adding bonus-material information.*** Add the description and length of each bonus item to the `Bonus` array you created from the previous step.

g) ***Creating a DVD object.*** Use information about the movie, its title, length and the array of bonus materials to make your DVD object.

h) ***Displaying the output.*** The **Information** `Button`'s `Click` event is already defined for you. Locate the event handler, add a `string` containing the complete information on the DVD object that you created earlier and display this `string` to a `MessageBox`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter information for several DVDs. After information is entered for each, click the **Create** `Button`. Then, click the **Information** `Button` and verify that the information being displayed is correct for your newly created DVD.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer: [*Note*: Because the `Bonus` and `DVDObject` classes are created from scratch, we have not highlighted any lines of code in `Bonus.cs` or `DVDObject.cs`.]

```
1  // Exercise 19.17 Solution
2  // Bonus.cs
3  // represent bonus items on a DVD
4
5  using System;
6
7  namespace DVDBurner
8  {
9     /// <summary>
10    /// Summary description for Bonus.
11    /// </summary>
12    public class Bonus
```

```
13    {
14       // name of bonus material
15       private string m_strName;
16
17       // length of the bonus material
18       private int m_intItemLength;
19
20       // Bonus constructor, name and item length
21       public Bonus( string nameValue, int lengthValue )
22       {
23          Name = nameValue;
24          ItemLength = lengthValue;
25
26       } // end constructor Bonus
27
28       // set or get name of bonus material
29       public string Name
30       {
31          // return m_strName
32          get
33          {
34             return m_strName; // return name
35
36          } // end get accessor
37
38          // set desciption name
39          set
40          {
41             // if description is greater than 20 characters
42             if ( value.Length > 20 )
43             {
44                // take first 20 characters
45                value = value.Substring( 0, 20 );
46                m_strName = value;
47             }
48             else
49             {
50                m_strName = value;
51             }
52
53          } // end set accessor
54
55       } // end property Name
56
57       // set or get amount of items
58       public int ItemLength
59       {
60          // return m_intItemLength
61          get
62          {
63             return m_intItemLength; // return length
64
65          } // end get accessor
66
67          // set minute value
68          set
69          {
70             // make sure minute value is non-negative
71             if ( value < 0 )
72             {
```

```
73                m_intItemLength = 0;
74             }
75             else
76             {
77                m_intItemLength = value;
78             }
79
80          } // end set accessor
81
82       } // end property ItemLength
83
84    } // end class Bonus
85 }
```

```
1  // Exercise 19.17 Solution
2  // DVDObject.cs
3  // represent items on a DVD
4
5  using System;
6
7  namespace DVDBurner
8  {
9     /// <summary>
10    /// Summary description for DVDObject.
11    /// </summary>
12    public class DVDObject
13    {
14       private string m_strMovieTitle; // name of movie
15       private Bonus[] m_objBonusMaterial; // array of Bonus objects
16       private int m_intMovieLength; // length of movie
17
18       // DVDObject constructor
19       public DVDObject(
20          string nameValue, Bonus[] bonusValue,
21          int movieLengthValue )
22       {
23          // call property to set values
24          MovieTitle = nameValue;
25          MovieLength = movieLengthValue;
26
27          // assign bonusValue array to m_objBonusMaterial
28          m_objBonusMaterial = bonusValue;
29
30       } // end constructor DVDObject
31
32       // set or get movie title
33       public string MovieTitle
34       {
35          // return m_strMovieTitle
36          get
37          {
38             return m_strMovieTitle;
39
40          } // end get accessor
41
42          // set movie title
43          set
44          {
45             // if title is greater than 20 characters
```

```
46             if ( value.Length > 20 )
47             {
48                // take first 20 characters
49                m_strMovieTitle = value.Substring( 0, 20 );
50             }
51             else
52             {
53                m_strMovieTitle = value;
54             }
55
56          } // end set accessor
57
58       } // end property MovieTitle
59
60       // get only property
61       public string BonusMaterials
62       {
63          // display bonus material information
64          get
65          {
66             // information on bonus materials
67             string strBonusMaterial = "";
68             int intCount; // counter variable
69
70             // loop through each bonus material
71             for ( intCount = 0; intCount < m_objBonusMaterial.Length;
72                intCount++ )
73             {
74                // format string to contain minutes and seconds
75                strBonusMaterial +=
76                   m_objBonusMaterial[ intCount ].Name + ": " +
77                   m_objBonusMaterial[ intCount ].ItemLength +
78                   " minute(s).\r\n";
79             }
80
81             return strBonusMaterial; // return string
82
83          } // end get accessor
84
85       } // end property BonusMaterials
86
87       // set or get movie length
88       public int MovieLength
89       {
90          // return m_intMovieLength
91          get
92          {
93             return m_intMovieLength; // return length
94
95          } // end get accessor
96
97          // set minute value
98          set
99          {
100            // make sure minute value is non-negative
101            if ( value > 0 )
102            {
103               m_intMovieLength = value;
104            }
105            else
```

```
106             {
107                m_intMovieLength = value;
108             }
109
110          } // end set accessor
111
112       } // end property MovieLength
113
114    } // end class DVDObject
115 }
```

```
1  // Exercise 19.17 Solution
2  // DVDBurner.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace DVDBurner
12 {
13    /// <summary>
14    /// Summary description for FrmDVDBurner.
15    /// </summary>
16    public class FrmDVDBurner : System.Windows.Forms.Form
17    {
18       // GroupBox containing DVD title and length
19       private System.Windows.Forms.GroupBox fraDVD;
20
21       // Label and TextBox to input DVD title
22       private System.Windows.Forms.Label lblMovieTitle;
23       private System.Windows.Forms.TextBox txtTitle;
24
25       // Label and TextBox to input length of movie in minutes
26       private System.Windows.Forms.Label lblMinutes;
27       private System.Windows.Forms.TextBox txtMovieMinute;
28
29       // GroupBox containing bonus materials information
30       private System.Windows.Forms.GroupBox fraBonus;
31
32       // Labels and TextBoxes to input first bonus material's
33       // description and length in minutes
34       private System.Windows.Forms.Label lblBonusDescription1;
35       private System.Windows.Forms.TextBox txtDescription1;
36       private System.Windows.Forms.Label lblBonusMinutes;
37       private System.Windows.Forms.TextBox txtMinutes1;
38
39       // Labels and TextBoxes to input second bonus material's
40       // description and length in minutes
41       private System.Windows.Forms.Label lblBonusDescription2;
42       private System.Windows.Forms.TextBox txtDescription2;
43       private System.Windows.Forms.Label lblBonusMinutes2;
44       private System.Windows.Forms.TextBox txtMinutes2;
45
46       // Labels and TextBoxes to input third bonus material's
47       // description and length in minutes
48       private System.Windows.Forms.Label lblBonusDescription3;
```

```
49          private System.Windows.Forms.TextBox txtDescription3;
50          private System.Windows.Forms.Label lblBonusMinutes3;
51          private System.Windows.Forms.TextBox txtMinutes3;
52
53          // Button to create a DVD
54          private System.Windows.Forms.Button btnCreate;
55
56          // Button to display DVD information
57          private System.Windows.Forms.Button btnInformation;
58
59          // Label to display information to the user
60          private System.Windows.Forms.Label lblDisplay;
61
62          /// <summary>
63          /// Required designer variable.
64          /// </summary>
65          private System.ComponentModel.Container components = null;
66
67          private DVDObject m_objDVD; // create instance variable
68
69          public FrmDVDBurner()
70          {
71             //
72             // Required for Windows Form Designer support
73             //
74             InitializeComponent();
75
76             //
77             // TODO: Add any constructor code after InitializeComponent
78             // call
79             //
80          }
81
82          /// <summary>
83          /// Clean up any resources being used.
84          /// </summary>
85          protected override void Dispose( bool disposing )
86          {
87             if( disposing )
88             {
89                if (components != null)
90                {
91                   components.Dispose();
92                }
93             }
94             base.Dispose( disposing );
95          }
96
97          // Windows Form Designer generated code
98
99          /// <summary>
100         /// The main entry point for the application.
101         /// </summary>
102         [STAThread]
103         static void Main()
104         {
105            Application.Run( new FrmDVDBurner() );
106         }
107
108         // display information about DVD
```

```
109        private void btnInformation_Click(
110           object sender, System.EventArgs e )
111        {
112           string strInformation = ""; // output string
113
114           lblDisplay.Text = ""; // clear Label
115
116           // add title and length to string
117           // add information about bonus materials
118           strInformation = m_objDVD.MovieTitle + ": " +
119              m_objDVD.MovieLength + " minute(s)\r\n" +
120              "Bonus Materials:\r\n" + m_objDVD.BonusMaterials;
121
122           // display output in a MessageBox
123           MessageBox.Show( strInformation, "DVD Description",
124              MessageBoxButtons.OK, MessageBoxIcon.Information );
125
126        } // end method btnInformation_Click
127
128        // create a DVD
129        private void btnCreate_Click(
130           object sender, System.EventArgs e )
131        {
132           // array of Bonus objects
133           Bonus[] objBonus = new Bonus[ 3 ];
134           int intBonusLength; //  bonus material minutes
135
136           // store movie name
137           string strMovieTitle = txtTitle.Text;
138
139           // movie minutes
140           int intMovieMinutes =
141              Int32.Parse( txtMovieMinute.Text );
142
143           // bonus material description (name)
144           string strBonus1 = txtDescription1.Text;
145           string strBonus2 = txtDescription2.Text;
146           string strBonus3 = txtDescription3.Text;
147
148           // store minutes from TextBox
149           intBonusLength = Int32.Parse( txtMinutes1.Text );
150
151           // add bonus material name and time to array
152           objBonus[ 0 ] = new Bonus( strBonus1, intBonusLength );
153
154           // store minutes from TextBox
155           intBonusLength = Int32.Parse( txtMinutes2.Text );
156
157           // add bonus material name and time to array
158           objBonus[ 1 ] = new Bonus( strBonus2, intBonusLength );
159
160           // store minutes from TextBox
161           intBonusLength = Int32.Parse( txtMinutes3.Text );
162
163           // add bonus material name and time to array
164           objBonus[ 2 ] = new Bonus( strBonus3, intBonusLength );
165
166           // call constructor for new object
167           m_objDVD = new DVDObject(
168              strMovieTitle, objBonus, intMovieMinutes );
```

```
169
170          // let the user know about progress
171          lblDisplay.Text = "Your DVD was created successfully!";
172
173          // enable Information Button
174          btnInformation.Enabled = true;
175
176       } // end method btnCreate_Click
177
178    } // end class FrmDVDBurner
179 }
```

20 TUTORIAL

Shipping Hub Application

Introducing Collections, the `foreach` Statement and Access Keys

Solutions

Instructor's Manual Exercise Solutions Tutorial 20

MULTIPLE-CHOICE QUESTIONS

20.1 _____ are specifically designed to store groups of values.

a) Collections
b) Properties
c) Accessors
d) None of the above.

20.2 _____ provides a quick and convenient way to navigate through controls on a `Form`.

a) *Tab*
b) *Enter*
c) *Caps Lock*
d) *Alt*

20.3 An `ArrayList` differs from an array in that an `ArrayList` can _____.

a) store objects of any type
b) resize dynamically
c) be accessed programmatically
d) All of the above.

20.4 The element in a `foreach` statement _____.

a) must be of type `int`
b) must be of (or convertible to) the same type as the collection or array type
c) must be of type `ArrayList`
d) None of the above.

20.5 The control that receives the focus the first time *Tab* is pressed has a `TabIndex` property set to _____.

a) `First`
b) `0`
c) `Next`
d) `1`

20.6 Users should be able to use *Tab* to transfer the focus to _____.

a) only `Buttons`
b) only `TextBoxes`
c) only controls that have an `AcceptTab` property
d) only the controls that receive user input

20.7 To ensure that the proper controls obtain the focus when *Tab* is pressed, use the _____.

a) `TabIndex` property
b) `TabStop` and `TabIndex` properties
c) `TabStop` property
d) `Focus` property

20.8 To add a value to the end of an `ArrayList`, call the _____ method.

a) `Add`
b) `AddToEnd`
c) `AddAt`
d) `InsertAt`

20.9 To remove a value from a specific index in the `ArrayList`, use method _____.

a) `Remove`
b) `RemoveAt`
c) `Delete`
d) `DeleteAt`

20.10 To display an ampersand character on a control, type _____ in its `Text` property.

a) `&_`
b) `&`
c) `&&`
d) `_&`

Answers: 20.1) a. 20.2) a. 20.3) b. 20.4) b. 20.5) b. 20.6) d. 20.7) b. 20.8) a. 20.9) b. 20.10) c.

EXERCISES

20.11 (***Modified Salary Survey Application***) Modify the **Salary Survey** application you created in Exercise 17.12 by using a `foreach` loop to replace the `for` loop that is used in Tutorial 17 (Fig. 20.26).

Figure 20.26 Modified **Salary Survey** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial20\Exercises\SalarySurveyModified` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SalarySurvey.sln` in the `SalarySurveyModified` directory to open the application.

c) ***Locating the event handler.*** Double click the **Show Totals** `Button` to bring up the event handler. The code to handle the `Click` event should include two statements, one to clear the items in the `ListBox` and the other to add a header.

d) ***Creating a counter variable.*** The `foreach` loop allows you to loop through each element in a specified collection. The `for` loop from Exercise 17.12 handles the `string` (`m_strSalaryRanges`) and `int` (`m_intSalaries`) arrays. This presents a problem. You cannot loop through both of these arrays using the same element reference. (One is an `int`, and the other is a `string`.) To handle this you need to create a common counter variable, which you will use to loop through the indices of both arrays. This is possible because the lengths of both arrays are the same.

e) ***Adding an element reference.*** It does not matter which array you decide to use in this exercise, because these arrays are of the same length. Declare an element reference with the correct type.

f) ***Create the `foreach` loop.*** Use the new element reference that you have created along with the array of your choice to create the `foreach` loop statement.

g) ***Adding text to the `ListBox`.*** Add the statement to output to the `ListBox`—exactly the same as the one from Exercise 17.12. The only difference will be the name of the counter variable that you decide to use.

h) ***Increment the counter variable.*** To successfully loop through both arrays and output the data, you need to increment the counter variable. This ensures that the proper data is added to the `ListBox` through each iteration.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter several sales amounts using the **Calculate** `Button`. Click the **Show Totals** `Button` and verify that the proper amounts are displayed for each salary range, based on the salaries calculate from your input.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 20.11 Solutions
2  // SalarySurvey.cs (Modified)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
```

```
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SalarySurvey
{
   /// <summary>
   /// Summary description for FrmSalarySurvey.
   /// </summary>
   public class FrmSalarySurvey : System.Windows.Forms.Form
   {
      // Label and TextBox to input sales for an employee
      private System.Windows.Forms.Label lblInputSales;
      private System.Windows.Forms.TextBox txtInputSales;

      // Button to calculate an employee's salary
      private System.Windows.Forms.Button btnCalculate;

      // Labels to display an employee's salary
      private System.Windows.Forms.Label lblTotalSalaryLabel;
      private System.Windows.Forms.Label lblTotalSalary;

      // Button to display number of employees who have
      // salaries in various ranges
      private System.Windows.Forms.Button btnShowTotals;

      // Label and ListBox to display number of employees who have
      // salaries in various ranges
      private System.Windows.Forms.Label lblSurveyResults;
      private System.Windows.Forms.ListBox lstSalaryTotals;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // salary ranges
      string[] m_strSalaryRanges = new string[] {
         "$200 - $299", "$300 - $399", "$400 - $499",
         "$500 - $599", "$600 - $699", "$700 - $799",
         "$800 - $899", "$900 - $999", "$1000 +  " };

      // number of employees in each salary range
      int[] m_intSalaries;

      public FrmSalarySurvey()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
```

```
67          /// </summary>
68          protected override void Dispose( bool disposing )
69          {
70             if( disposing )
71             {
72                if (components != null)
73                {
74                   components.Dispose();
75                }
76             }
77             base.Dispose( disposing );
78          }
79
80          // Windows Form Designer generated code
81
82          /// <summary>
83          /// The main entry point for the application.
84          /// </summary>
85          [STAThread]
86          static void Main()
87          {
88             Application.Run( new FrmSalarySurvey() );
89          }
90
91          // handles Form Load event
92          private void FrmSalarySurvey_Load(
93             object sender, System.EventArgs e )
94          {
95             m_intSalaries = new int[
96                m_strSalaryRanges.GetUpperBound( 0 ) + 1 ];
97
98          } // end method FrmSalarySurvey_Load
99
100         // handles Calculate Button's Click event
101         private void btnCalculate_Click(
102            object sender, System.EventArgs e )
103         {
104            // obtain total sales
105            decimal decSales = Decimal.Parse( txtInputSales.Text );
106
107            // check for negative input
108            if ( decSales < 0M )
109            {
110               MessageBox.Show( "Please enter a positive sales value.",
111                  "Invalid Input", MessageBoxButtons.OK,
112                  MessageBoxIcon.Exclamation );
113            }
114            else
115            {
116               // employee's base salary
117               decimal decTotalSalary = 200M;
118
119               // add commision to total salary
120               decTotalSalary += decSales * 0.09M;
121
122               // display salary in a Label
123               lblTotalSalary.Text =
124                  String.Format( "{0:C}", decTotalSalary );
125
126               // increment the correct counter in array intSalaries
```

```
127             if ( decTotalSalary < 300M )
128             {
129                m_intSalaries[ 0 ]++;
130             }
131             else if ( decTotalSalary < 400M )
132             {
133                m_intSalaries[ 1 ]++;
134             }
135             else if ( decTotalSalary < 500M )
136             {
137                m_intSalaries[ 2 ]++;
138             }
139             else if ( decTotalSalary < 600M )
140             {
141                m_intSalaries[ 3 ]++;
142             }
143             else if ( decTotalSalary < 700M )
144             {
145                m_intSalaries[ 4 ]++;
146             }
147             else if ( decTotalSalary < 800M )
148             {
149                m_intSalaries[ 5 ]++;
150             }
151             else if ( decTotalSalary < 900M )
152             {
153                m_intSalaries[ 6 ]++;
154             }
155             else if ( decTotalSalary < 1000M )
156             {
157                m_intSalaries[ 7 ]++;
158             }
159             else if ( decTotalSalary >= 1000M )
160             {
161                m_intSalaries[ 8 ]++;
162             }
163
164             txtInputSales.Clear(); // clear TextBox
165
166          } // end else
167
168       } // end method btnCalculate_Click
169
170       // handles Show Total Button's Click event
171       private void btnShowTotals_Click(
172          object sender, System.EventArgs e )
173       {
174          int intCounter = 0; // counter variable
175
176          // clear all items in the ListBox
177          lstSalaryTotals.Items.Clear();
178
179          // add header to ListBox
180          lstSalaryTotals.Items.Add( "Salary range:Total" );
181
182          // loop through each range in m_strSalaryRanges and
183          // add each element from two arrays to the ListBox
184          foreach ( string strRange in m_strSalaryRanges )
185          {
186             lstSalaryTotals.Items.Add(
```

```
            m_strSalaryRanges[ intCounter ] + "\t" +
            m_intSalaries[ intCounter ] );
         intCounter++; // increment the counter
      }

   } // end method btnShowTotals_Click

} // end class FrmSalarySurvey
}
```

20.12 (*Modified Shipping Hub Application*) Modify the **Shipping Hub** application created in this tutorial, so that the user can double click a package in the `lstPackages ListBox`. When a package number is double clicked, the package's information should be displayed in a `MessageBox` (Fig. 20.27).

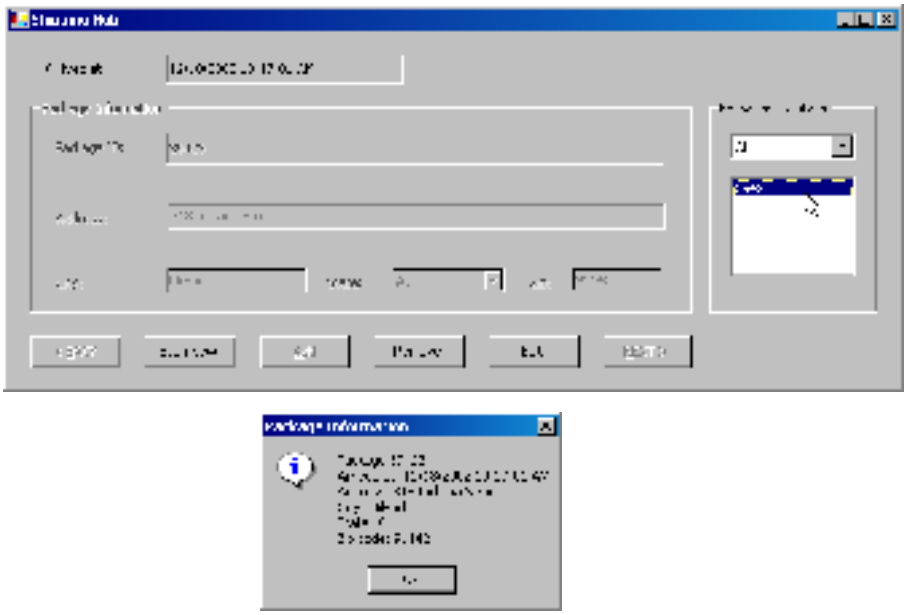


Figure 20.27 Modified **Shipping Hub** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial20\Exercises\ShippingHubModified` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ShippingHub.sln` in the `ShippingHubModified` directory to open the application.

c) ***Viewing the event handler.*** Click **ShippingHub.cs** in the **Solution Explorer** and select **View > Code**. Scroll to the end of code listing to locate the `ListBox`'s `DoubleClick` event handler.

d) ***Initializing necessary variables.*** To loop through the packages in the `ArrayList` of `Packages`, you need to create a reference of the `Package` type. It is also helpful to create a `string` variable to store the information about the given package. Write code in the `DoubleClick` event handler to declare the `strPackage string`. A `ListBox`'s `DoubleClick` event is raised when the control is double clicked.

e) ***Check whether the user has selected a valid item.*** To determine whether the user has selected a valid item (and not an empty element in the `ListBox`), write an `if` statement to make sure that the `ListBox` is not empty when the user selected an item. [*Hint*: A `SelectedIndex` value of `-1` means that no item is currently selected.]

f) ***Writing a `foreach` loop.*** Use the `Package` reference you declared in *Step d* to create a `foreach` loop with the `m_objList` collection.

g) ***Determining whether the current selected package is correct.*** Insert an `if` statement to determine whether the current object that is selected from the `m_objList` collec-

tion matches the selected item from the ListBox. Because the packages are listed in the ListBox by their package number, use that information in your if statement. Once the correct package is matched, store that package's information in the strPackage string.

h) ***Inserting the else block***. Make sure to notify the user if an invalid item has been selected from the ListBox. If this occurs, add a message to the strPackage string that will be displayed in the MessageBox.

i) ***Displaying the MessageBox***. Call the MessageBox's Show method to display the text you have added to the strPackage string. This displays either the information for the package they have selected or the message telling them they have selected an invalid package.

j) ***Running the application.*** Select **Debug > Start** to run your application. Add several packages. In the **Packages by State** GroupBox, select a state for which there are packages being sent. Double click one of the packages listed in the **Packages by State** ListBox, and verify that the correct information is displayed in a MessageBox.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 20.12 Solution
// ShippingHub.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ShippingHub
{
   /// <summary>
   /// Summary description for FrmShippingHub.
   /// </summary>
   public class FrmShippingHub : System.Windows.Forms.Form
   {
      // Labels for the arrival time
      private System.Windows.Forms.Label lblArrived;
      private System.Windows.Forms.Label lblArrivalTime;

      // GroupBox for the package information
      private System.Windows.Forms.GroupBox fraAddress;

      // Labels for the package ID
      private System.Windows.Forms.Label lblPackageID;
      private System.Windows.Forms.Label lblPackageNumber;

      // Labels, TextBoxes and ComboBox for the full address
      private System.Windows.Forms.Label lblAddress;
      private System.Windows.Forms.TextBox txtAddress;
      private System.Windows.Forms.Label lblCity;
      private System.Windows.Forms.TextBox txtCity;
      private System.Windows.Forms.Label lblState;
      private System.Windows.Forms.ComboBox cboState;
      private System.Windows.Forms.Label lblZip;
      private System.Windows.Forms.TextBox txtZip;

      // GroupBox for choosing packages by state.
      // contains a ComboBox and a ListBox for doing so.
      private System.Windows.Forms.GroupBox fraListByState;
```

```
42          private System.Windows.Forms.ComboBox cboViewPackages;
43          private System.Windows.Forms.ListBox lstPackages;
44
45          // Buttons to go back one package, scan a new package,
46          // add a package, remove a package, edit an existing package
47          // and go forward one package.
48          private System.Windows.Forms.Button btnBack;
49          private System.Windows.Forms.Button btnNew;
50          private System.Windows.Forms.Button btnAdd;
51          private System.Windows.Forms.Button btnRemove;
52          private System.Windows.Forms.Button btnEditUpdate;
53          private System.Windows.Forms.Button btnNext;
54
55          /// <summary>
56          /// Required designer variable.
57          /// </summary>
58          private System.ComponentModel.Container components = null;
59
60          private ArrayList m_objList; // list of packages
61          private Package m_objPackage; // current package
62          private int m_intPosition; // position of current package
63          private Random m_objRandom; // random number for package ID
64          private int m_intPackageID; // individual package number
65
66          public FrmShippingHub()
67          {
68             //
69             // Required for Windows Form Designer support
70             //
71             InitializeComponent();
72
73             //
74             // TODO: Add any constructor code after InitializeComponent
75             // call
76             //
77          }
78
79          /// <summary>
80          /// Clean up any resources being used.
81          /// </summary>
82          protected override void Dispose( bool disposing )
83          {
84             if( disposing )
85             {
86                if (components != null)
87                {
88                   components.Dispose();
89                }
90             }
91             base.Dispose( disposing );
92          }
93
94          // Windows Form Designer generated code
95
96          /// <summary>
97          /// The main entry point for the application.
98          /// </summary>
99          [STAThread]
100         static void Main()
101         {
```

```
102          Application.Run( new FrmShippingHub() );
103       }
104
105       // Form Load event
106       private void FrmShippingHub_Load(
107          object sender, System.EventArgs e )
108       {
109          m_intPosition = 0; // set initial position to zero
110          m_objRandom = new Random(); // create new Random object
111          m_intPackageID = m_objRandom.Next(
112             1, 100000 ); // new package ID
113
114          // show first state in ComboBox (using the Items property)
115          cboState.Text = Convert.ToString(
116             cboState.Items[ 0 ] );
117          m_objList = new ArrayList(); // list of packages
118
119       } // end method FrmShippingHub_Load
120
121       // New Button Click event
122       private void btnNew_Click(
123          object sender, System.EventArgs e )
124       {
125          m_intPackageID++; // increment package ID
126          m_objPackage = new Package(
127             m_intPackageID ); // create package
128
129          ClearControls(); // clear fields
130
131          // display package number and arrival time
132          lblPackageNumber.Text =
133             m_objPackage.PackageNumber.ToString();
134          lblArrivalTime.Text =
135             m_objPackage.ArrivalTime.ToString();
136
137          // only allow user to add package
138          fraAddress.Enabled = true; // disable GroupBox and controls
139          SetButtons( false ); // enable/disable Buttons
140          btnAdd.Enabled = true; // enable Add Button
141          btnNew.Enabled = false; // disable Scan New Button
142          txtAddress.Focus(); // transfer focus to txtAddress TextBox
143
144       } // end method btnNew_Click
145
146       // Add Button Click event
147       private void btnAdd_Click(
148          object sender, System.EventArgs e )
149       {
150          SetPackage(); // set Package properties from TextBoxes
151          m_objList.Add( m_objPackage ); // add package to ArrayList
152
153          // disable GroupBox and its controls
154          fraAddress.Enabled = false;
155          SetButtons( true ); // enable appropriate Buttons
156
157          // package cannot be added until Scan New is clicked
158          btnAdd.Enabled = false; // disable Add Button
159
160          // if package's state displayed, add ID to ListBox
161          if ( cboState.Text == cboViewPackages.Text )
```

```
162          {
163             lstPackages.Items.Add( m_objPackage.PackageNumber );
164          }
165
166          cboViewPackages.Text = m_objPackage.State; // list packages
167          btnNew.Enabled = true; // enable Scan New Button
168
169       } // end method btnAdd_Click
170
171       // Back Button Click event
172       private void btnBack_Click(
173          object sender, System.EventArgs e )
174       {
175          // move backward one package in the list
176          if ( m_intPosition > 0 )
177          {
178             m_intPosition--;
179          }
180          else // wrap to end of list
181          {
182             m_intPosition = m_objList.Count - 1;
183          }
184
185          LoadPackage(); // load package data from item in list
186
187       } // end method btnBack_Click
188
189       // Next Button Click event
190       private void btnNext_Click(
191          object sender, System.EventArgs e )
192       {
193          // move forward one package in the list
194          if ( m_intPosition < m_objList.Count - 1 )
195          {
196             m_intPosition++;
197          }
198          else
199          {
200             m_intPosition = 0; // wrap to beginning of list
201          }
202
203          LoadPackage(); // load package data from item in list
204
205       } // end method btnNext_Click
206
207       // Remove Button Click Event
208       private void btnRemove_Click(
209          object sender, System.EventArgs e )
210       {
211          // remove ID from ListBox if state displayed
212          if ( cboState.Text == cboViewPackages.Text )
213          {
214             lstPackages.Items.Remove( m_objPackage.PackageNumber );
215          }
216
217          // remove package from list
218          m_objList.RemoveAt( m_intPosition );
219
220          // load next package in list if there is one
221          if ( m_objList.Count > 0 )
```

```
222          {
223             // if not at first position, go to previous one
224             if ( m_intPosition > 0 )
225             {
226                m_intPosition--;
227             }
228
229             LoadPackage(); // load package data from item in list
230          }
231          else
232          {
233             ClearControls(); // clear fields
234          }
235
236          SetButtons( true ); // enable appropriate Buttons
237
238       } // end method btnRemove_Click
239
240       // Edit Button Click event
241       private void btnEditUpdate_Click(
242          object sender, System.EventArgs e )
243       {
244          // when Button reads "Edit", allow user to
245          // edit package information only
246          if ( btnEditUpdate.Text == "&Edit" )
247          {
248             fraAddress.Enabled = true;
249             SetButtons( false );
250             btnEditUpdate.Enabled = true;
251
252             // change Button text from "Edit" to "Update"
253             btnEditUpdate.Text = "&Update";
254          }
255          else
256          {
257             // when Button reads "Update" remove the old package
258             // data and add new data from TextBoxes
259             SetPackage();
260             m_objList.RemoveAt( m_intPosition );
261             m_objList.Insert( m_intPosition, m_objPackage );
262
263             // display state in ComboBox
264             cboViewPackages.Text = m_objPackage.State;
265
266             // when done, return to normal operating state
267             fraAddress.Enabled = false; // disable GroupBox
268             SetButtons( true ); // enable appropriate Buttons
269
270             // change Button text from "Update" to "Edit"
271             btnEditUpdate.Text = "&Edit";
272
273          } // end if
274
275       } // end method btnEditUpdate_Click
276
277       // set package properties
278       private void SetPackage()
279       {
280          m_objPackage.Address = txtAddress.Text;
281          m_objPackage.City = txtCity.Text;
```

```
282            m_objPackage.State =
283               Convert.ToString( cboState.SelectedItem );
284            m_objPackage.Zip = Int32.Parse( txtZip.Text );
285
286         } // end method SetPackage
287
288         // load package information into Form
289         private void LoadPackage()
290         {
291            // retrieve package from list
292            m_objPackage = ( Package ) m_objList[ m_intPosition ];
293
294            // display package data
295            txtAddress.Text = m_objPackage.Address;
296            txtCity.Text = m_objPackage.City;
297            cboState.Text = m_objPackage.State;
298            txtZip.Text = m_objPackage.Zip.ToString( "00000" );
299            lblArrivalTime.Text =
300               m_objPackage.ArrivalTime.ToString();
301            lblPackageNumber.Text =
302               m_objPackage.PackageNumber.ToString();
303
304         } // end method LoadPackage
305
306         // clear all the input controls on the Form
307         private void ClearControls()
308         {
309            txtAddress.Clear();
310            txtCity.Clear();
311            txtZip.Clear();
312            cboState.SelectedText = "";
313            lblArrivalTime.Text = "";
314            lblPackageNumber.Text = "";
315
316         } // end method ClearControls
317
318         // enable/disable Buttons
319         private void SetButtons( bool blnState )
320         {
321            btnRemove.Enabled = blnState;
322            btnEditUpdate.Enabled = blnState;
323            btnNext.Enabled = blnState;
324            btnBack.Enabled = blnState;
325
326            // disable navigation if not multiple packages
327            if ( m_objList.Count < 1 )
328            {
329               btnNext.Enabled = false;
330               btnBack.Enabled = false;
331            }
332
333            // if no items, disable Remove and Edit/Update Buttons
334            if ( m_objList.Count == 0 )
335            {
336               btnEditUpdate.Enabled = false;
337               btnRemove.Enabled = false;
338            }
339
340         } // end method SetButtons
341
```

```
342       // event raised when user selects a new state in ComboBox
343       private void cboViewPackages_SelectedIndexChanged(
344          object sender, System.EventArgs e )
345       {
346          string strState =
347             Convert.ToString( cboViewPackages.SelectedItem );
348
349          lstPackages.Items.Clear(); // clear ListBox
350
351          // list all packages for current state in ListBox
352          foreach ( Package objViewPackage in m_objList)
353          {
354             // determine if state package is being shipped to
355             // matches the state selected in the ComboBox
356             if ( objViewPackage.State == strState )
357             {
358                // add package number to the ListBox
359                lstPackages.Items.Add(
360                   objViewPackage.PackageNumber );
361             }
362
363          } // end foreach
364
365       } // end method cboViewPackages_SelectedIndexChanged
366
367       // display package information for selected package
368       private void lstPackages_DoubleClick(
369          object sender, System.EventArgs e )
370       {
371          string strPackage = ""; // String for package information
372
373          // check if the lstPackages ListBox is empty
374          if ( lstPackages.SelectedIndex != -1 )
375          {
376             foreach ( Package objPackageInformation in m_objList )
377             {
378                // if the package currently in objPackageInformation
379                // matches the user's selected package
380                if ( objPackageInformation.PackageNumber ==
381                   Convert.ToInt32( lstPackages.SelectedItem ) )
382                {
383                   strPackage += ( "Package " +
384                      objPackageInformation.PackageNumber +
385                      "\r\nArrived at: " +
386                      objPackageInformation.ArrivalTime +
387                      "\r\nAddress:" + objPackageInformation.Address +
388                      "\r\nCity: " +  objPackageInformation.City +
389                      "\r\nState: " + objPackageInformation.State +
390                      "\r\nZip code: " +
391                      objPackageInformation.Zip.ToString( "00000" ) );
392                }
393             }
394          }
395          else
396          {
397             // if the user select a blank item in the ListBox
398             strPackage = "Please select a package";
399          }
400
401          MessageBox.Show( strPackage, "Package Information",
```

```
            MessageBoxButtons.OK, MessageBoxIcon.Information );

        } // end method lstPackages_DoubleClick

    } // end class FrmShippingHub
}
```

20.13 (*Controls Collection Application*) C# provides many different types of collections. One such collection is the `Controls` collection, which is used to provide access to all of the controls on a `Form`. Create an application that uses the `Controls` collection and a `foreach` loop to iterate through each control on the `Form`. As each control is encountered, add the control's name to a `ListBox`, and change the control's background color (in Fig. 20.28, `Color.Wheat` is used).

Figure 20.28 Controls Collection application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial20\Exercises\ControlsCollection` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ControlsCollection.sln` in the `ControlsCollection` directory to open the application.

c) ***Generating an event handler.*** Double click the **Submit** `Button` in design view to create an event handler for the click event.

d) ***Declaring a control variable.*** Declare a reference of the `Control` type. This reference represents each element in the `foreach` statement as it iterates through each `Control` on the `Form`.

e) ***Clearing the `ListBox`.*** To ensure that the information in the `ListBox` is updated each time the **Submit** `Button` is clicked, clear the `ListBox` of all items.

f) ***Writing a `foreach` loop.*** To create the `foreach` loop, use the control variable that you created to iterate through the `Form`'s `Controls` collection.

g) ***Adding each control's name to the `ListBox`.*** Use the `ListBox`'s `Add` method to insert the name of each control on the `Form`. Recall that a control's `Name` property contains the name of the control.

h) ***Changing the control's background color.*** Use the `Control`'s `BackColor` property to change the control's background color. Set the property to a new color using a member of the `Color` structure. [*Hint*: Type the word `Color` followed by the member-access operator to display a list of predefined colors using the *Intellisense* feature.] Note that the color of the `PictureBox` does not appear to change because its image displays in the control's foreground.

i) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Submit** `Button`. Verify that the controls' background colors change, and that all the controls are listed in the **List of controls:** `ListBox`.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 20.13 Solution
// ControlsCollection.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ControlsCollection
{
   /// <summary>
   /// Summary description for FrmControlsCollection.
   /// </summary>
   public class FrmControlsCollection : System.Windows.Forms.Form
   {
      // some controls on the form
      private System.Windows.Forms.Label lblName;
      private System.Windows.Forms.TextBox txtName;
      private System.Windows.Forms.Label lblBook;
      private System.Windows.Forms.ComboBox cboBook;
      private System.Windows.Forms.Label lblCover;
      private System.Windows.Forms.PictureBox picBook;

      // Label and ListBox to list the controls on the form
      private System.Windows.Forms.Label lblList;
      private System.Windows.Forms.ListBox lstList;

      // Button to loop through the controls
      private System.Windows.Forms.Button btnSubmit;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmControlsCollection()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
```

```
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmControlsCollection() );
75       }
76
77       // handles Submit Button's Click event
78       private void btnSubmit_Click(
79          object sender, System.EventArgs e )
80       {
81          lstList.Items.Clear(); // clear ListBox
82
83          // iterate through controls collection
84          foreach ( Control objControl in Controls )
85          {
86             // list name of each control
87             lstList.Items.Add( objControl.Name );
88
89             // change background color
90             objControl.BackColor = Color.Wheat;
91          }
92
93       } // end method btnSubmit_Click
94
95    } // end class FrmControlsCollection
96 }
```

What does this code do? ▶

20.14 What is the result of the following code?

```
1  ArrayList intList = new ArrayList();
2  string strOutput = "";
3
4  intList.Add( 1 );
5  intList.Add( 3 );
6  intList.Add( 5 );
7
8  foreach ( int intListItems in intList )
9  {
10    strOutput += ( " " + intListItems.ToString() );
11 }
12
13 MessageBox.Show( strOutput, "Mystery",
14    MessageBoxButtons.OK, MessageBoxIcon.Information );
```

Answer: This code creates an `ArrayList` and adds to it the values `1`, `3` and `5`. The values are then appended to each other (separated by spaces) using a `foreach` statement, and the result is displayed in a `MessageBox`. The complete code reads:

```
// Exercise 20.14 Solution
// MessageBox.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace MessageBoxes
{
   /// <summary>
   /// Summary description for FrmMessageBox.
   /// </summary>
   public class FrmMessageBox : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblOutputOut;
      private System.Windows.Forms.Label lblOutput;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmMessageBox()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
```

```
60         static void Main()
61         {
62            Application.Run( new FrmMessageBox() );
63         }
64
65         private void FrmMessageBox_Load(
66            object sender, System.EventArgs e )
67         {
68            ArrayList intList = new ArrayList();
69            string strOutput = "";
70
71            intList.Add( 1 );
72            intList.Add( 3 );
73            intList.Add( 5 );
74
75            foreach ( int intListItems in intList )
76            {
77               strOutput += ( " " + intListItems.ToString() );
78            }
79
80            MessageBox.Show( strOutput, "Mystery",
81               MessageBoxButtons.OK, MessageBoxIcon.Information );
82
83         } // end method FrmMessageBox_Load
84
85      } // end class FrmMessageBox
86   }
```

What's wrong with this code?

20.15 This code should iterate through an array of `Packages` in the `objList` `ArrayList` and print each package's number in the `lblDisplay` `Label`. Assume `objList` has already been created and has had packages added to it. Find the error(s) in the following code.

```
1  foreach ( ArrayList objValue in objList )
2  {
3     lblDisplay.Text += ( " " + objValue.PackageNumber );
4  }
```

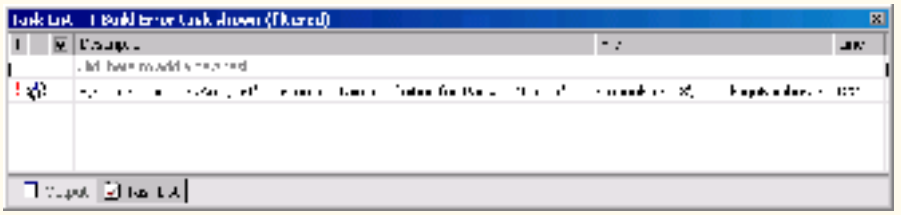
Answer: The reference that specifies the element is of the wrong type to iterate through this list of `Packages`. The element reference should be of type `Package`. The complete incorrect code reads:

```
1  // Exercise 20.15 Solution
2  // PackageNumber.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
```

```
10
11 namespace PackageNumber
12 {
13    /// <summary>
14    /// Summary description for FrmPackageNumber.
15    /// </summary>
16    public class FrmPackageNumber : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.Label lblPackageNumbers;
20
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.Container components = null;
25
26       public FrmPackageNumber()
27       {
28          //
29          // Required for Windows Form Designer support
30          //
31          InitializeComponent();
32
33          //
34          // TODO: Add any constructor code after InitializeComponent
35          // call
36          //
37       }
38
39       /// <summary>
40       /// Clean up any resources being used.
41       /// </summary>
42       protected override void Dispose( bool disposing )
43       {
44          if( disposing )
45          {
46             if (components != null)
47             {
48                components.Dispose();
49             }
50          }
51          base.Dispose( disposing );
52       }
53
54       // Windows Form Designer generated code
55
56       /// <summary>
57       /// The main entry point for the application.
58       /// </summary>
59       [STAThread]
60       static void Main()
61       {
62          Application.Run( new FrmPackageNumber() );
63       }
64
65       // prints each package number in an ArrayList
66       private void FrmPackageNumber_Load(
67          object sender, System.EventArgs e )
68       {
69          ArrayList objList = new ArrayList();
```

```
70
71          objList.Add( new Package( 3 ) );
72          objList.Add( new Package( 15 ) );
73          objList.Add( new Package( 89 ) );
74          objList.Add( new Package( 101 ) );
75          objList.Add( new Package( 150 ) );
76          objList.Add( new Package( 176 ) );
77          objList.Add( new Package( 215 ) );
78          objList.Add( new Package( 226 ) );
79
80          foreach ( ArrayList objValue in objList )
81          {
82             lblDisplay.Text += ( " " + objValue.PackageNumber );
83          }
84
85       } // end method FrmPackageNumber_Load
86
87    } // end class FrmPackageNumber
88 }
```

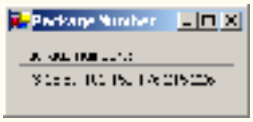
`objValue` should be of type `Package`, not `ArrayList`



Answer: The complete corrected code should read:

```
1  // Exercise 20.15 Solution
2  // PackageNumber.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace PackageNumber
12 {
13    /// <summary>
14    /// Summary description for FrmPackageNumber.
15    /// </summary>
16    public class FrmPackageNumber : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblDisplay;
19       private System.Windows.Forms.Label lblPackageNumbers;
20
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.Container components = null;
25
26       public FrmPackageNumber()
27       {
28          //
```

```
29          // Required for Windows Form Designer support
30          //
31          InitializeComponent();
32
33          //
34          // TODO: Add any constructor code after InitializeComponent
35          // call
36          //
37       }
38
39       /// <summary>
40       /// Clean up any resources being used.
41       /// </summary>
42       protected override void Dispose( bool disposing )
43       {
44          if( disposing )
45          {
46             if (components != null)
47             {
48                components.Dispose();
49             }
50          }
51          base.Dispose( disposing );
52       }
53
54       // Windows Form Designer generated code
55
56       /// <summary>
57       /// The main entry point for the application.
58       /// </summary>
59       [STAThread]
60       static void Main()
61       {
62          Application.Run( new FrmPackageNumber() );
63       }
64
65       // prints each package number in an ArrayList
66       private void FrmPackageNumber_Load(
67          object sender, System.EventArgs e )
68       {
69          ArrayList objList = new ArrayList();
70
71          objList.Add( new Package( 3 ) );
72          objList.Add( new Package( 15 ) );
73          objList.Add( new Package( 89 ) );
74          objList.Add( new Package( 101 ) );
75          objList.Add( new Package( 150 ) );
76          objList.Add( new Package( 176 ) );
77          objList.Add( new Package( 215 ) );
78          objList.Add( new Package( 226 ) );
79
80          foreach ( Package objValue in objList )
81          {
82             lblDisplay.Text += ( " " + objValue.PackageNumber );
83          }
84
85       } // end method FrmPackageNumber_Load
86
87    } // end class FrmPackageNumber
88 }
```



Programming Challenge

20.16 (***Enhanced Shipping Hub Application***) Enhance the **Shipping Hub** application created in Exercise 20.12 to allow the user to move a maximum of five packages from the warehouse to a truck for shipping (Fig. 20.29). If you have not completed Exercise 20.12, follow the steps in Exercise 20.12 before proceeding to the next step. If you have completed Exercise 20.12, copy the code you added to the `lstPackages ListBox`'s `DoubleClick` event handler to the same event handler in this application before beginning the exercise.

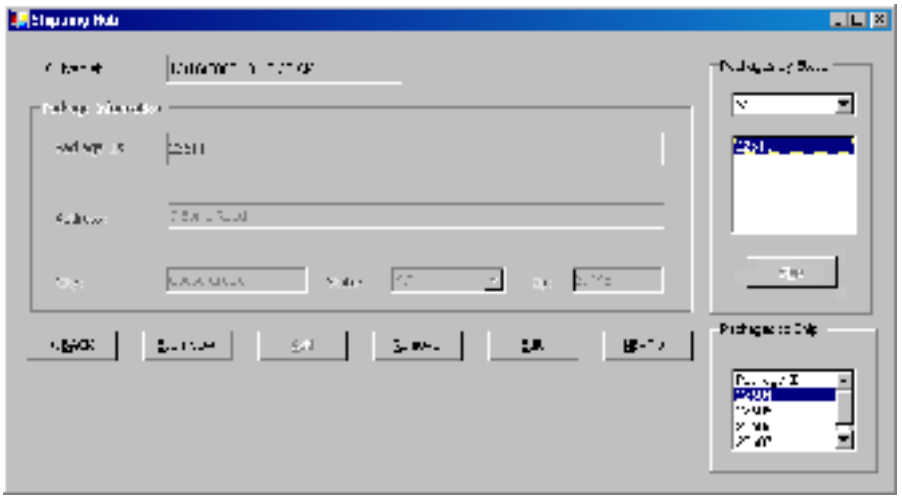


Figure 20.29 **Enhanced Shipping Hub** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial20\Exercises\ShippingHubEnhanced` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `ShippingHub.sln` in the `ShippingHubEnhanced` directory to open the application.

c) ***Enabling the Ship `Button`.*** The **Ship** `Button` should not be enabled until a package is selected in the `lstPackage ListBox`. Double click the `lstPackage ListBox` from design view to define the event handler. Use the `Button`'s `Enabled` property to enable the `Button` if the `SelectedIndex` of the `ListBox` is not `-1`. This means that when the user selects a package from the `ListBox`, the user can send the package to the truck by clicking the **Ship** `Button`. Also, insert a line of code after the `foreach` statement in the `SelectedIndexChanged` event handler to disable the **Ship** `Button` when a user chooses a different state.

d) ***Defining the Ship `Button`'s `Click` event***. Double click the **Ship** `Button` in **Design View** to define the `Click` event.

e) ***Incrementing the counter***. Because you are only allowing five packages to be "shipped," declare an instance variable that will track how many packages have been placed onto the truck. Increment the variable each time the **Ship** `Button` is clicked.

f) ***Creating temporary variables***. Create two temporary `Package` references to store the correct package's information. Use `objTempPackage` as the reference to the element in the collection type of a `foreach` statement, and the `objTruckPackage` as a reference to the package added to the truck.

g) ***Using the `if...else` statement***. Use an `if...else` statement to allow packages to be placed onto the truck if the number of packages on the truck is less than five.

h) ***Using the foreach loop***. Use a `foreach` loop to iterate through the values in `m_objList`. Each iteration should determine whether the current package is the one selected from the `ListBox`.

i) ***Adding the package to the truck***. When the `foreach` loop has located the correct package, add that package to the truck by adding the reference to `objTempPackage` to the truck's `ArrayList`, `m_objTruckList`. Then assign the value in `objTempPackage` (the package sent to the truck) to `objTruckPackage`.

j) ***Removing the package***. When the `foreach` loop completes, remove the package meant for the truck from `m_objList` and the `lstPackages ListBox`.

k) ***Displaying the package in the `ListBox`***. Use a `foreach` loop that iterates through each package in the `m_objTruckList ArrayList` and displays each package in the `lstTruck ListBox`.

l) ***Refreshing the GUI***. Call the `ClearControls` and `SetButtons` methods to clear the `TextBoxes` and enable the appropriate `Buttons`. Also, set the **Ship** `Button`'s `Enabled` property to `false`.

m) ***Coding the `else` block.*** Display a `MessageBox` that notifies the user if the number of packages on the truck is already five. Then, disable the **Ship** `Button`.

n) ***Running the application.*** Select **Debug > Start** to run your application. Add several packages. In the **Packages by State** `GroupBox`, select several packages and add them to the **Packages to Ship** `ListBox`. Verify that you can add only 5 packages to this `ListBox`.

o) ***Closing the application.*** Close your running application by clicking its close box.

p) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 20.16 Solution
// ShippingHub.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ShippingHub
{
   /// <summary>
   /// Summary description for FrmShippingHub.
   /// </summary>
   public class FrmShippingHub : System.Windows.Forms.Form
   {
      // Labels for the arrival time
      private System.Windows.Forms.Label lblArrived;
      private System.Windows.Forms.Label lblArrivalTime;

      // GroupBox for the package information
      private System.Windows.Forms.GroupBox fraAddress;

      // Labels for the package ID
      private System.Windows.Forms.Label lblPackageID;
      private System.Windows.Forms.Label lblPackageNumber;

      // Labels, TextBoxes and ComboBox for the full address
      private System.Windows.Forms.Label lblAddress;
      private System.Windows.Forms.TextBox txtAddress;
      private System.Windows.Forms.Label lblCity;
      private System.Windows.Forms.TextBox txtCity;
```

```
34          private System.Windows.Forms.Label lblState;
35          private System.Windows.Forms.ComboBox cboState;
36          private System.Windows.Forms.Label lblZip;
37          private System.Windows.Forms.TextBox txtZip;
38
39          // GroupBox for choosing packages by state.
40          // contains a ComboBox and a ListBox for doing so.
41          // also contains Button to ship a package.
42          private System.Windows.Forms.GroupBox fraListByState;
43          private System.Windows.Forms.ComboBox cboViewPackages;
44          private System.Windows.Forms.ListBox lstPackages;
45          private System.Windows.Forms.Button btnShip;
46
47          // GroupBox with ListBox inside for displaying package IDs
48          private System.Windows.Forms.GroupBox fraPackagesToShip;
49          private System.Windows.Forms.ListBox lstTruck;
50
51          // Buttons to go back one package, scan a new package,
52          // add a package, remove a package, edit an existing package
53          // and go forward one package.
54          private System.Windows.Forms.Button btnBack;
55          private System.Windows.Forms.Button btnNew;
56          private System.Windows.Forms.Button btnAdd;
57          private System.Windows.Forms.Button btnRemove;
58          private System.Windows.Forms.Button btnEditUpdate;
59          private System.Windows.Forms.Button btnNext;
60
61          /// <summary>
62          /// Required designer variable.
63          /// </summary>
64          private System.ComponentModel.Container components = null;
65
66          private ArrayList m_objList; // list of packages
67          private Package m_objPackage; // current package
68          private int m_intPosition; // position of current package
69          private Random m_objRandom; // random number for package ID
70          private int m_intPackageID; // individual package number
71          private ArrayList m_objTruckList; // list of shipment
72          private int m_intCounter = 0; // count packages on truck
73
74          public FrmShippingHub()
75          {
76             //
77             // Required for Windows Form Designer support
78             //
79             InitializeComponent();
80
81             //
82             // TODO: Add any constructor code after InitializeComponent
83             // call
84             //
85          }
86
87          /// <summary>
88          /// Clean up any resources being used.
89          /// </summary>
90          protected override void Dispose( bool disposing )
91          {
92             if( disposing )
93             {
```

```
94             if (components != null)
95             {
96                components.Dispose();
97             }
98          }
99          base.Dispose( disposing );
100       }
101
102       // Windows Form Designer generated code
103
104       /// <summary>
105       /// The main entry point for the application.
106       /// </summary>
107       [STAThread]
108       static void Main()
109       {
110          Application.Run( new FrmShippingHub() );
111       }
112
113       // Form Load event
114       private void FrmShippingHub_Load(
115          object sender, System.EventArgs e )
116       {
117          m_intPosition = 0; // set initial position to zero
118          m_objRandom = new Random(); // create new Random object
119          m_intPackageID = m_objRandom.Next(
120             1, 100000 ); // new package ID
121
122          // show first state in ComboBox (using the Items property)
123          cboState.Text = Convert.ToString(
124             cboState.Items[ 0 ] );
125          m_objList = new ArrayList(); // list of packages
126          m_objTruckList = new ArrayList(); // truck list
127
128       } // end method FrmShippingHub_Load
129
130       // New Button Click event
131       private void btnNew_Click(
132          object sender, System.EventArgs e )
133       {
134          m_intPackageID++; // increment package ID
135          m_objPackage = new Package(
136             m_intPackageID ); // create package
137
138          ClearControls(); // clear fields
139
140          // display package number and arrival time
141          lblPackageNumber.Text =
142             m_objPackage.PackageNumber.ToString();
143          lblArrivalTime.Text =
144             m_objPackage.ArrivalTime.ToString();
145
146          // only allow user to add package
147          fraAddress.Enabled = true; // disable GroupBox and controls
148          SetButtons( false ); // enable/disable Buttons
149          btnAdd.Enabled = true; // enable Add Button
150          btnNew.Enabled = false; // disable Scan New Button
151          txtAddress.Focus(); // transfer focus to txtAddress TextBox
152
153       } // end method btnNew_Click
```

```
154
155       // Add Button Click event
156       private void btnAdd_Click(
157          object sender, System.EventArgs e )
158       {
159          SetPackage(); // set Package properties from TextBoxes
160
161          // disable GroupBox and its controls
162          fraAddress.Enabled = false;
163          SetButtons( true ); // enable appropriate Buttons
164          m_objList.Add( m_objPackage ); // add package to ArrayList
165
166          // package cannot be added until Scan New is clicked
167          btnAdd.Enabled = false; // disable Add Button
168
169          // if package's state displayed, add ID to ListBox
170          if ( cboState.Text == cboViewPackages.Text )
171          {
172             lstPackages.Items.Add( m_objPackage.PackageNumber );
173          }
174
175          cboViewPackages.Text = m_objPackage.State; // list packages
176          btnNew.Enabled = true; // enable Scan New Button
177
178       } // end method btnAdd_Click
179
180       // Back Button Click event
181       private void btnBack_Click(
182          object sender, System.EventArgs e )
183       {
184          // move backward one package in the list
185          if ( m_intPosition > 0 )
186          {
187             m_intPosition--;
188          }
189          else // wrap to end of list
190          {
191             m_intPosition = m_objList.Count - 1;
192          }
193
194          LoadPackage(); // load package data from item in list
195
196       } // end method btnBack_Click
197
198       // Next Button Click event
199       private void btnNext_Click(
200          object sender, System.EventArgs e )
201       {
202          // move forward one package in the list
203          if ( m_intPosition < m_objList.Count - 1 )
204          {
205             m_intPosition++;
206          }
207          else
208          {
209             m_intPosition = 0; // wrap to beginning of list
210          }
211
212          LoadPackage(); // load package data from item in list
213
```

```
214       } // end method btnNext_Click
215
216       // Remove Button Click Event
217       private void btnRemove_Click(
218          object sender, System.EventArgs e )
219       {
220          // remove ID from ListBox if state displayed
221          if ( cboState.Text == cboViewPackages.Text )
222          {
223             lstPackages.Items.Remove( m_objPackage.PackageNumber );
224          }
225
226          // remove package from list
227          m_objList.RemoveAt( m_intPosition );
228
229          // load next package in list if there is one
230          if ( m_objList.Count > 0 )
231          {
232             // if not at first position, go to previous one
233             if ( m_intPosition > 0 )
234             {
235                m_intPosition--;
236             }
237
238             LoadPackage(); // load package data from item in list
239          }
240          else
241          {
242             ClearControls(); // clear fields
243          }
244
245          SetButtons( true ); // enable appropriate Buttons
246
247       } // end method btnRemove_Click
248
249       // Edit Button Click event
250       private void btnEditUpdate_Click(
251          object sender, System.EventArgs e )
252       {
253          // when Button reads "Edit", allow user to
254          // edit package information only
255          if ( btnEditUpdate.Text == "&Edit" )
256          {
257             fraAddress.Enabled = true;
258             SetButtons( false );
259             btnEditUpdate.Enabled = true;
260
261             // change Button text from "Edit" to "Update"
262             btnEditUpdate.Text = "&Update";
263          }
264          else
265          {
266             // when Button reads "Update" remove the old package
267             // data and add new data from TextBoxes
268             SetPackage();
269             m_objList.RemoveAt( m_intPosition );
270             m_objList.Insert( m_intPosition, m_objPackage );
271
272             // display state in ComboBox
273             cboViewPackages.Text = m_objPackage.State;
```

```
274
275            // when done, return to normal operating state
276            fraAddress.Enabled = false; // disable GroupBox
277            SetButtons( true ); // enable appropriate Buttons
278
279            // change Button text from "Update" to "Edit"
280            btnEditUpdate.Text = "&Edit";
281
282         } // end if
283
284      } // end method btnEditUpdate_Click
285
286      // set package properties
287      private void SetPackage()
288      {
289         m_objPackage.Address = txtAddress.Text;
290         m_objPackage.City = txtCity.Text;
291         m_objPackage.State =
292            Convert.ToString( cboState.SelectedItem );
293         m_objPackage.Zip = Int32.Parse( txtZip.Text );
294
295      } // end method SetPackage
296
297      // load package information into Form
298      private void LoadPackage()
299      {
300         // retrieve package from list
301         m_objPackage = ( Package ) m_objList[ m_intPosition ];
302
303         // display package data
304         txtAddress.Text = m_objPackage.Address;
305         txtCity.Text = m_objPackage.City;
306         cboState.Text = m_objPackage.State;
307         txtZip.Text = m_objPackage.Zip.ToString( "00000" );
308         lblArrivalTime.Text =
309            m_objPackage.ArrivalTime.ToString();
310         lblPackageNumber.Text =
311            m_objPackage.PackageNumber.ToString();
312
313      } // end method LoadPackage
314
315      // clear all the input controls on the Form
316      private void ClearControls()
317      {
318         txtAddress.Clear();
319         txtCity.Clear();
320         txtZip.Clear();
321         cboState.SelectedText = "";
322         lblArrivalTime.Text = "";
323         lblPackageNumber.Text = "";
324
325      } // end method ClearControls
326
327      // enable/disable Buttons
328      private void SetButtons( bool blnState )
329      {
330         btnRemove.Enabled = blnState;
331         btnEditUpdate.Enabled = blnState;
332         btnNext.Enabled = blnState;
333         btnBack.Enabled = blnState;
```

```
334
335          // disable navigation if not multiple packages
336          if ( m_objList.Count < 1 )
337          {
338             btnNext.Enabled = false;
339             btnBack.Enabled = false;
340          }
341
342          // if no items, disable Remove and Edit/Update Buttons
343          if ( m_objList.Count == 0 )
344          {
345             btnEditUpdate.Enabled = false;
346             btnRemove.Enabled = false;
347          }
348
349       } // end method SetButtons
350
351       // event raised when user selects a new state in ComboBox
352       private void cboViewPackages_SelectedIndexChanged(
353          object sender, System.EventArgs e )
354       {
355          string strState =
356             Convert.ToString( cboViewPackages.SelectedItem );
357
358          lstPackages.Items.Clear(); // clear ListBox
359
360          // list all packages for current state in ListBox
361          foreach ( Package objViewPackage in m_objList )
362          {
363             // determine if state package is being shipped to
364             // matches the state selected in the ComboBox
365             if ( objViewPackage.State == strState )
366             {
367                // add package number to the ListBox
368                lstPackages.Items.Add(
369                   objViewPackage.PackageNumber );
370             }
371
372          } // end foreach
373
374          btnShip.Enabled = false; // disable Ship Button
375
376       } // end method cboViewPackages_SelectedIndexChanged
377
378       // display package information for selected package
379       private void lstPackages_DoubleClick(
380          object sender, System.EventArgs e )
381       {
382          string strPackage = ""; // String for package information
383
384          // check if the lstPackages ListBox is empty
385          if ( lstPackages.SelectedIndex != -1 )
386          {
387             foreach ( Package objPackageInformation in m_objList )
388             {
389                // if the package currently in objPackageInformation
390                // matches the user's selected package
391                if ( objPackageInformation.PackageNumber ==
392                   Convert.ToInt32( lstPackages.SelectedItem ) )
393                {
```

```
394               strPackage += ( "Package " +
395                  objPackageInformation.PackageNumber +
396                  "\r\nArrived at: " +
397                  objPackageInformation.ArrivalTime +
398                  "\r\nAddress:" + objPackageInformation.Address +
399                  "\r\nCity: " +  objPackageInformation.City +
400                  "\r\nState: " + objPackageInformation.State +
401                  "\r\nZip code: " +
402                  objPackageInformation.Zip.ToString( "00000" ) );
403            }
404         }
405      }
406      else
407      {
408         // if the user select a blank item in the ListBox
409         strPackage = "Please select a package";
410      }
411
412      MessageBox.Show( strPackage, "Package Information",
413         MessageBoxButtons.OK, MessageBoxIcon.Information );
414
415   } // end method lstPackages_DoubleClick
416
417   // allow packages to be shipped
418   private void btnShip_Click(
419      object sender, System.EventArgs e )
420   {
421      Package objTruckPackage = null; // package to remove
422
423      m_intCounter++; // increment package count
424
425      // if there is less than 6 packages in m_intCounter
426      if ( m_intCounter <= 5 )
427      {
428         // for each package from the m_objList ArrayList
429         foreach ( Package objTempPackage in m_objList )
430         {
431            // move package to truck
432            if ( objTempPackage.PackageNumber ==
433               Convert.ToInt32( lstPackages.SelectedItem ) )
434            {
435               m_objTruckList.Add( objTempPackage );
436               objTruckPackage = objTempPackage;
437            }
438         }
439
440         // remove the package from warehouse
441         m_objList.Remove( objTruckPackage );
442
443         // remove selected package
444         lstPackages.Items.Remove( lstPackages.SelectedItem );
445
446         lstTruck.Items.Clear(); // clear ListBox
447         lstTruck.Items.Add( "Package ID:" ); // add header
448
449         // list all packages in ListBox
450         foreach ( Package objTempPackage in m_objTruckList )
451         {
452            // add package to lstTruck ListBox
453            lstTruck.Items.Add( objTempPackage.PackageNumber );
```

```
454         }
455
456         btnShip.Enabled = false; // disable the Ship Button
457         ClearControls(); // clear the TextBoxes
458         SetButtons( true ); // enable appropriate Buttons
459      }
460      else
461      {
462         MessageBox.Show( "Truck can only hold 5 packages",
463            "Limit Exceeded",
464            MessageBoxButtons.OK, MessageBoxIcon.Exclamation );
465
466         btnShip.Enabled = false;
467      }
468
469   } // end method btnShip_Click
470
471   // disable Ship button when no package is selected
472   private void lstPackages_SelectedIndexChanged(
473      object sender, System.EventArgs e )
474   {
475      if ( lstPackages.SelectedIndex != -1 )
476      {
477         btnShip.Enabled = true; // enable the Ship Button
478      }
479
480   } // end method lstPackages_SelectedIndexChanged
481
482 } // end class FrmShippingHub
483 }
```

21 TUTORIAL

"Cat and Mouse" Painter Application

Introducing the `Graphics` Object and Mouse Events

Solutions

Instructor's Manual Exercise Solutions Tutorial 21

MULTIPLE-CHOICE QUESTIONS

21.1 The *x*- and *y*-coordinates of the `MouseEventArgs` object are relative to _____.

a) the screen
b) the application
c) the `Form` or control that contains the control that raised the event
d) None of the above.

21.2 The _____ method of the `Graphics` class draws a solid ellipse.

a) `FillEllipse`
b) `Ellipse`
c) `SolidEllipse`
d) `DrawEllipse`

21.3 The _____ object passed to a mouse event handler contains information about the mouse event that was raised.

a) `EventHandler`
b) `MouseEventHandler`
c) `MouseEventArgs`
d) `EventArgs`

21.4 The _____ event is raised when a mouse button is pressed.

a) `MousePress`
b) `MouseClick`
c) `MouseDown`
d) `MouseButtonDown`

21.5 A _____ is used to fill a shape with color using a `Graphics` object.

a) painter
b) brush
c) paint bucket
d) marker

21.6 A _____ event is raised every time the mouse interacts with a control.

a) control
b) mouse pointer
c) mouse
d) user

21.7 The _____ property of `MouseEventArgs` specifies which mouse button was pressed.

a) `Source`
b) `Button`
c) `WhichButton`
d) `ButtonPressed`

21.8 The _____ class contains methods for drawing text, lines, rectangles and other shapes.

a) `Pictures`
b) `Drawings`
c) `Graphics`
d) `Illustrations`

21.9 An ellipse with its _____ is a circle.

a) height twice the length of its width
b) width set to zero
c) height half the length of its width
d) height equal to its width

21.10 The _____ method creates a `Graphics` object.

a) `NewGraphics`
b) `CreateGraphics`
c) `PaintGraphics`
d) `InitializeGraphics`

Answers: 21.1) c. 21.2) a. 21.3) c. 21.4) c. 21.5) b. 21.6) c. 21.7) b. 21.8) c. 21.9) d. 21.10) b.

EXERCISES

21.11 ***(Line Length Application)*** The **Line Length** application should draw a straight black line on the `Form` and calculate the length of the line (Fig. 21.28). The line should begin at the coordinates where the mouse button is pressed and should stop at the point where the mouse button is released. The application should display the line's length (that is, the distance between the two endpoints) in the `Label` **Length =**. Use the following formula to calculate

the line's length, where (x_1, y_1) is the first endpoint (the coordinates where the mouse button is pressed) and (x_2, y_2) is the second endpoint (the coordinates where the mouse button is released). To calculate the distance (or length) between the two points, use the following equation:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

To draw a straight line, you need to use the `DrawLine` method on a `Graphics` object. When drawing lines, you should use a **`Pen`** object, which is an object used to specify characteristics of lines and curves. Use the following method call to draw a black line between the two points using a `Graphics` object reference `objGraphic`:

```
objGraphic.DrawLine( new Pen( Color.Black ),x1, y1, x2, y2 );
```

Figure 21.28 **Line Length** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial21\Exercises\LineLength` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `LineLength.sln` in the `LineLength` directory to open the application.

c) ***Declaring instance variables.*** Declare a reference to a `Graphics` object that you will use to draw a line. Then, declare four `int`s in which you will store the *x*- and *y*-coordinates of the two points.

d) ***Adding a `Load` event handler.*** Create a `Load` event handler. Add code to the `Load` event handler to initialize the `Graphics` object.

e) ***Adding a `MouseDown` event handler.*** Create a `MouseDown` event handler. Add code to the `MouseDown` event handler to store the coordinates of the first endpoint of the line.

f) ***Creating the `Distance` method.*** Define a method named `Length` that returns the distance between two endpoints as a `double`. The method should use the following statement to perform the line length calculation, where `intXDistance` is the difference between the *x*-coordinates of the two points and `intYDistance` is the difference between the *y*-coordinates of the two points:

```
Math.Sqrt( ( intXDistance * intXDistance ) +
   ( intYDistance * intYDistance ) );
```

g) ***Adding a `MouseUp` event handler.*** Create a `MouseUp` event handler. First store the coordinates of the line's second endpoint. Then, call the `Length` method to obtain the distance between the two endpoints (the line's length). Finally, display the line on the `Form` and the line's length in the **Length =** `Label`, as in Fig. 21.28.

h) ***Running the application.*** Select **Debug > Start** to run your application. Draw several lines and view their lengths. Verify that the length values are accurate.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 21.11 Solution
// LineLength.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace LineLength
{
   /// <summary>
   /// Summary description for FrmLineLength.
   /// </summary>
   public class FrmLineLength : System.Windows.Forms.Form
   {
      // Labels to display length of line drawn
      private System.Windows.Forms.Label lblOutput;
      private System.Windows.Forms.Label lblLength;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // declare Graphics object
      private Graphics m_objGraphic;

      private int m_intX1; // first point's x-coordinate
      private int m_intY1; // first point's y-coordinate
      private int m_intX2; // second point's x-coordinate
      private int m_intY2; // second point's y-coordinate

      public FrmLineLength()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
```

```
60             base.Dispose( disposing );
61          }
62
63          // Windows Form Designer generated code
64
65          /// <summary>
66          /// The main entry point for the application.
67          /// </summary>
68          [STAThread]
69          static void Main()
70          {
71             Application.Run( new FrmLineLength() );
72          }
73
74          // handles Forms' Load event
75          private void FrmLineLength_Load(
76             object sender, System.EventArgs e )
77          {
78             // create Graphics object
79             m_objGraphic = CreateGraphics();
80
81          } // end method FrmLineLength_Load
82
83          // handles Form's MouseDown event
84          private void FrmLineLength_MouseDown(
85             object sender, System.Windows.Forms.MouseEventArgs e )
86          {
87             lblLength.Text = ""; // clear Label
88
89             // get x- and y- coordinates of mouse click
90             m_intX1 = e.X;
91             m_intY1 = e.Y;
92
93          } // end method FrmLineLength_MouseDown
94
95          // returns distance between two points
96          private double Length()
97          {
98             // horizontal distance
99             int intXDistance = m_intX1 - m_intX2;
100
101            // vertical distance
102            int intYDistance = m_intY1 - m_intY2;
103
104            return Math.Sqrt( ( intXDistance * intXDistance ) +
105               ( intYDistance * intYDistance ) );
106
107         } // end method Length
108
109         // handles Form's MouseUp event
110         private void FrmLineLength_MouseUp(
111            object sender, System.Windows.Forms.MouseEventArgs e )
112         {
113            // final point
114            m_intX2 = e.X;
115            m_intY2 = e.Y;
116
117            // distance between two points
118            double dblDistance = Length();
119
```

```
120         // draw line connecting the two points
121         m_objGraphic.DrawLine( new Pen( Color.Black ),
122            m_intX1, m_intY1, m_intX2, m_intY2 );
123
124         // display distance in Label
125         lblLength.Text = String.Format( "{0:F}", dblDistance );
126
127      } // end method FrmLineLength_MouseUp
128
129   } // end class FrmLineLength
130 }
```

21.12 ***(Circle Painter Application)*** The **Circle Painter** application should draw a blue circle with a randomly chosen size when the user presses a mouse button anywhere over the `Form` (Fig. 21.29). The application should randomly select a circle diameter in the range from `5` to `199`, inclusive. To draw a blue circle with a given diameter (`intDiameter`), use the following statement:

```
objGraphic.DrawEllipse( new Pen( Color.Blue ), e.X, e.Y,
   intDiameter, intDiameter );
```

The `DrawEllipse` method, when passed a `Pen` (instead of a brush) as an argument, draws the outline of an ellipse. Recall that an ellipse is a circle if the height and width arguments are the same (in this case, the randomly selected `intDiameter`). Use the *x*- and *y*- coordinates of the `MouseDown` event as the *x*- and *y*- coordinates of the circle's bounding rectangle (that is, the second and third arguments to the `DrawEllipse` method). Notice that the first argument to the `DrawEllipse` method is a `Pen` object. See Exercise 21.11 for a description of `Pen`.

Figure 21.29 **Circle Painter** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial21\Exercises\CirclePainter` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CirclePainter.sln` in the `CirclePainter` directory to open the application.

c) ***Adding a `MouseDown` event handler.*** Create a `MouseDown` event handler. In the event handler, retrieve the *x*- and *y*-coordinates of the location of the mouse pointer when a mouse button was pressed. Then, generate a random number to use as the circle's diameter, using a `Random` object, and store it in a variable. Finally, call the `DrawEllipse` method on a reference to a `Graphics` object to draw a blue circle on the `Form` with the diameter generated by the `Random` object.

d) ***Running the application.*** Select **Debug > Start** to run your application. Draw several blue circles and make sure that they are of different sizes.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 21.12 Solution
// CirclePainter.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CirclePainter
{
   /// <summary>
   /// Summary description for FrmCirclePainter.
   /// </summary>
   public class FrmCirclePainter : System.Windows.Forms.Form
   {
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCirclePainter()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmCirclePainter() );
```

```
60         }
61
62         // handles Form's MouseDown event
63         private void FrmCirclePainter_MouseDown(
64            object sender, System.Windows.Forms.MouseEventArgs e )
65         {
66            // initialize Graphics object
67            Graphics objGraphic = CreateGraphics();
68
69            // initialize Random object
70            Random objRandom = new Random();
71
72            // generate a random circle diameter
73            int intDiameter = objRandom.Next( 5, 200 );
74
75            // draw a blue circle
76            objGraphic.DrawEllipse( new Pen( Color.Blue ), e.X, e.Y,
77               intDiameter, intDiameter );
78
79         } // end method FrmCirclePainter_MouseDown
80
81      } // end class FrmCirclePainter
82   }
```

21.13 ***(Advanced Circle Painter Application)*** In this exercise, you will enhance the application you created in Exercise 21.12. The advanced **Circle Painter** application should draw blue circles with a randomly generated diameter when the user presses the left mouse button. When the user presses the right mouse button, the application should draw a red circle with a randomly generated diameter (Fig. 21.30).

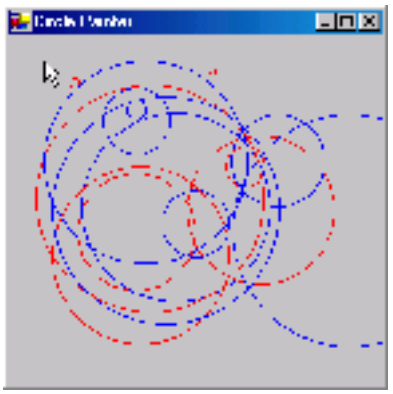


Figure 21.30 Advanced **Circle Painter** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial21\Exercises\AdvancedCirclePainter` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CirclePainter.sln` file in the `AdvancedCirclePainter` directory to open the application.

c) ***Drawing the appropriate circle.*** Use the `Button` property of `MouseEventArgs` reference `e` to determine which mouse button was pressed. They, call the `DrawEllipse` method on a reference to a `Graphics` object to draw a blue circle on the `Form` if the left mouse button was clicked or a red circle if the right mouse button was clicked.

d) ***Running the application.*** Select **Debug > Start** to run your application. Draw several blue circles of different sizes using the left mouse button, then draw several red circles of different sizes using the right mouse button.

e) ***Closing the application.*** Close your running application by clicking its close box.

f) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 21.13 Solution
// CirclePainter.cs (Advanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CirclePainter
{
   /// <summary>
   /// Summary description for FrmCirclePainter.
   /// </summary>
   public class FrmCirclePainter : System.Windows.Forms.Form
   {
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCirclePainter()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
```

```
59            Application.Run( new FrmCirclePainter() );
60         }
61
62         // handles Form's MouseDown event
63         private void FrmCirclePainter_MouseDown(
64            object sender, System.Windows.Forms.MouseEventArgs e )
65         {
66            // initialize Graphics object
67            Graphics objGraphic = CreateGraphics();
68
69            // initialize Random object
70            Random objRandom = new Random();
71
72            // generate a random circle diameter
73            int intDiameter = objRandom.Next( 5, 200 );
74
75            // left mouse button pressed
76            if ( e.Button == MouseButtons.Left )
77            {
78               // draw a blue circle if left mouse button pressed
79               objGraphic.DrawEllipse( new Pen( Color.Blue ), e.X, e.Y,
80                  intDiameter, intDiameter );
81            }
82            // right mouse button pressed
83            else if ( e.Button == MouseButtons.Right )
84            {
85               // draw a red circle if right mouse button pressed
86               objGraphic.DrawEllipse( new Pen( Color.Red ), e.X, e.Y,
87                  intDiameter, intDiameter );
88            }
89
90         } // end method FrmCirclePainter_MouseDown
91
92      } // end class FrmCirclePainter
93   }
```

What does this code do?

21.14 Consider the example in Fig. 21.27. Suppose we change the `MouseMove` event handler to the code below. What happens when the user moves the mouse? Assume that the `lblDisplay Label` has been placed on the `Form`.

```
1  private void FrmPainter_MouseMove(
2      object sender, System.Windows.Forms.MouseEventArgs e )
3  {
4     lblDisplay.Text = "I'm at " + e.X + ", " + e.Y + ".";
5
6  } // end method FrmPainter_MouseMove
```

Answer: The `Label` continuously displays the mouse pointer's current position on the `Form`. The complete code reads:

```
1  // Exercise 21.14 Solution
2  // Painter.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
```

```
using System.Windows.Forms;
using System.Data;

namespace MousePointer
{
   /// <summary>
   /// Summary description for FrmPainter.
   /// </summary>
   public class FrmPainter : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblPosition;
      private System.Windows.Forms.Label lblDisplay;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmPainter()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmPainter() );
      }

      // displays the mouse's current x and y coordinates
      private void FrmPainter_MouseMove(
         object sender, System.Windows.Forms.MouseEventArgs e )
      {
```

```
68          lblDisplay.Text = "I'm at " + e.X + ", " + e.Y + ".";
69
70       } // end method FrmPainter_MouseMove
71
72    } // end class FrmPainter
73 }
```

What's wrong with this code?

21.15 The following code should draw a `BlueViolet` circle of diameter `4` that corresponds to the movement of the mouse. Find the error(s) in the following code:

```
1  private void FrmPainter_MouseMove(
2      object sender, System.Windows.Forms.MouseEventArgs e )
3  {
4     if ( m_blnshouldPaint == true )
5     {
6        Graphics objGraphic = Graphics();
7
8        objGraphic.FillEllipse = (
9           new SolidBrush( Color.BlueViolet ), e.Y, e.X, 5, 4 );
10    }
11
12 } // end method FrmPainter_MouseMove
```

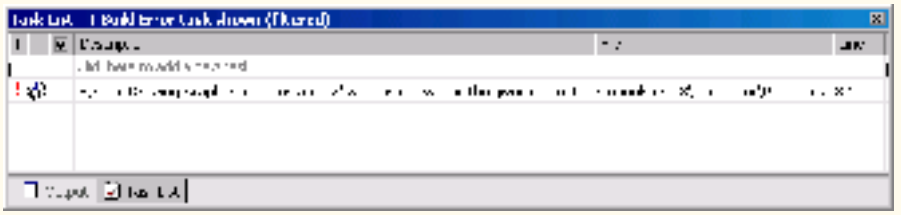
Answer: The position arguments in the `FillEllipse` method have been transposed. Use method `CreateGraphics` to initialize a `Graphics` object. A circle's height and width must be equal, so the fourth argument passed to method `FillEllipse` should be `4`. There should be no assignment operator between the word `FillEllipse` and the parenthesis. The complete incorrect code reads:

```
1  // Exercise 21.15 Solution
2  // Painter.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Painter
12 {
13    /// <summary>
```

```
14    /// Summary description for FrmPainter.
15    /// </summary>
16    public class FrmPainter : System.Windows.Forms.Form
17    {
18       /// <summary>
19       /// Required designer variable.
20       /// </summary>
21       private System.ComponentModel.Container components = null;
22
23       private bool m_blnShouldPaint = true;
24
25       public FrmPainter()
26       {
27          //
28          // Required for Windows Form Designer support
29          //
30          InitializeComponent();
31
32          //
33          // TODO: Add any constructor code after InitializeComponent
34          // call
35          //
36       }
37
38       /// <summary>
39       /// Clean up any resources being used.
40       /// </summary>
41       protected override void Dispose( bool disposing )
42       {
43          if( disposing )
44          {
45             if (components != null)
46             {
47                components.Dispose();
48             }
49          }
50          base.Dispose( disposing );
51       }
52
53       // Windows Form Designer generated code
54
55       /// <summary>
56       /// The main entry point for the application.
57       /// </summary>
58       [STAThread]
59       static void Main()
60       {
61          Application.Run(new FrmPainter());
62       }
63
64       // draws a violet ellipse wherever the mouse moves
65       private void FrmPainter_MouseMove(
66          object sender, System.Windows.Forms.MouseEventArgs e )
67       {
68          if ( m_blnshouldPaint == true )
69          {
70             Graphics objGraphic = Graphics();
71
72             objGraphic.FillEllipse = (
73                new SolidBrush( Color.BlueViolet ), e.Y, e.X, 5, 4 );
```

Remove the = operator, `e.X` and `e.Y` should be swapped, width and height should be equal, use `CreateGraphics` to ininitialize a `Graphics` object

```
74            }
75
76         } // end method FrmPainter_MouseMove
77
78      } // end class FrmPainter
79   }
```



Answer: The complete corrected code should read:

```
1   // Exercise 21.15 Solution
2   // Painter.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace Painter
12  {
13     /// <summary>
14     /// Summary description for FrmPainter.
15     /// </summary>
16     public class FrmPainter : System.Windows.Forms.Form
17     {
18        /// <summary>
19        /// Required designer variable.
20        /// </summary>
21        private System.ComponentModel.Container components = null;
22
23        private bool m_blnShouldPaint = true;
24
25        public FrmPainter()
26        {
27           //
28           // Required for Windows Form Designer support
29           //
30           InitializeComponent();
31
32           //
33           // TODO: Add any constructor code after InitializeComponent
34           // call
35           //
36        }
37
38        /// <summary>
39        /// Clean up any resources being used.
40        /// </summary>
```

```
41         protected override void Dispose( bool disposing )
42         {
43            if( disposing )
44            {
45               if (components != null)
46               {
47                  components.Dispose();
48               }
49            }
50            base.Dispose( disposing );
51         }
52
53         // Windows Form Designer generated code
54
55         /// <summary>
56         /// The main entry point for the application.
57         /// </summary>
58         [STAThread]
59         static void Main()
60         {
61            Application.Run( new FrmPainter() );
62         }
63
64         // draws a violet ellipse wherever the mouse moves
65         private void FrmPainter_MouseMove(
66            object sender, System.Windows.Forms.MouseEventArgs e )
67         {
68            if ( m_blnshouldPaint == true )
69            {
70               Graphics objGraphic = CreateGraphics();
71
72               objGraphic.FillEllipse(
73                  new SolidBrush( Color.BlueViolet ), e.X, e.Y, 4, 4 );
74            }
75
76         } // end method FrmPainter_MouseMove
77
78      } // end class FrmPainter
79   }
```

Programming Challenge ▶ **21.16** ***(Advanced Painter Application)*** Extend the `Painter` application to enable a user to change the size and color of the circles drawn (Fig. 21.31).

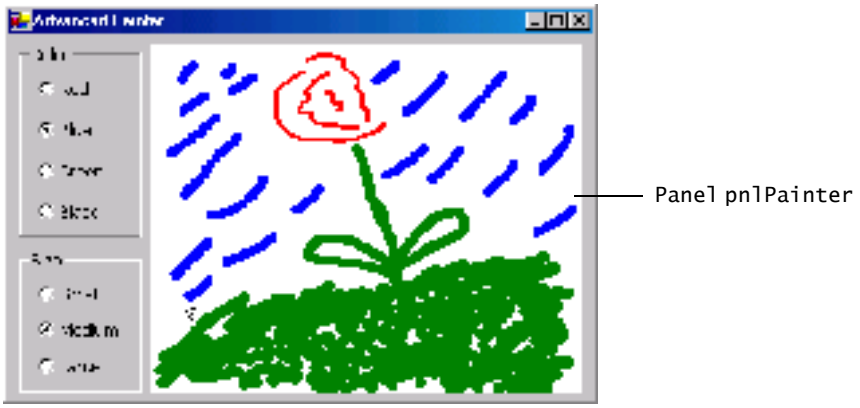


Figure 21.31 Advanced Painter application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial21\Exercises\AdvancedPainter` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AdvancedPainter.sln` in the `AdvancedPainter` directory to open the application.

c) ***Understanding the provided instance variables.*** The template already provides you with four instance variables. The `m_objBrushColor` variable is a `Color` value that specifies the color of the brush used in the **Advanced Painter** application. The `m_blnShouldPaint` and `m_blnShouldErase` variables perform the same functions as in this tutorial's **Painter** application. The `m_intDiameter` variable stores the diameter of the circle to be drawn.

d) ***Declaring an enumeration to store the circle diameter sizes.*** Declare an enumeration `Sizes` to store the possible values of `m_intDiameter`. Set the `SMALL` constant to `4`, `MEDIUM` to `8` and `LARGE` to `10`.

e) ***Adding event handlers for the Color `RadioButtons`.*** The **Color** `RadioButtons`' event handlers should set `m_objBrushColor` to their specified colors (`Color.Red`, `Color.Blue`, `Color.Green` or `Color.Black`).

f) ***Adding event handlers for the Size `RadioButtons`.*** The **Size** `RadioButtons`' event handlers should set `m_intDiameter` to `Sizes.SMALL` (for the **Small** `RadioButton`), `Sizes.MEDIUM` (for the **Medium** `RadioButton`) or `Sizes.LARGE` (for the **Large** `RadioButton`).

g) ***Adding a mouse event handler to a `Panel`.*** To associate mouse events with the `Panel`, select `pnlPainter` from the **Class Name** `ComboBox`. Then, select the appropriate mouse event from the **Method Name** `ComboBox`.

h) ***Coding the `MouseDown` and `MouseUp` event handlers.*** The `MouseUp` and `MouseDown` event handlers behave exactly as they do in the **Painter** application.

i) ***Coding the `MouseMove` event handler.*** The `MouseMove` event handler behaves as the one in **Painter** application does. The color of the brush that draws the circle when `m_blnShouldPaint` is `true` is specified by `m_objBrushColor`. The eraser color is specified by the `Panel`'s `BackColor` property.

j) ***Running the application.*** Select **Debug > Start** to run your application. Start drawing on the `Panel` using different brush sizes and colors. Use the right mouse button to erase part of your drawing.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 21.16 Solution
2  // AdvancedPainter.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
```

```
 7  using System.ComponentModel;
 8  using System.Windows.Forms;
 9  using System.Data;
10
11  namespace AdvancedPainter
12  {
13     /// <summary>
14     /// Summary description for FrmAdvancedPainter.
15     /// </summary>
16     public class FrmAdvancedPainter : System.Windows.Forms.Form
17     {
18        // GroupBox containing RadioButtons to choose color
19        private System.Windows.Forms.GroupBox grpColor;
20        private System.Windows.Forms.RadioButton radBlack;
21        private System.Windows.Forms.RadioButton radGreen;
22        private System.Windows.Forms.RadioButton radBlue;
23        private System.Windows.Forms.RadioButton radRed;
24
25        // GroupBox containing RadioButtons to choose size
26        private System.Windows.Forms.GroupBox grpSize;
27        private System.Windows.Forms.RadioButton radLarge;
28        private System.Windows.Forms.RadioButton radMedium;
29        private System.Windows.Forms.RadioButton radSmall;
30
31        // Panel for drawing area
32        private System.Windows.Forms.Panel pnlPainter;
33
34        /// <summary>
35        /// Required designer variable.
36        /// </summary>
37        private System.ComponentModel.Container components = null;
38
39        // create Color object and initialize to Black
40        private Color m_clrBrushColor = Color.Black;
41
42        // specify whether application should paint
43        private bool m_blnShouldPaint = false;
44
45        // specify whether application should erase
46        private bool m_blnShouldErase = false;
47
48        // diameter of MouseDown circle (initially set to small)
49        private int m_intDiameter = 4;
50
51        // size constants for diameter of MouseDown circle
52        private enum Sizes
53        {
54           SMALL = 4,
55           MEDIUM = 8,
56           LARGE = 10,
57        };
58
59        public FrmAdvancedPainter()
60        {
61           //
62           // Required for Windows Form Designer support
63           //
64           InitializeComponent();
65
66           //
```

```
67            // TODO: Add any constructor code after InitializeComponent
68            // call
69            //
70         }
71
72         /// <summary>
73         /// Clean up any resources being used.
74         /// </summary>
75         protected override void Dispose( bool disposing )
76         {
77            if( disposing )
78            {
79               if (components != null)
80               {
81                  components.Dispose();
82               }
83            }
84            base.Dispose( disposing );
85         }
86
87         // Windows Form Designer generated code
88
89         /// <summary>
90         /// The main entry point for the application.
91         /// </summary>
92         [STAThread]
93         static void Main()
94         {
95            Application.Run( new FrmAdvancedPainter() );
96         }
97
98         // handles radRed's CheckedChanged event
99         private void radRed_CheckedChanged(
100           object sender, System.EventArgs e )
101        {
102           // set brush color to red
103           if ( radRed.Checked == true )
104           {
105              m_clrBrushColor = Color.Red;
106           }
107
108        } // end method radRed_CheckedChanged
109
110        // handles radBlue's CheckedChanged event
111        private void radBlue_CheckedChanged(
112           object sender, System.EventArgs e )
113        {
114           // set brush color to blue
115           if ( radBlue.Checked == true )
116           {
117              m_clrBrushColor = Color.Blue;
118           }
119
120        } // end method radBlue_CheckedChanged
121
122        // handles radGreen's CheckedChanged event
123        private void radGreen_CheckedChanged(
124           object sender, System.EventArgs e )
125        {
126           // set brush color to green
```

```
127         if ( radGreen.Checked == true )
128         {
129            m_clrBrushColor = Color.Green;
130         }
131
132      } // end method radGreen_CheckedChanged
133
134      // handles radBlack's CheckedChanged event
135      private void radBlack_CheckedChanged(
136         object sender, System.EventArgs e )
137      {
138         // set brush color to black
139         if ( radBlack.Checked == true )
140         {
141            m_clrBrushColor = Color.Black;
142         }
143
144      } // end method radBlack_CheckedChanged
145
146      // handles radSmall's CheckedChanged event
147      private void radSmall_CheckedChanged(
148         object sender, System.EventArgs e )
149      {
150         // draw small circles
151         if ( radSmall.Checked == true )
152         {
153            m_intDiameter = ( int ) Sizes.SMALL;
154         }
155
156      } // end method radSmall_CheckedChanged
157
158      // handles radMedium's CheckedChanged event
159      private void radMedium_CheckedChanged(
160         object sender, System.EventArgs e )
161      {
162         // draw medium circles
163         if ( radMedium.Checked == true )
164         {
165            m_intDiameter = ( int ) Sizes.MEDIUM;
166         }
167
168      } // end method radMedium_CheckedChanged
169
170      // handles radLarge's CheckedChanged event
171      private void radLarge_CheckedChanged(
172         object sender, System.EventArgs e )
173      {
174         // draw large circles
175         if ( radLarge.Checked == true )
176         {
177            m_intDiameter = ( int ) Sizes.LARGE;
178         }
179
180      } // end method radLarge_CheckedChanged
181
182      // draw when mouse button pressed down
183      private void pnlPainter_MouseDown(
184         object sender, System.Windows.Forms.MouseEventArgs e )
185      {
186         // draw if left mouse button held down
```

```
187         if ( e.Button == MouseButtons.Left )
188         {
189            m_blnShouldPaint = true;
190         }
191         // erase if right mouse button held down
192         else if ( e.Button == MouseButtons.Right )
193         {
194            m_blnShouldErase = true;
195         }
196
197      } // end method pnlPainter_MouseDown
198
199      private void pnlPainter_MouseUp(
200         object sender, System.Windows.Forms.MouseEventArgs e )
201      {
202         m_blnShouldPaint = false; // do not draw
203         m_blnShouldErase = false; // do not erase
204
205      } // end method pnlPainter_MouseUp
206
207      private void pnlPainter_MouseMove(
208         object sender, System.Windows.Forms.MouseEventArgs e )
209      {
210         // create Graphics object for the Panel
211         Graphics objGraphic = pnlPainter.CreateGraphics();
212
213         // draw circles with specified brush color and size
214         if ( m_blnShouldPaint == true )
215         {
216            objGraphic.FillEllipse(
217               new SolidBrush( m_clrBrushColor ),
218               e.X, e.Y, m_intDiameter, m_intDiameter );
219         }
220         // draw circles with Panel's background color and
221         // specified size
222         else if ( m_blnShouldErase == true )
223         {
224            objGraphic.FillEllipse(
225               new SolidBrush( pnlPainter.BackColor ),
226               e.X, e.Y, m_intDiameter, m_intDiameter );
227         }
228
229      } // end method pnlPainter_MouseMove
230
231   } // end class FrmAdvancedPainter
232 }
```

TUTORIAL 22

Typing Application

Introducing Keyboard Events, Menus and Dialogs

Solutions

Instructor's Manual Exercise Solutions Tutorial 22

MULTIPLE-CHOICE QUESTIONS

22.1 When creating a menu, typing a _____ in front of a menu item name will create an access shortcut for that item.

a) `&` b) `!`
c) `$` d) `#`

22.2 *Alt*, *Shift* and *Control* are _____ keys.

a) modifier b) ASCII
c) function d) special

22.3 `KeyChar` is a property of _____.

a) `KeyEventArgs` b) `Key`
c) `KeyArgs` d) `KeyPressEventArgs`

22.4 Typing a hyphen (`-`) as a menu item's `Text` property will create a(n) _____.

a) separator bar b) access shortcut
c) new submenu d) keyboard shortcut

22.5 A _____ provides a group of related commands for Windows applications.

a) separator bar b) hot key
c) menu d) margin indicator bar

22.6 The _____ enumeration specifies key codes and modifiers.

a) `Keyboard` b) `Key`
c) `KeyboardTypes` d) `Keys`

22.7 The _____ event is raised when a key is pressed by the user.

a) `KeyPress` b) `KeyHeld`
c) `KeyDown` d) Both a and c.

22.8 Which of the following is not a keyboard event?

a) `KeyPress` b) `KeyDown`
c) `KeyUp` d) `KeyClicked`

22.9 Which of the following is not a structure?

a) `Char` b) `Color`
c) `String` d) `DateTime`

22.10 The _____ type allows you to determine which `Button` the user clicked to exit a dialog.

a) `DialogButtons` b) `DialogResult`
c) `Buttons` d) `ButtonResult`

Answers: 22.1) a. 22.2) a. 22.3) d. 22.4) a. 22.5) c. 22.6) d. 22.7) d. 22.8) d. 22.9) c. 22.10) b.

EXERCISES

22.11 (***Enhanced Inventory Application with Keyboard Events***) Enhance the **Inventory** application that you developed in Tutorial 4 to prevent the user from entering input that is not a number. Use keyboard events to allow the user to press the number keys, the left and right arrows and the *Backspace* keys. If a key other than these is pressed, display a `MessageBox` instructing the user to enter a number (Fig. 22.31).

Figure 22.31 Enhanced **Inventory** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial22\Exercises\KeyEventInventory` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Inventory.sln` in the `KeyEventInventory` directory to open the application.

c) ***Adding the `KeyDown` event handler for the first `TextBox`.*** Add an empty `KeyDown` event handler for the **Cartons per shipment:** `TextBox`.

d) ***Adding a `switch` statement.*** Add a `switch` statement to the `KeyDown` event handler that determines whether a number key, a left or right arrow or the *Backspace* key was pressed.

e) ***Adding the `default` statement.*** Add a `default` statement that will determine whether a key other than a valid one for this application was pressed. If an invalid key was pressed, display a `MessageBox` that instructs the user to enter a number.

f) ***Adding the `KeyDown` event handler for the second `TextBox`.*** Repeat *Steps c–e*, only this time create a `KeyDown` event handler for the **Items per carton:** `TextBox`. This event handler should perform the same functionality as the one for the **Cartons per shipment:** `TextBox`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Try entering letters or pressing the up and down arrow keys in the `TextBoxes`. A `MessageBox` should be displayed. Enter valid input and click the **Calculate Total** `Button`. Verify that the correct output is displayed.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 22.11 Solution
// Inventory.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Inventory
{
   /// <summary>
   /// Summary description for FrmInventory.
   /// </summary>
   public class FrmInventory : System.Windows.Forms.Form
   {
      // Label and TextBox display the number of cartons per shipment
      private System.Windows.Forms.Label lblCartons;
      private System.Windows.Forms.TextBox txtCartons;

      // Label and TextBox display the number of items per carton
      private System.Windows.Forms.Label lblItems;
      private System.Windows.Forms.TextBox txtItems;

      // Labels display the total value
      private System.Windows.Forms.Label lblTotal;
```

```
28          private System.Windows.Forms.Label lblTotalResult;
29
30          // Button calculates total value
31          private System.Windows.Forms.Button btnCalculate;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          public FrmInventory()
39          {
40             //
41             // Required for Windows Form Designer support
42             //
43             InitializeComponent();
44
45             //
46             // TODO: Add any constructor code after InitializeComponent
47             // call
48             //
49          }
50
51          /// <summary>
52          /// Clean up any resources being used.
53          /// </summary>
54          protected override void Dispose( bool disposing )
55          {
56             if( disposing )
57             {
58                if (components != null)
59                {
60                   components.Dispose();
61                }
62             }
63             base.Dispose( disposing );
64          }
65
66          // Windows Form Designer generated code
67
68          /// <summary>
69          /// The main entry point for the application.
70          /// </summary>
71          [STAThread]
72          static void Main()
73          {
74             Application.Run( new FrmInventory() );
75          }
76
77          // handles Click event
78          private void btnCalculate_Click(
79             object sender, System.EventArgs e )
80          {
81             // multiply values input and display result in Label
82             lblTotalResult.Text = Convert.ToString(
83                Int32.Parse( txtCartons.Text ) *
84                Int32.Parse( txtItems.Text ) );
85
86          } // end method btnCalculate_Click
87
```

```
88          // handles txtCartons' KeyDown event
89          private void txtCartons_KeyDown(
90             object sender, System.Windows.Forms.KeyEventArgs e )
91          {
92             DialogResult result; // store result of MessageBox
93
94             switch ( e.KeyData )
95             {
96                // numbers
97                case Keys.D0:
98                case Keys.D1:
99                case Keys.D2:
100               case Keys.D3:
101               case Keys.D4:
102               case Keys.D5:
103               case Keys.D6:
104               case Keys.D7:
105               case Keys.D8:
106               case Keys.D9:
107
108               case Keys.Back: // backspace
109               case Keys.Enter: // enter
110               case Keys.Left: case Keys.Right: // arrows
111
112                  break;
113
114               default: // all other keys
115
116                  // show MessageBox
117                  result = MessageBox.Show( "Enter numbers only",
118                     "Invalid Input", MessageBoxButtons.OK,
119                     MessageBoxIcon.Exclamation );
120
121                  // clear TextBox if invalid input entered
122                  if ( result == DialogResult.OK )
123                  {
124                     txtCartons.Clear();
125                  }
126
127                  break;
128
129            } // end switch
130
131         } // end method txtCartons_KeyDown
132
133         // handles txtItems' KeyDown event
134         private void txtItems_KeyDown(
135            object sender, System.Windows.Forms.KeyEventArgs e )
136         {
137            DialogResult result; // store result of MessageBox
138
139            switch ( e.KeyData )
140            {
141               // numbers
142               case Keys.D0:
143               case Keys.D1:
144               case Keys.D2:
145               case Keys.D3:
146               case Keys.D4:
147               case Keys.D5:
```

```
148         case Keys.D6:
149         case Keys.D7:
150         case Keys.D8:
151         case Keys.D9:
152
153         case Keys.Back: // backspace
154         case Keys.Enter: // enter
155         case Keys.Left: case Keys.Right: // arrows
156
157            break;
158
159         default: // all other keys
160
161            // show MessageBox
162            result = MessageBox.Show( "Enter numbers only",
163               "Invalid Input", MessageBoxButtons.OK,
164               MessageBoxIcon.Exclamation );
165
166            // clear TextBox if invalid input entered
167            if ( result == DialogResult.OK )
168            {
169               txtCartons.Clear();
170            }
171
172            break;
173
174      } // end switch
175
176   } // end method txtItems_KeyDown
177
178} // end class FrmInventory
179 }
```

22.12 (***Bouncing Ball Application***) Write an application that allows the user to play a game, the goal of which is to prevent a bouncing ball from falling off the bottom of the `Form`. When the user presses the *S* key, a blue ball will bounce off the top, left and right sides (the "walls") of the `Form`. There should be a horizontal bar on the bottom of the `Form`, which serves as a paddle to prevent the ball from hitting the bottom of the `Form`. (The ball can bounce off the paddle, but not the bottom of the `Form`.) The user can move the paddle using the left and right arrow keys. If the ball hits the paddle, the ball should bounce up, and the game should continue. If the ball hits the bottom of the `Form`, the game should end. The paddle's width should decrease every 20 seconds to make the game more challenging. The GUI is provided for you (Fig. 22.32).

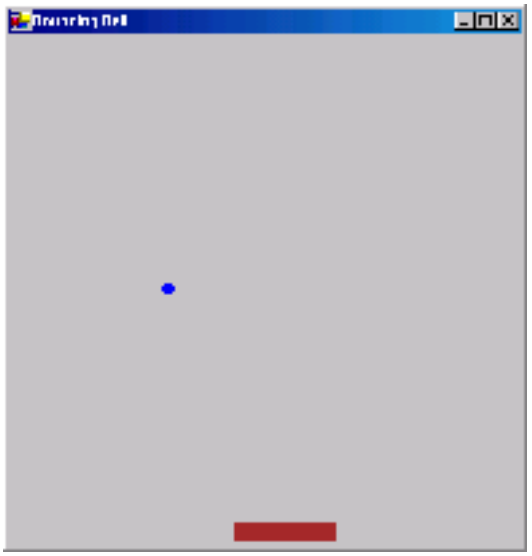


Figure 22.32 **Bouncing Ball** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial22\Exercises\BouncingBall` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `BouncingBall.sln` in the `BouncingBall` directory to open the application.

c) ***Creating the KeyDown event handler.*** Insert a `KeyDown` event handler for the `Form`.

d) ***Writing code to start the game.*** Write an `if` statement in the `KeyDown` event handler that tests whether the user presses the *S* key. You can use the `KeyDown` event handler for the *S* key in this case, because you do not care whether the user presses an uppercase *S* or a lowercase *S*. If the user presses the *S* key, start the two `Timers` that are provided in the template.

e) ***Inserting code to move the paddle left.*** Write an `if` statement that tests if the user pressed the left-arrow key and if the paddle's horizontal position is greater than zero. If the paddle's horizontal position equals zero, the left edge of the paddle is touching the left wall and the paddle should not be allowed to move farther to the left. If both the conditions in the `if` statement are true, decrease the paddle's *x*-coordinate by `10`.

f) ***Inserting code to move the paddle right.*** Write an `if` statement that tests if the user pressed the right-arrow key and whether the paddle's *x*-coordinate is less than the width of the `Form` minus the width of the paddle. If the paddle's *x*-coordinate equals the `Form`'s width minus the width of the paddle, the paddle's right edge is touching the right wall and the paddle should not be allowed to move farther to the right. If both the conditions in the `if` statement are true, increase the paddle's *x*-coordinate by `10`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Press the *S* key to begin the game and use the paddle to keep the bouncing ball from dropping off the `Form`. Continue doing this until 20 seconds have passed, and verify that the paddle is decreased in size at that time.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 22.12 Solution
2  // BouncingBall.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
```

```
using System.Windows.Forms;
using System.Data;

namespace BouncingBall
{
   /// <summary>
   /// Summary description for FrmBouncingBall.
   /// </summary>
   public class FrmBouncingBall : System.Windows.Forms.Form
   {
      // Timer to control the speed of the ball
      private System.Windows.Forms.Timer tmrMoveBall;

      // Timer to control the size of the paddle
      private System.Windows.Forms.Timer tmrShrinkSlider;

      private System.ComponentModel.IContainer components;

      private int intX; // ball's x-coordinate
      private int intY; // ball's y-coordinate
      private int intRectangleX; // paddle's x-coordinate
      private int intRectangleWidth; // paddle's width

      private int intDeltaX; // ball's x rate of change
      private int intDeltaY; // ball's y rate of change

      private bool xLeft; //  tests if ball can move left
      private bool yUp; // tests if ball can move up

      private const int intMAX_X = 400; // x boundary
      private const int intMAX_Y = 400; // y boundary

      // object to generate random numbers
      private Random objRandom = new Random();

      public FrmBouncingBall()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
```

```
68             base.Dispose( disposing );
69          }
70
71          // Windows Form Designer generated code
72
73          /// <summary>
74          /// The main entry point for the application.
75          /// </summary>
76          [STAThread]
77          static void Main()
78          {
79             Application.Run( new FrmBouncingBall() );
80          }
81
82          // handles Form's Load event
83          private void FrmBouncingBall_Load(
84             object sender, System.EventArgs e )
85          {
86             intX = objRandom.Next( 100, 301 ); // ball's initial x
87             intY = objRandom.Next( 100, 301 ); // ball's initial y
88             intRectangleX = 175; // rectangle's initial x position
89             intRectangleWidth = 80; // rectangle's initial width
90
91             xLeft = false; // ball can move left
92             yUp = false; // ball can move up
93             intDeltaX = 2; // move ball 2 positions right
94             intDeltaY = 2; // move ball 2 positions down
95
96          } // end method FrmBouncingBall_Load
97
98          // method to draw the ball and paddle
99          protected override void OnPaint( PaintEventArgs paintEvent )
100         {
101            // create graphics object
102            Graphics objGraphic = CreateGraphics();
103
104            // create new brush
105            SolidBrush objBrush = new SolidBrush( Color.Blue );
106
107            // draw ball
108            objGraphic.FillEllipse( objBrush, intX, intY, 10, 10 );
109
110            // set color for, and draw paddle
111            objBrush.Color = Color.Brown;
112            objGraphic.FillRectangle(
113               objBrush, intRectangleX, 380, intRectangleWidth, 15 );
114
115         } // end method OnPaint
116
117         // move the ball every tick
118         private void tmrMoveBall_Tick(
119            object sender, System.EventArgs e )
120         {
121            // determine new x position
122            if ( xLeft == true )
123            {
124               intX += intDeltaX;
125            }
126            else
127            {
```

```
128            intX -= intDeltaX;
129         }
130
131         // determine new y position
132         if ( yUp )
133         {
134            intY += intDeltaY;
135         }
136         else
137         {
138            intY -= intDeltaY;
139         }
140
141         if ( intY <= 0 )
142         {
143            yUp = true;
144            intDeltaY = objRandom.Next( 2, 6 );
145         }
146         else if ( intY >= 370 && intX >= intRectangleX
147            && intX <= ( intRectangleX + intRectangleWidth ) )
148         {
149            yUp = false;
150            intDeltaY = objRandom.Next( 2, 6 );
151         }
152         else if ( intY >= 410 ) // end game if ball hits floor
153         {
154            tmrMoveBall.Enabled = false;
155            tmrShrinkSlider.Enabled = false;
156            MessageBox.Show( "Game Over" );
157         }
158
159         if ( intX <= 0 )
160         {
161            xLeft = true;
162            intDeltaX = objRandom.Next( 2, 6 );
163         }
164         else if ( intX >= intMAX_X - 10 )
165         {
166            xLeft = false;
167            intDeltaX = objRandom.Next( 2, 6 );
168         }
169
170         Invalidate(); // refresh Form
171
172      } // end method tmrMoveBall_Tick
173
174      // shrinks the paddle every 20 seconds
175      private void tmrShrinkSlider_Tick(
176         object sender, System.EventArgs e )
177      {
178         // shrink paddle if paddle greater than twice ball's width
179         if ( intRectangleWidth >= 20 )
180         {
181            intRectangleWidth = intRectangleWidth / 2;
182         }
183
184      } // end method tmrShrinkSlider_Tick
185
186      // handles Form's KeyDown event
187      private void FrmBouncingBall_KeyDown(
```

```
188         object sender, System.Windows.Forms.KeyEventArgs e )
189      {
190         // start game if user presses S key
191         if ( e.KeyCode == Keys.S )
192         {
193            tmrMoveBall.Enabled = true; // start tmrMoveBall
194            tmrShrinkSlider.Enabled = true; // start tmrShrinkpaddle
195         }
196
197         if ( e.KeyCode == Keys.Left && intRectangleX >= 0 )
198         {
199            intRectangleX -= 10; // move paddle to the left
200         }
201
202         if ( e.KeyCode == Keys.Right &&
203            intRectangleX <= intMAX_X - intRectangleWidth )
204         {
205            intRectangleX += 10; // move paddle to the right
206         }
207
208      } // end method FrmBouncingBall_KeyDown
209
210   } // end class FrmBouncingBall
211 }
```

22.13 (***Modified Painter Application***) Modify the **Painter** application that you developed in Tutorial 21 to include menus that allow the user to select the size and color of the painted ellipses and the color of the `Form` (Fig. 22.33). (The menus replace the `RadioButtons`.) Also, add a multiline `TextBox` to allow the user to type text to accompany the painting. The user should be able to use menus to select the font style and color of the text and the background color of the `TextBox`.

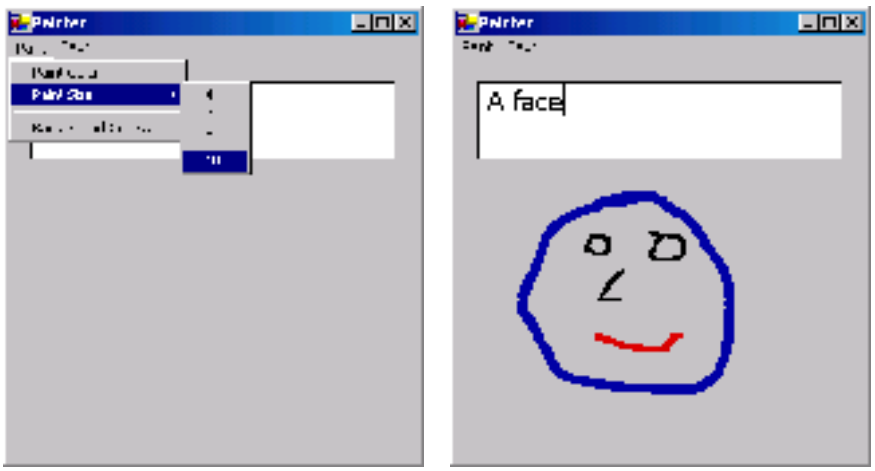


Figure 22.33 Modified **Painter** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial22\Exercises\ModifiedPainter` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Painter.sln` in the `ModifiedPainter` directory to open the application.

c) ***Creating the menus.*** Create two menus. The first one should be titled **Paint** and should contain a **Paint Color**... menu item, a **Paint Size** submenu that contains menu items **4**, **6**, **8** and **10**, a separator bar and a **Background Color**... menu item. The second menu should be titled **Text** and have **Text Color**... and **Font**... menu items, a

separator bar and a **TextBox Color**... menu item. Rearrange and comment the control declarations appropriately.

d) ***Changing the paint color.*** Add an event handler for the **Paint Color**... menu item. This event handler should display a **Color** dialog that allows the user to change the value stored in `m_paintColor`.

e) ***Changing the paint size.*** Add an event handler for each of the **Size** submenu's menu items. Each event handler should change the value stored in `m_intDiamter` to the value displayed on the menu (that is, clicking the **4** menu item will change the value of `m_intDiameter` to 4).

f) ***Changing the background color.*** Add an event handler for the **Background Color...** menu item. This event handler should display a **Color** dialog that allows the user to change the value stored in `m_backgroundColor` and also change the `BackColor` property of the `Form`. To change the background color of the `Form`, assign the value specifying the background color to `BackColor`. For instance, the statement `BackColor = Color.White;` changes the background color of the `Form` to white.

g) ***Changing the text color.*** Add an event handler for the **Text Color**... menu item. This event handler should display a **Color** dialog that allows the user to change the color of the text displayed in the `TextBox`.

h) ***Changing the text style.*** Add an event handler for the **Font...** menu item. This event handler should display a **Font** dialog that allows the user to change the style of the text displayed in the `TextBox`.

i) ***Changing the `TextBox`'s background color.*** Add an event handler for the **TextBox Color...** menu item. This event handler should display a **Color** dialog that allows the user to change the background color of the `TextBox`.

j) ***Running the application.*** Select **Debug > Start** to run your application. Use the menus to draw shapes of various colors and brush sizes. Enter text to describe your drawing. Use the other menu options to change the color of the `Form`, the `TextBox` and the text in the `TextBox`.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 22.13 Solution
2  // Painter.cs (Modified)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Painter
12 {
13    /// <summary>
14    /// Summary description for FrmPainter.
15    /// </summary>
16    public class FrmPainter : System.Windows.Forms.Form
17    {
18       // TextBox to input text to accompany drawing
19       private System.Windows.Forms.TextBox txtOutput;
20
21       // MainMenu to set options
22       private System.Windows.Forms.MainMenu mnuMainMenu;
23
24       // MenuItems to set brush color, brush size and
25       // background color
26       private System.Windows.Forms.MenuItem mnuitmPaint;
```

```
27          private System.Windows.Forms.MenuItem mnuitmColor;
28          private System.Windows.Forms.MenuItem mnuitmSize;
29          private System.Windows.Forms.MenuItem mnuitmFour;
30          private System.Windows.Forms.MenuItem mnuitmSix;
31          private System.Windows.Forms.MenuItem mnuitmEight;
32          private System.Windows.Forms.MenuItem mnuitmTen;
33          private System.Windows.Forms.MenuItem mnuitmPaintHorizBar;
34          private System.Windows.Forms.MenuItem mnuitmBackground;
35
36          // MenuItems to set text font, text color and text
37          // background color
38          private System.Windows.Forms.MenuItem mnuitmFont;
39          private System.Windows.Forms.MenuItem mnuitmText;
40          private System.Windows.Forms.MenuItem mnuitmTextColor;
41          private System.Windows.Forms.MenuItem mnuitmTextHorizBar;
42          private System.Windows.Forms.MenuItem mnuitmTextBoxColor;
43
44          /// <summary>
45          /// Required designer variable.
46          /// </summary>
47          private System.ComponentModel.Container components = null;
48
49          // specify whether moving the mouse should erase
50          private bool m_blnShouldErase = false;
51
52          // specify whether moving the mouse should draw
53          private bool m_blnShouldPaint = false;
54
55          // diameter of MouseDown circle
56          private int m_intDiameter;
57
58          private Color m_paintColor; // paint color
59          private Color m_backgroundColor; // background color
60
61          // declare Graphics object
62          private Graphics m_objGraphic;
63
64          public FrmPainter()
65          {
66             //
67             // Required for Windows Form Designer support
68             //
69             InitializeComponent();
70
71             //
72             // TODO: Add any constructor code after InitializeComponent
73             // call
74             //
75          }
76
77          /// <summary>
78          /// Clean up any resources being used.
79          /// </summary>
80          protected override void Dispose( bool disposing )
81          {
82             if( disposing )
83             {
84                if (components != null)
85                {
86                   components.Dispose();
```

```
87             }
88          }
89          base.Dispose( disposing );
90       }
91
92       // Windows Form Designer generated code
93
94       /// <summary>
95       /// The main entry point for the application.
96       /// </summary>
97       [STAThread]
98       static void Main()
99       {
100         Application.Run( new FrmPainter() );
101      }
102
103      // handles FrmPainter's Load event
104      private void FrmPainter_Load(
105         object sender, System.EventArgs e )
106      {
107         m_blnShouldErase = false; // should not be painting
108         m_blnShouldPaint = false; // should not be erasing
109         m_intDiameter = 8; // set paint size
110         m_paintColor = Color.BlueViolet; // set paint color
111         m_backgroundColor = FrmPainter.DefaultBackColor;
112
113         // create Graphics object
114         m_objGraphic = CreateGraphics();
115
116      } // end method FrmPainter_Load
117
118      // handles FrmPainter's MouseDown event
119      private void FrmPainter_MouseDown(
120         object sender, System.Windows.Forms.MouseEventArgs e )
121      {
122         // draw on Form if the left button is held down
123         if ( e.Button == MouseButtons.Left )
124         {
125            m_blnShouldPaint = true;
126         }
127         // erase blue-violet circles if right button is held down
128         else if ( e.Button == MouseButtons.Right )
129         {
130            m_blnShouldErase = true;
131         }
132
133      } // end method FrmPainter_MouseDown
134
135      private void FrmPainter_MouseUp(
136         object sender, System.Windows.Forms.MouseEventArgs e )
137      {
138         m_blnShouldPaint = false; // do not draw on the Form
139         m_blnShouldErase = false; // do not erase
140
141      } // end method FrmPainter_MouseUp
142
143      private void FrmPainter_MouseMove(
144         object sender, System.Windows.Forms.MouseEventArgs e )
145      {
146         // draw circle if left mouse button is pressed
```

```
147          if ( m_blnShouldPaint == true )
148          {
149             m_objGraphic.FillEllipse(
150                new SolidBrush( m_paintColor ),
151                e.X, e.Y, m_intDiameter, m_intDiameter );
152          }
153          // mouse pointer "erases" if right mouse button is pressed
154          else if ( m_blnShouldErase == true )
155          {
156             m_objGraphic.FillEllipse(
157                new SolidBrush( m_backgroundColor ),
158                e.X, e.Y, m_intDiameter, m_intDiameter );
159          }
160
161       } // end method FrmPainter_MouseMove
162
163       // handles Color MenuItem's Click event
164       private void mnuitmColor_Click(
165          object sender, System.EventArgs e )
166       {
167          ColorDialog dlgColorDialog = new ColorDialog();
168          DialogResult result; // stores Button clicked
169
170          dlgColorDialog.FullOpen = true; // show all colors
171          result = dlgColorDialog.ShowDialog();
172
173          // do nothing if user clicked dialog's Cancel Button
174          if ( result == DialogResult.Cancel )
175          {
176             return;
177          }
178
179          // assign new color to Paint object
180          m_paintColor = dlgColorDialog.Color;
181
182       } // end method mnuitmColor_Click
183
184       // handles Size Four MenuItem's Click event
185       private void mnuitmFour_Click(
186          object sender, System.EventArgs e )
187       {
188          m_intDiameter = 4; // set paint size to four
189
190       } // end method mnuitmFour_Click
191
192       // handles Size Six MenuItem's Click event
193       private void mnuitmSix_Click(
194          object sender, System.EventArgs e )
195       {
196          m_intDiameter = 6; // set paint size to six
197
198       } // end method mnuitmSix_Click
199
200       // handles Size Eight MenuItem's Click event
201       private void mnuitmEight_Click(
202          object sender, System.EventArgs e )
203       {
204          m_intDiameter = 8; // set paint size to eight
205
206       } // end method mnuitmEight_Click
```

```
207
208         // handles Size Ten MenuItem's Click event
209         private void mnuitmTen_Click(
210            object sender, System.EventArgs e )
211         {
212            m_intDiameter = 10; // set paint size to ten
213
214         } // end method mnuitmTen_Click
215
216         // handles Background MenuItem's Click event
217         private void mnuitmBackground_Click(
218            object sender, System.EventArgs e )
219         {
220            ColorDialog dlgColorDialog = new ColorDialog();
221            DialogResult result; // stores Button clicked
222
223            dlgColorDialog.FullOpen = true; // show all colors
224            result = dlgColorDialog.ShowDialog();
225
226            // do nothing if user clicked dialog's Cancel Button
227            if ( result == DialogResult.Cancel )
228            {
229               return;
230            }
231
232            // set "erase" color
233            m_backgroundColor = dlgColorDialog.Color;
234            BackColor = dlgColorDialog.Color; // set Form's color
235
236         } // end method mnuitmBackground_Click
237
238         // handles Text Color MenuItem's Click event
239         private void mnuitmTextColor_Click(
240            object sender, System.EventArgs e )
241         {
242            ColorDialog dlgColorDialog = new ColorDialog();
243            DialogResult result; // stores Button clicked
244
245            dlgColorDialog.FullOpen = true; // show all colors
246            result = dlgColorDialog.ShowDialog();
247
248            // do nothing if user clicked dialog's Cancel Button
249            if ( result == DialogResult.Cancel )
250            {
251               return;
252            }
253
254            // assign new color to text
255            txtOutput.ForeColor = dlgColorDialog.Color;
256
257         } // end method mnuitmTextColor_Click
258
259         // handles Font MenuItem's Click event
260         private void mnuitmFont_Click(
261            object sender, System.EventArgs e )
262         {
263            FontDialog dlgFontDialog = new FontDialog();
264            DialogResult result; // stores Button clicked
265
266            // show dialog and get result
```

```
267          result = dlgFontDialog.ShowDialog();
268
269          // do nothing if user clicked dialog's Cancel Button
270          if ( result == DialogResult.Cancel )
271          {
272             return;
273          }
274
275          // assign new font value to TextBox
276          txtOutput.Font = dlgFontDialog.Font;
277
278       } // end method mnuitmFont_Click
279
280       // handles TextBox Color MenuItem's Click event
281       private void mnuitmTextBoxColor_Click(
282          object sender, System.EventArgs e )
283       {
284          ColorDialog dlgColorDialog = new ColorDialog();
285          DialogResult result; // stores Button clicked
286
287          dlgColorDialog.FullOpen = true; // show all colors
288          result = dlgColorDialog.ShowDialog();
289
290          // do nothing if user clicked dialog's Cancel Button
291          if ( result == DialogResult.Cancel )
292          {
293             return;
294          }
295
296          // assign background color of TextBox
297          txtOutput.BackColor = dlgColorDialog.Color;
298
299       } // end method mnuitmTextBoxColor_Click
300
301    } // end class FrmPainter
302 }
```

What does this code do?

22.14 What is the result of the following code? Assume the `Form` contains a `MainMenu` control, with a `MenuItem` named `mnuitmColor`. Also assume the `Form` contains a `Label` called `lblMystery`.

```
1  private void mnuitmColor_Click( object sender, System.EventArgs e )
2  {
3     ColorDialog dlgColorDialog = new ColorDialog();
4     DialogResult result;
5
6     dlgColorDialog.FullOpen = true;
7
8     result = dlgColorDialog.ShowDialog();
9
10    if ( result == DialogResult.Cancel )
11    {
12       return;
13    }
14
15    lblMystery.BackColor = dlgColorDialog.Color;
16
17 } // end method mnuitmColor_Click
```

Answer: When this event handler executes, the **Color** dialog is displayed. If the user chooses a color, that color is assigned to the `BackColor` of `lblMystery`. If the dialog's **Cancel** `Button` is clicked, the dialog closes and this event handler terminates. The complete code reads:

```
// Exercise 22.14 Solution
// ColorChanger.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace ColorChanger
{
   /// <summary>
   /// Summary description for FrmColorChanger.
   /// </summary>
   public class FrmColorChanger : System.Windows.Forms.Form
   {
      private System.Windows.Forms.MainMenu mnuMainMenu;
      private System.Windows.Forms.MenuItem mnuitmForm;
      private System.Windows.Forms.MenuItem mnuitmColor;
      private System.Windows.Forms.Label lblMystery;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmColorChanger()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
```

```
58         /// The main entry point for the application.
59         /// </summary>
60         [STAThread]
61         static void Main()
62         {
63            Application.Run( new FrmColorChanger() );
64         }
65
66         // changes the Form's color
67         private void mnuitmColor_Click(
68            object sender, System.EventArgs e )
69         {
70            ColorDialog dlgColorDialog = new ColorDialog();
71            DialogResult result;
72
73            dlgColorDialog.FullOpen = true;
74
75            result = dlgColorDialog.ShowDialog();
76
77            if ( result == DialogResult.Cancel )
78            {
79               return;
80            }
81
82            lblMystery.BackColor = dlgColorDialog.Color;
83
84         } // end method mnuitmColor_Click
85
86      } // end class FrmColorChanger
87   }
```

What's wrong with this code?

22.15 This code should allow a user to pick a font from a **Font** and set the text in `txtDisplay` to that font. Find the error(s) in the following code, assuming that a `TextBox` named `txtDisplay` exists on a `Form`, along with a `MenuItem` named `mnuitmFont`.

```
1   private void mnuitmFont_Click(
2       object sender, System.EventArgs e )
3   {
4      FontDialog dlgFontDialog;
5
6      dlgFontDialog = new FontDialog();
7      dlgFontDialog.ShowDialog();
8      txtDisplay.Font = dlgFontDialog.Font;
9
10  } // end method mnuitmFont_Click
```

Answer: The code should check whether the use clicked the **Cancel** `Button` in the dialog. If the user clicked **Cancel**, the method should be terminated. If the user did not click **Cancel**, the text should be set to the selected style. The complete incorrect code reads:

```
1  // Exercise 22.15 Solution
2  // FontChange.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace FontChange
12 {
13    /// <summary>
14    /// Summary description for FrmFontChange.
15    /// </summary>
16    public class FrmFontChange : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.TextBox txtDisplay;
19       private System.Windows.Forms.Label lblSample;
20       private System.Windows.Forms.MainMenu mnuMain;
21       private System.Windows.Forms.MenuItem menuItem1;
22       private System.Windows.Forms.MenuItem mnuitmFont;
23       /// <summary>
24       /// Required designer variable.
25       /// </summary>
26       private System.ComponentModel.Container components = null;
27
28       public FrmFontChange()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34
35          //
36          // TODO: Add any constructor code after InitializeComponent
37          // call
38          //
39       }
40
41       /// <summary>
42       /// Clean up any resources being used.
43       /// </summary>
44       protected override void Dispose( bool disposing )
45       {
46          if( disposing )
47          {
48             if (components != null)
49             {
50                components.Dispose();
51             }
52          }
53          base.Dispose( disposing );
54       }
55
56       // Windows Form Designer generated code
57
58       /// <summary>
59       /// The main entry point for the application.
60       /// </summary>
```

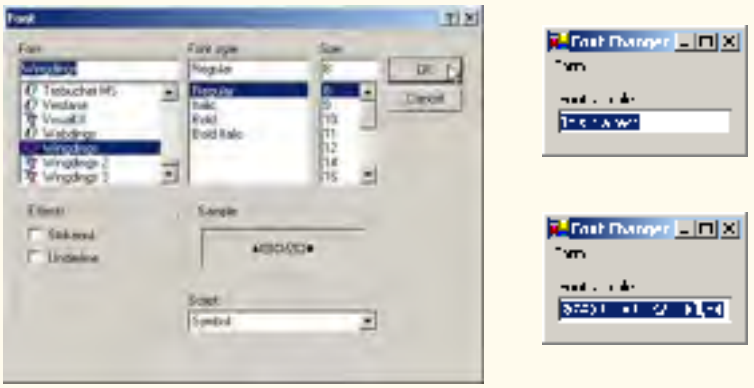
```
61         [STAThread]
62         static void Main()
63         {
64            Application.Run( new FrmFontChange() );
65         }
66
67         // changes the font of the Form's TextBox
68         private void mnuitmFont_Click(
69            object sender, System.EventArgs e )
70         {
71            FontDialog dlgFontDialog;
72
73            dlgFontDialog = new FontDialog();
74            dlgFontDialog.ShowDialog();
75            txtDisplay.Font = dlgFontDialog.Font;
76
77         } // end method mnuitmFont_Click
78
79      } // end class FrmFontChange
80   }
```

Should check if the user clicked cancel before applying the font

Answer: The complete corrected code should read:

```
1   // Exercise 22.15 Solution
2   // FontChange.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace FontChange
12  {
13     /// <summary>
14     /// Summary description for FrmFontChange.
15     /// </summary>
16     public class FrmFontChange : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.TextBox txtDisplay;
19        private System.Windows.Forms.Label lblSample;
20        private System.Windows.Forms.MainMenu mnuMain;
21        private System.Windows.Forms.MenuItem menuItem1;
22        private System.Windows.Forms.MenuItem mnuitmFont;
23        /// <summary>
24        /// Required designer variable.
25        /// </summary>
26        private System.ComponentModel.Container components = null;
27
28        public FrmFontChange()
29        {
30           //
```

```
31            // Required for Windows Form Designer support
32            //
33            InitializeComponent();
34
35            //
36            // TODO: Add any constructor code after InitializeComponent
37            // call
38            //
39         }
40
41         /// <summary>
42         /// Clean up any resources being used.
43         /// </summary>
44         protected override void Dispose( bool disposing )
45         {
46            if( disposing )
47            {
48               if (components != null)
49               {
50                  components.Dispose();
51               }
52            }
53            base.Dispose( disposing );
54         }
55
56         // Windows Form Designer generated code
57
58         /// <summary>
59         /// The main entry point for the application.
60         /// </summary>
61         [STAThread]
62         static void Main()
63         {
64            Application.Run( new FrmFontChange() );
65         }
66
67         // changes the font of the Form's TextBox
68         private void mnuitmFont_Click(
69            object sender, System.EventArgs e )
70         {
71            DialogResult result;
72            FontDialog dlgFontDialog;
73
74            dlgFontDialog = new FontDialog();
75            result = dlgFontDialog.ShowDialog();
76
77            if ( result == DialogResult.Cancel )
78            {
79               return;
80            }
81            else
82            {
83               txtDisplay.Font = dlgFontDialog.Font;
84            }
85
86         } // end method mnuitmFont_Click
87
88      } // end class FrmFontChange
89   }
```



Programming Challenge

22.16 (***Dvorak Keyboard Application***) Create an application that simulates the letters on the Dvorak keyboard. A Dvorak keyboard allows faster typing by placing the most commonly used keys in the most accessible locations. Use keyboard events to create an application similar to the **Typing** application, except that it simulates the Dvorak keyboard instead of the standard keyboard. The correct Dvorak key should be highlighted on the virtual keyboard and the correct character should be displayed in the `TextBox`. The keys and characters map as follows:

- On the top row, the *P* key of the Dvorak keyboard maps to the *R* key on a standard keyboard, and the *L* key of the Dvorak keyboard maps to the *P* key on a standard keyboard.
- On the middle row, the *A* key remains in the same position and the *S* key on the Dvorak keyboard maps to the semicolon key on the standard keyboard.
- On the bottom row, the *Q* key on the Dvorak keyboard maps to the *X* key on the standard keyboard and the *Z* key maps to the question mark key.
- All of the other keys on the Dvorak keyboard map to the locations shown in Fig. 22.34.

Figure 22.34 **Dvorak Keyboard** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial22\Exercises\DvorakKeyboard` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `DvorakKeyboard.sln` in the `DvorakKeyboard` directory to open the application.

c) ***Creating the KeyPress event handler.*** Add a `KeyPress` event handler for the `TextBox`.

d) ***Creating a switch statement.*** Add a `switch` statement to the `KeyPress` event handler. The `switch` statement should test whether all of the letter keys on the Dvorak keyboard were pressed except for the *S*, *W*, *V* and *Z* keys. If a Dvorak key was pressed, highlight it on the GUI and display the character in the `TextBox`.

e) ***Creating a KeyDown event handler.*** Add a `KeyDown` event handler for the `TextBox`. The *S*, *W*, *V* and *Z* keys do not map to a letter key on the standard keyboard; therefore, a `KeyDown` event handler must be used to determine whether one of these keys was pressed.

f) ***Adding a switch statement.*** Add a `switch` statement to your `KeyDown` event handler that determines whether *S*, *W*, *V* or *Z* was pressed. If one of these keys was pressed, highlight the key, and add the character to the `TextBox`.

g) ***Running the application.*** Select **Debug > Start** to run your application. Use your keyboard to enter text. Verify that the text entered is correct based on the rules in the exercise description. Make sure the correct `Buttons` on the `Form` are highlighted as you enter text.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 22.16 Solution
// DvorakKeyboard.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace DvorakKeyboard
{
   /// <summary>
   /// Summary description for FrmDvorakKeyboard.
   /// </summary>
   public class FrmDvorakKeyboard : System.Windows.Forms.Form
   {
      // Label to display directions to the user
      private System.Windows.Forms.Label lblPrompt;

      // TextBox to display output from typing
      private System.Windows.Forms.RichTextBox txtOutput;

      // Buttons to represent keys on the keyboard
      private System.Windows.Forms.Button btnP;
      private System.Windows.Forms.Button btnY;
      private System.Windows.Forms.Button btnF;
      private System.Windows.Forms.Button btnG;
      private System.Windows.Forms.Button btnC;
      private System.Windows.Forms.Button btnR;
      private System.Windows.Forms.Button btnL;
      private System.Windows.Forms.Button btnA;
      private System.Windows.Forms.Button btnO;
      private System.Windows.Forms.Button btnE;
      private System.Windows.Forms.Button btnU;
      private System.Windows.Forms.Button btnI;
      private System.Windows.Forms.Button btnD;
      private System.Windows.Forms.Button btnH;
```

```
39          private System.Windows.Forms.Button btnT;
40          private System.Windows.Forms.Button btnN;
41          private System.Windows.Forms.Button btnS;
42          private System.Windows.Forms.Button btnQ;
43          private System.Windows.Forms.Button btnJ;
44          private System.Windows.Forms.Button btnK;
45          private System.Windows.Forms.Button btnX;
46          private System.Windows.Forms.Button btnB;
47          private System.Windows.Forms.Button btnM;
48          private System.Windows.Forms.Button btnW;
49          private System.Windows.Forms.Button btnV;
50          private System.Windows.Forms.Button btnZ;
51
52          /// <summary>
53          /// Required designer variable.
54          /// </summary>
55          private System.ComponentModel.Container components = null;
56
57          // reference to last Button pressed
58          Button m_btnLastButton;
59
60          public FrmDvorakKeyboard()
61          {
62             //
63             // Required for Windows Form Designer support
64             //
65             InitializeComponent();
66
67             //
68             // TODO: Add any constructor code after InitializeComponent
69             // call
70             //
71          }
72
73          /// <summary>
74          /// Clean up any resources being used.
75          /// </summary>
76          protected override void Dispose( bool disposing )
77          {
78             if( disposing )
79             {
80                if (components != null)
81                {
82                   components.Dispose();
83                }
84             }
85             base.Dispose( disposing );
86          }
87
88          // Windows Form Designer generated code
89
90          /// <summary>
91          /// The main entry point for the application.
92          /// </summary>
93          [STAThread]
94          static void Main()
95          {
96             Application.Run( new FrmDvorakKeyboard() );
97          }
98
```

```
99          // handles TextBox's KeyPress event
100         private void txtOutput_KeyPress(
101            object sender, System.Windows.Forms.KeyPressEventArgs e )
102         {
103            // convert pressed key to uppercase
104            switch ( Char.ToUpper( e.KeyChar ) )
105            {
106               // following cases test if key pressed was a letter
107               case ( ( char ) ( Keys.A ) ): // a maps to a key
108                  ChangeColor( btnA );
109                  txtOutput.Text += "a";
110                  break;
111
112               case ( ( char ) ( Keys.B ) ): // x maps to b key
113                  ChangeColor( btnX );
114                  txtOutput.Text += "x";
115                  break;
116
117               case ( ( char ) ( Keys.C ) ): // j maps to c key
118                  ChangeColor( btnJ );
119                  txtOutput.Text += "j";
120                  break;
121
122               case ( ( char ) ( Keys.D ) ): // e maps to d key
123                  ChangeColor( btnE );
124                  txtOutput.Text += "e";
125                  break;
126
127               case ( ( char ) ( Keys.F ) ): // u maps to f key
128                  ChangeColor( btnU );
129                  txtOutput.Text += "u";
130                  break;
131
132               case ( ( char ) ( Keys.G ) ): // i maps to g key
133                  ChangeColor( btnI );
134                  txtOutput.Text += "i";
135                  break;
136
137               case ( ( char ) ( Keys.H ) ): // d maps to h key
138                  ChangeColor( btnD );
139                  txtOutput.Text += "d";
140                  break;
141
142               case ( ( char ) ( Keys.I ) ): // c maps to i key
143                  ChangeColor( btnC );
144                  txtOutput.Text += "c";
145                  break;
146
147               case ( ( char ) ( Keys.J ) ): // h maps to j key
148                  ChangeColor( btnH );
149                  txtOutput.Text += "h";
150                  break;
151
152               case ( ( char ) ( Keys.K ) ): // t maps to k key
153                  ChangeColor( btnT );
154                  txtOutput.Text += "t";
155                  break;
156
157               case ( ( char ) ( Keys.L ) ): // n maps to l key
158                  ChangeColor( btnN );
```

```
159            txtOutput.Text += "n";
160            break;
161
162         case ( ( char ) ( Keys.M ) ): // m maps to m key
163            ChangeColor( btnM );
164            txtOutput.Text += "m";
165            break;
166
167         case ( ( char ) ( Keys.N ) ): // b maps to n key
168            ChangeColor( btnB );
169            txtOutput.Text += "b";
170            break;
171
172         case ( ( char ) ( Keys.O ) ): // r maps to o key
173            ChangeColor( btnR );
174            txtOutput.Text += "r";
175            break;
176
177         case ( ( char ) ( Keys.P ) ): // l maps to p key
178            ChangeColor( btnL );
179            txtOutput.Text += "l";
180            break;
181
182         case ( ( char ) ( Keys.R ) ): // p maps to r key
183            ChangeColor( btnP );
184            txtOutput.Text += "p";
185            break;
186
187         case ( ( char ) ( Keys.S ) ): // o maps to s key
188            ChangeColor( btnO );
189            txtOutput.Text += "o";
190            break;
191
192         case ( ( char ) ( Keys.T ) ): // y maps to t key
193            ChangeColor( btnY );
194            txtOutput.Text += "y";
195            break;
196
197         case ( ( char ) ( Keys.U ) ): // g maps to u key
198            ChangeColor( btnG );
199            txtOutput.Text += "g";
200            break;
201
202         case ( ( char ) ( Keys.V ) ): // k maps to v key
203            ChangeColor( btnK );
204            txtOutput.Text += "k";
205            break;
206
207         case ( ( char ) ( Keys.X ) ): // q maps to x key
208            ChangeColor( btnQ );
209            txtOutput.Text += "q";
210            break;
211
212         case ( ( char ) ( Keys.Y ) ): // f maps to y key
213            ChangeColor( btnF );
214            txtOutput.Text += "f";
215            break;
216
217      } // ends test for letters
218
```

```
219         } // end method txtOutput_KeyPress
220
221         // handles TextBox's KeyDown event
222         private void txtOutput_KeyDown(
223            object sender, System.Windows.Forms.KeyEventArgs e )
224         {
225            switch ( e.KeyData )
226            {
227               // use KeyDown for these keys because the Dvorak
228               // representation does not map to letters of
229               // QWERTY keyboard
230               case Keys.OemSemicolon: // s maps to semicolon key
231                  ChangeColor( btnS );
232                  txtOutput.Text += "s";
233                  break;
234               case Keys.Oemcomma: // w maps to comma key
235                  ChangeColor( btnW );
236                  txtOutput.Text += "w";
237                  break;
238               case Keys.OemPeriod: // y maps to period key
239                  ChangeColor( btnV );
240                  txtOutput.Text += "v";
241                  break;
242               case Keys.OemQuestion: // z maps t question mark key
243                  ChangeColor( btnZ );
244                  txtOutput.Text += "z";
245                  break;
246            }
247
248         } // end method txtOutput_KeyDown
249
250         // handles TextBox's KeyUp event
251         private void txtOutput_KeyUp(
252            object sender, System.Windows.Forms.KeyEventArgs e )
253         {
254            ResetColor();
255
256         } // end method txtOutput_KeyUp
257
258         // highlight Button passed as argument
259         private void ChangeColor( Button btnButton )
260         {
261            ResetColor();
262            btnButton.BackColor = Color.LightGoldenrodYellow;
263            m_btnLastButton = btnButton;
264
265         } // end method ChangeColor
266
267         // changes m_btnLastButton's color if it refers to a Button
268         private void ResetColor()
269         {
270            if ( m_btnLastButton != null )
271            {
272               m_btnLastButton.BackColor = SystemColors.Control;
273            }
274
275         } // end method ResetColor
276
277      } // end class FrmDvorakKeyboard
278   }
```

23 TUTORIAL

Screen Scraping Application

Introducing `string` Processing

Solutions

Instructor's Manual Exercise Solutions Tutorial 23

MULTIPLE CHOICE QUESTIONS

23.1 Extracting desired information from Web pages is called _____.

a) Web crawling
b) screen scraping
c) querying
d) redirection

23.2 If the `IndexOf` method does not find the specified substring, it returns _____.

a) `false`
b) `0`
c) `-1`
d) None of the above.

23.3 The `String` class allows you to _____ `string`s.

a) search
b) retrieve characters from
c) replace characters in
d) All of the above.

23.4 _____ is a technology for describing Web content.

a) The `String` class
b) A `string` literal
c) HTML
d) A screen scraper

23.5 The `String` class is located in the _____ namespace.

a) `String`
b) `System.Strings`
c) `System.IO`
d) `System`

23.6 The _____ method creates a new `string` object by copying part of an existing `string` object.

a) `StringCopy`
b) `Substring`
c) `CopyString`
d) `CopySubString`

23.7 All `string` objects are _____.

a) the same size
b) always equal to each other
c) preceded by at least one whitespace character
d) immutable

23.8 The `IndexOf` method does not examine any characters that occur prior to the _____.

a) starting index
b) first match
c) last character of the `string`
d) None of the above.

23.9 The _____ method determines whether a `string` ends with a particular substring.

a) `CheckEnd`
b) `StringEnd`
c) `EndsWith`
d) `EndIs`

23.10 The `Trim` method removes all whitespace characters that appear _____ a `String`.

a) in
b) at the beginning of
c) at the end of
d) at the beginning and end of

Answers: 23.1) b. 23.2) c. 23.3) d. 23.4) c. 23.5) d. 23.6) b. 23.7) d. 23.8) a. 23.9) c. 23.10) d.

EXERCISES

23.11 (***Supply Calculator Application***) Write an application that calculates the cost of all the supplies added to the user's shopping list (Fig. 23.18). The application should contain two `ListBox`es. The first `ListBox` contains all the supplies offered and their respective prices. Users should be able to select the desired supplies from the first `ListBox` and add them to the second `ListBox`. Provide a **Calculate** `Button` that displays the total price for the user's shopping list (the contents of the second `ListBox`).

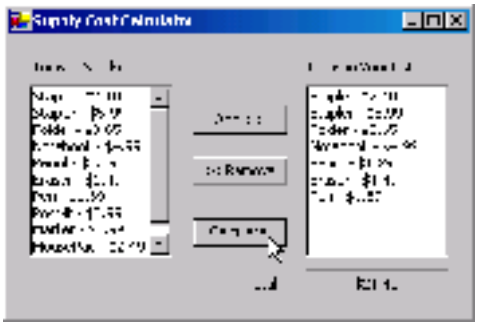


Figure 23.18 Supply Calculator application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial23\Exercises\SupplyCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SupplyCalculator.sln` in the `SupplyCalculator` directory to open the application.

c) ***Adding the Add >> Button's event handler.*** Double click the **Add >>** `Button` to create an empty event handler. Add code to the event handler that adds the selected item from the first `ListBox` to the `lstStock ListBox`. Make sure to check that at least one item is selected in the first `ListBox` before attempting to add an item to the `lstStock ListBox`.

d) ***Enabling the Buttons.*** Once the user adds something to the `lstStock ListBox`, set the `Enabled` properties of the **<< Remove** and **Calculate** `Buttons` to `true`.

e) ***Deselecting the items.*** Once the items are added to the `lstStock ListBox`, make sure that those items are deselected in the `lstSupply ListBox`. Also, clear the **Total:** `Label` to indicate to the user that a new total price must be calculated.

f) ***Adding the Remove Button's event handler.*** Double click the **<< Remove** `Button` to create an empty event handler. Use a `while` loop to remove any selected items in the `lstStock ListBox`. Make sure to check that at least one item is selected before attempting to remove an item. [*Hint*: The `lstStock.Items.RemoveAt( intIndex )` method will remove the item located at `intIndex` from the `lstStock ListBox`.]

g) ***Adding the Calculate Button's event handler.*** Double click the **Calculate** `Button` to create an empty event handler. Use a `for` statement to loop through all the items in the `lstStock ListBox`. Convert each item from the `ListBox` into a `string`. Then, use the `string` method `Substring` to extract the price of each item.

h) ***Displaying the total.*** Convert the `string` representing each item's price to a `decimal`, and add this to the overall total (of type `decimal`). Remember to output the value in currency format.

i) ***Running the application.*** Select **Debug > Start** to run your application. Use the **Add >>** and **<< Remove** `Buttons` to add and remove items from the **Items in Your List:** `ListBox`. Click the **Calculate** `Button` and verify that the total price displayed is correct.

j) ***Closing the application.*** Close your running application by clicking its close box.

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 23.11 Solution
2  // SupplyCalculator.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
```

```
11 namespace SupplyCalculator
12 {
13    /// <summary>
14    /// Summary description for FrmSupplyCalculator.
15    /// </summary>
16    public class FrmSupplyCalculator : System.Windows.Forms.Form
17    {
18       // Label and ListBox to display items in stock
19       private System.Windows.Forms.Label lblStockDescription;
20       private System.Windows.Forms.ListBox lstSupply;
21
22       // Buttons to add or remove an item to or from your list
23       private System.Windows.Forms.Button btnAdd;
24       private System.Windows.Forms.Button btnRemove;
25
26       // Button to calculate total cost of your items
27       private System.Windows.Forms.Button btnCalculate;
28
29       // Label and ListBox to display your items
30       private System.Windows.Forms.Label lblListDescription;
31       private System.Windows.Forms.ListBox lstStock;
32
33       // Labels to display total cost of your items
34       private System.Windows.Forms.Label lblTotalLabel;
35       private System.Windows.Forms.Label lblTotal;
36
37       /// <summary>
38       /// Required designer variable.
39       /// </summary>
40       private System.ComponentModel.Container components = null;
41
42       public FrmSupplyCalculator()
43       {
44          //
45          // Required for Windows Form Designer support
46          //
47          InitializeComponent();
48
49          //
50          // TODO: Add any constructor code after InitializeComponent
51          // call
52          //
53       }
54
55       /// <summary>
56       /// Clean up any resources being used.
57       /// </summary>
58       protected override void Dispose( bool disposing )
59       {
60          if( disposing )
61          {
62             if (components != null)
63             {
64                components.Dispose();
65             }
66          }
67          base.Dispose( disposing );
68       }
69
70       // Windows Form Designer generated code
```

```
71
72          /// <summary>
73          /// The main entry point for the application.
74          /// </summary>
75          [STAThread]
76          static void Main()
77          {
78             Application.Run( new FrmSupplyCalculator() );
79          }
80
81          // remove item from shopping list
82          private void btnRemove_Click(
83             object sender, System.EventArgs e )
84          {
85             // if there is at least one item selected
86             if ( lstStock.SelectedIndex != -1 )
87             {
88                // remove items while there are selected items
89                while ( lstStock.SelectedIndex != -1 )
90                {
91                   // remove item at the selected index
92                   lstStock.Items.RemoveAt( lstStock.SelectedIndex );
93                }
94             }
95             else
96             {
97                // display message if there is no item selected
98                MessageBox.Show( "Please select item to remove",
99                   "Cannot Remove", MessageBoxButtons.OK,
100                  MessageBoxIcon.Exclamation );
101            }
102
103            // if there is no item
104            if ( lstStock.Items.Count < 1 )
105            {
106               // disable the Remove and Calculate Buttons
107               btnRemove.Enabled = false;
108               btnCalculate.Enabled = false;
109            }
110
111            lblTotal.Text = ""; // clear lblTotal Label
112
113         } // end method btnRemove_Click
114
115         // add item to shopping list
116         private void btnAdd_Click(
117            object sender, System.EventArgs e )
118         {
119            // if there is at least one item selected
120            if ( lstSupply.SelectedIndex != -1 )
121            {
122               // add item to lstStock ListBox
123               lstStock.Items.Add( lstSupply.SelectedItem );
124
125               // enable the Remove and Calculate Buttons
126               btnRemove.Enabled = true;
127               btnCalculate.Enabled = true;
128               lstSupply.SelectedIndex = -1; // unselect items
129               lblTotal.Text = ""; // clear lblTotal
130            }
```

```
131
132       } // end method btnAdd_Click
133
134       // calculate total price for shopping list
135       private void btnCalculate_Click(
136          object sender, System.EventArgs e )
137       {
138          decimal decTotal = 0; // total amount
139          string strPrice; // temporary price variable
140          int intCounter; // counter variable
141
142          // run through list of item(s)
143          for ( intCounter = 0; intCounter < lstStock.Items.Count;
144             intCounter++ )
145          {
146             // retrieve price from items
147             strPrice = lstStock.Items[ intCounter ].ToString();
148
149             // get substring starting after the $
150             strPrice = strPrice.Substring(
151                strPrice.IndexOf( "$" ) + 1 );
152
153             // add price of each item to total
154             decTotal += Decimal.Parse( strPrice );
155          }
156
157          // display total
158          lblTotal.Text = String.Format( "{0:C}", decTotal );
159
160       } // end method btnCalculate_Click
161
162    } // end class FrmSupplyCalculator
163 }
```

23.12 (*Encryption Application*) Write an application that encrypts a message from the user (Fig. 23.19). The application should be able to encrypt the message in two different ways—substitution cipher and a transposition cipher (both described below). The user should be able to enter the message in a `TextBox` and select the desired method of encryption. Display the encrypted message in a `Label`.

In a substitution cipher, every character in the English alphabet is represented by a different character in the substitution alphabet. Every time a letter occurs in the English sentence, it is replaced by the letter in the corresponding index of the substitution string. In a transposition cipher, two `string`s are created. The first new `string` contains all the characters at the even indices of the input `string`. The second new `string` contains all of the characters at the odd indices. The new `string`s are the encrypted text. For example a transposition cipher for the word "code" would be: "cd oe."

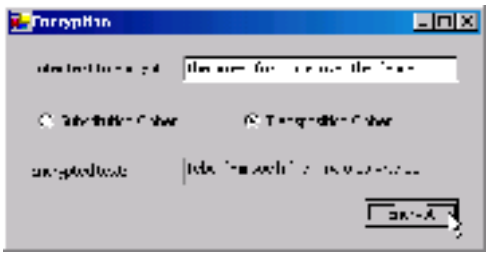


Figure 23.19 **Encryption** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial23\Exercises\Encryption` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Encryption.sln` in the `Encryption` directory to open the application.

c) ***Adding the Encrypt `Button`'s event handler.*** Double click the **Encrypt** `Button` to create an empty event handler.

d) ***Determine the cipher method.*** Use `if`...`else` statements to determine which method of encryption the user has selected and call the appropriate method.

e) ***Locating the `SubstitutionCipher` method.*** Locate the `SubstitutionCipher` method. The English and substitution alphabet `string`s have been defined for you in this method.

f) ***Converting the text input to lowercase.*** Add code to the `SubstitutionCipher` method that uses the `ToLower` method of the `string` class to make all the characters in the input `string` (`txtPlainText.Text`) lowercase.

g) ***Performing the substitution encryption.*** Use nested `for` loops to iterate through each character of the input `string`. When each character from the input `string` is found in the `string` holding the English alphabet, replace the character in the input `string` with the character located at the same index in the substitution `string`.

h) ***Displaying the `string`.*** Now that the `string` has been substituted with all the corresponding cipher characters, assign the cipher `string` to the `lblCipherText Label`.

i) ***Locating the `TranspositionCipher` method.*** Locate the `TranspositionCipher` method. Define three variables—a counter variable and two `string`s (each representing a word).

j) ***Extracting the first word.*** Use a `while` statement to retrieve all the "even" indices (starting from `0`) from the input `string`. Increment the counter variable by `2` each time, and add the characters located at even indices to the first `string` created in *Step h*.

k) ***Extracting the second word.*** Use another `while` statement to retrieve all the "odd" indices (starting from `1`) from the same input `string`. Increment the counter variable by `2`, and add the characters at odd indices to the second `string` that you created in *Step h*.

l) ***Outputting the result.*** Add the two `string`s together with a space in between, and output the result to the `lblCipherText Label`.

m) ***Running the application.*** Select **Debug > Start** to run your application. Enter text into the **Enter text to encrypt:** `TextBox`. Select the **Substitution Cipher** `RadioButton` and click the **Encrypt** `Button`. Verify that the output is the properly encrypted text using the substitution cipher. Select the **Transposition Cipher** `RadioButton` and click the **Encrypt** `Button`. Verify that the output is the properly encrypted text using the transposition cipher.

n) ***Closing the application.*** Close your running application by clicking its close box.

o) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 23.12 Solution
2  // Encryption.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Encryption
12 {
13    /// <summary>
14    /// Summary description for FrmEncryption.
15    /// </summary>
16    public class FrmEncryption : System.Windows.Forms.Form
```

```
17    {
18       // Label and TextBox to input text to encrypt
19       private System.Windows.Forms.Label lblInstruction;
20       private System.Windows.Forms.TextBox txtPlainText;
21
22       // RadioButtons to choose type of encryption
23       private System.Windows.Forms.RadioButton radSubstitution;
24       private System.Windows.Forms.RadioButton radTransposition;
25
26       // Labels to output the encrypted text
27       private System.Windows.Forms.Label lblResult;
28       private System.Windows.Forms.Label lblCipherText;
29
30       // Button to perform the encryption
31       private System.Windows.Forms.Button btnEncrypt;
32
33       /// <summary>
34       /// Required designer variable.
35       /// </summary>
36       private System.ComponentModel.Container components = null;
37
38       public FrmEncryption()
39       {
40          //
41          // Required for Windows Form Designer support
42          //
43          InitializeComponent();
44
45          //
46          // TODO: Add any constructor code after InitializeComponent
47          // call
48          //
49       }
50
51       /// <summary>
52       /// Clean up any resources being used.
53       /// </summary>
54       protected override void Dispose( bool disposing )
55       {
56          if( disposing )
57          {
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmEncryption() );
75       }
76
```

```
77          // using the substitution cipher
78          private void SubstitutionCipher()
79          {
80             // normal alphabet string
81             string strNormalAlphabet =
82                "abcdefghijklmnopqrstuvwxyz .!?,";
83
84             // substitution alphabet string
85             string strCipherAlphabet =
86                "cdefg.hijk!lmn opqr?stuv,wxyzab";
87
88             int intIndex1; // index variable in for loop
89             int intIndex2; // inner index variable
90             string strPlain; // string entered by the user
91             string strCipher = ""; // encrypted string
92
93             lblCipherText.Text = ""; // clear output TextBox
94
95             // change all the characters to lower case
96             strPlain = txtPlainText.Text.ToLower();
97
98             // iterate through the length of the string
99             for ( intIndex1 = 0; intIndex1 < txtPlainText.Text.Length;
100               intIndex1++ )
101            {
102               // iterate through alphabet and special character( .!?,")
103               for ( intIndex2 = 0; intIndex2 <= 30; intIndex2++ )
104               {
105                  // compare characters
106                  if ( strPlain[ intIndex1 ] ==
107                     strNormalAlphabet[ intIndex2 ] )
108                  {
109                     // build encrypted text
110                     strCipher += strCipherAlphabet[ intIndex2 ];
111                  }
112               }
113            }
114
115            lblCipherText.Text = strCipher; // output encrypted string
116
117         } // end method SubstitutionCipher
118
119         // using the transposition cipher
120         private void TranspositionCipher()
121         {
122            int intCounter = 0; // counter variable
123            string strFirstWord = ""; // first word
124            string strLastWord = ""; // second word
125
126            // create first word from the "even" index
127            while ( intCounter < txtPlainText.Text.Length )
128            {
129               // add character from specified location to strFirstWord
130               strFirstWord += txtPlainText.Text[ intCounter ];
131               intCounter += 2; // increment counter by 2;
132
133            } // loop through the entire string
134
135            // create second word from the "odd" indices
136            intCounter = 1;
```

```
137
138          while ( intCounter < txtPlainText.Text.Length )
139          {
140             // add character from specified location to strLastWord
141             strLastWord += txtPlainText.Text[ intCounter ];
142             intCounter += 2; // increment counter by 2
143
144          } // loop through the entire string
145
146          // output encrypted text
147          lblCipherText.Text = strFirstWord + " " + strLastWord;
148
149       } // end method TranspositionCipher
150
151       // encrypt a string of characters
152       private void btnEncrypt_Click(
153          object sender, System.EventArgs e )
154       {
155          // determine the selected RadioButton
156          if ( radSubstitution.Checked == true )
157          {
158             SubstitutionCipher(); // call SubstitutionCipher
159          }
160          else
161          {
162             TranspositionCipher(); // call TranspositionCipher
163          }
164
165       } // end method btnEncrypt_Click
166
167    } // end class FrmEncryption
168 }
```

23.13 (*Anagram Application*) Write an **Anagram** game that contains an array of preset words. The game should randomly select a word and scramble its letters. A `Label` displays the scrambled word for the user to guess. If the user guesses correctly, display a message, and repeat the process with a different word. If the guess is incorrect, display a message, and let the user try again (Fig. 23.20).

Figure 23.20 **Anagram** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial23\Exercises\Anagram` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Anagram.sln` in the `Anagram` directory to open the application.

c) ***Locating the `GenerateAnagram` method.*** Locate the `GenerateAnagram` method. It is the first method after the `FrmAnagram_Load` event handler.

d) ***Picking a random word***. Generate a random number to use as the index of the word in the `m_strAnagram` array. Retrieve the word from the `m_strAnagram` array, using the first random number as an index. Store the word in another `string` variable. Generate a second random number to store the index of a character to be moved.

e) ***Generating the scrambled word***. Use a `for` statement to iterate through the word 20 times. Each time the loop executes, pass the second random number created in *Step d* to the `string` indexer. Append the character returned by the indexer to the end of the `string`, and remove it from its original position. Next, generate a new random number to move a different character during the next iteration of the loop. Remember to output the final word to the `lblAnagram Label`.

f) ***Defining the Submit `Button`'s event handler***. Double click the **Submit** `Button` to generate an empty event handler.

g) ***Testing the user's input***. Use an `if`…`else` statement to determine whether the user's input matches the actual word. If the user is correct, clear and place the focus on the `TextBox` and generate a new word. Otherwise, select the user's text and place focus on the `TextBox`.

h) ***Running the application.*** Select **Debug > Start** to run your application. Submit correct answers and incorrect answers, and verify that the appropriate message is displayed each time.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 23.13 Solution
// Anagram.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Anagram
{
   /// <summary>
   /// Summary description for FrmAnagram.
   /// </summary>
   public class FrmAnagram : System.Windows.Forms.Form
   {
      // Label to display the anagram
      private System.Windows.Forms.Label lblAnagram;

      // Label and TextBox to input a guess
      private System.Windows.Forms.Label lblGuess;
      private System.Windows.Forms.TextBox txtGuess;

      // Button to submit a guess
      private System.Windows.Forms.Button btnSubmit;

      // Label to display the result of a guess
      private System.Windows.Forms.Label lblResult;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // array of words to be scrambled
```

```
37          private string[] m_strAnagram = {
38             "controls", "events", "properties", "visual", "program",
39             "application", "namespace", "debugger", "database", "files",
40             "inheritance","assembly", "multimedia", "methods",
41             "classes","arrays", "strings", "collections",
42             "integration", "structures" };
43
44          private Random m_objRandom = new Random(); // random number
45          private int m_intRandomNumber; // random index variable
46          private string m_strScrambled; // randomly chosen word
47
48          public FrmAnagram()
49          {
50             //
51             // Required for Windows Form Designer support
52             //
53             InitializeComponent();
54
55             //
56             // TODO: Add any constructor code after InitializeComponent
57             // call
58             //
59          }
60
61          /// <summary>
62          /// Clean up any resources being used.
63          /// </summary>
64          protected override void Dispose( bool disposing )
65          {
66             if( disposing )
67             {
68                if (components != null)
69                {
70                   components.Dispose();
71                }
72             }
73             base.Dispose( disposing );
74          }
75
76          // Windows Form Designer generated code
77
78          /// <summary>
79          /// The main entry point for the application.
80          /// </summary>
81          [STAThread]
82          static void Main()
83          {
84             Application.Run( new FrmAnagram() );
85          }
86
87          // generate new scrambled word
88          private void FrmAnagram_Load(
89             object sender, System.EventArgs e )
90          {
91             GenerateAnagram();
92
93          } // end method FrmAnagram_Load
94
95          // scramble words
96          private void GenerateAnagram()
```

```
97          {
98             // generate new random number
99             m_intRandomNumber = m_objRandom.Next( 0, 19 );
100
101            // select new word from array with m_intRandomNumber index
102            m_strScrambled = m_strAnagram[ m_intRandomNumber ];
103
104            // generate new random index
105            int intRandomIndex =
106               m_objRandom.Next( 0, m_strScrambled.Length - 1 );
107
108            int intCounter; // loop counter variable
109
110            // loop to generate scrambled word
111            for ( intCounter = 0; intCounter <= 20; intCounter++ )
112            {
113               // attach character at the end of string
114               m_strScrambled += m_strScrambled[ intRandomIndex ];
115
116               // remove character from the word
117               m_strScrambled =
118                  m_strScrambled.Remove( intRandomIndex, 1 );
119
120               // new random index
121               intRandomIndex =
122                  m_objRandom.Next( 0, m_strScrambled.Length - 1 );
123            }
124
125            lblAnagram.Text = m_strScrambled; // display scrambled word;
126
127         } // end method GenerateAnagram
128
129         // check if the user's answer is correct
130         private void btnSubmit_Click(
131            object sender, System.EventArgs e )
132         {
133            // answer is correct
134            if ( txtGuess.Text == m_strAnagram[ m_intRandomNumber ] )
135            {
136               lblResult.Text = "You are Correct!";
137               GenerateAnagram(); // generate new word
138               txtGuess.Clear(); // clear the TextBox
139               txtGuess.Focus(); // place focus on TextBox
140            }
141            else
142            {
143               // answer is incorrect
144               lblResult.Text = "Wrong answer. Try again!";
145               txtGuess.Focus(); // place focus on TextBox
146               txtGuess.SelectAll(); // select the answer
147            }
148
149         } // end method btnSubmit_Click
150
151      } // end class FrmAnagram
152   }
```

What does this code do? ▶ **23.14** What is assigned to `strResult` when the following code executes?

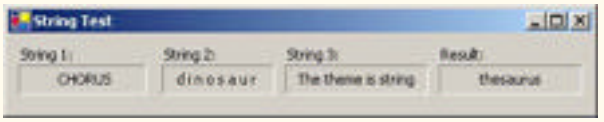
```
string strWord1 = "CHORUS";
string strWord2 = "d i n o s a u r";
string strWord3 = "The theme is string.";
string strResult;

strResult = strWord1.ToLower();
strResult = strResult.Substring( 4 );
strWord2 = strWord2.Replace( " ", "" );
strWord2 = strWord2.Substring( 4, 4 );
strResult = strWord2 + strResult;

strWord3 = strWord3.Substring(
   strWord3.IndexOf( " " ) + 1, 3 );

strResult = strWord3.Insert( 3, strResult );
```

Answer: After assigning initial values to `string`s `strWord1`, `strWord2` and `strWord3`, the code above changes the word `"CHORUS"` to all lowercase letters, resulting in `"chorus"` being assigned to `strResult`. `strResult` then is assigned the substring of `"chorus"` beginning at the fifth character (index 4), `"u"`, resulting in `strResult`'s value as `"us"`. The next line of code deletes spaces from `strWord2`, resulting in the word `"dinosaur"`. The substring of `"dinosaur"` beginning at the fifth character and of length 4 (`"saur"`) is then assigned to `strWord2`. Then, the value of `strResult` (`"us"`) is appended to the end of `strWord2` (`"us"`) and placed in `strResult`, yielding `"saurus"`. Following that, `strWord3` is assigned a substring of itself beginning one character after the first space character and of length 3 (`"the"`). Finally, the value `"saurus"` (`strResult`) is inserted into `strWord3` at the location of the fourth character (the end of the string) and assigned to `strResult`. The final value of `strResult` is the word `"thesaurus"`. The complete code reads:

```
// Exercise 23.14 Solution
// StringTest.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace StringTest
{
   /// <summary>
   /// Summary description for FrmStringTest.
   /// </summary>
   public class FrmStringTest : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblResultOut;
      private System.Windows.Forms.Label lblString3Out;
      private System.Windows.Forms.Label lblString2Out;
      private System.Windows.Forms.Label lblString1Out;
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblString3;
      private System.Windows.Forms.Label lblString2;
      private System.Windows.Forms.Label lblString1;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmStringTest()
```

```
32          {
33             //
34             // Required for Windows Form Designer support
35             //
36             InitializeComponent();
37
38             //
39             // TODO: Add any constructor code after InitializeComponent
40             // call
41             //
42          }
43
44          /// <summary>
45          /// Clean up any resources being used.
46          /// </summary>
47          protected override void Dispose( bool disposing )
48          {
49             if( disposing )
50             {
51                if (components != null)
52                {
53                   components.Dispose();
54                }
55             }
56             base.Dispose( disposing );
57          }
58
59          // Windows Form Designer generated code
60
61          /// <summary>
62          /// The main entry point for the application.
63          /// </summary>
64          [STAThread]
65          static void Main()
66          {
67             Application.Run( new FrmStringTest() );
68          }
69
70          // manipulates several strings and displays the result
71          private void FrmStringTest_Load(
72             object sender, System.EventArgs e )
73          {
74             string strWord1 = "CHORUS";
75             string strWord2 = "d i n o s a u r";
76             string strWord3 = "The theme is string.";
77             string strResult;
78
79             strResult = strWord1.ToLower();
80             strResult = strResult.Substring( 4 );
81             strWord2 = strWord2.Replace( " ", "" );
82             strWord2 = strWord2.Substring( 4, 4 );
83             strResult = strWord2 + strResult;
84
85             strWord3 = strWord3.Substring(
86                strWord3.IndexOf( " " ) + 1, 3 );
87
88             strResult = strWord3.Insert( 3, strResult );
89
90             lblResultOut.Text = strResult;
91
```

```
92         } // end method FrmStringTest_Load
93
94      } // end class FrmStringTest
95   }
```



What's wrong with this code?

23.15 This code should remove all commas from `strTest`. Find the error(s) in the following code.

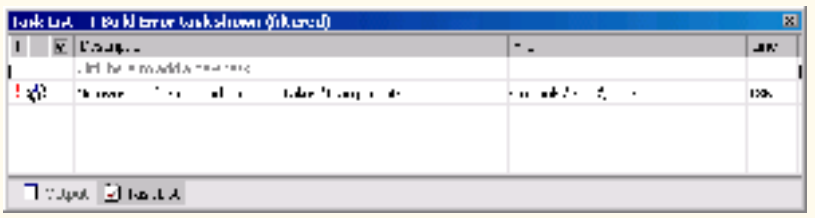
```
1  string strTest = "Bug,2,Bug";
2  strTest = strTest.Replace( "" );
```

Answer: The `Replace` method takes two arguments: one substring to search for, and another substring to replace all matching occurrences of the first argument. The complete incorrect code reads:

```
1  // Exercise 23.15 Solution
2  // RemoveCommas.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace RemoveCommas
12 {
13    /// <summary>
14    /// Summary description for FrmRemoveCommas.
15    /// </summary>
16    public class FrmRemoveCommas : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblResult;
19       private System.Windows.Forms.Label lblOriginalOut;
20       private System.Windows.Forms.Label lblModified;
21       private System.Windows.Forms.Label lblOriginal;
22       /// <summary>
23       /// Required designer variable.
24       /// </summary>
25       private System.ComponentModel.Container components = null;
26
27       public FrmRemoveCommas()
28       {
29          //
30          // Required for Windows Form Designer support
31          //
32          InitializeComponent();
33
34          //
35          // TODO: Add any constructor code after InitializeComponent
36          // call
37          //
```

```
38         }
39
40         /// <summary>
41         /// Clean up any resources being used.
42         /// </summary>
43         protected override void Dispose( bool disposing )
44         {
45            if( disposing )
46            {
47               if (components != null)
48               {
49                  components.Dispose();
50               }
51            }
52            base.Dispose( disposing );
53         }
54
55         // Windows Form Designer generated code
56
57         /// <summary>
58         /// The main entry point for the application.
59         /// </summary>
60         [STAThread]
61         static void Main()
62         {
63            Application.Run( new FrmRemoveCommas() );
64         }
65
66         // removes all commas from the string and displays the result
67         private void FrmRemoveCommas_Load(
68            object sender, System.EventArgs e )
69         {
70            string strTest = "Bug,2,Bug";
71            strTest = strTest.Replace( "" );
72
73            lblResult.Text = strTest;
74
75         } // end method FrmRemoveCommas_Load
76
77      } // end class FrmRemoveCommas
78   }
79
```

`Replace` takes 2 arguments, the `string` to search for and the `string` to replace it with



Answer: The complete corrected code should read:

```
1  // Exercise 23.15 Solution
2  // RemoveCommas.cs (Correct)
3
4  using System;
5  using System.Drawing;
```

```
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace RemoveCommas
{
   /// <summary>
   /// Summary description for FrmRemoveCommas.
   /// </summary>
   public class FrmRemoveCommas : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblResult;
      private System.Windows.Forms.Label lblOriginalOut;
      private System.Windows.Forms.Label lblModified;
      private System.Windows.Forms.Label lblOriginal;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmRemoveCommas()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmRemoveCommas() );
      }

```

```
66         // removes all commas from the string and displays the result
67         private void FrmRemoveCommas_Load(
68            object sender, System.EventArgs e )
69         {
70            string strTest = "Bug,2,Bug";
71            strTest = strTest.Replace( ",", "" );
72
73            lblResult.Text = strTest;
74
75         } // end method FrmRemoveCommas_Load
76
77      } // end class FrmRemoveCommas
78   }
```

Programming Challenge

23.16 (***Pig Latin Application***) Write an application that encodes English language phrases into pig Latin (Fig. 23.21). Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following method to form the pig Latin words:

> *To form the pig Latin version of English-language phrase, the translation proceeds one word at a time. To translate an English word into a pig Latin word, place the first letter of the English word (if it is not a vowel) at the end of the English word and add the letters "ay." If the first letter of the English word is a vowel, place it at the end of the word and add "y." Using this method, the word "jump" becomes "umpjay", the word "the" becomes "hetay" and the word "ace" becomes "ceay." Blanks between words remain blanks.*

Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks, and all words have two or more letters. Enable the user to input a sentence. The `TranslateToPigLatin` method should translate the sentence into pig Latin, word by word. [*Hint*: You will need to use the `Join` and `Split` methods of the `string` class demonstrated in Fig. 23.16 to form the pig Latin phrases].

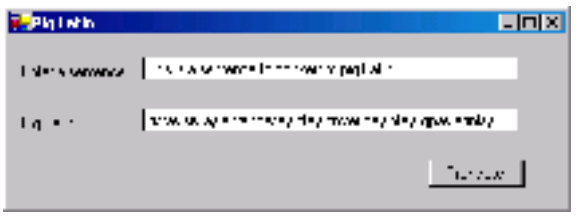


Figure 23.21 Pig Latin application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial23\Exercises\PigLatin` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `PigLatin.sln` in the `PigLatin` directory to open the application.

c) ***Splitting the sentence.*** Use the `Split` method on the `string` passed to the `TranslateToPigLatin` method. Assign the result of this operation to `strWords`.

d) ***Retrieving the word's first letter.*** Declare a `for` loop that iterates through your array of words. As you iterate through the array, store each word's first letter in `strTemporary`.

e) ***Determining the suffix.*** Use `if...else` statements to determine the suffix for each word. Store this suffix in `strSuffix`.

f) ***Generating new words.*** Generate the new words by arranging each word's pieces in the proper order.

g) ***Returning the new sentence.*** When the `for` loop finishes, use the `Join` method to combine all of the elements in `strWords`, and return the new pig Latin sentence.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter a sentence and click the **Translate** `Button`. Verify that the sentence is correctly converted into pig Latin.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 23.16 Solution
2  // PigLatin.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace PigLatin
12 {
13    /// <summary>
14    /// Summary description for FrmPigLatin.
15    /// </summary>
16    public class FrmPigLatin : System.Windows.Forms.Form
17    {
18       // Label and TextBox to input English phrase
19       private System.Windows.Forms.Label lblInput;
20       private System.Windows.Forms.TextBox txtInput;
21
22       // Label and TextBox to output Pig Latin phrase
23       private System.Windows.Forms.Label lblOutput;
24       private System.Windows.Forms.TextBox txtOutput;
25
26       // Button to translate from English to Pig Latin
27       private System.Windows.Forms.Button btnTranslate;
28
29       /// <summary>
30       /// Required designer variable.
31       /// </summary>
32       private System.ComponentModel.Container components = null;
33
34       public FrmPigLatin()
35       {
36          //
37          // Required for Windows Form Designer support
38          //
39          InitializeComponent();
40
41          //
42          // TODO: Add any constructor code after InitializeComponent
43          // call
44          //
45       }
46
47       /// <summary>
48       /// Clean up any resources being used.
49       /// </summary>
```

```
50          protected override void Dispose( bool disposing )
51          {
52             if( disposing )
53             {
54                if (components != null)
55                {
56                   components.Dispose();
57                }
58             }
59             base.Dispose( disposing );
60          }
61
62          // Windows Form Designer generated code
63
64          /// <summary>
65          /// The main entry point for the application.
66          /// </summary>
67          [STAThread]
68          static void Main()
69          {
70             Application.Run( new FrmPigLatin() );
71          }
72
73          // recieve sentence from user and send to TranslateToPigLatin
74          private void btnTranslate_Click(
75             object sender, System.EventArgs e )
76          {
77             // retrieve English phrase from user
78             string strPhrase = txtInput.Text;
79
80             // display output
81             txtOutput.Text = TranslateToPigLatin( strPhrase );
82
83          } // end method btnTranslate_Click
84
85          // translates the string input by the user
86          // from English to pig Latin
87          private string TranslateToPigLatin( string strEnglishPhrase )
88          {
89             string[] strWords; // array to hold each word
90             string strSuffix; // suffix for the end of each word
91             int intIndex; // index to iterate through the array
92             string strTemporary; // temporary string
93
94             strWords = strEnglishPhrase.Split(); // split words
95
96             for ( intIndex = 0; intIndex < strWords.Length;
97                intIndex++ )
98             {
99                // get first letter of each word
100               strTemporary =
101                  strWords[ intIndex ].Substring( 0, 1 ).ToLower();
102
103               // check if each word starts with a vowel
104               if ( strTemporary == "a" || strTemporary == "e" ||
105                  strTemporary == "i" || strTemporary == "o" ||
106                  strTemporary == "u" )
107               {
108                  strSuffix = "y";
109               }
```

```
110            else // if not, suffix is different
111            {
112               strSuffix = "ay";
113            }
114
115            // swap letters to create new word
116            strWords[ intIndex ] =
117               strWords[ intIndex ].Substring( 1 ) +
118               strTemporary + strSuffix;
119         }
120
121         // put words together and return the whole sentence
122         return String.Join( " ", strWords );
123
124      } // end method TranslateToPigLatin
125
126   } // end class FrmPigLatin
127 }
```

TUTORIAL 24

Ticket Information Application

Introducing Sequential-Access Files

Solutions

Instructor's Manual Exercise Solutions Tutorial 24

MULTIPLE-CHOICE QUESTIONS

24.1 Data maintained in a file is called _____.

a) persistent data
b) bits
c) secondary data
d) databases

24.2 Methods from the _____ class can be used to write data to a file.

a) `StreamReader`
b) `FileWriter`
c) `StreamWriter`
d) `WriteFile`

24.3 The _____ namespace provides the classes and methods that you need to use to perform file processing.

a) `System.IO`
b) `System.Files`
c) `System.Stream`
d) `System.Windows.Forms`

24.4 Sometimes a group of related files is called a _____.

a) field
b) database
c) collection
d) byte

24.5 A(n) _____ allows the user to select a file to open.

a) `CreateFileDialog`
b) `OpenFileDialog`
c) `MessageBox`
d) None of the above.

24.6 Digits, letters and special symbols are referred to as _____.

a) constants
b) `ints`
c) `strings`
d) characters

24.7 The _____ method reads a line from a file.

a) `ReadLine`
b) `Read`
c) `ReadAll`
d) `ReadToNewline`

24.8 A _____ contains information that is read in the order it was written to the file.

a) sequential-access file
b) text file
c) `StreamReader`
d) `StreamWriter`

24.9 The smallest data item that a computer can support is called a _____.

a) character set
b) character
c) special symbol
d) bit

24.10 Methods from the _____ class can be used to read data from a file.

a) `StreamWriter`
b) `FileReader`
c) `StreamReader`
d) `ReadFile`

Answers: 24.1) a. 24.2) c. 24.3) a. 24.4) b. 24.5) b. 24.6) d. 24.7) a. 24.8) a. 24.9) d. 24.10) c.

EXERCISES

24.11 (***Birthday Saver Application***) Create an application to store people's names and birthdays in a file (Fig. 24.37). The user creates a file and inputs each person's first name, last name and birthday on the `Form`. The information is then written to the file.

Figure 24.37 **Birthday Saver** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial24\Exercises\BirthdaySaver` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `BirthdaySaver.sln` in the `BirthdaySaver` directory to open the application.

c) ***Adding and customizing an `OpenFileDialog` component***. Add an `OpenFileDialog` component to the `Form`. Change its `Name` property to `objOpenFileDialog`. Set the `CheckFileExists` property to `false`. Rearrange and comment the control declaration appropriately.

d) ***Referencing namespace `System.IO`***. Reference `System.IO` to allow file processing.

e) ***Declaring a `StreamWriter` object***. Declare a `StreamWriter` object that can be used throughout the entire class.

f) ***Defining the Open File... `Button`'s `Click` event handler.*** Double click the **Open File...** `Button` to create the `btnOpen_Click` event handler. Write code to display the **Open** dialog. If the user clicks the **Cancel** `Button` in the dialog, then the event handler should perform no further actions. Otherwise, determine whether the user provided a file name that has the `.txt` extension (indicating a text file). If the user did not, display a `MessageBox` asking the user to select an appropriate file. If the user specified a valid file name, perform *Step g*.

g) ***Initializing the `StreamWriter`***. Initialize the `StreamWriter` in the event handler `btnOpenFile_Click`, passing the user-input file name as an argument. Allow the user to append information to the file by passing the `bool` value `true` as the second argument to the `StreamWriter`.

h) ***Defining the Enter `Button`'s `Click` event handler***. Double click the **Enter** `Button` to create the `btnEnter_Click` event handler. This event handler should write the entire name of the person on one line in the file. Then the person's birthday should be written on the next line in the file. Finally, the `TextBox`es on the `Form` should be cleared, and the `DateTimePicker`'s value should be set back to the current date.

i) ***Defining the Close File `Button`'s `Click` event handler***. Double click the **Close File** `Button` to create the `btnClose_Click` event handler. Close the `StreamWriter` connection in this event handler.

j) ***Running the application.*** Select **Debug > Start** to run your application. Open a file by clicking the **Open File...** `Button`. After a file has been opened, use the input fields provided to enter birthday information. After each person's name and birthday are typed in, click the **Enter** `Button`. When you are finished, close the file by clicking the **Close File** `Button`. Browse to the file and ensure that its contents contain the birthday information that you entered.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 24.11 Solution
2  // BirthdaySaver.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
```

```
10  using System.IO;
11
12  namespace BirthdaySaver
13  {
14     /// <summary>
15     /// Summary description for FrmBirthdaySaver.
16     /// </summary>
17     public class FrmBirthdaySaver : System.Windows.Forms.Form
18     {
19        // Label and TextBox to input first name
20        private System.Windows.Forms.Label lblFirstName;
21        private System.Windows.Forms.TextBox txtFirstName;
22
23        // Label and TextBox to input last name
24        private System.Windows.Forms.Label lblLastName;
25        private System.Windows.Forms.TextBox txtLastName;
26
27        // Label and DateTimePicker to choose birthday
28        private System.Windows.Forms.Label lblBirthday;
29        private System.Windows.Forms.DateTimePicker dtpBirthday;
30
31        // Button to open a file
32        private System.Windows.Forms.Button btnOpen;
33
34        // Button to put an entry into the file
35        private System.Windows.Forms.Button btnEnter;
36
37        // Button to close a file
38        private System.Windows.Forms.Button btnClose;
39
40        // OpenFileDialog to use files
41        private System.Windows.Forms.OpenFileDialog objOpenFileDialog;
42
43        /// <summary>
44        /// Required designer variable.
45        /// </summary>
46        private System.ComponentModel.Container components = null;
47
48        // StreamWriter used to write to file
49        private StreamWriter m_objOutput;
50
51        public FrmBirthdaySaver()
52        {
53           //
54           // Required for Windows Form Designer support
55           //
56           InitializeComponent();
57
58           //
59           // TODO: Add any constructor code after InitializeComponent
60           // call
61           //
62        }
63
64        /// <summary>
65        /// Clean up any resources being used.
66        /// </summary>
67        protected override void Dispose( bool disposing )
68        {
69           if( disposing )
```

```
70          {
71             if (components != null)
72             {
73                components.Dispose();
74             }
75          }
76          base.Dispose( disposing );
77       }
78
79       // Windows Form Designer generated code
80
81       /// <summary>
82       /// The main entry point for the application.
83       /// </summary>
84       [STAThread]
85       static void Main()
86       {
87          Application.Run( new FrmBirthdaySaver() );
88       }
89
90       // Open File... Button's Click event handler
91       private void btnOpen_Click(
92          object sender, System.EventArgs e )
93       {
94          // display Open dialog
95          DialogResult result = objOpenFileDialog.ShowDialog();
96
97          // exit event handler if user clicked Cancel Button
98          if ( result == DialogResult.Cancel )
99          {
100            return;
101         }
102
103         // get specified file name
104         string strFileName = objOpenFileDialog.FileName;
105
106         // show error if user specified invalid file
107         if ( strFileName.EndsWith( ".txt" ) == false )
108         {
109            MessageBox.Show( "File name must end with .txt",
110               "Invalid File Type",
111               MessageBoxButtons.OK, MessageBoxIcon.Error );
112         }
113         else
114         {
115            btnOpen.Enabled = false;
116            btnEnter.Enabled = true;
117            btnClose.Enabled = true;
118
119            // append data to file via StreamWriter
120            m_objOutput = new StreamWriter( strFileName, true );
121         }
122
123      } // end method btnOpen_Click
124
125      // clear all user input
126      private void ClearUserInput()
127      {
128         txtFirstName.Clear();
129         txtLastName.Clear();
```

```
130         dtpBirthday.Value = DateTime.Now;
131
132      } // end method ClearUserInput
133
134      // handles Enter Button's Click event
135      private void btnEnter_Click(
136         object sender, System.EventArgs e )
137      {
138         // write user input to file
139         m_objOutput.WriteLine( txtFirstName.Text + " " +
140            txtLastName.Text );
141         m_objOutput.WriteLine( dtpBirthday.Value.Month + "/" +
142            dtpBirthday.Value.Day + "/" + dtpBirthday.Value.Year );
143         ClearUserInput();
144
145      } // end method btnEnter_Click
146
147      // handles Close File Button's Click event
148      private void btnClose_Click(
149         object sender, System.EventArgs e )
150      {
151         m_objOutput.Close(); // close stream
152         btnOpen.Enabled = true;
153         btnEnter.Enabled = false;
154         btnClose.Enabled = false;
155
156      } // end method btnClose_Click
157
158   } // end class FrmBirthdaySaver
159 }
```

24.12 (***Photo Album Application***) Create an application to display images for the user, as shown in Fig. 24.38. This application should display the current image in a large `PictureBox` and display the previous and next images in smaller `PictureBox`es. A description of the book represented by the large image should be displayed in a multiline `TextBox`. The application should use the `Directory` class's methods to facilitate the displaying of the images.

Figure 24.38 **Photo Album** application GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial24\Exercises\PhotoAlbum` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `PhotoAlbum.sln` in the `PhotoAlbum` directory to open the application.

c) ***Creating instance variables.*** Create the `m_intCurrent` instance variable to represent the current image that is displayed, and set its value to `0`. Create the `m_strLargeImage` array (to store the path names of five large images), the `m_strSmallImage` array (to store the path names of five small images) and the `m_strDescriptions` array (to store the descriptions of the five books represented by the images).

d) ***Defining the `RetrieveData` method.*** Create a method named `RetrieveData` to store the path names of the larger images in `m_strLargeImages` and the path names of the smaller images in `m_strSmallImage`. Use the `Directory` class's `GetCurrentDirectory` method to determine the directory path for the `images\large` and `images\small` directories. The `books.txt` sequential-access file stores the file name of each image. The file is organized such that the file names of the small and large images are on the first line. These files have similar names. The small image's file name ends with _thumb.jpg (that is, *filename*`_thumb.jpg`), while the large image's file name ends with _large.jpg (that is, *filename*`_large.jpg`). The description of the book, which should be stored in the `m_strDescriptions` array, follows the file name.

e) ***Defining the `DisplayPicture` method.*** Create a method named `DisplayPicture` to display the current image in the large `PictureBox` and to display the previous and next images in the smaller `PictureBoxes`.

f) ***Using `if`...`else` in the `DisplayPicture` method.*** Use an `if`...`else` statement to display the images on the `Form`. If the `int` instance variable is `0`, display the image of the first book. Also, display the next book's image in the next image `PictureBox`. However, because there is no previous image, nothing should be displayed in the previous image `PictureBox`, and the **Previous Image** `Button` should be disabled. If the last image is displayed in the large `PictureBox`, then disable the **Next Image** `Button`, and do not display anything in the next image `PictureBox`. Otherwise, all three `PictureBoxes` should display their corresponding images, and the **Previous Image** and **Next Image** `Buttons` should be enabled.

g) ***Defining the `FrmPhotoAlbum_Load` event handler.*** Double click the `Form` to create the `FrmPhotoAlbum_Load` event handler. Invoke the `RetrieveData` and `DisplayPicture` methods in this event handler.

h) ***Defining the `btnPrevious_Click` event handler.*** Double click the **Previous Image** `Button` to create the `btnPrevious_Click` event handler. In this event handler, decrease the `int` instance variable by `1` and invoke method `DisplayPicture`.

i) ***Defining the `btnNext_Click` event handler.*** Double click the **Next Image** `Button` to create the `btnNext_Click` event handler. In this event handler, increment the `int` instance variable by `1` and invoke the `DisplayPicture` method.

j) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Previous Image** and **Next Image** `Buttons` to ensure that the proper images and descriptions are displayed.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 24.12 Solution
2  // PhotoAlbum.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
```

```
 9  using System.Data;
10  using System.IO;
11
12  namespace PhotoAlbum
13  {
14     /// <summary>
15     /// Summary description for FrmPhotoAlbum.
16     /// </summary>
17     public class FrmPhotoAlbum : System.Windows.Forms.Form
18     {
19        // PictureBox and TextBox to display a picture and its
20        // description
21        private System.Windows.Forms.PictureBox picMain;
22        private System.Windows.Forms.TextBox txtDescription;
23
24        // PictureBox to display and Button to move to the previous
25        // picture
26        private System.Windows.Forms.PictureBox picPrevious;
27        private System.Windows.Forms.Button btnPrevious;
28
29        // PictureBox to display and Button to move to the next
30        // picture
31        private System.Windows.Forms.PictureBox picNext;
32        private System.Windows.Forms.Button btnNext;
33
34        /// <summary>
35        /// Required designer variable.
36        /// </summary>
37        private System.ComponentModel.Container components = null;
38
39        // represents current image's index
40        private int m_intCurrent = 0;
41        private string[] m_strLargeImage = new string[ 6 ];
42        private string[] m_strSmallImage = new string[ 6 ];
43        private string[] m_strDescriptions = new string[ 6 ];
44
45        public FrmPhotoAlbum()
46        {
47           //
48           // Required for Windows Form Designer support
49           //
50           InitializeComponent();
51
52           //
53           // TODO: Add any constructor code after InitializeComponent
54           // call
55           //
56        }
57
58        /// <summary>
59        /// Clean up any resources being used.
60        /// </summary>
61        protected override void Dispose( bool disposing )
62        {
63           if( disposing )
64           {
65              if (components != null)
66              {
67                 components.Dispose();
68              }
```

```
69          }
70          base.Dispose( disposing );
71       }
72
73       // Windows Form Designer generated code
74
75       /// <summary>
76       /// The main entry point for the application.
77       /// </summary>
78       [STAThread]
79       static void Main()
80       {
81          Application.Run( new FrmPhotoAlbum() );
82       }
83
84       // handles Form's Load event
85       private void FrmPhotoAlbum_Load(
86          object sender, System.EventArgs e )
87       {
88          RetrieveData();
89          DisplayPicture(); // display first image
90
91       } // end method FrmPhotoAlbum_Load
92
93       // handles Previous Image Button's Click event
94       private void btnPrevious_Click(
95          object sender, System.EventArgs e )
96       {
97          m_intCurrent--;
98          DisplayPicture(); // display new image
99
100      } // end method btnPrevious_Click
101
102      // handles Next Image Button's click event
103      private void btnNext_Click(
104         object sender, System.EventArgs e )
105      {
106         m_intCurrent++;
107         DisplayPicture(); // display new image
108      } // end method btnNext_Click
109
110      // extract descriptions from file and images from the directory
111      private void RetrieveData()
112      {
113         // create directory path for large images
114         string strLargeDirectory =
115            Directory.GetCurrentDirectory() + "\images\large\\";
116
117         // create directory path for small images
118         string strSmallDirectory =
119            Directory.GetCurrentDirectory() + "\images\small\\";
120
121         // initialize StreamReader to read lines from file
122         StreamReader objInput = new StreamReader( "books.txt" );
123
124         string strImageName;
125         int intCounter = 0;
126
127         // loop through lines in file
128         while ( objInput.Peek() != -1 )
```

```
129          {
130             strImageName = objInput.ReadLine();
131             m_strLargeImage[ intCounter ] =
132                strLargeDirectory + strImageName + "_large.jpg";
133             m_strSmallImage[ intCounter ] =
134                strSmallDirectory + strImageName + "_thumb.jpg";
135             m_strDescriptions[ intCounter ] = objInput.ReadLine();
136
137             intCounter++;
138          }
139
140       } // end method RetrieveData
141
142       // displays images
143       private void DisplayPicture()
144       {
145          // set main image
146          picMain.Image =
147             Image.FromFile( m_strLargeImage[ m_intCurrent ] );
148
149          // if index is 0 (first image), do not show previous image
150          if ( m_intCurrent == 0 )
151          {
152             picPrevious.Image = null; // do not show previous image
153
154             // preview next image
155             picNext.Image =
156                Image.FromFile( m_strSmallImage[ m_intCurrent + 1 ] );
157
158             btnPrevious.Enabled = false; // disable Previous Button
159          }
160
161          // if index corresponds to last item in array,
162          // do not show next image
163          else if ( m_intCurrent ==
164             m_strLargeImage.GetUpperBound( 0 ) )
165          {
166             picPrevious.Image =
167                Image.FromFile( m_strSmallImage[ m_intCurrent - 1 ] );
168
169             picNext.Image = null; // do not show Next image
170             btnNext.Enabled = false; // disable Next Button
171          }
172
173          // show previous, current and next image
174          else
175          {
176             picPrevious.Image =
177                Image.FromFile( m_strSmallImage[ m_intCurrent - 1 ] );
178
179             picNext.Image =
180                Image.FromFile( m_strSmallImage[ m_intCurrent + 1 ] );
181
182             // enable Buttons
183             btnPrevious.Enabled = true;
184             btnNext.Enabled = true;
185          }
186
187          // set description
188          txtDescription.Text = m_strDescriptions[ m_intCurrent ];
```

```
189
190       } // end method DisplayPicture
191
192    } // end class FrmPhotoAlbum
193 }
```

24.13 (*Car Reservation Application*) Create an application to allows a user to reserve a car for the specified day (Fig. 24.39). The small car reservation company can rent out only four cars per day. Let the application allow the user to specify a certain day. If four cars have already been reserved for that day, then indicate to the user that no vehicles are available.

Figure 24.39 CarReservation application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial24\Exercises\CarReservation` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CarReservation.sln` in the `CarReservation` directory to open the application.

c) ***Adding a `MonthCalendar` control to the `Form`.*** Drag and drop a `MonthCalendar` control on the `Form`. Set the `Location` property of the control to `16, 32`. Rearrange and comment the new control declaration appropriately.

d) ***Referencing `System.IO` namespace.*** Reference the `System.IO` namespace to allow file processing.

e) ***Defining the `FrmReserve_Load` event handler.*** Double click the `Form` to create the `FrmReserve_Load` event handler.

f) ***Defining the `NumberOfReservations` method.*** Create a method named `NumberOfReservations` that takes one argument of the `DateTime` type. The method should create a `StreamReader` that reads from the `reservations.txt` file. Use a `while` statement to allow the `StreamReader` to search through the entire `reservations.txt` file to see how many cars have been rented for the day selected by the user. The method should close the `StreamReader` connection and return the number of cars rented for the day selected.

g) ***Defining the `CheckReservations` method.*** Create a method named `CheckReservations`. This method should invoke the `NumberOfReservations` method, passing it the user-selected day as an argument. The `CheckReservations` method should then retrieve the number returned by `NumberOfReservations` and determine if four cars have been rented for that day. If four cars have been rented, display a message dialog to the user stating that no cars are available that day for rental. If fewer than four cars have been rented for that day, create a `StreamWriter` object, passing `reservations.txt` as the first argument and `true` as the second argument. Write the day and the user's name to the `reservations.txt` file, and display a message dialog to the user stating that a car has been reserved.

h) ***Defining the `btnReserve_Click` event handler.*** Double click the **Reserve Car** `Button` to create the `btnReserve_Click` event handler. In this event handler, invoke the `CheckReservations` method and clear the **Name:** `TextBox`.

i) ***Running the application.*** Select **Debug > Start** to run your application. Enter several reservations, including four reservations for the same day. Enter a reservation for a day that already has four reservations to ensure that a message dialog will be displayed.

j) ***Closing the application.*** Close your running application by clicking its close box. Open `reservations.txt` to ensure that the proper data has been stored (based on the reservations entered in *Step i*).

k) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 24.13 Solution
// CarReservation.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;

namespace CarReservation
{
   /// <summary>
   /// Summary description for FrmCarReservation.
   /// </summary>
   public class FrmCarReservation : System.Windows.Forms.Form
   {
      // Label and MonthCalendar to select date of reservation
      private System.Windows.Forms.Label lblSelect;
      private System.Windows.Forms.MonthCalendar mvwDate;

      // Label and TextBox to input name
      private System.Windows.Forms.Label lblName;
      private System.Windows.Forms.TextBox txtName;

      // Button to reserve a car
      private System.Windows.Forms.Button btnReserve;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCarReservation()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
```

```
52          {
53             if( disposing )
54             {
55                if (components != null)
56                {
57                   components.Dispose();
58                }
59             }
60             base.Dispose( disposing );
61          }
62
63          // Windows Form Designer generated code
64
65          /// <summary>
66          /// The main entry point for the application.
67          /// </summary>
68          [STAThread]
69          static void Main()
70          {
71             Application.Run( new FrmCarReservation() );
72          }
73
74          // method to determine number of cars rented for that day
75          private int NumberOfReservations( DateTime objDay )
76          {
77             // set to the value of selected day in month view
78             int intChosenDay = objDay.Day;
79             string intFileDay;
80             int intCars = 0;
81
82             // StreamReader reads lines from file to strLine
83             StreamReader objFile;
84             string strLine;
85
86             // initialize StreamReader
87             objFile = new StreamReader( "reservations.txt" );
88
89             // loop through all file data
90             while ( objFile.Peek() != -1 )
91             {
92                strLine = objFile.ReadLine();
93                intFileDay = strLine;
94
95                // if days match, increment count
96                if ( intFileDay == intChosenDay.ToString() )
97                {
98                   intCars++;
99                }
100
101               // read name in
102               objFile.ReadLine();
103            }
104
105            objFile.Close();
106
107            return intCars;
108
109         } // end method NumberOfReservations
110
111         // method to check reservations for chosen day
```

```
112       private void CheckReservations()
113       {
114          int intReservations;
115
116          // gets data for selected day and stores in intReservations
117          intReservations = NumberOfReservations(
118             mvwDate.SelectionStart );
119
120          // determine if user can reserve car that day
121          if ( intReservations >= 4 )
122          {
123             MessageBox.Show( "Sorry, all cars have been reserved" +
124                " for this day. Please select another day.",
125                "No cars available.",
126                MessageBoxButtons.OK, MessageBoxIcon.Information );
127          }
128          else
129          {
130             // create StreamWriter to write to file
131             StreamWriter objWrite =
132                new StreamWriter( "Reservations.txt", true );
133
134             objWrite.WriteLine( mvwDate.SelectionStart.Day );
135             objWrite.WriteLine( txtName.Text );
136             objWrite.Close();
137
138             MessageBox.Show( "A car has been reserved for you",
139                "Car reserved", MessageBoxButtons.OK,
140                MessageBoxIcon.Information );
141          }
142
143       } // end method CheckReservations
144
145       // invoke when Reserve Button is clicked
146       private void btnReserve_Click(
147          object sender, System.EventArgs e )
148       {
149          // determine if name was provided
150          if ( txtName.Text == "" )
151          {
152             MessageBox.Show( "You must provide a name",
153                "Name Required", MessageBoxButtons.OK,
154                MessageBoxIcon.Information );
155          }
156          else
157          {
158             CheckReservations();
159             txtName.Clear();
160          }
161
162       } // end method btnReserve_Click
163
164    } // end class FrmCarReservation
165 }
```

What does this code do? ▶ **24.14** What is the result of the following code?

```
string strPath1 = "oldfile.txt";
string strPath2 = "newfile.txt";
string strLine;

StreamWriter objStreamWriter;
objStreamWriter = new StreamWriter( strPath2 );

StreamReader objStreamReader;
objStreamReader = new StreamReader( strPath1 );

while ( objStreamReader.Peek() > -1 )
{
   strLine = objStreamReader.ReadLine();
   objStreamWriter.WriteLine( strLine );
}

objStreamWriter.Close();
objStreamReader.Close();
```

Answer: The code copies the contents of `oldfile.txt` to `newfile.txt`. After the file transfer is completed, the files are closed by closing the `StreamReader` and `StreamWriter`. Note that if a blank line is encountered, copying stops at the blank line. The complete code reads:

```
// Exercise 24.14 Solution
// CopyFile.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;

namespace CopyFile
{
   /// <summary>
   /// Summary description for FrmCopyFile.
   /// </summary>
   public class FrmCopyFile : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblCopy;
      private System.Windows.Forms.Label lblOriginal;
      private System.Windows.Forms.TextBox txtCopy;
      private System.Windows.Forms.TextBox txtOriginal;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmCopyFile()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
```

```
37            // call
38            //
39         }
40
41         /// <summary>
42         /// Clean up any resources being used.
43         /// </summary>
44         protected override void Dispose( bool disposing )
45         {
46            if( disposing )
47            {
48               if (components != null)
49               {
50                  components.Dispose();
51               }
52            }
53            base.Dispose( disposing );
54         }
55
56         // Windows Form Designer generated code
57
58         /// <summary>
59         /// The main entry point for the application.
60         /// </summary>
61         [STAThread]
62         static void Main()
63         {
64            Application.Run( new FrmCopyFile() );
65         }
66
67         // copies newfile.txt to oldfile.txt and displays both
68         private void FrmCopyFile_Load(
69            object sender, System.EventArgs e )
70         {
71            string strPath1 = "oldfile.txt";
72            string strPath2 = "newfile.txt";
73            string strLine;
74
75            StreamWriter objStreamWriter;
76            objStreamWriter = new StreamWriter( strPath2 );
77
78            StreamReader objStreamReader;
79            objStreamReader = new StreamReader( strPath1 );
80
81            while ( objStreamReader.Peek() > -1 )
82            {
83               strLine = objStreamReader.ReadLine();
84               objStreamWriter.WriteLine( strLine );
85            }
86
87            objStreamWriter.Close();
88            objStreamReader.Close();
89
90            // display both files in TextBoxes
91            objStreamReader = new StreamReader( strPath1 );
92
93            while ( objStreamReader.Peek() > -1 )
94            {
95               strLine = objStreamReader.ReadLine();
96               txtCopy.Text += strLine; + "\r\n";
```

```
97              txtOriginal.Text += strLine + "\r\n";
98           }
99
100          objStreamWriter.Close();
101          objStreamReader.Close();
102
103       } // end method FrmCopyFile_Load
104
105    } // end class FrmCopyFile
106 }
```

What's wrong with this code?

24.15 Find the error(s) in the following code, which is supposed to read a line from `somefile.txt`, convert the line to uppercase and then append it to `somefile.txt`.

```
1  string strPath = "somefile.txt";
2  string strContents;
3
4  StreamWriter objStreamWriter;
5  objStreamWriter = new StreamWriter( strPath, true );
6
7  StreamReader objStreamReader;
8  objStreamReader = new StreamReader( strPath );
9
10 strContents = objStreamReader.ReadLine();
11
12 strContents = strContents.ToUpper();
13
14 objStreamWriter.Write( strContents );
15
16 objStreamReader.Close();
17 objStreamWriter.Close();
```

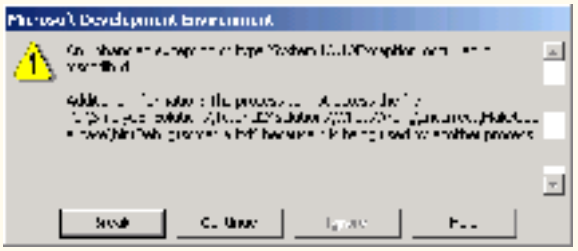
Answer: Once the `StreamWriter` opens the file specified by `strPath`, the file is marked open. While open, no other object can open the file. Thus a run-time error occurs when `StreamReader` tries to open the same file. The complete incorrect code reads:

```
1  // Exercise 24.15 Solution
2  // MakeUppercase.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.IO;
11
12 namespace MakeUppercase
```

```
13  {
14     /// <summary>
15     /// Summary description for FrmMakeUppercase.
16     /// </summary>
17     public class FrmMakeUppercase : System.Windows.Forms.Form
18     {
19        private System.Windows.Forms.Label lblResult;
20        private System.Windows.Forms.TextBox txtDisplay;
21        /// <summary>
22        /// Required designer variable.
23        /// </summary>
24        private System.ComponentModel.Container components = null;
25
26        public FrmMakeUppercase()
27        {
28           //
29           // Required for Windows Form Designer support
30           //
31           InitializeComponent();
32
33           //
34           // TODO: Add any constructor code after InitializeComponent
35           // call
36           //
37        }
38
39        /// <summary>
40        /// Clean up any resources being used.
41        /// </summary>
42        protected override void Dispose( bool disposing )
43        {
44           if( disposing )
45           {
46              if (components != null)
47              {
48                 components.Dispose();
49              }
50           }
51           base.Dispose( disposing );
52        }
53
54        // Windows Form Designer generated code
55
56        /// <summary>
57        /// The main entry point for the application.
58        /// </summary>
59        [STAThread]
60        static void Main()
61        {
62           Application.Run( new FrmMakeUppercase() );
63        }
64
65        // converts a line from somefile.txt to uppercase and
66        // appends it to the end of the file
67        private void FrmMakeUppercase_Load(
68           object sender, System.EventArgs e )
69        {
70           string strPath = "somefile.txt";
71           string strContents;
72
```

A file cannot be opened for both reading and writing at the same time

```
73          StreamWriter objStreamWriter;
74          objStreamWriter = new StreamWriter( strPath, true );
75
76          StreamReader objStreamReader;
77          objStreamReader = new StreamReader( strPath );
78
79          strContents = objStreamReader.ReadLine();
80
81          strContents = strContents.ToUpper();
82
83          objStreamWriter.Write( strContents );
84
85          objStreamReader.Close();
86          objStreamWriter.Close();
87
88          // display the results
89          objStreamReader = new StreamReader( strPath );
90
91          strContents = objStreamReader.ReadLine();
92          txtDisplay.Text += strContents + "\t";
93
94          strContents = strContents.ToUpper();
95          txtDisplay.Text += strContents;
96
97          objStreamReader.Close();
98
99       } // end method FrmMakeUppercase_Load
100
101   } // end class FrmMakeUppercase
102 }
```



Answer: The complete corrected code should read:

```
1  // Exercise 25.15 Solution
2  // MakeUppercase.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.IO;
11
12 namespace MakeUppercase
13 {
14    /// <summary>
15    /// Summary description for FrmMakeUppercase.
16    /// </summary>
```

```
17    public class FrmMakeUppercase : System.Windows.Forms.Form
18    {
19       private System.Windows.Forms.Label lblResult;
20       private System.Windows.Forms.TextBox txtDisplay;
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.Container components = null;
25
26       public FrmMakeUppercase()
27       {
28          //
29          // Required for Windows Form Designer support
30          //
31          InitializeComponent();
32
33          //
34          // TODO: Add any constructor code after InitializeComponent
35          // call
36          //
37       }
38
39       /// <summary>
40       /// Clean up any resources being used.
41       /// </summary>
42       protected override void Dispose( bool disposing )
43       {
44          if( disposing )
45          {
46             if (components != null)
47             {
48                components.Dispose();
49             }
50          }
51          base.Dispose( disposing );
52       }
53
54       // Windows Form Designer generated code
55
56       /// <summary>
57       /// The main entry point for the application.
58       /// </summary>
59       [STAThread]
60       static void Main()
61       {
62          Application.Run( new FrmMakeUppercase() );
63       }
64
65       // converts a line from somefile.txt to uppercase and
66       // appends it to the end of the file
67       private void FrmMakeUppercase_Load(
68          object sender, System.EventArgs e )
69       {
70          string strPath = "somefile.txt";
71          string strContents;
72
73          StreamWriter objStreamWriter;
74          StreamReader objStreamReader;
75
76          objStreamReader = new StreamReader( strPath );
```

```
77
78          strContents = objStreamReader.ReadLine();
79          strContents = strContents.ToUpper();
80
81          objStreamReader.Close();
82
83          objStreamWriter = new StreamWriter( strPath, true );
84
85          objStreamWriter.Write( strContents );
86
87          objStreamWriter.Close();
88
89          // display the results
90          objStreamReader = new StreamReader( strPath );
91
92          strContents = objStreamReader.ReadLine();
93          txtDisplay.Text += strContents + "\t";
94
95          strContents = strContents.ToUpper();
96          txtDisplay.Text += strContents;
97
98          objStreamReader.Close();
99
100       } // end method FrmMakeUppercase_Load
101
102    } // end class FrmMakeUppercase
103 }
```

Programming Challenge

24.16 (***File Scrape Application***) Create an application, similar to the screen scraping application of Tutorial 23, that opens a user-specified file and searches the file for the price of a book, returning it to the user (Fig. 24.40). [*Hints*: You will need to use the `ReadToEnd` method of the `StreamReader` class to retrieve the entire contents of the files. The book price appears, for example, in the sample `booklist.htm` file as `Our Price: <b>$59.99</b>`.]

Figure 24.40 **File Scrape** application GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial24\Exercises\FileScrape` to your `C:\SimplyCSP` directory. Notice that two HTML files—`booklist.htm` and `bookpool.htm`—are provided for you.

b) ***Opening the application's template file.*** Double click `FileScrape.sln` in the `FileScrape` directory to open the application.

c) ***Adding and customizing an `OpenFileDialog` component.*** Add an `OpenFileDialog` component to the `Form`. Change its `Name` property to `objOpenFileDialog`. Set the `CheckFileExists` property to `false`. Rearrange and comment the control declaration appropriately.

d) ***Creating an event handler.*** Create an event handler for the **Open...** `Button` that allows the user to select a file to search for prices.

e) ***Creating a second event handler.*** Create an event handler for the **Search** `Button`. This event handler should search the specified HTML file for the book price. When the price is found, display it in the output `Label`.

f) ***Running the application.*** Select **Debug > Start** to run your application. Click the **Open**... `Button` and select one of the `.htm` files provided in the `FileScrape` directory. Click the **Search** `Button` and view the price of the book. For `booklist.htm`, the price should be $59.99, and for `bookpool.htm`, 1the price should be $39.50.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 24.16 Solution
2  // FileScrape.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.IO;
11
12 namespace FileScrape
13 {
14    /// <summary>
15    /// Summary description for FrmFileScrape.
16    /// </summary>
17    public class FrmFileScrape : System.Windows.Forms.Form
18    {
19       // Label and TextBox to displays the file path
20       private System.Windows.Forms.Label lblPath;
21       private System.Windows.Forms.TextBox txtPath;
22
23       // Label and TextBox that displays the book price
24       // found in the HTML file
25       private System.Windows.Forms.Label lblResult;
26       private System.Windows.Forms.TextBox txtPrice;
27
28       // Button to open an HTML file
29       private System.Windows.Forms.Button btnOpen;
30
31       // Button to search the HTML file for the book price
32       private System.Windows.Forms.Button btnSearch;
33
34       // OpenFileDialog to open HTML files
35       private System.Windows.Forms.OpenFileDialog objOpenFileDialog;
36
37       /// <summary>
38       /// Required designer variable.
39       /// </summary>
40       private System.ComponentModel.Container components = null;
41
42       public FrmFileScrape()
43       {
44          //
45          // Required for Windows Form Designer support
46          //
47          InitializeComponent();
```

```
48
49          //
50          // TODO: Add any constructor code after InitializeComponent
51          // call
52          //
53       }
54
55       /// <summary>
56       /// Clean up any resources being used.
57       /// </summary>
58       protected override void Dispose( bool disposing )
59       {
60          if( disposing )
61          {
62             if (components != null)
63             {
64                components.Dispose();
65             }
66          }
67          base.Dispose( disposing );
68       }
69
70       // Windows Form Designer generated code
71
72       /// <summary>
73       /// The main entry point for the application.
74       /// </summary>
75       [STAThread]
76       static void Main()
77       {
78          Application.Run( new FrmFileScrape() );
79       }
80
81       // display OpenFile dialog when Open... Button is clicked
82       private void btnOpen_Click(
83          object sender, System.EventArgs e )
84       {
85          DialogResult result;
86
87          // show dialog to user for selecting file
88          result = objOpenFileDialog.ShowDialog();
89
90          if ( result == DialogResult.Cancel )
91          {
92             return; // if user cancels, do nothing
93          }
94          else
95          {
96             // store file name that user selected
97             txtPath.Text = objOpenFileDialog.FileName;
98          }
99
100      } // end method btnOpen_Click
101
102      private void btnSearch_Click(
103         object sender, System.EventArgs e )
104      {
105         string strFilePath;
106         int intMatchLocation;
107         int intRankBegin;
```

```
108         int intRankEnd;
109
110         // if TextBox is empty, show an error message and return
111         if ( txtPath.Text == "" )
112         {
113            MessageBox.Show( "No path selected",
114               "Path Not Chosen", MessageBoxButtons.OK,
115               MessageBoxIcon.Exclamation );
116            return;
117         }
118         else
119         {
120            // set filepath to value in txtPath
121            strFilePath = txtPath.Text;
122         }
123
124         // create stream reader to open and read text file
125         StreamReader objReader = new StreamReader( strFilePath );
126
127         // read file from beginning to end
128         string strContents = objReader.ReadToEnd();
129
130         // close the file to free resource
131         objReader.Close();
132
133         // find locations of text in file
134         intMatchLocation = strContents.IndexOf( "Our Price:", 0 );
135         intRankBegin = strContents.IndexOf( "$", intMatchLocation );
136         intRankEnd = strContents.IndexOf( "<", intRankBegin );
137
138         // extract price from file
139         txtPrice.Text = strContents.Substring( intRankBegin,
140            ( intRankEnd - intRankBegin ) );
141
142      } // end method btnSearch_Click
143
144   } // end class FrmFileScrape
145 }
```

25 TUTORIAL

ATM Application

Introducing Database Programming

Solutions

Instructor's Manual Exercises for Tutorial 25

[*Note:* The solutions for this tutorial use relative paths when connecting to a database. We have done this so that the solutions can be run from any location. The student has only learned to connect to databases using absolute paths, so their solutions must be run from a specific location.]

MULTIPLE-CHOICE QUESTIONS

25.1 A _____ provides mechanisms for storing and organizing data in a manner that is consistent with a database's format.

a) relational database
b) connection object
c) data command
d) database management system

25.2 In Microsoft Access, an entire row in a database table is known as a _____.

a) record
b) field
c) column
d) primary key

25.3 A primary key is used to _____.

a) create rows in a database
b) identify fields in a database
c) distinguish between rows in a table
d) read information from a database

25.4 A data command object allows you to _____.

a) connect to a database
b) read information from a database
c) execute a command to retrieve or modify database data
d) create a database

25.5 A data reader can _____.

a) retrieve information from a database
b) modify information stored in a database
c) establish a connection to a database
d) close a connection to a database

25.6 In a `SELECT` statement, what follows the `SELECT` keyword?

a) the name of the table
b) the name of the field(s)
c) the name of the database
d) the criteria that the row must meet

25.7 What does the following `SELECT` statement do?

```
SELECT Age FROM People WHERE LastName = 'Purple'
```

a) It selects the age of the person (or people) with the last name `Purple` from the `People` table.
b) It selects the value `Purple` from the `Age` table of the `People` database.
c) It selects the age of the person with the last name `Purple` from the `People` database.
d) It selects the `People` field from the `Age` table with the `LastName` value `Purple`.

25.8 The SQL _____ modifies information in a database.

a) `SELECT` statement
b) `MODIFY` statement
c) `CHANGE` statement
d) `UPDATE` statement

25.9 Assuming the account number is `2`, which of the following statements modifies the PIN field in the `Accounts` table?

a) `SELECT PIN FROM Accounts WHERE AccountNumber = 2`
b) `SELECT Accounts FROM AccountNumber = 2 WHERE PIN`
c) `UPDATE Accounts SET PIN=1243 WHERE AccountNumber = 2`
d) `UPDATE PIN=1243 SET AccountNumber = 2 WHERE Accounts`

25.10 A _____ is an organized collection of data.

a) row
b) database
c) data reader
d) primary key

Answers: 25.1) d. 25.2) a. 25.3) c. 25.4) c. 25.5) a. 25.6) b. 25.7) a. 25.8) d. 25.9) c. 25.10) b.

EXERCISES

25.11 ***(Stock Portfolio Application)*** A stockbroker wants an application that will display a client's stock portfolio (Fig. 25.38). All the companies that the user holds stock in should be displayed in a `ComboBox` when the application is loaded. When the user selects a company from the `ComboBox` and clicks the **Stock Information** `Button`, the stock information for that company should be displayed in `Label`s.

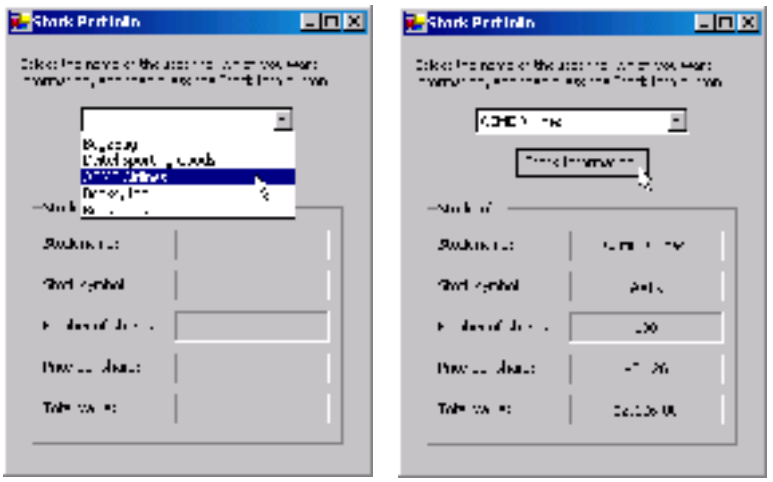


Figure 25.38 **Stock Portfolio** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial25\Exercises\StockPortfolio` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `StockPortfolio.sln` in the `StockPortfolio` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `stocks.mdb` database from your `C:\Examples\Tutorial25\Exercises\Databases` directory to your `C:\SimplyCSP\StockPortfolio` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `stock.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the `Form`.*** Add two command objects to the `Form`, and set both their `Connection` properties to the database connection object. Name the command objects `objSelectStockNameCommand` and `objSelectStockInformationCommand`. The first object will be used to retrieve the name of a stock, and the second item will be used to retrieve all of a stock's information, based on the name of the stock. Rearrange and comment the control declarations appropriately.

f) ***Setting the command objects' `CommandText` properties.*** Select the `objSelectStockNameCommand` object, then click the ellipsis `Button` that appears to the right of the `CommandText` property in the **Properties** window. In the **Query Builder**, select **stockName** from the **stocks** table and click **OK**. Select the `objSelectStockInformationCommand`, and open the **Query Builder**, as you did for `objSelectStockNameCommand`. This time, select the **stockSymbol**, **shares** and **price** items from the **stocks** table. Then, select the **stockName** item and provide it with the **=?** criteria value. Finally, uncheck the **stockName** item from the **stocks** table. Click **OK** to dismiss the **Query Builder**.

g) ***Adding a `Load` event to the `Form`.*** Add a `Load` event handler for the `Form`. Add code to this event handler to open a connection to the database. Use the `objSelectStockNameCommand` to retrieve the `StockNames`, and add them to the `ComboBox`.

h) ***Adding a `Click` event handler for the `btnStockInformation` `Button`.*** Add a `Click` event handler for the **Stock Information** `Button`. Add code to the event handler that passes the `SelectedItem` to the `StockData` method as a `string`. Then, close the connection.

i) ***Defining the `StockData` method.*** Create a `StockData` method that takes a `string` representing the name of the stock as an argument. Connect to the database, and

retrieve the information for the stock passed as an argument. Display the information in the corresponding `Labels` and close the connection to the database. Call the `ComputeTotalValueString` method, which you define in the next step, to calculate the total value.

j) ***Defining the `ComputeTotalValueString` method.*** Create the `ComputeTotalValueString` method to compute the total value by multiplying the number of shares by the price per share.

k) ***Running the application.*** Select **Debug > Start** to run your application. Select various stock companies from the `ComboBox`, and click the **Stock Information** `Button` to display a company's information.

l) ***Closing the application.*** Close your running application by clicking its close box.

m) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 25.11 Solution
2  // StockPortfolio.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.Data.OleDb;
11
12 namespace StockPortfolio
13 {
14    /// <summary>
15    /// Summary description for FrmStockPortfolio.
16    /// </summary>
17    public class FrmStockPortfolio : System.Windows.Forms.Form
18    {
19       // Label to display directions to the user
20       private System.Windows.Forms.Label lblPrompt;
21
22       // ComboBox to choose a stock name
23       private System.Windows.Forms.ComboBox cboStockNames;
24
25       // Button to retrieve stock information
26       private System.Windows.Forms.Button btnStockInformation;
27
28       // GroupBox containing stock information, including name,
29       // symbol, number of shares, price per share and total value
30       private System.Windows.Forms.GroupBox stockInfoGroupBox;
31       private System.Windows.Forms.Label lblName;
32       private System.Windows.Forms.Label lblNameOutput;
33       private System.Windows.Forms.Label lblSymbol;
34       private System.Windows.Forms.Label lblSymbolOutput;
35       private System.Windows.Forms.Label lblShareNumber;
36       private System.Windows.Forms.Label lblShareNumberOutput;
37       private System.Windows.Forms.Label lblSharePrice;
38       private System.Windows.Forms.Label lblSharePriceOutput;
39       private System.Windows.Forms.Label lblTotal;
40       private System.Windows.Forms.Label lblTotalOutput;
41
42       // OleDbConnection to connect to stocks database
43       private System.Data.OleDb.OleDbConnection objOleDbConnection;
44
45       // OleDbCommand to retrieve a stock name from the database
46       private System.Data.OleDb.OleDbCommand
```

```
47          objSelectStockNameCommand;
48
49       // OleDbCommand to retrieve stock information from the database
50       // given a stock name
51       private System.Data.OleDb.OleDbCommand
52          objSelectStockInformationCommand;
53
54       /// <summary>
55       /// Required designer variable.
56       /// </summary>
57       private System.ComponentModel.Container components = null;
58
59       public FrmStockPortfolio()
60       {
61          //
62          // Required for Windows Form Designer support
63          //
64          InitializeComponent();
65
66          //
67          // TODO: Add any constructor code after InitializeComponent
68          // call
69          //
70       }
71
72       /// <summary>
73       /// Clean up any resources being used.
74       /// </summary>
75       protected override void Dispose( bool disposing )
76       {
77          if( disposing )
78          {
79             if (components != null)
80             {
81                components.Dispose();
82             }
83          }
84          base.Dispose( disposing );
85       }
86
87       // Windows Form Designer generated code
88
89       /// <summary>
90       /// The main entry point for the application.
91       /// </summary>
92       [STAThread]
93       static void Main()
94       {
95          Application.Run( new FrmStockPortfolio() );
96       }
97
98       // helper method to compute total value of stock
99       // and return it as a string
100      private string ComputeTotalValueString(
101         string strShareNumber, string strSharePrice )
102      {
103         int intShareNumber = Int32.Parse( strShareNumber );
104         decimal decSharePrice = Decimal.Parse( strSharePrice );
105
106         // return the total value formatted as a currency
```

```
107          return String.Format( "{0:C}",
108             ( intShareNumber * decSharePrice ) );
109
110       } // end method ComputeTotalValueString
111
112       // handles Form Load event
113       private void FrmStockPortfolio_Load(
114          object sender, System.EventArgs e )
115       {
116          objOleDbConnection.Open(); // open connection
117
118          // create data reader to read from database
119          OleDbDataReader objReader;
120
121          objReader = objSelectStockNameCommand.ExecuteReader();
122
123          while ( objReader.Read() )
124          {
125             // add all stock names in database to the ComboBox
126             cboStockNames.Items.Add( objReader[ "stockName" ] );
127          }
128
129          objOleDbConnection.Close(); // close database connection
130
131       } // end method FrmStockPortfolio_Load
132
133       // handles Stock Information Button's Click event
134       private void btnStockInformation_Click(
135          object sender, System.EventArgs e )
136       {
137          // string representing stock selected
138          string strSelection =
139             Convert.ToString( cboStockNames.SelectedItem );
140
141          // display stock information
142          StockData( strSelection );
143
144       } // end method btnStockInformation_Click
145
146       // retrieve stock information from database
147       // and set corresponding output Labels with data
148       private void StockData( string strStock )
149       {
150          objOleDbConnection.Open(); // open connection
151
152          // specify which stock's information to retrieve
153          objSelectStockInformationCommand.Parameters[
154             "stockName" ].Value = strStock;
155
156          // create data reader to read from database
157          OleDbDataReader objReader;
158
159          objReader =
160             objSelectStockInformationCommand.ExecuteReader();
161          objReader.Read(); // read from the database
162
163          // set all output Labels
164          // with corresponding stock information
165          lblNameOutput.Text = strStock;
166          lblSymbolOutput.Text =
```

```
            Convert.ToString( objReader[ "stockSymbol" ] );

         lblShareNumberOutput.Text =
            Convert.ToString( objReader[ "shares" ] );

         decimal decSharePrice =
            Convert.ToDecimal( objReader[ "price" ] );

         lblSharePriceOutput.Text =
            String.Format( "{0:C}", decSharePrice );

         objReader.Close(); // close OleDbDataReader

         lblTotalOutput.Text = ComputeTotalValueString(
            lblShareNumberOutput.Text,
            Convert.ToString( decSharePrice ) );

         objOleDbConnection.Close(); // close database connection

      } // end method StockData

   } // end class FrmStockPortfolio
}
```

25.12 ***(Restaurant Bill Calculator Application)*** A restaurant wants you to develop an application that calculates a table's bill (Fig. 25.39). The application should display all the menu items from the restaurant's database in four `ComboBoxes`. Each `ComboBox` should contain a category of food offered by the restaurant (**Beverage**, **Appetizer**, **Main course** and **Dessert**). The user can choose from one of these `ComboBoxes` to add an item to a table's bill. When the table is finished ordering, the user can click the **Calculate Bill** `Button` to display the **Subtotal:**, **Tax:** and **Total:** for the table.

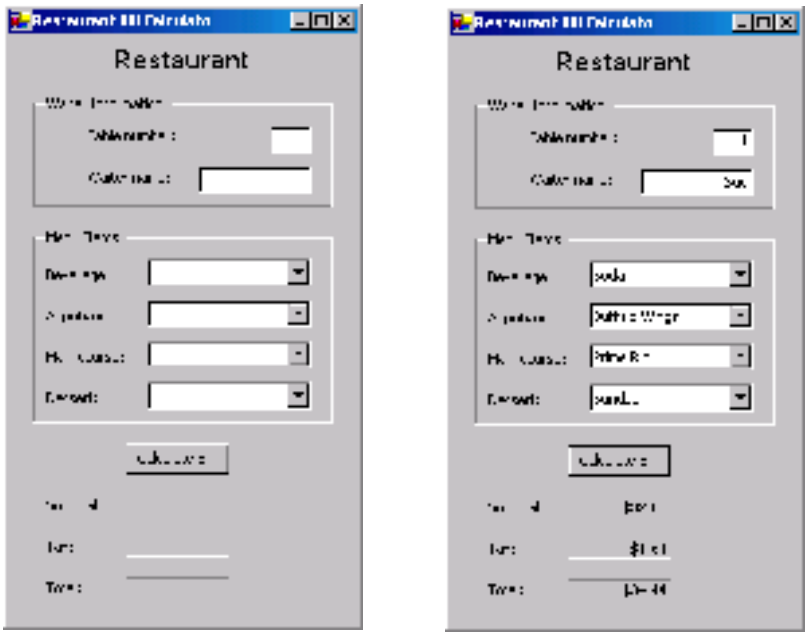


Figure 25.39 **Restaurant Bill Calculator** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial25\Exercises\RestaurantBillCalculator` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `RestaurantBillCalculator.sln` in the `RestaurantBillCalculator` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `menu.mdb` database from `C:\Examples\Tutorial25\Exercises\Databases` to your `C:\SimplyCSP\RestaurantBillCalculator` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `menu.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the `Form`.*** Add two command objects to the `Form`, and set both their `Connection` properties to the database connection object. Name the command objects `objSelectNameCommand` and `objSelectPriceCommand`. The first object will be used to retrieve the name of a menu item, based on category (for example, appetizer). The second command object will be used to retrieve a menu item's price, based on the item's name. Rearrange and comment the control declarations appropriately.

f) ***Setting the command objects' `CommandText` properties.*** Select the `objSelectNameCommand` object, and open the **Query Builder**. Add the **menu** table and select the **name** and **category** items. Provide the **category** item with the **=?** criteria value, and deselect **category** in the **menu** table. Click **OK**. Select the `objSelectPriceCommand`, and open the **Query Builder**. This time, select the items marked **price** and **name** from the **menu** table. Provide the **name** item with the **=?** criteria value. Finally, uncheck the **name** item in the **menu** table. Click **OK** to dismiss the **Query Builder**.

g) ***Adding a `Load` event to the `Form`.*** Create the `Load` event handler for the `Form`. Add code to the event handler that opens a connection to the database. Call the `LoadCategory` method four times, each time passing a different category and `ComboBox` as arguments. Close the connection to the database.

h) ***Coding the `LoadCategory` method.*** Create a `LoadCategory method` that takes a `string` representing the `Category` to load and the name of the `ComboBox` to add items to as arguments. Because the `Form`'s `Load` event handler is calling this method before it closes the connection to the database, the connection should still be open. Create a data reader to read all the items from the database for the specified `Category`, using `objSelectNameCommand`. Close the reader before exiting the method, so that a new reader can be created when the method is invoked again.

i) ***Adding a `SelectedIndexChanged` event handler for the `ComboBoxes`.*** Add a `SelectedIndexChanged` event handler for all the `ComboBoxes`. Add code to the event handler that adds the `string` representation of the `SelectedItem` to the `ArrayList`.

j) ***Adding a `Click` event handler for the `btnCalculateBill Button`.*** Add a `Click` event handler for the **Calculate Bill** `Button`. Add code to the event handler to ensure that a table number and waiter name have been entered. If one of these fields is empty, display a `MessageBox` informing the user that both fields must contain information. The event handler should then call the `CalculateSubtotal` method to calculate the subtotal of the bill. Display the subtotal, tax and total of the bill in the appropriate `Labels`.

k) ***Coding the `CalculateSubtotal` method.*** The `CalculateSubtotal` method should open a connection to the database and retrieve the `Price` field for all the menu items in the `m_objBillItems ArrayList` (using `objSelectPriceCommand`). This method should then calculate the total price of all the items in the `ArrayList` and return this value as a `decimal`. Remember to close the connection to the database.

l) ***Running the application.*** Select **Debug > Start** to run your application. Enter the bill information as shown in Fig. 25.39 and click the **Calculate** `Button` to ensure that your application works correctly.

m) ***Closing the application.*** Close your running application by clicking its close box.

n) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 25.12 Solution
// RestaurantBillCalculator.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace RestaurantBillCalculator
{
   /// <summary>
   /// Summary description for FrmRestaurantBillCalculator.
   /// </summary>
   public class FrmRestaurantBillCalculator :
      System.Windows.Forms.Form
   {
      // Label to display title
      private System.Windows.Forms.Label lblFoodHouse;

      // GroupBox containing Labels and TextBoxes to enter
      // table number and waiter name
      private System.Windows.Forms.GroupBox fraWaiterInformation;
      private System.Windows.Forms.Label lblTableNumber;
      private System.Windows.Forms.TextBox txtTableNumber;
      private System.Windows.Forms.Label lblWaiterName;
      private System.Windows.Forms.TextBox txtWaiterName;

      // GroupBox containing Labels and ComboBoxes for
      // choosing menu items
      private System.Windows.Forms.GroupBox fraMenuItems;
      private System.Windows.Forms.Label lblBeverage;
      private System.Windows.Forms.ComboBox cboBeverage;
      private System.Windows.Forms.Label lblAppetizer;
      private System.Windows.Forms.ComboBox cboAppetizer;
      private System.Windows.Forms.Label lblMainCourse;
      private System.Windows.Forms.ComboBox cboMainCourse;
      private System.Windows.Forms.Label lblDessert;
      private System.Windows.Forms.ComboBox cboDessert;

      // Button to calculate the bill
      private System.Windows.Forms.Button btnCalculateBill;

      // Labels and TextBoxes to display the subtotal, tax
      // and total bill
      private System.Windows.Forms.Label lblSubtotal;
      private System.Windows.Forms.Label lblSubtotalResult;
      private System.Windows.Forms.Label lblTax;
      private System.Windows.Forms.Label lblTaxResult;
      private System.Windows.Forms.Label lblTotal;
      private System.Windows.Forms.Label lblTotalResult;

      // OleDbConnection to connect to menu database
      private System.Data.OleDb.OleDbConnection objOleDbConnection;

      // OleDbCommand to retrieve an item's name from the database
      private System.Data.OleDb.OleDbCommand objSelectNameCommand;
```

```
60
61          // OleDbCommand to retrieve an item's price from the database
62          // given its name
63          private System.Data.OleDb.OleDbCommand objSelectPriceCommand;
64
65          /// <summary>
66          /// Required designer variable.
67          /// </summary>
68          private System.ComponentModel.Container components = null;
69
70          // hold all items on running bill
71          private ArrayList m_objBillItems = new ArrayList();
72
73          public FrmRestaurantBillCalculator()
74          {
75             //
76             // Required for Windows Form Designer support
77             //
78             InitializeComponent();
79
80             //
81             // TODO: Add any constructor code after InitializeComponent
82             // call
83             //
84          }
85
86          /// <summary>
87          /// Clean up any resources being used.
88          /// </summary>
89          protected override void Dispose( bool disposing )
90          {
91             if( disposing )
92             {
93                if (components != null)
94                {
95                   components.Dispose();
96                }
97             }
98             base.Dispose( disposing );
99          }
100
101         // Windows Form Designer generated code
102
103         /// <summary>
104         /// The main entry point for the application.
105         /// </summary>
106         [STAThread]
107         static void Main()
108         {
109            Application.Run( new FrmRestaurantBillCalculator() );
110         }
111
112         // handles Load Form event
113         private void FrmRestaurantBillCalculator_Load(
114            object sender, System.EventArgs e )
115         {
116            // open connection to the database
117            objOleDbConnection.Open();
118
119            // load all ComboBoxes with appropriate items
```

```
120          LoadCategory( "Beverage", cboBeverage );
121          LoadCategory( "Appetizer", cboAppetizer );
122          LoadCategory( "Main Course", cboMainCourse );
123          LoadCategory( "Dessert", cboDessert );
124
125          // close connection to the database
126          objOleDbConnection.Close();
127
128       } // end method FrmRestaurantBillCalculator_Load
129
130       // loads the specified category of menu items in
131       // their corresponding ComboBox
132       private void LoadCategory(
133          string strCategory, ComboBox cboCategory )
134       {
135          // specify category parameter
136          objSelectNameCommand.Parameters[ "category" ].Value =
137             strCategory;
138
139          // declare a reader for the database
140          OleDbDataReader objMenuReader;
141
142          objMenuReader =
143             objSelectNameCommand.ExecuteReader();
144
145          while ( objMenuReader.Read() )
146          {
147             // retrieve names of items in the specified category
148             // from database, then add to specified ComboBox
149             cboCategory.Items.Add( objMenuReader[ "Name" ] );
150          }
151
152          objMenuReader.Close(); // close the reader
153
154       } // end method LoadCategory
155
156       // handles SelectedIndexChanged event for cboBeverage ComboBox
157       private void cboBeverage_SelectedIndexChanged(
158          object sender, System.EventArgs e )
159       {
160          // add selected Beverage to m_objBillItems ArrayList
161          m_objBillItems.Add( cboBeverage.SelectedItem );
162
163       } // end method cboBeverage_SelectedIndexChanged
164
165       // handles SelectedIndexChanged event for cboAppetizer ComboBox
166       private void cboAppetizer_SelectedIndexChanged(
167          object sender, System.EventArgs e )
168       {
169          // add selected Appetizer to m_objBillItems ArrayList
170          m_objBillItems.Add( cboAppetizer.SelectedItem );
171
172       } // end method cboAppetizer_SelectedIndexChanged
173
174       // handles SelectedIndexChanged event for cboMainCourse
175       // ComboBox
176       private void cboMainCourse_SelectedIndexChanged(
177          object sender, System.EventArgs e )
178       {
179          // add selected MainCourse to m_objBillItems ArrayList
```

```
180         m_objBillItems.Add( cboMainCourse.SelectedItem );
181
182      } // end method cboMainCourse_SelectedIndexChanged
183
184      // handles SelectedIndexChanged event for cboDessert ComboBox
185      private void cboDessert_SelectedIndexChanged(
186         object sender, System.EventArgs e )
187      {
188         // add selected Dessert to m_objBillItems ArrayList
189         m_objBillItems.Add( cboDessert.SelectedItem );
190
191      } // end method cboDessert_SelectedIndexChanged
192
193      // handles click event for btnCalculateBill Button
194      private void btnCalculateBill_Click(
195         object sender, System.EventArgs e )
196      {
197         // must enter waiter name and table number
198         if ( txtWaiterName.Text == "" ||
199            txtTableNumber.Text == "" )
200         {
201            MessageBox.Show(
202               "Table Number and Waiter Name must be entered",
203               "Empty Field", MessageBoxButtons.OK,
204               MessageBoxIcon.Exclamation );
205         }
206         else // calculate the bill
207         {
208            // calculate the Subtotal
209            decimal decSubtotal = CalculateSubtotal();
210
211            // display the Subtotal in Label
212            lblSubtotalResult.Text =
213               String.Format( "{0:C}", decSubtotal );
214
215            // calculate tax and display in Label
216            decimal decTax = decSubtotal * 0.05M;
217
218            lblTaxResult.Text = String.Format( "{0:C}", decTax );
219
220            // calculate total and display in Label
221            lblTotalResult.Text =
222               String.Format( "{0:C}", ( decSubtotal + decTax ) );
223         }
224
225      } // end method btnCalculateBill_Click
226
227      // calculate the subtotal of the bill
228      private decimal CalculateSubtotal()
229      {
230         // open connection to the database
231         objOleDbConnection.Open();
232
233         // declare a reader for the database
234         OleDbDataReader objMenuReader;
235
236         int intCounter;
237         decimal decSubtotal = 0;
238
239         for ( intCounter = 0; intCounter < m_objBillItems.Count;
```

```
            intCounter++ )
         {
            // specify name of item to retrieve
            objSelectPriceCommand.Parameters[ "name" ].Value =
               Convert.ToString( m_objBillItems[ intCounter ] );

            // initialize objMenuReader
            objMenuReader =
               objSelectPriceCommand.ExecuteReader();
            objMenuReader.Read(); // read from the database

            // retrieve price of items with specified name
            // and add price to dblSubtotal
            decSubtotal +=
               Convert.ToDecimal( objMenuReader[ "Price" ] );

            objMenuReader.Close(); // close the reader
         }

         // close connection to the database
         objOleDbConnection.Close();

         return decSubtotal;

      } // end method CalculateSubtotal

   } // end class FrmRestaurantBillCalculator
}
```

25.13 ***(Airline Reservation Application)*** An airline company wants you to develop an application that displays flight information stored in their database (Fig. 25.40). The database contains two tables—one containing information about the flights and the other containing passenger information. The user should be able to choose a flight number from a `ComboBox`. When the **View Flight Information** `Button` is clicked, the application should display the date of the flight, the flight's departure and arrival cities and the names of the passengers scheduled to take the flight.

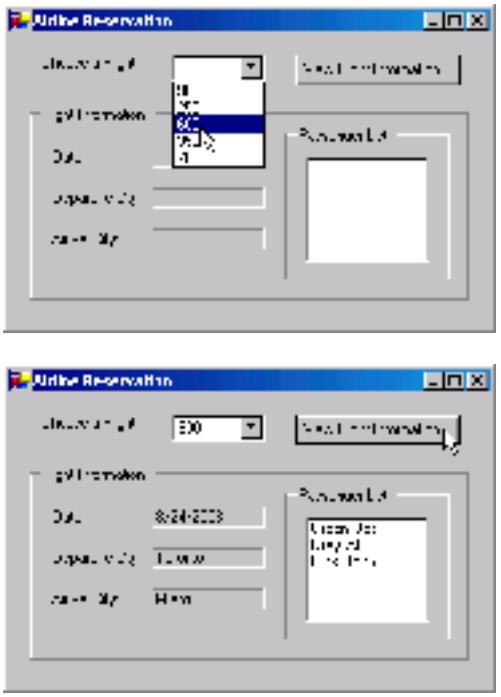


Figure 25.40 Airline Reservation application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial25\Exercises\AirlineReservation` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AirlineReservation.sln` in the `AirlineReservation` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `reservations.mdb` database from `C:\Examples\Tutorial25\Exercises\Databases` to your `C:\SimplyCSP\AirlineReservation` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `reservations.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the Form.*** Add three command objects to the `Form`, and set all their `Connection` properties to the database connection object. Name the command objects `objSelectFlightNumberCommand` (used to retrieve flight numbers), `objSelectFlightInformationCommand` (used to retrieve information about a flight based on the flight's number) and `objSelectPassengerInformationCommand` (used to retrieve information about a flight's passengers based on the flight's number). Rearrange and comment the control declarations appropriately.

f) ***Setting the command objects' `CommandText` properties.*** Select the `objSelectFlightNumberCommand` object, and open the **Query Builder**. Select **FlightNumber** from the **flights** table and click **OK**. Select the `objSelectFlightInformationCommand`, and open the **Query Builder**. This time, select the **Date**, **DepartureCity** and **ArrivalCity** items from the **flights** table. This action causes all items from the table to be returned. Then, select the **FlightNumber** item, and provide it with the **=?** criteria value. Finally, uncheck the **FlightNumber** item from the **flights** table. Click **OK** to dismiss the **Query Builder**. Select the `objSelectPassengerInformationCommand`, and open the **Query Builder**. Select the **LastName** and **FirstName** items from the **reservations** table. Then, select the **FlightNumber** item, and provide it with the **=?** criteria value. Finally, uncheck the **FlightNumber** item from the **reservations** table. Click **OK** to dismiss the **Query Builder**.

g) ***Adding a `Load` event to the `Form`.*** Create a `Load` event handler for the `Form` that opens a connection to the database. Retrieve all the `FlightNumbers` from the `Flights` table in the `reservations.mdb` database (using `objSelectFlightNumberCommand`), then add those `FlightNumbers` to the `ComboBox`.

h) ***Adding a `Click` event handler for the `btnViewFlightInformation Button`.*** Add a `Click` event handler for the **View Flight Information** `Button`. Add code to the event handler to pass the `SelectedItem` to the `DisplayFlightInformation` method.

i) ***Defining the `DisplayFlightInformation` method.*** The `DisplayFlightInformation` method should take as an argument a `string` representing the flight number chosen. You will need to define two readers in this method to read from the two tables in the database. Once you open the connection to the database, create a reader that reads the specified flight information from the `flights` table (using `objSelectFlightInformationCommand`). Display the flight information in the correct `Label`. Close this reader, and create a second reader that reads passenger information from the `reservations` table (using `objPassengerInformationCommand`). Retrieve from the table all the passengers scheduled to take the specified flight. Clear any old items from the `ListBox`, and display passengers' names in the `ListBox`.

j) ***Running the application.*** Select **Debug > Start** to run your application. Select various flight numbers from the `ComboBox`, and click the **View Flight Information** `Button` to display a flight's information.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 25.13 Solution
// AirlineReservation.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace AirlineReservation
{
   /// <summary>
   /// Summary description for FrmAirlineReservation.
   /// </summary>
   public class FrmAirlineReservation : System.Windows.Forms.Form
   {
      // Label and ComboBox to choose a flight
      private System.Windows.Forms.Label lblChooseFlight;
      private System.Windows.Forms.ComboBox cboChooseAFlight;

      // Button to view flight information
      private System.Windows.Forms.Button btnViewFlightInformation;

      // GroupBox containing flight information
      private System.Windows.Forms.GroupBox fraFlightInformation;

      // Labels to display date of flight, departure city, and
      // arrival city
      private System.Windows.Forms.Label lblDate;
      private System.Windows.Forms.Label lblDateOutput;
      private System.Windows.Forms.Label lblDeparture;
      private System.Windows.Forms.Label lblDepartureOutput;
      private System.Windows.Forms.Label lblArrival;
      private System.Windows.Forms.Label lblArrivalOutput;

      // GroupBox containing ListBox to display passenger list
      private System.Windows.Forms.GroupBox fraPassengerList;
      private System.Windows.Forms.ListBox lstDisplay;
```

```
41
42          // OleDbConnection to connect to reservations database
43          private System.Data.OleDb.OleDbConnection objOleDbConnection;
44
45          // OleDbCommand to retrieve flight number from database
46          private System.Data.OleDb.OleDbCommand
47             objSelectFlightNumberCommand;
48
49          // OleDbCommand to retrieve flight information from database
50          // given flight number
51          private System.Data.OleDb.OleDbCommand
52             objSelectFlightInformationCommand;
53
54          // OleDbCommand to retrieve passenger information from database
55          // given flight number
56          private System.Data.OleDb.OleDbCommand
57             objSelectPassengerInformationCommand;
58
59          /// <summary>
60          /// Required designer variable.
61          /// </summary>
62          private System.ComponentModel.Container components = null;
63
64          public FrmAirlineReservation()
65          {
66             //
67             // Required for Windows Form Designer support
68             //
69             InitializeComponent();
70
71             //
72             // TODO: Add any constructor code after InitializeComponent
73             // call
74             //
75          }
76
77          /// <summary>
78          /// Clean up any resources being used.
79          /// </summary>
80          protected override void Dispose( bool disposing )
81          {
82             if( disposing )
83             {
84                if (components != null)
85                {
86                   components.Dispose();
87                }
88             }
89             base.Dispose( disposing );
90          }
91
92          // Windows Form Designer generated code
93
94          /// <summary>
95          /// The main entry point for the application.
96          /// </summary>
97          [STAThread]
98          static void Main()
99          {
100            Application.Run( new FrmAirlineReservation() );
```

```
101       }
102
103       // handles Form Load event
104       private void FrmAirlineReservation_Load(
105          object sender, System.EventArgs e )
106       {
107          objOleDbConnection.Open(); // open database connection
108
109          // create reader to read information from database
110          OleDbDataReader objReader;
111
112          objReader =
113             objSelectFlightNumberCommand.ExecuteReader();
114
115          while ( objReader.Read() )
116          {
117             // retrieve flight number from database
118             // and add it to cboChooseAFlight
119             cboChooseAFlight.Items.Add(
120                Convert.ToString( objReader[ "FlightNumber" ] ) );
121          }
122
123          objOleDbConnection.Close(); // close database connection
124
125       } // end method FrmAirlineReservation_Load
126
127       // handles click event for btnViewFlightInformation Button
128       private void btnViewFlightInformation_Click(
129          object sender, System.EventArgs e )
130       {
131          // retrieve selected index
132          string strFlightNumber =
133             Convert.ToString( cboChooseAFlight.SelectedItem );
134
135          // display new flight information
136          DisplayFlightInformation( strFlightNumber );
137
138       } // end method btnViewFlightInformation_Click
139
140       // display flight information in GroupBox's Labels
141       private void DisplayFlightInformation( string strFlightNumber )
142       {
143          objOleDbConnection.Open(); // open database connection
144
145          objSelectFlightInformationCommand.Parameters[
146             "flightNumber" ].Value = strFlightNumber;
147
148          // create data reader to read information from database
149          OleDbDataReader objFlightReader;
150
151          objFlightReader =
152             objSelectFlightInformationCommand.ExecuteReader();
153
154          while ( objFlightReader.Read() )
155          {
156             // retrieve date, departure city, arrival city
157             lblDateOutput.Text =
158                Convert.ToString( objFlightReader[ "Date" ] );
159             lblDepartureOutput.Text =
160                Convert.ToString(
```

```
            objFlightReader[ "DepartureCity" ] );
         lblArrivalOutput.Text =
            Convert.ToString( objFlightReader[ "ArrivalCity" ] );
      }

      objFlightReader.Close(); // close the data reader

      // specify flight number to retrieve passenger information
      objSelectPassengerInformationCommand.Parameters[
         "FlightNumber" ].Value = strFlightNumber;

      // create data reader to read information from database
      OleDbDataReader objPassengerReader;

      objPassengerReader =
         objSelectPassengerInformationCommand.ExecuteReader();

      string strFirstName;
      string strLastName;

      lstDisplay.Items.Clear(); // clear previous passenger list

      while ( objPassengerReader.Read() )
      {
         strLastName =
            Convert.ToString( objPassengerReader[ "LastName" ] );
         strFirstName =
            Convert.ToString( objPassengerReader[ "FirstName" ] );

         lstDisplay.Items.Add(
            strLastName + ", " + strFirstName );
      }

      objPassengerReader.Close(); // close the data reader

      objOleDbConnection.Close(); // close database connection

   } // end method DisplayFlightInformatio

} // end class FrmAirlineReservation
}
```

What does this code do?

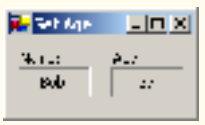
25.14 What does the following code do?

```
int m_intAge;
objSelectAgeData.Parameters[ "Name" ].Value = "Bob";

objOleDbConnection.Open();

OleDbDataReader objReader =
   objSelectAgeData.ExecuteReader();

objReader.Read();

m_intAge = Convert.ToInt32( objReader[ "Age" ] );

objReader.Close();
objOleDbConnection.Close();
```

Answer: This code segment connects to a database and retrieves the `Age` data for the person with the name `Bob`. This data is then stored in `m_intAge`. The code then closes both the data reader and the connection to the database. For this example, a sample database named `Employees.mdb` has been provided in the `bin\debug` directory. Follow the instructions previously presented in this tutorial to establish a connection to the database. The complete code reads:

```
// Exercise 25.14 Solution
// GetAge.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace GetAge
{
   /// <summary>
   /// Summary description for FrmGetAge.
   /// </summary>
   public class FrmGetAge : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblAgeOut;
      private System.Windows.Forms.Label lblAge;
      private System.Windows.Forms.Label lblNameOut;
      private System.Windows.Forms.Label lblName;
      private System.Data.OleDb.OleDbConnection objOleDbConnection;
      private System.Data.OleDb.OleDbCommand objSelectAgeData;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmGetAge()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
```

```
55             base.Dispose( disposing );
56          }
57
58          // Windows Form Designer generated code
59
60          /// <summary>
61          /// The main entry point for the application.
62          /// </summary>
63          [STAThread]
64          static void Main()
65          {
66             Application.Run(new FrmGetAge());
67          }
68
69          // retrieves and displays the age of Bob from "Employees.mdb"
70          // in the "bin\debug" directory
71          private void FrmGetAge_Load(
72             object sender, System.EventArgs e )
73          {
74             int m_intAge;
75             objSelectAgeData.Parameters[ "Name" ].Value = "Bob";
76
77             objOleDbConnection.Open();
78
79             OleDbDataReader objReader =
80                objSelectAgeData.ExecuteReader();
81
82             objReader.Read();
83
84             m_intAge = Convert.ToInt32( objReader[ "Age" ] );
85
86             objReader.Close();
87             objOleDbConnection.Close();
88
89             lblAgeOut.Text = Convert.ToString( m_intAge );
90
91          } // end method FrmGetAge_Load
92
93       } // end class FrmGetAge
94    }
```



What's wrong with this code? ▶

25.15 Find the error(s) in the following code. This method should modify the `Age` field of `strUserName`.

```
1  objUpdateAge[ "Age" ].Value = intAge;
2
3  objUpdateAge.Parameters[ "Original_NAME" ].Value =
4     strUserName;
5
6  objUpdateAge.ExecuteNonQuery();
7
```

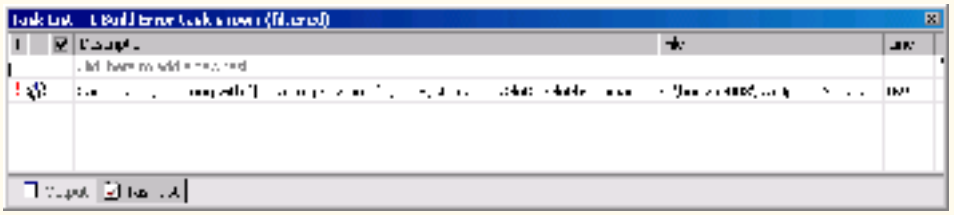
```
8  objOleDbConnection.Close();
9  objUpdateAge[ "Age" ].Value = intAge;
```

Answer: The UPDATE statement on line 1 is malformed. Also, no connection is made to the database before the UPDATE is attempted. For this example, a sample database named Sample.mdb has been provided in the bin\debug directory. Follow the instructions previously presented in this tutorial to establish a connection to the database. The complete incorrect code reads:

```
1  // Exercise 25.15 Solution
2  // UpdateAge.cs (Incorrect)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.Data.OleDb;
11
12 namespace UpdateAge
13 {
14    /// <summary>
15    /// Summary description for FrmUpdateAge.
16    /// </summary>
17    public class FrmUpdateAge : System.Windows.Forms.Form
18    {
19       private System.Windows.Forms.Label lblAgeOut;
20       private System.Windows.Forms.Label lblNameOut;
21       private System.Windows.Forms.Label lblName;
22       private System.Windows.Forms.Label Label1;
23       private System.Data.OleDb.OleDbConnection objOleDbConnection;
24       private System.Data.OleDb.OleDbCommand objUpdateAge;
25       private System.Data.OleDb.OleDbCommand objSelectAge;
26       /// <summary>
27       /// Required designer variable.
28       /// </summary>
29       private System.ComponentModel.Container components = null;
30
31       public FrmUpdateAge()
32       {
33          //
34          // Required for Windows Form Designer support
35          //
36          InitializeComponent();
37
38          //
39          // TODO: Add any constructor code after InitializeComponent
40          // call
41          //
42       }
43
44       /// <summary>
45       /// Clean up any resources being used.
46       /// </summary>
47       protected override void Dispose( bool disposing )
48       {
49          if( disposing )
50          {
51             if (components != null)
```

```
52             {
53                components.Dispose();
54             }
55          }
56          base.Dispose( disposing );
57       }
58
59       // Windows Form Designer generated code
60
61       /// <summary>
62       /// The main entry point for the application.
63       /// </summary>
64       [STAThread]
65       static void Main()
66       {
67          Application.Run( new FrmUpdateAge() );
68       }
69
70       // updates an age in the database "sample.mdb" located in
71       // the "bin\debug" directory
72       private void FrmUpdateAge_Load(
73          object sender, System.EventArgs e )
74       {
75          int intAge = 27;
76          string strUserName = "Bill";
77
78          objUpdateAge[ "Age" ].Value = intAge;
79
80          objUpdateAge.Parameters[ "Original_NAME" ].Value =
81             strUserName;
82
83          objUpdateAge.ExecuteNonQuery();
84
85          objOleDbConnection.Close();
86
87          // display the updated data
88          objOleDbConnection.Open();
89
90          OleDbDataReader objReader =
91             objSelectAge.ExecuteReader();
92
93          objReader.Read();
94
95          lblAgeOut.Text = Convert.ToString( objReader[ "Age" ] );
96
97          objReader.Close();
98          objOleDbConnection.Close();
99
100      } // end method FrmUpdateAge_Load
101
102   } // end class FrmUpdateAge
103 }
```

This statement should use the same syntax as lines 73–74. Also, no connection is ever established.



Answer: The complete corrected code should read:

```
1  // Exercise 25.15 Solution
2  // UpdateAge.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.Data.OleDb;
11
12 namespace UpdateAge
13 {
14    /// <summary>
15    /// Summary description for FrmUpdateAge.
16    /// </summary>
17    public class FrmUpdateAge : System.Windows.Forms.Form
18    {
19       private System.Windows.Forms.Label lblAgeOut;
20       private System.Windows.Forms.Label lblNameOut;
21       private System.Windows.Forms.Label lblName;
22       private System.Windows.Forms.Label Label1;
23       private System.Data.OleDb.OleDbConnection objOleDbConnection;
24       private System.Data.OleDb.OleDbCommand objUpdateAge;
25       private System.Data.OleDb.OleDbCommand objSelectAge;
26       /// <summary>
27       /// Required designer variable.
28       /// </summary>
29       private System.ComponentModel.Container components = null;
30
31       public FrmUpdateAge()
32       {
33          //
34          // Required for Windows Form Designer support
35          //
36          InitializeComponent();
37
38          //
39          // TODO: Add any constructor code after InitializeComponent
40          // call
41          //
42       }
43
44       /// <summary>
45       /// Clean up any resources being used.
46       /// </summary>
```

```
47          protected override void Dispose( bool disposing )
48          {
49             if( disposing )
50             {
51                if (components != null)
52                {
53                   components.Dispose();
54                }
55             }
56             base.Dispose( disposing );
57          }
58
59          // Windows Form Designer generated code
60
61          /// <summary>
62          /// The main entry point for the application.
63          /// </summary>
64          [STAThread]
65          static void Main()
66          {
67             Application.Run( new FrmUpdateAge() );
68          }
69
70          // updates an age in the database "sample.mdb" located in
71          // the "bin\debug" directory
72          private void FrmUpdateAge_Load(
73             object sender, System.EventArgs e )
74          {
75             int intAge = 27;
76             string strUserName = "Bill";
77
78             objUpdateAge.Parameters[ "Age" ].Value = intAge;
79
80             objUpdateAge.Parameters[ "Original_NAME" ].Value =
81                strUserName;
82
83             objOleDbConnection.Open();
84
85             objUpdateAge.ExecuteNonQuery();
86
87             objOleDbConnection.Close();
88
89             // display the updated data
90             objOleDbConnection.Open();
91
92             OleDbDataReader objReader =
93                objSelectAge.ExecuteReader();
94
95             objReader.Read();
96
97             lblAgeOut.Text =
98                Convert.ToString( objReader[ "Age" ] );
99
100            objReader.Close();
101            objOleDbConnection.Close();
102
103         } // end method FrmUpdateAge_Load
104
105      } // end class FrmUpdateAge
106   }
```

Programming Challenge

25.16 ***(Enhanced Restaurant Bill Calculator Application)*** Modify the application you developed in Exercise 25.12 to keep track of multiple table bills at the same time. The user should be able to calculate a bill for a table and save that table's subtotal and waiter's name. The user should also be able to retrieve that information at a later time. [*Hint*: This database contains two tables—one for the menu items, as before, and another for all the tables in the restaurant.] Sample outputs are shown in Fig. 25.41.

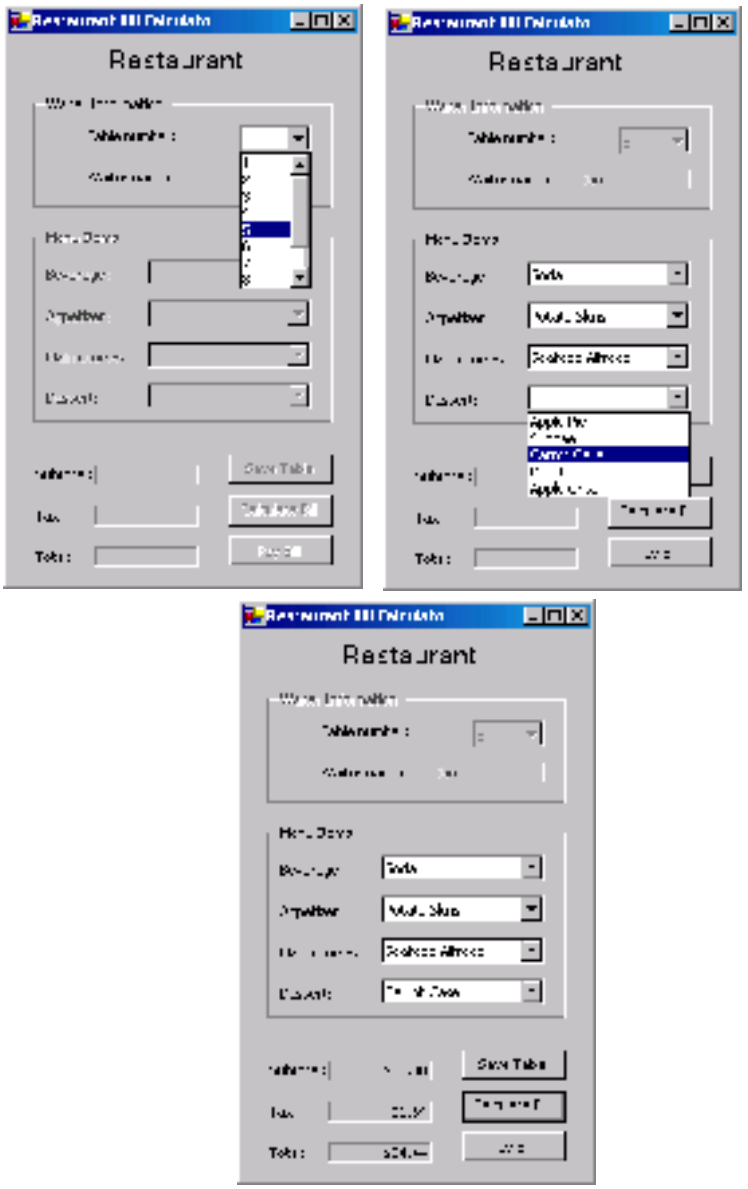


Figure 25.41 Enhanced **Restaurant Bill Calculator** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial25\Exercises\RestaurantBillCalculatorEnhanced` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `RestaurantBillCalculator.sln` in the `RestaurantBillCalculatorEnhanced` directory to open the application.

c) ***Copying the database to your working directory.*** Copy the `menu2.mdb` database from `C:\Examples\Tutorial25\Exercises\Databases` to your `C:\SimplyCSP\RestaurantBillCalculatorEnhanced` directory.

d) ***Adding a data connection to the Server Explorer.*** Click the Connect to Database icon in the **Server Explorer**, and add a data connection to the `menu2.mdb` database. Add an `OleDbConnection` object to the `Form`.

e) ***Adding command objects to the `Form`.*** Add five command objects to the `Form`, and set their `Connection` properties to the database connection object. Name the command objects `objSelectNameCommand`, `objSelectPriceCommand`, `objSelectTableNumberCommand`, `objSelectTableInfoCommand` and `objUpdateSubtotalCommand`. Rearrange and comment the control declarations appropriately.

f) ***Setting the command objects' `CommandText` properties.*** Set the `CommandText` properties of `objSelectNameCommand` and `objSelectPriceCommand` as you did in Exercise 25.12. Set `objSelectTableNumberCommand` to retrieve table numbers from the **tables** table. Set `objSelectTableInfoCommand` to retrieve the name of the waiter and the subtotal of a table, based on that table's number. Set `objUpdateSubtotalCommand` to modify the subtotal for a table, also based on that table's number. [*Note*: For the last command object, you will need to change the type of query in the **Query Builder**, as you did earlier in this tutorial.]

g) ***Copying your existing code.*** Copy the code for the application you created in Exercise 25.12 into the template application for this exercise. Place this code before the `ResetForm` method. Disregard any syntax errors that may appear in the **Task List** at this point.

h) ***Adding an instance variable.*** Add an instance variable (after the declaration of `m_objBillItems`) called `m_decSubtotal` that will hold the subtotal for each table when it is loaded in the application.

i) ***Modifying the `btnCalculateBill_Click` and `CalculateSubtotal` methods.*** Remove the portion of the `btnCalculateBill_Click` event handler that checked for a table number and waiter name—this information will be displayed shortly. Modify the `CalculateSubtotal` method to update the table's subtotal based on the table's previous subtotal and any new items selected.

j) ***Creating a method.*** Create the `LoadTables` method that reads the table numbers from the database and adds them to the **Table number**: `ComboBox`. This method should be called in `FrmRestaurantBillCalculator_Load` directly after the connection to the database is opened.

k) ***Adding an event handler.*** Add an event handler for the **Table number**: `ComboBox`. When a table is selected from the `ComboBox`, that table's data should be loaded from the database.

l) ***Creating an event handler for the Save Table `Button`.*** Create an event handler for the **Save Table** `Button`. This event handler should calculate the subtotal for the selected table. The event handler should then call the `UpdateTable` method, passing the subtotal and table number as arguments. Finally, call the `ResetForm` method to reset the data displayed in the GUI.

m) ***Creating an event handler for the Pay Bill `Button`.*** Create an event handler for the **Pay Bill** `Button`. This event handler should retrieve the current table number and call the `UpdateTable` method, passing a subtotal of `0` (for new customers) and table number as arguments. Finally, call the `ResetForm` method to reset the data displayed in the GUI.

n) ***Creating method `UpdateTable`.*** Create an `UpdateTable` method that takes the subtotal and table number as arguments. This method should save the table data in the database.

o) ***Running the application.*** Select **Debug > Start** to run your application. Enter bill information for several tables. Ensure that your application works correctly by testing each of the three `Button`s on various tables.

p) ***Closing the application.*** Close your running application by clicking its close box.

q) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 25.16 Solution
// RestaurantBillCalculator.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;

namespace RestaurantBillCalculator
{
   /// <summary>
   /// Summary description for FrmRestaurantBillCalculator.
   /// </summary>
   public class FrmRestaurantBillCalculator :
      System.Windows.Forms.Form
   {
      // Label to display title
      private System.Windows.Forms.Label lblFoodHouse;

      // GroupBox containing Labels and ComboBox to enter
      // table number and waiter name
      private System.Windows.Forms.GroupBox fraWaiterInformation;
      private System.Windows.Forms.Label lblTableNumber;
      private System.Windows.Forms.ComboBox cboTables;
      private System.Windows.Forms.Label lblWaiterName;
      private System.Windows.Forms.Label lblWaiterNameOutput;

      // GroupBox containing Labels and ComboBoxes for
      // choosing menu items
      private System.Windows.Forms.GroupBox fraMenuItems;
      private System.Windows.Forms.Label lblBeverage;
      private System.Windows.Forms.ComboBox cboBeverage;
      private System.Windows.Forms.Label lblAppetizer;
      private System.Windows.Forms.ComboBox cboAppetizer;
      private System.Windows.Forms.Label lblMainCourse;
      private System.Windows.Forms.ComboBox cboMainCourse;
      private System.Windows.Forms.Label lblDessert;
      private System.Windows.Forms.ComboBox cboDessert;

      // Labels and TextBoxes to display the subtotal, tax
      // and total bill
      private System.Windows.Forms.Label lblSubtotal;
      private System.Windows.Forms.Label lblSubtotalResult;
      private System.Windows.Forms.Label lblTax;
      private System.Windows.Forms.Label lblTaxResult;
      private System.Windows.Forms.Label lblTotal;
      private System.Windows.Forms.Label lblTotalResult;

      // Button to save the table information
      private System.Windows.Forms.Button btnSaveTable;
```

```
54
55          // Button to calculate the bill
56          private System.Windows.Forms.Button btnCalculateBill;
57
58          // Button to pay the bill
59          private System.Windows.Forms.Button btnPayBill;
60
61          // OleDbConnection to connect to the menu database
62          private System.Data.OleDb.OleDbConnection objOleDbConnection;
63
64          // OleDbCommand to select a waiter name from the database
65          // given a category
66          private System.Data.OleDb.OleDbCommand objSelectNameCommand;
67
68          // OleDbCommand to select table information from the database
69          // given a table number
70          private System.Data.OleDb.OleDbCommand
71             objSelectTableInfoCommand;
72
73          // OleDbCommand to select a price from the database
74          // given an item name
75          private System.Data.OleDb.OleDbCommand objSelectPriceCommand;
76
77          // OleDbCommand to select a table number from the database
78          private System.Data.OleDb.OleDbCommand
79             objSelectTableNumberCommand;
80
81          // OleDbCommand to update a subtotal in the database
82          // given a table number
83          private System.Data.OleDb.OleDbCommand
84             objUpdateSubtotalCommand;
85
86          /// <summary>
87          /// Required designer variable.
88          /// </summary>
89          private System.ComponentModel.Container components = null;
90
91          // hold all items on running bill
92          private ArrayList m_objBillItems = new ArrayList();
93
94          // current table subtotal
95          private decimal m_decSubtotal = 0;
96
97          public FrmRestaurantBillCalculator()
98          {
99             //
100            // Required for Windows Form Designer support
101            //
102            InitializeComponent();
103
104            //
105            // TODO: Add any constructor code after InitializeComponent
106            // call
107            //
108         }
109
110         /// <summary>
111         /// Clean up any resources being used.
112         /// </summary>
113         protected override void Dispose( bool disposing )
```

```
114         {
115            if( disposing )
116            {
117               if (components != null)
118               {
119                  components.Dispose();
120               }
121            }
122            base.Dispose( disposing );
123         }
124
125         // Windows Form Designer generated code
126
127         /// <summary>
128         /// The main entry point for the application.
129         /// </summary>
130         [STAThread]
131         static void Main()
132         {
133            Application.Run( new FrmRestaurantBillCalculator() );
134         }
135
136         // handles Form Load event
137         private void FrmRestaurantBillCalculator_Load(
138            object sender, System.EventArgs e )
139         {
140            // open connection to the database
141            objOleDbConnection.Open();
142
143            // load cboTables ComboBox with table numbers
144            LoadTables();
145
146            // load all ComboBoxes with appropriate items
147            LoadCategory( "Beverage", cboBeverage );
148            LoadCategory( "Appetizer", cboAppetizer );
149            LoadCategory( "Main Course", cboMainCourse );
150            LoadCategory( "Dessert", cboDessert );
151
152            // close connection to the database
153            objOleDbConnection.Close();
154
155         } // end method FrmRestaurantBillCalculator_Load
156
157         // loads the specified category of menu items in
158         // their corresponding ComboBox
159         private void LoadCategory(
160            string strCategory, ComboBox cboCategory )
161         {
162            // specify category parameter
163            objSelectNameCommand.Parameters[ "category" ].Value =
164               strCategory;
165
166            // declare a reader for the database
167            OleDbDataReader objMenuReader;
168
169            objMenuReader =
170               objSelectNameCommand.ExecuteReader();
171
172            while ( objMenuReader.Read() )
173            {
```

```
174            // retrieve names of items in the specified category
175            // from database, then add to specified ComboBox
176            cboCategory.Items.Add( objMenuReader[ "Name" ] );
177         }
178
179         objMenuReader.Close(); // close the reader
180
181      } // end method LoadCategory
182
183      // handles SelectedIndexChanged event for cboBeverage ComboBox
184      private void cboBeverage_SelectedIndexChanged(
185         object sender, System.EventArgs e )
186      {
187         // add selected Beverage to m_objBillItems ArrayList
188         m_objBillItems.Add( cboBeverage.SelectedItem );
189
190      } // end method cboBeverage_SelectedIndexChanged
191
192      // handles SelectedIndexChanged event for cboAppetizer ComboBox
193      private void cboAppetizer_SelectedIndexChanged(
194         object sender, System.EventArgs e )
195      {
196         // add selected Appetizer to m_objBillItems ArrayList
197         m_objBillItems.Add( cboAppetizer.SelectedItem );
198
199      } // end method cboAppetizer_SelectedIndexChanged
200
201      // handles SelectedIndexChanged event for cboMainCourse
202      // ComboBox
203      private void cboMainCourse_SelectedIndexChanged(
204         object sender, System.EventArgs e )
205      {
206         // add selected MainCourse to m_objBillItems ArrayList
207         m_objBillItems.Add( cboMainCourse.SelectedItem );
208
209      } // end method cboMainCourse_SelectedIndexChanged
210
211      // handles SelectedIndexChanged event for cboDessert ComboBox
212      private void cboDessert_SelectedIndexChanged(
213         object sender, System.EventArgs e )
214      {
215         // add selected Dessert to m_objBillItems ArrayList
216         m_objBillItems.Add( cboDessert.SelectedItem );
217
218      } // end method cboDessert_SelectedIndexChanged
219
220      // handles click event for btnCalculateBill Button
221      private void btnCalculateBill_Click(
222         object sender, System.EventArgs e )
223      {
224         // calculate the Subtotal
225         decimal decSubtotal = CalculateSubtotal();
226
227         // display the Subtotal in Label
228         lblSubtotalResult.Text =
229            String.Format( "{0:C}", decSubtotal );
230
231         // calculate tax and display in Label
232         decimal decTax = decSubtotal * 0.05M;
233
```

```
234          lblTaxResult.Text = String.Format( "{0:C}", decTax );
235
236          // calculate total and display in Label
237          lblTotalResult.Text =
238             String.Format( "{0:C}", ( decSubtotal + decTax ) );
239
240       } // end method btnCalculateBill_Click
241
242       // calculate the subtotal of the bill
243       private decimal CalculateSubtotal()
244       {
245          // open connection to the database
246          objOleDbConnection.Open();
247
248          // declare a reader for the database
249          OleDbDataReader objMenuReader;
250
251          int intCounter;
252          decimal decSubtotal = m_decSubtotal;
253
254          for ( intCounter = 0; intCounter < m_objBillItems.Count;
255             intCounter++ )
256          {
257             // specify name of item to retrieve
258             objSelectPriceCommand.Parameters[ "name" ].Value =
259                Convert.ToString( m_objBillItems[ intCounter ] );
260
261             // initialize objMenuReader
262             objMenuReader =
263                objSelectPriceCommand.ExecuteReader();
264             objMenuReader.Read(); // read from the database
265
266             // retrieve price of items with specified name
267             // and add price to dblSubtotal
268             decSubtotal +=
269                Convert.ToDecimal( objMenuReader[ "Price" ] );
270
271             objMenuReader.Close(); // close the reader
272          }
273
274          // close connection to the database
275          objOleDbConnection.Close();
276
277          return decSubtotal;
278
279       } // end method CalculateSubtotal
280
281       // load tables from database
282       private void LoadTables()
283       {
284          // declare a reader for the database
285          OleDbDataReader objTableReader;
286
287          objTableReader =
288             objSelectTableNumberCommand.ExecuteReader();
289
290          while ( objTableReader.Read() )
291          {
292             // retrieve names of items in the specified category
293             // from database, then add to specified ComboBox
```

```
294            cboTables.Items.Add( objTableReader[ "TableNumber" ] );
295         }
296
297         objTableReader.Close();
298
299      } // end method LoadTables
300
301      // handles SelectedIndexChanged event for cboTables ComboBox
302      private void cboTables_SelectedIndexChanged(
303         object sender, System.EventArgs e )
304      {
305         // enable all Menu Items GroupBox
306         fraMenuItems.Enabled = true;
307
308         // enable all Buttons
309         btnSaveTable.Enabled = true;
310         btnCalculateBill.Enabled = true;
311         btnPayBill.Enabled = true;
312
313         // clear Tax and Total output Labels
314         lblTaxResult.Text = "";
315         lblTotalResult.Text = "";
316
317         // open connection to the database
318         objOleDbConnection.Open();
319
320         // SELECT statement used to retrieve item names
321         objSelectTableInfoCommand.Parameters[ "tableNumber" ].Value
322            = Convert.ToInt32( cboTables.SelectedItem );
323
324         // declare a reader for the database
325         OleDbDataReader objTableReader;
326
327         objTableReader =
328            objSelectTableInfoCommand.ExecuteReader();
329
330         while ( objTableReader.Read() )
331         {
332            // retrieve waiter name and subtotal from database
333            lblWaiterNameOutput.Text =
334               Convert.ToString( objTableReader[ "WaiterName" ] );
335            m_decSubtotal =
336               Convert.ToDecimal( objTableReader[ "Subtotal" ] );
337
338            lblSubtotalResult.Text =
339               String.Format( "{0:C}", m_decSubtotal );
340         }
341
342         objTableReader.Close(); // close the reader
343
344         // close connection to the database
345         objOleDbConnection.Close();
346
347         // disable Waiter Information GroupBox
348         fraWaiterInformation.Enabled = false;
349
350      } // end method cboTables_SelectedIndexChanged
351
352      private void btnSaveTable_Click(
353         object sender, System.EventArgs e )
```

```
354         {
355            decimal decSubtotal = CalculateSubtotal();
356            int intTableNumber =
357               Convert.ToInt32( cboTables.SelectedItem );
358
359            // update table
360            UpdateTable( decSubtotal, intTableNumber );
361
362            ResetForm(); // reset all controls
363
364         } // end method btnSaveTable_Click
365
366         private void btnPayBill_Click(
367            object sender, System.EventArgs e )
368         {
369            int intTableNumber =
370               Convert.ToInt32( cboTables.SelectedItem );
371
372            UpdateTable( 0, intTableNumber ); // update table
373            ResetForm(); // reset all controls
374
375         } // end method btnPayBill_Click
376
377         // update table information in database
378         private void UpdateTable(
379            decimal decSubtotal, int intTableNumber )
380         {
381            objOleDbConnection.Open(); // open database connection
382
383            // specify parameters for UPDATE statement
384            objUpdateSubtotalCommand.Parameters[
385               "subtotal" ].Value = decSubtotal;
386            objUpdateSubtotalCommand.Parameters[
387               "Original_TableNumber" ].Value =
388               Convert.ToString( intTableNumber );
389
390            // execute update statement
391            objUpdateSubtotalCommand.ExecuteNonQuery();
392
393            objOleDbConnection.Close(); // close database connection
394
395         } // end method  UpdateTable
396
397         // reset all controls on the Form
398         private void ResetForm()
399         {
400            // clear all Labels
401            lblSubtotalResult.ResetText();
402            lblTaxResult.ResetText();
403            lblTotalResult.ResetText();
404            lblWaiterNameOutput.ResetText();
405
406            // set Selected index to -1 so
407            // no item is selected in ComboBoxes
408            cboBeverage.SelectedIndex = -1;
409            cboAppetizer.SelectedIndex = -1;
410            cboMainCourse.SelectedIndex = -1;
411            cboDessert.SelectedIndex = -1;
412            cboTables.SelectedIndex = -1;
413
```

```
414          // disable Menu Items GroupBox
415          fraMenuItems.Enabled = false;
416
417          // enable Waiter Information GroupBox
418          fraWaiterInformation.Enabled = true;
419
420          // disable all Buttons
421          btnSaveTable.Enabled = false;
422          btnCalculateBill.Enabled = false;
423          btnPayBill.Enabled = false;
424
425          // clear subtotal
426          m_decSubtotal = 0;
427
428          // reinitialize objBillItems ArrayList
429          m_objBillItems = new ArrayList();
430
431       } // end method ResetForm
432
433    } // end class FrmRestaurantBillCalculator
434 }
```

TUTORIAL 26

Check Writer Application

Introducing Graphics and Printing

Solutions

Instructor's Manual Exercise Solutions Tutorial 26

MULTIPLE-CHOICE QUESTIONS

26.1 The RGB value (0, 0, 255) represents _____.

a) `Color.Red` b) `Color.Green`
c) `Color.Blue` d) `Color.Yellow`

26.2 The _____ property of the `PrintPreviewDialog` object makes text appear smoother.

a) `AntiAlias` b) `UseAntiAlias`
c) `Alias` d) `UseAlias`

26.3 Use a _____ object to allow users to preview a document before it is printed.

a) `PrintPreviewDialog` b) `PrintDocument`
c) `Print` d) `PrintPreviewControl`

26.4 The _____ event occurs when the data required to print the current page is needed.

a) `OnPaint` b) `Print`
c) `Document` d) `PrintPage`

26.5 To display the preview dialog of the _____ object, call the `ShowDialog` method.

a) `PrintPreviewDialog` b) `PrintDocument`
c) `PrintDialog` d) Both a and b.

26.6 Set the _____ property to `false` to indicate that there are no more pages to print.

a) `Document` b) `HasMorePages`
c) `TerminatePrint` d) Both a and b.

26.7 The `Print` method of class `PrintDocument` sends a _____ object to the printer for printing.

a) `Graphics` b) `PrintDocument`
c) `PrintPreviewDialog` d) `Brush`

26.8 The _____ keyword references the current object.

a) `me` b) `class`
c) `this` d) `property`

26.9 Opacity is the _____ value of a color.

a) red b) transparency
c) dithering d) blue

26.10 Design units are used to specify the _____ of a `Font`.

a) `Size` b) `Name`
c) `FontFamily` d) `Style`

Answers: 26.1) c. 26.2) b. 26.3) a. 26.4) d. 26.5) a. 26.6) b. 26.7) a. 26.8) c. 26.9) b. 26.10) a.

EXERCISES

26.11 (***Check Writer Modified to Print Background Images***) Modify the **Check Writer** application to display and print a background for the check. The GUI should look similar to Fig. 26.32. Users can select a background image. The image should appear in the **Print preview** dialog box and also should print as a background to the check.

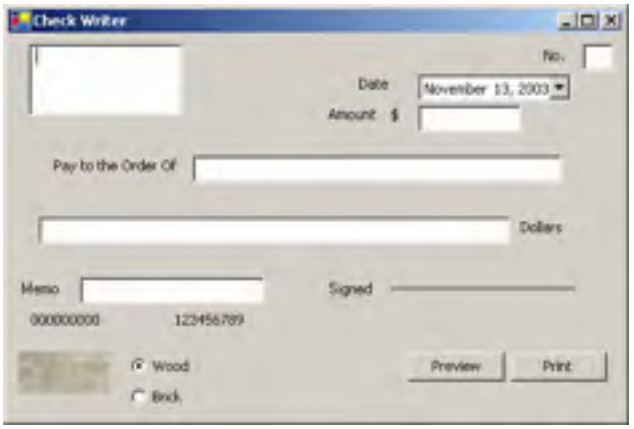


Figure 26.32 Modified **Check Writer** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial26\Exercises\ModifiedCheckWriter` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CheckWriter.sln` in the `ModifiedCheckWriter` directory to open the application.

c) ***Adding an instance variable.*** Add an instance variable of the `string` type that will contain the name of the background image.

d) ***Defining the `Load` event handler.*** Double click an empty area of the `Form` to create its `Load` event handler. Define the event handler to initialize the `string` instance variable to "`wood.jpg`". Set the `Image` property of the `PictureBox` to this image. [*Hint*: use the `FromFile` method of the `Image` class.]

e) ***Defining the `CheckedChanged` event handler.*** Double click the **Wood** `RadioButton` to create its `CheckedChanged` event handler. Define the event handler to notify the application when users have made a background selection. If the **Wood** `RadioButton` is selected, then a preview of the wooden background should display in the `picPreview PictureBox`. Otherwise, if the **Brick** `RadioButton` is selected, then a preview of the brick background should display in the `picPreview PictureBox`.

f) ***Modifying the `objPrintDocument_PrintPage` event handler.*** Modify the `objPrintDocument_PrintPage` event handler to print the background image. [*Hint*: Use the `DrawImage` method to display the background image to print. `DrawImage` takes five arguments: The image file, the *x*-coordinate, the *y*-coordinate, the width and the height.] To print the image in the background, the `DrawImage` method must be the first method called on the `Graphics` object.

g) ***Running the application.*** Select **Debug > Start** to run your application. Fill out the information for a check. Preview and print the check with each background to ensure that your application works correctly.

h) ***Closing the application.*** Close your running application by clicking its close box.

i) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 26.11 Solution
2  // CheckWriter.cs (Modified)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10 using System.Drawing.Printing;
11
12 namespace CheckWriter
```

```
13  {
14     /// <summary>
15     /// Summary description for FrmCheckWriter.
16     /// </summary>
17     public class FrmCheckWriter : System.Windows.Forms.Form
18     {
19        // Label to display payer information
20        private System.Windows.Forms.TextBox txtPayer;
21
22        // Label and TextBox to display number of check
23        private System.Windows.Forms.Label lblNumber;
24        private System.Windows.Forms.TextBox txtNumber;
25
26        // Label and DateTimePicker to display date of check
27        private System.Windows.Forms.Label lblDate;
28        private System.Windows.Forms.DateTimePicker dtpDate;
29
30        // Label and TextBox to display check amount
31        private System.Windows.Forms.Label lblAmount;
32        private System.Windows.Forms.TextBox txtAmount;
33
34        // Label and TextBox to display recipient
35        private System.Windows.Forms.Label lblPayee;
36        private System.Windows.Forms.TextBox txtPayee;
37
38        // TextBox and Label to display written check amount
39        private System.Windows.Forms.TextBox txtPayment;
40        private System.Windows.Forms.Label lblDollars;
41
42        // Label and TextBox to display memo text
43        private System.Windows.Forms.Label lblMemo;
44        private System.Windows.Forms.TextBox txtMemo;
45
46        // Labels to display MICR numbers
47        private System.Windows.Forms.Label lblABA;
48        private System.Windows.Forms.Label lblAccount;
49
50        // Labels to display signature
51        private System.Windows.Forms.Label lblSigned;
52        private System.Windows.Forms.Label lblUnderline;
53
54        // Button to preview the check
55        private System.Windows.Forms.Button btnPreview;
56
57        // Button to print the check
58        private System.Windows.Forms.Button btnPrint;
59
60        // PrintPreviewDialog to print and preview the check
61        private System.Windows.Forms.PrintPreviewDialog objPreview;
62
63        // PictureBox to display background image
64        private System.Windows.Forms.PictureBox picPreview;
65
66        // RadioButtons to choose background for the check
67        private System.Windows.Forms.RadioButton radWood;
68        private System.Windows.Forms.RadioButton radBrick;
69
70        /// <summary>
71        /// Required designer variable.
72        /// </summary>
```

```
73          private System.ComponentModel.Container components = null;
74
75          private Font m_objFont; // instance variable to store font
76          private string m_strPath;
77
78          public FrmCheckWriter()
79          {
80             //
81             // Required for Windows Form Designer support
82             //
83             InitializeComponent();
84
85             //
86             // TODO: Add any constructor code after InitializeComponent
87             // call
88             //
89          }
90
91          /// <summary>
92          /// Clean up any resources being used.
93          /// </summary>
94          protected override void Dispose( bool disposing )
95          {
96             if( disposing )
97             {
98                if (components != null)
99                {
100                  components.Dispose();
101               }
102            }
103            base.Dispose( disposing );
104         }
105
106         // Windows Form Designer generated code
107
108         /// <summary>
109         /// The main entry point for the application.
110         /// </summary>
111         [STAThread]
112         static void Main()
113         {
114            Application.Run( new FrmCheckWriter() );
115         }
116
117         // PrintPage event raised for each page to be printed
118         private void objPrintDocument_PrintPage(
119            Object sender, PrintPageEventArgs e )
120         {
121            float fltYPosition;
122            float fltXPosition;
123
124            // represent left margin of page
125            float fltLeftMargin = e.MarginBounds.Left;
126
127            // represent top margin of page
128            float fltTopMargin = e.MarginBounds.Top;
129            string strLine = "";
130
131            // if m_strPath has value, display
132            // specified image on Image control
```

```
133         if ( m_strPath != "" )
134         {
135            Image objImage;
136            objImage = Image.FromFile( m_strPath );
137
138            // print image so it is the rear-most object
139            e.Graphics.DrawImage( Image.FromFile( m_strPath ),
140               fltLeftMargin, fltTopMargin, this.Width,
141               this.Height - 60 );
142         }
143
144         // iterate over the form, printing each control
145         foreach ( Control objControl in this.Controls )
146         {
147            // we do not want to print buttons
148            if ( objControl.GetType().Name != "Button" )
149            {
150               strLine = objControl.Text;
151
152               switch ( objControl.Name )
153               {
154                  // underline the date
155                  case "dtpDate":
156                     m_objFont = new Font( "Tahoma", ( float ) 8.25,
157                        FontStyle.Underline );
158                     break;
159
160                  // draw a box around amount
161                  case "txtAmount":
162                     e.Graphics.DrawRectangle( Pens.Black,
163                        txtAmount.Location.X + fltLeftMargin,
164                        txtAmount.Location.Y + fltTopMargin - 4,
165                        txtAmount.Width, txtAmount.Height );
166
167                     m_objFont = objControl.Font; // default font
168                     break;
169
170                  default:
171                     m_objFont = objControl.Font; // default font
172                     break;
173
174               } // end switch
175
176               // set string positions relative to page margins
177               fltXPosition = fltLeftMargin + objControl.Location.X;
178               fltYPosition = fltTopMargin + objControl.Location.Y;
179
180               // draw text in graphics object
181               e.Graphics.DrawString( strLine, m_objFont,
182                  Brushes.Black, fltXPosition, fltYPosition );
183
184            } // end if
185
186         } // end foreach
187
188         // draw box around check
189         e.Graphics.DrawRectangle( Pens.Black, fltLeftMargin,
190            fltTopMargin, this.Width, this.Height - 60 );
191
192         // indicate that there are no more pages to print
```

```
193            e.HasMorePages = false;
194
195         } // end method objPrintDocument_PrintPage
196
197         // print document
198         private void btnPrint_Click(
199            object sender, System.EventArgs e )
200         {
201            // create new object to assist in printing
202            PrintDocument objPrintDocument = new PrintDocument();
203
204            // add PrintPage event handler
205            objPrintDocument.PrintPage += new PrintPageEventHandler(
206               objPrintDocument_PrintPage );
207
208            // if no printers installed, display error message
209            if ( PrinterSettings.InstalledPrinters.Count == 0 )
210            {
211               ErrorMessage();
212               return; // exit event handler
213            }
214
215            // print the document
216            objPrintDocument.Print();
217
218         } // end method btnPrint_Click
219
220         // display document in print preview dialog
221         private void btnPreview_Click(
222            object sender, System.EventArgs e )
223         {
224            // create new object to assist in previewing
225            PrintDocument objPrintDocument = new PrintDocument();
226
227            // add PrintPage event handler
228            objPrintDocument.PrintPage += new PrintPageEventHandler(
229               objPrintDocument_PrintPage );
230
231            // if no printers installed, display error message
232            if ( PrinterSettings.InstalledPrinters.Count == 0 )
233            {
234               ErrorMessage();
235               return; // exit event handler
236            }
237
238            objPreview.Document = objPrintDocument; // specify document
239            objPreview.ShowDialog(); // show print preview
240
241         } // end method btnPreview_Click
242
243         // display an error message to the user
244         void ErrorMessage()
245         {
246            MessageBox.Show( "No printers installed. You must " +
247               "have a printer installed to preview or print " +
248               "the document.", "Print Error",
249               MessageBoxButtons.OK, MessageBoxIcon.Error );
250
251         } // end method ErrorMessage
252
```

```
253       // handles Form's Load event
254       private void FrmCheckWriter_Load(
255          object sender, System.EventArgs e )
256       {
257          // set PictureBox image
258          m_strPath = "wood.jpg";
259          picPreview.Image = Image.FromFile( m_strPath );
260
261       } // end method FrmCheckWriter_Load
262
263       // display selected image in Image control
264       private void radWood_CheckedChanged(
265          object sender, System.EventArgs e )
266       {
267          // if Wood RadioButton is selected
268          // then set m_strPath to wood image
269          if ( radWood.Checked == true )
270          {
271             m_strPath = "wood.jpg";
272          }
273          // otherwise set m_strPath to brick image
274          else
275          {
276             m_strPath = "bricks.jpg";
277          }
278
279          // set PictureBox image
280          picPreview.Image = Image.FromFile( m_strPath );
281
282       } // end method radWood_CheckedChanged
283
284    } // end class FrmCheckWriter
285 }
```

26.12 (***Company Logo Application***) Develop a **Company Logo** application that allows users to design a company logo (Fig. 26.33).

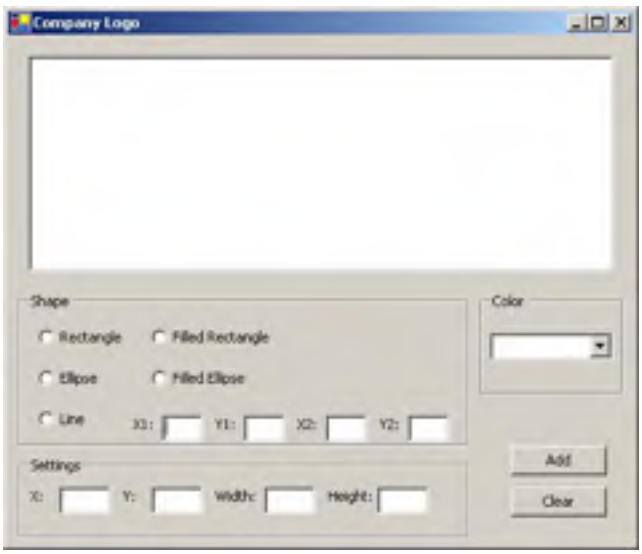


Figure 26.33 **Company Logo** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial26\Exercises\CompanyLogo` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CompanyLogo.sln` in the `CompanyLogo` directory to open the application.

c) ***Defining the Add `Button`'s `Click` event handler.*** Create the **Add** `Button`'s `Click` event handler. Define the event handler so that the shape the user specifies is drawn on the `PictureBox`. Use the `CreateGraphics` method on the `PictureBox` to retrieve the `Graphics` object used to draw on the `PictureBox`. [*Note*: The `TextBox`es labelled **X1:**, **Y1:**, **X2:** and **Y2:** must be filled out to draw a line. The `TextBox`es labelled **X:**, **Y:**, **Width:** and **Height:** must be filled out to draw a rectangle, filled rectangle, ellipse, or filled ellipse.]

d) ***Defining the Clear `Button`'s `Click` event handler.*** Create the **Clear** `Button`'s `Click` event handler, and define it so that the `PictureBox` is cleared. [*Hint*: To clear the entire `PictureBox`, use the `PictureBox`'s `Invalidate` method. The `Invalidate` method is often used to refresh (update) the graphics of a control. By using the `Invalidate` method without specifying a graphic to draw, the `PictureBox` clears.] Also ensure that all `TextBox`es are cleared when the **Clear** `Button` is clicked.

e) ***Running the application.*** Select **Debug > Start** to run your application. Draw each shape, using various colors, and test the **Add** and **Clear** `Button`s to ensure that your application works correctly.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 26.12 Solution
// CompanyLogo.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CompanyLogo
{
   /// <summary>
   /// Summary description for FrmCompanyLogo.
   /// </summary>
   public class FrmCompanyLogo : System.Windows.Forms.Form
   {
      // PictureBox to display the company logo
      private System.Windows.Forms.PictureBox picImage;

      // GroupBox to add shapes to the logo
      private System.Windows.Forms.GroupBox fraShape;

      // RadioButtons to choose a shape to create
      private System.Windows.Forms.RadioButton radRectangle;
      private System.Windows.Forms.RadioButton radFilledRectangle;
      private System.Windows.Forms.RadioButton radEllipse;
      private System.Windows.Forms.RadioButton radFilledEllipse;
      private System.Windows.Forms.RadioButton radLine;

      // Labels and TextBoxes to input two points to connect
      // to create a line segment
      private System.Windows.Forms.Label lblX1;
      private System.Windows.Forms.TextBox txtX1;
      private System.Windows.Forms.Label lblY1;
      private System.Windows.Forms.TextBox txtY1;
      private System.Windows.Forms.Label lblX2;
```

```
38          private System.Windows.Forms.TextBox txtX2;
39          private System.Windows.Forms.Label lblY2;
40          private System.Windows.Forms.TextBox txtY2;
41
42          // GroupBox containing Labels and TextBoxes to choose
43          // the dimensions of a shape
44          private System.Windows.Forms.GroupBox fraSettings;
45          private System.Windows.Forms.Label lblX;
46          private System.Windows.Forms.TextBox txtXPosition;
47          private System.Windows.Forms.Label lblY;
48          private System.Windows.Forms.TextBox txtYPosition;
49          private System.Windows.Forms.Label lblWidth;
50          private System.Windows.Forms.TextBox txtWidth;
51          private System.Windows.Forms.Label lblHeight;
52          private System.Windows.Forms.TextBox txtHeight;
53
54          // GroupBox containing a ComboBox to choose the color
55          // of the shape
56          private System.Windows.Forms.GroupBox fraColor;
57          private System.Windows.Forms.ComboBox cboColor;
58
59          // Button to add a shape to the logo
60          private System.Windows.Forms.Button btnAdd;
61
62          // Button to clear the company logo
63          private System.Windows.Forms.Button btnClear;
64
65          /// <summary>
66          /// Required designer variable.
67          /// </summary>
68          private System.ComponentModel.Container components = null;
69
70          public FrmCompanyLogo()
71          {
72             //
73             // Required for Windows Form Designer support
74             //
75             InitializeComponent();
76
77             //
78             // TODO: Add any constructor code after InitializeComponent
79             // call
80             //
81          }
82
83          /// <summary>
84          /// Clean up any resources being used.
85          /// </summary>
86          protected override void Dispose( bool disposing )
87          {
88             if( disposing )
89             {
90                if (components != null)
91                {
92                   components.Dispose();
93                }
94             }
95             base.Dispose( disposing );
96          }
97
```

```
98          // Windows Form Designer generated code
99
100         /// <summary>
101         /// The main entry point for the application.
102         /// </summary>
103         [STAThread]
104         static void Main()
105         {
106            Application.Run( new FrmCompanyLogo() );
107         }
108
109         // adds shapes specified by users to the PictureBox
110         private void btnAdd_Click(
111            object sender, System.EventArgs e )
112         {
113            // create a graphics object to draw shapes
114            Graphics objgraphics = picImage.CreateGraphics();
115
116            // create brush object used for solid shapes
117            SolidBrush objBrush = new SolidBrush( Color.FromName(
118               Convert.ToString( cboColor.SelectedItem ) ) );
119
120            // create pen object used for unfilled shapes
121            Pen objPen = new Pen( objBrush );
122
123            // create rectangle
124            if ( radRectangle.Checked == true )
125            {
126               // determine if all values are provided
127               if ( txtXPosition.Text != "" && txtYPosition.Text != ""
128                  && txtWidth.Text != "" && txtHeight.Text != "" )
129               {
130                  objgraphics.DrawRectangle( objPen,
131                     Int32.Parse( txtXPosition.Text ),
132                     Int32.Parse( txtYPosition.Text ),
133                     Int32.Parse( txtWidth.Text ),
134                     Int32.Parse( txtHeight.Text ) );
135               }
136               else
137               {
138                  MessageBox.Show(
139                     "You did not provide all the setting values",
140                     "Missing setting values", MessageBoxButtons.OK,
141                     MessageBoxIcon.Information );
142               }
143            }
144            else if ( radFilledRectangle.Checked == true )
145            {
146               // determine if all values are provided
147               if ( txtXPosition.Text != "" && txtYPosition.Text != ""
148                  && txtWidth.Text != "" && txtHeight.Text != "" )
149               {
150                  // create filled rectangle
151                  objgraphics.FillRectangle( objBrush,
152                     Int32.Parse( txtXPosition.Text ),
153                     Int32.Parse( txtYPosition.Text ),
154                     Int32.Parse( txtWidth.Text ),
155                     Int32.Parse( txtHeight.Text ) );
156               }
157               else
```

```
158             {
159                MessageBox.Show(
160                   "You did not provide all the setting values",
161                   "Missing setting values", MessageBoxButtons.OK,
162                   MessageBoxIcon.Information );
163             }
164          }
165          else if ( radEllipse.Checked == true )
166          {
167             // determine if all values are provided
168             if ( txtXPosition.Text != "" && txtYPosition.Text != ""
169                && txtWidth.Text != "" && txtHeight.Text != "" )
170             {
171                // create filled ellipse
172                objgraphics.DrawEllipse( objPen,
173                   Int32.Parse( txtXPosition.Text ),
174                   Int32.Parse( txtYPosition.Text ),
175                   Int32.Parse( txtWidth.Text ),
176                   Int32.Parse( txtHeight.Text ) );
177             }
178             else
179             {
180                MessageBox.Show(
181                   "You did not provide all the setting values",
182                   "Missing setting values", MessageBoxButtons.OK,
183                   MessageBoxIcon.Information );
184             }
185          }
186          else if ( radFilledEllipse.Checked == true )
187          {
188             // determine if all values are provided
189             if ( txtXPosition.Text != "" && txtYPosition.Text != ""
190                && txtWidth.Text != "" && txtHeight.Text != "" )
191             {
192                // draw ellipse
193                objgraphics.FillEllipse( objBrush,
194                   Int32.Parse( txtXPosition.Text ),
195                   Int32.Parse( txtYPosition.Text ),
196                   Int32.Parse( txtWidth.Text ),
197                   Int32.Parse( txtHeight.Text ) );
198             }
199             else
200             {
201                MessageBox.Show(
202                   "You did not provide all the setting values",
203                   "Missing setting values", MessageBoxButtons.OK,
204                   MessageBoxIcon.Information );
205             }
206          }
207          else if ( radLine.Checked == true )
208          {
209             // determine if all values are provided
210             if ( txtX1.Text != "" && txtY1.Text != "" &&
211                txtX2.Text != "" && txtY2.Text != "" )
212             {
213                // draw line
214                objgraphics.DrawLine( objPen,
215                   Int32.Parse( txtX1.Text ),
216                   Int32.Parse( txtY1.Text ),
217                   Int32.Parse( txtX2.Text ),
```

```
218                Int32.Parse( txtY2.Text ) );
219             }
220             else
221             {
222                MessageBox.Show(
223                   "You did not provide all the x and y values",
224                   "Missing X Y values", MessageBoxButtons.OK,
225                   MessageBoxIcon.Information );
226             }
227          }
228
229       } // end method btnAdd_Click
230
231       // clear the application GUI
232       private void btnClear_Click(
233          object sender, System.EventArgs e )
234       {
235          // clear all fields
236          txtX1.Text = "";
237          txtX2.Text = "";
238          txtY1.Text = "";
239          txtY2.Text = "";
240          txtXPosition.Text = "";
241          txtYPosition.Text = "";
242          txtWidth.Text = "";
243          txtHeight.Text = "";
244
245          // clear the PictureBox
246          picImage.Invalidate();
247
248       } // end method btnClear_Click
249
250    } // end class FrmCompanyLogo
251 }
```

26.13 (*Letterhead Application*) Create a **Letterhead** application that allows users to design stationery for company documents (Fig. 26.34). Allow users to specify the image that will serve as the letterhead.

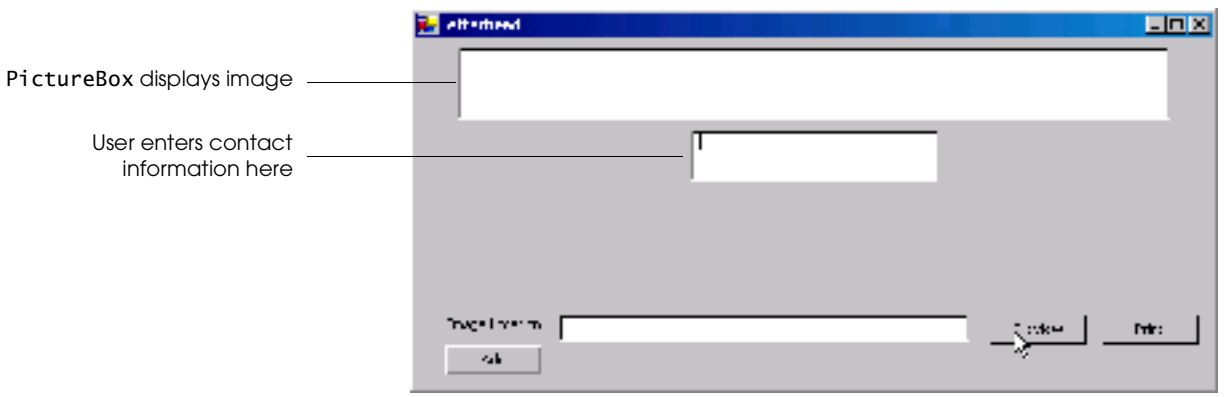


Figure 26.34 **Letterhead** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial26\Exercises\Letterhead` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Letterhead.sln` in the `Letterhead` directory to open the application.

c) ***Creating a PrintPreviewDialog control.*** Add a `PrintPreviewDialog` control to allow users to preview the letterhead before it is printed. Rearrange and comment the control declaration appropriately.

d) ***Defining the PrintPage event handler.*** Allow users to print the document by defining the `PrintPage` event handler as you did in the **Check Writer** application.

e) ***Defining the btnPrint_Click event handler.*** The `btnPrint_Click` event handler should tell the `PrintDocument` where to find the `PrintPage` event handler, as in the **Check Writer** application, and print the document.

f) ***Defining the btnPreview_Click event handler.*** The `btnPreview_Click` event handler should tell the `PrintDocument` where to find the `PrintPage` event handler, as in the **Check Writer** application, and then show the preview dialog.

g) ***Defining the btnAdd_Click event handler.*** The `btnAdd_Click` event handler should set the `PictureBox`'s `Image` property to the value entered in the `TextBox`. [*Hint*: use the `FromFile` method of the `Image` class.] If no value was entered, display an error message.

h) ***Running the application.*** Select **Debug > Start** to run your application. The `Letterhead.png` image file, located in `C:\Examples\Tutorial26\Exercises\Images`, has been provided for you to test the application's letterhead image capability. Preview and print the letterhead to ensure that your application works correctly.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 26.13 Solution
// Letterhead.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Drawing.Printing;

namespace Letterhead
{
   /// <summary>
   /// Summary description for FrmLetterHead.
   /// </summary>
   public class FrmLetterHead : System.Windows.Forms.Form
   {
      // PictureBox to display the letterhead
      private System.Windows.Forms.PictureBox picImage;

      // TextBox to input letterhead text
      private System.Windows.Forms.TextBox txtInformation;

      // Label and TextBox to input image location
      private System.Windows.Forms.Label lblImage;
      private System.Windows.Forms.TextBox txtImage;

      // Button to add the image
      private System.Windows.Forms.Button btnAdd;

      // Button to preview the letterhead
      private System.Windows.Forms.Button btnPreview;

      // Button to print the letterhead
      private System.Windows.Forms.Button btnPrint;
```

```
37
38          // PrintPreviewDialog to preview and print the letterhead
39          private System.Windows.Forms.PrintPreviewDialog objPreview;
40
41          /// <summary>
42          /// Required designer variable.
43          /// </summary>
44          private System.ComponentModel.Container components = null;
45
46          // create font object
47          Font m_objFont;
48
49          public FrmLetterHead()
50          {
51             //
52             // Required for Windows Form Designer support
53             //
54             InitializeComponent();
55
56             //
57             // TODO: Add any constructor code after InitializeComponent
58             // call
59             //
60          }
61
62          /// <summary>
63          /// Clean up any resources being used.
64          /// </summary>
65          protected override void Dispose( bool disposing )
66          {
67             if( disposing )
68             {
69                if (components != null)
70                {
71                   components.Dispose();
72                }
73             }
74             base.Dispose( disposing );
75          }
76
77          // Windows Form Designer generated code
78
79          /// <summary>
80          /// The main entry point for the application.
81          /// </summary>
82          [STAThread]
83          static void Main()
84          {
85             Application.Run( new FrmLetterHead() );
86          }
87
88          // PrintPage event raised for each page to be printed.
89          private void objPrintDocument_PrintPage(
90             object sender, PrintPageEventArgs e )
91          {
92             float fltYPosition;
93             float fltXPosition;
94             float fltLeftMargin = e.MarginBounds.Left;
95             float fltTopMargin = e.MarginBounds.Top;
96
```

```
97          string strPath;
98
99          // get location of image
100         strPath = txtImage.Text;
101
102         // make sure image location was provided
103         if ( strPath != "" )
104         {
105            Image objImage;
106            objImage = Image.FromFile( strPath );
107
108            // print image so it is on top of page
109            e.Graphics.DrawImage( Image.FromFile( strPath ),
110               fltLeftMargin + picImage.Location.X,
111               fltTopMargin + picImage.Location.Y,
112               picImage.Size.Width, picImage.Size.Height );
113         }
114
115         // if contact information is provided, print data
116         if ( txtInformation.Text != "" )
117         {
118            // specifies font of text
119            m_objFont = new Font( "Tahoma", 12, FontStyle.Bold );
120
121            fltXPosition = fltLeftMargin + txtInformation.Location.X;
122            fltYPosition = fltTopMargin + txtInformation.Location.Y;
123
124            // print information
125            e.Graphics.DrawString( txtInformation.Text, m_objFont,
126               Brushes.Black, fltXPosition, fltYPosition );
127         }
128
129         // indicate there are no more pages to print
130         e.HasMorePages = false;
131
132      } // end method objPrintDocument_PrintPage
133
134      // print the document
135      private void btnPrint_Click(
136         object sender, System.EventArgs e )
137      {
138         // create new object to assist in printing
139         PrintDocument objPrintDocument = new PrintDocument();
140
141         // add PrintPage event handler
142         objPrintDocument.PrintPage += new PrintPageEventHandler(
143            objPrintDocument_PrintPage );
144
145         // print the document
146         objPrintDocument.Print();
147
148      } // end method btnPrint_Click
149
150      // display document in print preview dialog
151      private void btnPreview_Click(
152         object sender, System.EventArgs e )
153      {
154         // create new object to assist in previewing
155         PrintDocument objPrintDocument = new PrintDocument();
156
```

```
157          // add PrintPage event handler
158          objPrintDocument.PrintPage += new PrintPageEventHandler(
159             objPrintDocument_PrintPage );
160
161          objPreview.Document = objPrintDocument; // specify document
162          objPreview.ShowDialog(); // show print preview
163
164       } // end method btnPreview_Click
165
166       // add the specified image to the Image control
167       private void btnAdd_Click(
168          object sender, System.EventArgs e )
169       {
170          // import the image specified
171          if ( txtImage.Text != "" )
172          {
173             picImage.Image = Image.FromFile( txtImage.Text );
174          }
175          else
176          {
177             MessageBox.Show( "You must enter a path for your image.",
178                "Input Error", MessageBoxButtons.OK,
179                MessageBoxIcon.Information );
180          }
181
182       } // end method btnAdd_Click
183
184    } // end class FrmLetterHead
185 }
```

What does this code do? ▶

26.14 What is the result of the following code? Assume that `objOutput_PrintPage` is defined.

```
1  private void btnPrint_Click(
2     object sender, System.EventArgs e )
3  {
4     PrintDocument objOutput = new PrintDocument();
5
6     // add PrintPage event handler
7     objOutput.PrintPage += new PrintPageEventHandler(
8        objOutput_PrintPage );
9
10    objOutput.Print();
11
12 } // end method btnPrint_Click
```

Answer: The code indicates that the `PrintPage` event for `objOutput` should invoke the `objOutput_PrintPage` event handler. The complete code reads:

```
1  // Exercise 26.14 Solution
2  // PrintDocument.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
```

```
 9  using System.Data;
10  using System.Drawing.Printing;
11
12  namespace PrntDocument
13  {
14     /// <summary>
15     /// Summary description for FrmPrintDocument.
16     /// </summary>
17     public class FrmPrintDocument : System.Windows.Forms.Form
18     {
19        private System.Windows.Forms.Label lblinput;
20        private System.Windows.Forms.TextBox txtInput;
21        private System.Windows.Forms.Button btnPrint;
22        /// <summary>
23        /// Required designer variable.
24        /// </summary>
25        private System.ComponentModel.Container components = null;
26
27        public FrmPrintDocument()
28        {
29           //
30           // Required for Windows Form Designer support
31           //
32           InitializeComponent();
33
34           //
35           // TODO: Add any constructor code after InitializeComponent
36           // call
37           //
38        }
39
40        /// <summary>
41        /// Clean up any resources being used.
42        /// </summary>
43        protected override void Dispose( bool disposing )
44        {
45           if( disposing )
46           {
47              if (components != null)
48              {
49                 components.Dispose();
50              }
51           }
52           base.Dispose( disposing );
53        }
54
55        // Windows Form Designer generated code
56
57        /// <summary>
58        /// The main entry point for the application.
59        /// </summary>
60        [STAThread]
61        static void Main()
62        {
63           Application.Run( new FrmPrintDocument() );
64        }
65
66        // defines an event handler for the PrintPage event
67        private void btnPrint_Click(
68           object sender, System.EventArgs e )
```

```
69          {
70             PrintDocument objOutput = new PrintDocument();
71
72             // add PrintPage event handler
73             objOutput.PrintPage += new PrintPageEventHandler(
74                objOutput_PrintPage );
75
76             objOutput.Print();
77
78          } // end method btnPrint_Click
79
80          // prints an empty page
81          private void objOutput_PrintPage( object sender,
82             System.Drawing.Printing.PrintPageEventArgs e )
83          {
84             // if no printers installed, display error message
85             if ( PrinterSettings.InstalledPrinters.Count == 0 )
86             {
87                MessageBox.Show( "Printer Unavailable", "Error",
88                   MessageBoxButtons.RetryCancel,
89                   MessageBoxIcon.Error );
90
91                return; // exit event handler
92             }
93
94          } // end method objOutput_PrintPage
95
96       } // end class FrmPrintDocument
97    }
```

What's wrong with this code?

26.15 Find the error(s) in the following code. This is the definition for a `Click` event handler for a `Button`. This event handler should draw a filled rectangle on a `PictureBox` control.

```
1   private void btnDrawImage_Click(
2       object sender, System.EventArgs e)
3   {
4      // create an orange colored brush
5      SolidBrush objBrush = new SolidBrush( Orange );
6
7      // create a Graphics object to draw on the PictureBox
8      Graphics objGraphics = picPictureBox.AcquireGraphics();
9
10     // draw a filled rectangle
11     objGraphics.FillRectangle( objBrush, 2, 3, 40, 30 );
12
13  } // end method btnDrawImage_Click
```

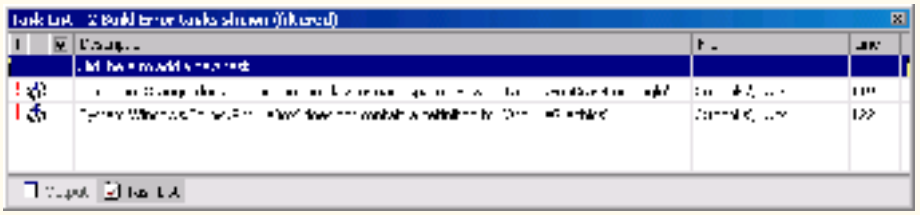
Answer: When specifying a color for the `SolidBrush` you must precede the color name with "`Color.`" You cannot just write the color name. Also, to retrieve the `Graphics` object, the

CreateGraphics method must be used, not AcquireGraphics. The complete incorrect code reads:

```
// Exercise 25.15 Solution
// DrawRectangle.cs (Incorrect)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace DrawRectangle
{
   /// <summary>
   /// Summary description for FrmDrawRectangle.
   /// </summary>
   public class FrmDrawRectangle : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Button btnDrawImage;
      private System.Windows.Forms.Label lblImage;
      private System.Windows.Forms.PictureBox picPictureBox;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmDrawRectangle()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
```

The argument to `SolidBrush` should be `Color.Orange`, and `AcquireGraphics` should be `CreateGraphics`

```
59          [STAThread]
60          static void Main()
61          {
62             Application.Run( new FrmDrawRectangle() );
63          }
64
65          // draws a rectangle in the picture box
66          private void btnDrawImage_Click(
67             object sender, System.EventArgs e )
68          {
69             // create an orange colored brush
70             SolidBrush objBrush = new SolidBrush( Orange );
71
72             // create a Graphics object to draw on the PictureBox
73             Graphics objGraphics = picPictureBox.AcquireGraphics();
74
75             // draw a filled rectangle
76             objGraphics.FillRectangle( objBrush, 2, 3, 40, 30 );
77
78          } // end method btnDrawImage_Click
79
80       } // end class FrmDrawRectangle
81    }
```



Answer: The complete corrected code should read:

```
1   // Exercise 25.15 Solution
2   // DrawRectangle.cs (Correct)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace DrawRectangle
12  {
13     /// <summary>
14     /// Summary description for FrmDrawRectangle.
15     /// </summary>
16     public class FrmDrawRectangle : System.Windows.Forms.Form
17     {
18        private System.Windows.Forms.Button btnDrawImage;
19        private System.Windows.Forms.Label lblImage;
20        private System.Windows.Forms.PictureBox picPictureBox;
21        /// <summary>
22        /// Required designer variable.
23        /// </summary>
24        private System.ComponentModel.Container components = null;
```

```
25
26          public FrmDrawRectangle()
27          {
28             //
29             // Required for Windows Form Designer support
30             //
31             InitializeComponent();
32
33             //
34             // TODO: Add any constructor code after InitializeComponent
35             // call
36             //
37          }
38
39          /// <summary>
40          /// Clean up any resources being used.
41          /// </summary>
42          protected override void Dispose( bool disposing )
43          {
44             if( disposing )
45             {
46                if (components != null)
47                {
48                   components.Dispose();
49                }
50             }
51             base.Dispose( disposing );
52          }
53
54          // Windows Form Designer generated code
55
56          /// <summary>
57          /// The main entry point for the application.
58          /// </summary>
59          [STAThread]
60          static void Main()
61          {
62             Application.Run( new FrmDrawRectangle() );
63          }
64
65          // draws a rectangle in the picture box
66          private void btnDrawImage_Click(
67             object sender, System.EventArgs e )
68          {
69             // create an orange colored brush
70             SolidBrush objBrush = new SolidBrush( Color.Orange );
71
72             // create a Graphics object to draw on the PictureBox
73             Graphics objGraphics = picPictureBox.CreateGraphics();
74
75             // draw a filled rectangle
76             objGraphics.FillRectangle( objBrush, 2, 3, 40, 30 );
77
78          } // end method btnDrawImage_Click
79
80       } // end class FrmDrawRectangle
81    }
```

Programming Challenge

26.16 (***Screen Saver Application***) Develop an application that simulates a screen saver. This application should add random-colored, random-sized, solid and hollow shapes at different positions of the screen. Copy the `C:\Exercises\Tutorial26\ScreenSaver` directory, and place it in your `C:\SimplyCSP` directory. The design of the `Form` has been created, which consists of a black `Form` and a `Timer` control. In the `ScreenSaver.cs` code view, the `DisplayShape` method has been provided, and the `Timer`'s `tick` event handler has already been defined for you.

You must write the rest of the `DisplayShape` method code. Create the `Graphics` object from the `Form` using the `Form`'s `CreateGraphics` method, and specify random colors, sizes and positions for the filled and hollow shapes that will be displayed on the screen. The width and height of the shapes should be no larger than 100 pixels.

Answer:

```
1  // Exercise 26.16 Solution
2  // ScreenSaver.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ScreenSaver
12 {
13    /// <summary>
14    /// Summary description for FrmShapeChanger.
15    /// </summary>
16    public class FrmShapeChanger : System.Windows.Forms.Form
17    {
18       // Timer to update screen saver at intervals
19       private System.Windows.Forms.Timer tmrScreenSaver;
20
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.IContainer components;
25
26       private double m_dblCount = 0.0;
27
28       public FrmShapeChanger()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34
```

```
35          //
36          // TODO: Add any constructor code after InitializeComponent
37          // call
38          //
39       }
40
41       /// <summary>
42       /// Clean up any resources being used.
43       /// </summary>
44       protected override void Dispose( bool disposing )
45       {
46          if( disposing )
47          {
48             if (components != null)
49             {
50                components.Dispose();
51             }
52          }
53          base.Dispose( disposing );
54       }
55
56       // Windows Form Designer generated code
57
58       /// <summary>
59       /// The main entry point for the application.
60       /// </summary>
61       [STAThread]
62       static void Main()
63       {
64          Application.Run( new FrmShapeChanger() );
65       }
66
67       // create a specified graphic on the form
68       private void DisplayShape()
69       {
70          // create a Graphics object
71          Graphics objGraphicsObject = this.CreateGraphics();
72
73          // create random object for random number generation
74          Random objRandom = new Random();
75
76          // create random color
77          Color objColor = Color.FromArgb(
78             objRandom.Next( 0, 255 ), objRandom.Next( 0, 255 ),
79             objRandom.Next( 0, 255 ) );
80
81          // create brush object used for solid shapes
82          SolidBrush objBrush = new SolidBrush( objColor );
83
84          // create pen object used for unfilled shapes
85          Pen objPen = new Pen( objBrush );
86
87          // create random number used to create random shape
88          int intShape = objRandom.Next( 0, 5 );
89
90          // set to width of form
91          int intWidth = this.Size.Width;
92
93          // set to height of form
94          int intHeight = this.Size.Height;
```

```
95
96          // decide which shape to draw
97          switch ( intShape )
98          {
99             case 0:
100
101               // create filled rectangle
102               objGraphicsObject.FillRectangle( objBrush,
103                  objRandom.Next( 0, intWidth ),
104                  objRandom.Next( 0, intHeight ),
105                  objRandom.Next( 10, 100 ),
106                  objRandom.Next( 10, 100 ) );
107               break;
108
109            case 1:
110
111               // create filled ellipse
112               objGraphicsObject.FillEllipse( objBrush,
113                  objRandom.Next( 0, intWidth ),
114                  objRandom.Next( 0, intHeight ),
115                  objRandom.Next( 10, 100 ),
116                  objRandom.Next( 10, 100 ) );
117               break;
118
119            case 2:
120
121               // draw ellipse
122               objGraphicsObject.DrawEllipse( objPen,
123                  objRandom.Next( 0, intWidth ),
124                  objRandom.Next( 0, intHeight ),
125                  objRandom.Next( 10, 100 ),
126                  objRandom.Next( 10, 100 ) );
127               break;
128
129            case 3:
130
131               // draw rectangle
132               objGraphicsObject.DrawRectangle( objPen,
133                  objRandom.Next( 0, intWidth ),
134                  objRandom.Next( 0, intHeight ),
135                  objRandom.Next( 10, 100 ),
136                  objRandom.Next( 10, 100 ) );
137               break;
138
139            case 4:
140
141               // draw line
142               objGraphicsObject.DrawLine( objPen,
143                  objRandom.Next( 0, intWidth ),
144                  objRandom.Next( 0, intHeight ),
145                  objRandom.Next( 10, intWidth ),
146                  objRandom.Next( 10, intHeight )
147               break;
148
149         } // end switch
150
151      } // end method DisplayShape
152
153      // invoked each time tmrScreenSaver ticks
154      private void tmrScreenSaver_Tick(
```

```
155          object sender, System.EventArgs e )
156       {
157          m_dblCount += 0.25;
158
159          // draw shape every half second
160          if ( m_dblCount % 2.5 == 0 )
161          {
162             DisplayShape(); // draws another shape
163          }
164
165       } // end method tmrScreenSaver_Tick
166
167    } // end class FrmShapeChanger
168 }
```

26.17 (*Screen Saver Enhancement Application*) Enhance the **Screen Saver** application from Exercise 26.16 by modifying the `Timer` control's `Tick` event handler. Add code to this event handler so that after a specified amount of time, the screen should clear the displayed shapes. After the screen clears, random shapes should continue to display. Also, use the `Color.FromArgb` method so that you can specify random opacity (alpha values) for the colors. You should pass four arguments to this method. The first argument is the alpha value, the second is the red value, the third is the green value and the fourth is the blue value.

Answer:

```
1  // Exercise 26.17 Solution
2  // ScreenSaver.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace ScreenSaver
12 {
13    /// <summary>
14    /// Summary description for FrmShapeChanger.
15    /// </summary>
16    public class FrmShapeChanger : System.Windows.Forms.Form
17    {
18       // Timer to update screen saver at intervals
19       private System.Windows.Forms.Timer tmrScreenSaver;
20
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.IContainer components;
25
26       private double m_dblCount = 0.0;
27
28       public FrmShapeChanger()
29       {
30          //
31          // Required for Windows Form Designer support
32          //
33          InitializeComponent();
34
35          //
```

```
36          // TODO: Add any constructor code after InitializeComponent
37          // call
38          //
39       }
40
41       /// <summary>
42       /// Clean up any resources being used.
43       /// </summary>
44       protected override void Dispose( bool disposing )
45       {
46          if( disposing )
47          {
48             if (components != null)
49             {
50                components.Dispose();
51             }
52          }
53          base.Dispose( disposing );
54       }
55
56       // Windows Form Designer generated code
57
58       /// <summary>
59       /// The main entry point for the application.
60       /// </summary>
61       [STAThread]
62       static void Main()
63       {
64          Application.Run( new FrmShapeChanger() );
65       }
66
67       // create a specified graphic on the form
68       private void DisplayShape()
69       {
70          // create a Graphics object
71          Graphics objGraphicsObject = this.CreateGraphics();
72
73          // create random object for random number generation
74          Random objRandom = new Random();
75
76          // create random color with random opacity
77          Color objColor = Color.FromArgb(
78             objRandom.Next( 0, 255 ), objRandom.Next( 0, 255 ),
79             objRandom.Next( 0, 255 ), objRandom.Next( 0, 255 ) );
80
81          // create brush object used for solid shapes
82          SolidBrush objBrush = new SolidBrush( objColor );
83
84          // create pen object used for unfilled shapes
85          Pen objPen = new Pen( objBrush );
86
87          // create random number used to create random shape
88          int intShape = objRandom.Next( 0, 5 );
89
90          // set to width of form
91          int intWidth = this.Size.Width;
92
93          // set to height of form
94          int intHeight = this.Size.Height;
95
```

```
96          // decide which shape to draw
97          switch ( intShape )
98          {
99             case 0:
100
101               // create filled rectangle
102               objGraphicsObject.FillRectangle( objBrush,
103                  objRandom.Next( 0, intWidth ),
104                  objRandom.Next( 0, intHeight ),
105                  objRandom.Next( 10, 100 ),
106                  objRandom.Next( 10, 100 ) );
107               break;
108
109            case 1:
110
111               // create filled ellipse
112               objGraphicsObject.FillEllipse( objBrush,
113                  objRandom.Next( 0, intWidth ),
114                  objRandom.Next( 0, intHeight ),
115                  objRandom.Next( 10, 100 ),
116                  objRandom.Next( 10, 100 ) );
117               break;
118
119            case 2:
120
121               // draw ellipse
122               objGraphicsObject.DrawEllipse( objPen,
123                  objRandom.Next( 0, intWidth ),
124                  objRandom.Next( 0, intHeight ),
125                  objRandom.Next( 10, 100 ),
126                  objRandom.Next( 10, 100 ) );
127               break;
128
129            case 3:
130
131               // draw rectangle
132               objGraphicsObject.DrawRectangle( objPen,
133                  objRandom.Next( 0, intWidth ),
134                  objRandom.Next( 0, intHeight ),
135                  objRandom.Next( 10, 100 ),
136                  objRandom.Next( 10, 100 ) );
137               break;
138
139            case 4:
140
141               // draw line
142               objGraphicsObject.DrawLine( objPen,
143                  objRandom.Next( 0, intWidth ),
144                  objRandom.Next( 0, intHeight ),
145                  objRandom.Next( 10, intWidth ),
146                  objRandom.Next( 10, intHeight ) );
147               break;
148
149         } // end switch
150
151      } // end method DisplayShape
152
153      // invoked each time tmrScreenSaver ticks
154      private void tmrScreenSaver_Tick(
155         object sender, System.EventArgs e )
```

```
156       {
157          m_dblCount += 0.25;
158
159          // draw shape every half second
160          if ( m_dblCount % 2.5 == 0 )
161          {
162             DisplayShape(); // draws another shape
163          }
164
165          if ( m_dblCount % 200 == 0 )
166          {
167             this.Invalidate(); // clear screen
168          }
169
170       } // end method tmrScreenSaver_Tick
171
172    } // end class FrmShapeChanger
173 }
```

27

TUTORIAL

Phone Book Application

Introducing Multimedia Using Microsoft Agent

Solutions

Instructor's Manual Exercise Solutions Tutorial 27

MULTIPLE-CHOICE QUESTIONS

27.1 The _____ method is used to specify what the Microsoft Agent will say.

a) `Speak` b) `Say`
c) `Command` d) `Voice`

27.2 The _____ method is used to activate a Microsoft Agent character's animation.

a) `Show` b) `Play`
c) `Speak` d) `Appear`

27.3 The `MoveTo` method takes three arguments. What do the first two arguments represent?

a) The direction in which the Agent should move (left, right, up, down).
b) The name of the character and its position.
c) The *x*-coordinate and *y*-coordinate of the location to which the Agent should move.
d) The name of the character and the direction of movement.

27.4 Which method of `IAgentCtlCharacter` displays the Microsoft Agent character on the screen?

a) `Play` b) `Show`
c) `Speak` d) `Appear`

27.5 Use the _____ event handler to execute code when users click **Hide** in the Agent character context menu.

a) `Hide` b) `HideEvent`
c) `Command` d) `Disappear`

27.6 The `Add` method of the `Commands` property _____.

a) adds a new command to the command list
b) joins two commands together
c) displays the `Commands` pop-up window
d) Both a and c.

27.7 The _____ event handler controls what occurs when users speak to the Agent.

a) `Command` b) `ClickEvent`
c) `Click` d) `SelectedIndexChanged`

27.8 _____ specifies the *x*-coordinate of the mouse cursor on the screen.

a) `Cursor.Location.X` b) `Cursor.Position.X`
c) `Mouse.Location.X` d) `Mouse.Position.X`

27.9 Specifying _____ as a parameter to Peedy's `Play` method causes him to smile.

a) `"Think"` b) `"Smile"`
c) `"Pleased"` d) `"Happy"`

27.10 Specifying _____ as a parameter to Peedy's `Play` method causes him to rest.

a) `"RestPose"` b) `"Rest"`
c) `"Think"` d) `"Pose"`

Answers: 27.1) a. 27.2) b. 27.3) c. 27.4) b. 27.5) b. 27.6) a. 27.7) a. 27.8) b. 27.9) c. 27.10) a.

EXERCISES

27.11 (*Appointment Book Application Using Microsoft Agent*) Write an application that allows users to add appointments to an appointment book that uses Microsoft Agent (Fig. 27.33). When users speak a person's name, Merlin returns the time and date of the appointment that users have with that person. If users say "Today," Merlin returns a list of the users' appointments for the day.

Figure 27.33 Appointment Book GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial27\Exercises\AppointmentBook` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `AppointmentBook.sln` in the `AppointmentBook` directory to open the application.

c) ***Downloading the Merlin Microsoft Agent***. Download the `Merlin.acs` character file from the Microsoft Web site.

d) ***Adding the Agent Control to the `Form`.*** Add the Microsoft Agent control to the `Form`. Rearrange and comment the control declaration appropriately.

e) ***Creating instance variables***. Create three instance variables of the `ArrayList` type to store the date, time and person with which the user has an appointment. Create an instance variable of the `AgentObjects.IAgentCtlCharacter` type (as you did in the **Phone Book** application).

f) ***Defining the `FrmAppointments_Load` event handler***. Load Merlin's character file, display him on the screen and add the `Today` command to the command list.

g) ***Defining the `btnAdd_Click` event handler***. Define this event handler so that the information provided by the user is added to its corresponding `ArrayList`. The **Appointment With:** `TextBox` input should be added to the `ArrayList` containing the names of people with whom the user has an appointment. The input for the appointment date and time should also be added to their respective `ArrayLists`. Display an error message if the user leaves the **Appointment With:** or the **Appointment Time:** `TextBox` empty.

h) ***Adding voice-enabled commands***. Within the `btnAdd_Click` event handler, add a voice-enabled command to allow a user to speak the name of the person with whom the user has an appointment to the command list. This allows a user to check whether there is an appointment with someone by speaking the person's name. The command should also appear in the `Commands` context menu.

i) ***Defining the Agent's `Command` event handler***. As you did in the **Phone Book** application, define what occurs when a user speaks or selects a command. If the user specifies the `Today` command, Merlin should tell the user the names of all the people with whom the user has an appointment today. If the user specifies a specific name, Merlin should state the time and date at which the user has an appointment with this person. If the user did not schedule any appointments, Merlin should inform the user that no appointments were scheduled.

j) ***Running the application.*** Select **Debug > Start** to run your application. Add several appointments to ensure that your application runs correctly as in Fig. 27.33.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 27.11 Solution
// AppointmentBook.cs

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace AppointmentBook
{
   /// <summary>
   /// Summary description for FrmAppointments.
   /// </summary>
   public class FrmAppointments : System.Windows.Forms.Form
   {
      // Label and TextBox to input name of person appointment
      // is with
      private System.Windows.Forms.Label lblWith;
      private System.Windows.Forms.TextBox txtWith;

      // Label and DateTimePicker to choose appointment date
      private System.Windows.Forms.Label lblDate;
      private System.Windows.Forms.DateTimePicker dtpAddDate;

      // Label and TextBox to input appointment time
      private System.Windows.Forms.Label lblTime;
      private System.Windows.Forms.TextBox txtTime;

      // Button to add the appointment to the book
      private System.Windows.Forms.Button btnAdd;

      // MSAgent to use the Merlin agent
      private AxAgentObjects.AxAgent objMainAgent;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // create three ArrayLists
      private ArrayList m_objPerson = new ArrayList();
      private ArrayList m_objDate = new ArrayList();
      private ArrayList m_objTime = new ArrayList();

      // represent current agent
      private AgentObjects.IAgentCtlCharacter m_objMSpeaker;

      public FrmAppointments()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
```

```
60          //
61       }
62
63       /// <summary>
64       /// Clean up any resources being used.
65       /// </summary>
66       protected override void Dispose( bool disposing )
67       {
68          if( disposing )
69          {
70             if (components != null)
71             {
72                components.Dispose();
73             }
74          }
75          base.Dispose( disposing );
76       }
77
78       // Windows Form Designer generated code
79
80       /// <summary>
81       /// The main entry point for the application.
82       /// </summary>
83       [STAThread]
84       static void Main()
85       {
86          Application.Run( new FrmAppointments() );
87       }
88
89       // handles Form's Load event
90       private void FrmAppointments_Load(
91          object sender, System.EventArgs e )
92       {
93          // load agent character file
94          objMainAgent.Characters.Load( "Merlin", "Merlin.acs" );
95
96          // specify current agent
97          m_objMSpeaker = objMainAgent.Characters[ "Merlin" ];
98
99          // show Merlin on screen
100         m_objMSpeaker.Show( 0 );
101
102         // add voice enabled command to Agent object
103         m_objMSpeaker.Commands.Add( "Today", "Today",
104            "Today", true, true );
105
106      } // end method FrmAppointments_Load
107
108      // add appointments to ArrayLists
109      private void btnAdd_Click(
110         object sender, System.EventArgs e )
111      {
112         if ( txtWith.Text == "" || txtTime.Text == "" )
113         {
114            MessageBox.Show(
115               "You left one or more fields empty above",
116               "Empty Field", MessageBoxButtons.OK,
117               MessageBoxIcon.Error );
118         }
119         else
```

```
120            {
121               // add information to ArrayLists
122               m_objPerson.Add( txtWith.Text );
123               m_objDate.Add( dtpAddDate.Value.ToLongDateString() );
124               m_objTime.Add( txtTime.Text );
125
126               m_objMSpeaker.Commands.Add( txtWith.Text,
127                  txtWith.Text, txtWith.Text, true, true );
128
129               // clear TextBoxes
130               txtWith.Clear();
131               txtTime.Clear();
132            }
133
134         } // end method btnAdd_Click
135
136         // determine what Agent does when it hears command
137         private void objMainAgent_Command(
138            object sender, AxAgentObjects._AgentEvents_CommandEvent e )
139         {
140            // get UserInput object
141            AgentObjects.IAgentCtlUserInput objCommand =
142               ( AgentObjects.IAgentCtlUserInput ) e.userInput;
143
144            int intCount;
145
146            // boolean to determine if user has appointments
147            bool blnAppointments = false;
148            string strAppointments =
149               "Today you have an appointment with:";
150
151            // determine if user spoke Today command
152            if ( objCommand.Name == "Today" )
153            {
154               // search objDate ArrayList for appointments
155               // set for today
156               for ( intCount = 0; intCount < m_objDate.Count;
157                  intCount++ )
158               {
159                  if ( Convert.ToString( m_objDate[ intCount ] ) ==
160                     DateTime.Today.Date.ToLongDateString() )
161                  {
162                     // add the appointment to string Agent speaks
163                     strAppointments = strAppointments +
164                        Convert.ToString( m_objPerson[ intCount ] )
165                        + " at " +
166                        Convert.ToString( m_objTime[ intCount ] )
167                        + ", ";
168
169                     blnAppointments = true;
170                  }
171               }
172
173            } // end if
174
175            else
176            {
177               // determine if user made command using person's name
178               for ( intCount = 0; intCount < m_objPerson.Count;
179                  intCount++ )
```

```
180            {
181               if ( objCommand.Name ==
182                  Convert.ToString( m_objPerson[ intCount ] ) )
183               {
184                  // specify string Agent speaks
185                  strAppointments = "You have to meet with" +
186                     Convert.ToString( m_objPerson[ intCount ] )
187                     + " on: " +
188                     Convert.ToString( m_objDate[ intCount ] )
189                     + " at " +
190                     Convert.ToString( m_objTime[ intCount ] )
191                     + ", ";
192
193                  blnAppointments = true;
194               }
195            }
196
197         } // end else
198
199         // if user has no appointments, specify in strAppointments
200         if ( blnAppointments == false )
201         {
202            strAppointments = "You have no scheduled appointments.";
203         }
204
205         // Agent speaks strAppointments string
206         m_objMSpeaker.Speak( strAppointments, "" );
207
208      } // end method objMainAgent_Command
209
210   } // end class FrmAppointments
211 }
```

27.12 ***(Craps Game Application Enhancement Using Microsoft Agent)*** Modify the **Craps Game** application from Tutorial 16 to include a Microsoft Agent character (Fig. 27.34).

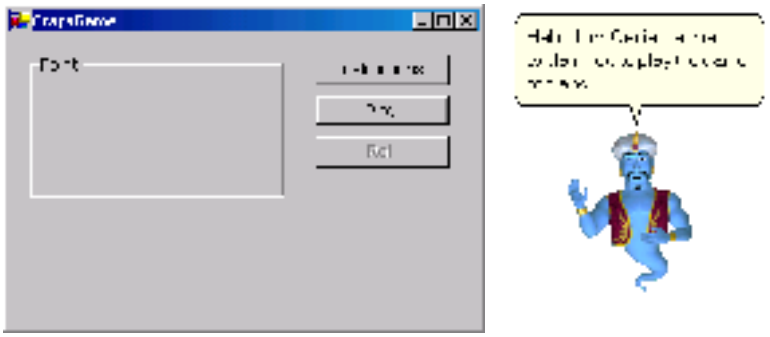


Figure 27.34 Enhanced **Craps Game** GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial27\Exercises\CrapsGameEnhancement` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `CrapsGame.sln` in the `CrapsGameEnhancement` directory to open the application.

c) ***Downloading the Genie Microsoft Agent.*** Download the `Genie.acs` character file from the Microsoft Web site.

d) ***Adding the Agent control to the Form.*** Add the Microsoft Agent control to the `Form`. Rearrange and comment the control declaration appropriately.

e) ***Creating an instance variable.*** Create an instance variable of the `AgentObjects.IAgentCtlCharacter` type (as you did in the **Phone Book** application).

f) ***Defining the `FrmCrapsGame_Load` event handler.*** Load Genie's character file, and display him on the screen.

g) ***Modifying the `btnPlay_Click` event handler.*** Add code to the `btnPlay_Click` event handler to control the Agent. When the user wins the game, Genie should play his `Pleased` animation and congratulate the user. If the user loses, Genie should play his `Confused` animation and say that the user lost. If the user neither wins nor loses, Genie should tell the user to roll again. Make sure to reset him to his `RestPose` after he plays any animation.

h) ***Defining the `btnRoll_Click` event handler.*** Add code to the `btnRoll_Click` event handler to control the Agent. If users "make their point," Genie should play his `Pleased` animation and state that the user won. If the user rolls a 7, Genie should play his `Confused` animation and say that the user lost. Otherwise, Genie should tell the user to roll again.

i) ***Defining the `btnInstructions_Click` event handler.*** Define the event handler `btnInstructions_Click` to make Genie introduce himself to the user. Genie should then explain the rules to the game of craps.

j) ***Running the application.*** Select **Debug > Start** to run your application. Play several games to ensure that your application runs correctly. Also be sure to test the **Instructions** `Button`.

k) ***Closing the application.*** Close your running application by clicking its close box.

l) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 27.12 Solution
// CrapsGame.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;

namespace CrapsGame
{
   /// <summary>
   /// Summary description for FrmCrapsGame.
   /// </summary>
   public class FrmCrapsGame : System.Windows.Forms.Form
   {
      // GroupBox to display point dice
      private System.Windows.Forms.GroupBox fraPointDiceGroup;

      // PictureBoxes of point dice
      private System.Windows.Forms.PictureBox picPointDie1;
      private System.Windows.Forms.PictureBox picPointDie2;

      // PictureBoxes of current dice
      private System.Windows.Forms.PictureBox picDie1;
      private System.Windows.Forms.PictureBox picDie2;

      // Button to make Genie say the instructions
      private System.Windows.Forms.Button btnInstructions;

      // Button to play a game
```

```
34          private System.Windows.Forms.Button btnPlay;
35
36          // Button to roll the dice
37          private System.Windows.Forms.Button btnRoll;
38
39          // Label to display status of the game
40          private System.Windows.Forms.Label lblStatus;
41
42          // MSAgent to use Genie agent
43          private AxAgentObjects.AxAgent objMainAgent;
44
45          /// <summary>
46          /// Required designer variable.
47          /// </summary>
48          private System.ComponentModel.Container components = null;
49
50          // die-roll constants
51          enum DiceNames
52          {
53             SNAKE_EYES = 2,
54             TREY = 3,
55             CRAPS = 7,
56             LUCKY_SEVEN = 7,
57             YO_LEVEN = 11,
58             BOX_CARS = 12,
59          }
60
61          // file name and directory constants
62          const string m_strFILE_PREFIX = "/images/die";
63          const string m_strFILE_SUFFIX = ".png";
64
65          // instance variables
66          private int m_intMyPoint = 0;
67          private int m_intMyDie1;
68          private int m_intMyDie2;
69          private Random m_objRandom = new Random();
70
71          // create object that represents current agent
72          private AgentObjects.IAgentCtlCharacter m_objMSpeaker;
73
74          public FrmCrapsGame()
75          {
76             //
77             // Required for Windows Form Designer support
78             //
79             InitializeComponent();
80
81             //
82             // TODO: Add any constructor code after InitializeComponent
83             // call
84             //
85          }
86
87          /// <summary>
88          /// Clean up any resources being used.
89          /// </summary>
90          protected override void Dispose( bool disposing )
91          {
92             if( disposing )
93             {
```

```
94             if (components != null)
95             {
96                components.Dispose();
97             }
98          }
99          base.Dispose( disposing );
100       }
101
102       // Windows Form Designer generated code
103
104       /// <summary>
105       /// The main entry point for the application.
106       /// </summary>
107       [STAThread]
108       static void Main()
109       {
110          Application.Run( new FrmCrapsGame() );
111       }
112
113       // handles Form's Load event
114       private void FrmCrapsGame_Load(
115          object sender, System.EventArgs e )
116       {
117          // load Genie character file
118          objMainAgent.Characters.Load( "Genie", "Genie.acs" );
119
120          m_objMSpeaker = objMainAgent.Characters[ "Genie" ];
121
122          // show Genie on the screen
123          m_objMSpeaker.Show( 0 );
124
125       } // end method FrmCrapsGame_Load
126
127       // begin new game and determine point
128       private void btnPlay_Click(
129          object sender, System.EventArgs e )
130       {
131          // initialize variables for new game
132          m_intMyPoint = 0;
133          fraPointDiceGroup.Text = "Point";
134          lblStatus.Text = "";
135
136          // remove point-die images
137          picPointDie1.Image = null;
138          picPointDie2.Image = null;
139
140          int intSum = RollDice(); // roll dice
141
142          // check die roll
143          switch ( intSum )
144          {
145             // win on first roll
146             case ( int ) DiceNames.LUCKY_SEVEN:
147             case ( int ) DiceNames.YO_LEVEN:
148
149                btnRoll.Enabled = false; // disable Roll Button
150                lblStatus.Text = "You win!!!";
151
152                // play Agent's Pleased animation
153                m_objMSpeaker.Play( "Pleased" );
```

```
154
155               // make Agent speak
156               m_objMSpeaker.Speak(
157                  "Congratulations, you won!", "" );
158
159               m_objMSpeaker.Play( "RestPose" );
160
161               break;
162
163            // lose on first roll
164            case ( int ) DiceNames.SNAKE_EYES:
165            case ( int ) DiceNames.TREY:
166            case ( int ) DiceNames.BOX_CARS:
167
168               btnRoll.Enabled = false;
169               lblStatus.Text = "Sorry, you lose.";
170
171               // play Agent's Confused animation
172               m_objMSpeaker.Play( "Confused" );
173
174               // make Agent speak
175               m_objMSpeaker.Speak( "Sorry, you lost!", "" );
176
177               m_objMSpeaker.Play( "RestPose" );
178
179               break;
180
181            // player must match point
182            default:
183
184               m_intMyPoint = intSum;
185               fraPointDiceGroup.Text = "Point is " + intSum;
186               lblStatus.Text = "Roll again!";
187               m_objMSpeaker.Speak( "Please roll again.", "" );
188               DisplayDie( picPointDie1, m_intMyDie1 );
189               DisplayDie( picPointDie2, m_intMyDie2 );
190               btnPlay.Enabled = false; // disable Play Button
191               btnRoll.Enabled = true;  // enable Roll Button
192               break;
193
194         } // end switch
195
196      } // end method btnPlay_Click
197
198      // determine outcome of next roll
199      private void btnRoll_Click(
200         object sender, System.EventArgs e )
201      {
202         int intSum = RollDice();
203
204         // determine outcome of roll
205         if ( intSum == m_intMyPoint ) // player matches point
206         {
207            lblStatus.Text = "You win!!!";
208            m_objMSpeaker.Play( "Pleased" );
209            m_objMSpeaker.Speak( "Congratulations, you won!", "" );
210            m_objMSpeaker.Play( "RestPose" );
211
212            btnRoll.Enabled = false;
213            btnPlay.Enabled = true;
```

```
214          }
215          else if ( intSum == ( int ) DiceNames.CRAPS )
216          {
217             // player loses
218             lblStatus.Text = "Sorry, you lose.";
219             m_objMSpeaker.Play( "Confused" );
220             m_objMSpeaker.Speak( "Sorry, you lost!", "" );
221             m_objMSpeaker.Play( "RestPose" );
222
223             btnRoll.Enabled = false;
224             btnPlay.Enabled = true;
225          }
226
227       } // end method btnRoll_Click
228
229       // display die image
230       private void DisplayDie( PictureBox picDie, int intFace )
231       {
232          // assign die images to PictureBox
233          picDie.Image =
234             Image.FromFile( Directory.GetCurrentDirectory() +
235             m_strFILE_PREFIX + intFace + m_strFILE_SUFFIX );
236
237       } // end method DisplayDie
238
239       // generate random die rolls
240       private int RollDice()
241       {
242          // roll the dice
243          int intDie1 = m_objRandom.Next( 1, 7 );
244          int intDie2 = m_objRandom.Next( 1, 7 );
245
246          // display image corresponding to each die
247          DisplayDie( picDie1, intDie1 );
248          DisplayDie( picDie2, intDie2 );
249
250          // set values
251          m_intMyDie1 = intDie1;
252          m_intMyDie2 = intDie2;
253
254          return ( intDie1 + intDie2 ); // return sum of dice values
255
256       } // end method RollDice
257
258       // Agent explains instructions of the game
259       private void btnInstructions_Click(
260          object sender, System.EventArgs e )
261       {
262          m_objMSpeaker.Play( "Wave" );
263
264          m_objMSpeaker.Speak( "Hello. I am Genie. Let me " +
265             "explain how to play the game of craps.", "" );
266
267          m_objMSpeaker.Speak( "Click the Play button to begin " +
268             "the game.", "" );
269
270          m_objMSpeaker.Speak( "Clicking Play causes you to roll"
271             + " two dice.", "" );
272
273          m_objMSpeaker.Speak( "If the sum of your dice roll is "
```

```
274            + "7 or 11 on your first throw, then you win.", "" );
275
276         m_objMSpeaker.Speak( "However, if the sum is 2, 3 or "
277            + "12 on your first throw, then you lose.", "" );
278
279         m_objMSpeaker.Speak( "If the sum is 4, 5, 6, 7, 8, 9 "
280            + "or 10 on your first throw, then that sum "
281            + "becomes your 'point'", "" );
282
283         m_objMSpeaker.Speak( "To win, you must continue rolling"
284            + " the dice until you roll the point value again.","" );
285
286         m_objMSpeaker.Speak( "You lose if you roll a 7 before "
287            + "reaching your point value.", "" );
288
289         m_objMSpeaker.Play( "RestPose" );
290
291      } // end method btnInstructions_Click
292
293   } // end class FrmCrapsGame
294 }
```

27.13 (*Security Panel Application Enhancement Using Microsoft Agent*) Modify the **Security Panel** application from Tutorial 12 to include Microsoft Agent (Fig. 27.35).

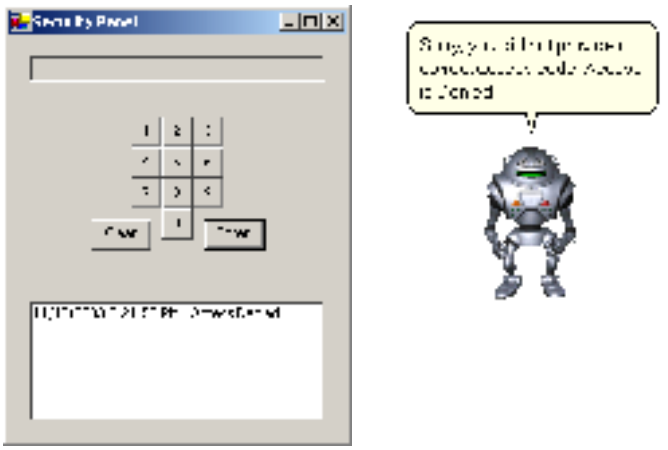


Figure 27.35 Modified **Security Panel** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial27\Exercises\SecurityPanelEnhancement` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `SecurityPanel.sln` in the `SecurityPanelEnhancement` directory to open the application.

c) ***Downloading the Robby Microsoft Agent.*** Download the `Robby.acs` character file from the Microsoft Web site.

d) ***Adding the Agent control to the Form.*** Add the Microsoft Agent control to the `Form`.

e) ***Creating an instance variable.*** Create an instance variable of the `AgentObjects.IAgentCtlCharacter` type (as you did in the **Phone Book** application). Rearrange and comment the control declarations appropriately.

f) ***Defining the `FrmSecurityPanel_Load` event handler.*** Load Robby's character file, and display him on the screen. Command Robby to tell users to input their access codes.

g) ***Modifying the btnEnter_Click event handler.*** Add code to the `btnEnter_Click` event handler to use the Microsoft Agent. If the user enters a valid access code, Robby should welcome the user and state the type of employee that the access code represents. If the access code is invalid, then Robby should state that an invalid code was provided and that access is denied.

h) ***Running the application.*** Select **Debug > Start** to run your application. Enter various correct and incorrect codes to ensure that your application runs correctly.

i) ***Closing the application.*** Close your running application by clicking its close box.

j) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 27.13 Solution
2  // SecurityPanel.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace SecurityPanel
12 {
13    /// <summary>
14    /// Summary description for FrmSecurityPanel.
15    /// </summary>
16    public class FrmSecurityPanel : System.Windows.Forms.Form
17    {
18       // TextBox to display security code
19       private System.Windows.Forms.TextBox txtSecurityCode;
20
21       // Buttons to input security code
22       private System.Windows.Forms.Button btnOne;
23       private System.Windows.Forms.Button btnTwo;
24       private System.Windows.Forms.Button btnThree;
25       private System.Windows.Forms.Button btnFour;
26       private System.Windows.Forms.Button btnFive;
27       private System.Windows.Forms.Button btnSix;
28       private System.Windows.Forms.Button btnSeven;
29       private System.Windows.Forms.Button btnEight;
30       private System.Windows.Forms.Button btnNine;
31       private System.Windows.Forms.Button btnZero;
32       private System.Windows.Forms.Button btnClear;
33       private System.Windows.Forms.Button btnEnter;
34
35       // ListBox to display access log
36       private System.Windows.Forms.ListBox lstLogEntry;
37
38       // MSAgent to use the Robby agent
39       private AxAgentObjects.AxAgent objMainAgent;
40
41       /// <summary>
42       /// Required designer variable.
43       /// </summary>
44       private System.ComponentModel.Container components = null;
45
46       private AgentObjects.IAgentCtlCharacter m_objMSpeaker;
47
48       public FrmSecurityPanel()
```

```
49          {
50             //
51             // Required for Windows Form Designer support
52             //
53             InitializeComponent();
54
55             //
56             // TODO: Add any constructor code after InitializeComponent
57             // call
58             //
59          }
60
61          /// <summary>
62          /// Clean up any resources being used.
63          /// </summary>
64          protected override void Dispose( bool disposing )
65          {
66             if( disposing )
67             {
68                if (components != null)
69                {
70                   components.Dispose();
71                }
72             }
73             base.Dispose( disposing );
74          }
75
76          // Windows Form Designer generated code
77
78          /// <summary>
79          /// The main entry point for the application.
80          /// </summary>
81          [STAThread]
82          static void Main()
83          {
84             Application.Run( new FrmSecurityPanel() );
85          }
86
87          // handles Enter button's Click event
88          private void btnEnter_Click(
89             object sender, System.EventArgs e )
90          {
91             int intAccessCode;  // stores access code entered
92             string strMessage;  // displays access status of users
93
94             intAccessCode = Int32.Parse( txtSecurityCode.Text );
95             txtSecurityCode.Clear();
96
97             switch ( intAccessCode )  // check access code input
98             {
99                // access code less than 10
100               case 0:
101               case 1:
102               case 2:
103               case 3:
104               case 4:
105               case 5:
106               case 6:
107               case 7:
108               case 8:
```

```
109             case 9:
110                strMessage = "Restricted Access";
111                m_objMSpeaker.Speak( "Welcome. You have "
112                   + "restricted access.", "" );
113                break;
114
115             // access code equal to 1645 or 1689
116             case 1645:
117             case 1689:
118
119                strMessage = "Technicians";
120                m_objMSpeaker.Speak( "Welcome. Your access"
121                   + " code indicates that you are "
122                   + "part of the Technician Personnel.", "" );
123                break;
124
125             // access code equal to 8345
126             case 8345:
127
128                strMessage = "Custodians";
129                m_objMSpeaker.Speak( "Welcome. Your access"
130                   + " code indicates that you are "
131                   + "part of Custodial Services.", "" );
132                break;
133
134             // access code equal to 9998 or between
135             // 1006 and 1008, inclusive
136             case 9998:
137             case 1006:
138             case 1007:
139             case 1008:
140
141                strMessage = "Scientists";
142                m_objMSpeaker.Speak( "Welcome. Your access"
143                   + " code indicates that you are "
144                   + "part of the Scientific Personnel.", "" );
145                break;
146
147             // if no other case is true
148             default:
149
150                strMessage = "Access Denied";
151                m_objMSpeaker.Speak( "Sorry, you did not "
152                   + "provide a correct access code."
153                   + " Access is Denied.", "" );
154                break;
155
156          } // end switch
157
158          // display time and message in ListBox
159          lstLogEntry.Items.Add( DateTime.Now + "    " + strMessage );
160
161       } // end method btnEnter_Click
162
163       private void btnZero_Click(
164          object sender, System.EventArgs e )
165       {
166          txtSecurityCode.Text += "0"; // concatenate "0" to display
167
168       } // end method btnZero_Click
```

```
169
170         private void btnOne_Click(
171            object sender, System.EventArgs e )
172         {
173            txtSecurityCode.Text += "1";  // concatenate "1" to display
174
175         } // end method btnOne_Click
176
177         private void btnTwo_Click(
178            object sender, System.EventArgs e )
179         {
180            txtSecurityCode.Text += "2";  // concatenate "2" to display
181
182         } // end method btnTwo_Click
183
184         private void btnThree_Click(
185            object sender, System.EventArgs e )
186         {
187            txtSecurityCode.Text += "3";  // concatenate "3" to display
188
189         } // end method btnThree_Click
190
191         private void btnFour_Click(
192            object sender, System.EventArgs e )
193         {
194            txtSecurityCode.Text += "4";  // concatenate "4" to display
195
196         } // end method btnFour_Click
197
198         private void btnFive_Click(
199            object sender, System.EventArgs e )
200         {
201            txtSecurityCode.Text += "5";  // concatenate "5" to display
202
203         } // end method btnFive_Click
204
205         private void btnSix_Click(
206            object sender, System.EventArgs e )
207         {
208            txtSecurityCode.Text += "6";  // concatenate "6" to display
209
210         } // end method btnSix_Click
211
212         private void btnSeven_Click(
213            object sender, System.EventArgs e )
214         {
215            txtSecurityCode.Text += "7";  // concatenate "7" to display
216
217         } // end method btnSeven_Click
218
219         private void btnEight_Click(
220            object sender, System.EventArgs e )
221         {
222            txtSecurityCode.Text += "8";  // concatenate "8" to display
223
224         } // end method btnEight_Click
225
226         private void btnNine_Click(
227            object sender, System.EventArgs e )
228         {
```

```
229          txtSecurityCode.Text += "9";  // concatenate "9" to display
230
231       } // end method btnNine_Click
232
233       private void btnClear_Click(
234          object sender, System.EventArgs e )
235       {
236          txtSecurityCode.Clear();  // clear text from TextBox
237
238       } // end method btnClear_Click
239
240       // handles Form's Load event
241       private void FrmSecurityPanel_Load(
242          object sender, System.EventArgs e )
243       {
244          // load Agent character file
245          objMainAgent.Characters.Load( "Robby", "Robby.acs" );
246
247          // specify current agent
248          m_objMSpeaker = objMainAgent.Characters[ "Robby" ];
249
250          // show Robby on screen
251          m_objMSpeaker.Show( 0 );
252
253          // Robby speaks
254          m_objMSpeaker.Speak( "Please enter your security "
255             + "access code.", "" );
256
257       } // end method FrmSecurityPanel_Load
258
259    } // end class FrmSecurityPanel
260 }
```

What does this code do? ▶

27.14 After the user clicks the **Call** `Button`, what does the following event handler do?

```
1  private void btnCall_Click(
2     object sender, System.EventArgs e )
3  {
4     objMainAgent.Characters.Load( "Genie","Genie.acs" );
5
6     objMSpeaker = objMainAgent.Characters[ "Genie" ];
7
8     objMSpeaker.Show( 0 );
9
10    objMSpeaker.Speak( "Hello, I'm Genie the special agent!", "" );
11
12 } // end method btnCall_Click
```

Answer: The agent object is loaded as "Genie." Genie appears and says, "`Hello, I'm Genie the special agent!`" The complete code reads:

```
1  // Exercise 27.14 Solution
2  // Genie.cs
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
```

```
using System.Windows.Forms;
using System.Data;
using System.IO;

namespace Genie
{
   /// <summary>
   /// Summary description for FrmGenie.
   /// </summary>
   public class FrmGenie : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblCall;
      private System.Windows.Forms.Button btnCall;
      private AxAgentObjects.AxAgent objMainAgent;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmGenie()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      private AgentObjects.IAgentCtlCharacter objMSpeaker;

      // Windows Form Designer generated code

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.Run( new FrmGenie() );
      }

```

```
68          // calls "Genie" the agent
69          private void btnCall_Click(
70             object sender, System.EventArgs e )
71          {
72             objMainAgent.Characters.Load( "Genie","Genie.acs" );
73
74             objMSpeaker = objMainAgent.Characters[ "Genie" ];
75
76             objMSpeaker.Show( 0 );
77
78             objMSpeaker.Speak( "Hello, I'm Genie the special agent!",
79                "" );
80
81          } // end method btnCall_Click
82
83       } // end class FrmGenie
84    }
```

What's wrong with this code?

27.15 Find the error(s) in the following code. The event handler should have an agent object appear and say, "Hello, my name is Merlin." This should happen when the user clicks the **Call** `Button`.

```
1   private void btnCall_Click( object sender, System.EventArgs e )
2   {
3      AgentObjects.IAgentCtlCharacter objMSpeaker;
4
5      objMainAgent.Characters.Load( "Merlin", "Merlin.acs" );
6
7      objMSpeaker = objMainAgent.Characters[ "Merlin.acs" ];
8
9      objMSpeaker.Show( 0 );
10
11     objMSpeaker.Play( "Hello, my name is Merlin", "" );
12
13  } // end method btnCall_Click
```

Answer: To have the agent say "Hello, my name is Merlin", you must call the `Speak` method rather than the `Play` method. Also, the name of the character should be passed on line 7, not the name of the file that defines the character. The complete incorrect code reads:

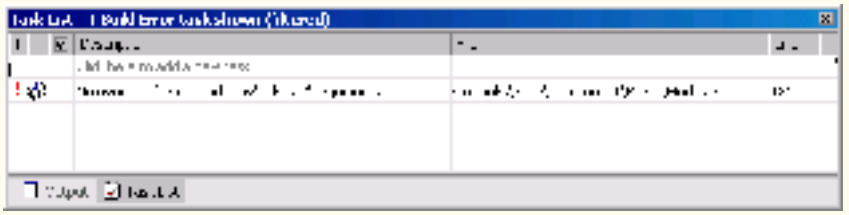
```
1   // Exercise 27.15 Solution
2   // Merlin.cs (Incorrect)
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  namespace Merlin
12  {
```

```
13      /// <summary>
14      /// Summary description for FrmMerlin.
15      /// </summary>
16      public class FrmMerlin : System.Windows.Forms.Form
17      {
18         private System.Windows.Forms.Label lblCall;
19         private System.Windows.Forms.Button btnCall;
20         private AxAgentObjects.AxAgent objMainAgent;
21         /// <summary>
22         /// Required designer variable.
23         /// </summary>
24         private System.ComponentModel.Container components = null;
25
26         public FrmMerlin()
27         {
28            //
29            // Required for Windows Form Designer support
30            //
31            InitializeComponent();
32
33            //
34            // TODO: Add any constructor code after InitializeComponent
35            // call
36            //
37         }
38
39         /// <summary>
40         /// Clean up any resources being used.
41         /// </summary>
42         protected override void Dispose( bool disposing )
43         {
44            if( disposing )
45            {
46               if (components != null)
47               {
48                  components.Dispose();
49               }
50            }
51            base.Dispose( disposing );
52         }
53
54         // Windows Form Designer generated code
55
56         /// <summary>
57         /// The main entry point for the application.
58         /// </summary>
59         [STAThread]
60         static void Main()
61         {
62            Application.Run(new FrmMerlin());
63         }
64
65         // calls "Merlin" the agent
66         private void btnCall_Click( object sender, System.EventArgs e )
67         {
68            AgentObjects.IAgentCtlCharacter objMSpeaker;
69
70            objMainAgent.Characters.Load( "Merlin", "Merlin.acs" );
71
72            objMSpeaker = objMainAgent.Characters[ "Merlin.acs" ];
```

Invalid character name

To have the Agent talk or print text, use the **Speak** method

```
73
74          objMSpeaker.Show( 0 );
75
76          objMSpeaker.Play( "Hello, my name is Merlin", "" );
77
78       } // end method btnCall_Click
79
80    } // end class FrmMerlin
81 }
```



Answer: The complete corrected code should read:

```
1  // Exercise 27.15 Solution
2  // Merlin.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Merlin
12 {
13    /// <summary>
14    /// Summary description for FrmMerlin.
15    /// </summary>
16    public class FrmMerlin : System.Windows.Forms.Form
17    {
18       private System.Windows.Forms.Label lblCall;
19       private System.Windows.Forms.Button btnCall;
20       private AxAgentObjects.AxAgent objMainAgent;
21       /// <summary>
22       /// Required designer variable.
23       /// </summary>
24       private System.ComponentModel.Container components = null;
25
26       public FrmMerlin()
27       {
28          //
29          // Required for Windows Form Designer support
30          //
31          InitializeComponent();
32
33          //
34          // TODO: Add any constructor code after InitializeComponent
35          // call
36          //
37       }
38
```

```
39          /// <summary>
40          /// Clean up any resources being used.
41          /// </summary>
42          protected override void Dispose( bool disposing )
43          {
44             if( disposing )
45             {
46                if (components != null)
47                {
48                   components.Dispose();
49                }
50             }
51             base.Dispose( disposing );
52          }
53
54          // Windows Form Designer generated code
55
56          /// <summary>
57          /// The main entry point for the application.
58          /// </summary>
59          [STAThread]
60          static void Main()
61          {
62             Application.Run( new FrmMerlin() );
63          }
64
65          // calls "Merlin" the agent
66          private void btnCall_Click( object sender, System.EventArgs e )
67          {
68             AgentObjects.IAgentCtlCharacter objMSpeaker;
69
70             objMainAgent.Characters.Load( "Merlin", "Merlin.acs" );
71
72             objMSpeaker = objMainAgent.Characters[ "Merlin" ];
73
74             objMSpeaker.Show( 0 );
75
76             objMSpeaker.Speak( "Hello, my name is Merlin", "" );
77
78          } // end method btnCall_Click
79
80       } // end class FrmMerlin
81    }
82
```

Programming Challenge

27.16 (*Car Payment Calculator Application Enhancement Using Microsoft Agent*) Enhance the **Car Payment Calculator** application from Tutorial 9 to use the Microsoft Agent Robby. When the application is run, Robby should appear on the screen and wave to users. He should then explain what the application does. After users have entered information into each field of the **Car Payment Calculator** and clicked the **Calculate** `Button`, Robby should speak the calculated payment amounts and the number of months for which they were calculated. Copy the directory `C:\Examples\Tutorial27\Exercises\CarPayment-`

CalculatorEnhancement to your C:\SimplyCSP directory. Double click the CarPayment-Calculator.sln file to open the application in Visual Studio .NET.

Answer:

```
// Exercise 27.16 Solution
// CarPayment.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace CarPaymentCalculator
{
   /// <summary>
   /// Summary description for FrmCarPayment.
   /// </summary>
   public class FrmCarPayment : System.Windows.Forms.Form
   {
      // Label and TextBox for sticker price
      private System.Windows.Forms.Label lblStickerPrice;
      private System.Windows.Forms.TextBox txtStickerPrice;

      // Label and TextBox for down payment
      private System.Windows.Forms.Label lblDownPayment;
      private System.Windows.Forms.TextBox txtDownPayment;

      // Label and textbox for interest rate
      private System.Windows.Forms.TextBox txtInterest;
      private System.Windows.Forms.Label lblInterest;

      // Button to calculate total cost
      private System.Windows.Forms.Button btnCalculate;

      // ListBox to display total cost
      private System.Windows.Forms.ListBox lstPayments;

      // MSAgent to use the Robby agent
      private AxAgentObjects.AxAgent objMainAgent;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      private AgentObjects.IAgentCtlCharacter m_objMSpeaker;

      public FrmCarPayment()
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
         // call
         //
      }
```

```
58
59          /// <summary>
60          /// Clean up any resources being used.
61          /// </summary>
62          protected override void Dispose( bool disposing )
63          {
64             if( disposing )
65             {
66                if (components != null)
67                {
68                   components.Dispose();
69                }
70             }
71             base.Dispose( disposing );
72          }
73
74          // Windows Form Designer generated code
75
76          /// <summary>
77          /// The main entry point for the application.
78          /// </summary>
79          [STAThread]
80          static void Main()
81          {
82             Application.Run( new FrmCarPayment() );
83          }
84
85          // handles Calculate Button's Click event
86          private void btnCalculate_Click(
87             object sender, System.EventArgs e )
88          {
89             int intYears = 2;                // repetition counter
90             int intMonths = 0;               // payment period
91             int intPrice = 0;                // car price
92             int intDownPayment = 0;          // down payment
93             double dblInterest = 0;          // interest rate
94             decimal decMonthlyPayment = 0;   // monthly payment
95             int intLoanAmount = 0;           // cost after down payment
96             double dblMonthlyInterest = 0;   // monthly interest rate
97             string strAgentSpeak = "You will have to pay: ";
98
99             // remove text displayed in ListBox
100            lstPayments.Items.Clear();
101
102            // add header to ListBox
103            lstPayments.Items.Add( "Months\t\tMonthly Payments" );
104
105            // retrieve user input and assign values
106            // to their respective variables
107            intDownPayment = Int32.Parse( txtDownPayment.Text );
108            intPrice = Int32.Parse( txtStickerPrice.Text );
109            dblInterest = Double.Parse( txtInterest.Text ) / 100;
110
111            // determine amount borrowed and monthly interest rate
112            intLoanAmount = intPrice - intDownPayment;
113            dblMonthlyInterest = dblInterest / 12;
114
115            // loop four times
116            while ( intYears <= 5 )
117            {
```

```
118            // calculate payment period
119            intMonths = 12 * intYears;
120
121            // calculate monthly payment using FCL Math Method
122            // for raising to a power
123            decMonthlyPayment = ( decimal )
124               ( intLoanAmount * dblMonthlyInterest *
125               Math.Pow( 1 + dblMonthlyInterest, intMonths ) /
126               ( Math.Pow( 1 + dblMonthlyInterest, intMonths )
127               - 1 ) );
128
129            // display payment value
130            lstPayments.Items.Add( intMonths + "\t\t" +
131               String.Format( "{0:C}", decMonthlyPayment ) );
132
133            if ( intYears < 5 )
134            {
135               // specify format of payment values for first
136               // four years
137               strAgentSpeak += String.Format( "{0:C}",
138                  decMonthlyPayment ) + " per month, over "
139                  + intMonths + " months, ";
140            }
141            else
142            {
143               // specify format for payment value of fifth year
144               strAgentSpeak += "or " + String.Format( "{0:C}",
145                  decMonthlyPayment ) + " per month, over "
146                  + intMonths + " months.";
147            }
148
149            intYears++;  // increment counter
150
151         } // end while
152
153         // Robby speaks strAgentSpeak string
154         m_objMSpeaker.Speak( strAgentSpeak , "" );
155
156      } // end method btnCalculate_Click
157
158      // handles Form's Load event
159      private void FrmCarPayment_Load(
160         object sender, System.EventArgs e )
161      {
162         // load Robby character into agent object
163         objMainAgent.Characters.Load( "Robby", "Robby.acs" );
164
165         m_objMSpeaker = objMainAgent.Characters[ "Robby" ];
166
167         // show Robby on the screen
168         m_objMSpeaker.Show( 0 );
169
170         // play Wave animation
171         m_objMSpeaker.Play( "Wave" );
172
173         // make Robby speak instructions for application
174         m_objMSpeaker.Speak(
175            "Hello, I will be your assistant.", "" );
176
177         m_objMSpeaker.Play( "RestPose" );
```

```
178
179          m_objMSpeaker.Speak(
180             "You need to enter the price of a car,"
181             + "  the down-payment amount and "
182             + " the annual interest rate of the loan.", "" );
183
184          m_objMSpeaker.Speak(
185             "After you input this information, "
186             + "I will calculate the monthly payments you will "
187             + "need to make for two-, three-, four- and "
188             + "five-year loans.", "" );
189
190       } // end method FrmCarPayment_Load
191
192    } // end class FrmCarPayment
193 }
```

28 TUTORIAL

Bookstore Application: Web Applications

Introducing Internet Information Services

Solutions

Instructor's Manual Exercise Solutions Tutorial 28

MULTIPLE-CHOICE QUESTIONS

28.1 ASPX pages have the _____ extension.

a) `.html`
b) `.wbform`
c) `.csaspx`
d) `.aspx`

28.2 _____ applications divide functionality into separate tiers.

a) *n*-tier
b) Multi-tier
c) Both a and b.
d) None of the above.

28.3 All tiers of a multi-tier application _____.

a) must be located on the same computer
b) must be located on different computers
c) can be located on the same computer or on different computers
d) must be arranged so that the client and middle tier are on the same computer and the information tier is on a different computer

28.4 The client tier interacts with the _____ tier to access information from the _____ tier.

a) middle; information
b) information; middle
c) information; bottom
d) bottom; information

28.5 A _____ is specialized software that responds to client requests by providing resources.

a) host
b) host name
c) DNS server
d) Web server

28.6 A(n) _____ can be thought of as an address that is used to direct a browser to a resource on the Web.

a) middle tier
b) ASPX page
c) URL
d) query string

28.7 A _____ represents a group of _____ on the Internet.

a) domain; hosts
b) host; domain names
c) host name; hosts
d) None of the above.

28.8 _____ is a Web server.

a) IIS
b) `localhost`
c) Visual Studio .NET
d) `wwwroot`

28.9 A _____ is a Web server that is located on a computer across a network, such as the Internet.

a) `localhost`
b) local Web server
c) remote Web server
d) None of the above.

28.10 The _____ tier is the application's user interface.

a) middle
b) client
c) bottom
d) information

Answers: 28.1) d. 28.2) c. 28.3) c. 28.4) a. 28.5) d. 28.6) c. 28.7) a. 28.8) a. 28.9) c. 28.10) b.

EXERCISES

28.11 (***Phone Book Application***) Over the next three tutorials, you will create a **Phone Book** application. This phone book should be a Web-based version of the **Phone Book**

application created in Tutorial 27. [*Note*: This Web application will not use Microsoft Agent.] The **Phone Book** application should consist of two ASPX pages, named `PhoneBook` and `PhoneNumber`. The `PhoneBook` page displays a `DropDownList` (a Web control similar to a `ComboBox` Windows `Form` control) that contains the names of several people. The names are retrieved from the `db_Phone.mdb` database. When a name is selected and the **Get Number** `Button` is clicked, the client browser is redirected to the `PhoneNumber` page. The telephone number of the selected name should be retrieved from a database and displayed in a `Label` on the `PhoneNumber` page. For this exercise, you need only organize the components (the `PhoneBook` and `PhoneNumber` ASPX pages, the `db_Phone.mdb` database and the code that performs the specified functionality) of this Web application into separate tiers. Decide which components belong in which tiers. You will begin building the solution, using Visual Studio .NET, in the next tutorial.

Answer: The client tier should contain the ASPX pages' GUIs. One page will contain a `DropDownList` control. The middle tier should contain the code used to retrieve the names and phone numbers from the database. The information tier should contain the `db_Phone.mdb` database where the phone-number information is stored.

28.12 (***US State Facts Application***) Over the next three tutorials, you will create a **US State Facts** application. This application is designed to allow users to review their knowledge about specific U.S. states. This application should consist of two ASPX pages. The first page (named `States`) should display a `ListBox` containing 10 different state names. These state names are stored in the `db_StateFacts.mdb` database. The user should be allowed to select a state name and click a `Button` to retrieve information about the selected state from the database. The information should be displayed on a different ASPX page (named `StateFacts`). The `StateFacts` page should display an image of the state flag and list the state capital, state flower, state tree and state bird (retrieved from the database) in a `Table`. You will be provided with images of the state flags. For this exercise, you need only organize the components (the `States` and `StateFacts` ASPX pages, the `db_StateFacts.mdb` database and the code that performs the specified functionality) of this Web application into separate tiers. Decide which components belong in which tiers. You will begin building the solution, using Visual Studio .NET, in the next tutorial.

Answer: The client tier should contain the ASPX pages' GUIs, including a `ListBox` for the different state names. The middle tier should contain the code used to retrieve the state names and information from the database. The information tier should contain the `db_StateFacts.mdb` database where the state information is stored.

28.13 (***Road Sign Review Application***) Over the next three tutorials, you will create a **Road Sign Review** application. The **Road Sign Review** application should consist of two ASPX pages. This application displays road signs for users to review and allows them to schedule a driving test. The first page (named `RoadSigns`) should display 15 road signs in a `Table`. You will be provided images of the road signs. When the mouse pointer is moved over a sign, the name of the sign will appear in a tooltip in the Web browser window. The table should display the images by retrieving their information from the `db_RoadSigns.mdb` database. This page also will contain two `TextBoxes` and a `Button` to allow users to provide their information to register for a driving test. When users click the **Register** `Button`, the second page (`RoadTestRegistered`) displays information confirming that the user has registered for a driving test. For this exercise, you need only organize the components (the `RoadSigns` and `RoadTestRegistered` ASPX pages, the `db_RoadSigns.mdb` database and the code that performs the specified functionality) of this Web application into separate tiers. Decide which components belong in which tiers. You will begin building the solution, using Visual Studio .NET, in the next tutorial.

Answer: The client tier should contain the ASPX pages' GUIs, one of which will contain a `Table` control for the various road signs. The middle tier should contain the code used to retrieve the road sign names from the database. The information tier should contain the `db_RoadSigns.mdb` database where the road-signs information is stored.

TUTORIAL 29

Bookstore Application: Client Tier

Introducing Web Controls

Solutions

Instructor's Manual Exercise Solutions Tutorial 29

MULTIPLE-CHOICE QUESTIONS

29.1 You change the _____ property of an ASPX page to specify the color that displays in the background of the page.

a) `BackColor`
b) `bgColor`
c) `BackgroundColor`
d) `Color`

29.2 `Button`, `Label` and `Table` controls for ASPX pages can be accessed from the _____ tab.

a) **Web Forms**
b) **Components**
c) **Data**
d) Both a and b.

29.3 The _____ attribute specifies the position of a Web control on an ASPX page.

a) `position`
b) `location`
c) `style`
d) `coordinate`

29.4 Unlike the Windows `Form` Designer, the Web Form Designer _____.

a) does not provide two viewing modes
b) provides two viewing modes
c) allows you to design the graphical user interface
d) does not allow you to design the user interface

29.5 The `BorderStyle` property of the `Image` control _____.

a) specifies the color of the border
b) specifies the type of border that displays around the `Image` control
c) specifies the width of the border
d) Both a and b.

29.6 Setting the `BorderStyle` property to `Outset` makes a control appear _____.

a) raised
b) with a bold border
c) with the specified border width
d) with the specified border color

29.7 Every _____ of a `Table` Web control can contain one or more _____.

a) `TableRow`; `TableColumns`
b) `TableColumn`; `TableRows`
c) `TableRow`; `TableCells`
d) `TableCell`; `TableRows`

29.8 For you to create an **ASP.NET Web Application** project, _____ must be running.

a) IIS
b) Microsoft Access
c) Microsoft Word
d) Internet Explorer

29.9 The _____ mode allows you to create the ASPX page's GUI by dragging and dropping controls on the page.

a) **HTML**
b) **Design**
c) **Visual**
d) **GUI**

29.10 To specify the position of a Web control, set the _____ and _____ values of the _____ attribute.

a) `X`, `Y`, `style`
b) `X`, `Y`, `position`
c) `TOP`, `LEFT`, `style`
d) `TOP`, `LEFT`, `position`

Answers: 29.1) b. 29.2) a. 29.3) c. 29.4) b. 29.5) b. 29.6) a. 29.7) c. 29.8) a. 29.9) b. 29.10) c.

EXERCISES

[Note: In these exercises, we may ask you to set an ASPX page as the application's start page, meaning that this page will appear first when the application is run. You can set an ASPX page as the start page by right clicking the file in the **Solution Explorer** *and selecting* **Set As Start Page**.*]*

29.11 (***Phone Book Application: GUI***) Create the user interface for the **Phone Book** application. The design for the two pages for this application is displayed in Fig. 29.25.

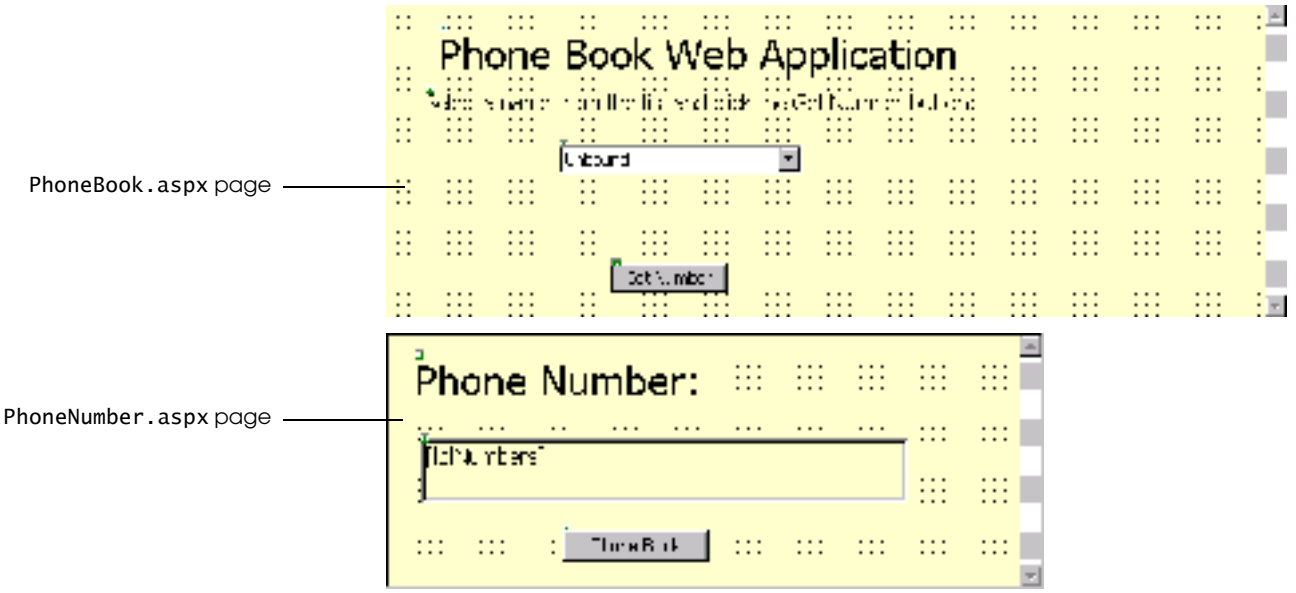


Figure 29.25 **Phone Book** application ASPX pages' design.

a) ***Creating an ASP.NET Web application***. Create an ASP.NET Web application, and name it `PhoneBook`. Rename the ASPX page to `PhoneBook.aspx`, and set `PhoneBook.aspx` as the start page.

b) ***Changing the background color***. Change the background color of your ASPX page (`PhoneBook.aspx`) to the light-yellow **Web Palette** color (located in the sixth column of the 12th row) by using the `bgColor` property as demonstrated in this tutorial. Change the `title` of the ASPX page to `Phone Book`.

c) ***Adding a `Label`***. Create a `Label`, set the font size to `X-Large` and change the `Text` property to `Phone Book Web Application`. Set the `LEFT:` portion of the `style` attribute value to `40px` and the `TOP:` portion to `17px`. Name the control `lblPhoneBook`.

d) ***Adding another `Label`***. Create another `Label`, and set the `Text` property to `Select a name from the list and click the Get Number Button`. Set the `LEFT:` portion of the `style` attribute value to `30px` and the `TOP:` portion to `65px`. Name this Web control `lblInstructions`.

e) ***Adding a `DropDownList` Web control***. Create a `DropDownList` Web control by dragging and dropping it from the **Toolbox** onto the ASPX page. The `DropDownList` Web control looks similar to the `ComboBox` Windows `Form` control. Set the width to `190px`, and set the `LEFT:` portion of the `style` attribute value to `134px` and the `TOP:` portion to `108px`. Name the `DropDownList` `cboNames`.

f) ***Adding a `Button`***. Create a `Button`, set its width to `90px` and change the `Text` property to `Get Number`. Set the `LEFT:` portion of the `style` attribute value to `175px` and the `TOP:` portion to `200px`. Name the Web control `btnGet`.

g) ***Adding another ASPX page to the Phone Book application***. Add another ASPX page to the **Phone Book** application, name it `PhoneNumber.aspx` and change the background to the light-yellow color. Change the `title` property to `Phone Number`.

h) ***Adding a `Label` to `PhoneNumber.aspx`***. Create a `Label` and name it `lblPhoneNumber`. Set the font size to `X-Large` and change the `Text` property to `Phone Number:`. Set the `LEFT:` portion of the `style` attribute value to `20px` and the `TOP:` portion to `15px`.

i) ***Adding another Label.*** Create another `Label`, set its `BorderStyle` to `Inset` and set its height and width to `50px` and `380px`, respectively. Clear the text of the `Label`. Name the `Label` `lblNumbers`, and set the `LEFT:` portion of the `style` attribute value to `25px` and the `TOP:` portion to `80px`.

j) ***Adding a Button to the PhoneNumber.aspx page.*** Create a `Button`, set its width to `115px` and change the `Text` property to `Phone Book`. Set the `LEFT:` portion of the `style` attribute value to `135px` and the `TOP:` portion to `150px`. Name the `Button` `btnPhoneBook`.

k) ***Saving the solution file.*** Save the solution file to the `PhoneBook` directory located in the root directory of your Web server, as you did in *Step 8* of the box, *Creating an ASP.NET Web Application*.

Answer: For this exercise, readers are asked to create the visual design of the page. They create the design by dragging and dropping controls on the page. There is no code paste-up for this exercise.

29.12 (***US State Facts Application: GUI***) Create the user interface for the **US State Facts** application. The design for the two pages of this application is displayed in Fig. 29.26.

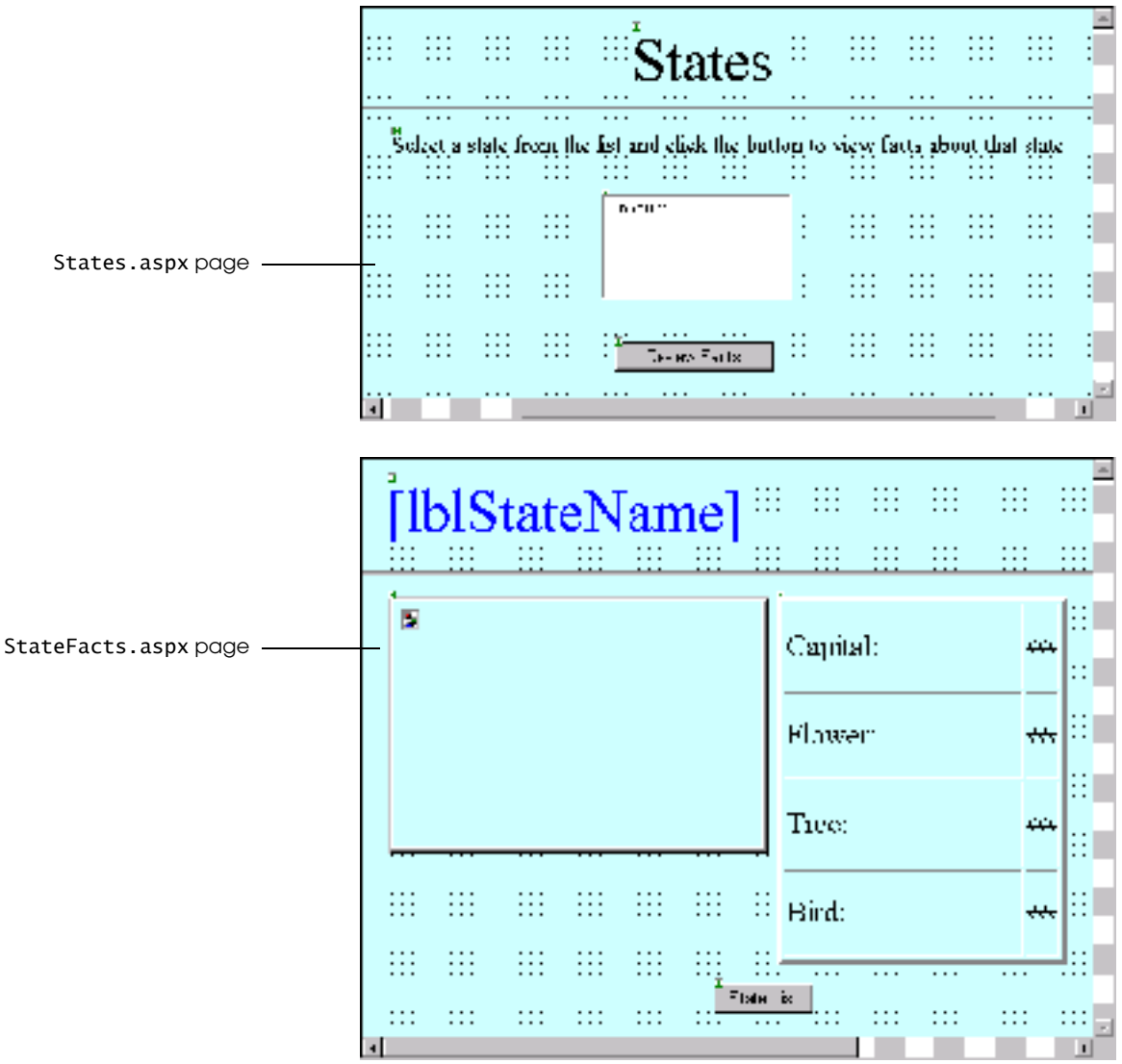


Figure 29.26 **US State Facts** application ASPX pages' design.

a) ***Creating an ASP.NET Web application***. Create a new ASP.NET Web application, and name it `USStateFacts`. Rename the first ASPX page to `States.aspx`, and set `States.aspx` as the start page.

b) ***Changing the background color***. Change the background color of the `States.aspx` page to the light-blue **Web Palette** color (located in the sixth column of the second row) by using the `bgColor` property as demonstrated in this tutorial. Change the `title` property of the ASPX page to `States`.

c) ***Adding a `Label` to `States.aspx`***. Create a `Label` Web control, and place it on the page. Set the font size to `XX-Large`, and change the `Text` property to `States`. Change the `LEFT:` portion of its `style` attribute value to `390px`, and set the `TOP:` portion to `15px`. Name the Web control `lblStates`.

d) ***Adding a horizontal rule to `States.aspx`***. Create a horizontal rule, place it on the ASPX page and set its `width` to `150%`. When setting its position, change the `TOP:` value to `80px`, set the `LEFT:` value to `0px` and specify the `Height:` as `4px`. Name the horizontal rule `hrzStates`.

e) ***Adding another `Label` to `States.aspx`***. Create another `Label`, and place it beneath the horizontal rule. Change the font size to `Medium`, and set the `Text` property to `Select a state from the list and click the button to view facts about that state`. Change the `LEFT:` portion of its `style` attribute value to `195px`, and set the `TOP:` portion to `100px`. Name this Web control `lblInstructions`.

f) ***Adding a `ListBox` to `States.aspx`***. Create a `ListBox`, and place it on the ASPX page. Set its `Height` property to `100px` and its `Width` property to `155px`. Set the `LEFT:` portion of the `style` attribute value to `365px` and the `TOP:` portion to `150px`. Name the `ListBox` `lstStates`.

g) ***Adding a `Button` to `States.aspx`***. Create a `Button`, and place it on the page. Set its `Text` property to `Review Facts` and its `Width` property to `130px`. Change the `LEFT:` portion of the `style` attribute value to `375px` and the `TOP:` portion to `270px`. Name the `Button` `btnFacts`.

h) ***Adding another ASPX page to the US State Facts application***. Add another ASPX page to the **US State Facts** application, name it `StateFacts.aspx` and change the background color to light blue.

i) ***Adding a `Label` to `StateFacts.aspx`***. Create a `Label`, name it `lblStateName`, set its font size to `XX-Large` and change its `ForeColor` property to `Blue`. Clear the `Label`'s text. Set its position by setting the `LEFT:` portion of the `style` attribute value to `20px` and the `TOP:` portion to `15px`.

j) ***Adding a horizontal rule***. Place the horizontal rule beneath the `Label` and set its `TOP:` position to `90px`, its `LEFT:` position to `0px` and its `Height:` to `4px`. Change the `width` to `150%`. Name the horizontal rule `hrzStateFacts`.

k) ***Adding an `Image` control to `StateFacts.aspx`***. Create an `Image` control, and set its `BorderStyle` to `Outset`. Change the `BorderWidth` to `5px`. Set its height to `200px` and its width to `300px`. Set the position of the `Image` by changing the `LEFT:` portion of the `style` attribute value to `20px` and the `TOP:` portion to `110px`. Name the Web control `imgFlag`.

l) ***Adding a `Table` to `StateFacts.aspx`***. Create a `Table` with four rows and two columns. Set the `BorderStyle` to `Outset`, the `BorderWidth` to `5px` and `GridLines` to `Both`. Set the height and width of each `TableCell` of the first column to `70px` and `200px`, respectively, and set the `Font` property's `Size` to `Large`. Set the `Text` property of the cells in the first column to `Capital:`, `Flower:`, `Tree:` and `Bird:`, respectively. Change the `LEFT:` portion of the `style` attribute value to `335px` and the `TOP:` portion to `110px`. Name the `Table` control `tblState`.

m) ***Adding a `Button` to `StateFacts.aspx`***. Create a `Button`, change its text to `State List` and change the `LEFT:` portion of the `style` attribute value to `285px` and the `TOP:` portion to `425px`. Name the `Button` control `btnStateList`.

n) ***Saving the solution file***. Save the solution file to the `USStateFacts` directory located in the root directory of your Web server, as you did in *Step 8* of the box, *Creating an ASP.NET Web Application*.

Answer: For this exercise, readers are asked to create the visual design of the page. They create the design by dragging and dropping controls on the page. There is no code paste-up for this exercise.

29.13 (*Road Sign Review Application: GUI*) Create the user interface for the **Road Sign Review** application. The design for the two pages of this application is displayed in Fig. 29.27.

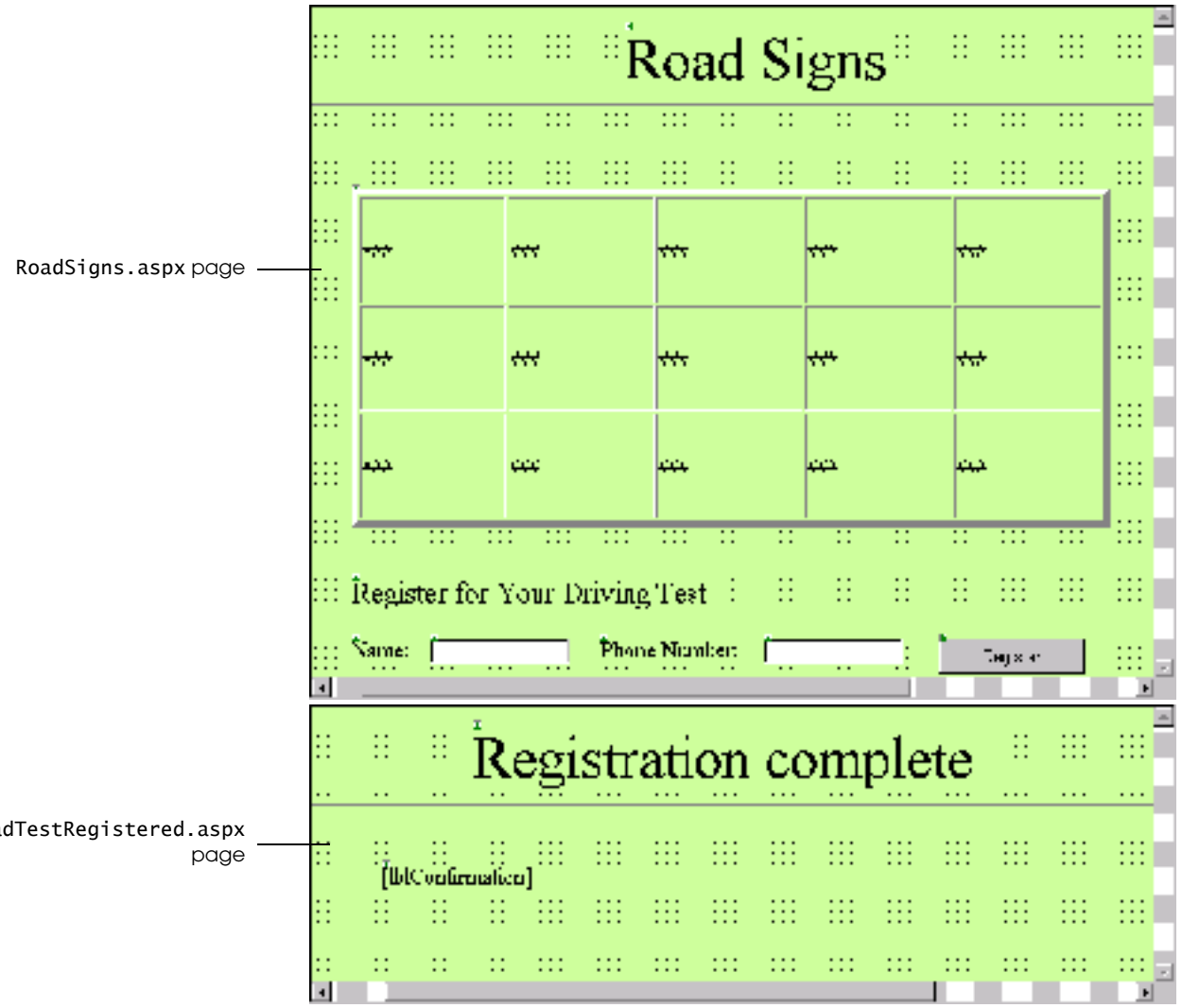


Figure 29.27 **Road Signs** application ASPX pages' design.

a) ***Creating an ASP.NET Web application***. Create a new ASP.NET Web application, and name it **RoadSignReview**. Change the name of the existing ASPX page to `RoadSigns.aspx`, and set `RoadSigns.aspx` as the start page.

b) ***Changing the background color***. Change the background color of `RoadSigns.aspx` to the light-green **Web Palette** color (located in the sixth column of the 14th row) by using the `bgColor` property as demonstrated in this tutorial. Change the `title` of the ASPX page to `RoadSigns`.

c) ***Adding a `Label` to `RoadSigns.aspx`***. Create a `Label`, and set its font size to `XX-Large`. Change the `Text` property to `Road Signs`. Set its position by changing the `style` attribute value's `LEFT:` portion to `295px` and the `TOP:` portion to `16px`. Name the `Label` control `lblRoadSigns`.

d) ***Adding a horizontal rule to `RoadSigns.aspx`***. Create a horizontal rule. Set its `width` to `150%`, and set the `TOP:` position to `80px`, the `LEFT:` position to `0px` and the height to `4px`. Name the horizontal rule `hrzRoadSigns`.

e) ***Adding a Table to RoadSigns.aspx***. Create a `Table` with three rows and five columns. Set the `BorderStyle` to `Outset`, the `BorderWidth` to `5px` and the `GridLines` property to `Both`. Also, set the `Table`'s `Height` property to `279px` and `Width` property to `626px`. Set each row's height to `50px` and each `TableCell`'s width to `20px`. Change the `style` attribute value by setting `LEFT:` to `70px` and `TOP:` to `150px`. Name the `Table` control `tblRoadSigns`.

f) ***Adding a Label to RoadSigns.aspx***. Create a `Label`, and set its font size to `Large`. Change the `Text` property to `Register for Your Driving Test`. Set its position by changing the `style` attribute value's `LEFT:` portion to `70px` and `TOP:` portion to `470px`. Name the Web control `lblRegister`.

g) ***Adding a Label and TextBox to RoadSigns.aspx***. Create a `Label` and set its text to `Name:`. Set its font size to `Medium`, and change its position to `LEFT: 70px` and `TOP: 520px`. Name the `Label` control `lblName`. Create a `TextBox`, and place it next to the **Name:** `Label`. Set its height to `20px` and width to `115px`. Change the position to `LEFT: 135px` and `TOP: 520px`. Name the `TextBox` control `txtName`.

h) ***Adding another Label and TextBox pair to RoadSigns.aspx***. Create a `Label` and set its text to `Phone Number:`. Set its font size to `Medium`, and change its position to `LEFT: 275px` and `TOP: 520px`. Name the `Label` control `lblPhoneNumber`. Create a `TextBox`, and place it next to the **Phone Number:** `Label`. Set its height to `20px` and width to `115px`. Change its position to `LEFT: 410px` and `TOP: 520px`. Name the `TextBox` control `txtPhoneNumber`.

i) ***Adding a Button to RoadSigns.aspx***. Create a `Button`, set its `Text` to `Register`, and change its height and width to `30px` and `120px`, respectively. Change the position of the `Button` by setting the `LEFT:` portion of the style attribute value to `555px` and the `TOP:` portion to `520px`. Name the `Button` control `btnRegister`.

j) ***Adding another ASPX page to the Road Sign Review application***. Add another ASPX page to the application, name it `RoadTestRegistered.aspx` and change the background color to light green.

k) ***Adding a Label to RoadTestRegistered.aspx***. Create a `Label`, setting its font size to `XX-Large` and its `Text` property to `Registration Complete`. Change its position by setting the `LEFT:` portion of its `style` attribute value to `200px` and the `TOP:` portion to `15px`. Name the `Label` control `lblRegistration`.

l) ***Adding a horizontal rule to RoadTestRegistered.aspx***. Create a horizontal rule. Set its `width` to `150%`, the `TOP:` position to `80px`, the `LEFT:` position to `0px` and the height to `4px`. Name the horizontal rule `hrzRoadTestRegistered`.

m) ***Adding another Label to RoadTestRegistered.aspx***. Create a `Label`, name it `lblConfirmation` and set its font size to `Medium`. Delete the `Text` property value, leaving it blank. Change its position by setting the `LEFT:` portion of its `style` attribute value to `125px` and the `TOP:` portion to `130px`.

n) ***Saving the solution file***. Save the solution file to the `RoadSignReview` directory located in the root directory of your Web server, as you did in *Step 8* of the box, *Creating an ASP.NET Web Application*.

Answer: For this exercise, readers are asked to create the visual design of the page. They create the design by dragging and dropping controls on the page. There is no code paste-up for this exercise.

TUTORIAL 30

Bookstore Application: Information Tier

Examining the Database and Creating Database Components

Solutions

Instructor's Manual Exercise Solutions Tutorial 30

MULTIPLE-CHOICE QUESTIONS

30.1 _____ is an example of a database product.

a) Microsoft Access
b) Microsoft SQL Server
c) Oracle
d) All of the above.

30.2 An advantage of using information in a database is that _____.

a) the data can be updated in real time
b) information that changes need be updated only in one location
c) Both a and b.
d) None of the above.

30.3 When a funnel appears to the right of a field's `CheckBox` in the **Query Builder** dialog, it indicates that _____.

a) information will be updated in the specified field
b) a value will be retrieved from that field according to specified criteria
c) the field will not be included in the SQL statement
d) None of the above.

30.4 The `Parameters` property of _____ contains a collection of parameters.

a) `OleDbConnection`
b) `OleDbDataConnection`
c) `OleDbDataCommand`
d) `OleDbCommand`

30.5 The _____ can be used to create an `OleDbConnection`.

a) **Server Explorer** window
b) **Query Builder** tool
c) Both a and b.
d) None of the above.

30.6 You use the _____ object to create SQL statements for retrieving data from a database.

a) `OleDbConnection`
b) `OleDbDataReader`
c) `OleDbCommand`
d) None of the above.

30.7 The _____ is used when creating SQL statements visually for the `OleDbCommand` object's `CommandText` property.

a) **Server Explorer** window
b) **Query Builder** tool
c) Both a and b.
d) None of the above.

30.8 You use the _____ object to open a connection to the database.

a) `OleDbConnection`
b) `OleDbDataReader`
c) `OleDbCommand`
d) None of the above.

30.9 You use the _____ property of the `OleDbCommand` object to specify values for information that is not known in advance.

a) `Connection`
b) `Parameters`
c) `Field`
d) `Name`

30.10 Another name for the database tier is _____.

a) the information tier
b) the bottom tier
c) Both a and b.
d) None of the above.

Answers: 30.1) d. 30.2) c. 30.3) b. 30.4) d. 30.5) a. 30.6) c. 30.7) b. 30.8) a. 30.9) b. 30.10) c.

EXERCISES

30.11 (*Phone Book Application: Database*) Create the database connections and data command objects for the **Phone Book** application by using the **Server Explorer** window and the **Query Builder** tool.

a) ***Opening the application***. Open the **Phone Book** application that you created in Tutorial 29.

b) ***Copying the db_Phone.mdb database to the Databases directory.*** Copy the `C:\Examples\Tutorial30\Exercises\Databases\db_Phone.mdb` database to the `Databases` directory in IIS's `wwwroot` directory.

c) ***Using Server Explorer to add a connection to the database.*** In the **Server Explorer** window, add a connection to the `db_Phone.mdb` database. Drag and drop the connection object on the `PhoneBook.aspx` page. Name the connection object `objOleDbConnection`.

d) ***Using Query Builder for the PhoneBook.aspx page.*** Add an `OleDbCommand` to the `PhoneBook.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This command should retrieve all the names of the people from the database. Name this command object `objSelectNames`.

e) ***Adding a connection to the database to the PhoneNumber.aspx page.*** Using the **Server Explorer** window, drag and drop a database connection object on the `PhoneNumber.aspx` page. Name this connection object `objOleDbConnection`.

f) ***Using Query Builder for PhoneBook.aspx***. Add an `OleDbCommand` to the `PhoneNumber.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This configuration should retrieve the phone number of the person whose name will be selected from the `DropDownList` in the `PhoneBook.aspx` page by the user. You need to set the criteria to specify which person's phone number will be retrieved from the database. Name this command object `objSelectPhoneNumber`.

g) ***Saving the project.*** Select **File > Save All** to save your modified code.

Answer: Connect to the database by using the **Server Explorer** window, and specify data command objects by using the **Query Builder** tool.

30.12 (*US State Facts Application: Database*) Create the database connections and data command objects for the **US State Facts** application by using the **Server Explorer** window and the **Query Builder** tool.

a) ***Opening the application***. Open the **US State Facts** application that you created in Tutorial 29.

b) ***Copying the db_StateFacts.mdb database to the Databases directory.*** Copy the `C:\Examples\Tutorial30\Exercises\Databases\db_StateFacts.mdb` database to the `Databases` directory in IIS's `wwwroot` directory.

c) ***Using Server Explorer to add a connection to the database.*** In the **Server Explorer** window, add a connection to the `db_StateFacts.mdb` database. Drag and drop the connection object on the `States.aspx` page. Name this connection object `objOleDbConnection`.

d) ***Using Query Builder for the States.aspx page.*** Add an `OleDbCommand` to the `States.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This command should retrieve the names of the states from the **name** field of the `states` table in the database. Name this command object `objSelectNames`.

e) ***Adding a connection to the database to the StateFacts.aspx page.*** Using the **Server Explorer** window, drag and drop a database connection object on the database on the `StateFacts.aspx` page. Name this connection object `objOleDbConnection`.

f) ***Using Query Builder for StateFacts.aspx***. Add an `OleDbCommand` to the `StateFacts.aspx` page. Set the `Connection` property to the `OleDbConnection` object you added to the ASPX page, and use **Query Builder** to set the `CommandText`

property of the `OleDbCommand`. This configuration should retrieve all the information from the `states` table of the database about the state selected by the user. You need to set the criteria to specify which state's information will be retrieved from the database. Name this command object `objSelectStateInformation`.

g) ***Saving the project.*** Select **File > Save All** to save your modified code.

Answer: Connect to the database by using the **Server Explorer** window, and specify data command objects by using the **Query Builder** tool.

30.13 (*Road Sign Review Application: Database*) Create the database connections and data command objects for the **Road Sign Review** application by using the **Server Explorer** window and the **Query Builder** tool.

a) ***Opening the application***. Open the **Road Sign Review** application that you created in Tutorial 29.

b) ***Copying the `db_RoadSigns.mdb` database to the `Databases` directory.*** Copy the `C:\Examples\Tutorial30\Exercises\Databases\db_RoadSigns.mdb` database to the `Databases` directory in IIS's `wwwroot` directory.

c) ***Using Server Explorer to add a connection to the database.*** In the **Server Explorer** window, add a connection to the `db_RoadSigns.mdb` database. Drag and drop the connection object on the `RoadSigns.aspx` page. Name this command object `objOleDbConnection`.

d) ***Using Query Builder for the `RoadSigns.aspx` page.*** Add an `OleDbCommand` to the `RoadSigns.aspx` page. Set the `Connection` property to the `OleDbConnection` object that you added to the ASPX page, and use **Query Builder** to set the `CommandText` property of the `OleDbCommand`. This configuration should retrieve all the information about all the road signs from the `signs` table of the database. You will not need to specify criteria for this exercise, because all the information from the database needs to be retrieved. Name this command object `objSelectSignInformation`.

e) ***Saving the project.*** Select **File > Save All** to save your modified code.

Answer: Connect to the database by using the **Server Explorer** window, and specify data command objects by using the **Query Builder** tool.

31 TUTORIAL

Bookstore Application: Middle Tier

Introducing Code-Behind Files

Solutions

Instructor's Manual Exercise Solutions Tutorial 31

MULTIPLE-CHOICE QUESTIONS

31.1 The `Page_Load` event handler _____.

a) redirects the client browser to different Web pages
b) defines the functionality when a `Button` is clicked
c) executes any processing necessary to display a Web page
d) defines the functionality when a Web control is selected

31.2 The `Response.Redirect` method _____.

a) refreshes the current Web page
b) sends the client browser to a specified Web page
c) responds to user input
d) responds to the click of a `Button`

31.3 `Session` items are used in the **Bookstore** application because _____.

a) variables in ASP.NET Web applications must be created as `Session` items
b) values need to be shared among Web pages
c) `Session` items are simpler to create than instance variables
d) Both a and b.

31.4 Session state is used for _____ in ASP.NET.

a) tracking user-specific data
b) running an application
c) using a database
d) None of the above.

31.5 The file extension for an ASPX code-behind file written in C# is _____.

a) `.asp`
b) `.aspx`
c) `.aspx.cs`
d) `.code`

31.6 The `Response` object is a predefined ASP.NET object that _____.

a) connects to a database
b) retrieves information from a database
c) creates Web controls
d) provides methods for responding to client requests

31.7 The `Response.Redirect` method takes a(n) _____ as an argument.

a) URL
b) `int` value
c) `bool` value
d) `OleDbConnection` object

31.8 The _____ property specifies the image that an `Image` control displays.

a) `ImageGIF`
b) `ImageUrl`
c) `Image`
d) `Display`

31.9 The C# file that contains the ASPX page's corresponding class is called the _____.

a) ASPX file
b) code-behind file
c) class file
d) None of the above.

31.10 Information can be maintained across Web pages by adding a _____ to the `Session` object.

a) key-value pair
b) number
c) database connection object
d) None of the above.

Answers: 31.1) c. 31.2) b. 31.3) b. 31.4) a. 31.5) c. 31.6) d. 31.7) a. 31.8) b. 31.9) b. 31.10) a.

EXERCISES

31.11 (*Phone Book Application: Functionality*) Define the middle tier for the **Phone Book** application (Fig. 31.21).

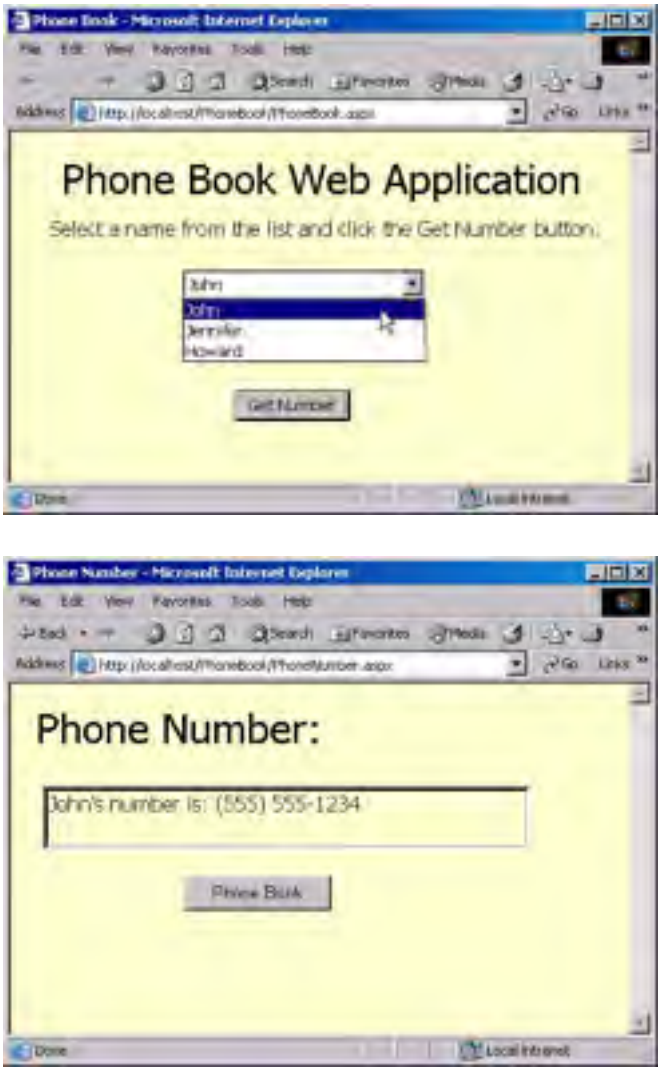


Figure 31.21 **Phone Book** application

a) ***Opening the application***. Open the **Phone Book** application that you created in Tutorial 29 and continued to develop in Tutorial 30.

b) ***Using `System.Data.OleDb` in `PhoneBook.aspx.cs`.*** Use the `System.Data.OleDb` namespace in `PhoneBook.aspx.cs`.

c) ***Rearranging and commenting the control declarations.*** Organize the control declarations as they appear on the Web Form. Add a comment above each declaration.

d) ***Defining the `Page_Load` event handler of `PhoneBook.aspx` page.*** Use the `Open` method to open the connection to the database. Create a data reader to read the information specified by the data command object.

e) ***Populating the `DropDownList` with names.*** Add a `while` statement to `PhoneBook.aspx`'s `Page_Load` method. This loop should add to the `DropDownList` each person's name read by the data reader.

f) ***Closing the reader and connection.*** Close the data reader and the connection to the database by invoking their `Close` methods.

g) ***Creating the Get Number `Button`'s `Click` event handler for the `PhoneBook.aspx` page.*** Double click the **Get Number** `Button` to create the `Button`'s `Click` event handler.

h) ***Creating a Session item.*** In the `Click` event handler, create a `Session` item to store the selected name.

i) ***Redirecting to the PhoneNumber.aspx page.*** In the `Click` event handler, use the `Response.Redirect` method to redirect the client browser to the `PhoneNumber.aspx` page.

j) ***Using System.Data.OleDb in PhoneNumber.aspx.cs.*** Use the `System.Data.OleDb` namespace in `PhoneNumber.aspx.cs`.

k) ***Rearranging and commenting the control declarations.*** Organize the control declarations as they appear on the Web Form. Add a comment above each declaration.

l) ***Defining the Page_Load event handler for the PhoneNumber.aspx page.*** Use the `Open` method to open the connection to the database. Access the `Session` item to retrieve the selected name. Specify this name as the parameter value for the `OleDbCommand` object. Create a data reader to read the information specified by the data command object.

m) ***Displaying the selected name and phone number.*** In the `Page_Load` event handler, read the desired phone number from the data reader. Display the selected name and corresponding phone number in the `lblNumbers Label`.

n) ***Closing the reader and connection.*** Close the data reader and the connection to the database by invoking their `Close` methods.

o) ***Creating the Phone Book Button's Click event handler for the PhoneNumber.aspx page.*** Double click the **Phone Book** `Button` to create the `Button`'s `Click` event handler.

p) ***Redirecting to the PhoneBook.aspx page.*** In the `Click` event handler, use the `Response.Redirect` method to redirect the client browser to the `PhoneBook.aspx` page.

q) ***Running the application.*** Select **Debug > Start** to run your application. Select a name from the `ComboBox`, then click the **Get Number** `Button`. Confirm that the information is displayed correctly on the `PhoneNumber.aspx` page. Click the **Phone Book** `Button` to return to the `PhoneBook.aspx` page. Test the other two entries in the `ComboBox` to ensure that your application works correctly.

r) ***Closing the application.*** Close your running application by clicking the browser window's close box.

s) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 31.11 Solution
2  // PhoneBook.aspx.cs
3
4  using System;
5  using System.Collections;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Web;
10 using System.Web.SessionState;
11 using System.Web.UI;
12 using System.Web.UI.WebControls;
13 using System.Web.UI.HtmlControls;
14 using System.Data.OleDb;
15
16 namespace PhoneBook
17 {
18    /// <summary>
19    /// Summary description for PhoneBook.
20    /// </summary>
21    public class PhoneBook : System.Web.UI.Page
22    {
23       // Label to display title
```

```
24          protected System.Web.UI.WebControls.Label lblPhoneBook;
25
26          // Label to display instructions
27          protected System.Web.UI.WebControls.Label lblInstructions;
28
29          // DropDownList to choose name in phone book
30          protected System.Web.UI.WebControls.DropDownList cboNames;
31
32          // Button to get the phone number of the selected name
33          protected System.Web.UI.WebControls.Button btnGet;
34
35          // OleDbConnection to connect to the phone book database
36          protected System.Data.OleDb.OleDbConnection objOleDbConnection;
37
38          // OleDbCommand to retrieve names from the database
39          protected System.Data.OleDb.OleDbCommand objSelectNames;
40
41          // invoke when page is loaded
42          private void Page_Load( object sender, System.EventArgs e )
43          {
44             objOleDbConnection.Open(); // open connection to database
45
46             // declare reader to read from database
47             OleDbDataReader objReader;
48
49             // create reader to read from database
50             objReader = objSelectNames.ExecuteReader();
51
52             // add names to DropDownList
53             while ( objReader.Read() )
54             {
55                // add item to DropDownList
56                cboNames.Items.Add(
57                   Convert.ToString( objReader[ "Name" ] ) );
58             }
59
60             objReader.Close(); // close the reader
61
62             // close the connection to the database
63             objOleDbConnection.Close();
64
65          } // end method Page_Load
66
67          // Web Form Designer generated code
68
69          // handles Get Button's Click event
70          private void btnGet_Click(
71             object sender, System.EventArgs e )
72          {
73             Session[ "Name" ] =
74                Convert.ToString( cboNames.SelectedItem );
75
76             // redirect to another ASPX page
77             Response.Redirect( "PhoneNumber.aspx" );
78
79          } // end method btnGet_Click
80
81       } // end class PhoneBook
82    }
```

```
1  // Exercise 31.11 Solution
2  // PhoneNumber.aspx.cs
3
4  using System;
5  using System.Collections;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Web;
10 using System.Web.SessionState;
11 using System.Web.UI;
12 using System.Web.UI.WebControls;
13 using System.Web.UI.HtmlControls;
14 using System.Data.OleDb;
15
16 namespace PhoneBook
17 {
18    /// <summary>
19    /// Summary description for PhoneNumber.
20    /// </summary>
21    public class PhoneNumber : System.Web.UI.Page
22    {
23       // Label to display title
24       protected System.Web.UI.WebControls.Label lblPhoneNumber;
25
26       // Label to display phone number information
27       protected System.Web.UI.WebControls.Label lblNumbers;
28
29       // Button to go to the phone book page
30       protected System.Web.UI.WebControls.Button btnPhoneBook;
31
32       // OleDbConnection to connect to the phone book database
33       protected System.Data.OleDb.OleDbConnection objOleDbConnection;
34
35       // OleDbCommand to retrieve a phone number from the database
36       // given a name
37       protected System.Data.OleDb.OleDbCommand objSelectPhoneNumber;
38
39       // invoke when page is loaded
40       private void Page_Load( object sender, System.EventArgs e )
41       {
42          // represents the name
43          string strName = Convert.ToString( Session[ "Name" ] );
44
45          objOleDbConnection.Open();// open connection to the database
46
47          // specify name to retrieve phone number for
48          objSelectPhoneNumber.Parameters[ "Name" ].Value = strName;
49
50          // declare reader to read from the database
51          OleDbDataReader objReader;
52
53          // create reader to read from the database
54          objReader = objSelectPhoneNumber.ExecuteReader();
55
56          objReader.Read(); // start data reader
57
58          // display name and number in Label
59          lblNumbers.Text = strName + "'s number is: " +
60             Convert.ToString( objReader[ "Phone_Numbers" ] );
```

```
61
62          objReader.Close(); // close data reader
63
64          // close connection to the database
65          objOleDbConnection.Close();
66
67       } // end method Page_Load
68
69       // Web Form Designer generated code
70
71       // handles Phone Book Button's Click event
72       private void btnPhoneBook_Click(
73          object sender, System.EventArgs e )
74       {
75          Response.Redirect( "PhoneBook.aspx" );
76
77       } // end method btnPhoneBook_Click
78
79    } // end class PhoneNumber
80 }
```

31.12 (*US State Facts Application: Functionality*) Define the middle tier for the **US State Facts** application (Fig. 31.22).

a) ***Opening the application***. Open the **US State Facts** application that you created in Tutorial 29 and continued to develop in Tutorial 30.

b) ***Copying the `FlagImages` directory to your project directory.*** Copy the `C:\Examples\Tutorial31\Exercises\Images\FlagImages` directory to the `USStateFacts` directory.

c) ***Using `System.Data.OleDb` in `States.aspx.cs`.*** Use the `System.Data.OleDb` namespace in `States.aspx.cs` before the class declaration.

d) ***Rearranging and commenting the control declarations.*** Organize the control declarations as they appear on the Web Form. Add a comment above each declaration.

e) ***Defining the `Page_Load` event handler for the `States.aspx` page.*** Use the `Open` method to open the connection to the database. Create a data reader to read the information specified by the data command object.

f) ***Populating the `ListBox` with state names in the `States.aspx` page.*** Add a `while` statement to `States.aspx`'s `Page_Load` method. This loop should add to the `ListBox` the name of each state read by the data reader.

g) ***Creating a `Button`'s `Click` event handler for the `States.aspx` page.*** Double click the **Review Facts** `Button` to create the `Button`'s `Click` event handler.

h) ***Creating a `Session` item.*** Create a `Session` item in the `Click` event handler and assign it to the state name that the user selects from the `ListBox`.

i) ***Redirecting to the `StateFacts.aspx` page.*** In the `Click` event handler, use the `Redirect.Response` method to redirect the client browser to the `StateFacts.aspx` page.

j) ***Using `System.Data.OleDb` in `StateFacts.aspx.cs`.*** Use the `System.Data.OleDb` namespace in `StateFacts.aspx.cs`.

k) ***Rearranging and commenting the control declarations.*** Organize the control declarations as they appear on the Web Form. Add a comment above each declaration.

l) ***Defining the `Page_Load` event handler of `StateFacts.aspx` page.*** Use the `Open` method to open the connection to the database. Access the `Session` object to retrieve the selected state name. Specify this name as a parameter value for the `OleDbCommand` object. Create a data reader to read the information specified by the data command object.

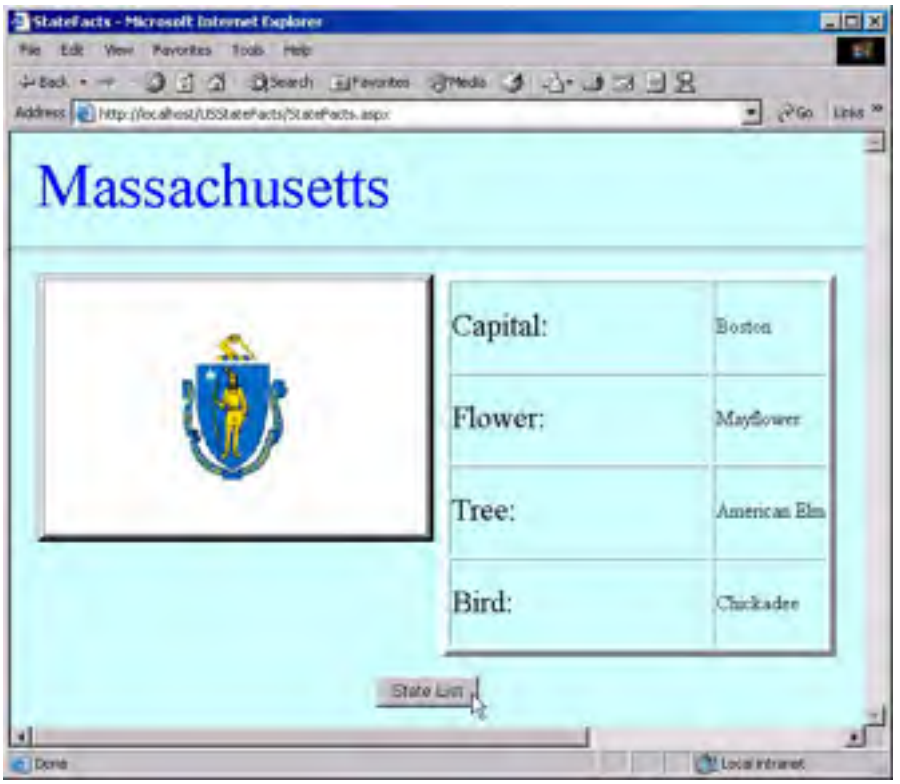


Figure 31.22 **US State Facts** Application

m) ***Displaying the state facts in the `Table`.*** In the `Page_Load` event handler, use the data reader to retrieve the desired state's facts. Display the selected state's name in the `lblStateName Label`. Set the `ImageUrl` property of the `Image` control to the location of the selected state's flag image. Display the name of the state capital, flower, tree and bird in the `Table` on the `StateFacts.aspx` page.

n) ***Closing the connection.*** Close the connection to the database by invoking the `Close` method.

o) ***Creating the State List `Button`'s `Click` event handler for the `StateFacts.aspx` page.*** Double click the **State List** `Button` to create the `Button`'s `Click` event handler.

p) ***Redirecting to the States.aspx page.*** In the `Click` event handler use the `Redirect.Response` method to redirect the client browser to the `States.aspx` page.

q) ***Running the application.*** Select **Debug > Start** to run your application. Select a state from the `ComboBox`, then click the **Review Facts** `Button`. Confirm that the information is displayed correctly on the `StateFacts.aspx` page. Click the **State List** `Button` to return to the `States.aspx` page. Test several other entries in the `ComboBox` to ensure that your application works correctly.

r) ***Closing the application.*** Close your running application by clicking the browser window's close box.

s) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 31.12 Solution
2  // States.aspx.cs
3
4  using System;
5  using System.Collections;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Web;
10 using System.Web.SessionState;
11 using System.Web.UI;
12 using System.Web.UI.WebControls;
13 using System.Web.UI.HtmlControls;
14 using System.Data.OleDb;
15
16 namespace USStateFacts
17 {
18    /// <summary>
19    /// Summary description for States.
20    /// </summary>
21    public class States : System.Web.UI.Page
22    {
23       // Label to display title
24       protected System.Web.UI.WebControls.Label lblStates;
25
26       // Label to display instructions
27       protected System.Web.UI.WebControls.Label lblInstructions;
28
29       // ListBox to choose a state
30       protected System.Web.UI.WebControls.ListBox lstStates;
31
32       // Button to get the facts for the selected state
33       protected System.Web.UI.WebControls.Button btnFacts;
34
35       // OleDbConnection to connect to the state facts database
36       protected System.Data.OleDb.OleDbConnection objOleDbConnection;
37
38       // OleDbCommand to retrieve the state names from the database
39       protected System.Data.OleDb.OleDbCommand objSelectNames;
40
41       // invoked when Page is loaded
42       private void Page_Load( object sender, System.EventArgs e )
43       {
44          objOleDbConnection.Open();// open connection to the database
45
46          // declare reader to read from database
47          OleDbDataReader objReader;
48
```

```
49          // create reader to read from database
50          objReader = objSelectNames.ExecuteReader();
51
52          // add names to the ListBox
53          while ( objReader.Read() )
54          {
55             // add item to ListBox
56             lstStates.Items.Add(
57                Convert.ToString( objReader[ "Name" ] ) );
58          }
59
60          objReader.Close(); // close the reader
61
62          // close the connection to the database
63          objOleDbConnection.Close();
64
65       } // end method Page_Load
66
67       // Web Form Designer generated code
68
69       // handles Facts Button Click event
70       private void btnFacts_Click(
71          object sender, System.EventArgs e )
72       {
73          // create session variable named strName
74          Session[ "strName" ] = lstStates.SelectedItem.Text;
75
76          // redirect to another ASPX page
77          Response.Redirect( "StateFacts.aspx" );
78
79       } // end method btnFacts_Click
80
81    } // end class States
82 }
```

```
1  // Exercise 31.12 Solution
2  // StateFacts.aspx.cs
3
4  using System;
5  using System.Collections;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Web;
10 using System.Web.SessionState;
11 using System.Web.UI;
12 using System.Web.UI.WebControls;
13 using System.Web.UI.HtmlControls;
14 using System.Data.OleDb;
15
16 namespace USStateFacts
17 {
18    /// <summary>
19    /// Summary description for StateFacts.
20    /// </summary>
21    public class StateFacts : System.Web.UI.Page
22    {
23       // Label to display title
24       protected System.Web.UI.WebControls.Label lblStateName;
```

```
25
26       // Image to display the state flag
27       protected System.Web.UI.WebControls.Image imgFlag;
28
29       // Table to display the state facts
30       protected System.Web.UI.WebControls.Table tblState;
31
32       // Button to go back to the states page
33       protected System.Web.UI.WebControls.Button btnStateList;
34
35       // OleDbConnection to connect to the state facts database
36       protected System.Data.OleDb.OleDbConnection objOleDbConnection;
37
38       // OleDbCommand to retrieve state facts given a state name
39       protected System.Data.OleDb.OleDbCommand
40          objSelectStateInformation;
41
42       // invoked when Page is loaded
43       private void Page_Load( object sender, System.EventArgs e )
44       {
45          // display name of selected book
46          lblStateName.Text =
47             Convert.ToString( Session[ "strName" ] );
48
49          objOleDbConnection.Open(); // open connection to database
50
51          // specify state name to retrieve information about
52          objSelectStateInformation.Parameters[ "Name" ].Value =
53             Convert.ToString( Session[ "strName" ] );
54
55          // declare database reader to read from database
56          OleDbDataReader objReader;
57
58          // create database reader to read from database
59          objReader = objSelectStateInformation.ExecuteReader();
60
61          // while reader is reading database, retrieve data from
62          // specified positions and display them on page
63          while ( objReader.Read() )
64          {
65             // display flag for selected state
66             imgFlag.ImageUrl = "FlagImages/" +
67                Convert.ToString( objReader[ "flag" ] );
68
69             // display information from database in Table
70             tblState.Rows[ 0 ].Cells[ 1 ].Text =
71                Convert.ToString(  objReader[ "capital" ] );
72             tblState.Rows[ 1 ].Cells[ 1 ].Text =
73                Convert.ToString(objReader[ "flower" ] );
74             tblState.Rows[ 2 ].Cells[ 1 ].Text =
75                Convert.ToString(objReader[ "tree" ] );
76             tblState.Rows[ 3 ].Cells[ 1 ].Text =
77                Convert.ToString(objReader[ "bird" ] );
78          }
79
80          objOleDbConnection.Close(); // close connection to database
81
82       } // end method Page_Load
83
84       // Web Form Designer generated code
```

```
85
86          // handles State List Button's Click event
87          private void btnStateList_Click(
88             object sender, System.EventArgs e )
89          {
90             // redirects to States.aspx page
91             Response.Redirect( "States.aspx" );
92
93          } // end method btnStateList_Click
94
95       } // end class StateFacts
96    }
```

31.13 (*Road Sign Review Application: Functionality*) Define the middle tier for the **Road Sign Review** application (Fig. 31.23).

Figure 31.23 **Road Sign Review** Application

a) ***Opening the application***. Open the **Road Sign Review** application that you created in Tutorial 29 and continued to develop in Tutorial 30.

b) ***Copying the `SignImages` directory to your project directory.*** Copy the `C:\Examples\Tutorial31\Exercises\Images\SignImages` directory to the `RoadSignReview` directory.

c) ***Using `System.Data.OleDb` in `RoadSigns.aspx.cs`.*** Use the `System.Data.OleDb` namespace in `RoadSigns.aspx.cs`.

d) ***Rearranging and commenting the control declarations.*** Organize the control declarations as they appear on the Web Form. Add a comment above each declaration.

e) ***Defining the `Page_Load` event handler for the `RoadSigns.aspx` page.*** Use the `Open` method to open the connection to the database. Create a data reader to read the information specified by the data command object.

f) ***Populating the `Table` with sign images in the `RoadSigns.aspx` page.*** Add a `while` statement to `RoadSigns.aspx`'s `Page_Load` method. This loop should display an image of the sign and display the sign name in the `ToolTip` property. This property specifies the text that displays in a tooltip box when the mouse hovers over the `Image`. The sign image and name should be retrieved using the data reader. To display an `Image` in a cell of the `Table`, you need to create an `Image` control, specify a cell and use the cell's `Controls.Add` method to add an image to that cell. For example, to create an `Image` control programmatically, type `Image imgImageName = new Image()`. You then need to set the `ImageUrl` property to the location of the desired image. To display an `Image` control in the first cell of the first row, you would write the line `Table.Rows[ 0 ].Cells[ 0 ].Controls.Add( imgImageName )`. Also, if you wish to specify text for a tooltip, you must set the cell's `ToolTip` property—for example, `Table.Rows[ 0 ].Cells[ 0 ].ToolTip = "This is a tooltip"`.

g) ***Closing the reader and connection.*** Close the data reader and the connection to the database by invoking their `Close` methods.

h) ***Creating the Register `Button`'s `Click` event handler for `RoadSigns.aspx`.*** Double click the **Register** `Button` of `RoadSigns.aspx` to create the `Button`'s `Click` event handler.

i) ***Creating `Session` items.*** Create two `Session` items in the `Click` event handler, then set the first one equal to the user input for the **Name:** `TextBox`. The second `Session` item should equal the user input for the **Phone Number:** `TextBox`.

j) ***Redirecting to the `RoadTestRegistered.aspx` page.*** In the `Click` event handler, use the `Redirect.Response` method to redirect the client browser to the `RoadTestRegistered.aspx` page.

k) ***Rearranging and commenting the control declarations in `RoadTestRegistered.aspx.cs`.*** Organize the control declarations as they appear on the Web Form. Add a comment above each declaration.

l) ***Defining the `Page_Load` method of `RoadTestRegistered.aspx` page.*** Use the `Session` items to display a confirmation to the user about the user's registration information. Display the confirmation using the `lblConfirmation Label`. Display the user's name, then display text which states that the user will be contacted shortly at the phone number provided. This information should be displayed in a `Label`.

m) ***Running the application.*** Select **Debug > Start** to run your application. Type your name and phone number in the appropriate `TextBox`es, then click the **Register** `Button`. Confirm that the information is displayed correctly on the `RoadTestRegistered.aspx` page.

n) ***Closing the application.*** Close your running application by clicking the browser window's close box.

o) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 31.13 Solution
2  // RoadSigns.aspx.cs
3
4  using System;
5  using System.Collections;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Web;
10 using System.Web.SessionState;
11 using System.Web.UI;
12 using System.Web.UI.WebControls;
13 using System.Web.UI.HtmlControls;
14 using System.Data.OleDb;
15
16 namespace RoadSignReview
17 {
```

```
18    /// <summary>
19    /// Summary description for RoadSigns.
20    /// </summary>
21    public class RoadSigns : System.Web.UI.Page
22    {
23       // Label to display title
24       protected System.Web.UI.WebControls.Label lblRoadSigns;
25
26       // Table to display road signs
27       protected System.Web.UI.WebControls.Table tblRoadSigns;
28
29       // Label to display instructions
30       protected System.Web.UI.WebControls.Label lblRegister;
31
32       // Label and TextBox to input name
33       protected System.Web.UI.WebControls.Label lblName;
34       protected System.Web.UI.WebControls.TextBox txtName;
35
36       // Label and TextBox to input phone number
37       protected System.Web.UI.WebControls.Label lblPhoneNumber;
38       protected System.Web.UI.WebControls.TextBox txtPhoneNumber;
39
40       // Button to register for the driving test
41       protected System.Web.UI.WebControls.Button btnRegister;
42
43       // OleDbConnection to connect to road signs database
44       protected System.Data.OleDb.OleDbConnection objOleDbConnection;
45
46       // OleDbCommand to retrieve sign information from the database
47       protected System.Data.OleDb.OleDbCommand
48          objSelectSignInformation;
49
50       // invoked when Page is loaded
51       private void Page_Load( object sender, System.EventArgs e )
52       {
53          objOleDbConnection.Open();// open connection to the database
54
55          // declare reader to read from database
56          OleDbDataReader objReader;
57
58          // create reader to read from database
59          objReader = objSelectSignInformation.ExecuteReader();
60
61          int intRow = 0;
62          int intColumn = 0;
63
64          while ( objReader.Read() )
65          {
66             // move to next row
67             if ( intColumn > 4 )
68             {
69                intRow++;
70                intColumn = 0;
71             }
72
73             // create new Image control
74             System.Web.UI.WebControls.Image imgCellImage
75                = new System.Web.UI.WebControls.Image();
76
77             // set ImageUrl property
```

```
78              imgCellImage.ImageUrl = "SignImages/" +
79                 Convert.ToString( objReader[ "sign" ] );
80
81              // add image to table
82              tblRoadSigns.Rows[ intRow ].Cells[ intColumn ].
83                 Controls.Add( imgCellImage );
84
85              // add image name to ToolTip
86              tblRoadSigns.Rows[ intRow ].Cells[ intColumn ].ToolTip =
87                 Convert.ToString( objReader[ "name" ] );
88
89              intColumn++; // increment column location
90           }
91
92           objReader.Close(); // close the reader
93
94           // close the connection to the database
95           objOleDbConnection.Close();
96
97        } // end method Page_Load
98
99        // Web Form Designer generated code
100
101       // handles Register Button's Click event
102       private void btnRegister_Click(
103          object sender, System.EventArgs e )
104       {
105          // create Session variables
106          Session[ "strName" ] = txtName.Text;
107          Session[ "strPhoneNumber" ] = txtPhoneNumber.Text;
108
109          // redirect to another ASPX page
110          Response.Redirect( "RoadTestRegistered.aspx" );
111
112       } // end method btnRegister_Click
113
114    } // end class RoadSigns
115 }
```

```
1  // Exercise 31.13 Solution
2  // RoadTestRegistered.aspx.cs
3
4  using System;
5  using System.Collections;
6  using System.ComponentModel;
7  using System.Data;
8  using System.Drawing;
9  using System.Web;
10 using System.Web.SessionState;
11 using System.Web.UI;
12 using System.Web.UI.WebControls;
13 using System.Web.UI.HtmlControls;
14
15 namespace RoadSignReview
16 {
17    /// <summary>
18    /// Summary description for RoadTestRegistered.
19    /// </summary>
20    public class RoadTestRegistered : System.Web.UI.Page
```

```
21    {
22       // Label to display title
23       protected System.Web.UI.WebControls.Label lblRegistration;
24
25       // Label to display confirmation to user
26       protected System.Web.UI.WebControls.Label lblConfirmation;
27
28       // invoked when Page is loaded
29       private void Page_Load( object sender, System.EventArgs e )
30       {
31          // display output
32          lblConfirmation.Text =
33             Convert.ToString( Session[ "strName" ] ) +
34             ", you have registered to take your driving test. " +
35             "We will call you back at " +
36             Convert.ToString( Session[ "strPhoneNumber" ] ) +
37             " to confirm a date.";
38
39       } // end method Page_Load
40
41       // Web Form Designer generated code
42
43    } // end class RoadTestRegistered
44 }
```

32 TUTORIAL

Enhanced Car Payment Calculator Application

Introducing Exception Handling

Solutions

Instructor's Manual Exercise Solutions Tutorial 32

MULTIPLE-CHOICE QUESTIONS

32.1 Dealing with exceptional situations as an application executes is called _____.

a) exception detection
b) exception handling
c) exception resolution
d) exception debugging

32.2 A(n) _____ is always followed by at least one `catch` block or a `finally` block.

a) `if` statement
b) event handler
c) `try` block
d) None of the above.

32.3 The method call `Int32.Parse( "123.4a" )` will throw a(n) _____.

a) `FormatException`
b) `ParsingException`
c) `DivideByZeroException`
d) None of the above.

32.4 If no exceptions are thrown in a `try` block, _____.

a) the `catch` block(s) are skipped
b) all `catch` blocks are executed
c) an error occurs
d) the default exception is thrown

32.5 A(n) _____ is an exception that does not have an exception handler, and therefore might cause the application to terminate execution.

a) uncaught block
b) uncaught exception
c) error handler
d) thrower

32.6 A `try` block can have _____ associated with it.

a) only one `catch` block
b) several `finally` blocks
c) one or more `catch` blocks
d) None of the above.

32.7 The _____ statement is used to rethrow an exception from inside a `catch` block.

a) `rethrow`
b) `throw`
c) `try`
d) `catch`

32.8 The exception you wish to handle should be declared as a parameter of the _____ block.

a) `try`
b) `catch`
c) `finally`
d) None of the above.

32.9 A `finally` block is located _____.

a) after the `try` block, but before each `catch` block
b) before the `try` block
c) after the `try` block and the `try` block's corresponding `catch` blocks
d) Either b or c.

32.10 A _____ is executed if an exception is thrown from a `try` block or if no exception is thrown.

a) `catch` block
b) `finally` block
c) exception handler
d) All of the above.

Answers: 32.1) b. 32.2) c. 32.3) a. 32.4) a. 32.5) b. 32.6) c. 32.7) b. 32.8) b. 32.9) c. 32.10) b.

EXERCISES

32.11 (***Enhanced Miles Per Gallon Application***) Modify the **Miles Per Gallon** application (Exercise 13.13) to use exception handling to process the `FormatExceptions` that occur when converting the `strings` in the `TextBoxes` to `doubles` (Fig. 32.15). The original application allowed the user to input the number of miles driven and the number of gallons used for

a tank of gas to determine the number of miles the user was able to drive on one gallon of gas.

Figure 32.15 Enhanced **Miles Per Gallon** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial32\Exercises\EnhancedMilesPerGallon` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `MilesPerGallon.sln` in the `EnhancedMilesPerGallon` directory to open the application.

c) ***Adding a `try` block.*** Find the `btnCalculateMPG_Click` event handler. Enclose all of the code in this event handler in a `try` block.

d) ***Adding a `catch` block.*** After the `try` block you added in *Step c*, add a `catch` block to handle any `FormatExceptions` that may occur in the `try` block. Inside the `catch` block, add code to display an error message dialog.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter invalid data as shown in Fig. 32.15 and click the **Calculate MPG** `Button`. A `MessageBox` should appear asking you to enter valid input. Enter valid input and click the **Calculate MPG** `Button` again. Verify that the correct output is displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 32.11 Solution
2  // MilesPerGallon.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace MilesPerGallon
12 {
13    /// <summary>
14    /// Summary description for FrmMilesPerGallon.
15    /// </summary>
16    public class FrmMilesPerGallon : System.Windows.Forms.Form
17    {
18       // Label and TextBox for miles driven information
19       private System.Windows.Forms.Label lblMilesDriven;
20       private System.Windows.Forms.TextBox txtMilesDriven;
```

```
21
22          // Label and TextBox for gallons used information
23          private System.Windows.Forms.Label lblGallonsUsed;
24          private System.Windows.Forms.TextBox txtGallonsUsed;
25
26          // Label and output Label for miles per gallon calculation
27          private System.Windows.Forms.Label lblMilesPerGallon;
28          private System.Windows.Forms.Label lblOutputValue;
29
30          // Button to calculate miles per gallon
31          private System.Windows.Forms.Button btnCalculateMPG;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          public FrmMilesPerGallon()
39          {
40             //
41             // Required for Windows Form Designer support
42             //
43             InitializeComponent();
44
45             //
46             // TODO: Add any constructor code after InitializeComponent
47             // call
48             //
49          }
50
51          /// <summary>
52          /// Clean up any resources being used.
53          /// </summary>
54          protected override void Dispose( bool disposing )
55          {
56             if( disposing )
57             {
58                if (components != null)
59                {
60                   components.Dispose();
61                }
62             }
63             base.Dispose( disposing );
64          }
65
66          // Windows Form Designer generated code
67
68          /// <summary>
69          /// The main entry point for the application.
70          /// </summary>
71          [STAThread]
72          static void Main()
73          {
74             Application.Run( new FrmMilesPerGallon() );
75          }
76
77          // calculate and return miles per gallon
78          private double MilesPerGallon(
79             double dblMilesDriven, double dblGallonsUsed )
80          {
```

```
81             return dblMilesDriven / dblGallonsUsed;
82
83          } // end method MilesPerGallon
84
85          // handles Calculate Button's Click event
86          private void btnCalculateMPG_Click(
87             object sender, System.EventArgs e )
88          {
89             // retrieve user input
90             try
91             {
92                // display miles per gallon
93                lblOutputValue.Text = String.Format( "{0:F}",
94                   MilesPerGallon(
95                   Double.Parse( txtMilesDriven.Text ),
96                   Double.Parse( txtGallonsUsed.Text ) ) );
97             }
98
99             // prompt for input in correct format
100            catch ( FormatException formatExceptionParameter )
101            {
102               MessageBox.Show(
103                  "Please enter decimal numbers for " +
104                  "the miles driven and gallons.",
105                  "Invalid Number Format", MessageBoxButtons.OK,
106                  MessageBoxIcon.Error );
107            }
108
109         } // end method btnCalculateMPG_Click
110
111      } // end class FrmMilesPerGallon
112   }
```

32.12 (***Enhanced Prime Numbers Application***) Modify the **Prime Numbers** application (Exercise 13.17) to use exception handling to process the `FormatExceptions` that occur when converting the `strings` in the `TextBoxes` to `ints` (Fig. 32.16). The original application took two numbers (representing a lower bound and an upper bound) and determined all of the prime numbers within the specified bounds, inclusive. An `int` greater than 1 is said to be prime if it is divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime numbers, but 4, 6, 8 and 9 are not.

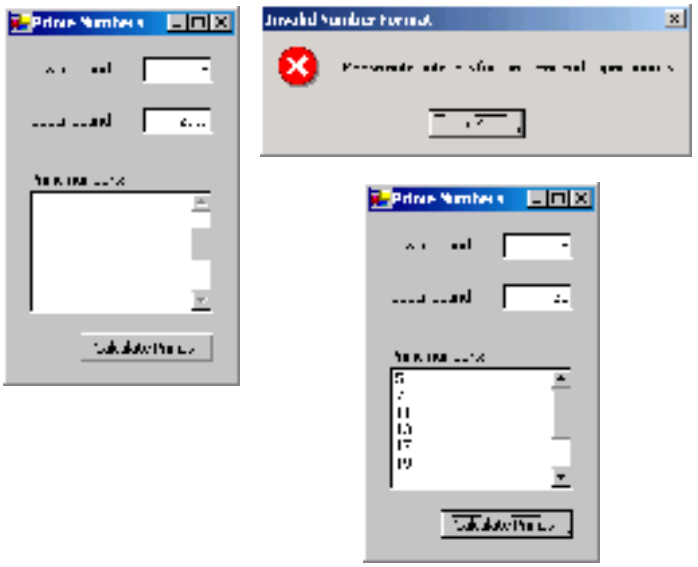


Figure 32.16 Enhanced **Prime Numbers** application's GUI.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial32\Exercises\EnhancedPrimeNumbers` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `PrimeNumbers.sln` in the `EnhancedPrimeNumbers` directory to open the application.

c) ***Adding a `try` block***. Find the `btnCalculatePrimes_Click` event handler. Enclose all the code following the variable declarations in a `try` block.

d) ***Adding a `catch` block***. Add a `catch` block that catches any `FormatExceptions` that may occur in the `try` block you added to `btnCalculatePrimes_Click` in *Step c*. Inside the `catch` block, add code to display an error message dialog.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter invalid data as shown in Fig. 32.16 and click the **Calculate Primes** `Button`. A `MessageBox` should appear asking you to enter valid input. Enter valid input and click the **Calculate Primes** `Button` again. Verify that the correct output is displayed.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
1  // Exercise 32.12 Solution
2  // PrimeNumbers.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace PrimeNumbers
12 {
13    /// <summary>
14    /// Summary description for FrmPrimeNumbers.
15    /// </summary>
16    public class FrmPrimeNumbers : System.Windows.Forms.Form
17    {
18       // Label and TextBox for lower bound
19       private System.Windows.Forms.Label lblLowerBound;
20       private System.Windows.Forms.TextBox txtLowerBound;
```

```
21
22          // Label and TextBox for upper bound
23          private System.Windows.Forms.Label lblUpperBound;
24          private System.Windows.Forms.TextBox txtUpperBound;
25
26          // Label and TextBox for outputting prime numbers
27          private System.Windows.Forms.Label lblPrimeNumbers;
28          private System.Windows.Forms.TextBox txtPrimeNumbers;
29
30          // Button to trigger calculation
31          private System.Windows.Forms.Button btnCalculatePrimes;
32
33          /// <summary>
34          /// Required designer variable.
35          /// </summary>
36          private System.ComponentModel.Container components = null;
37
38          public FrmPrimeNumbers()
39          {
40             //
41             // Required for Windows Form Designer support
42             //
43             InitializeComponent();
44
45             //
46             // TODO: Add any constructor code after InitializeComponent
47             // call
48             //
49          }
50
51          /// <summary>
52          /// Clean up any resources being used.
53          /// </summary>
54          protected override void Dispose( bool disposing )
55          {
56             if( disposing )
57             {
58                if (components != null)
59                {
60                   components.Dispose();
61                }
62             }
63             base.Dispose( disposing );
64          }
65
66          // Windows Form Designer generated code
67
68          /// <summary>
69          /// The main entry point for the application.
70          /// </summary>
71          [STAThread]
72          static void Main()
73          {
74             Application.Run( new FrmPrimeNumbers() );
75          }
76
77          // determine if number is prime
78          private bool Prime( int intNumber )
79          {
80             int intCount; // declare counter
```

```
81
82          // set square root of intNumber as limit
83          int intLimit = ( int ) Math.Sqrt( intNumber );
84
85          // loop until intCount reaches square root of intNumber
86          for ( intCount = 2; intCount <= intLimit; intCount++ )
87          {
88             if ( intNumber % intCount == 0 )
89             {
90                return false; // number is not prime
91             }
92          }
93
94          return true; // number is prime
95
96       } // end method Prime
97
98       // handles Calculate Primes Button's Click event
99       private void btnCalculatePrimes_Click(
100         object sender, System.EventArgs e )
101      {
102         // declare variables
103         int intLowerBound;
104         int intUpperBound;
105         int intCounter;
106         string strOutput = "";
107
108         // attempt to retrieve input from user
109         try
110         {
111            intLowerBound = Int32.Parse( txtLowerBound.Text );
112            intUpperBound = Int32.Parse( txtUpperBound.Text );
113
114            if ( intLowerBound <= 0 || intUpperBound <= 0 )
115            {
116               MessageBox.Show( "Bounds must be greater than 0",
117                  "Invalid Bounds", MessageBoxButtons.OK,
118                  MessageBoxIcon.Exclamation );
119            }
120            else if ( intUpperBound < intLowerBound )
121            {
122               MessageBox.Show( "Upper bound cannot be less than " +
123                  "lower bound", "Invalid Bounds",
124                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation );
125            }
126            else
127            {
128               // loop from lower bound to upper bound
129               for ( intCounter = intLowerBound;
130                  intCounter <= intUpperBound; intCounter++ )
131               {
132                  // if prime number, display in TextBox
133                  if ( Prime( intCounter ) == true )
134                  {
135                     strOutput += ( intCounter + "\r\n" );
136                  }
137               }
138            }
139
140         } // end try block
```

```
141
142          // display a message if the user did not
143          // input integer values
144          catch ( FormatException formatExceptionParameter )
145          {
146             MessageBox.Show(
147                "Please enter integers for the lower and upper " +
148                "bounds.", "Invalid Number Format",
149                MessageBoxButtons.OK, MessageBoxIcon.Error )
150
151          } // end catch block
152
153          txtPrimeNumbers.Text = strOutput;
154
155       } // end method btnCalculatePrimes_Click
156
157    } // end class FrmPrimeNumbers
158 }
```

32.13 (***Enhanced Letterhead Designer Application***) Modify the **Letterhead** application (Exercise 26.13) to use exception handling to process the `FileNotFoundException` that may occur when the user specifies the image that will serve as the letterhead (Fig. 32.17). We will define what a `FileNotFoundException` is shortly. The application should still allow users to design stationery for company documents.

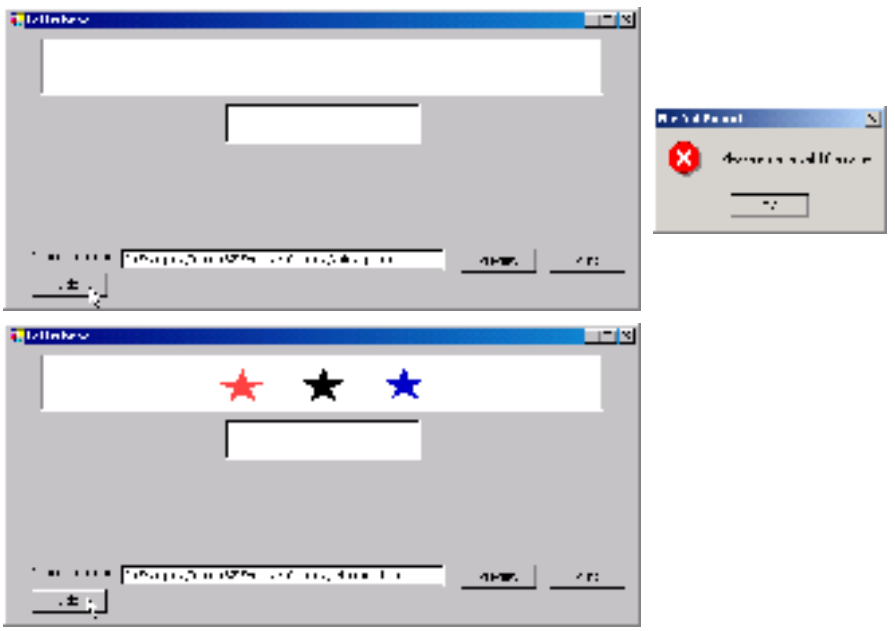


Figure 32.17 Enhanced **Letterhead** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial32\Exercises\EnhancedLetterhead` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `Letterhead.sln` in the `EnhancedLetterhead` directory to open the application.

c) ***Adding a `try` block to the `btnAdd_Click` event handler.*** Find the `btnAdd_Click` event handler. Enclose the body of `btnAdd_Click` in a `try` block.

d) ***Adding a `catch` block to the `btnAdd_Click` event handler.*** Add a `catch` block that catches any `FileNotFoundExceptions` that may occur in the `try` block that you added in *Step c*. A **`FileNotFoundException`** is thrown when you attempt to access a file that does not exist. Inside the `catch` block, add code to display an error message dialog.

e) ***Running the application.*** Select **Debug > Start** to run your application. Enter an incorrect path for your image, and ensure that a `MessageBox` appears indicating the invalid input.

f) ***Closing the application.*** Close your running application by clicking its close box.

g) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answer:

```
// Exercise 32.13 Solution
// Letterhead.cs (Enhanced)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Drawing.Printing;
using System.IO;

namespace Letterhead
{
   /// <summary>
   /// Summary description for FrmLetterHead.
   /// </summary>
   public class FrmLetterHead : System.Windows.Forms.Form
   {
      // PictureBox to display the letterhead
      private System.Windows.Forms.PictureBox picImage;

      // TextBox to input letterhead text
      private System.Windows.Forms.TextBox txtInformation;

      // Label and TextBox to input image location
      private System.Windows.Forms.Label lblImage;
      private System.Windows.Forms.TextBox txtImage;

      // Button to add the image
      private System.Windows.Forms.Button btnAdd;

      // Button to preview the letterhead
      private System.Windows.Forms.Button btnPreview;

      // Button to print the letterhead
      private System.Windows.Forms.Button btnPrint;

      // PrintPreviewDialog to preview and print the letterhead
      private System.Windows.Forms.PrintPreviewDialog objPreview;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      // create font object
      Font m_objFont;

      public FrmLetterHead()
      {
         //
```

```
53          // Required for Windows Form Designer support
54          //
55          InitializeComponent();
56
57          //
58          // TODO: Add any constructor code after InitializeComponent
59          // call
60          //
61       }
62
63       /// <summary>
64       /// Clean up any resources being used.
65       /// </summary>
66       protected override void Dispose( bool disposing )
67       {
68          if( disposing )
69          {
70             if (components != null)
71             {
72                components.Dispose();
73             }
74          }
75          base.Dispose( disposing );
76       }
77
78       // Windows Form Designer generated code
79
80       /// <summary>
81       /// The main entry point for the application.
82       /// </summary>
83       [STAThread]
84       static void Main()
85       {
86          Application.Run( new FrmLetterHead() );
87       }
88
89       // PrintPage event raised for each page to be printed.
90       private void objPrintDocument_PrintPage(
91          object sender, PrintPageEventArgs e )
92       {
93          float fltYPosition;
94          float fltXPosition;
95          float fltLeftMargin = e.MarginBounds.Left;
96          float fltTopMargin = e.MarginBounds.Top;
97
98          string strPath;
99
100         // get location of image
101         strPath = txtImage.Text;
102
103         // make sure image location was provided
104         if ( strPath != "" )
105         {
106            Image objImage;
107            objImage = Image.FromFile( strPath );
108
109            // print image so it is on top of page
110            e.Graphics.DrawImage( Image.FromFile( strPath ),
111               fltLeftMargin + picImage.Location.X,
112               fltTopMargin + picImage.Location.Y,
```

```
113                picImage.Size.Width, picImage.Size.Height );
114          }
115
116          // if contact information is provided, print data
117          if ( txtInformation.Text != "" )
118          {
119             // specifies font of text
120             m_objFont = new Font( "Tahoma", 12, FontStyle.Bold );
121
122             fltXPosition = fltLeftMargin + txtInformation.Location.X;
123             fltYPosition = fltTopMargin + txtInformation.Location.Y;
124
125             // print information
126             e.Graphics.DrawString( txtInformation.Text, m_objFont,
127                Brushes.Black, fltXPosition, fltYPosition );
128          }
129
130          // indicate there are no more pages to print
131          e.HasMorePages = false;
132
133       } // end method objPrintDocument_PrintPage
134
135       // print the document
136       private void btnPrint_Click(
137          object sender, System.EventArgs e )
138       {
139          // create new object to assist in printing
140          PrintDocument objPrintDocument = new PrintDocument();
141
142          // add PrintPage event handler
143          objPrintDocument.PrintPage += new PrintPageEventHandler(
144             objPrintDocument_PrintPage );
145
146          // print the document
147          objPrintDocument.Print();
148
149       } // end method btnPrint_Click
150
151       // display document in print preview dialog
152       private void btnPreview_Click(
153          object sender, System.EventArgs e )
154       {
155          // create new object to assist in previewing
156          PrintDocument objPrintDocument = new PrintDocument();
157
158          // add PrintPage event handler
159          objPrintDocument.PrintPage += new PrintPageEventHandler(
160             objPrintDocument_PrintPage );
161
162          objPreview.Document = objPrintDocument; // specify document
163          objPreview.ShowDialog(); // show print preview
164
165       } // end method btnPreview_Click
166
167       // add the specified image to the Image control
168       private void btnAdd_Click(
169          object sender, System.EventArgs e )
170       {
171          // try to import the image specified
172          try
```

```
173          {
174             if ( txtImage.Text != "" )
175             {
176                picImage.Image = Image.FromFile( txtImage.Text );
177             }
178             else
179             {
180                // display an error if the user did not enter anything
181                MessageBox.Show(
182                   "You must enter a path for your image.",
183                   "Input Error", MessageBoxButtons.OK,
184                   MessageBoxIcon.Information );
185             }
186          }
187
188          // handle case where the image could not be found
189          catch (
190             FileNotFoundException fileNotFoundExceptionParameter )
191          {
192             MessageBox.Show( "Please enter a valid file name.",
193                "File Not Found", MessageBoxButtons.OK,
194                MessageBoxIcon.Error );
195          }
196
197       } // end method btnAdd_Click
198
199    } // end class FrmLetterHead
200 }
```

What does this code do? ▶

32.14 What does the following code do, assuming that `dblValue1` and `dblValue2` are both declared as `doubles`?

```
1  try
2  {
3     dblValue1 = Double.Parse( txtInput1.Text );
4     dblValue2 = Double.Parse( txtInput2.Text );
5
6     txtOutput.Text = Convert.ToString( dblValue1 * dblValue2 );
7  }
8
9  catch ( FormatException formatExceptionParameter )
10 {
11    MessageBox.Show(
12       "Please enter decimal values.",
13       "Invalid Number Format",
14       MessageBoxButtons.OK, MessageBoxIcon.Error );
15 }
```

Answer: This code multiplies two `doubles` if both inputs in `txtInput1` and `txtInput2` can be converted to type `double` (that is, they are decimal or integer values). Otherwise it displays an error message dialog that informs the user to enter decimal values in the `TextBoxes`. The complete code reads:

```
1  // Exercise 32.14 Solution
2  // Multiplier.cs
3
4  using System;
```

```
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Multiplier
12 {
13    /// <summary>
14    /// Summary description for FrmMultiplier.
15    /// </summary>
16    public class FrmMultiplier : System.Windows.Forms.Form
17    {
18       // Label and TextBox for first value
19       private System.Windows.Forms.Label lblValue1;
20       private System.Windows.Forms.TextBox txtInput1;
21
22       // Label and TextBox for second value
23       private System.Windows.Forms.Label lblValue2;
24       private System.Windows.Forms.TextBox txtInput2;
25
26       // Label and TextBox for output
27       private System.Windows.Forms.Label lblProduct;
28       private System.Windows.Forms.TextBox txtOutput;
29
30       // Button to perform multiplication
31       private System.Windows.Forms.Button btnMultiply;
32
33       /// <summary>
34       /// Required designer variable.
35       /// </summary>
36       private System.ComponentModel.Container components = null;
37
38       public FrmMultiplier()
39       {
40          //
41          // Required for Windows Form Designer support
42          //
43          InitializeComponent();
44
45          //
46          // TODO: Add any constructor code after InitializeComponent
47          // call
48          //
49       }
50
51       /// <summary>
52       /// Clean up any resources being used.
53       /// </summary>
54       protected override void Dispose( bool disposing )
55       {
56          if( disposing )
57          {
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
```

```
65
66          // Windows Form Designer generated code
67
68          /// <summary>
69          /// The main entry point for the application.
70          /// </summary>
71          [STAThread]
72          static void Main()
73          {
74             Application.Run( new FrmMultiplier() );
75          }
76
77          // handles btnMultiply's Click event
78          private void btnMultiply_Click(
79             object sender, System.EventArgs e )
80          {
81             double dblValue1;
82             double dblValue2;
83
84             try
85             {
86                dblValue1 = Double.Parse( txtInput1.Text );
87                dblValue2 = Double.Parse( txtInput2.Text );
88
89                txtOutput.Text =
90                   Convert.ToString( dblValue1 * dblValue2 );
91             }
92
93             catch ( FormatException formatExceptionParameter )
94             {
95                MessageBox.Show(
96                   "Please enter decimal values.",
97                   "Invalid Number Format",
98                   MessageBoxButtons.OK, MessageBoxIcon.Error );
99             }
100
101         } // end method btnMultiply_Click
102
103      } // end class FrmMultiply
104   }
```

What's wrong with this code? ▶

32.15 The following code should add integers from two `TextBoxes` and display the result in `txtResult`. Assume that `intValue1` and `intValue2` are declared as `ints`. Find the error(s) in the following code:

```
1  try
2  {
3     intValue1 = Int32.Parse( txtInput1.Text );
```

```
   intValue2 = Int32.Parse( txtInput2.Text );

   txtOutput.Text = Convert.ToString( intValue1 + intValue2 );

   catch ( )
   {
      MessageBox.Show(
         "Please enter valid ints.",
         "Invalid Number Format",
         MessageBoxButtons.OK, MessageBoxIcon.Error );
   }
}
```

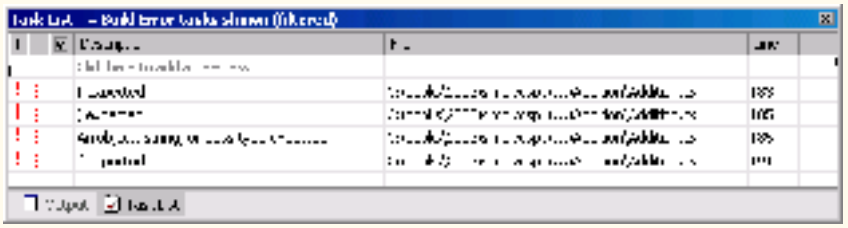
Answer: There are two errors in this code. First, the `catch` block appears inside the `try` block. All `catch` blocks must follow immediately after the `try` block. Second, you will need to specify which type of exception is being caught between the parentheses after the `catch` statement (in this case, a `FormatException`). The complete incorrect code reads:

```
// Exercise 32.15 Solution
// Addition.cs (Incorrect)

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Addition
{
   /// <summary>
   /// Summary description for FrmAddition.
   /// </summary>
   public class FrmAddition : System.Windows.Forms.Form
   {
      // Label and TextBox for first value
      private System.Windows.Forms.Label lblValue1;
      private System.Windows.Forms.TextBox txtInput1;

      // Label and TextBox for second value
      private System.Windows.Forms.Label lblValue2;
      private System.Windows.Forms.TextBox txtInput2;

      // Label and TextBox for output
      private System.Windows.Forms.Label lblProduct;
      private System.Windows.Forms.TextBox txtOutput;

      // Button to perform addition
      private System.Windows.Forms.Button btnAdd;

      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public FrmAddition()
      {
         //
         // Required for Windows Form Designer support
         //
```

```
43          InitializeComponent();
44
45          //
46          // TODO: Add any constructor code after InitializeComponent
47          // call
48          //
49       }
50
51       /// <summary>
52       /// Clean up any resources being used.
53       /// </summary>
54       protected override void Dispose( bool disposing )
55       {
56          if( disposing )
57          {
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmAddition() );
75       }
76
77       // handles btnAdd's Click event
78       private void btnAdd_Click(
79          object sender, System.EventArgs e )
80       {
81          int intValue1;
82          int intValue2;
83
84          try
85          {
86             intValue1 = Convert.ToInt32( txtInput1.Text );
87             intValue2 = Convert.ToInt32( txtInput2.Text );
88
89             txtOutput.Text =
90                Convert.ToString( intValue1 + intValue2 );
91
92          catch ( )
93          {
94             MessageBox.Show(
95                "Please enter valid integers.",
96                "Invalid Number Format",
97                MessageBoxButtons.OK, MessageBoxIcon.Error );
98          }
99
100         }
101
102      } // end method btnAdd_Click
```

`catch` block should not be inside `try` block, specify exception type in `catch` statement

```
103
104    } // end class FrmAddition
105 }
```

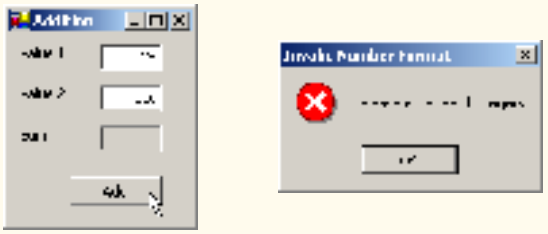


Answer: The complete corrected code should read:

```
1  // Exercise 32.15 Solution
2  // Addition.cs (Correct)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace Addition
12 {
13    /// <summary>
14    /// Summary description for FrmAddition.
15    /// </summary>
16    public class FrmAddition : System.Windows.Forms.Form
17    {
18       // Label and TextBox for first value
19       private System.Windows.Forms.Label lblValue1;
20       private System.Windows.Forms.TextBox txtInput1;
21
22       // Label and TextBox for second value
23       private System.Windows.Forms.Label lblValue2;
24       private System.Windows.Forms.TextBox txtInput2;
25
26       // Label and TextBox for output
27       private System.Windows.Forms.Label lblProduct;
28       private System.Windows.Forms.TextBox txtOutput;
29
30       // Button to perform addition
31       private System.Windows.Forms.Button btnAdd;
32
33       /// <summary>
34       /// Required designer variable.
35       /// </summary>
36       private System.ComponentModel.Container components = null;
37
38       public FrmAddition()
39       {
40          //
41          // Required for Windows Form Designer support
42          //
43          InitializeComponent();
44
```

```
45          //
46          // TODO: Add any constructor code after InitializeComponent
47          // call
48          //
49       }
50
51       /// <summary>
52       /// Clean up any resources being used.
53       /// </summary>
54       protected override void Dispose( bool disposing )
55       {
56          if( disposing )
57          {
58             if (components != null)
59             {
60                components.Dispose();
61             }
62          }
63          base.Dispose( disposing );
64       }
65
66       // Windows Form Designer generated code
67
68       /// <summary>
69       /// The main entry point for the application.
70       /// </summary>
71       [STAThread]
72       static void Main()
73       {
74          Application.Run( new FrmAddition() );
75       }
76
77       // handles btnAdd's Click event
78       private void btnAdd_Click(
79          object sender, System.EventArgs e )
80       {
81          int intValue1;
82          int intValue2;
83
84          try
85          {
86             intValue1 = Int32.Parse( txtInput1.Text );
87             intValue2 = Int32.Parse( txtInput2.Text );
88
89             txtOutput.Text =
90                Convert.ToString( intValue1 + intValue2 );
91          }
92
93          catch ( FormatException formatExceptionParameter )
94          {
95             MessageBox.Show(
96                "Please enter valid integers.",
97                "Invalid Number Format",
98                MessageBoxButtons.OK, MessageBoxIcon.Error );
99
100         }
101
102      } // end method btnAdd_Click
103
104   } // end class FrmAddition
```

```
105 }
```



Programming Challenge ▶

32.16 (*Enhanced Vending Machine Application*) The **Vending Machine** application from Tutorial 3 has been modified to use exception handling to process the `IndexOutOfRangeExceptions` that occur when selecting items out of the range `0` through `7` (Fig. 32.18). This type of exception will be defined shortly. To get a snack, the user must type the number of the desired snack in the `TextBox`, then press the **Dispense Snack**: `Button`. The name of the snack is displayed in the output `Label`.

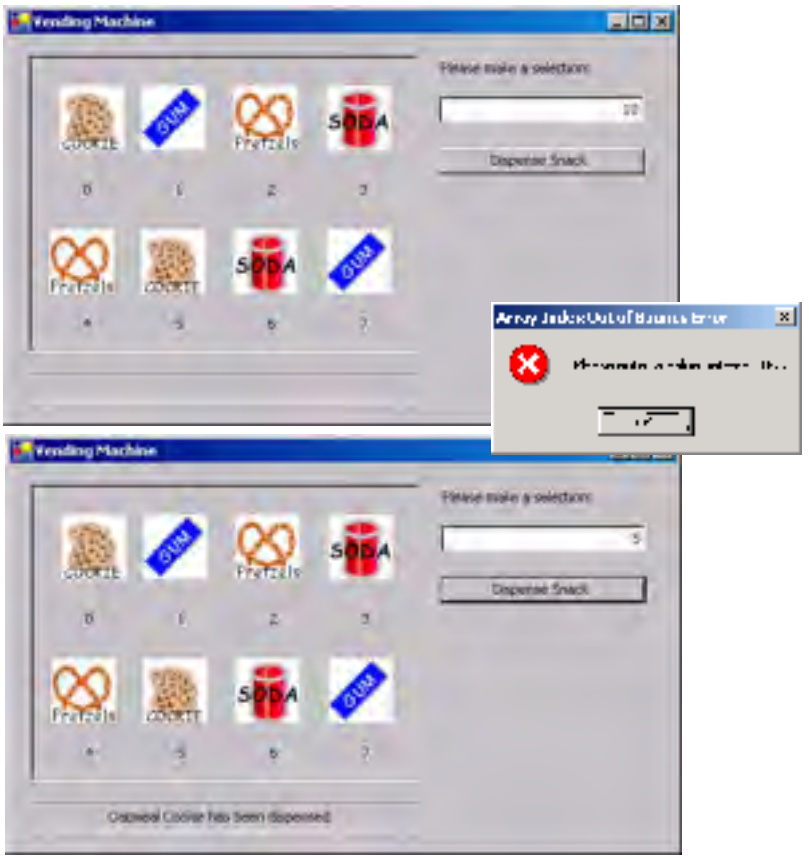


Figure 32.18 **Vending Machine** application.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\Tutorial32\Exercises\EnhancedVendingMachine` to your `C:\SimplyCSP` directory.

b) ***Opening the application's template file.*** Double click `VendingMachine.sln` in the `EnhancedVendingMachine` directory to open the application.

c) ***Adding a `try` block.*** Find the `btnDispense_Click` event handler. Enclose all of the code in the event handler in a `try` block.

d) ***Adding a `catch` block***. Add a `catch` block that catches any `FormatExceptions` that may occur in the `try` block that you added to `btnDispense_Click` in *Step c*. Inside the `catch` block, add code to display an error message dialog.

e) ***Adding a second `catch` block***. Immediately following the `catch` block you added in *Step c*, add a second `catch` block to catch any `IndexOutOfRangeExceptions` that may occur. An **`IndexOutOfRangeException`** occurs when the application attempts to access an array with an invalid index. Inside the `catch` block, add code to display an error message dialog.

f) ***Running the application.*** Select **Debug > Start** to run your application. Make an out of range selection (for instance, `32`) and click the **Dispense Snack** `Button`. Verify that the proper `MessageBox` is displayed for the invalid input. Enter letters for a selection and click the **Dispense Snack** `Button`. Verify that the proper `MessageBox` is displayed for the invalid input.

g) ***Closing the application.*** Close your running application by clicking its close box.

h) ***Closing the IDE.*** Close Visual Studio .NET by clicking its close box.

Answers:

```
1  // Exercise 32.16 Solution
2  // VendingMachine.cs (Enhanced)
3
4  using System;
5  using System.Drawing;
6  using System.Collections;
7  using System.ComponentModel;
8  using System.Windows.Forms;
9  using System.Data;
10
11 namespace VendingMachine
12 {
13    /// <summary>
14    /// Summary description for FrmVendingMachine.
15    /// </summary>
16    public class FrmVendingMachine : System.Windows.Forms.Form
17    {
18       // Panel enclosing snack choices
19       private System.Windows.Forms.Panel windowPanel;
20
21       // Labels and PictureBoxes for each snack
22       private System.Windows.Forms.Label lbl0;
23       private System.Windows.Forms.Label lbl1;
24       private System.Windows.Forms.Label lbl2;
25       private System.Windows.Forms.Label lbl3;
26       private System.Windows.Forms.Label lbl4;
27       private System.Windows.Forms.Label lbl5;
28       private System.Windows.Forms.Label lbl6;
29       private System.Windows.Forms.Label lbl7;
30       private System.Windows.Forms.PictureBox picItem0;
31       private System.Windows.Forms.PictureBox picItem1;
32       private System.Windows.Forms.PictureBox picItem2;
33       private System.Windows.Forms.PictureBox picItem3;
34       private System.Windows.Forms.PictureBox picItem4;
35       private System.Windows.Forms.PictureBox picItem5;
36       private System.Windows.Forms.PictureBox picItem6;
37       private System.Windows.Forms.PictureBox picItem7;
38
39       // Label for the output
40       private System.Windows.Forms.Label lblOutput;
41
42       // Label and TextBox for the user's selection
```

```
43          private System.Windows.Forms.Label lblSelection;
44          private System.Windows.Forms.TextBox txtSelection;
45
46          // Button to dispense the snack
47          private System.Windows.Forms.Button btnDispense;
48
49          /// <summary>
50          /// Required designer variable.
51          /// </summary>
52          private System.ComponentModel.Container components = null;
53
54          // declare an array of snack names
55          string[] strSnacks = {
56             "Chocolate Chip Cookie", "Bubble Gum",
57             "Plain Pretzel", "Soda",
58             "Salted Pretzel", "Oatmeal Cookie",
59             "Diet Soda", "Sugar-free Gum" };
60
61          public FrmVendingMachine()
62          {
63             //
64             // Required for Windows Form Designer support
65             //
66             InitializeComponent();
67
68             //
69             // TODO: Add any constructor code after InitializeComponent
70             // call
71             //
72          }
73
74          /// <summary>
75          /// Clean up any resources being used.
76          /// </summary>
77          protected override void Dispose( bool disposing )
78          {
79             if( disposing )
80             {
81                if (components != null)
82                {
83                   components.Dispose();
84                }
85             }
86             base.Dispose( disposing );
87          }
88
89          // Windows Form Designer generated code
90
91          /// <summary>
92          /// The main entry point for the application.
93          /// </summary>
94          [STAThread]
95          static void Main()
96          {
97             Application.Run( new FrmVendingMachine() );
98          }
99
100         // method to dispense snack
101         private void btnDispense_Click(
102            object sender, System.EventArgs e )
```

```
103       {
104          // try to get user input
105          try
106          {
107             // get user input
108             int intSelection = Int32.Parse( txtSelection.Text );
109
110             lblOutput.Text =
111                Convert.ToString( strSnacks[ intSelection ] ) +
112                " has been dispensed";
113          }
114
115          // handle case when user enters invalid input
116          catch ( FormatException formatExceptionParameter )
117          {
118             MessageBox.Show(
119                "Please enter an integer value.",
120                "Number Format Exception",
121                MessageBoxButtons.OK, MessageBoxIcon.Error );
122          }
123
124          // handle case when user inputs number not in array bounds
125          catch ( IndexOutOfRangeException indexExceptionParameter )
126          {
127             MessageBox.Show(
128                "Please enter a value between 0-7.",
129                "Array Index Out of Bounds Error",
130                MessageBoxButtons.OK, MessageBoxIcon.Error );
131          }
132
133       } // end method btnDispense_Click
134
135    } // end class FrmVendingMachine
136 }
```