Java InstantCode: Developing Applications Using Jabber

SkillSoft Press © 2004

This code-rich reference includes many applications, such as instant technical support, airline reservation, group chatting, contact list, and chat room applications.

Table of Contents

Introduction
Copyright
Chapter 1 - Introduction
Chapter 2 - Creating a Connector Application
Chapter 3 - Creating a Chat Room Application
Chapter 4 - Creating a Group Chatting Application
Chapter 5 - Creating an Instant Technical Support Application
Chapter 6 - Creating an Airline Reservation Application
Chapter 7 - Creating a Contact List Application
Index
List of Figures
List of Listings
CD Content

Introduction

About InstantCode Books

The InstantCode series is designed to provide you - the developer - with code you can use for common tasks in the workplace. The goal of the InstantCode series is not to provide comprehensive information on specific technologies - this is typically well-covered in other books. Instead, the purpose of this series is to provide actual code listings that you can immediately put to use in building applications for your particular requirements.

How These Books are Structured

The underlying philosophy of the InstantCode series is to present code listings that you can download and apply to your own business needs. To support this, these books are divided into chapters, each covering an independent task.

Each chapter includes a brief description of the task, followed by an overview of the element of the book's subject technology that we will use to perform that task. Each section ends with a code listing: each of the individual code segments in the chapter is independently downloadable, as is the complete chapter code. You will be able to download source code files, as well as application files.

Who Should Read These Books

These books are written for software development professionals who have basic knowledge of the associated technology and want to develop customized technology solutions.

About the Book

Jabber is an open source set of streaming XML protocols used for messaging, such as Instant Messaging (IM), and remote system monitoring. Jabber allows real-time communication between two entities over the Internet via a proxy, known as the Jabber server. Jabber community uses Jabber protocol to develop IM based applications. Jabber Software Foundation (JSF) manages the Jabber protocol. Jabber Inc markets commercial solutions developed using Jabber. This book includes many application, which includes instant technical support, airline reservation, group chatting, contact list, and chat room applications.

About the Author

Anurag Sharma has 3 years of working experience in the field of Software Engineering. He has authored various books on JCA and JSAPI.

Credits

I would like to thank Radhika Chauhan and Gaurav Bathla for helping me complete the book on time and providing continuous support and encouragement.

Copyright

Java InstantCode: Developing Applications Using Jabber

Published by SkillSoft Corporation
20 Industrial Park Drive
Nashua, NH 03062
(603) 324-3000

information@skillsoft.com

Chapter 1: Introduction

Jabber is an open source set of streaming XML protocols for messaging, such as Instant Messaging (IM) and remote system monitoring that allow real-time communication between two entities over the Internet via a proxy, known as the Jabber server.

Jabber community uses Jabber protocol to develop IM based applications. Jabber Software Foundation (JSF) manages the Jabber protocol. Jabber Inc markets commercial solutions developed using Jabber.

Features of Jabber

Jabber follows the client-server architecture and provides the following features:

- Open Protocols: Provides streaming XML protocols known as Extensible Messaging and Presence Protocol (XMPP). The open-source community started Jabber. You can implement the Jabber protocols and may use any code license.
- XML Data Format: Supports XML, which is an integral part of Jabber. This makes Jabber flexible to transfer structured data.
- Standard-based addressing: Allows you to include new IM end users and identify individual entities. A Jabber server represents each end user as a different entity.
- Distributed network: Provides a client-server architecture similar to that of an e-mail. Jabber utilizes a distributed network of servers by using a common protocol to interact with clients and manage IM.
- Exchange Messages: Allows end users to send and receive instant messages in real-time environment. It provides both one-to-one chat and group chat.
- Roster: Maintains a server-side contact list called roster. An end user can manage the roster by creating groups and assigning contacts to these groups. End users can also update, add, and delete contact information from a roster.
- Presence: Sets indicators to inform whether or not an end user is present in the chat session.

Jabber Architecture

Jabber IM system work on a client-server architecture, which is similar to that of an e-mail. In a Jabber IM system, you can develop and run a customized Jabber server. For communication, every end user connects to a personal server, which maintains and receives information for the end user. The Jabber server uses two ports, 5222 and 5269. The 5222 port is used for communication between client and server applications and the 5269 port is used for server-to-server communication. The protocol used for communication between Jabber client and Jabber server or Jabber server and Jabber server is Transmission Control Protocol (TCP). To transmit messages, the Jabber server uses the push technique instead of the pull technique. This means that client does not ask for new messages but the server sends the messages to the client as soon as the server receives them.

Jabber Server

Jabber servers are modular and can communicate with any other Jabber server. A Jabber server handles all client connections, communicates directly with the Jabber clients, and maintains co-ordination between different server components. Each Jabber server contains definite code packages, which can handle services such as maintaining contact lists, registration and authentication of end users, and storage of offline messages. The basic functioning of the Jabber servers can be extended by using external components to handle advanced capabilities, such as providing gateways to other IM systems.

Jabber Client

A Jabber client allows you to:

- Communicate with Jabber servers by using the TCP sockets.
- Parse and comprehend XML data over an XML stream.
- Understand the main Jabber data types.

There are various Jabber client libraries, which handle the complex functions such as parsing of data at the client end. This allows you to concentrate on the user interface of the client application.

Note:

You can develop Jabber client in any language that supports XML messaging. The applications in this book use Java language to develop IM applications.

Jabber Identifier

Jabber Identifier (JID) provides a unique identification to different entities that communicate with each other by using Jabber protocols. The format of a JID is:

```
[node@]domain[/resource]
```

The elements in the JID are:

- The node identifier that signifies an end user. It is a mandatory element for a valid JID.
- The domain identifier that signifies the Jabber server. It is a mandatory element for a valid JID.
- The resource identifier signifies specific resources that belong to an end user. End users can maintain multiple simultaneous connections to the same Jabber server by using different resources.

Jabber XML Protocol

Jabber supports communication by using XML streams. The three core XML elements in the Jabber XML protocol are:

- <message/>: Contains messages send between two Jabber end users. Jabber supports various message types, such as normal messages, chat, group chat, headline, and error messages.
- <presence/>: Provides information about whether or not a Jabber entity, represented by JID, is available.
- <iq/>(Info/Query): Structures a basic conversation between two Jabber entities and allows the participating objects to exchange XML-formatted data.

Advantages of Jabber

Jabber allows you to develop products for real-time communication and presence-based services in which a Jabber server maintains chat sessions of end users. Some advantages of Jabber are:

- Open Source: Provides open-source protocols. Jabber also provides free and paid implementations for servers, clients, and components that can be integrated into your existing applications. The free implementation of servers includes jabberd version 1.0 onwards, OpenIM and ejabberd. The paid implementation of servers includes Antepo and Merak. The free implementation of client includes Agile, Colibri, and e4Applet. The paid implementation of client includes Akeni, Chatopus, and IMChat. The freeware components include oajabber. The paid compenents includes myMatrix and XMPP Com Library.
- XML Message Structure: Utilizes XML message structure for data streaming between end users.
- Decentralization: Allows you to develop and run your customized Jabber server due to similarity between Jabber network and e-mail architecture. This allows companies and individuals to run their customized IM services.
- Extensibility: Allows extending the basic XML format by using the XML and custom namespaces to customize the protocols for specialized applications.
- Diversity: Provides flexibility to build real-time applications and services apart from IM, such as file sharing, gaming, and remote system monitoring.

Future Enhancements

There is a scope for improvement to extend the functionality provided by Jabber. The Jabber community is currently working on enhancements that can be incorporated in future releases. The enhancements that Jabber server should support are:

- Support for enhanced basic functions, such as voice chat and whiteboard.
- Support for standard enterprise features, such as quality of service support.
- Support for improved security, and error-detection and correction mechanisms to formulate more advanced communication systems.

Chapter 2: Creating a Connector Application

Download CD Content

A Jabber session consists of two parallel XML streams, one from the client to the server and the other from the server to the client. The Connector application in this chapter uses the Jabber server, Ejabberd, to allow communication between end users.

Ejabberd is a free and open source distributed Jabber server. You can download Ejabberd from the following URL:

http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=10

Ejabberd is Extensible Messaging and Presence Protocol (XMPP) compliant and provides various features, such as built-in Web-based administration interface, support for multi-user chat sessions, and support for Lightweight Directory Access Protocol (LDAP) authentication.

This chapter describes how to develop the Connector application, which allows end users to send and receive messages from other end users connected to the Jabber server. To send and receive messages, end users need to first log on or sign up for a new account.

Architecture of the Connector Application

The Connector application provides an interface with two text areas, Client Response and Server Response. The application allows end users to view the XML streams sent and received by the Connector application in the Client Response and the Server Response text area respectively. The Client Response text area shows the request sent to the Jabber server. The Server Response text area shows the response received from the Jabber server. The Connector application allows end users to view XML streams for various tasks, such as sign up, log on, send message, enter chat room, and log off.

The Connector application uses the following files:

- JabberClient.java: Allows end users to establish a connection with the Jabber server and communicate with other end users connected to the server.
- ChatRoom.java: Allows end users to enter any chat room that exists on the Jabber server.
- MessageClass.java: Allows end users to send messages to other end users on the Internet.
- SignUp.java: Allows end users to sign up for a new account.
- UserLogin.java: Allows end users to log on as existing users.
- SocketClass.java: Opens a socket to send and receive messages with the help of the Jabber server.

Figure 2-1 shows the architecture of the Connector application:

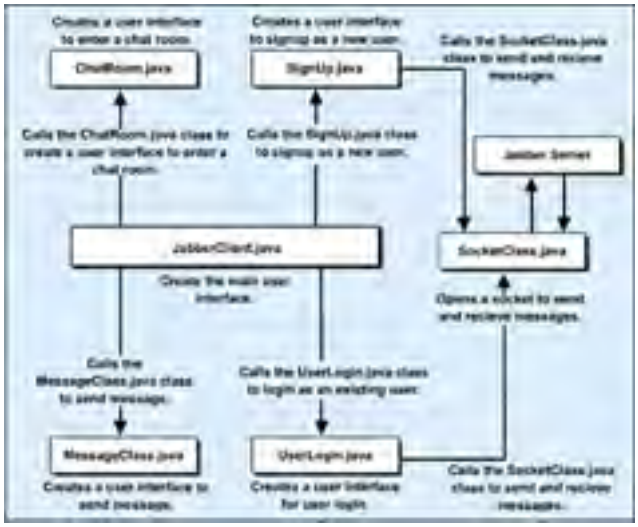


Figure 2-1: Architecture of the Connector Application

The JabberClient.java file creates the user interface of the Connector application that contains the File menu and two text areas, Client Request and Server Response. The File menu allows end users to sign up for a new account or log on as existing end users.

If the end user selects File-> Login, the JabberClient.java file calls the UserLogin.java file, which allows the existing end users to log on. The UserLogin.java file provides an interface with various labels, three text boxes, and the Submit button.

If the end user selects File-> Signup, the JabberClient.java file calls the SignUp.java file, which allows end users to sign up for a new account. The SignUp.java file provides an interface with various labels, text boxes, and two buttons, OK and Cancel.

If the end user selects File-> Send Message, the JabberClient.java file calls the MessageClass.java file, which allows end users to send a message to other end users connected to the Jabber server. The MessageClass.java file provides an interface with various labels, text boxes, and two buttons, Send Message and Close.

If the end user selects File-> Enter Chat Room, the JabberClient.java file calls the ChatRoom.java file, which allows end users to enter any particular chat room that exists on the Jabber server. The ChatRoom.java file provides an interface with a label, text box, and two buttons, OK and Cancel.

Team LiB

PREVIOUS NEXT

Creating the User Interface for the Connector Application

The JabberClient.java file creates the user interface for the Connector application. Listing 2-1 shows the contents of the JabberClient.java file:

Listing 2-1: The JabberClient.java File

```
/*Imports required swing classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*
class JabberClient-This class is the main class of the application. This class initializes the
interface and loads all components like the JTextPane, button before displaying the result.
Constructor:
JabberClient-This constructor creates GUI.
Methods:
getStatus: This method is called to retrieve user status.
setStatus: This method is called to set user status.
setStatusLabel: This method is called to show status in to label.
setServerResponse: This method is called to show server response into text pane.
setMessageInToTextBox: This method is called to retrieve server name.
getServerName: This method is called to retrieve server name.
getUserName: This method is called to retrieve user name.
setUserNameAndPassword: This method is called to sets user name, password, and server name.
getRoomName: This method is called to retrieve room name.
setRoomName: This method is called to set the name of the chat room specified by the end user.
disableAndEnableMenuItems: This method is called to sets enabling or disabling of menu items.
main - This method creates the object of JabberClient.
*/
public class  JabberClient extends JFrame implements ActionListener
{
   /*Declares object of String class.*/
   String username="";
   String password="";
   String servername="";
   String usermode="notconnected";
   String roomname="";
   /*Declares object of Container class.*/
   Container container = null;
   /*Declares object of JMenuBar class.*/
   JMenuBar menubar;
   /*Declares object of JMenu class.*/
   JMenu filemenu;
   /*Declares object of JMenuItem class.*/
   JMenuItem login=null;
   JMenuItem signup = null;
   JMenuItem logoff = null;
   JMenuItem unsubscribe = null;
   JMenuItem sendmessagemenuitem = null;
   JMenuItem enterroommenuitem = null;
   JMenuItem exit = null;
   /*
   Declares object of JCheckBoxMenuItem class.
   */
   JCheckBoxMenuItem available = null;
   JCheckBoxMenuItem unavailable = null;
   JCheckBoxMenuItem chat = null;
   JCheckBoxMenuItem away = null;
   JCheckBoxMenuItem dod = null;
   JCheckBoxMenuItem gfc = null;
   JCheckBoxMenuItem newstatus = null;
   /*Declares object of JTextPane class.*/
   JTextPane clienttextpane = null;
   JTextPane servertextpane = null;
   /*Declares object of JScrollPane class.*/
   JScrollPane scpane = null;
   /*Declares object of JLabel class.*/
   JLabel statuslabel = null;
   /*Declares object of JButton class.*/
   JButton submitbutton = null;
   /*Declares object of SocketClass class.*/
   SocketClass socketclass;
   public JabberClient()
   {
      super("Connector Application");
      /*
      Initializes and sets the look and feel of the application to windows look and feel.
      */
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

```
}
catch(Exception e)
{
   System.out.println("Unable to set WLF"+e);
}
container =  this.getContentPane();
/*
Declares and initializes object of Dimension class.*/
Dimension screendimension=Toolkit.getDefaultToolkit().getScreenSize();
/* Sets window's location on screen.*/
setLocation(screendimension.width/2- 200,screendimension.height/2-350);
/*
Declares and initializes object of JPanel class.
*/
JPanel panel = new JPanel(false);
/*
Initializes object of JMenuBar class.
*/
menubar = new JMenuBar();
/*Add menubar to panel.*/
panel.add(menubar);
/*Initializes object of JMenu class.*/ filemenu = new JMenu("File");
/*
Initializes object of JMenuItem class.
*/
login= new JMenuItem("Login");
signup= new JMenuItem("Signup");
logoff= new JMenuItem("Logoff");
unsubscribe= new JMenuItem("Unsubscribe");
sendmessagemenuitem=new JMenuItem("Send Message");
enterroommenuitem=new JMenuItem("Enter Chat Room");
exit= new JMenuItem("Exit");
/*Adds menu to the menubar.*/
menubar.add(filemenu);
/*Adds menuitems to the menu.*/
filemenu.add(login);
filemenu.add(signup);
filemenu.add(logoff);
filemenu.add(unsubscribe);
filemenu.add(sendmessagemenuitem);
filemenu.add(enterroommenuitem);
filemenu.add(exit);
setJMenuBar(menubar);
/*
Adds action listener to the menuitems.
*/
login.addActionListener(this);
signup.addActionListener(this);
logoff.addActionListener(this);
unsubscribe.addActionListener(this);
sendmessagemenuitem.addActionListener(this);
senterroommenuitem.addActionListener(this);
sendmessagemenuitem.setEnabled(false);
enterroommenuitem.setEnabled(false);
exit.addActionListener(this);
/*
Declares and initializes objects of the JPanel class.
*/
JPanel rootpanel =  new JPanel(new GridLayout(2, 1, 6, 6));
JPanel clientpanel = new JPanel(new BorderLayout());
/*
Initializes object of JTextPane class.
*/
clienttextpane = new JTextPane();
/*
Declares and initializes objects of the JLabel class.
*/
JLabel clientrequestlabel = new JLabel("Client Request");
/*
Sets font of the clientrequestlabel label.
*/
clientrequestlabel.setFont(newFont("Verdana", Font.BOLD, 10));
/*
Adds clientrequestlabel to the clientpanel panel.
*/
clientpanel.add(clientrequestlabel, BorderLayout.NORTH);
/*
Sets size of the clienttextpane textpane
*/
clienttextpane.setPreferredSize(new Dimension(180, 120));
/*
Initializes object of JScrollPane class.
*/
scpane = new JScrollPane(clienttextpane, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
/*Adds scrollpane to the panel.*/
clientpanel.add(scpane,BorderLayout.CENTER);
/*
```

```
   Declares and initializes objects of the JPanel class.
   */
   JPanel submitpanel=new JPanel(new FlowLayout());
   /*
   Initializes object of JButton class.
   */
   submitbutton = new JButton("Submit");
   /*Adds action listener to the button.*/
   submitbutton.addActionListener(this);
   /*Adds button to the panel.*/
   submitpanel.add(submitbutton);
   clientpanel.add(submitpanel, BorderLayout.SOUTH);
   /*
   Declares and initializes objects of the JPanel class.
   */
   JPanel serverresponsepanel = new JPanel(new BorderLayout());
   /*
   Initializes objects of the JTextPane class.
   */
   servertextpane = new JTextPane();
   /*
   Declares and initializes objects of the JLabel class.
   */
   JLabel serverresponselabel = new JLabel("Server Response");
   /*Sets font of label.*/
   serverresponselabel.setFont(new Font("Verdana", Font.BOLD,10));
   /* Adds label to panel.*/
   serverresponsepanel.add(serverresponselabel, BorderLayout.NORTH);
   /*Sets size of textpane.*/
   servertextpane.setPreferredSize(new Dimension(180, 120));
   /*
   Initializes object of JScrollPane class.
   */
   scpane  = new JScrollPane(servertextpane, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
   /*Adds scrollpane to the panel.*/
   serverresponsepanel.add(scpane, BorderLayout.CENTER);
   /*
   Declares and initializes objects of the JPanel class.
   */
   JPanel emptypanel=new JPanel(new FlowLayout());
   /*
   Declares and initializes objects of the JLabel class.
   */
   JLabel empty = new JLabel(" ");
   emptypanel.add(empty);
   serverresponsepanel.add(emptypanel,BorderLayout.SOUTH);
   /*Adds clientpanel to the rootpanel.*/
   rootpanel.add(clientpanel);
   /*
   Adds serverresponsepanel to the rootpanel.
   */
   rootpanel.add(serverresponsepanel);
   /*
   Initializes objects of the JLabel class.
   */
   statuslabel = new JLabel("Status : Not connected");
   /*Sets font of the statuslabel label.*/
   statuslabel.setFont(new Font("Verdana", Font.BOLD, 10));
   /*Initializes object of SocketClass.*/
   socketclass=new SocketClass(this);
   /*Adds rootpanel to the container.*/
   container.add(rootpanel);
   /*Adds statuslabel to the container.*/
   container.add(statuslabel,BorderLayout.SOUTH);
   /*Sets clienttextpane disable.*/
   clienttextpane.setEnabled(false);
   /*Sets servertextpane disable.*/
   servertextpane.setEnabled(false);
   /* Sets logoff disable.*/
   logoff.setEnabled(false);
   /*Sets unsubscribe disable.*/
   unsubscribe.setEnabled(false);
   setSize(420, 650);
   setVisible(true);
}
/*
disableAndEnableMenuItems: This method is called to sets enabling or disabling of menu items.
Parameter: flag
Return Value: N/A
*/
public void disableAndEnableMenuItems(boolean flag)
{
   signup.setEnabled(flag);
   login.setEnabled(flag);
   logoff.setEnabled(!flag);
   unsubscribe.setEnabled(!flag);
```

```
   sendmessagemenuitem.setEnabled(!flag);
   enterroommenuitem.setEnabled(!flag);
}
/*
setRoomName: This method is called to set the name of the chat room specified by the end user.
Parameter: room-Object of String class.
Return Value: N/A
*/
public void setRoomName(String room)
{
   roomname=room;
}
/*
getRoomName: This method is called to retrieve room name
Parameter: N/A
Return Value: String
*/
public String getRoomName()
{
   return roomname;
}
/*
setUserNameAndPassword: This method is called to sets user name, password, and server name.
Parameter: N/A
Return Value: N/A
*/
public void setUserNameAndPassword(String username, String password, String servername)
{
   this.username=username;
   this.password=password;
   this.servername=servername;
}
/*
getUserName: This method is called to retrieve user name.
Parameter: N/A
Return Value: String
*/
public String getUserName()
{
   return username;
}
/*
getServerName: This method is called to retrieve server name.
Parameter: N/A
Return Value: String
*/
public String getServerName()
{
   return servername;
}
/*
setMessageInToTextBox: This method is called to retrieve server name.
Parameter: messagestring - object of String class.
Return Value: N/A
*/
public void setMessageInToTextBox(String messagestring)
{
   clienttextpane.setText(messagestring);
}
/*
setServerResponse: This method is called to show server response into text pane.
Parameter: servermessage - object of String class.
Return Value: N/A
*/
public void setServerResponse(String servermessage)
{
   servertextpane.setText(servertextpane.getText()+"\n"+servermessage);
}
/*
setStatusLabel: This method is called to show status in to label.
Parameter: st - object of String class.
Return Value: N/A
*/
public void setStatusLabel(String st)
{
   statuslabel.setText("Status : "+st);
}
/*
setStatus: This method is called to set user status.
Parameter: status - object of String class.
Return Value: N/A
*/
public void setStatus(String status)
{
   usermode=status;
}
/*
getStatus: This method is called to retrieve user status.
```

```
   Parameter: N/A
   Return Value: String
   */
   public String getStatus()
   {
      return usermode;
   }
   public void actionPerformed(ActionEvent ae)
   {
      if(ae.getSource() == login)
      {
         UserLogin userlogin=new UserLogin(socketclass,this);
      }
      else if(ae.getSource() == signup)
      {
         SignUp sign =new SignUp(socketclass,this);
         sign.show();
      }
      else if(ae.getSource() == logoff)
      {
         clienttextpane.setText("</stream:stream>");
         setStatus("endofstream");
      }
      else if(ae.getSource() == unsubscribe)
      {
         String removesubscriptionstring="";
         removesubscriptionstring="<iq type='set' from='"+username+"@"+servername+"/Home'
         id='1'>";
         removesubscriptionstring=removesubscriptionstring+"<query xmlns='jabber:iq:register'>";
         removesubscriptionstring=removesubscriptionstring+"<remove/>";
         removesubscriptionstring=removesubscriptionstring+"</query>";
         removesubscriptionstring=removesubscriptionstring+"</iq>";
         clienttextpane.setText(removesubscriptionstring);
         setStatus("removereg");
      }
      else if(ae.getSource() == exit)
      {
         setVisible(false);
      }
      else if(ae.getSource() == sendmessagemenuitem)
      {
         MessageClass messageclass=new MessageClass(this);
      }
      else if(ae.getSource()== enterroommenuitem)
      {
         ChatRoom chatroom=new ChatRoom(this);
      }
      else if(ae.getSource() == submitbutton)
      {
         if (!clienttextpane.getText().trim().equals(""))
         {
            socketclass.sendXMLToJabber(clienttextpane.getText().trim());
            if (clienttextpane.getText().indexOf("<message")!=-1)
            {
               setStatusLabel(" Message has been send");
            }
            servertextpane.setText("");
            if (usermode.equals("waitforauth"))
            {
               setStatusLabel("Waiting for authentication");
            }
            if (usermode.equals("endofstream"))
            {
               setStatusLabel("Not Connected");
            }
            if (usermode.equals("removereg"))
            {
               setStatusLabel("Wait for remove registration");
            }
            System.out.println("submit");
         }
      }
   }
   public static void main(String args[])
   {
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      JabberClient jabclient = new JabberClient();
   }
}
```

Download this listing.

In the above code, the main() method creates an instance of the JabberClient class.

The methods defined in Listing 2-1 are:

- `getStatus()`: Retrieves the status of the end user, such as log on, log off, sign up, or unsubscribe.
- `setStatus()`: Sets the status of the end user.
- `setStatusLabel()`: Shows the status of the end user in the interface provided by the Connecter application.
- `setServerResponse()`: Shows the server response in the Server Response text area provided by the Connector application.
- `setMessageInToTextBox()`: Shows the message in the Client Response text area.
- `getServerName()`: Retrieves the name of the Jabber server.
- `getUserName()`: Retrieves the name of the end user currently logged in.
- `setUserNameAndPassword()`: Sets the end user name, password, and Jabber server name.
- `getRoomName()`: Retrieves the name of the chat room specified by the end user.
- `setRoomName()`: Sets the name of the chat room specified by the end user.
- `disableAndEnableMenuItems()`: Enables or disables File menu items depending on the options selected by the end user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the action the end user performs.

The JabberClient.java file generates the main window of the Connector application, as shown in Figure 2-2:

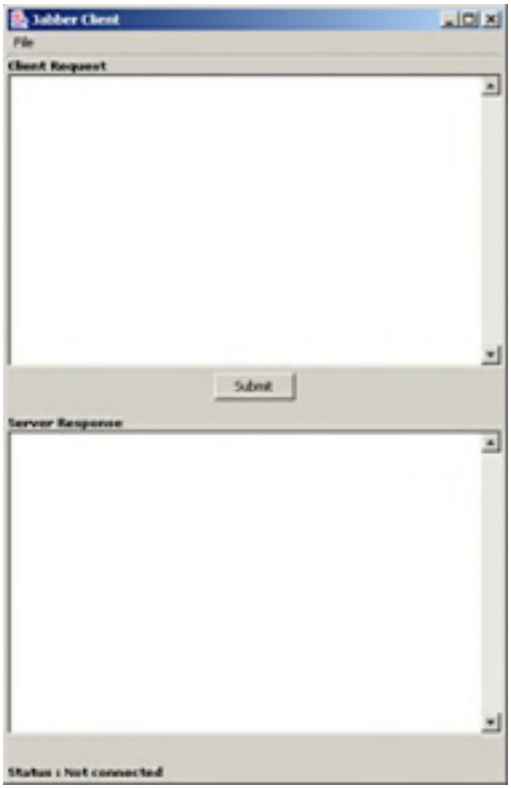


Figure 2-2: User Interface of the Connector Application

Select the File menu to view the File Menu options. Figure 2-3 shows the File menu options of the Connector application:

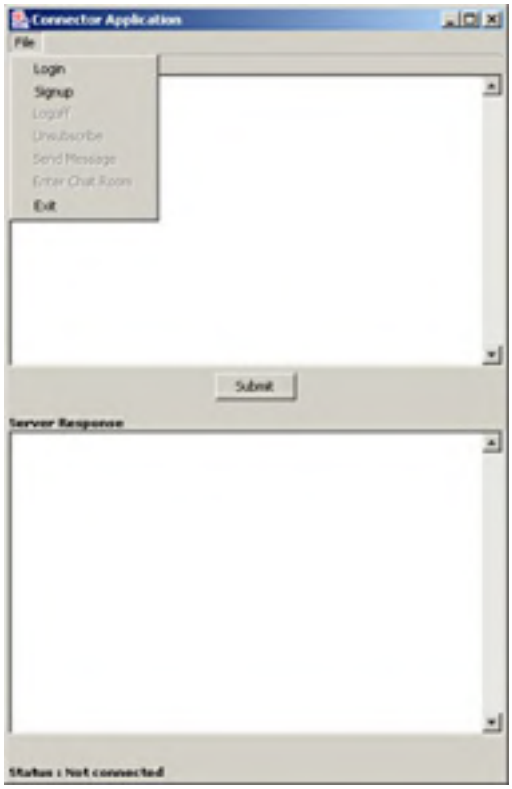


Figure 2-3: File Menu of the Connector Application

Select File-> Login to log on as an existing end user.

Select File-> Signup to sign up for a new account.

Select File-> Logoff and click the Submit button to log off from a chat session. The Connector application shows the confirmation message after logging out, as shown in Figure 2-4:

Figure 2-4: The Logoff Confirmation Message

Select File-> Unsubscribe and click Submit to unsubscribe from the Jabber server. The Connector application shows the confirmation message after unsubscribing an end user, as shown in Figure 2-5:

Figure 2-5: The Unsubscribe Confirmation Message

Select File-> Exit to close the application.

Creating the Sign Up Window

The SignUp.java file creates the user interface to allow an end user to sign up as a new user. Listing 2-2 shows the contents of the SignUp.java file:

Listing 2-2: The SignUp.java File

```
/*Imports required swing classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*
class SignUp - This class is used to create a GUI for signup a new user.
Constructor:
SignUp-This constructor creates GUI.
Methods:
actionPerformed - This method is called when end user clicks on send button.
checkfornumber -This function is used to check that supplied value contains a numeric value or not.
checkDateIsValid -This function is used to check that supplied date is a valid date.
isValidNumber -This function is used to check that supplied value is a valid number or not.
sendVCard -This function is used to create XML message of user profile.
*/
public class  SignUp extends JFrame implements ActionListener
{
   Container container;
   /*Declares objects of JLabel class.*/
   JLabel titlelabel;
   JLabel firstnamelabel = null;
   JLabel lastnamelabel = null;
   JLabel nicknamelabel=null;
   JLabel birthdaylabel=null;
   JLabel usernamelabel = null;
   JLabel passwordlabel = null;
   JLabel telephonelabel=null;
   JLabel confirmpassword = null;
   JLabel citylabel=null;
   JLabel countrylabel=null;
   JLabel addresslabel = null;
   /*Declares objects of JTextField class.*/
   JTextField firstnametext = null;
   JTextField lastnametext = null;
   JTextField nicknametext = null;
   JTextField usernametext = null;
   JPasswordField passwordtext = null;
   JPasswordField confirmpasswordtext = null;
   JTextField birthdaytext=null;
   JTextField telephonetext=null;
   JTextField addresstext = null;
   JTextField citytext=null;
   JTextField countrytext = null;
   JTextField titletext=null;
   /*Declares objects of JButton class.*/
   JButton okbutton = null;
   JButton cancelbutton = null;
   /*Declares objects of JPanel class.*/
   JPanel signuppanel;
   JPanel buttonpanel;
   /*Declares objects of SocketClass class.*/
   SocketClass socketclass;
   /*Declares objects of JabberClient class.*/
   JabberClient jabberclient;
   public SignUp(SocketClass socketclass,JabberClient jc)
   {
      super("Sign Up");
      container = this.getContentPane();
      this.socketclass=socketclass;
      jabberclient=jc;
      /*
      Declares and initializes the object of JPanel class.
      */
      JPanel jpanel = new JPanel();
      /*
      Declares and initializes the object of JLabel class.
      */
      JLabel labelheading =new JLabel("Sign Up");
      labelheading.setFont(new Font("Verdana", Font.BOLD,12));
      jpanel.add(labelheading);
      container.add(jpanel,BorderLayout.NORTH);
      /*
      Initializes the objects of JLabel class.*/
      titlelabel=new JLabel("Title");
      firstnamelabel = new JLabel("First Name :");
```

```
    lastnamelabel = new JLabel("Last Name :");
    nicknamelabel=new JLabel("Nick Name");
    birthdaylabel=new JLabel("DOB (dd/mm/yyyy)");
    usernamelabel = new JLabel("User Name :");
    passwordlabel = new JLabel("Password :");
    confirmpassword = new JLabel("Confirm Password :");
    telephonelabel=new JLabel("Telephone");
    addresslabel = new JLabel("Address :");
    citylabel=new JLabel("City");
    countrylabel=new JLabel("Country");
    /*
    Initializes the objects of JLabel class.*/
    titletext=new JTextField();
    firstnametext = new JTextField();
    lastnametext = new JTextField();
    nicknametext=new JTextField();
    usernametext = new JTextField();
    passwordtext = new JPasswordField();
    confirmpasswordtext = new JPasswordField();
    birthdaytext=new JTextField();
    telephonetext=new JTextField();
    addresstext =  new JTextField();
    citytext=new JTextField();
    countrytext =new JTextField();
    /*
    Initializes the objects of JPanel class.*/
    signuppanel = new JPanel(new GridLayout(12, 6, 7, 7));
    /*
    Adds labels and textfields to the panel.
    */
    signuppanel.add(titlelabel);
    signuppanel.add(titletext);
    signuppanel.add(firstnamelabel);
    signuppanel.add(firstnametext);
    signuppanel.add(lastnamelabel);
    signuppanel.add(lastnametext);
    signuppanel.add(nicknamelabel);
    signuppanel.add(nicknametext);
    signuppanel.add(birthdaylabel);
    signuppanel.add(birthdaytext);
    signuppanel.add(telephonelabel);
    signuppanel.add(telephonetext);
    signuppanel.add(addresslabel);
    signuppanel.add(addresstext);
    signuppanel.add(citylabel);
    signuppanel.add(citytext);
    signuppanel.add(countrylabel);
    signuppanel.add(countrytext);
    signuppanel.add(usernamelabel);
    signuppanel.add(usernametext);
    signuppanel.add(passwordlabel);
    signuppanel.add(passwordtext);
    signuppanel.add(confirmpassword);
    signuppanel.add(confirmpasswordtext);
    /*Sets the size of the addresstext.*/
    addresstext.setPreferredSize(new Dimension(120,28));
    /*Adds signuppanel to the container.*/
    container.add(signuppanel);
    /*
    Initializes the object of JPanel class.*/
    buttonpanel = new JPanel();
    /*
    Initializes the objects of JButton class.
    */
    okbutton = new JButton("OK");
    cancelbutton = new JButton("Cancel");
    /*
    Adds the ActionListener to the objects of the JButton class.
    */
    okbutton.addActionListener(this);
    cancelbutton.addActionListener(this);
    /*
    Adds the action command to the objects of the JButton class.
    */
    okbutton.setActionCommand("ok");
    cancelbutton.setActionCommand("cancel");
    buttonpanel.add(okbutton);
    buttonpanel.add(cancelbutton);
    container.add(buttonpanel, BorderLayout.SOUTH);
    setSize(350, 400);
    setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
    String actioncommand=ae.getActionCommand();
    String sessionstart="";
    if (actioncommand.equals("ok"))
    {
```

```
            if (!socketclass.isConnected)
            {
               socketclass.socketoOpenClass("localhost", 5222, 1000);
            }
            if (jabberclient.getStatus().equals("notconnected")||jabberclient.getStatus().
            equals("waitforreg")||jabberclient.getStatus().equals("waitforauth"))
            {
               sessionstart=socketclass.sendSessionStartMessage();
            }
            String firstname=firstnametext.getText().trim();
            String username=usernametext.getText().trim();
            String password=passwordtext.getText().trim();
            String confirmpassword=confirmpasswordtext.getText().trim();
            String dobstring=birthdaytext.getText().trim();
            String telephonestring=telephonetext.getText().trim();
            String lastnamestring=lastnametext.getText().trim();
            String errormessage="";
            if (firstname.equals(""))
            {
               errormessage="First Name is a required field.\n";
            }
            else
            {
               errormessage=checkfornumber(firstname, "First Name");
            }
            errormessage=errormessage+checkfornumber(lastnamestring, "Last Name");
            if (username.equals(""))
            {
               errormessage=errormessage+"User Name is a required field.\n";
            }
            if (password.equals(""))
            {
               errormessage=errormessage+"Password is a required field.\n";
            }
            if (!confirmpassword.equals(password))
            {
               errormessage=errormessage+"Confirm password and Password should be same.\n";
            }
            errormessage=errormessage+checkDateIsValid(dobstring,"DOB");
            errormessage=errormessage+isValidNumber(telephonestring, "Telephone");
            if (!errormessage.equals(""))
            {
               JOptionPane.showMessageDialog(null,errormessage, "Error", JOptionPane.PLAIN_MESSAGE);
               return ;
            }
            socketclass.setUserNameAndPassword(username, password, "localhost");
            jabberclient.setMessageInToTextBox(sessionstart+"\n"+socketclass.sendRegistration()+"\n"+
            sendVCard());
            jabberclient.setStatus("waitforreg");
            setVisible(false);
         }
         else if(actioncommand.equals("cancel"))
         {
            setVisible(false);
         }
      }
      /*
      checkfornumber-This function is used to check that supplied value contains a numeric value or not
      Parameters: value-object of String class, fieldname-object of String class.
      Return Value: String
      */
      public String checkfornumber(String value, String fieldname)
      {
         String message="";
         for(int i=0;i<value.length();i++)
         {
            if  ("1234567890".indexOf(value.charAt(i))!=-1)
            {
               message="Please enter a valid value for "+fieldname+".\n";
               return message;
            }
         }
         return "";
      }
      /*
      checkDateIsValid-This function is used to check that supplied date is a valid date.
      Parameters: datestring-object of String class, field-object of String class.
      Return Value: String
      */
      public String checkDateIsValid(String datestring,String field)
      {
         String[] spliteddate=datestring.split("/");
         String doberror="";
         int day;
         int month;
         int year;
         System.out.println(spliteddate.length);
         try
```

```
        {
           day=Integer.parseInt(spliteddate[0]);
           month=Integer.parseInt(spliteddate[1]);
           year=Integer.parseInt(spliteddate[2]);
        }
        catch(NumberFormatException nfe)
        {
           doberror="Please enter a valid value for "+field+".\n";
        return doberror;
        }
        if (spliteddate.length!=3)
        {
           doberror="Please enter a valid value for "+field+".\n";
        }
        else if (day>31||day<0||month>12||month<0||year>2000||year<1900)
        {
           doberror="Please enter a valid value for "+field+".\n";
        }
        return doberror;
     }
     /*
     isValidNumber-This function is used to check that supplied value is a valid number or not.
     Parameters: number-object of String class, field-object of String class.
     Return Value: String
     */
     public String isValidNumber(String number,String field)
     {
        String message="";
        for (int i=0;i<number.length();i++ )
        {
           if ("1234567890".indexOf(number.charAt(i))==-1)
           {
              message=message+"Please enter a valid value for "+field+".\n";
              return message;
           }
        }
        return "";
     }
     /*
     sendVCard -This function is used to create XML message of user profile.
     Parameters: N/A
     Return Value: String
     */
     public String sendVCard()
     {
        String vcardstring="";
        vcardstring="<iq type='set' id='1'>\n";
        vcardstring=vcardstring+"<vCard  xmlns='vcard-temp'>\n";
        vcardstring=vcardstring+"<FN>"+firstnametext.getText().trim()+"</FN>\n";
        vcardstring=vcardstring+"<N>\n";
        vcardstring=vcardstring+"<FAMILY>??</FAMILY>\n";
        vcardstring=vcardstring+"<GIVEN>"+firstnametext.getText().trim()+"</GIVEN>\n";
        vcardstring=vcardstring+"<MIDDLE/></N>\n";
        vcardstring=vcardstring+"<NICKNAME>"+nicknametext.getText().trim()+"</NICKNAME>\n";
        vcardstring=vcardstring+"<URL>??</URL>\n";
        vcardstring=vcardstring+"<BDAY>"+birthdaytext.getText().trim()+"</BDAY>\n";
        vcardstring=vcardstring+"<ORG>\n<ORGNAME>??</ORGNAME>\n";
        vcardstring=vcardstring+"<ORGUNIT/>\n";
        vcardstring=vcardstring+"</ORG>\n";
        vcardstring=vcardstring+"<TITLE>"+titletext.getText().trim()+"</TITLE>\n";
        vcardstring=vcardstring+"<ROLE>??</ROLE>\n";
        vcardstring=vcardstring+"<TEL>\n<VOICE/>\n<HOME>"+
        telephonetext.getText().trim()+ "</HOME>\n</TEL>\n";
        vcardstring=vcardstring+"<TEL>\n<FAX/>\n<HOME/>\n</TEL>\n";
        vcardstring=vcardstring+"<TEL>\n<MSG/>\n<HOME/>\n</TEL>\n";
        vcardstring=vcardstring+"<ADR>\n<HOME/>\n<EXTADD/>\n<STREET/>\n";
        vcardstring=vcardstring+"<LOCALITY>"+citytext.getText().trim()+"</LOCALITY>\n";
        vcardstring=vcardstring+"<REGION>??</REGION>\n";
        vcardstring=vcardstring+"<PCODE>??</PCODE>\n";
        vcardstring=vcardstring+"<COUNTRY>"+countrytext.getText().trim()+"</COUNTRY>\n";
         vcardstring=vcardstring+"</ADR>\n<TEL>\n<VOICE/>\n<WORK/>\n</
         TEL>\n<TEL>\n<FAX/>\n<WORK/>\n</TEL>\n";
        vcardstring=vcardstring+"<TEL>\n<MSG/>\n<WORK/>\n</TEL>\n";
        vcardstring=vcardstring+"<ADR>\n<WORK/>\n<EXTADD/>\n<STREET/>\n<
        LOCALITY>??</LOCALITY>\n";
        vcardstring=vcardstring+"<REGION>??</REGION>\n";
        vcardstring=vcardstring+"<PCODE>??</PCODE>\n";
        vcardstring=vcardstring+"<COUNTRY>??</COUNTRY>\n";
        vcardstring=vcardstring+"</ADR>\n";
        vcardstring=vcardstring+"<EMAIL>\n<INTERNET/>\n <PREF/>??</EMAIL>\n";
        vcardstring=vcardstring+"</vCard>\n</iq>\n";
        return vcardstring;
     }
}
```

Download this listing.

In the above code, the constructor of the SignUp class takes an object of the SocketClass class and an object of the JabberClient class as input parameters. This allows an end user to invoke the methods of the SocketClass and JabberClient classes.

The methods defined in Listing 2-2 are:

- actionPerformed: Acts as an event listener and activates an appropriate method based on the action the end user performs.
- Checkfornumber: Checks whether or not the information specified by the end user contains a numeric value.
- checkDateIsValid: Checks whether or not the date entered by the end user is a valid date.
- isValidNumber: Checks whether or not the value entered by the end user is numeric.
- sendVcard: Creates an XML message for the profile of an end user.

In Listing 2-2, the vCard format is set in the sendVcard method. The vCard format is a standard format for electronic business card data, which is useful for exchanging personal directory data across the Internet. The <vCard xmlns='vcard-temp'> XML tag sets the Vcard format for end users.

The SignUp.java file generates the Sign Up window, as shown in Figure 2-6:

Figure 2-6: The Sign Up Window

Creating the Login Window

The UserLogin.java file creates the user interface to log on as an existing user for the Connector application. Listing 2-3 shows the contents of the UserLogin.java file:

Listing 2-3: The UserLogin.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*
class UserLogin - This class is used to create a GUI for user login.
Constructor:
UserLogin-This constructor creates GUI.
Methods:
actionPerformed - This method is called when end user clicks on send button.
*/
public class UserLogin extends JDialog implements ActionListener
  {
      Container contentpane;
      /*
      Declares object of JPasswordField class.
      */
      JPasswordField passwordtextfield;
      /*
      Declares objects of JTextField class.
      */
      JTextField nametextfield;
      JTextField servernametextfield;
      /*Declares object of JButton class.*/
      JButton submitbutton;
      /*Declares objects of String class.*/
      String servername;
      String username;
      String password;
      String resource;
      String registrationstring="";
      /*Declares objects of JPanel class.*/
      JPanel note;
      JPanel combine;
      /*
      Declares objects of JabberClient class.
      */
      JabberClient jabberclient;
      /*
      Declares objects of SocketClass class.
      */
      SocketClass socketclass;
      boolean submitclickd=false;
      public UserLogin (SocketClass socketclass,JabberClient jabberhandler)
      {
         setTitle("Login Window");
         this.socketclass=socketclass;
         contentpane=getContentPane();
         jabberclient=jabberhandler;
         addWindowListener(new WindowAdapter()
         {
            public void windowClosing(WindowEvent we)
            {
               setVisible(false);
            }
         });
         /*
         Initializes the objects of JTextField class.
         */
         nametextfield=new JTextField();
         servernametextfield=new JTextField();
         /*
         Initializes the object of JPasswordField class.
         */
         passwordtextfield=new JPasswordField();
         /*
         Initializes the object of JButton class.
         */
         submitbutton=new JButton("Submit");
         /*
         Initializes the objects of JPanel class.
         */
```

```
    combine=new JPanel();
    note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
    submitbutton.setActionCommand("submit");
    submitbutton.addActionListener(this);
    submitbutton.setMnemonic('s');
    resource="Home";
    /*
    Declares and Initializes the objects of JPanel class.
    */
    JPanel controlpanel=new JPanel();
    JPanel lablepanel=new JPanel();
    /*
    Declares and Initializes the objects of JLabel class.
    */
    JLabel room= new JLabel("User Name");
    JLabel name= new JLabel("Password ");
    /*
    Sets the GridLayout of the controlpanel.
    */
    controlpanel.setLayout(new GridLayout(4, 2, 3, 3));
    controlpanel.setBorder(BorderFactory.createTitledBorder("Login Information"));
    /*
    Adds the labels and textfields to controlpanel.
    */
    controlpanel.add(room);
    controlpanel.add(nametextfield);
    controlpanel.add(name);
    controlpanel.add(passwordtextfield);
    controlpanel.add(serverl);
    controlpanel.add(servernametextfield);
    /*
    Declares and Initializes the objects of JLabel class.
    */
    JLabel heading=new JLabel("Login Window",JLabel.CENTER);
    /*Adds the heading to lablepanel.*/
    lablepanel.add(heading,BorderLayout.NORTH);
    /*Sets the font to the heading.*/
    heading.setFont(new Font("verdana", 1, 12));
    /*Adds the lablepanel to contentpane.*/
    contentpane.add(lablepanel,BorderLayout.NORTH);
    contentpane.add(controlpanel,BorderLayout.CENTER);
    /*
    Declares and Initializes the objects of JPanel class.
    */
    JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
    JPanel bp=new JPanel(new FlowLayout());
    bp.add(submitbutton);
    buttonpanel.add(note);
    buttonpanel.add(bp);
    contentpane.add(buttonpanel,BorderLayout.SOUTH);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(5, 5, 300, 240);
    submitbutton.addActionListener(this); show();
    getRootPane().setDefaultButton(submitbutton);
  }
public void actionPerformed(ActionEvent ae)
  {
    String actioncommand=ae.getActionCommand();
    servername=servernametextfield.getText().trim();
    username=nametextfield.getText().trim();
    password=passwordtextfield.getText().trim();
    if (username.equals(""))
    {
       JOptionPane.showMessageDialog(null, "User Name is a required
       field.", "Error", JOptionPane.PLAIN_MESSAGE);
       return;
    }
    if (password.equals(""))
    {
       JOptionPane.showMessageDialog(null,"Password is a required
       field.","Error",JOptionPane.PLAIN_MESSAGE);
       return;
    }
    if (actioncommand.equals("submit"))
    {
       String sessionstart="";
       if (!submitclickd)
       {
          submitclickd=true;
          if (servername.equals(""))
          { JOptionPane.showMessageDialog(null, "Server name is a required
          field.","Error",JOptionPane.PLAIN_MESSAGE);
          return;
       }
       if (jabberclient.getServerName().equals(servername))
       {
       }
       else
```

```
            {
               if (socketclass.isConnected)
               {
                  socketclass.closeConnection();
               }
               socketclass.socketoOpenClass(servername, 5222, 1000);
               if ((!socketclass.isConnected))
               {
                  setVisible(false);
                  return;
               }
            }
            if (jabberclient.getStatus().equals("notconnected")||jabberclient.getStatus().
            equals("waitforreg")||jabberclient.getStatus().equals("waitforauth"))
            {
               sessionstart=socketclass.sendSessionStartMessage();
            }
            jabberclient.setUserNameAndPassword(username,password,servername);
            socketclass.setUserNameAndPassword(username,password,servername);
            String authstring=socketclass.sendAuthorized();
            jabberclient.setMessageInToTextBox(sessionstart+"\n"+authstring);
            jabberclient.setStatus("waitforauth");
            setVisible(false);
            }
            else
            {
               submitclickd=false;
            }
         }
      }
   }
```

Download this listing.

In the above code, the constructor of the UserLogin class takes an object of the SocketClass class and an object of the JabberClient class as input parameters. This allows an end user to invoke the methods of the SocketClass and JabberClient classes. The UserLogin class defines the actionPerformed() method. The actionPerformed() method acts as an event listener and activates an appropriate method, based on the action an end user performs. End users invoke the actionPerformed() method by selecting any option from the User Login window.

Select File-> Login to log on as an existing end user. The UserLogin.java file generates the Login window, as shown in Figure 2-7:

Figure 2-7: The Login Window

PREVIOUS NEXT

Creating the Socket Class to Send and Receive Messages

The SocketClass.java file opens the socket to establish a connection with the Jabber server, to send and receive messages between an end user and the Jabber server. Listing 2-4 shows the contents of the SocketClass.java file:

Listing 2-4: The SocketClass.java File

```
/*Imports required util classes*/
import java.util.*;
/*Imports required io classes*/
import java.io.*;
/*Imports required net classes*/
import java.net.*;
/*Imports required swing classes*/
import javax.swing.*;
/*
class SocketClass-This class is used to open a socket and sends and receives Jabber messages.
Constructor:
SocketClass-This constructor creates GUI.
sendername: This method is called to substracte sender name from input message.
checkForError: This method is called to check whether the input string contains an error message or
not.
sendXMLToJabber: This method is called to send XML message to Jabber server.
closeConnection: This method is called to close client socket.
run: This method is called to listen server response sended by jabber server.
*/
class SocketClass implements Runnable
{
   /*Declares objects of Vector class.*/
   Vector tagvactor;
   /*Declares objects of PrintWriter class.*/
   PrintWriter out = null;
   /*
   Declares objects of BufferedReader class.
   */
   BufferedReader in = null;
   Socket clientsocket;
   Thread inputmessagethread;
   /*Declares objects of String class.*/
   String errortype;
   String resource="Home";
   String username="";
   String password="";
   String sendername="";
   String  recevername="";
   String registrationstring;
   String servermessage="";
   String ipaddress="";
   String endtagstr="";
   /*Declares object of UserLogin class.*/
   UserLogin userloginhandler;
   /*Declares object of JabberClient class.*/
   JabberClient jabberclient;
   int portno;
   int wait=1000;
   public boolean isConnected=false;
   public SocketClass(JabberClient jc)
   {
      jabberclient=jc;
   }
   /*
   isConnected: This method is called retrieve socket state.
   Parameter: N/A
   Return Value: boolean
   */
   public boolean isConnected()
   {
      return isConnected;
   }
   /*
   setUserLoginHandler: This method is called to set handler of the object of UserLogin class.
   Parameter: userlogin-Object of UserLogin  class.
   Return Value: N/A
   */
   public void setUserLoginHandler(UserLogin userlogin)
   {
      userloginhandler=userlogin;
   }
   /*
   socketoOpenClass: This method is called to open a socket and gets input and output streams.
   Parameter: ipaddress-Object of String class, portno, wait
   Return Value: N/A
   */
   public void socketoOpenClass(String ipaddress, int portno, int wait)
```

```
{
   this.ipaddress=ipaddress;
   this.portno=portno;
   this.wait=wait;
   openPort(ipaddress,portno,wait);
   if (clientsocket!=null)
   {
      isConnected=true;
   }
   else
   { JOptionPane.showMessageDialog(null, "Server not found.", "Error", JOptionPane.PLAIN_MESSAGE)
   return;
   }
   try
   {
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   inputmessagethread=new Thread(this);
   inputmessagethread.start();
}
/*
sendSessionStartMessage: This method is called to create a XML message string.
Parameter: N/A
Return Value: String
*/
public String sendSessionStartMessage()
{
   String sessionStratString;
   sessionStratString="<?xml version=\"1.0\"   encoding=\"UTF-8\" ?>\n";
   sessionStratString=sessionStratString+" <stream:stream";
   sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
   sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   return sessionStratString;
}
/*
sendAuthorized:  This method is called to create a XML message string.
Parameter: ipaddress - Object of String class, portno, wait
Return Value: N/A
*/
public String sendAuthorized()
{
   String authentication;
   authentication="<iq type=\"set\" id=\"1\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+" <username>"+username+"</username>";
   authentication=authentication+" <password>"+password+"</password>";
   authentication=authentication+" <resource>"+resource+"</resource> ";
   authentication=authentication+" </query> ";
   authentication=authentication+"</iq>";
   return authentication;
}
/*
sendRegistration:  This method is called to create a XML message string.
Parameter: ipaddress - Object of String class, portno, wait
Return Value: N/A
*/
public String sendRegistration()
{
   String registrationstr="";
   registrationstr="<iq type='set' to='"+username+".localhost' id='1'>";
   registrationstr=registrationstr+"<query xmlns='jabber:iq:register'>";
   registrationstr=registrationstr+"<username>"+username+"</username>";
   registrationstr=registrationstr+"<password>"+password+"</password>";
   registrationstr=registrationstr+"</query></iq>";
   return registrationstr;
}
/*
setUserNameAndPassword:  This method is called to set user name, password and servername
Parameter: username - Object of String class, password - Object of  String class, servername -
Object of  String class.
Return Value: N/A
*/
public void setUserNameAndPassword(String username, String password, String servername)
{
   this.username=username;
   this.password=password;
   this.ipaddress=servername;
}
/*
openPort:  This method is called to open a client socket.
Parameter: ipaddress - Object of String class, portno, timeinsec
```

```
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      SocketOpener sc=new SocketOpener(); Clientsocket=sc.openSocket(ipaddress,portno,timeinsec);
   }
}
/*
run:  This method is called to listen server   response sended by jabber server
Parameter: N/A
Return Value: N/A
*/
public void run()
{
   int i=0;
   String errororwarning="";
   String errorstring="";
   String inputstring="";
   boolean starttag=false;
   boolean endtag=false;
   boolean startwritting=false;
   String messagebody="";
   String tagentity="";
   Vector tagvactor;
   tagvactor=new Vector();
   int vactorindex=0;
   int starttagsing=0;
   while (true)
   {
      try
      {
         i=in.read();
         if (i==-1)
         {
            break;
         }
         else
         {
            inputstring=inputstring+(char)i;
            ervermessage=servermessage+(char)i;
            if ((char)i=='<')
            {
               starttag=true;
               starttagsing=1;
               stagentity="";
            }
            else
            {
               if ((char)i=='/' && starttag==true)
               {
                  if (starttagsing==1)
                  {
                     starttag=false;
                     endtag=true;
                     startwritting=false;
                     vactorindex=vactorindex-1;
                     if (vactorindex>=0) agvactor.removeElementAt(vactorindex);
                  }
                  tagentity=tagentity+(char)i;
               }
               else
               {
               starttagsing=0;
               if((char)i=='>')
            {
               jabberclient.setServerResponse(servermessage);
               servermessage="";
               if (endtag&&endtagstr.equals("stream:stream"))
               {
                  jabberclient.setStatusLabel("Logged Off");
                  jabberclient.setStatus("notconnected");
                  jabberclient.disableAndEnableMenuItems(true);
               }
               if (endtag)
               {
                  endtagstr="";
                  endtag=false;
               }
               if (starttag)
               {
                  if (tagentity.endsWith("/"))
```

```
                        { tagentity=tagentity.substring(0,tagentity.leng  th()-1);
                        }
                        sendername(tagentity);
                        if (tagentity.equals("remove"))
                        {jabberclient.setStatusLabel("Not Connected");
                        jabberclient.setStatus(""); jabberclient.disableAndEnableMenuItems(true);
                        } errorstring=checkForError(tagentity);
                        if (errorstring.equals("unauthorized"))
                        {
                           jabberclient.setStatus(""); JOptionPane.showMessageDialog(null, "User name o
                           password is invalid.", "Error",JOptionPane.PLAIN_MESSAGE);
                        }
                        if (errorstring.equals("user exists"))
                        {
                           if (jabberclient.getStatus().equals("waitforreg"))
                           {
                              jabberclient.setStatus("");
                              JOptionPane.showMessageDialog(null,"User ID already exists, Please choose
                              other User ID.", "Error", JOptionPane.PLAIN_MESSAGE);
                           }
                        }
                        if (errorstring.equals("User not exists.")&&messagebody.equals(""))
                        {
                           jabberclient.setStatusLabel(" Room does not exist");
                           JOptionPane.showMessageDialog(null,"Room does not
                           exist.","Error",JOptionPane.PLAIN_MESSAGE);
                        }
                     else if (errorstring.equals("User notexists.")&&(!messagebody.equals("")))
                     {
                        messagebody="";
                        JOptionPane.showMessageDialog(null,"User dose not
                        exists.","Error",JOptionPane.PLAIN_MESSAGE);
                     }
                     if (tagentity.equals("feature var='http://jabber.org/protocol/muc'"))
                     {
                        sendXMLToJabber("<presence from='"+username+"@conference."+ipaddress+"/Home'
                        to='"+jabberclient.getRoomName()+"@conference."+ipaddress+"/"+username+"'/>");
                     }
                     if (tagentity.indexOf("type='result'")!=-1)
                     {
                        String status=jabberclient.getStatus();
                        if (status.equals("waitforauth"))
                        {
                           jabberclient.setStatusLabel("User authenticated");
                           jabberclient.setStatus("auth");
                           jabberclient.disableAndEnableMenuItems(false);
                        }
                        if (status.equals("waitforreg"))
                        {jabberclient.setStatusLabel("Registered");
                        jabberclient.setStatus("reg");
                        }
                     }
                     startwritting=true;
                     tagvactor.insertElementAt(tagentity,vactorindex);
                     vactorindex=vactorindex+1;
                     starttag=false;
                     }
                  }
                  else
                  {
                     if (startwritting==true&&tagentity.trim().equals("body"))
                     {
                        messagebody=messagebody+(char)i;
                     }
                     if (starttag)
                     {
                        tagentity=tagentity+(char)i;
                     }
                     if (endtag)
                     {
                        endtagstr=endtagstr+(char)i;
                     }
                  }
               }
            }
         }
      }
     catch(IOException ie)
     {
     }
   }
   isConnected=false;
   }
/*
closeConnection: This method is called to close client socket.
Parameter: N/A
```

```
Return Value: N/A
*/
public void closeConnection()
{
   try
   {
      out.close();
      in.close();
      clientsocket.close();
   }
   catch (IOException e)
   {
      System.out.println(e);
   }
   isConnected=false;
}
/*
sendXMLToJabber: This method is called to send XML message to Jabber server.
Parameter: outputmessage - Object of String class.
Return Value: N/A
*/
public void sendXMLToJabber(String outputmessage)
{
   String[] tokenizerstring=outputmessage.split("\n");
   for (int i=0;i<tokenizerstring.length;i++ )
   {
      try
      {
         out.println(tokenizerstring[i]);
         out.flush();
      }
      catch(Exception e)
      {
         System.out.println("error");
         return;
      }
      out.flush();
   }
}
/*
checkForError:  This method is called to check whether the input string contains an error message or
not.
Parameter: tagentity - Object of String class.
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if ( tagentity.equals("error code='403' type='auth'"))
   {
      return "unauthorized";
   }
   if (tagentity.equals("error  code='404' type='cancel'"))
   {
      return "User not exists.";
   }
   if ((tagentity.indexOf("code='401'")>1)||(tagentity.indexOf("code=\"401\"")>1))
   {
      return "unauthorized";
   }
   if (tagentity.equals("error code='409' type='cancel'"))
   {
      return "user exists";
   }
   if ((tagentity.indexOf("code='409'")>1)||(tagentity.indexOf("code=\"409\"")>1))
   {
      return "user exists";
   }
   return error;
}
/*
sendername: This method is called to  substracte sender name from input message.
Parameter: tagentity - Object of String class.
Return Value: N/A
*/
public void sendername(String tagentity)
{
   if (tagentity.startsWith("message"))
   {
      sendername="";
      recevername=""; if(tagentity.indexOf("from=")>0&&tagentity.indxOf("to=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-2);
      }
      if (tagentity.indexOf("to=")>0&&tagentity.indexOf("type=")>0)
      { recevername=tagentity.substring(tagentity.indexOf("to=")+4,tagentity.indexOf("type=")-2);
      }
```

```
    }
  }
}
class SocketOpener
{
   private Socket socket;
   public Socket openSocket(String hostip,int portnumber, int timeinsec)
   {
      try
      {
         socket=new Socket(hostip,portnumber);
      }
      catch(IOException ie)
      {
         System.out.println("IE:::::::::: " +ie);
      }
      return getSocket();
   }
   public SocketOpener()
   {
   }
   public Socket getSocket()
   {
      return socket;
   }
};
```

Download this listing.

In the above code, the constructor of the SocketClass class takes the object of the JabberClient class to invoke the methods of the JabberClient class.

The methods defined in Listing 2-4 are:

- `sendername()`: Retrieves the sender's name from the input message received from the Jabber server.
- `checkForError()`: Checks whether or not the input string contains an error message.
- `sendXMLToJabber()`: Sends an XML message to the Jabber server.
- `closeConnection()`: Closes the client socket from the Jabber server.
- `run()`: Listens to the server response sent by the Jabber server.
- `isConnected()`: Determines whether or not a client socket is connected to the Jabber server.
- `setUserLoginHandler()`: Sets the handler of the object of the UserLogin class.
- `socketoOpenClass()`: Calls the openPort() method to open a client socket and gets input and output streams to send and receive messages.
- `sendSessionStartMessage()`: Creates an XML message string to initialize the session of an end user.
- `sendAuthorized()`: Creates an XML message string to authenticate the end users.
- `sendRegistration()`: Creates an XML message string to sign up for a new account with the Jabber server.
- `setUserNameAndPassword()`: Sets the user name, password, and Jabber server name.
- `openPort()`: Opens a client socket by using the IP address and port number of the Jabber server.

Sending Messages

The MessageClass.java file creates the user interface that allows the end user to specify the user ID and message. The MessageClass class sends the message to the user ID specified by the end user. Listing 2-5 shows the contents of the MessageClass.java file:

Listing 2-5: The MessageClass.java File

```
/*Imports required swing classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes.*/
import java.awt.*;
import java.awt.event.*;
/*
class MessageClass-This class is used to create a GUI for sending messages.
Constructor:
MessageClass-This constructor creates GUI.
Methods:
actionPerformed-This method is called when end user clicks on send button.
*/
class MessageClass extends JFrame implements ActionListener
{
   /*Declares objects of JLabel class.*/
   JLabel useridlabel;
   JLabel messagelabel;
   JLabel lastmessagelabel;
   JLabel firstmessagelabel;
   /*Declares object of JTextField class.*/
   JTextField useridtext;
   /*Declares object of JTextArea class.*/
   JTextArea messagetextpane;
   /* Declares object of JScrollPane class.*/
   JScrollPane messagescrollpane;
   /*Declares objects of JButton class.*/
   JButton closebutton;
   JButton sendbutton;
   /* Declares objects of JPanel class.*/
   JPanel usermessagepanel;
   JPanel messagepanel;
   JPanel buttonpanel;
   /*Declares objects of JabberClient class.*/
   JabberClient jabberclient;
   public MessageClass(JabberClient jabberclient)
   {
      super("Message Window");
      Container contentpane=getContentPane();
      this.jabberclient=jabberclient;
      contentpane.add(new JLabel("  "),BorderLayout.WEST);
      /*
      Initializes the object of JLabel class
      */
      useridlabel=new JLabel("User ID");
      /*
      Initializes the object of JTextField class
      */
      useridtext=new JTextField();
      /*
      Initializes the object of JLabel class
      */
      messagelabel=new JLabel("Message");
      /*
      Initializes the object of JTextArea class
      */
      messagetextpane=new JTextArea();
      /*
      Initializes the object of JScrollPane class
      */
      messagescrollpane=new JScrollPane(messagetextpane, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
      JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
      /*
      Initializes the object of JPanel class
      */
      usermessagepanel=new JPanel();
      /*
      Initializes the objects of JButton class.
      */
      closebutton=new JButton("Close");
      sendbutton=new JButton("Send Message");
      /*
      Adds the ActionListener to the object of the JButton class.
      sendbutton.addActionListener(this);
      closebutton.addActionListener(this);
      /*
```

```
      Sets the ActionCommand of the object of JButton class.
      */
      sendbutton.setActionCommand("send");
      closebutton.setActionCommand("close");
      /*
      Declare and initialize the object of GridLayout class.
      */
      GridLayout usermessagegridlayout=new GridLayout(2, 2, 5, 5);
      /*
      Set the Gridlayout to usermessagepanel object.
      */
      usermessagepanel.setLayout(usermessagegridlayout);
      /*
      Initializes the objects of JLabel class.
      */
      firstmessagelabel=new JLabel("Message", JLabel.RIGHT);
      usermessagepanel.add(firstmessagelabel);
      /*
      Initializes the objects of JLabel class.
      */
      lastmessagelabel=newJLabel("Window", JLabel.LEFT);
      firstmessagelabel.setFont(newFont("Verdana", Font.BOLD, 12));
      lastmessagelabel.setFont(new Font("Verdana", Font.BOLD, 12));
      usermessagepanel.add(lastmessagelabel);
      /*
      Adds useridlabel to usermessagepanel
      */
      usermessagepanel.add(useridlabel);
      /*Adds useridtext to usermessagepanel*/
      usermessagepanel.add(useridtext);
      /*
      Initializes the objects of JPanel class.
      */
      messagepanel=new JPanel();
      /*
      Declare and initialize the object of GridLayout class.
      */
      GridLayout messagegridlayout=new GridLayout(1, 2, 5, 5);
      /*
      Set the Gridlayout to messagepanel object.
      */
      messagepanel.setLayout(messagegridlayout);
      /*Adds messagelabel to messagepanel*/
      messagepanel.add(messagelabel);
      /*
      Adds messagescrollpane to messagepanel
      */
      messagepanel.add(messagescrollpane);
      /*
      Declare and initialize the object of JPanel  class.
      */
      JPanel toppanel=new JPanel(new  GridLayout(2,1,5,5));
      /*Adds usermessagepanel to toppanel*/
      toppanel.add(usermessagepanel);
      /* Adds messagepanel to toppanel*/
      toppanel.add(messagepanel);
      /*Adds toppanel to contentpane*/contentpane.add(toppanel,BorderLayout.CENTER);
      buttonpanel=new JPanel(new GridLayout(1, 2, 0, 0));
      buttonpanel.add(new JLabel(" "));
      /*
      Declare and initialize the object of JPanel class.
      */
      JPanel subbuttonpanel=new JPanel();
      /* Adds buttons to subbuttonpanel*/
      subbuttonpanel.add(sendbutton);
      subbuttonpanel.add(closebutton);
      buttonpanel.add(subbuttonpanel);
      contentpane.add(buttonpanel,BorderLayout.SOUTH);
      useridtext.setPreferredSize(new  Dimension(80,20));
      setSize(400,200);
      setVisible(true);
   }
   public void actionPerformed(ActionEvent ae)
   {
      String actioncommand=ae.getActionCommand();
      String userid=useridtext.getText().trim();
      String message=messagetextpane.getText().trim();
      String errorstring="";
      if (actioncommand.equals("send"))
      {
         if (userid.equals(""))
         {
            errorstring="User ID is a required field.\n";
         }
         if (message.equals(""))
         {
            errorstring=errorstring+"Message field cannot be left blank.";
         }
```

```
            if (!errorstring.equals(""))
            {
               JOptionPane.showMessageDialog(null,errorstring,"Error",JOptionPane.PLAIN_MESSAGE);
               return;
            }
            String username=jabberclient.getUserName();
            String servername=jabberclient.getServerName();
            String sendmessagestring="";
            sendmessagestring="<message type='chat' to='"+userid+"/Home'
            from='"+username+"@"+servername+"/Home'>\n";
            sendmessagestring=sendmessagestring+"<body>\n"+message+"\n";
            sendmessagestring=sendmessagestring+"</body>\n</message>";
            jabberclient.setMessageInToTextBox(sendmessagestring);
         }
         setVisible(false);
      }
}
```

Download this listing.

In the above code, the constructor of the MessageClass class takes an object of the JabberClient class as an input parameter. This allows an end user to invoke the methods of the JabberClient class.

The methods defined in Listing 2-5 are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.
- `getText()`: Retrieves the text message specified by the end user.

Select File-> Send Message to send the message to the specified end user. The Message Window appears, as shown in Figure 2-8:

Figure 2-8: The Message Window

The MessageClass.java file allows end users to specify the User ID and Message in the Message Window.

Creating the Chat Room Interface

The ChatRoom.java file creates the user interface that allows end users to enter the chat room by specifying the name of the chat room for the Connector application. Listing 2-6 shows the contents of the ChatRoom.java file:

Listing 2-6: The ChatRoom.java File

```
/*Imports required swing classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*
class ChatRoom-This class is used to create a GUI to enter in a existing room
Constructor:
ChatRoom-This constructor creates GUI.
Methods:
*/
class ChatRoom extends JFrame implements ActionListener
{
   /*Declares objects of JLabel class.*/
   JLabel roomnamelabel;
   JLabel pageheadelabel;
   /*Declares object of JTextField class.*/
   JTextField roomnametext;
   /* Declares objects of JPanel class.*/
   JPanel chatroompanel;
   JPanel buttonpanel;
   /* Declares objects of JButton class.*/
   JButton okbutton;
   JButton cancelbutton;
   /* Declares object of String class.*/
   String roomnamestring;
   /* Declares object of JabberClient class.*/
   JabberClient jabberclient;
   public ChatRoom(JabberClient jabberclient)
   {
      super("Enter Chat Room");
      Container contentpane=getContentPane();
      this.jabberclient=jabberclient;
      /* Initializes object of JLabel class.*/
      pageheadelabel=new JLabel("Enter Chat Room", JLabel.CENTER);
      /*
      Sets the font to the object of JLabel  class.
      */
      Pageheadelabel.setFont(new Font("Verdana", Font.BOLD, 12));
      /* Adds the label to the contentpane.*/
      contentpane.add(pageheadelabel, BorderLayout.NORTH);
      /*
      Initializes the object of JLabel  class.*/
      roomnamelabel=new JLabel("Room Name");
      /*
      Initializes the object of JTextField class.
      */
      roomnametext=new JTextField();
      /*
      Initializes the objects of JPanel class.*/
      chatroompanel=new JPanel();
      buttonpanel=new JPanel();
      /*
      Sets the size of the objects of JLabel class.
      */
      roomnamelabel.setPreferredSize(new Dimension(120, 23));
      roomnametext.setPreferredSize(new Dimension(170, 23));
      /*
      Declare and initialize the object of FlowLayout class.
      */
      FlowLayout chatroomflowlayout=new FlowLayout();
      /*
      Set the FlowLayout to chatroompanel object.
      */
      chatroompanel.setLayout(chatroomflowlayout);
      /*Adds the label to the panel.*/
      chatroompanel.add(roomnamelabel);
      /*Adds the roomnametext to the panel.*/
      chatroompanel.add(roomnametext);
      contentpane.add(chatroompanel);
      /*
      Initializes the objects of JButton class.*/
      okbutton=new JButton("OK");
      cancelbutton=new JButton("Cancel");
      /*
      Adds the okbutton to the buttonpanel.
```

```
        */
        buttonpanel.add(okbutton);
        /* Sets the size of the buttons.*/
        okbutton.setPreferredSize(new Dimension(80, 23));
        cancelbutton.setPreferredSize(new Dimension(80, 23));
        /*
        Adds the ActionListener to the buttons.
        */
        okbutton.addActionListener(this);
        cancelbutton.addActionListener(this);
        okbutton.setActionCommand("ok");
        cancelbutton.setActionCommand("cancel");
        buttonpanel.add(cancelbutton);
        contentpane.add(buttonpanel,BorderLayout.SOUTH);
        setSize(320,140);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String actioncommand=ae.getActionCommand();
        String roomquerystring="";
        if (actioncommand.equals("ok"))
        {
            roomnamestring=roomnametext.getText().trim();
            if (roomnamestring.equals(""))
            {
                JOptionPane.showMessageDialog(null,"Room name is a required
                field.","Error",JOptionPane.PLAIN_MESSAGE);
                return;
            }
            roomquerystring="<iq from='"+jabberclient.getUserName()+"@"+jabberclient.
            getServerName()+"/Home' id='1' to='"+roomnamestring+"@conference."+jabberclient.
            getServerName()+"' type='get'>";
            roomquerystring=roomquerystring+"<query
            xmlns='http://jabber.org/protocol/disco#info'/></iq>";
            jabberclient.setMessageInToTextBox(roomquerystring);
            jabberclient.setRoomName(roomnamestring);
        }
        setVisible(false);
    }
}
```

Download this listing.

In the above code, the constructor of the ChatRoom.java class takes an object of the JabberClient class as an input parameter to invoke the methods of the JabberClient class. The ChatRoom class allows you to create the Enter Chat Room window to enter the name of the chat room in the Connector application.

The methods defined in Listing 2-6 are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.
- `getText()`: Requests the end users to specify the name of the chat room.

Select File-> Enter Chat Room to enter the chat room. The ChatRoom.java generates the Enter Chat Room window, as shown in Figure 2-9:

Figure 2-9: The Enter Chat Room Window

Unit Testing

To test the Connector application:

1. Download and install the free implementation of Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Copy the JabberClient.java, ChatRoom.java, MessageClass.java, SignUp.java, User Login.java, and SocketClass.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder where you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

4. To run the Connector application, specify the following command at the command prompt:

   ```
   java JabberClient
   ```

5. The user interface of the Connector application appears. Select File-> Signup to sign up for a new account in the Connector application. The Sign Up window appears.

6. Fill up the fields of the Sign Up window, as shown in Figure 2-10:

Figure 2-10: Specifying Information in the Sign Up Window

7. Click OK to generate the XML message.

8. Click Submit to create a new account in the Jabber server. The Jabber server returns a confirmation response, as shown in Figure 2-11:

Figure 2-11: The Signup Confirmation

9. Select File-> Login to log on to the Connector application as an existing user. The Login window appears.
10. Fill up the login information, as shown in Figure 2-12:

Figure 2-12: Specifying Information in the Login Window

11. Click the Submit button. The confirmation XML message appears, as shown in Figure 2-13:

Figure 2-13: The Login Confirmation Message

The Client Request text area shows the request for logging on to the Jabber server in the form of an XML stream and the Server Response text area shows the XML stream after successfully logging on to the Jabber server

12. Select File-> Logoff and click the Submit button to log off from the Connector application. The Logoff confirmation XML message appears.
13. Select File-> Unsubscribe and click the Submit button to unsubscribe from the Jabber server. The Unsubscribe confirmation XML message appears.
14. Select File-> Send Message to send a message to another end user connected to the Jabber server. The Send Message window appears.
15. Specify the text for the message in the Send Message window, as shown in Figure 2-14:

Figure 2-14: Specifying the Message Text

16. Click the Submit button. The confirmation message appears, as shown in Figure 2-15:

Figure 2-15: Message Confirmation

The Client Request text area shows the request, which includes a text message and the Server Response text area shows the XML stream after successfully sending the message to the Jabber server

17. Select File-> Enter Chat Room to enter a particular chat room.
18. Fill the name of the chat room in the Enter Chat Room window, as shown in Figure 2-16:

Figure 2-16: Specifying the Chat Room Name

19. Click the Submit button. The confirmation message appears, as shown in Figure 2-17:

Figure 2-17: Enter Chat Room Confirmation

The Client Request text area shows the request to enter the chat room specified by an end user and the Server Response text area shows the XML stream after successfully entering the chat room.

◄ PREVIOUS NEXT ►

Chapter 3: Creating a Chat Room Application

Download CD Content

The Jabber protocol helps you develop chat room applications hosted on the Jabber server. End users can join a chat room to send and receive messages and view a list of other end users of that chat room.

This chapter describes how to develop the Chat Room application by using the Jabber protocol. The application allows end users to create or join a chat room and send messages to other end users of an active chat room.

Architecture of the Chat Room Application

The Chat Room application uses the following files:

- LoginGUI.java: Creates the Login window for the Chat Room application to allow end users to establish a connection with the Jabber server. This file allows end users to login as existing end users or create a new account.
- MainGUI.java: Allows end users to view a list of existing chat rooms and select any chat room from the given list.
- ChatRoom.java: Allows end users to enter a chat room by specifying a chat room name and a user name to participate in the group chat.
- RoomInformation.java: Allows end users to create a chat room by specifying a chat room name and a user name to initiate the group chat.
- GroupChat.java: Allows end users to participate in a group chat.
- SocketOpener.java: Opens a socket to send and receive messages through the Jabber server.

Figure 3-1 shows the architecture of the Chat Room application:

Figure 3-1: Architecture of the Chat Room Application

The LoginGUI.java file creates the Login window that contains various labels, text boxes, and the Submit button. This file allows end users to login as existing users or register for a new account.

When end users enter login information and click the Submit button on the Login window, the LoginGUI.java file calls the MainGUI.java file. The MainGUI.java file allows end users to create a chat room or join an existing chat room. The MainGUI.java file allows end users to select a chat room from the list of existing chat rooms provided by the user interface of the Chat Room application.

When end users select File->Create New Chat Room on the user interface of the Chat Room application, the MainGUI.java file calls the RoomInformation.java file to create a chat room. The RoomInformation.java file allows end users to specify a name for the chat room and a user name for the end user.

When end users select File->Go to Chat Room on the user interface of the Chat Room application, the MainGUI.java file calls the ChatRoom.java file to enter an existing chat room. The ChatRoom.java file allows end users to specify the name of the chat room and the user name to join the existing chat room.

When end users select any chat room from the list of existing chat rooms, the MainGUI.java file calls the SocketOpener.java file. The SocketOpener.java file opens the socket to send and receive messages between two end users. The SocketOpener.java file calls the GroupChat.java file to allow end users to participate in a group chat.

PREVIOUS NEXT

Creating the Login Window

The LoginGUI.java file creates the Login window for the Chat Room application. Listing 3-1 shows the contents of the LoginGUI.java file:

Listing 3-1: The LoginGUI.java File

```
/*Imports required swing package classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Imports required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*
Import required io and net classes
*/
import java.io.*;
import java.net.*;
/*
class LoginGUI - This class is used to create a GUI to get the information from user to login .
Constructor:
LoginGUI - This constructor creates GUI.
Methods:
getUserName:get the user name
getServerName:get the server name
*/
public class LoginGUI extends JFrame implements ActionListener, KeyListener
{
   /*Declare objects of Container class.*/
   Container c=null;
   /*Declare objects of JTextField class.*/
   JTextField namet=new JTextField();
   JTextField server=new JTextField();
   JPasswordField passwordt=new JPasswordField();
   /*Declare objects of JButton class.*/
   JButton b=new JButton("Submit");
   /*Declare objects of Socket class.*/
   Socket clientsocket;
   /*Declare String variable*/
   String servername;
   String username;
   String password;
   String resource="Home";
   String  recerverid="user5@localhost";
   String roomname;
   String nickname;
   String registrationstring="";
   String servererror="";
   /*
   Declare objects of MutableAttributeSet class.
   */
   MutableAttributeSet sendernameattrib;
   /*Declare objects of JPanel class.*/
   JPanel note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
   JPanel combine=new JPanel();
   /* Constructor for class*/
   public LoginGUI ()
   {
      /*set the title for window*/
      super("Chat Room Application");
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib,"Verdana");
      StyleConstants.setForeground(sendernameattrib,Color.blue);
      this.clientsocket=clientsocket;
      c=getContentPane();
      /*Declare objects of JPanel class.*/
      JPanel p=new JPanel();
      JLabel room= new JLabel("User Name");
      JLabel name= new JLabel("Password");
      JLabel serverl= new JLabel("Server IP");
      JLabel note11= new JLabel("* If you are not a registered user then, your account will be creat
      automatically.");
      JLabel note1= new JLabel("* If you are not a registerd user your account will be");
      JLabel note2= new JLabel("created automatically.");
      JPanel lable=new JPanel();
      /*set the layout for window*/
      p.setLayout(new GridLayout(4, 2, 3, 3));
      /*set the border for user*/
      p.setBorder(BorderFactory.createTitledBorder("Login Information"));
      /*Add the component in container*/
      p.add(room);
      p.add(namet);
      p.add(name);
      p.add(passwordt);
```

```
   p.add(serverl);
   p.add(server);
   /*set the color for label*/
   note1.setForeground(Color.red);
   note2.setForeground(Color.red);
   /*add the label to container*/
   note.add(note1);
   note.add(note2);
   /*Declare objects of JLabel class.*/
   JLabel heading=new JLabel("Login Window", JLabel.CENTER);
   lable.add(heading,BorderLayout.NORTH);
   /*set the heading for heading*/
   heading.setFont(new Font("verdana", 1, 12));
   c.add(lable,BorderLayout.NORTH);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   c.add(p,BorderLayout.CENTER);
   /*Declare objects of JPanel class.*/
   JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
   JPanel bp=new JPanel(new FlowLayout());
   bp.add(b);
   buttonpanel.add(note);
   buttonpanel.add(bp);
   c.add(buttonpanel,BorderLayout.SOUTH);
   /*set the bounds for window*/
   setBounds(5, 5, 371, 240);
   /*add the listener to button*/
   b.addActionListener(this);
   b.addKeyListener(this);
   /*called the function to show the window*/
   show();
}
/*function to handle the key event */
public void keyPressed(KeyEvent e)
{
   if(e.getKeyCode()==KeyEvent.VK_ENTER)
   {
      servername=server.getText();
      username=namet.getText();
      password=passwordt.getText();
      MainGUI mn=new MainGUI(username,password,servername,this);
   }
}
/*function to handle the key event*/
public void keyTyped(KeyEvent e)
{
}
/*function to handle the key event*/
public void keyReleased(KeyEvent e)
{
   if(e.getKeyCode()==KeyEvent.VK_ENTER)
   {
      servername=server.getText();
      username=namet.getText();
      password=passwordt.getText();
      MainGUI mn=new MainGUI(username,password,servername,this);
      servererror=mn.servererror;
   }
}
/* function to handle the  event*/
public void actionPerformed(ActionEvent e)
{
   servername=server.getText();
   username=namet.getText();
   password=passwordt.getText();
   try
   {
      MainGUI mn=new MainGUI(username,password,servername,this);
   }
   catch(Exception ee)
   {
      System.out.println("Server not found ::: "+ee);
   }
}
/*function to get username*/
public String getUserName()
{
   return username;
}
/*function to get password*/
public String getPassword()
{
   return password;
}
/* function to get servername*/
public String getServarName()
{
   return servername;
```

```
    }
    public static void main(String[] args)
    {
        /*
        Initialize and set the Look and Feel of the application to Windows Look and Feel.
        */
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        LoginGUI lgui=new LoginGUI();
    }
}
```

Download this listing.

In the above code, the main() method creates an instance of the LoginGUI class.

The methods defined in Listing 3-1 are:

- getUserName(): Retrieves the name of the end user currently logged in.
- getServerName(): Retrieves the name of the Jabber server specified by an end user.
- getPassword(): Retrieves the password specified by an end user.
- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the Login window.

The LoginGUI.java file creates the Login window of the Chat Room application, as shown in Figure 3-2:

Figure 3-2: The Login Window

Creating the User Interface of the Chat Room Application

The MainGUI.java file creates the user interface of the Chat Room application. Listing 3-2 shows the contents of the MainGUI.java file:

Listing 3-2: The MainGUI.java File

```
/*Imports required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net package classes*/
import java.io.*;
import java.net.*;
/*Import required util package classes*/
import java.util.*;
import java.lang.*;
/*Imports required swing package classes*/
import javax.swing.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import java.util.Vector;
/*
class MainGUI - This class is used to create a GUI to display the room list.
Constructor:
PortModem - This constructor creates GUI.
MainGUI:
makeTree - this function is used to create a tree for room list
*/
public class MainGUI extends JFrame implements ActionListener, Runnable, TreeSelectionListener
{
   /*Declare variables*/
   Socket clientsocket1;
   SocketOpener clientsocket;
   /*Declare objects of JMenuBar class*/
   JMenuBar jb=new JMenuBar();
   /*Declare objects of JMenu class*/
   JMenu menu=new JMenu("File");
   JMenuItem menuItem1=new JMenuItem("Go to Chat Room");
   JMenuItem menuItem2=new JMenuItem("Create New Chat Room");
   /*Declare the integer variable*/
   int portno=5222;
   int wait=1000;
   /*Declare the String variable*/
   String firstname="";
   String lastname="";
   String emailid="";
   String ipaddress;
   String errortype="";
   String servername;
   String username;
   String password;
   String resource="Home";
   String  recerverid="user5@manish";
   String roomname;
   String nickname;
   String servererror="";
   /*Declare the object of Button class*/
   Button sendbutton;
   Button closebutton;
   /*Declare the Boolean variable*/
   private  boolean isConnected=false;
   /*Declare objects of Thread class*/
   Thread inputmessagethread;
   /*Declare objects of MutableAttributeSet class*/
   MutableAttributeSet sendernameattrib,recervernameatrib, normaltextattrib;
   /* Declare objects of Document class*/
   Document contentmodel;
   /*Declare objects of JScrollPane class*/
   JScrollPane treeView;
   /*Declare objects of JTree class*/
   JTree tree;
   LoginGUI logingui;
   /*Function to make tree for room list*/
   public void makeTree()
   {
      tree=new JTree();
      DefaultMutableTreeNode top = new DefaultMutableTreeNode("Existing Rooms");
```

```
        DefaultMutableTreeNode regional = new DefaultMutableTreeNode("Regional ");
        DefaultMutableTreeNode culture = new DefaultMutableTreeNode("Culture");
        DefaultMutableTreeNode sports = new DefaultMutableTreeNode("Sports");
        DefaultMutableTreeNode education = new DefaultMutableTreeNode("Education");
        DefaultMutableTreeNode science = new DefaultMutableTreeNode("Science");
        DefaultMutableTreeNode software = new DefaultMutableTreeNode("Software");
        tree = new JTree(top);
        top.add(regional);
        top.add(culture);
        top.add(sports);
        top.add(education);
        top.add(science);
        top.add(software);
        tree.getSelectionModel().setSelectionMode
        (TreeSelectionModel.SINGLE_TREE_SELECTION);
        DefaultMutableTreeNode apolloNode =
        (DefaultMutableTreeNode)top.getFirstChild();
        TreeNode[] pathToRoot = top.getPath();
        TreePath path = new TreePath(pathToRoot);
        tree.expandPath(path);
        tree.addTreeSelectionListener(this);
        treeView = new JScrollPane(tree);
    }
    /*Constructor of class*/
    public MainGUI(String user, String password, String server, LoginGUI lgui)
    {
        super("Chat Room Application");
        /*
        Initialize and set the Look and Feel of the application to Windows Look and Feel.
        */
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        servername=server;
        username=user;
        this.password=password;
        ipaddress=server;
        logingui=lgui;
        SocketOpener.setMainPointer(this);
        SocketOpener.setLogin(logingui);
        jb.add(menu);
        menu.add(menuItem1);
        menu.add(menuItem2);
        this.setJMenuBar(jb);
        menuItem1.addActionListener(this);
        menuItem2.addActionListener(this);
        normaltextattrib=new SimpleAttributeSet();
        makeTree();
        getContentPane().add(treeView);
        setBounds(5,5,350,350);
        setVisible(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        start1();
    }
    public String getRoomName()
    {
        return roomname;
    }
    public String getNickName()
    {
        return nickname;
    }
    /*Function to handle the tree event*/
    public void valueChanged(TreeSelectionEvent e)
    {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode)
        tree.getLastSelectedPathComponent();
        if (node==null) return;
        roomname=node.toString().trim();
        if(!roomname.equals("Existing Rooms"))
        {
            nickname=JOptionPane.showInputDialog(this, "Please specify your user name for this room.");
            if(nickname!=null)
            {
                if(nickname.trim().equals(""))
                {
                    JOptionPane.showMessageDialog(this, "User name should not be blank", "Message",
                    JOptionPane.PLAIN_MESSAGE);
                    nickname=JOptionPane.showInputDialog(this, "Please specify your user name for this
                    room."
                );
                return;
            }
```

```
        }
        if(nickname==null)
        {
           return;
        }
        if((nickname.trim()!=null)||(!nickname.trim().equals("")))
        {
           String roomString="<presence from=\""+username.trim()+"@conference."+servername+"/Home\"
           to=\""+roomname.trim()+"@conference."+servername+"/"+nickname.trim()+"\"/>";
           SocketOpener.sendXMLToJabber(roomString);
        }
     }
  }
  /* Function to open a socket*/
  public void start1()
  {
     try
     {
        openPort(servername,5222,1000);
     }
     catch(Exception e)
     {
     }
     if (clientsocket1!=null)
     {
        isConnected=true;
     }
     Thread inputthread=new Thread(this);
     inputthread.start();
     try
     {
        String sessionStratString;
        String authentication;
        String registrationstring;
        sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
        sessionStratString=sessionStratString+" <stream:stream";
        sessionStratString=sessionStratString+" to= \""+servername+"\"";
        sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
        sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
        clientsocket.sendXMLToJabber(sessionStratString);
        if ((firstname.equals("false"))||(lastname.equals("false"))||(emailid.equals("false")))
        {
        }
        else
        {
           registrationstring="<iq type=\"set\" to=\""+username+"@localhost\" id=\"1001\">";
           registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
           registrationstring=registrationstring+"<username>"+username+"</username>";
           registrationstring=registrationstring+"<password>"+password+"</password>";
           registrationstring=registrationstring+"<first>"+firstname+"</first>";
           registrationstring=registrationstring+"<last>"+lastname+"</last>";
           registrationstring=registrationstring+"<email>"+emailid+"</email>";
           registrationstring=registrationstring+" </query> ";
           registrationstring=registrationstring+"</iq>";
        }
        authentication="<iq type=\"set\" id=\"1301\">";
        authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
        authentication=authentication+" <username>"+username+"</username>";
        authentication=authentication+" <password>"+password+"</password>";
        authentication=authentication+" <resource>"+resource+"</resource> ";
        authentication=authentication+" </query> ";
        authentication=authentication+"</iq>";
        clientsocket.sendXMLToJabber(authentication);
     }
     catch(Exception ie)
     {
     }
  }
  /*run function to handle the activity by thread*/
  public void run()
  {
     clientsocket.runinput();
  }
  /*Function to open a socket*/
  private void openPort(String ipaddress,int portno,int timeinsec)
  {
     String host;
     if (ipaddress.trim()=="")
     {
     }
     else
     {
        SocketOpener sc=new SocketOpener(ipaddress, portno);
        clientsocket=sc.openSocket(ipaddress, portno, timeinsec);
     }
  }
  /* Function to handle the event*/
```

```
public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==menuItem1)
   {
      ChatRoom r=new ChatRoom(clientsocket1, this);
   }
   if(e.getSource()==menuItem2)
   {
      RoomInformation r=new RoomInformation(clientsocket1, this);
   }
}
}
```

Download this listing.

In the above code, the constructor of the MainGUI class takes an object of the LoginGUI class and three strings as input parameters: user, password, and server. The object of the LoginGUI class allows an end user to invoke the methods of the LoginGUI class. The user string retrieves the user name of the end user. The password string retrieves the password of the end user, and the server string retrieves the ip address of the Jabber server to connect.

The methods defined in Listing 3-2 are:

- `makeTree()`: Maintains a tree structure of the chat rooms listed in the user interface.
- `valueChanged()`: Retrieves the name of the chat room selected by an end user from the various chat rooms listed in the user interface.
- `openPort ()`: Opens a socket to send and receive messages.
- `getRoomName()`: Retrieves the name of the chat room specified by an end user to participate in a group chat.
- `getNickName()`: Retrieves the name of an end user to initiate the chat.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by selecting any option from the user interface.

The MainGUI.java file creates the main window of the Chat Room application, as shown in Figure 3-3:

Figure 3-3: User Interface of the Chat Room Application

Select the File menu to view the File menu options.

Figure 3-4 shows the File menu options of the Chat Room application:

Figure 3-4: The File Menu of the Chat Room Application

Select File->Go to Chat Room to enter an existing chat room.

Select File->Create New Chat to create a new chat room.

Select any chat room from the Existing Rooms tree structure to initiate a chat.

PREVIOUS NEXT

Joining a Chat Room

The ChatRoom.java file creates the user interface to enter into an existing chat room to initiate a chat. Listing 3-3 shows the contents of the ChatRoom.java file:

Listing 3-3: The ChatRoom.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*
class ChatRoom-This class is used to create a GUI to take the room name and nick from user.
Constructor:
ChatRoom-This constructor creates GUI.
Methods:
getRoomName-to get room name
getNickName-to get nick name
*/
public class ChatRoom extends JDialog implements ActionListener
{
   /*Declare objects of Container class*/
   Container c=null;
   /*Declare objects of JTextField class*/
   JTextField roomt=new JTextField();
   JTextField namet=new JTextField();
   /*Declare objects of JButton class*/
   JButton b=new JButton("Submit");
   /*Declare objects of MainGUI class*/
   MainGUI maingui;
   /*Declare objects of Socket class*/
   Socket clientsocket;
   static Socket clientsocket1;
   /*Declare String variables*/
   String servername="localhost";
   String password="user4";
   String resource="Home";
   String  recerverid="user5@localhost";
   String roomname;
   String nickname;
   String username;
   /*Declare objects of MutableAttributeSet class*/
   MutableAttributeSet sendernameattrib;
   /*Constructor for class*/
   public ChatRoom(Socket clientsocket, MainGUI maingui)
   {
      setTitle("Chat Room Application");
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib,"Verdana");
      StyleConstants.setForeground(sendernameattrib,Color.blue);
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
      }
      this.clientsocket =clientsocket;
      c=getContentPane();
      this.maingui=maingui;
      /*Declare objects of JPanel class*/
      JPanel p = new JPanel();
      JLabel room = new JLabel("Room Name");
      JLabel name = new JLabel("User Name");
      JPanel lable = new JPanel();
      /*set the layout*/
      p.setLayout(new GridLayout(3, 2, 10, 10));
      /*set the border to panel*/
      p.setBorder(BorderFactory.createTitledBorder("Chat Room"));
      /*add the component to container*/
      p.add(room);
      p.add(roomt);
      p.add(name);
      p.add(namet);
      /*Declare objects of JLabel class*/
      JLabel heading=new JLabel("Chat Room", JLabel.CENTER);
      /*set the font heading*/
      heading.setFont(new Font("verdana", 1, 12));
      lable.add(heading, BorderLayout.NORTH);
```

```
        /*add panel to container*/
        c.add(lable,BorderLayout.NORTH);
        /*set default close operation for window*/
        setDefaultCloseOperation(1);
        /* set resizable true to show the window*/
        setResizable(false);
        c.add(p,BorderLayout.CENTER);
        /*Declare objects of JPanel class*/
        JPanel buttonpanel=new JPanel(new FlowLayout());
        /*add button to container*/
        buttonpanel.add(b);
        c.add(buttonpanel,BorderLayout.SOUTH);
        /*set bounds for window*/
        setBounds(5, 5, 300, 200);
        /*add action listener to button*/
        b.addActionListener(this);
        /*calls the show function to show the window*/
        show();
    }
    /*function to handle the event*/
    public void actionPerformed(ActionEvent e)
    {
        if((JButton)e.getSource()==b)
        {
            if(roomt.getText()==null||roomt.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please specify the room name." );
                return;
            }
            if(namet.getText()==null||namet.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please specify the user name." );
                return;
            }
            String dialog=null;
            roomname=roomt.getText();
            nickname=namet.getText();
            this.username=maingui.username;
            String roomString="<presence from=\""+username+"@conference.localhost/Home\"
            to=\""+roomname.trim()+"@conference."+servername+"/"+nickname.trim()+"\"/>";
            SocketOpener.sendXMLToJabber(roomString);
            SocketOpener.setChatPointer(this, maingui);
            return;
        }
    }
    /*function to get the room name*/
    public String getRoomName()
    {
        return roomname;
    }
    /*function to get the nick name*/
    public String getNickName()
    {
        return nickname;
    }
}
```

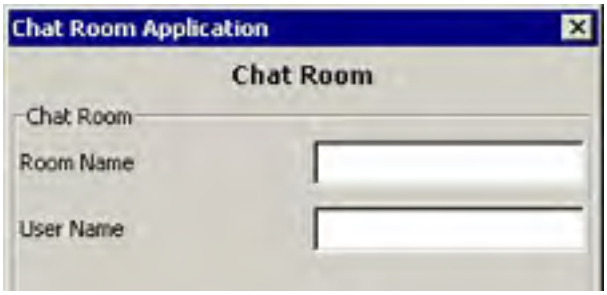
Download this listing.

In the above code, the constructor of the ChatRoom class takes an object of the MainGUI class and the Socket class as input parameters. The ChatRoom class allows end users to invoke the methods of the MainGUI class and the Socket class.

The methods defined in Listing 3-3 are:

- getRoomName(): Retrieves the name of the chat room specified by an end user.
- getNickName(): Retrieves the user name of an end user for the selected chat room.
- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the user interface.

The ChatRoom.java file creates the Chat Room window that allows end users to participate in a chat, as shown in Figure 3-5:



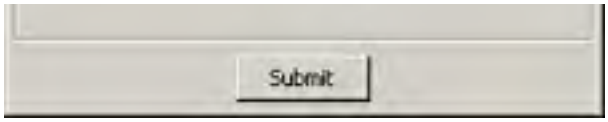


Figure 3-5: Joining a Chat Room

◄ PREVIOUS NEXT ►

Creating a Chat Room

The RoomInformation.java file allows end users to create a chat room. Listing 3-4 shows the contents of the RoomInformation.java file:

Listing 3-4: The RoomInformation.java File

```
/*Imports required swing package classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Imports required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required io and net package classes*/
import java.io.*;
import java.net.*;
/*
class RoomInformation - This class is used to create a GUI to take the room information from user.
Constructor:
RoomInformation - This constructor creates GUI.
Methods:
getRoomName - to get room name
getNickName - to get nick name
*/
public class RoomInformation extends JDialog implements ActionListener
{
   /*Declare objects of Container class*/
   Container c=null;
   /*Declare objects of JTextField class*/
   JTextField roomt=new JTextField();
   JTextField namet=new JTextField();
   /* Declare objects of JButton class*/
   JButton b=new JButton("Submit");
   /*Declare objects of MainGUI class*/
   MainGUI maingui;
   /*Declare objects of Socket class*/
   Socket clientsocket;
   static Socket clientsocket1;
   /*Declare String variables*/
   String servername="localhost";
   String password="user4";
   String resource="Home";
   String recerverid="user5@localhost";
   String roomname;
   String nickname;
   String username;
   /* Declare objects of MutableAttributeSet class*/
   MutableAttributeSet sendernameattrib;
   /*Constructor for class*/
   public RoomInformation(Socket clientsocket, MainGUI maingui)
   {
      setTitle("Chat Room Application");
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib, "Verdana");
      StyleConstants.setForeground(sendernameattrib, Color.blue);
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      this.clientsocket = clientsocket;
      c=getContentPane();
      this.maingui=maingui;
      /*Declare objects of JPanel class*/
      JPanel p=new JPanel();
      JLabel room = new JLabel("Room Name");
      JLabel name = new JLabel("User Name");
      JPanel lable = new JPanel();
      /*set the layout*/
      p.setLayout(new GridLayout(3, 2, 10, 10));
      /*set the border to panel*/
      p.setBorder(BorderFactory.createTitledBorder("Room Information"));
      /*add the component to container*/
      p.add(room);
      p.add(roomt);
      p.add(name);
      p.add(namet);
      /*Declare objects of JLabel class*/
      JLabel heading=new JLabel("Create Room", JLabel.CENTER);
      /*set the font heading*/
```

```
        heading.setFont(new Font("verdana", 1, 12));
        lable.add(heading, BorderLayout.NORTH);
        c.add(lable,BorderLayout.NORTH);
        /*set default close operation for window*/
        setDefaultCloseOperation(1);
        /*set resizable true to show the window*/
        setResizable(false);
        /*add panel to container*/
        c.add(p,BorderLayout.CENTER);
        /*Declare objects of JPanel class*/
        JPanel buttonpanel=new JPanel(new FlowLayout());
        /*add button to container*/
        buttonpanel.add(b);
        /*Add button panel to container*/
        c.add(buttonpanel,BorderLayout.SOUTH);
        /*set bounds for window*/
        setBounds(5, 5, 300, 200);
        /*add action listener to button*/
        b.addActionListener(this);
        /*calls the show function to show the window*/
        show();
    }
    /*function to handle the event*/
    public void actionPerformed(ActionEvent e)
    {
        if((JButton)e.getSource()==b)
        {
            if(roomt.getText()==null||roomt.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please Insert the room Name." );
                return;
            }
            if(namet.getText()==null||namet.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please specify the user name." );
                return;
            }
            String dialog=null;
            roomname=roomt.getText();
            nickname=namet.getText();
            this.username=maingui.username;
            String roomString="<presence from=\""+username+"@conference.localhost/Home\"
            to=\""+roomname.trim()+"@conference."+servername+"/"+nickname.trim()+"\"/>";
            SocketOpener.sendXMLToJabber(roomString);
            SocketOpener.setRoomPointer(this,maingui);
            return;
        }
    }
    /*function to get the room name*/
    public String getRoomName()
    {
        return roomname;
    }
    /*function to get the nick name*/
    public String getNickName()
    {
        return nickname;
    }
}
```

Download this listing.

In the above code, the constructor of the RoomInformation class takes an object of the MainGUI class and the Socket class as input parameters. The RoomInformation class allows end users to invoke the methods of the MainGUI class and the Socket class.

The methods defined in Listing 3-4 are:

- `getRoomName()`: Retrieves the name of the chat room specified by an end user to create a chat room.
- `getNickName()`: Retrieves the user name of an end user for the created chat room.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the user interface.

The RoomInformation.java file creates the Create Room window that allows end users to create a chat room, as shown in Figure 3-6:

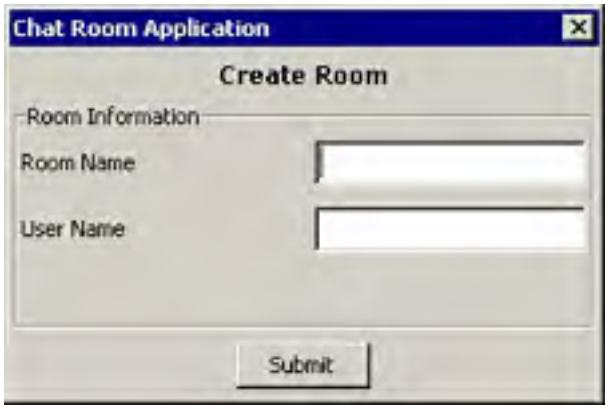


Figure 3-6: Creating a Chat Room

PREVIOUS NEXT

Creating the SocketOpener Class to Send and Receive Messages

The SocketOpener.java file creates a socket with the Jabber server to send and receive messages between end users. Listing 3-5 shows the contents of the SocketOpener.java file:

Listing 3-5: The SocketOpener.java File

```
/*Imports required swing package classes.*/
import javax.swing.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
/*Import required awt package classes.*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net package classes.*/
import java.io.*;
import java.net.*;
/*Import required util package classes.*/
import java.util.*;
import java.lang.*;
/*
class SocketOpener - This class is used for socket opening.
Constructor:
SocketOpener - This constructor accept host name and IP.
Methods:
openSocket - open the socket
getErrorString - get the error string send by server
writemessage-used to write the message in the textpane.
*/
public class SocketOpener
{
   /*Declare the int variables.*/
   private int openerport;
   /* Declare objects of Socket classes.*/
   static private Socket socket;
   static MutableAttributeSet sendernameattrib, nameattribut, informationattrib, welcomenameattrib;
   static Document contentmodel;
   /*Declare objects of Stream classes.*/
   static PrintWriter out = null;
   static BufferedReader in = null;
   /*Declare boolean variable.*/
   private  static boolean isConnected=false;
   /*Declare objects of Hasmap and vector class.*/
   Vector nicknamevector=new Vector();
   HashMap temphashmap=new HashMap();
   /*Declare objects of GroupChat class.*/
   static GroupChat gc;
   /*Declare objects of RoomInformation class.*/
   static RoomInformation socketroomgui;
   /*Declare objects of ChatRoom class.*/
   static ChatRoom socketchatmgui;
   /*Declare objects of MainGUI class.*/
   static MainGUI socketmaingui;
   /*Declare objects of LoginGUI class.*/
   static LoginGUI logingui;
   /*Declare the string variable*/
   static String errortype="";
   private String openerhost;
   String roomname="";
   String firstname="";
   String lastname="";
   String emailid="";
   String resource="Home";
   String temproomname="";
   Color textcolor;
   String nickname="";
   String servererror="";
   /* function to open socket*/
   public  SocketOpener openSocket(String hostip, int portnumber, int timeinsec)
   {
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib, "Verdana");
      StyleConstants.setForeground(sendernameattrib, Color.black);
      welcomenameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(welcomenameattrib, "Verdana");
      StyleConstants.setForeground(welcomenameattrib, Color.blue);
      nameattribut=new SimpleAttributeSet();
      StyleConstants.setFontFamily(nameattribut,"Verdana");
      StyleConstants.setForeground(nameattribut, Color.red);
      informationattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(informationattrib, "Verdana");
      StyleConstants.setItalic(informationattrib,true);
      StyleConstants.setForeground(informationattrib, Color.red);
      socketope();
```

```
   try
   {
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
   }
   catch(Exception exception)
   {
      JOptionPane.showMessageDialog(logingui, "Server not found", "Server not found", JOptionPane
      socketmaingui.servererror="Server error";
   }
   return getSocketClass();
}
/*Function to initialize the Socket class*/
public  void socketope()
{
   try
   {
      socket=new Socket(openerhost, openerport);
   }
   catch(IOException ie)
   {
   }
}
/*Constructor for SocketOpener class*/
public SocketOpener(String host, int port)
{
   socket=null;
   openerhost=host;
   openerport=port;
}
/*
Function returns the object of SocketOpener class
*/
public SocketOpener getSocketClass()
{
   return this;
}
/*Function to write a message*/
public void writemessage(String messagebody, MutableAttributeSet attribut)
{
   try
   {
      gc.writeintotextfield(messagebody, attribut);
   }
   catch(Exception ble)
   {
   }
}
/* Function returns the error string send by server*/
public static String getErrorString(String codeid)
{
   if (codeid=="'401'")
   {
      errortype="minor";
      return "You have sent malformed syntax which can't be understood by server.";
   }
   if (codeid=="'404'")
   {
      errortype="major";
      return "No Server exists.";
   }
   if (codeid=="'405'")
   {
      errortype="minor";
      return "You have no right to send this command.";
   }
   if (codeid=="'409'")
   {
      errortype="major";
      return "A user with this jabber id is already exist. Please choose another user id.";
   }
   if (codeid=="'403'")
   {
      errortype="auth";
      return "Invalid password";
   }
   return "";
}
public void sendername(String tagentity)
{
   String sendername="";
   String newuserid="";
   tagentity=tagentity.trim();
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("id=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("id="));
```

```
                gc.writeintotextfield("\n"+sendername,nameattribut);
            }
        }
        if (tagentity.startsWith("message"))
        {
            if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
            {
                sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to="));
                if(!nickname.trim().equalsIgnoreCase(sendername.trim()))
                {
                    gc.writeintotextfield("\n"+sendername.substring(sendername.indexOf("/")+1,
                    sendername.indexOf("'"))+":",nameattribut);
                }
                else
                {
                    gc.writeintotextfield("\n"+sendername.substring(sendername.indexOf("/")+1,
                    sendername.indexOf("'"))+": ",sendernameattrib);
                }
            }
        }
        if (tagentity.startsWith("presence"))
        {
            if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
            {
                nickname=tagentity.substring(tagentity.indexOf("from=")+6, tagentity.indexOf("to=")-1);
                if (tagentity.indexOf("type='unavailable'")>0||tagentity.indexOf("type=\"unavailable\"")
                {
                    gc.removeuser(nickname.substring(nickname.indexOf("/")+1, nickname.indexOf("'")));
                    String nickname1="\n"+nickname.substring(nickname.indexOf("/")+1,
                    nickname.indexOf("'"))+"  has left the room";
                    gc.writeintotextfield(nickname1, informationattrib);
                }
                else
                {
                if (gc!=null)
                {
                    gc.addNewUser(nickname.substring(nickname.indexOf("/")+1,nickname.indexOf("'")));
                    String nickname1="\n"+nickname.substring(nickname.indexOf("/")+1,
                    nickname.indexOf("'"))+"  has joined the room";
                    gc.writeintotextfield(nickname1, informationattrib);
                }
                else
                {
                    nicknamevector.add(nickname.substring(nickname.indexOf("/")+1,nickname.indexOf("'")))
                }
            }
        }
    }
    if (tagentity.startsWith("item"))
    {
        if (tagentity.indexOf("jid")<0&&(tagentity.indexOf("affiliation='none'")>0)
        &&(tagentity.indexOf("role='participant'")>0))
        {
            if (socketmaingui!=null)
            {
                temproomname=socketmaingui.getRoomName();
                nickname=socketmaingui.getNickName();
            }
            if (socketchatmgui!=null)
            {
                temproomname=socketchatmgui.getRoomName();
                nickname=socketchatmgui.getNickName();
                gc=new GroupChat(temproomname,nickname,openerhost);
                socketchatmgui.hide();
                gc.setUserName(logingui.getUserName());
                for (int i=0;i<nicknamevector.size();i++)
                {
                    gc.addNewUser((String)nicknamevector.get(i));
                }
                gc.hashmap.putAll(temphashmap);
                roomname=temproomname;
                gc.writeintotextfield("Welcome to " +roomname+ "  room ", welcomenameattrib);
            }
            if (temproomname.equals("Regional")||temproomname.equals("Culture")||
            temproomname.equals("Sports")||temproomname.equals("Education")||
            temproomname.equals("Science")||temproomname.equals("Software"))
            {
                if (gc==null)
                {
                gc=new GroupChat(temproomname, nickname, openerhost);
                gc.setUserName(logingui.getUserName());
                for (int i=0;i<nicknamevector.size();i++)
                {
                    gc.addNewUser((String)nicknamevector.get(i));
                }
                gc.hashmap.putAll(temphashmap);
                roomname=temproomname;
                gc.writeintotextfield("Welcome to " +roomname+ " room ", welcomenameattrib);
```

```
            }
            else
            {
               if (!gc.isVisible())
               {
                  gc.clearPane();
                  gc.writeintotextfield("Welcome to " +roomname+ " room ", welcomenameattrib);
                  gc.setVisible(true);
               }
            }
         }
          if(temproomname.equals("Regional")||temproomname.equals("Culture")||
          temproomname.equals("Sports")||temproomname.equals("Education")||
          temproomname.equals("Science")||
          temproomname.equals("Software"))
         {
         }
         else
         {
            temphashmap.clear();
            nicknamevector.clear();
            if (socketchatmgui==null)
            {
               JOptionPane.showMessageDialog(null, "Room already exists.");
            }
         }
         return;
      }
      if (tagentity.indexOf("jid=")>0&&tagentity.indexOf("affiliation=")>0)
      {
         newuserid=tagentity.substring(tagentity.indexOf("jid=")+5,tagentity.indexOf("affiliation=")-2)
         if(newuserid.equals(logingui.getUserName()+"@"+logingui.getServarName().trim()+"/Home"))
         {
            if (tagentity.indexOf("affiliation='owner'")>0||tagentity.indexOf
            ("affiliation=\"owner\"")>0)
            {
               if (socketroomgui!=null)
               {
                  socketroomgui.hide();
                  roomname=socketroomgui.getRoomName().trim();
                  nickname=socketroomgui.getNickName().trim();
               }
               else
               {
               try
               {
                  roomname=socketmaingui.getRoomName().trim();
                  nickname=socketmaingui.getNickName().trim();
               }
               catch(Exception e)
               {
                  JOptionPane.showMessageDialog(socketroomgui, "Room does not  exist");
                  return;
               }
            }
            gc=new GroupChat(roomname,nickname,openerhost);
            gc.setUserName(logingui.getUserName());
            for (int i=0;i<nicknamevector.size();i++)
            {
               gc.addNewUser((String)nicknamevector.get(i));
            }
            gc.hashmap.putAll(temphashmap);
            gc.writeintotextfield("Welcome to " +roomname+ "  room", welcomenameattrib);
            }
            else
            {
               if (roomname.equals("Regional")||roomname.equals("Culture")||roomname.equals
               ("Sports")||roomname.equals("Education")||roomname.equals("Science")
               ||roomname.equals("Software"))
               {
                  gc=new GroupChat(roomname, nickname, openerhost);
                  gc.setUserName(logingui.getUserName());
               }
               else
               {
                  if (gc!=null)
                  {
                  if (gc.isVisible())
                  {
                     gc.hide();
                     JOptionPane.showMessageDialog(null, "Room is already exist");
                     return;
                  }
               }
            }
            if (!socketroomgui.isVisible())
            {
               socketroomgui.setVisible(true);
```

```
            }
         }
      }
      if (tagentity.indexOf("role='none'")>0||tagentity.indexOf("role=\"none\"")>0)
      {
         if (!newuserid.equals(""))
         {
            gc.removeUserinHash(nickname,newuserid);
         }
      }
      else
      {
         if (!newuserid.equals(""))
         {
            if (gc!=null)
            {
               gc.addNewUserinHash(nickname,newuserid);
            }
            else
            {
               temphashmap.put(nickname,newuserid);
            }
         }
      }
   }
   }
   }
   /*
   Function to set the object of RoomInformation and MainGUI class
   */
   public static void setRoomPointer(RoomInformation room,MainGUI maingui)
   {
      socketroomgui=room;
      socketmaingui=maingui;
   }
   /*
   Function to set the object of ChatRoom and MainGUI class
   */
   public static void setChatPointer(ChatRoom chatroom,MainGUI maingui)
   {
      socketchatmgui=chatroom;
      socketmaingui=maingui;
   }
   /*Function to set the object of MainGUI class*/
   public static void setMainPointer(MainGUI maingui)
   {
      socketmaingui=maingui;
   }
   /*Function to set the object of LoginGUI class*/
   public static void setLogin(LoginGUI lgui)
   {
      logingui=lgui;
   }
   /*Function to check xml error*/
   public static String checkForError(String tagentity)
   {
      String error="";
      String codeid="";
      if (tagentity.indexOf("error")>0)
      {
      if (tagentity.indexOf("from=")>0)
      {
         codeid=tagentity.substring(tagentity.indexOf("id=")+3, tagentity.indexOf("type="));
         return "Invalid password.";
      }
      return "No user exists for this user ID.";
   }
   if (tagentity.indexOf("result")>0)
   {
      if(!socketmaingui.isVisible())
   {
      socketmaingui.setVisible(true);
      logingui.setVisible(false);
   }
   }
   return error;
   }
   /*Function to read the message from socket*/
   public void runinput()
   {
      int i=0;
      String errororwarning="";
      String errorstring="";
      String inputstring="";
      boolean starttag=false;
      boolean endtag=false;
```

```
boolean startwritting=false;
String messagebody="";
String tagentity="";
Vector tagvactor;
tagvactor=new Vector();
int vactorindex=0;
int starttagsing=0;
while (true)
{
   try
   {
      i=in.read();
      if (i==-1)
      {
         break;
      }
      else
      {
         inputstring=inputstring+(char)i;
         if ((char)i=='<')
         {
            starttag=true;
            starttagsing=1;
            tagentity="";
         }
         else
         {
            if ((char)i=='/'&& starttag==true &&starttagsing==1 )
            {
               starttag=false;
               endtag=true;
               if (!messagebody.trim().equals(""))
               {
                  writemessage(messagebody,sendernameattrib);
               }
               startwritting=false;
               messagebody="";
               vactorindex=vactorindex-1;
               if (vactorindex>=0)
               tagvactor.removeElementAt(vactorindex);
            }
            else
            {
            starttagsing=0;
            if((char)i=='>')
            {
               if (starttag)
               {
                  sendername(tagentity);
                  errorstring=checkForError(tagentity);
                  if (errorstring.equals("Invalid password."))
                  {
                     JOptionPane.showMessageDialog(logingui, "Invalid password", "Invalid
                     password",JOptionPane.ERROR_MESSAGE);
                     return;
                  }
                  if (!errorstring.equals(""))
                  {
                     String username=logingui.getUserName();
                     String password=logingui.getPassword();
                     String servarn=logingui.getServarName();
                     String registrationstring="<iq type=\"set\" to=\""+username+"@"+servarn+"\"
                     id=\"1001\">";
                     registrationstring=registrationstring+"<query
                     xmlns=\"jabber:iq:register\">";
                     registrationstring=registrationstring+"<username>"+username+
                     "</username>";registrationstring=registrationstring+"<password>
                     "+password+"</password>";
                      registrationstring=registrationstring+"<first>
                      "+firstname+"</first>";
                     registrationstring=registrationstring+"<last>"+lastname+"</last>";
                     registrationstring=registrationstring+"<email>"+emailid+"</email>";
                     registrationstring=registrationstring+" </query> ";
                     registrationstring=registrationstring+"</iq>";
                     String authentication="<iq type=\"set\" id=\"1301\">";
                     authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
                     authentication=authentication+" <username>"+username+"</username>";
                     authentication=authentication+" <password>"+password+"</password>";
                     authentication=authentication+" <resource>"+resource+"</resource>";
                     authentication=authentication+" </query> ";
                     authentication=authentication+"</iq>";
                     if(errortype=="auth")
                     {
                        errororwarning="Error";
                     }
                     if(errortype=="major")
                     {
                        errororwarning="Error";
```

```
                                JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                                JOptionPane.PLAIN_MESSAGE);
                                sendXMLToJabber(registrationstring);
                                sendXMLToJabber(authentication);
                                if (socketmaingui!=null)
                                {
                                   socketmaingui.hide();
                                }
                             }
                             else
                             {
                                errororwarning="Warning";
                                sendXMLToJabber(registrationstring);
                                sendXMLToJabber(authentication);
                                if (socketmaingui!=null)
                                {
                                   socketmaingui.hide();
                                }
                             }
                          }
                          startwritting=true;
                          tagvactor.insertElementAt(tagentity,vactorindex);
                          vactorindex=vactorindex+1;
                          starttag=false;
                       }
                    }
                    else
                    {
                       if (startwritting==true&&tagentity.trim().equals("body"))
                       {
                          messagebody=messagebody+(char)i;
                          messagebody=messagebody.trim();
                       }
                       if (starttag)
                       {
                          tagentity=tagentity+(char)i;
                       }
                    }
                 }
              }
           }
        }
     }
     catch(IOException ie)
     {
        JOptionPane.showMessageDialog(socketchatmgui, "Room not found", "Room not
        found", JOptionPane.ERROR_MESSAGE);
        return;
     }
}
isConnected=false;
}
/*Function to send xml to server*/
public static void sendXMLToJabber(String outputmessage)
{
   out.println(outputmessage);
   out.flush();
}
/*Function to set the object of GroupChat class*/
public static void setgc(GroupChat groupchat)
{
   gc=groupchat;
}
}
```

Download this listing.

In the above code, the constructor of the SocketOpener class takes host and port as input parameters. The host string retrieves the name of the Jabber server and the port integer retrieves the port number to open the socket with the Jabber server.

The methods defined in Listing 3-5 are:

- `openSocket()`: Opens the socket to send and receive messages.
- `getErrorString()`: Retrieves the error string sent by the Jabber server.
- `writemessage()`: Writes the text message in the text area of the Group Chat window.
- `sendXMLToJabber()`: Sends an XML (eXtensible Markup Language) message to the Jabber server.
- `setgc()`: Sets the object of the GroupChat class.
- `runinput()`: Listens for the server response sent by the Jabber server.
- `checkForError()`: Checks whether or not the input string contains an error message.
- `setLogin()`: Sets the object of the LoginGUI class.

- `setChatPointer()`: Sets the objects of the ChatRoom and MainGUI classes.
- `setRoomPointer()`: Sets the objects of the RoomInformation and MainGUI classes.
- `sendername()`: Retrieves the sender's name from the input message received from the Jabber server.
- `getErrorString()`: Retrieves the message sent by the Jabber server.
- `socketope()`: Initializes the Socket class to invoke the methods of the Socket class.

◄ PREVIOUS NEXT ►

Creating a Group Chat Window

The GroupChat.java file creates the Group Chat window of the chat room selected by an end user to send messages to all end users in the chat room. Listing 3-6 shows the contents of the GroupChat.java file:

Listing 3-6: The GroupChat.java File

```
/*Import required swing package classes*/
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.*;
/*Import required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required util package classes*/
import java.util.*;
public class  GroupChat extends JFrame implements ActionListener, KeyListener
{
   /*Declare object of Container class.*/
   Container cont = null;
   /*Declare object of JMenuBar class.*/
   JMenuBar jb=new JMenuBar();
   JMenu menu=new JMenu("File");
   /*Declare object of JMenuItem class.*/
   JMenuItem menuItem1=new JMenuItem("Got To Chat Room");
   JMenuItem menuItem2=new JMenuItem("Create New Chat Room");
   /*Declare object of JPanel class.*/
   JPanel jp=new JPanel(new BorderLayout());
   /*Declare object of JButton class.*/
   JButton button=new JButton("Send");
   JButton colorb=new JButton("Text Color");
   /*Declare object of JTextField class.*/
   JTextField text=new JTextField();
   JList list;
   JColorChooser chooser=new JColorChooser();
   JDialog dialog;
   private DefaultListModel listModel;
   JTextPane je=new JTextPane();
   HashMap hashmap;
   /*Declare object of JScrollPane class.*/
   JScrollPane scrollPane3 = new JScrollPane(text);
   JScrollPane scrollPane1 = new JScrollPane(je);
   JScrollPane scrollPane;
   JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
   /*Declare the string variable*/
   String chatText=null;
   String roomname;
   String nickname;
   String servername;
   String leftmessage="";
   String username="";
   /*Declare object of Document class.*/
   Document contentmodel;
   /*Declare object of MutableAttributeSet class.*/
   MutableAttributeSet nicknameattrib;
   /*Function to clear the text from textpane*/
   public void clearPane()
   {
      je.setText("");
   }
   /* Constructor for class*/
   public GroupChat(String proomname,String pnickname,String pservername)
   {
      /*set the window title*/
      super(" Chat Room Application: "+proomname);
      cont = getContentPane();
      je.setEditable(false);
      nicknameattrib=new SimpleAttributeSet();
      /*set the attribute for text*/
      StyleConstants.setFontFamily(nicknameattrib,"Verdana");
      StyleConstants.setForeground(nicknameattrib,Color.black);
      contentmodel=je.getDocument();
      roomname=proomname;
      nickname=pnickname;
      servername=pservername;
      hashmap=new HashMap();
      listModel=new DefaultListModel();
      list=new JList(listModel);
      scrollPane= new JScrollPane(list);
      list.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
      list.setVisibleRowCount(5);
      jb.add(menu);
      menu.add(menuItem1);
      menu.add(menuItem2);
```

```
   Dimension minimumSize = new Dimension(300, 150);
   splitPane.setTopComponent(scrollPane1);
   splitPane.setBottomComponent(scrollPane);
   splitPane.setDividerLocation(400);
   splitPane.setPreferredSize(new Dimension(700, 200));
   list.setSelectionBackground(Color.cyan);
   Dimension d=button.getPreferredSize();
   button.setPreferredSize(new Dimension(130, 20));
   colorb.setPreferredSize(new Dimension(130, 20));
   text.setPreferredSize(new Dimension(400, 20));
   jp.add(text, BorderLayout.WEST);
   jp.add(button, BorderLayout.EAST);
   cont.add(jp, BorderLayout.SOUTH);
   cont.add(splitPane, BorderLayout.CENTER);
   setBounds(5, 5, 550, 550);
   getRootPane().show();
   menuItem1.addActionListener(this);
   menuItem2.addActionListener(this);
   button.addActionListener(this);
   button.addKeyListener(this);
   text.addKeyListener(this);
   colorb.addActionListener(this);
   this.addWindowListener(windowListener);
   this.show();
}
/*
Function to write the message in to the text pane
*/
public void writeintotextfield(String writeintotextfield, MutableAttributeSet psendernameattrib)
{
   try
   {
      contentmodel.insertString(contentmodel.getLength(),writeintotextfield,psendernameattrib);
   }
   catch(Exception e)
   {
   }
}
/*Inner class to handle window event*/
WindowListener windowListener = new WindowAdapter()
{
   public void windowClosing (WindowEvent w)
   {
      leftmessage="<presence type=\"unavailable\" from=\""+username+"@"+servername+"/Home\"
      to=\""+roomname+"@conference."+servername+"/"+nickname+"\"></presence>";
      SocketOpener.sendXMLToJabber(leftmessage);
      listModel.clear();
      clearPane();
      setVisible(false);
   }
   public void windowOpened(WindowEvent e)
   {
      leftmessage="<presence type=\"available\" from=\""+username+"@"+servername+"/Home\"
      to=\""+roomname+"@conference."+servername+"/"+nickname+"\"></presence>";
      SocketOpener.sendXMLToJabber(leftmessage);
      setVisible(true);
   }
};
/*Function to handle key event*/
public void keyPressed(KeyEvent e)
{
   if(e.getKeyCode()==KeyEvent.VK_ENTER)
   {
      String sendxmlstring="";
      chatText=text.getText();
      if (chatText.trim().equals(""))
      {
      }
      else
      {
         sendxmlstring="<message to=\""+roomname+"@conference."+servername+"\"
         from=\""+username+"@conference.localhost/Home\" type=\"groupchat\">";
         sendxmlstring=sendxmlstring+"<body>"+chatText+"</body></message>";
         SocketOpener.sendXMLToJabber(sendxmlstring);
      }
      text.setText("");
      }
   }
   /*Function to handle key event*/
   public void keyTyped(KeyEvent e)
   {
   }
   /*Function to handle key event*/
   public void keyReleased(KeyEvent e)
   {
   }
   /* Function to handle  event*/
```

```
        public void actionPerformed(ActionEvent e)
        {
           boolean modal = false;
           if(e.getSource()==button)
           {
              String sendxmlstring="";
              chatText=text.getText();
              if (chatText.trim().equals(""))
              {
              }
              else
              {
                 sendxmlstring="<message to=\""+roomname+"@conference."+servername+"\"
                 from=\""+username+"@conference.localhost/Home\" type=\"groupchat\">";
                 sendxmlstring=sendxmlstring+"<body>"+chatText+"</body></message>";
                 SocketOpener.sendXMLToJabber(sendxmlstring);
              }
              text.setText("");
           }
           if(e.getSource()==menuItem2)
           {
              String dialog=JOptionPane.showInputDialog(this, "Please Insert the New Room Name.");
              return;
           }
        }
        /*Function to add user in a list*/
        public void addNewUser(String newuser)
        {
           int index=0;
           String newusersearch="<html><body ><font color=red
           face=verdana>"+newuser+"</body></html>";
           String usersearch="<html><body><font color=black
           face=verdana>"+newuser+"</body></html>";
           if (!((listModel.contains((Object)newusersearch))||(listModel.contains((Object)usersearch))
           {
              if (newuser.equals(nickname))
              {
                 listModel.insertElementAt("<html><body ><font color=red
                 face=verdana>"+newuser+"</body></html>", index);
              }
              else
              {
                 listModel.insertElementAt("<html><body><font color=black
                 face=verdana>"+newuser+"</body></html>", index);
              }
           }
        }
        /* Function to remove user from the list*/
        public void removeuser(String username)
        {
        String rusername="<html><body><font color=black
        face=verdana>"+username+"</body></html>";
        if (listModel.contains((Object)rusername))
        {
           boolean n=listModel.removeElement((Object)rusername);
        }
     }
     /* Function to add user in hashmap*/
     public void addNewUserinHash(String nickname,String newuserid)
     {
        hashmap.put(nickname,newuserid);
     }
     /* Function to remove user in hashmap*/
     public void removeUserinHash(String nickname,String newuserid)
     {
        hashmap.remove(nickname);
     }
     /* Function to set username*/
     public void setUserName(String uname)
     {
        username=uname;
     }
}
```

Download this listing.

In the above code, the constructor of the GroupChat class takes three strings as input parameters: proomname, pnickname, and pservername. The proomname string retrieves the name of the chat room specified by an end user. The pnickname string retrieves the user name of an end user, and the pservername string retrieves the name of the Jabber server.

The methods defined in Listing 3-6 are:

- writeintotextfield(): Writes the message text in the text area of the Group Chat window.
- keyPressed(): Acts as a key listener and activates an appropriate method based on the key pressed by an end user.

- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the Group Chat window.
- clearPane(): Refreshes the text area every time an end user opens the Group Chat window of any chat room.
- addNewUser(): Adds a new end user to the list of existing end users in a chat room.
- removeuser(): Removes an end user from the list of existing end users.
- setUserName(): Sets the name of the end user who is currently logged in to the Chat Room application.

The GroupChat.java file allows end users to send messages to other end users in the chat room, as shown in Figure 3-7:

Figure 3-7: Creating a Group Chat Window

◄ PREVIOUS NEXT ►

Unit Testing

To test the Chat Room application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the LoginGUI.java, MainGUI.java, ChatRoom.java, RoomInformation.java, GroupChat.java, and SocketOpener.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

5. To run the Chat Room application, specify the following command at the command prompt:

   ```
   java LoginGUI
   ```

6. The Login window of the Chat Room application appears. Specify the login information, as shown in Figure 3-8:

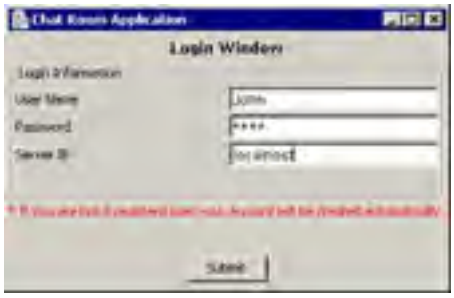


Figure 3-8: Specifying Login Information

7. Click the Submit button to send the login information to the Jabber server for registered end users or open a new account in the Jabber server for unregistered end users.

8. The user interface of the Chat Room application appears. Select File-> Go to Chat Room to specify the name of the chat room to join.

9. Enter the name of the chat room and the user name to join the chat room, as shown in Figure 3-9:

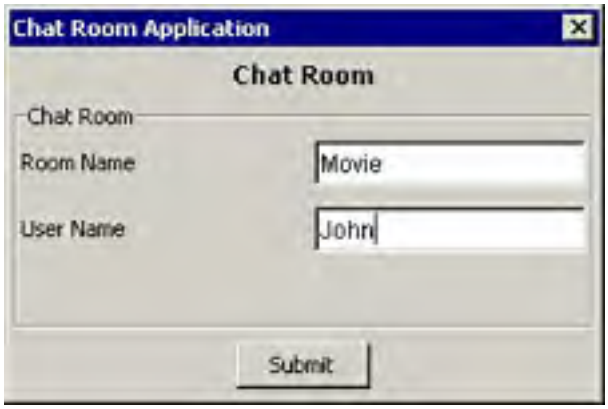


Figure 3-9: Entering a Chat Room

10. Click the Submit button to open the Group Chat window of the specified chat room, as shown in Figure 3-10:

Figure 3-10: The Group Chat Window of the Specified Chat Room

11. To create a chat room, select File-> Create New Chat Room from the user interface of the Chat Room application.
12. Enter the name of the chat room and the user name, as shown in Figure 3-11:

Figure 3-11: Entering Chat Room Name and User Name

13. Click the Submit button to create a new chat room and open the Group Chat window of the new chat room, as shown in Figure 3-12:

Figure 3-12: The Group Chat Window of the New Chat Room

14. Type the required message in the chat window and click the Send button to send the message to all end users logged in to the chat room, as shown in Figure 3-13:

Figure 3-13: Sending a Message

15. To initiate a group chat, select any chat room from the Existing Rooms tree structure of the Chat Room application, as shown in Figure 3-14:

Figure 3-14: Selecting an Existing Room

The Input Dialog box appears to specify the user name for the selected chat room, as shown in Figure 3-15:

Figure 3-15: The Input Dialog Box

16. Enter the user name and click the OK button. The Group Chat window for the selected room appears, as shown in Figure 3-16:

Figure 3-16: The Group Chat Window of the Selected Chat Room

PREVIOUS NEXT

Chapter 4: Creating a Group Chatting Application

Download CD Content

The Jabber protocol allows you to create and rename chat groups hosted on the Jabber server. End users can add user names to a particular chat group and initiate a private or group chat.

This chapter describes how to develop the Group Chatting application, which allows end users to send group messages to other end users connected to the Jabber server.

Architecture of the Group Chatting Application

The Group Chatting application uses the following files:

- GroupLoginGUI.java: Allows end users to establish a connection with the Jabber server to communicate with other end users connected to the server. This file allows end users to log in or register automatically to the Jabber server.
- GroupList.java: Shows the tree structure that contains the names of various chat groups and end users in each group. This file allows end users to select a group to start a chat session. End users can select any specific user within a group to start a private chat.
- AddUser.java: Allows end users to add other end users to a particular group. This file also allows end users to create a new group.
- ChatWindow.java: Allows end users to participate in a private chat.
- EditGroup.java: Allows end users to rename an existing chat group.
- GroupChat.java: Allows end users to participate in a group chat.
- SocketConnection.java: Opens a socket to send and receive messages through the Jabber server.
- GroupTree.java: Creates a tree structure of various groups in the Group Chatting application.

Figure 4-1 shows the architecture of the Group Chatting application:

Figure 4-1: Architecture of the Group Chatting Application

The GroupLoginGUI.java file creates the Login window of the Group Chatting application that contains various labels, text boxes, and the Submit button. The GroupLoginGUI.java file allows end users to log on as an existing user or register for a new account.

When an end user enters the login information and clicks the Submit button on the Login window, the GroupLoginGUI.java file calls the GroupList.java file to create the user interface for the Group Chatting application. The GroupList.java file allows end users to create a group or add other end users to a particular group. The GroupList.java file creates the tree structure to show the list of existing chat groups.

If an end user selects File-> Add User, the GroupList.java file calls the AddUser.java file to add other end users to an end user's personal list. The AddUser.java file also allows end users to create a chat group. The AddUser.java file provides an interface with various labels, two text boxes, and the Submit button.

If an end user selects File-> Edit Group, the GroupList.java file calls the EditGroup.java file to rename an existing group. The EditGroup.java file provides an interface with various labels, two text boxes, and the Submit button.

If an end user selects a specific end user name present in a particular group, the GroupList.java file calls the SocketConnection.java file that opens a socket to send and receive messages to the selected end user. The SocketConnection.java file calls the ChatWindow.java file, which allows an end user to start a private chat with the selected end user. The ChatWindow.java file provides an interface containing a text area to view text messages received from the Jabber server. The ChatWindow.java file also provides the Send button to send text messages to the Jabber server.

If an end user selects a chat group from the tree structure provided by the Group Chatting application, the GroupList.java file calls the SocketConnection.java file. This file opens a socket to send and receive messages to other end users present in the selected group. The SocketConnection.java file calls the GroupChat.java file to start a group chat session.

PREVIOUS NEXT

Creating the Login Window

The Group.java file creates the Login window for the Group Chatting application. Listing 4-1 shows the contents of the GroupLoginGUI.java file:

Listing 4-1: The GroupLoginGUI.java File

```
/*
Import required swing classes
*/
import javax.swing.*;
import javax.swing.text.*;
/*
Import required awt classes
*/
import java.awt.*;
import java.awt.event.*;
/*
Import required io and net classes
*/
import java.io.*;
import java.net.*;
/*
class GroupLoginGUI - This class is used to create a GUI to get the login information from user.
Constructor:
GroupLoginGUI - This constructor creates GUI.
Methods:
getUserName - Used to get the user name
getPassword - Used to get the password
getServarName - Used to get the servername
main - This method creates the main window of the application and displays all the components.
*/
public class GroupLoginGUI extends JFrame implements ActionListener, KeyListener
{
   /*Declare objects of Container class.*/
   Container container=null;
   /*Declare objects of JTextField class.*/
   JTextField namet=new JTextField();
   JTextField server=new JTextField();
   /*Declare objects of JPasswordField class.*/
   JPasswordField passwordt=new JPasswordField();
   /*Declare objects of JButton class.*/
   JButton submit=new JButton("Submit");
   /*Declare objects of SocketConnection class.*/
   Socket clientsocket;
   /*Declare string variable.*/
   String servername;
   String username;
   String password;
   /*Declare objects of JPanel class.*/
   JPanel note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
   JPanel panel=new JPanel();
   JPanel lable=new JPanel();
   /*Declare objects of JLabel class.*/
   JLabel room= new JLabel("User Name");
   JLabel name= new JLabel("Password ");
   JLabel serverl= new JLabel("Server IP");
   JLabel note1= new JLabel("*If you are not a registered user your account will be");
   JLabel note2= new JLabel("created automatically.");
   /*Constructor for class.*/
   public GroupLoginGUI ()
   {
      /*Set the window title*/
      super("Login Window");
      this.clientsocket=clientsocket;
      container=getContentPane();
      panel.setLayout(new GridLayout(4, 2, 3, 3));
      panel.setBorder(BorderFactory.createTitledBorder("Login Information"));
      panel.add(room);
      panel.add(namet);
      panel.add(name);
      panel.add(passwordt);
      panel.add(serverl);
      panel.add(server);
      note1.setForeground(Color.red);
      note2.setForeground(Color.red);
      note.add(note1);
      note.add(note2);
      JLabel heading=new JLabel("Login Window", JLabel.CENTER);
      lable.add(heading,BorderLayout.NORTH);
      heading.setFont(new Font("verdana", 1, 12));
      container.add(lable,BorderLayout.NORTH);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        container.add(panel,BorderLayout.CENTER);
        JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
        JPanel button=new JPanel(new FlowLayout());
        button.add(submit);
        buttonpanel.add(note);
        buttonpanel.add(button);
        container.add(buttonpanel,BorderLayout.SOUTH);
        setBounds(5, 5, 371, 240);
        /*Add the listener to the button*/
        submit.addActionListener(this);
        submit.addKeyListener(this);
        show();
    }
    public void keyPressed(KeyEvent e)
    {
        if(e.getKeyCode()==KeyEvent.VK_ENTER)
        {
        servername=server.getText();
        username=namet.getText();
        password=passwordt.getText();
        GroupList grp=new GroupList(this, servername, username, password);
        }
    }
    public void keyTyped(KeyEvent e)
    {
    }
    public void keyReleased(KeyEvent e)
    {
    }
    public void actionPerformed(ActionEvent e)
    {
        servername=server.getText();
        username=namet.getText();
        password=passwordt.getText();
        try
        {
            GroupList grp=new GroupList(this, servername, username, password);
            hide();
        }
        catch(Exception ee)
        {
        ee.printStackTrace();
        }
    }
    public String getUserName()
    {
        return username;
    }
    public String getPassword()
    {
        return password;
    }
    public String getServarName()
    {
        return servername;
    }
    public static void main(String[] args)
    {
        /*Set the window look and feel*/
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        GroupLoginGUI lgui=new GroupLoginGUI();
    }
}
```

Download this listing.

In the above code, the main() method creates an instance of the GroupLoginGUI class.

The methods defined in Listing 4-1 are:

- `getUserName()`: Retrieves the name of the end user currently logged in.
- `getPassword()`: Retrieves the password specified by the end user.
- `getServerName()`: Retrieves the name of the Jabber server.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs. End users invoke the actionPerformed() method by clicking the Submit button on the Login window.

The GroupLoginGUI.java file creates the Login window of the Group Chatting application, as shown in Figure 4-2:

Figure 4-2: The Login Window

◄ PREVIOUS NEXT ►

Creating the User Interface of the Group Chatting Application

The GroupList.java file creates the user interface of the Group Chatting application. Listing 4-2 shows the contents of the GroupList.java file:

Listing 4-2: The GroupList.java File

```
/*Import required  tree package classes*/
import javax.swing.tree.*;
import javax.swing.*;
/*Import required  awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required  event package classes*/
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeModelEvent;
/*Import required  io package classes*/
import java.io.*;
/*Import required  net package classes*/
import java.net.*;
/*Import required  util package classes*/
import java.util.*;
/*
class GroupList - This class is used to create a GUI to show the group and user.
Constructor:
GroupList-This constructor creates GUI.
Methods:
openPort - Used to open the port
clearTree - Used to remove the tree node
addToTree-Used to set the group and user into tree
*/
public class GroupList extends JFrame implements TreeSelectionListener,ActionListener,Runnable
{
   /*Declare object of JMenuBar class.*/
   JMenuBar jb=new JMenuBar();
   /*Declare object of JMenu class.*/
   JMenu menu=new JMenu("File");
   /*Declare object of JMenuItem class.*/
   JMenuItem edit=new JMenuItem("Edit Group");
   JMenuItem addmember=new JMenuItem("Add User");
   /*Declare string variable.*/
   String firstname="";
   String lastname="";
   String emailid="";
   String ipaddress;
   String servername;
   String username;
   String password;
   String resource="Home";
   String errortype="";
   String roomname;
   String nickname;
   /*Declare integer*/
   int portno=5222;
   int wait=1000;
   /*Declare object of Hashtable class.*/
   Hashtable has=new Hashtable();
   Hashtable hasuser=new Hashtable();
   Hashtable edituser=new Hashtable();
   /*Declare object of Button class.*/
   Button sendbutton;
   Button closebutton;
   /*Declare the boolean variable.*/
   boolean existing;
   private boolean isConnected=false;
   /*Declare object of Thread class.*/
   Thread inputmessagethread;
   /*Declare object of JScrollPane class.*/
   JScrollPane treeView;
   /*Declare object of JTree class.*/
   JTree tree;
   /*Declare object of DefaultMutableTreeNode class.*/
   DefaultMutableTreeNode groupnode;
   DefaultMutableTreeNode groupnode1;
   DefaultMutableTreeNode top= new DefaultMutableTreeNode("Existing Category");
   DefaultTreeModel treeModel;
   /*Declare object of GroupLoginGUI class.*/
   GroupLoginGUI groupLogin;
   /*Declare object of Socket class.*/
   Socket clientsocket1;
   SocketConnection clientsocket;
   /*Declare object of SocketConnection class.*/
```

```
SocketConnection sc;
/*Declare object of GroupTree class.*/
GroupTree dynamic;
/*Declare object of EditGroup class.*/
EditGroup ag;
/*Declare object of ChatWindow class.*/
ChatWindow chat;
/*Constructor for class*/
public GroupList(GroupLoginGUI groupLogin, String server, String user, String password)
{
   super("Group Chatting Application");
   this.servername=server;
   this.username=user;
   this.password=password;
   this.groupLogin=groupLogin;
   this.password=password;
   start1();
   dynamic=new GroupTree(sc, groupLogin, this);
   jb.add(menu);
   menu.add(addmember);
   menu.add(edit);
   this.setJMenuBar(jb);
   edit.addActionListener(this);
   addmember.addActionListener(this);
   setBounds(5,5,350,350);
   setVisible(false);
   sc.sendRosterRequest();
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   getContentPane().add(dynamic);
   show();
}
/*
Function to get object of ChatWindow class
Parameter: N/A
Return Value: ChatWindow
*/
public ChatWindow getChat()
{
   return this.chat;
}
public void setChat(ChatWindow ch)
{
   this.chat=ch;
}
/*
Function to clear the tree
Parameter: N/A
Return Value: N/A
*/
public void clearTree()
{
   dynamic.clearTree();
   edituser.clear();
   has.clear();
   hasuser.clear();
}
public void valueChanged(TreeSelectionEvent e)
{
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)
   tree.getLastSelectedPathComponent();
   TreeNode parent =node.getParent();
   boolean b=node.isLeaf();
   if(b)
   {
      String user=node.toString();
   }
}
public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==edit)
   {
      ag=new EditGroup(this, dynamic,sc);
   }
   if(e.getSource()==addmember)
   {
      AddUser am=new AddUser(this, dynamic, sc);
   }
}
/*
Function to open the port and write the XML messages to server
Parameter: N/A
Return Value: N/A
*/
public void start1()
{
   try
   {
      openPort(servername, 5222, 1000);
```

```
    }
    catch(Exception e)
    {
    }
    if (clientsocket1!=null)
    {
       isConnected=true;
    }
    Thread inputthread=new Thread(this);
    inputthread.start();
    try
    {
       String sessionStratString;
       String authentication;
       String registrationstring;
       sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
       sessionStratString=sessionStratString+" <stream:stream";
       sessionStratString=sessionStratString+" to= \""+servername +"\"";
       sessionStratString=sessionStratString+" xmlns=\"jabber:client\"";
       sessionStratString=sessionStratString+"
       xmlns:stream=\"http://etherx.jabber.org/streams\">";
       sc.sendXMLToJabber(sessionStratString);
       System.out.println("sessionStratString"+sessionStratString);
       if ((firstname.equals("false"))||(lastname.equals("false"))||(emailid.equals("false")))
       {
          System.out.println("sessionStratString if"+sessionStratString);
       }
       else
       {
          registrationstring="<iq type=\"set\" to=\""+username+"@localhost\" id=\"1001\">";
          registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
          registrationstring=registrationstring+"<username>"+username+"</username>";
          registrationstring=registrationstring+"<password>"+password+"</password>";
          registrationstring=registrationstring+"<first>"+firstname+"</first>";
          registrationstring=registrationstring+"<last>"+lastname+"</last>";
          registrationstring=registrationstring+"<email>"+emailid+"</email>";
          registrationstring=registrationstring+" </query> ";
          registrationstring=registrationstring+"</iq>";
          System.out.println("registrationstring"+registrationstring);
       }
       authentication="<iq type=\"set\" id=\"1301\">";
       authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
       authentication=authentication+" <username>"+username+"</username>";
       authentication=authentication+" <password>"+password+"</password>";
       authentication=authentication+" <resource>"+resource+"</resource> ";
       authentication=authentication+" </query> ";
       authentication=authentication+"</iq>";
       sc.sendXMLToJabber(authentication);
       System.out.println("authentication"+authentication);
    }
    catch(Exception ie)
    {
    }
}
/*
Function to get the user name
Parameter: N/A
Return Value: String
*/
public String getUserName()
{
    String user=dynamic.getNodeName();
    return user;
}
public void run()
{
    System.out.println("Socket is opened runnnn"+sc);
    sc.runinput();
}
/*
Function to open port
Parameter: ipaddress-Object of string class
portno-int
timeinsec-int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
    String host;
    if (ipaddress.trim()=="")
    {
    }
    else
    {
       sc=new SocketConnection(ipaddress,portno,this,dynamic,groupLogin);
       clientsocket=sc.openSocket(ipaddress,portno,timeinsec);
       if(clientsocket==null)
       {
```

```
            System.out.println("Socket is opened null null"+clientsocket);
         }
      }
   }
   public static void main(String arg[])
   {
      /*Set the window look and feel*/
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }
   /*
   Function to add the user list and user to the tree
   Parameter: groupname-Object of string class
   jid-Object of string class
   contactpersonname-Object of string class
   Return Value: N/A
   */
   public void addToTree(String groupname,String jid,String contactpersonname)
   {
      edituser.put(jid,groupname);
      boolean elementfound=false;
      groupnode=new DefaultMutableTreeNode(groupname);
      Enumeration e=has.elements();
      while(e.hasMoreElements())
      {
         String haselm=(String)e.nextElement();
         if(haselm.equals(groupname))
         {
            elementfound=true;
         }
      }
      if(!elementfound)
      {
         if(groupnode!=null)
         {
         if(!(hasuser.containsValue(groupnode)))
         {
            groupnode = dynamic.addObject(null, groupname);
            groupnode1=newDefaultMutableTreeNode(contactpersonname);
            groupnode.add(groupnode1);
            hasuser.put(groupname,groupnode);
            has.put(jid,groupname);
         }
      }
   }
   else
   {
      if(!hasuser.containsValue(contactpersonname.trim()))
      {
         DefaultMutableTreeNode d2=(DefaultMutableTreeNode)hasuser.get(groupname);
         dynamic.addObject(d2,contactpersonname);
         hasuser.put(jid,contactpersonname.trim());
         has.put(jid,groupname);
      }
   }
}
/*
Function to write the text into the textpane
Parameter:messagebody-Object of string class
user-Object of string class
Return Value: N/A
*/
public void insertTextInToTextPane(String messagebody,String user)
{
   String[] splitedusername=user.split("@");
   ChatWindow chatwindowobject=(ChatWindow)dynamic.mapObj.get(splitedusername[0]);
   if (chatwindowobject!=null)
   {
      chatwindowobject.insertTextInToTextPane(messagebody,splitedusername[0]);
   }
   else
   {
      ChatWindow ch=new ChatWindow(username,splitedusername[0],servername,sc);
      ch.insertTextInToTextPane(messagebody,splitedusername[0]);
      dynamic.mapObj.put(splitedusername[0],ch);
   }
}
}
```

Download this listing.

In the above code, the constructor of the GroupList class takes an object of the GroupLoginGUI class and three strings as input parameters: user, password, and server. The user string retrieves the user name specified by an end user. The password string retrieves the password specified by an end user, and the server string retrieves the name of the Jabber server specified by an end user. The object of the GroupLoginGUI class allows an end user to invoke the methods of the GroupLoginGUI class.

The methods defined in Listing 4-2 are:

- `openPort()`: Opens a client socket by using the Internet Protocol (IP) address and port number of the Jabber server.
- `clearTree()`: Refreshes the tree structure containing group and end user names.
- `addToTree()`: Adds the newly created group to the tree structure of the available groups.
- `getChat()`: Retrieves the object of the ChatWindow class.
- `insertTextInToTextPane()`: Inserts the text message specified by an end user into the text area of the chat window.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.

The GroupList.java file creates the main window of the Group Chatting application, as shown in Figure 4-3:

Figure 4-3: User Interface of the Group Chatting Application

Select the File menu to view the File menu options.

Figure 4-4 shows the File menu options of the Group Chatting application:

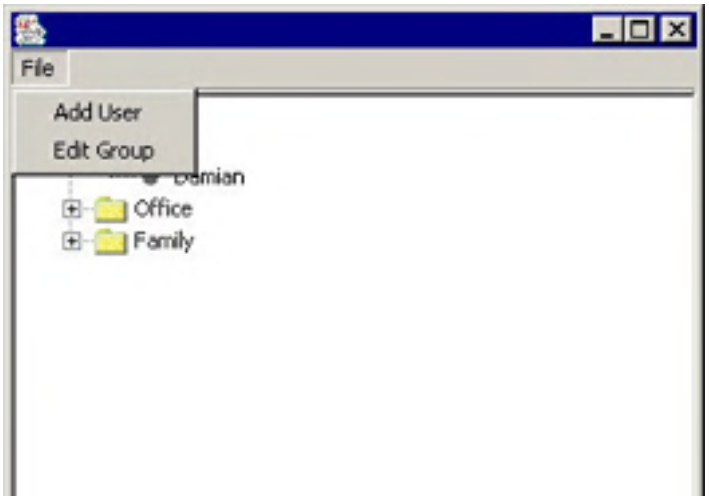


Figure 4-4: The File Menu of the Group Chatting Application

Select File-> Add User to add other end users to an end user's personal list of a particular chat group.

Select File-> Edit Group to rename an existing group.

Select any group or any specific end user from a particular group to initiate the chat.

◄ PREVIOUS NEXT ►

Adding Users

The AddUser.java file creates the user interface to add other end users to a particular group. Listing 4-3 shows the contents of the AddUser.java file:

Listing 4-3: The AddUser.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
import javax.swing.tree.*;
/*
class AddUser - This class is used to create a GUI to take the information from user.
Constructor:
AddUser-This constructor creates GUI.
*/
public class AddUser extends JFrame implements ActionListener
{
/*Declare object of Container class.*/
Container container=null;
/*Declare object of JPanel class.*/
JPanel panel=new JPanel();
JPanel lable=new JPanel();
JPanel buttonpanel=new JPanel(new GridLayout(1,1,5,5));
JPanel bp=new JPanel(new FlowLayout(FlowLayout.CENTER));
/*Declare object of JLabel class.*/
JLabel membername= new JLabel("User Name");
JLabel groupbername= new JLabel("Group Name");
JLabel heading=new JLabel("User Information",JLabel.CENTER);
/*Declare object of JTextField class.*/
JTextField membernamet=new JTextField();
JTextField groupnamet=new JTextField();
/*Declare object of JButton class.*/
JButton submit=new JButton("Submit");
/*Declare object of GroupList class.*/
GroupList group;
/*Declare object of GroupTree class.*/
GroupTree dynamic;
/*Declare object of SocketConnection class.*/
SocketConnection clientsocket;
/*Declare string variable.*/
String servername="localhost";
String friends=new String("Friends");
String family=new String("Family");
String office=new String("Office");
String colleague=new String("Colleague");
String selectedgroup="";
/*Constructor */
public AddUser (GroupList group,GroupTree dynamic,SocketConnection clientsocket)
{
   /*Set the window title*/
   super("Group Chatting Application");
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   this.group=group;
   this.dynamic=dynamic;
   this.clientsocket=clientsocket;
   container=getContentPane();
   /*set the panel's layout*/
   panel.setLayout(new GridLayout(2, 1, 1, 1));
   panel.setBorder(BorderFactory.createTitledBorder("Add User" ));
   /*Add the component to panel*/
   panel.add(membername);
   panel.add(membernamet);
   panel.add(groupbername);
   panel.add(groupnamet);
   /*Add the label to panel*/
   lable.add(heading,BorderLayout.NORTH);
   /*set the heading of font*/
   heading.setFont(new Font("verdana", 1, 12));
   /*Add the panel to container*/
```

```
    container.add(lable,BorderLayout.NORTH);
    /*Set the default close operation for window*/
    setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    /*Add the panel to container*/
    container.add(panel,BorderLayout.CENTER);
    /*Add the submit button to panel*/
    bp.add(submit);
    buttonpanel.add(bp);
    /*Add the panel to container*/
    container.add(buttonpanel,BorderLayout.SOUTH);
    setBounds(5,5,250,160);
    /*Add the listener to the button*/
    submit.addActionListener(this);
    /*Called the show function to display the window*/
    show();
}
/*Method to describe the event*/
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==submit)
    {
        String user=membernamet.getText();
        selectedgroup=groupnamet.getText();
        String from=dynamic.groupLogin.getUserName();
        String messagestring="<message type='chat'  from='"+from+"@localhost/Home'
        to='"+user+"@"+servername+"/Home' >";
        messagestring=messagestring+"<body>";
        messagestring=messagestring+"";
        messagestring=messagestring+"</body></message>";
        clientsocket.sendXMLToJabber(messagestring);
        String tag=clientsocket.tagentity;
        String userGroup="<iq type=\"set\" id=\"uniquevalue\">";
        userGroup=userGroup+"<query xmlns=\"jabber:iq:roster\">";
        userGroup=userGroup+"<item jid=\""+user+"@localhost/Home\"  name=\""+user+"\" ";
        userGroup=userGroup+"subscription=\"none\"  ask=\"subscribe\">";
        userGroup=userGroup+"<group>"+selectedgroup+"</group></item></
        query></iq>";
        System.out.println("UserGroup"+userGroup);
        clientsocket.createGroup(userGroup);
        clientsocket.sendRosterRequest();
        hide();
    }
}
}
```

Download this listing.

In the above code, the constructor of the AddUser class takes an object of the GroupList class, GroupTree class, and SocketConnection class as input parameters. This allows end users to invoke the methods of the GroupList class, GroupTree class, and SocketConnection class. The AddUser class defines the actionPerformed() method, which acts as an event listener and activates an appropriate method based on the action an end user performs.

The AddUser.java file creates the window to add other end users to a group, as shown in Figure 4-5:

Figure 4-5: Adding an End User

Renaming the Group

The EditGroup.java file creates the user interface to rename an existing chat group. Listing 4-4 shows the contents of the EditGroup.java file:

Listing 4-4: The EditGroup.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
import javax.swing.tree.*;
import java.util.*;
/*
class EditGroup - This class is used to create a GUI to take the information from user.
Constructor:
EditGroup-This constructor creates GUI.
Methods:
getOldGroupName:-get the froup name
main - This method creates the look and feel.
*/
public class EditGroup extends JFrame implements ActionListener
{
/*Declare objects of Container class.*/
Container container=null;
/*Declare objects of panel class.*/
JPanel panel=new JPanel();
JPanel lable=new JPanel();
JPanel buttonpanel=new JPanel(new GridLayout(1,1,5,5));
JPanel bp=new JPanel(new FlowLayout(FlowLayout.CENTER));
/*Declare objects of label  class.*/
JLabel oldgroupname= new JLabel("Old Group Name");
JLabel newgroupname= new JLabel("New Group Name");
JLabel heading=new JLabel("Rename Group", JLabel.CENTER);
JTextField oldgroupnamet=new JTextField();
JTextField newgroupnamet=new JTextField();
/*Declare objects of button class.*/
JButton submit=new JButton("Submit");
/*Declare objects of GroupList class.*/
GroupList group;
/*Declare objects of DefaultMutableTreeNode class.*/
DefaultMutableTreeNode newgroup;
/*Declare objects of GroupTree class.*/
GroupTree dynamic;
/*Declare objects of SocketConnection  class.*/
SocketConnection clientsocket;
/*Declare objects of string variable.*/
String userGroup="";
String ngroupname="";
String ogroupname="";
/*Constructor*/
public EditGroup (GroupList group, GroupTree dynamic, SocketConnection clientsocket)
{
   /*Set the window title*/
   super("Group Chatting Application");
   /*Set the window look and feel*/
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   this.group=group;
   this.dynamic=dynamic;
   this.clientsocket=clientsocket;
   container=getContentPane();
   /*Set the layout of panel*/
   panel.setLayout(new GridLayout(2, 1, 5, 5));
   /*Set the border of panel*/
   panel.setBorder(BorderFactory.createTitledBorder("Group Information"));
   /*Add the component to the panel*/
   panel.add(oldgroupname);
   panel.add(oldgroupnamet);
   panel.add(newgroupname);
   panel.add(newgroupnamet);
   lable.add(heading,BorderLayout.NORTH);
   /*Set the font of heading*/
```

```
   heading.setFont(new Font("verdana", 1, 12));
   container.add(lable,BorderLayout.NORTH);
   /*Set the default close operation*/
   setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
   container.add(panel,BorderLayout.CENTER);
   bp.add(submit);
   buttonpanel.add(bp);
   /*Add panel to the container*/
   container.add(buttonpanel,BorderLayout.SOUTH);
   /*Set the bounds for window*/
   setBounds(5, 5, 250, 160);
   /*Add the action listener to the button*/
   submit.addActionListener(this);
   /*Called the show function to display the window*/
   show();
}
/*
Function to get the groupname
Parameter: N/A
Return Value: N/A
*/
public String getOldGroupName()
{
   return ogroupname;
}
/* Method to describe the event*/
public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==submit)
   {
      ngroupname=newgroupnamet.getText();
      ogroupname=oldgroupnamet.getText();
      Vector vec=new Vector();
      Enumeration ee=group.edituser.keys();
      while(ee.hasMoreElements())
      {
         String haselm=(String)ee.nextElement();
         String usertoedit=haselm.substring(1,haselm.length());
         String spliteduser[]=usertoedit.split("@");
         Object nname=group.edituser.get(haselm);
         String nnamestr=nname.toString();
         if(nnamestr.equals(ogroupname))
         {
            vec.add(spliteduser[0]);
         }
      }
      for(int j=0;j<vec.size();j++)
      {
         userGroup="<iq type=\"set\" id=\"uniquevalue\">";
         userGroup=userGroup+"<query xmlns=\"jabber:iq:roster\">";
         userGroup=userGroup+"<item jid=\""+(String)vec.elementAt(j)+"@localhost/Home\"
         name=\""+(String)vec.elementAt(j)+"\" ";
         userGroup=userGroup+"subscription=\"none\"  ask=\"subscribe\">";
         userGroup=userGroup+"<group>"+ngroupname+"</group></item></
         query></iq>";
         clientsocket.createGroup(userGroup);
      }
      clientsocket.sendRosterRequest();
      }
   }
}
```

Download this listing.

In the above code, the constructor of the EditGroup class takes an object of the GroupList class, GroupTree class, and SocketConnection class as input parameters. The EditGroup class allows end users to invoke the methods of the GroupList class, GroupTree class, and SocketConnection class.

The methods defined in Listing 4-4 are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.
- `getOldGroupName()`: Retrieves the old name of a group after an end user renames the group.

The EditGroup.java file creates the Rename Group window to rename a group, as shown in Figure 4-6:

Figure 4-6: Renaming a Group

PREVIOUS NEXT

Initiating a Private Chat

The ChatWindow.java file creates the user interface that allows end users to initiate a private chat. Listing 4-5 shows the contents of the ChatWindow.java file:

Listing 4-5: The ChatWindow.java File

```
/*Import required swing classes*/
import javax.swing.*;
/*Import required swing event classes*/
import javax.swing.event.*;
/*Import required swing text classes*/
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
public class  ChatWindow extends JFrame implements ActionListener
{
   /*Declare object of Container class.*/
   Container container;
   GridBagLayout gridbaglayout;
   /*Declare object of JLabel class.*/
   JLabel fromuser = null;
   JLabel touser = null;
   JLabel fromusername = null;
   JLabel tousername = null;
   /*Declare object of JTextPane class.*/
   JTextPane textarea = null;
   JTextField textfield = null;
   /*Declare object of JButton class.*/
   JButton sendbutton = null;
   /*Declare object of JScrollPane class.*/
   JScrollPane scpane;
   /*Declare string variable.*/
   String user="";
   String receiver="";
   String servername="";
   String messagestring="";
   /*Declare object of SocketConnection class.*/
   SocketConnection clientsocket;
   Document contentmodel;
   MutableAttributeSet recervernameatrib;
   ChatWindow chat;
   /*
   class ChatWindow - This class is used to create a GUI for chatting.
   Constructor:
   ChatWindow-This constructor creates GUI.
   Methods:
   setObj - Used to set the use object of ChatWindow
   insertTextInToTextPane - Used to write the text into the textpane
   */
   /* Constructor for class*/
   public ChatWindow(String user, String receiver, String servername, SocketConnection clientsocket)
   {
      this.user=user;
      this.receiver=receiver;
      this.servername=servername;
      this.clientsocket=clientsocket;
      fromuser = new JLabel("From :");
      fromusername  = new JLabel(user);
      String touser1= receiver;
      touser = new JLabel("To :");
      tousername = new JLabel(touser1);
      /*Set the window title*/
      setTitle("Group Chatting Application: "+touser1);
      container = this.getContentPane();
      recervernameatrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(recervernameatrib,"Verdana");
      StyleConstants.setForeground(recervernameatrib,Color.red);
      gridbaglayout=new GridBagLayout();
      GridBagConstraints gridbagconstraints=new GridBagConstraints();
      /*Creating a panel with gridbaglayout*/
      JPanel jpanel1 = new JPanel(gridbaglayout);
      gridbagconstraints.fill=GridBagConstraints.BOTH;
      gridbagconstraints.insets=new Insets(5, 5, 0, 0);
      gridbagconstraints.gridx=0;
      gridbagconstraints.gridy=0;
      gridbagconstraints.weightx=1;
      gridbagconstraints.weighty=1;
      gridbagconstraints.anchor=GridBagConstraints.WEST;
      gridbaglayout.setConstraints(fromuser, gridbagconstraints);
      fromuser.setPreferredSize(new Dimension(140, 28));
      /*Adding component to panel*/
      jpanel1.add(fromuser);
```

```
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 20);
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(fromusername,gridbagconstraints);
    fromusername.setPreferredSize(new Dimension(20, 28));
    fromusername.setFont(new Font("Verdana", Font.BOLD, 10));
    /*Adding the component to panel*/
    jpanel1.add(fromusername);
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=2;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(touser,gridbagconstraints);
    touser.setPreferredSize(new Dimension(140, 28));
    /*Adding the component to panel*/
    jpanel1.add(touser);
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=3;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.WEST;
    gridbaglayout.setConstraints(tousername, gridbagconstraints);
    tousername.setPreferredSize(new Dimension(20,28));
    tousername.setFont(new Font("Verdana", Font.BOLD, 10));
    /*Adding component to panel*/
    jpanel1.add(tousername);
    /*Adding panel to container*/
    container.add(jpanel1,BorderLayout.NORTH);
    /*Creating a new panel*/
    JPanel jpanel2 = new JPanel();
    /*Creating new textpane*/
    textarea = new JTextPane();
    contentmodel=textarea.getDocument();
    /*Set the size of textpane*/
    textarea.setPreferredSize(new Dimension(80, 20));
    /*Creating a scrollpane and adding a textpane in to the scrollpane*/
    scpane  = new JScrollPane(textarea,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.
    HORIZONTAL_SCROLLBAR_AS_NEEDED);
    container.add(scpane,BorderLayout.CENTER);
    /*Creating a panel with gridbaglayout*/
    JPanel jpanel3 = new JPanel(gridbaglayout);
    textfield = new JTextField();
    /*Creating a object of send button*/
    sendbutton = new JButton("Send");
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=0;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.NORTH;
    gridbaglayout.setConstraints(textfield, gridbagconstraints);
    textfield.setPreferredSize(new Dimension(120, 28));
    /*Adding a textfield into panel*/
    jpanel3.add(textfield);
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=0;
    gridbagconstraints.weighty=0;
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(sendbutton, gridbagconstraints);
    sendbutton.setPreferredSize(new Dimension(60, 28));
    /*Adding a send button into panel*/
    jpanel3.add(sendbutton);
    /*Adding action listener in to sendbutton*/
    sendbutton.addActionListener(this);
    /*Adding action listener in to textfield*/
    textfield.addActionListener(this);
    /*Adding a panel to container*/
    container.add(jpanel3,BorderLayout.SOUTH);
    /*Set the bounds for window*/
    setBounds(100,100,350,280);
    /*Called the show method to display the window*/
    show();
}
/*Method to describe the event*/
```

```
   public void actionPerformed(ActionEvent e)
   {
      if(e.getSource()==sendbutton)
      {
         if (!textfield.getText().trim().equals(""))
         {
         String[] usersplited=user.split("@");
         messagestring="<message type='chat' from='"+usersplited[0]+"@localhost/Home'
         to='"+receiver+"@"+servername+"/Home' >";
         messagestring=messagestring+"<body>";
         messagestring=messagestring+textfield.getText().trim();
         messagestring=messagestring+"</body></message>";
         try
         {
            contentmodel.insertString(contentmodel.getLength(), "\n"+user+":
            "+textfield.getText(),recervernameatrib);
         }
         catch(Exception ble)
         {
         }
         clientsocket.sendXMLToJabber(messagestring);
         textfield.setText("");
      }
   }
   if(e.getSource()==textfield)
   {
   if (!textfield.getText().trim().equals(""))
   {
      String[] usersplited=user.split("@");
      messagestring="<message type='chat' from='"+usersplited[0]+"@localhost/Home'
      to='"+receiver+"@"+servername+"/Home' >";
      messagestring=messagestring+"<body>";
      messagestring=messagestring+textfield.getText().trim();
      messagestring=messagestring+"</body></message>";
      try
      {
         contentmodel.insertString(contentmodel.getLength(), "\n"+user+":
         "+textfield.getText(), recervernameatrib);
      }
      catch(Exception ble)
      {
      }
      clientsocket.sendXMLToJabber(messagestring);
      textfield.setText("");
      }
   }
}
/*
Function to set the use object of ChatWindow
Parameter: chat - Object of ChatWindow class.
Return Value: N/A
*/
public void setObj(ChatWindow chat)
{
   this.chat=chat;
}
/*
Used to write the text into the textpane
Parameter: message1 - Object of String class.
sendername1-Object of String class.
Return Value: N/A
*/
public void insertTextInToTextPane(String message1, String sendername1)
{
   try
   {
      contentmodel.insertString(contentmodel.getLength(), "\n"+sendername1+":
      "+message1,recervernameatrib);
   }
   catch(BadLocationException ble)
   {
   }
}
}
```

Download this listing.

In the above code, the constructor of the ChatWindow class takes the object of the SocketConnection class and three strings as input parameters: user, server, and receiver. The user string retrieves the name of an end user. The server string retrieves the name of the Jabber server and the receiver string retrieves the name of the end user at the receiving end. The object of the SocketConnection class helps in invoking the methods of the SocketConnection class.

The methods defined in Listing 4-5 are:

- `setObj()`: Sets the object of the ChatWindow class.

- `insertTextInToTextPane()`: Inserts the text message specified by an end user into the text area provided by the Group Chatting application.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The ChatWindow.java file creates a Private Chat window that allows the end user to initiate a private chat, as shown in Figure 4-7:

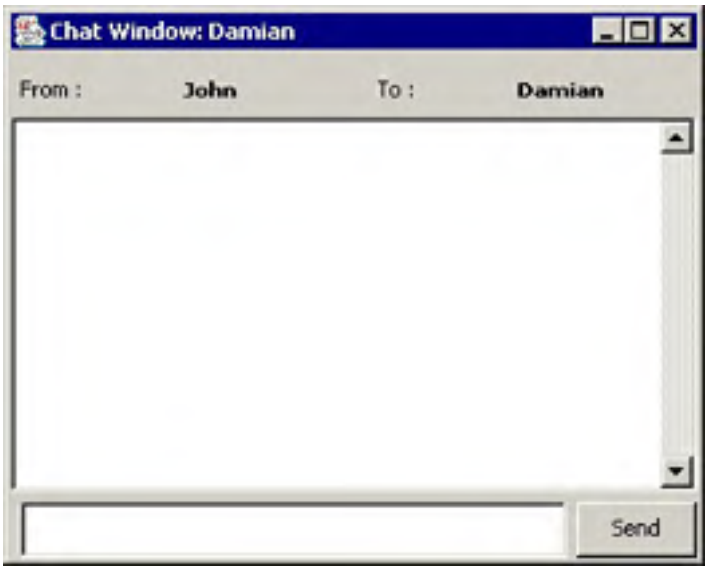


Figure 4-7: Private Chat Window

PREVIOUS NEXT

Initiating a Group Chat

The GroupChat.java file creates the user interface that allows end users to participate in a group chat. Listing 4-6 shows the contents of the GroupChat.java file:

Listing 4-6: The GroupChat.java File

```
/*Import required swing classes*/
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required util classes*/
import java.util.*;
/*
class GroupChat - This class is used to create a GUI for chatting.
Constructor:
GroupChat-This constructor creates GUI.
Methods:
clearPane - Used to clear the text
addElement - Used to add user to the list
writeintotextfield-Used to write text into textpane
*/
public class  GroupChat extends JFrame implements ActionListener
{
   /*Declare object of Container class.*/
   Container cont = null;
   /*Declare object of JPanel class.*/
   JPanel jp=new JPanel(new BorderLayout());
   /*Declare object of JButton class.*/
   JButton button=new JButton("Send");
   /*Declare object of JTextField class.*/
   JTextField text=new JTextField();
   /*Declare object of JList class.*/
   JList list;
   /*Declare object of DefaultListModel class.*/
   private DefaultListModel listModel;
   /*Declare object of JTextPane class.*/
   JTextPane  je=new JTextPane();
   /*Declare object of JScrollPane class.*/
   JScrollPane scrollPane1 = new JScrollPane(je);
   JScrollPane scrollPane ;
   /*Declare object of JSplitPane class.*/
   JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
   /*Declare the string variable*/
   String chatText=null;
   String roomname;
   String user;
   String username="";
   String servername;
   /*Declare object of Document class.*/
   Document contentmodel;
   /*Declare object of MutableAttributeSet class.*/
   MutableAttributeSet nicknameattrib;
   SocketConnection sc;
   /*Used to clear the text*/
   public void clearPane()
   {
      je.setText("");
   }
   public void addElement(String element)
   {
      listModel.addElement(element);
   }
   /*Constructor for class*/
   public GroupChat(String user, String servername, String roomname, SocketConnection clientsocket)
   {
      /*Set the title for window*/
      super(" Group Chatting Application: "+roomname);
      cont = getContentPane();
      /*Set the textpane noneditable*/
      je.setEditable(false);
      /*Set the attribute for the font*/
      nicknameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(nicknameattrib, "Verdana");
      StyleConstants.setForeground(nicknameattrib, Color.black);
      contentmodel=je.getDocument();
      this.roomname=roomname;
      this.user=user;
      this.servername=servername;
      /*Create the instance of DefaultListModel*/
      listModel=new DefaultListModel();
```

```
        String groupmessage="<presence from=\""+user+"@conference.localhost/Home\"
        to=\""+roomname.trim()+"@conference."+servername+"/"+user.trim()+"\"/>";
        sc=clientsocket;
        sc=sc.getSocketClass();
        sc.setGroupuser(this);
        /*Called the sendXMLToJabber function to send the message to jabber server*/
        sc.sendXMLToJabber(groupmessage);
        /*Add the listModel in to List */
        list=new JList(listModel);
        /*Add the list into JScrollPane*/
        scrollPane= new JScrollPane(list);
        list.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
        list.setVisibleRowCount(5);
        /*Set the scrollpane into splitpane*/
        splitPane.setTopComponent(scrollPane1);
        splitPane.setBottomComponent(scrollPane);
        /*Set the divider of splitpane*/
        splitPane.setDividerLocation(400);
        splitPane.setPreferredSize(new Dimension(700, 200));
        list.setSelectionBackground(Color.cyan);
        /*Set the size of button*/
        button.setPreferredSize(new Dimension(130, 20));
        /*Set the size of textfield*/
        text.setPreferredSize(new Dimension(400, 20));
        /*Add the text field in to panel*/
        jp.add(text,BorderLayout.WEST);
        /*Add the button field in to panel*/
        jp.add(button,BorderLayout.EAST);
        /*Add the panel in to container*/
        cont.add(jp,BorderLayout.SOUTH);
        cont.add(splitPane,BorderLayout.CENTER);
        /*Set bonds for window*/
        setBounds(5,5,550,550);
        getRootPane().show();
        /*Add the listener to the button*/
        button.addActionListener(this);
        /*Called the function to show the window*/
        this.show();
    }
    /*
    Used to write text into textpane
    Parameter: writeintotextfield-Object of String class
    psendernameattrib-Object of MutableAttributeSet class
    Return Value: N/A
    */
    public void writeintotextfield(String writeintotextfield, MutableAttributeSet psendernameattrib)
    {
        try
        {
            scontentmodel.insertString(contentmodel.getLength(), writeintotextfield, psendernameattrib)
        }
    catch(Exception e)
    {
    }
}
/*
Method to describe the event
Parameter: e-Object of ActionEvent class
Return Value: N/A
*/
public void actionPerformed(ActionEvent e)
{
    String sendxmlstring="<message to=\""+roomname+"@conference."+servername+"\"
    from=\""+user+"@conference.localhost/Home\" type=\"groupchat\">";
    sendxmlstring=sendxmlstring+"<body>"+text.getText()+"</body></message>";
    sc.sendXMLToJabber(sendxmlstring);
    text.setText("");
}
}
```

Download this listing.

In the above code, the constructor of the GroupChat class takes the object of the SocketConnection class and three strings as input parameters: user, server, and roomname. The user string retrieves the name of an end user. The server string retrieves the name of the Jabber server, and the roomname string retrieves the name of the chat room, which an end user selects. The object of the SocketConnection class helps in invoking the methods of the SocketConnection class.

The methods defined in Listing 4-6 are:

- clearPane(): Refreshes the text area every time an end user selects a particular group.
- addElement(): Adds other end users to the selected group.
- itemintotextfield(): Inserts the text message specified by an end user into the text area provided by the Group Chatting application.

- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs.

The GroupChat.java file creates the Group Chat window that allows end users to participate in a group chat, as shown in Figure 4-8:

Figure 4-8: The Group Chat Window

PREVIOUS NEXT

Creating the Socket Class to Send and Receive Messages

The SocketConnection.java file creates a socket with the Jabber server to send and receive messages between end users. Listing 4-7 shows the contents of the SocketConnection.java file:

Listing 4-7: The SocketConnection.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*Import required util classes*/
import java.util.*;
import java.lang.*;
import javax.swing.tree.*;
/*
class SocketConnection - This class is used to create a GUI to show the group and user.
Constructor:
SocketConnection-This constructor creates GUI.
Methods:
openSocket - Used to open the port.
socketope - Used to instantiate the socket class.
getErrorString-Used to get the error string.
sendername-Used to get the sendername.
checkForError-Used to check XML error.
runinput-Used to read the server output
*/
public  class  SocketConnection
{
   /*Declare the integer*/
   private int openerport;
   /*Declare object of Socket class.*/
   static private Socket socket;
   /*Declare object of PrintWriter class.*/
   static PrintWriter out = null;
   /*Declare object of BufferedReader class.*/
   static BufferedReader in = null;
   /*Declare object of MutableAttributeSet class.*/
   static MutableAttributeSet sendernameattrib,nameattribut,informationattrib,welcomenameattrib ;
   static Document contentmodel;
   /*Declare string variables.*/
   private String openerhost;
   static String errortype="";
   String roomname="";
   String firstname="";
   String lastname="";
   String emailid="";
   String resource="Home";
   String temproomname="";
   String username="";
   String password="";
   String groupname="";
   String nickname="";
   String servererror="";
   String chattype="";
   String contactpersonname="";
   String jid="";
   String sendername="";
   String tagentity="";
   /*Declare boolean variables*/
   private  static boolean isConnected=false;
   boolean waitForResult=false;
   boolean rosterflag=false;
   boolean waitforreg=false;
   boolean waitforauth=false;
   boolean waitforgroup=false;
   boolean groupnameb=false;
   boolean nouserexists=false;
   /*Declare object of GroupChat class.*/
   GroupChat groupuser;
   /*Declare object of GroupList class.*/
   GroupList group;
   /*Declare object of ChatWindow class.*/
   ChatWindow chat;
   /*Declare object of GroupTree class.*/
   GroupTree dynamic;
   /*Declare object of GroupLoginGUI class.*/
   GroupLoginGUI logingui;
```

```
/*Declare object of nicknamevector class.*/
Vector nicknamevector=new Vector();
/*Declare object of HashMap class.*/
HashMap temphashmap=new HashMap();
/*Declare object of Hashtable class.*/
Hashtable onlineuser=new Hashtable();
/*function to open socket*/
public  SocketConnection openSocket(String hostip,int portnumber,int timeinsec)
{
   sendernameattrib=new SimpleAttributeSet();
   StyleConstants.setFontFamily(sendernameattrib,"Verdana");
   StyleConstants.setForeground(sendernameattrib,Color.black);
   welcomenameattrib=new SimpleAttributeSet();
   StyleConstants.setFontFamily(welcomenameattrib,"Verdana");
   StyleConstants.setForeground(welcomenameattrib,Color.blue);
   nameattribut=new SimpleAttributeSet();
   StyleConstants.setFontFamily(nameattribut,"Verdana");
   StyleConstants.setForeground(nameattribut,Color.red);
   informationattrib=new SimpleAttributeSet();
   StyleConstants.setFontFamily(informationattrib,"Verdana");
   StyleConstants.setItalic(informationattrib,true);
   StyleConstants.setForeground(informationattrib,Color.red);
   socketope();
   try
   {
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(
      socket.getInputStream()));
   }
   catch(Exception exception)
   {
      JOptionPane.showMessageDialog(logingui,"Server not found","Server not
      found",JOptionPane.ERROR_MESSAGE);
   }
   return getSocketClass();
}
/*
Used to create instance of socket class
Parameter: N/A
Return Value: N/A
*/
public  void socketope()
{
   try
   {
      socket=new Socket(openerhost,openerport);
   }
   catch(IOException ie)
   {
   }
}
/* Constructor for class */
public SocketConnection(String host,int port,GroupList group,GroupTree dynamic,
GroupLoginGUI logingui)
{
   this.group=group;
   this.dynamic=dynamic;
   this.logingui=logingui;
   socket=null;
   openerhost=host;
   openerport=port;
}
/* Used to get the reference of socket class*/
public SocketConnection getSocketClass()
{
   return this;
}
/*
Function to write a get error
Parameter: codeid-Object of string class.
Return Value: String
*/
public static String getErrorString(String codeid)
{
   if (codeid=="'401'")
   {
      errortype="minor";
      return "You have sent malformed syntax, which can not be understood by server.";
   }
   if (codeid=="'404'")
   {
      errortype="major";
      return "No Server exist .";
   }
   if (codeid=="'405'")
   {
      errortype="minor";
      return "You have no right to send this command.";
```

```
            }
            if (codeid=="'409'")
            {
               errortype="major";
               return "A user with this jabber id is already exist. Please choose another user id.";
            }
            if (codeid=="'403'")
               {
               errortype="auth";
               return "Invalid password";
            }
            return "";
         }
         /*
         Function used to get the sendername
         Parameter: tagentity-Object of string class.
         Return Value: N/A.
         */
         public void sendername(String tagentity)
         {
            if (tagentity.startsWith("message"))
            {
               sendername="";
               if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
               {
                  sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-2);
               }
            }
         }
         /*
         Function to check xml error
         Parameter: tagentity-Object of string class
         Return Value: String
         */
         public static String checkForError(String tagentity)
         {
         String error="";
         String codeid="";
         if (tagentity.indexOf("error")>0)
         {
         if (tagentity.indexOf("from=")>0)
         {
            codeid=tagentity.substring(tagentity.indexOf("id=")+3,tagentity.indexOf("type="));
         }
      }
      if (tagentity.indexOf("result")>0)
      {
      }
      return error;
      }
      /*
      Used to set the object of GroupUser
      Parameter: gc-Object of GroupChat class
      Return Value: N/A
      */
      public void setGroupuser(GroupChat gc)
      {
         groupuser=gc;
      }
      /*
      Used to read the server output
      Parameter: N/A
      Return Value: N/A
      */
      public void runinput()
      {
         int i=0;
         String errororwarning="";
         String errorstring="";
         String inputstring="";
         boolean starttag=false;
         boolean endtag=false;
         boolean startwritting=false;
         String messagebody="";
         Vector tagvactor;
         tagvactor=new Vector();
         int vactorindex=0;
         int starttagsing=0;
         while (true)
         {
            try
            {
               i=in.read();
               if (i==-1)
               {
                  break;
               }
               else
```

```
{
   inputstring=inputstring+(char)i;
   if ((char)i=='<')
   {
      starttag=true;
      starttagsing=1;
      tagentity="";
   }
   else
   {
      if ((char)i=='/' &&starttag==true )
      {
         if(starttagsing==1)
         {
         starttag=false;
         endtag=true;
         waitforgroup=false;
         if (!groupname.trim().equals(""))
         {
            String contactperson="";
            contactperson=contactpersonname.substring(1,contactpersonname.length()-1);
            group.addToTree(groupname,jid,contactperson);
         }
         jid="";
         contactpersonname="";
         groupname="";
            if(!messagebody.trim().equals(""))
         {
         if (chattype.equals("chat"))
         {
            group.insertTextInToTextPane(messagebody,sendername);
         }
         else
         {
            String username=logingui.getUserName();
            String name[]=sendername.split("/");
            System.out.println("sender name "+name[1]);
            System.out.println("receiver name "+username);
            if(name[1].equals(username))
            {
               groupuser.writeintotextfield("\n"+name[1]+":"+messagebody, nameattribut);
            }
            else
            {
               groupuser.writeintotextfield("\n"+name[1]+":"+messagebody, sendernameattrib)
            }
         }
      }
      startwritting=false;
      messagebody="";
      vactorindex=vactorindex-1;
      if (vactorindex>=0)
      tagvactor.removeElementAt(vactorindex);
   }
   if (starttag)
   {
      stagentity=tagentity+(char)i;
   }
}
else
{
   starttagsing=0;
   if((char)i=='>')
   {
   if (starttag)
   {
   sendername(tagentity);
   errorstring=checkForError(tagentity);
   if((tagentity.indexOf("type='error'")>1) &&
   (tagentity.indexOf("id='1301'")>1))
   {
      String username=logingui.getUserName();
      String password=logingui.getPassword();
      String servarn=logingui.getServarName();
      String registrationstring="<iq type=\"set\" to=\""+username+"@"+servarn+"\"
      id=\"1001\">";
      registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
      registrationstring=registrationstring+"<username>"+username+"</username>";
      registrationstring=registrationstring+"<password>"+password+"</password>";
      registrationstring=registrationstring+"<first>"+firstname+"</first>";
      registrationstring=registrationstring+"<last>"+lastname+"</last>";
      registrationstring=registrationstring+"<email>"+emailid+"</email>";
      registrationstring=registrationstring+" </query> ";
      registrationstring=registrationstring+"</iq>";
      String authentication="<iq type=\"set\" id=\"1301\">";
      authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
      authentication=authentication+" <username>"+username+"</username>";
      authentication=authentication+" <password>"+password+"</password>";
```

```
            authentication=authentication+" <resource>"+resource+"</resource> ";
            authentication=authentication+" </query> ";
            authentication=authentication+"</iq>";
            sendXMLToJabber(authentication);
            sendXMLToJabber(registrationstring);
         }
         if (errorstring.equals("unauthorized"))
         {
            sendRegistration();
            waitforreg=true;
            waitforauth=false;
         }
         if (tagentity.startsWith("message"))
         {
            if (tagentity.indexOf("type='groupchat'")>0)
         {
            chattype="groupchat";
         }
         else
         {
            chattype="chat";
         }
      }
      if((tagentity.indexOf("type='result'")>1)&&(waitforreg)&&
      (!errorstring.equals("user exist")))
      {
         sendAuthorized();
         waitforauth=true;
         waitforreg=false;
      }
      if((tagentity.indexOf("type='result'")>1)&&(waitforauth))
      {
         waitforauth=false;
      }
      if(tagentity.startsWith("presence"))
      {
         if(tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
         {
         nickname=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-1);
         String loginuser[]=nickname.split("/");
         if (tagentity.indexOf("type='unavailable'")>0||tagentity.indexOf
         ("type=\"unavailable\"")>0)
         {
         }
         else
         {
            Enumeration e=group.hasuser.keys();
            while(e.hasMoreElements())
            {
               String haselm=(String)e.nextElement();
               String[] userid=nickname.split("/");
               Object nname=group.hasuser.get(haselm);
               String nnamestr=nname.toString();
               if(userid[1].equals(loginuser[1]))
               {
                  String loginusertoadd=loginuser[1].substring(0,loginuser[1].length()-1);
                  groupuser.addElement(loginusertoadd);
                  break;
               }
               if(userid[1].equals(nnamestr+"'"))
               {
                  groupuser.addElement(nnamestr);
               }
            }
         }
      }
   }
   if(tagentity.indexOf("item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'")!=-1)
   {
      String user=group.getUserName();
      String removeuser;
      removeuser="<iq type='set' id='1'>";
      removeuser=removeuser+"<query xmlns='jabber:iq:roster'>";
      removeuser=removeuser+"<item jid='"+user+"@localhost/Home' name='"+user+"'
      subscription='remove'/>";
      removeuser=removeuser+"</query></iq>";
      sendXMLToJabber(removeuser);
      String removestring;
      JOptionPane.showMessageDialog(group,"No user exists with this user
      id.",errororwarning,JOptionPane.PLAIN_MESSAGE);
      nouserexists=true;
      return;
   }
   if (rosterflag)
   {
   if (tagentity.indexOf("item")!=-1 &&
   (!nouserexists)&&(tagentity.indexOf("affiliation")==-1))
   {
```

```
            contactpersonname=tagentity.substring(tagentity.indexOf("name")+5,
            tagentity.indexOf("jid")).trim();
            jid=tagentity.substring(tagentity.indexOf("jid")+4,tagentity.length());
            waitforgroup=true;
         }
         if (tagentity.indexOf("query")!=-1)
         {
            rosterflag=false;
         }
      }
      if ((tagentity.indexOf("type='result'")>1)&&(!waitforauth)&&(!waitforreg))
      {
         waitForResult=true;
      }
      else
      {
         if (tagentity.indexOf("xmlns='jabber:iq:roster'")>1&&waitForResult)
         {
            rosterflag=true;
            waitForResult=false;
         }
         waitForResult=false;
      }
      if (errorstring.equals("user exist")&&waitforreg)
      {
         JOptionPane.showMessageDialog(null, "A user with this user ID is already exists.Please enter o
         user ID.", errororwarning, JOptionPane.PLAIN_MESSAGE);
      }
      if (!errorstring.equals(""))
      {
         if (errortype=="major")
         {
            errororwarning="Error";
            JOptionPane.showMessageDialog(null, errorstring,errororwarning, JOptionPane.PLAIN_MESSAGE);
         }
         else
         {
            errororwarning="Warning";
            JOptionPane.showMessageDialog(null, errorstring,errororwarning, JOptionPane.PLAIN_MESSAGE);
         }
      }
      startwritting=true;
      tagvactor.insertElementAt(tagentity, vactorindex);
      vactorindex=vactorindex+1;
      starttag=false;
   }
   }
   else
   {
      if (waitforgroup && tagentity.trim().equals("group"))
      {
         groupname=groupname+(char)i;
      }
      if(startwritting==true && tagentity.trim().equals("body"))
      {
         messagebody=messagebody+(char)i;
      }
      if (starttag)
      {
         tagentity=tagentity+(char)i;
      }
   }
   }
   }
   }
   }
   catch(IOException ie)
   {
   }
   }
   isConnected=false;
   }
   /*
   Function to set the object of ChatWindow
   Parameter:chat- Object of ChatWindow class
   Return Value: N/A
   */
   public void setObjChat(ChatWindow chat)
   {
   this.chat=chat;
   }
   public void sendAuthorized()
   {
      String authentication;
      authentication="<iq type=\"set\" id=\"1\">";
      authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
```

```
   authentication=authentication+"<username>"+username+"</username>";
   authentication=authentication+"<password>"+password+"</password>";
   authentication=authentication+"<resource>"+resource+"</resource>";
   authentication=authentication+"</query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
}
/*
Used to send the request to jabber server to get the user list
Parameter: N/A
Return Value: N/A
*/
public void sendRosterRequest()
{
   group.clearTree();
   String rosterstring="";
   rosterstring="<iq type=\"get\" id=\"1\">";
   rosterstring=rosterstring+"<query xmlns=\"jabber:iq:roster\"/></iq>";
   sendXMLToJabber(rosterstring);
}
/*
Used to create a group by sending a message to jabber server
Parameter:group- Object of string class
Return Value: N/A
*/
public void createGroup(String group)
{
   sendXMLToJabber(group);
}
public void sendRegistration()
{
   String registrationstring;
   registrationstring="<iq type=\"set\" to=\""+username+".localhost\" id=\"1\">";
   registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
   registrationstring=registrationstring+"<username>"+username+"</username>";
   registrationstring=registrationstring+"<password>"+password+"</password></
   query></iq>";
}
/*
Used to send the xml messages to jabber server
Parameter:outputmessage- Object of string class
Return Value: N/A
*/
public static void sendXMLToJabber(String outputmessage)
{
   out.println(outputmessage);
   out.flush();
}
}
```

Download this listing.

In the above code, the constructor of the SocketConnection class takes the integer string, the GroupTree class object of the GroupList class, and the GroupLoginGUI class as input parameters. The SocketConnection class allows end users to invoke the methods of the GroupList class, GroupTree class, and GroupLoginGUI class. The host string retrieves the name of the Jabber server and the port integer retrieves the port number of the Jabber server.

The methods defined in Listing 4-7 are:

- openSocket(): Opens a client socket by using the IP address and port number of the Jabber server.
- Socketope(): Instantiates the Socket class to invoke methods of the Socket class.
- getErrorString(): Retrieves the error string sent by the Jabber server.
- Sendername(): Retrieves the sender's name from the input message received from the Jabber server.
- checkForError(): Checks whether or not the input string contains an error message.
- runinput(): Reads the response sent by the Jabber server.

Creating a Group Tree

The GroupTree.java file creates and maintains the tree structure of the available groups and end users present in each group. Listing 4-8 shows the contents of the GroupTree.java file:

Listing 4-8: The GroupTree.java File

```
/*Import required awt classes*/
import java.awt.GridLayout;
import java.awt.Toolkit;
/*Import required swing classes*/
import javax.swing.JPanel;
import javax.swing.JScrollPane;
/*Import required Tree classes*/
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;
/*Import required Tree event classes*/
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeSelectionEvent;
/*Import required util package classes*/
import java.util.*;
/*
class GroupTree - This class is used to create a dynamic tree
Constructor:
GroupTree-This constructor creates a tree.
Methods:
clearTree - Remove all nodes except the root node.
getNodeName - Used to get the  node
addObject-Use to add the node to the tree
*/
public class GroupTree extends JPanel implements TreeSelectionListener
{
/*Declare object of DefaultMutableTreeNode class.*/
protected DefaultMutableTreeNode rootNode;
protected DefaultMutableTreeNode parentNode = null;
protected DefaultTreeModel treeModel;
/*Declare object of JTree class.*/
protected JTree tree;
private Toolkit toolkit = Toolkit.getDefaultToolkit();
/*Declare object of SocketConnection class.*/
SocketConnection clientsocket;
/*Declare object of GroupLoginGUI class.*/
GroupLoginGUI groupLogin;
/*Declare object of GroupList class.*/
GroupList group;
/*Declare object of ChatWindow class.*/
ChatWindow ch;
/*Declare the string variable*/
String user="";
String servername="";
String resource="Home";
/*Declare object of HashMap class.*/
HashMap mapObj=new HashMap();
/*Declare object of Vector class.*/
Vector vec=new Vector();
/*Constructor for class*/
public GroupTree(SocketConnection clientsocket,GroupLoginGUI groupLogin,GroupList group)
{
   /*Set the window title*/
   super(new GridLayout(1,0));
   this. clientsocket=clientsocket;
   this.groupLogin=groupLogin;
   this.group=group;
   /*Create the object of DefaultMutableTreeNode to create root node*/
   rootNode = new DefaultMutableTreeNode("Group");
   /*Add the rootnode into treeModel*/
   treeModel = new DefaultTreeModel(rootNode);
   /*Add the treeModelListener to treemodel*/
   treeModel.addTreeModelListener(new MyTreeModelListener());
   /*Add the treeModel into the tree*/
   tree = new JTree(treeModel);
   /*Set the tree in editable mode*/
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode
   (TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);
   tree.addTreeSelectionListener(this);
```

```
   /*Add the tree to the scrollpane*/
   JScrollPane scrollPane = new JScrollPane(tree);
   /*Add the scrollpane to container*/
   add(scrollPane);
}
/*
Remove all nodes except the root node.
Parameter: N/A
Return Value: N/A
*/
public void clearTree()
{
   rootNode.removeAllChildren();
   treeModel.reload();
}
/*
Function to get the username
Parameter: N/A
Return Value: String
*/
public String getNodeName()
{
   String user="";
   TreePath currentSelection = tree.getSelectionPath();
   if (currentSelection != null)
      {
         DefaultMutableTreeNode currentNode = (DefaultMutableTreeNode)
         (currentSelection.getLastPathComponent());
         MutableTreeNode parent = (MutableTreeNode)(currentNode.getParent());
         if (parent != null)
         {
            user=currentNode.toString();
         }
      }
      return user;
   }
/*
Used to add node to the tree
Parameter: child-Object of Object class
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(Object child)
{
DefaultMutableTreeNode parentNode = null;
TreePath parentPath = tree.getSelectionPath();
if (parentPath == null)
{
   parentNode = rootNode;
}
else
{
   parentNode = (DefaultMutableTreeNode)
   (parentPath.getLastPathComponent());
}
return addObject(parentNode, child, true);
}
/*
Used to add node to the tree
Parameter: parent-Object of DefaultMutableTreeNode class
child-Object of Object class
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent,Object child)
{
   return addObject(parent, child, true);
}
/*
Used to add node to the tree
Parameter: parent-Object of DefaultMutableTreeNode class
child-Object of Object class
shouldBeVisible-boolean
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent, Object child,
boolean shouldBeVisible)
{
   DefaultMutableTreeNode childNode =new DefaultMutableTreeNode(child);
   if (parent == null)
{
   parent = rootNode;
}
treeModel.insertNodeInto(childNode, parent,parent.getChildCount());
if(shouldBeVisible)
{
```

```
   tree.scrollPathToVisible(new TreePath(childNode.getPath()));
}
return childNode;
}
/* Inner class for event listener */
class MyTreeModelListener implements TreeModelListener
{
public void treeNodesChanged(TreeModelEvent e)
{
   DefaultMutableTreeNode node;
   node = (DefaultMutableTreeNode)(e.getTreePath().getLastPathComponent());
   try
   {
      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)(node.getChildAt(index));
   }
   catch (NullPointerException exc)
   {
   }
}
public void treeNodesInserted(TreeModelEvent e)
{
}
public void treeNodesRemoved(TreeModelEvent e)
{
}
public void treeStructureChanged(TreeModelEvent e)
{
}
}
/*
Method to describe the event
Parameter: e-Object of TreeSelectionEvent class
Return Value:N/A
*/
public void valueChanged(TreeSelectionEvent e)
{
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
   if (node!=null)
   {
      TreePath currentSelection = tree.getAnchorSelectionPath();
      TreeNode parent =node.getParent();
      TreeNode parentnode =node.getRoot();
      Enumeration enu=parentnode.children();
      boolean b=node.isLeaf();
      TreeNode root =node.getParent();
      if((b) &&!(parent.toString().equals("Group")))
      {
         String receiver=node.toString();
         user=groupLogin.getUserName();
         servername=groupLogin.getServarName();
         ch=new ChatWindow(user,receiver,servername,clientsocket);
         mapObj.put(receiver,ch);
      }
      else
      {
         if((root!=null) &&(parent.toString().equals("Group")))
         {
            DefaultMutableTreeNode groupuser = (DefaultMutableTreeNode)
            tree.getLastSelectedPathComponent();
            user=groupLogin.getUserName();
            servername=groupLogin.getServarName();
            String roomname=node.toString();
            GroupChat gp=new GroupChat(user,servername,roomname,clientsocket);
         }
      }
   }
}
/*
Used to set the ChatWindow Object
Parameter: ch-Object of ChatWindow class
Return Value:N/A
*/
public void setChat(ChatWindow ch)
{
   this.ch=ch;
}
/*
Used to get the ChatWindow Object
Parameter: N/A
Return Value:N/A
*/
public ChatWindow getChat()
{
   return this.ch;
}
}
```

Download this listing.

In the above code, the constructor of the GroupTree class takes an object of the GroupList class, GroupLoginGUI class, and SocketConnection class as input parameters. This allows end users to invoke the methods of the GroupList class, GroupLoginGUI class, and SocketConnection class.

The methods defined in Listing 4-8 are:

- clearTree(): Refreshes the tree structure containing group and end user names for each group.
- getNodeName(): Retrieves the name of the group selected by an end user.
- addObject(): Adds end users or groups to the tree structure.
- setChat(): Sets the object of the ChatWindow class.

The GroupTree.java file creates and maintains the tree structure of the available groups and end users in each group, as shown in Figure 4-9:

Figure 4-9: Creating a Group Tree

PREVIOUS NEXT

Unit Testing

To test the Group Chatting application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of Java 2 SDK (J2SDK) by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the GroupLoginGUI.java, GroupList.java, AddUser.java, GroupChat.java, EditGroup.java, GroupTree.java, ChatWindow.java, and SocketConnection.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the javac command, as shown:

   ```
   javac *.java
   ```

5. To run the Group Chatting application, specify the following command at the command prompt:

   ```
   java GroupLoginGUI
   ```

6. The Login window of the Group Chatting application appears. Specify the login information, as shown in Figure 4-10:

Figure 4-10: Specifying Login Information

7. Click the Submit button to send the login information to the Jabber Server.

8. The user interface of the Group Chatting application appears. Select File->Add User to add other end users to the selected group.

9. Enter the name of another end user and the group name to add the specified user to a group, as shown in Figure 4-11:

Figure 4-11: Adding an End User to a Group

10. Click the Submit button to add an end user to a group.

11. Select File->Edit Group to rename a group.

12. Enter the name of the old group and the new name to rename an existing group, as shown in Figure 4-12:

Figure 4-12: Entering the Old and New Group Names

13. Click the Submit button to rename the group. The group with the new name appears, as shown in Figure 4-13:

Figure 4-13: Renamed Group

14. Select an end user name from the tree structure of the available groups to initiate a private chat.
15. Type the message that you want to send and click the Send button to send the text message to the selected end user, as shown in Figure 4-14:

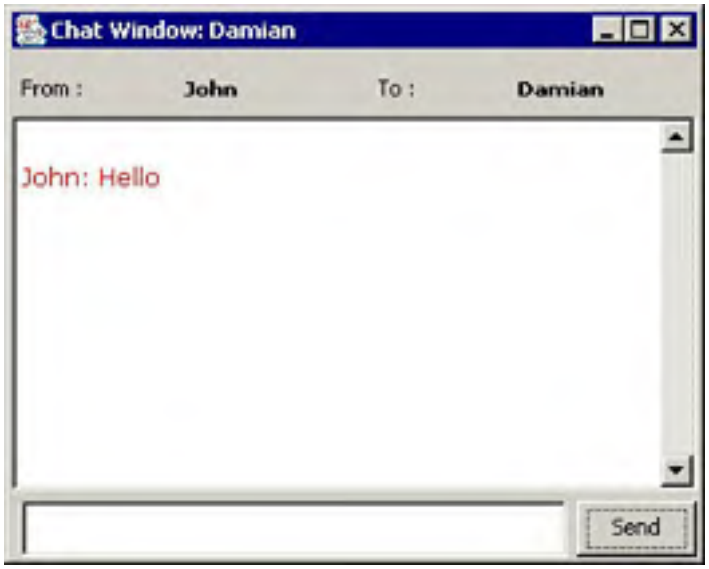


Figure 4-14: Sending a Private Message

16. Select a specific group from the tree structure of the available groups to participate in a group chat.
17. Type the required message and click the Send button to send the text message to all end users logged on to the selected group, as shown in Figure 4-15:

Figure 4-15: Sending a Group Message

PREVIOUS NEXT

Chapter 5: Creating an Instant Technical Support Application

Download CD Content

The Jabber protocol allows you to develop applications for instant messaging between end users through the Jabber server. This chapter describes how to develop an Instant Technical Support application that allows end users to send messages to a technical support executive for online help.

Architecture of the Instant Technical Support Application

The Instant Technical Support application provides a user interface to send and receive messages by using a Jabber server. The Instant Technical Support application allows end users to change font, style, and size of the text.

The Instant Technical Support application uses the following files:

- welcome.html: Creates the home page for the Instant Technical Support application.
- signuppage.html: Creates a Web interface that allows end users to sign up for a new account.
- loginpage.html: Creates a Web interface that allows end users to login.
- Chathtml.html: Sends the information specified by end users to the ChatMainApplet.java file.
- ChatMainApplet.java: Creates the user interface for the Instant Technical Support application that allows end users to establish a connection with the Jabber server. The ChatMainApplet.java file allows end users to communicate with the technical support executive connected to the Jabber server.

Figure 5-1 shows the architecture of the Instant Technical Support application:

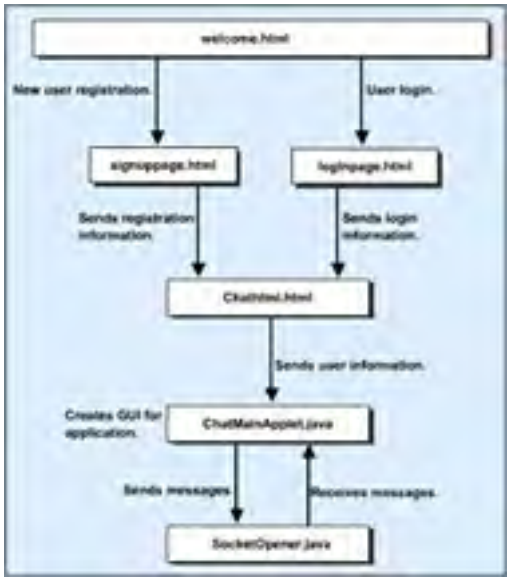


Figure 5-1: Architecture of the Instant Technical Support Application

If an end user clicks the Sign up button on the home page, the welcome.html file calls the signuppage.html file, which allows the end users to sign up for a new account. The signuppage.html file provides an interface with various labels, text boxes, and two buttons, OK and Cancel.

If an end user clicks the Click here button on the home page, the welcome.html page calls the loginpage.html file, which allows existing end users to login. The loginpage.html file provides an interface with various labels, two text boxes, and the Submit button.

Creating the Home Page

The welcome.html file creates the home page for the Instant Technical Support application. Listing 5-1 shows the contents of the welcome.html file:

Listing 5-1: The welcome.html File

```
<HTML>
<HEAD>
<TITLE>
Welcome to Instant Technical Support
</TITLE>
<SCRIPT language="javascript">
function openwindow(pagename,windowname)
{
   window.open(pagename, windowname, 'width=500, height=300, left=300, top=100, screenX=500,
   screenY=100');
}
</SCRIPT>
</HEAD>
<BODY>
<TABLE WIDTH="100%" CELLSPACING=0 CELLPADDING=0 BORDER=0>
<TR>
<TD>
<TABLE WIDTH="100%" CELLSPACING=0 CELLPADDING=0 BORDER=0>
<TR>
<TD ALIGN="CENTER">
<FONT SIZE="5" FACE="verdana" COLOR="">
<U>
Welcome to Instant Technical Support
</U>
</FONT>
</TD>
</TR>
</TABLE>
</TD>
</TR>
<TR>
<TD>
 
</TD>
</TR>
<TR>
<TD>
 
</TD>
</TR>
<TR>
<TD>
<TABLE WIDTH="100%" CELLSPACING=3 CELLPADDING=0 BORDER=0 ALIGN="CENTER" >
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="60%" ALIGN="CENTER" colspan=2>
<FONT SIZE="4" COLOR="">
</FONT>
</TD>
<TD WIDTH="40%" ALIGN="LEFT">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
```

```
 
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="LEFT">
Existing User
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
<INPUT TYPE="BUTTON" NAME="existinguser" style="width:100" VALUE="Click here"
onclick="javascript:openwindow('loginpage.html', 'LoginPage');">
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="LEFT">
New User
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
<INPUT TYPE="BUTTON" NAME="newuser" style="width:100" VALUE="Sign up"
onclick="javascript:openwindow('signuppage.html', 'NewUser');">
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="20%">
 
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Download this listing.

The welcome.html file creates the home page for the Instant Technical Support application, as shown in Figure 5-2:

Figure 5-2: The Home Page

Click the Click here button to login as an existing user.

Click the Sign up button to sign up for a new account.

Creating the Signup Page

The signuppage.html file creates the sign up page that allows end users to create a new account. Listing 5-2 shows the contents of the signuppage.html file:

Listing 5-2: The signuppage.html File

```
<HTML>
<HEAD>
<TITLE>Signup Page</TITLE>
<SCRIPT language="javascript">
function openwindow(pagename,windowname)
{
   if(document.profileenteryform.username.value=="")
   {
      alert('User name is a required field.');
      return false;
   }
   if(document.profileenteryform.password.value=="")
   {
      alert('Password is a required field.');
      return false;
   }
   if(document.profileenteryform.cpassword.value=="")
   {
      alert('Confirm Password is a required field.');
      return false;
   }
   if(document.profileenteryform.password.value!=document.profileenteryform.password.value)
   {
      alert('Password and Confirm Password must be same.');
      return false;
   }
   if(document.profileenteryform.firstname.value=="")
   {
      alert('First name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.firstname.value))
      {
         alert('Please enter a valid value for first name.');
         return false;
      }
   }
   if(document.profileenteryform.lastname.value=="")
   {
      alert('Last Name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.lastname.value))
      {
         alert('Please enter a valid value for last name.');
         return false;
      }
   }
   if(document.profileenteryform.emailid.value=="")
   {
      alert('Email Id is a required field.');
      return false;
   }
   else
   {
   if(!isEmailAddr(document.profileenteryform.emailid.value))
   {
      alert('Please enter a valid email ID.');
      return false;
   }
}
function checkForInt(valuestring)
{
   var validcharstring=false;
   var intstring="0123456789!@#$%^&*()_-+="
   for(i=0;i<valuestring.length;i++)
   {
      if (intstring.indexOf(valuestring.charAt(i))!=-1)
      {
         return validcharstring;
      }
   }
   validcharstring=true;
```

```
   return validcharstring;
}
function isEmailAddr(email)
{
   var result = false;
   var theStr = new String(email);
   var index = theStr.indexOf("@");
   if (index > 0)
   {
      var pindex = theStr.indexOf(".",index);
      if ((pindex > index+1) && (theStr.length > pindex+1))
      result = true;
   }
   return result;
}
user_name=document.profileenteryform.username.value;
pwd=document.profileenteryform.password.value;
fname=document.profileenteryform.firstname.value;
lname=document.profileenteryform.lastname.value;
emailid=document.profileenteryform.emailid.value;
pagename=pagename+"?username="+user_name+"&password="+pwd+"&firstname="+fname+"&lastname="
+lname+"&emailid="+emailid;
window.open(pagename, windowname,' width=470, height=515, left=300, top=100,screenX=30, screenY=100'
this.close();
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="profileenteryform">
<TABLE WIDTH="100%">
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
<FONT SIZE="4" COLOR="">
Signup for New Account
</FONT>
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD>
<TABLE ALIGN="CENTER" WIDTH="100%" >
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="20%">
User Name
</TD>
<TD WIDTH="40%">
<INPUT TYPE="INPUT" NAME="username" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Password
</TD>
<TD WIDTH="30%">
<INPUT TYPE="password" NAME="password" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
```

```
 
</TD>
<TD WIDTH="30%">
Confirm Password
</TD>
<TD WIDTH="30%">
<INPUT TYPE="PASSWORD" NAME="cpassword" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
First Name
</TD>
<TD WIDTH="30%">
<INPUT TYPE="INPUT" NAME="firstname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Last Name
</TD>
<TD WIDTH="30%">
<INPUT TYPE="INPUT" NAME="lastname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Email ID
</TD>
<TD WIDTH="30%">
<INPUT TYPE="INPUT" NAME="emailid" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
 
</TD>
<TD WIDTH="30%" colspan=2>
<INPUT TYPE="BUTTON" NAME="ok" style="width:30%" VALUE="OK"
onclick="javascript:openwindow('chathtml.html', 'chatpage');">
<INPUT TYPE="BUTTON" NAME="cancel"  style="width:30%" VALUE="Cancel"
onclick="javascript:window.close();">
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Download this listing.

The Signup.html page creates the signup page, as shown in Figure 5-3:

Figure 5-3: The Signup Page

◄ PREVIOUS NEXT ►

Creating the Login Page

The loginpage.html file creates the login page to help end users login as existing users. Listing 5-3 shows the contents of the loginpage.html file:

Listing 5-3: The loginpage.html File

```
<HTML>
<HEAD>
<TITLE>Login Page</TITLE>
<SCRIPT language="javascript">
function openwindow(pagename,windowname)
{
   if(document.loginform.username.value=="")
   {
      alert('User name is a required field.');
      return false;
   }
   if(document.loginform.username.value=="")
   {
      alert('Password is a required field.');
      return false;
   }
   user_name=document.loginform.username.value;
   pwd=document.loginform.password.value;
   pagename=pagename+"?username="+user_name+"&password="+pwd;
   window.open(pagename,windowname,'width=470,height=515,left=300,top=100,screenX=30,screenY=100');
   this.close();
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="loginform">
<TABLE WIDTH="100%" ALIGN="CENTER">
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%" colspan=2>
<FONT SIZE="4" COLOR="">
Enter your user name and password to login
</FONT>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
User Name
</TD>
<TD WIDTH="45%">
<INPUT TYPE="input" NAME="username" VALUE="" maxlength="100">
</TD>
<TD WIDTH="10%">
```

```
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
Password
</TD>
<TD WIDTH="45%">
<INPUT TYPE="password" NAME="password" VALUE="" maxlength="100">
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="45%">
<INPUT TYPE="BUTTON" NAME="okbutton" style="width:100" VALUE="OK"
onclick="javascript:openwindow('chathtml.html','chat');">
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Download this listing.

The loginpage.html page creates the login page, as shown in Figure 5-4:

Figure 5-4: The Login Page

Sending Login Information to the Jabber Server

The chathtml.html file calls the ChatMainApplet.java file to send the login information specified by an end user to the Jabber server. Listing 5-4 shows the contents of the chathtml.html file:

Listing 5-4: The chathtml.html File

```
<HTML>
<HEAD>
<TITLE>Instant Technical Support Application</TITLE>
<SCRIPT>
function PageQuery(q)
{
   if(q.length > 1)
   this.q = q.substring(1, q.length);
   else
   this.q = null;
   this.keyValuePairs = new Array();
   if(q)
   {
      for(var i=0; i <this.q.split("&").length; i++)
      {
      this.keyValuePairs[i] = this.q.split("&")[i];
      }
   }
   this.getKeyValuePairs = function()
   {
      return this.keyValuePairs;
   }
   this.getValue = function(s)
   {
      for(var j=0; j <this.keyValuePairs.length; j++)
         {
         if(this.keyValuePairs[j].split("=")[0] == s)
         return this.keyValuePairs[j].split("=")[1];
      }
      return false;
   }
   this.getParameters = function()
   {
      var a = new Array(this.getLength());
      for(var j=0; j <this.keyValuePairs.length; j++)
      {
         a[j] = this.keyValuePairs[j].split("=")[0];
      }
      return a;
   }
   this.getLength = function()
   {
      return this.keyValuePairs.length;
   }
}
function queryString(key)
{
   var page = new PageQuery(window.location.search);
   return page.getValue(key);
}
function appletparam(key)
{
if(queryString(key)=='false')
{
   pagename='loginpage.html';
   windowname='login';
   window.open(pagename, windowname,'width=400, height=200, left=300,top=100, screenX=500,
   screenY=100');
}
else
{
   return queryString(key);
}
}
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
function writeAppletTag()
{
   document.writeln('<applet code="ChatMainApplet" width="450" height="500">');
   document.writeln(buildParamTag('username', appletparam('username')));
   document.writeln(buildParamTag('password', appletparam('password')));
   document.writeln(buildParamTag('firstname', appletparam('firstname')));
   document.writeln(buildParamTag('lastname', appletparam('lastname')));
   document.writeln(buildParamTag('emailid', appletparam('emailid')));
   document.writeln('</APPLET>');
}
function buildParamTag(name, value)
```

```
{
   return '<PARAM NAME="' + name + '" VALUE="' + value + '">';
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
writeAppletTag();
</SCRIPT>
</BODY>
</HTML>
```

Download this listing.

In the above code, the writeAppletTag() function calls the ChatMainApplet.java file to create the user interface of the Instant Technical Support application.

◄ PREVIOUS NEXT ►

Creating the User Interface of the Instant Technical Support application

The ChatMainApplet.java file creates the user interface for the Instant Technical Support application that allows end users to exchange messages with the technical support executive through the Jabber server. Listing 5-5 shows the contents of the ChatMainApplet.java file:

Listing 5-5: The ChatMainApplet.java File

```
/*Imports required javax.swing package classes.*/
import javax.swing.*;
/*
Imports required javax.swing.event package classes
*/
import javax.swing.event.*;
/*Imports required javax.awt package classes.*/
import java.awt.*;
/*Imports required java.awt.event package classes.*/
import java.awt.event.*;
/*Imports required java.io package classes.*/
import java.io.*;
/*Imports required java.net classes.*/
import java.net.*;
/*Imports required io java.util classes.*/
import java.util.*;
/*Imports required javax.swing.text package classes.*/
import javax.swing.text.*;
/*
class ChatMainApplet - This class is the main class of the application. This class initializes
the interface and loads all components like the button, textfields before displaying the result.
Methods:
sendXMLToJabber - Send message to jabber server.
sendmessage - Convert message into XML format.
checkForError - Check for error message.
readDate - Read next message from socket.
*/
public class ChatMainApplet extends JApplet implements ItemListener,Runnable,ActionListener,KeyListe
{
/*Declare object of JTextField class.*/
JTextField inputtext;
/*Declare object of JTextPane class.*/
JTextPane output;
/*Declare object of JScrollPane class.*/
JScrollPane outputscrollpane;
/*Declare objects of JComboBox class.*/
private JComboBox fontChoice;
private JComboBox styleChoice;
private JComboBox sizeChoice;
/*Declare objects of JButton class.*/
Socket clientsocket;
/*Declare objects of JButton class.*/
Button sendbutton;
Button closebutton;
/*Declare object of PrintWriter class.*/
PrintWriter out = null;
BufferedReader in = null;
Thread inputmessagethread;
/*Declare object of Document class.*/
Document contentmodel;
/*Declares object of JPanel class.*/
JPanel panel;
JPanel bottompanel;
/*
Declare object of MutableAttributeSet class.
*/
MutableAttributeSet sendernameattrib,recervernameatrib,normaltextattrib;
/*Declare object of String class.*/
String ipaddress;
String recerverid;
String username;
String password;
String resource;
String firstname="";
String lastname="";
String emailid="";
String errortype="";
boolean isError=false;
private  boolean isConnected=false;
JLabel fontlabel;
JLabel fromlabel;
int portno;
```

```
int wait;
public void init()
{
/*
Initializes and sets the Look and Feel of the application to Windows Look and Feel.
*/
try
{
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
   System.out.println("Unable to set WLF"+e);
}
/*
Declares and initializes object of Container class.
*/
Container contenpane=getContentPane();
/*
Sets container layout as BorderLayout.
*/
contenpane.setLayout(new BorderLayout());
/*
Initializes object of JTextPane class.
*/
output = new JTextPane();
/* Sets background color to white.*/
output.setBackground(Color.white);
/*
Initializes object of the JScrollPane class.
*/
outputscrollpane=new JScrollPane(output,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
output.setEditable(false);
/*
Add outputscrollpane to contentpanel.
*/
contenpane.add(outputscrollpane, "Center");
/*
Initializes object of the JPanel class.
*/
panel = new JPanel();
/*
Initializes object of the JLabel class.
*/
fontlabel=new JLabel(" Select Font ");
/*
Add fontlabel to the panel.
*/
panel.add(fontlabel);
/*
Declares and initializes a String array containing the font list.
*/
String[] fontNames = getToolkit().getFontList();
String defaultFontName = getFont().getName();
/*
Initializes object of the JComboBox class.
*/
fontChoice = new JComboBox();
fontChoice.addItem( "" );
fontChoice.setSelectedIndex( 0 );
for ( int i = 0; i < fontNames.length; ++i )
{
   fontChoice.addItem( fontNames[i] );
   if ( fontNames[i].equals( defaultFontName ) )
   fontChoice.setSelectedIndex( i + 1 );
}
panel.add( fontChoice );
/*
Initializes object of the JComboBox class.
*/
styleChoice = new JComboBox();
/*
Add items to the styleChoice combobox.
*/
styleChoice.addItem( "Plain" );
styleChoice.addItem( "Bold" );
styleChoice.addItem( "Italic" );
styleChoice.addItem( "Bold Italic" );
styleChoice.setSelectedIndex(0);
/*Add object of JLabel to the panel.*/
panel.add(new JLabel("Select Style"));
/*
Add object of JComboBox to the panel.
*/
panel.add(styleChoice);
/*Add object of JLabel to the panel.*/
```

```
panel.add(new JLabel("Select size"));
/*
Initializes object of the JComboBox class.
*/
sizeChoice = new JComboBox();
/*Add items to the sizeChoice combobox.*/
sizeChoice.addItem( "6" );
sizeChoice.addItem( "8" );
sizeChoice.addItem( "10" );
sizeChoice.addItem( "12" );
sizeChoice.addItem( "15" );
sizeChoice.addItem( "20" );
sizeChoice.addItem( "25" );
sizeChoice.setSelectedIndex( 3 );
/*
Add object of JComboBox to the panel.
*/
panel.add(sizeChoice);
/*Add action listener to sizeChoice.*/
sizeChoice.addActionListener(this);
/*Add action command to sizeChoice.*/
sizeChoice.setActionCommand("size");
/*Add action listener to styleChoice.*/
styleChoice.addActionListener(this);
/*Add action command to styleChoice.*/
styleChoice.setActionCommand("style");
/*Add action listener to fontChoice.*/
fontChoice.addActionListener(this);
/*Add action command to fontChoice.*/
fontChoice.setActionCommand("font");
/*
Initializes object of JTextField class.
*/
inputtext = new JTextField();
/*Add key listener to the inputtext.*/
inputtext.addKeyListener(this);
/*Set button size.*/
inputtext.setPreferredSize(new Dimension(20, 30));
/*
Initializes object of the JPanel class.
*/
bottompanel=new JPanel();
/*Set panel size.*/
bottompanel.setPreferredSize(new Dimension(20, 120));
/*Initialize object of GridLayout.*/
GridLayout gridlayout=new GridLayout(2, 1, 0, 0);
/*Sets  BorderLayout layout to container.*/
bottompanel.setLayout(gridlayout);
/*Set panel size.*/
bottompanel.setPreferredSize(new Dimension(20, 120));
bottompanel.add(panel);
/*
Declare and initializes object of BorderLayout.
*/
BorderLayout bottomgridlayout=new BorderLayout();
/*
Declare and initializes object of JPanel.
*/
JPanel subbottompanel=new JPanel();
/*Set background color to white.*/
subbottompanel.setBackground(Color.WHITE);
/*Sets Layout as BorderLayout.*/
subbottompanel.setLayout(bottomgridlayout);
/*Sets panel size.*/
subbottompanel.setPreferredSize(new Dimension(20, 120));
/*
Add object of JLabel class to the panel.
*/
subbottompanel.add(new JLabel(" "), BorderLayout.NORTH);
/*
Add object of JTextField class to the panel.
*/
subbottompanel.add(inputtext,BorderLayout.CENTER);
/*
Declares and initializes object of GridLayout.
*/
JPanel sendbuttonpanel=new JPanel(new FlowLayout(1, 1, 3));
/*
Initializes object of Button class.
*/
sendbutton=new Button("Send");
/*Add action listener to button.*/
sendbutton.addActionListener(this);
/*Add action command to sendbutton.*/
```

```
sendbutton.setActionCommand("send");
sendbutton.setEnabled(true);
/* Sets button size.*/
sendbutton.setSize(new Dimension(10, 10));
/*
Add object of Button class to the JPanel.
*/
sendbuttonpanel.add(sendbutton);
/*Set background color to white.*/
sendbuttonpanel.setBackground(Color.WHITE);
/*
Add object of JPanel class to the JPanel.
*/
subbottompanel.add(sendbuttonpanel, BorderLayout.EAST);
/*
Add object of JLabel class to the JPanel.
*/
subbottompanel.add(new JLabel(" "), BorderLayout.SOUTH);
/*
Add object of JPanel class to the JPanel.
*/
bottompanel.add(subbottompanel);
/*
Add object of JPanel class to the ContentPane.
*/
contenpane.add(bottompanel,"South");
ipaddress="localhost";
portno=5222;
wait=5000;
resource="Home";
/*Sets jabber id.*/
recerverid="tcuser1@localhost";
username=getParameter("username");
password=getParameter("password");
firstname=getParameter("firstname");
lastname=getParameter("lastname");
emailid=getParameter("emailid");
/*
Initializes object of the SimpleAttributeSet class.
*/
recervernameatrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(recervernameatrib, "Verdana");
StyleConstants.setForeground(recervernameatrib, Color.red);
/*
Initializes object of SimpleAttributeSet class.
*/
normaltextattrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(normaltextattrib, "Verdana");
StyleConstants.setForeground(normaltextattrib, Color.black);
contentmodel=output.getDocument();
/*
Initializes object of SimpleAttributeSet class.
*/
sendernameattrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(sendernameattrib, "Verdana");
StyleConstants.setForeground(sendernameattrib, Color.blue);
String labelfromstring=" From: "+username;
String  labeltostring="To: Technical Support";
/*Initializes object of JLabel class.*/
fromlabel=new JLabel(labelfromstring,JLabel.LEFT);
fromlabel.setFont(new Font("verdana", 0, 12));
/*
Declares and initializes object of JLabel class.
*/
JLabel tolabel=new JLabel(labeltostring);
tolabel.setFont(new Font("verdana", 0, 12));
/*
Declares and initializes object of JPanel class.
*/
JPanel toppanel=new JPanel(new GridLayout(1, 2, 0, 10));
/*
Add object of JLabel class to the JPanel.
*/
toppanel.add(fromlabel);
/*
Add object of JLabel class to the JPanel.
*/
toppanel.add(tolabel);
/*
Add object of JPanel class to the ContenPane.
*/
contenpane.add(toppanel, BorderLayout.NORTH);
Font newfont=getNewFont();
inputtext.setFont(newfont);
startSession();
}
public void actionPerformed(ActionEvent ae)
{
```

```
   String source=ae.getActionCommand();
   if (source=="exit")
   {
      isConnected=false;
   }
   if (source=="send")
   {
      sendmessage();
   }
   if (source=="font"||source=="style"||source=="size")
   {
      inputtext.setFont(getNewFont());
   }
}
/*
sendmessage - This method is used to insert message into textpane and calls sendXMLToJabber function
Parameters: NA.
Return Value: NA.
*/
public void sendmessage()
{
   String outputstring;
   outputstring="<message to=\""+recerverid+"\">";
   outputstring=outputstring+"<body>";
   outputstring=outputstring+ inputtext.getText();
   outputstring=outputstring+"</body>";
   outputstring=outputstring+"</message>";
   try
   {
      contentmodel.insertString(contentmodel.getLength(), username+":",recervernameatrib);
   }
   catch(BadLocationException ble)
   {
   }
   output.setCharacterAttributes(normaltextattrib,true);
   try
   {
   if (((String)fontChoice.getSelectedItem()).trim().equals(""))
   {
      StyleConstants.setFontFamily(recervernameatrib,"Verdana");
      StyleConstants.setForeground(recervernameatrib,Color.red);
   }
   StyleConstants.setFontFamily(normaltextattrib, (String)fontChoice.getSelectedItem());
   StyleConstants.setFontSize(normaltextattrib,
   Integer.parseInt((String)sizeChoice.getSelectedItem()));
   if (styleChoice.getSelectedItem().equals("Bold"))
   {
      StyleConstants.setBold(normaltextattrib,true);
   }
   else if(styleChoice.getSelectedItem().equals("Italic"))
   {
      StyleConstants.setItalic(normaltextattrib,true);
   }
   else if(styleChoice.getSelectedItem().equals("Bold Italic"))
   {
      StyleConstants.setBold(normaltextattrib,true);
      StyleConstants.setItalic(normaltextattrib,true);
   }
   contentmodel.insertString(contentmodel.getLength(),inputtext.getText(), normaltextattrib);
}
catch(BadLocationException ble)
{
}
try
{
   contentmodel.insertString(contentmodel.getLength(), "\n",normaltextattrib);
}
catch(BadLocationException ble)
{
}
sendXMLToJabber(outputstring);
inputtext.setText("");
}
/*
sendXMLToJabber - This method is used to send xml message to jabber server.
Parameters: outputmessage - objcet of String class.
Return Value: NA
*/
public void sendXMLToJabber(String outputmessage)
{
   String[] tokenizerstring=outputmessage.split("\n");
   for (int i=0;i<tokenizerstring.length;i++ )
   {
      try
      {
         out.println(tokenizerstring[i]);
         out.flush();
      }
```

```
      catch(Exception e)
      {
         System.out.println("error");
         return;
      }
      out.flush();
   }
}
/*
startSession - This method is used to send xml message to jabber server.
Parameters: NA
Return Value: NA
*/
public void startSession()
{
   openPort(ipaddress, portno, wait);
   if (clientsocket!=null)
   {
      isConnected=true;
   }
   try
   {
      /*
      Initializes object of PrintWriter class.
      */
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      /*
      Initializes object of BufferedReader class.
      */
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
      /* Initializes object of Thread class.*/
      inputmessagethread=new Thread(this);
      /* Starts a new thread */
      inputmessagethread.start();
      /* Declare objects of String class.*/
      String sessionStratString;
      String authentication;
      String registrationstring="";
      sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
      sessionStratString=sessionStratString+" <stream:stream";
      sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
      sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
      sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
      sendXMLToJabber(sessionStratString);
      if ((firstname.equals("false"))||(lastname.equals("false"))||(emailid.equals("false")))
      {
      }
      else
      {
         registrationstring="<iq type=\"set\" to=\""+username+".localhost\" id=\"1001\">";
         registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
         registrationstring=registrationstring+"<username>"+username+"</username>";
         registrationstring=registrationstring+"<password>"+password+"</password>";
         registrationstring=registrationstring+"</query> ";
         registrationstring=registrationstring+"</iq>";
         sendXMLToJabber(registrationstring);
      }
      authentication="<iq type=\"set\" id=\"1001\">";
      authentication=authentication+"<query xmlns=\"jabber:iq:auth\">";
      authentication=authentication+"<username>"+username+"</username>";
      authentication=authentication+"<password>"+password+"</password>";
      authentication=authentication+"<resource>"+resource+"</resource> ";
      authentication=authentication+"</query> ";
      authentication=authentication+"</iq>";
      sendXMLToJabber(authentication);
   }
   catch(IOException ie)
   {
   }
}
/*
run - This method is called when a new thread is started.
Parameters: NA
Return Value: NA
*/
public void run()
{
   int i=0;
   String errororwarning="";
   String errorstring="";
   String inputstring="";
   String messagebody="";
   String tagentity="";
   boolean starttag=false;
   boolean endtag=false;
   boolean startwritting=false;
   Vector tagvactor=new Vector();
   int vactorindex=0;
```

```
int starttagsing=0;
while (true)
{
try
{
   i=in.read();
   if (i==-1)
   {
      break;
   }
   else
   {
   inputstring=inputstring+(char)i;
   if ((char)i=='<')
   {
      starttag=true;
      starttagsing=1;
      tagentity="";
   }
   else
   {
   if ((char)i=='/' && starttag==true )
   {
      if (starttagsing==1)
      {
         starttag=false;
         endtag=true;
         if (!messagebody.trim().equals(""))
         {
            writemessage(messagebody+"\n",normaltextattrib);
         }
         startwritting=false;
         messagebody="";
         vactorindex=vactorindex-1;
         if (vactorindex>=0)
         tagvactor.removeElementAt(vactorindex);
         }
      }
      else
      {
         starttagsing=0;
         if((char)i=='>')
         {
         if (starttag)
         {
            sendername(tagentity);
            errorstring=checkForError(tagentity);
            if (!errorstring.equals(""))
            {
               if (errortype=="major")
               {
                  errororwarning="Error";
                  JOptionPane.showMessageDialog(null,"No user exixts for this
                  userid.", errororwarning,JOptionPane.PLAIN_MESSAGE);
                  sendbutton.setEnabled(false);
                  fromlabel.setText(" From: ");
                  stop();
                  destroy();
               }
               else
               {
                  errororwarning="Warning";
                  JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                  JOptionPane.PLAIN_MESSAGE);
                  sendbutton.setEnabled(false);
                  fromlabel.setText(" From: ");
                  stop();
                  destroy();
               }
            }
            else
            {
            }
            startwritting=true;
            tagvactor.insertElementAt(tagentity,vactorindex);
            vactorindex=vactorindex+1;
            starttag=false;
            }
         }
         else
         {
         if (startwritting==true&&tagentity.trim().equals("body"))
         {
            messagebody=messagebody+(char)i;
         }
         if (starttag)
         {
            tagentity=tagentity+(char)i;
```

```
            }
         }
      }
   }
}
}
catch(IOException ie)
{
}
}
isConnected=false;
try
{
   inputmessagethread.join();
}
catch(Exception e)
{
}
}
/*
checkForError - This method is used to check error messages send by jabber server.
Parameters: tagentity - Object of String class
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if (!isError)
   {
      if (!firstname.equals("false"))
      {
      if (tagentity.indexOf("error")>0)
      {
         isError=true;
         return "User name is already exists. Please enter other User name.";
      }
   }
   if ((tagentity.indexOf("code='401'")>1)||(tagentity.indexOf("code=\"401\"")>1))
   {
      isError=true;
      return "User name or Password is invalid.";
   }
   if (tagentity.indexOf("error")>0)
   {
      if ((tagentity.equals("error code='409'"))||(tagentity.equals("error code=\"409\"")))
      {
         isError=true;
         return "User name is already exists.Please enter other User name.";
      }
   }
}
return error;
}
/*
sendername - This method is used to retrieve sender name from message.
Parameters: tagentity - Object of String class
Return Value: NA
*/
public void sendername(String tagentity)
{
   String sendername="";
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("id=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("id="));
         writemessage("Technical support: ",sendernameattrib);
      }
   }
}
/*
writemessage - This method is used to retrieve sender name from message.
Parameters: messagebody - Object of String class, attribut - Object of MutableAttributeSet.
Return Value: NA
*/
public void writemessage(String messagebody, MutableAttributeSet attribut)
{
   try
   {
      contentmodel.insertString(contentmodel.getLength(), messagebody,attribut);
   }
   catch(BadLocationException ble)
   {
   }
}
/*
openPort - This method is used to open a socket with the Jabber server.
```

```
Parameters: ipaddress - Object of String class
Return Value: NA
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      /*
      Declare and initialize object of SocketOpener class.
      */
      SocketOpener sc=new SocketOpener(ipaddress,portno);
      clientsocket=sc.openSocket(ipaddress,portno,timeinsec);
   }
}
public void itemStateChanged(ItemEvent itemevent)
{
   String eventsource=(String)itemevent.getSource();
}
/*
getNewFont - This method is used to get the font name selected by the end user.
Parameters: ipaddress - Object of String class
Return Value: Font
*/
public Font getNewFont()
{
   int fontsize=8;
   int fontstyle=0;
   String fontname="";
   Font font=null;
   if (sizeChoice.getSelectedIndex()!=-1)
   {
      fontsize=Integer.parseInt((String)sizeChoice.getSelectedItem());
   }
   if (styleChoice.getSelectedIndex()!=-1)
   {
      fontstyle=styleChoice.getSelectedIndex();
   }
   if (fontChoice.getItemCount()>0)
   {
      if (fontChoice.getSelectedItem()!="")
      {
         fontname=(String)fontChoice.getSelectedItem();
      }
      else
      {
         fontname="serif";
      }
   }
   font=new Font(fontname,fontstyle,fontsize);
   return font;
}
public boolean action( Event event, Object what )
{
   if( event.target == fontChoice || event.target == styleChoice ||
    event.target == sizeChoice )
   {
      inputtext.setFont(getNewFont());
      return true;
   }
   if (event.target==sendbutton)
   {
   }
   return false;
}
public void destroy()
{
   try
   {
   clientsocket.close();
   }
   catch(Exception e)
   {
   }
}
/*
readData - This method is used to read next message from socket.
Parameters: NA
Return Value: String
*/
private String readData() throws IOException
{
   StringBuffer s = new StringBuffer();
```

```
   int ch;
   int a=0;
   while ((ch = in.read()) > 0)
   {
      s.append((char)ch);
      a=a+1;
      if (a>10)
      {
      return ch == 0 ? s.toString() : null;
      }
   }
   return ch == 0 ? s.toString() : null;
}
/*
keyTyped - This method is used to handle the key typed event from the text field
Parameters: e - Object of KeyEvent class
Return Value: String
*/
public void keyTyped(KeyEvent e)
{
}
/*
keyPressed - This method is used to handle the key pressed event from the text field.
Parameters: e - Object of KeyEvent class
Return Value: NA
*/
public void keyPressed(KeyEvent e)
{
}
/*
keyReleased - This method is used to handle the key released event from the text field.
Parameters: e - Object of KeyEvent class
Return Value: NA
*/
public void keyReleased(KeyEvent e)
{
   int keyCode = e.getKeyCode();
   if (keyCode==13||keyCode==10)
   {
      sendmessage();
   }
}
};
/*
class SocketOpener - This class opens a socket.
the interface and loads all components like the button, textfields before displaying the result.
Methods:
openSocket - open a socket.
getSocket - Returns a socket objects.
*/
class SocketOpener
{
   private String openerhost;
   private int openerport;
   private Socket socket;
   public SocketOpener(String host,int port)
   {
      socket=null;
      openerhost=host;
      openerport=port;
   }
   /*
   openSocket - This method is used to open a socket.
   Parameters: hostip - Object of String class.portnumber - int,timeinsec - int
   Return Value: Socket
   */
   public  Socket openSocket(String hostip,int portnumber,int timeinsec)
   {
   try
   {
      socket=new Socket(openerhost,openerport);
   }
   catch(IOException ie)
   {
   }
   return getSocket();
}
/*
getSocket - This method is used to get object of Socket class.
Parameters: NA
Return Value: Socket
*/
public Socket getSocket()
{
   return socket;
}
};
```

Download this listing.

In the above code, the constructor of the ChatMainApplet class calls the init() method to create the user interface for the Instant Technical Support application. The init() method calls the startSession() method of the ChatMainApplet class to initialize the chat session with the Jabber server.

The methods defined in Listing 5-5 are:

- init(): Creates the user interface for the Instant Technical Support application.
- `sendmessage()`: Shows the query specified by an end user in the text area provided by the Instant Technical Support application. This method calls the sendXMLToJabber() method.
- `sendXMLToJabber()`: Sends the specified query to the Jabber server.
- `startSession()`: Sends the extensible markup language (XML) message to start the session.
- `checkForError()`: Checks the error messages sent by the Jabber server.
- `sendername()`: Retrieves the sender's name from the message received by the Jabber server.
- `openPort()`: Opens a socket to send and receive messages with the Jabber server.
- `getNewFont()`: Retrieves the font name selected by the end user.
- `readData()`: Reads the response to the specified query from the Jabber server.

The ChatMainApplet.java file creates the user interface of the Instant Technical Support application, as shown in Figure 5-5:

Figure 5-5: User Interface of the Instant Technical Support Application

Unit Testing

To test the Instant Technical Support application:

1. Download and install the free implementation of the Jabber server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the ChatMainApplet.java, welcome.html, signuppage.html, loginpage.html, and chathtml.html files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

5. Create a virtual directory named TechnicalSupport in a Web server.

6. To run the Instant Technical Support application, specify the following uniform resource locator (URL) in the Web browser:

   ```
   http://localhost/TechnicalSupport/welcome.html
   ```

7. The home page of the Instant Technical Support application appears. Click the Sign up button to signup for a new account in the Instant Technical Support application. The Signup page appears.

8. Enter data in the fields of the Signup page, as shown in Figure 5-6:

Figure 5-6: The Signup Information

9. Click OK to create a new account. The user interface of the Instant Technical Support application appears.

10. Specify the query text in the bottom text area provided by the Instant Technical Support application, as shown in Figure 5-7:

Figure 5-7: Specifying the Query Text

11. Click the Send button. The Instant Technical Support application sends the query to the technical support executive.

12. The specified query appears on the user interface in the upper text area, as shown in Figure 5-8:

Figure 5-8: Sending the Query

13. The technical support executive answers the query and sends the response back to the end user, as shown in Figure 5-9:

Figure 5-9: Receiving the Response for the Specified Query

Chapter 6: Creating an Airline Reservation Application

Download CD Content

The Jabber protocol allows you to develop an application to communicate with the Jabber server for accessing an Extensible Markup Language (XML) file that contains data. This chapter describes how to develop the Airline Reservation application that allows end users to search for airline schedules and make airline reservations by specifying the departure and arrival dates of the journey.

Architecture of the Airline Reservation Application

The Airline Reservation application uses the following files:

- UserInformation.html: Allows end users to specify personal information and shows the user interface of the Airline Reservation application.
- Reservation.html: Opens the user interface of the Airline Reservation application by calling the ReservationStatus.java file.
- ReservationStatus.java: Allows end users to select the starting place, destination, departure date, and arrival date of the journey. This file allows end users to submit the information to the Jabber server through the SocketClass.java file.
- SocketClass.java: Opens a socket and establishes a connection with the Jabber server to send and receive messages.
- SocketConnector.java: Reads the airlines.xml file based on the search information specified by end users. The SocketConnector.java file then connects with the Jabber server to send the airline schedules and confirmation message, written in the airlines.xml file, to the Airline Reservation application.
- ReservationResultTable.java: Allows end users to select the required schedule and send schedule information to the Jabber server.
- XMLReader.java: Allows the Airline Reservation application and the Jabber server to read the airlines.xml file.
- airlines.xml: Contains the information related to airline reservation. The Airline Reservation application reads this XML file to show the required information to end users.

Figure 6-1 shows the architecture of the Airline Reservation application:

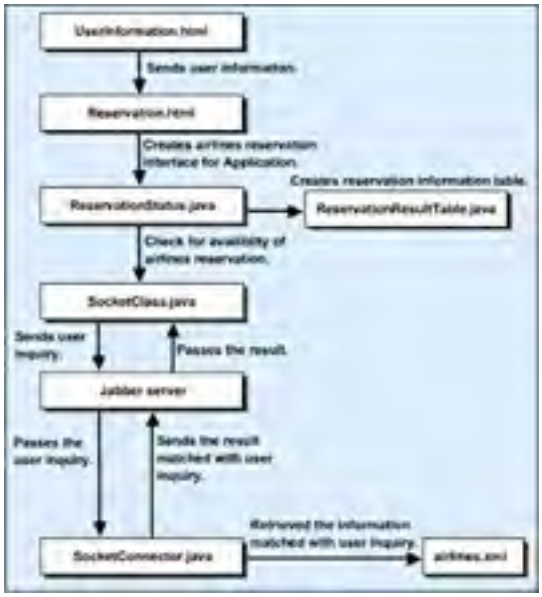


Figure 6-1: Architecture of the Airline Reservation Application

The UserInformation.html file creates the Web interface of Airlines Enquiry System that contains five labels, five text fields, and two command buttons. The UserInformation.html file allows end users to specify personal information.

If end users press the Cancel button on the Web interface, the Airline Enquiry System window closes. If an end user presses the OK button, the end user's information is submitted to the Reservation.html file. The Reservation.html file calls the ReservationStatus.java file.

The ReservationStatus.java file creates the user interface for the Airline Reservation application that contains combo boxes for starting place, destination, departure date, and arrival date. The user interface of the Airline Reservation application also contains two buttons, Reset and Search. When an end user clicks the Reset button, the Airline Reservation application refreshes the values stored in the combo boxes. When an end user clicks the Search button, the Airline Reservation application checks the existing airline schedule in the Jabber server based on the requirements specified by the end user and displays the schedule in the user interface.

The Jabber server passes an end user's query to the SocketConnector.java file. The SocketConnector.java file reads the airlines.xml file to search for the query specified by the end user. The airlines.xml file stores the database of existing airline schedules. If an end user's query matches any existing airline schedule, the SocketConnector.java file returns the information to the Jabber server. The Jabber server sends the information to the Airline Reservation application. The application shows the information received from the Jabber server in the Airline Reservation window by using the ReservationResultTable.java file.

End users can select any of the available airline schedules to reserve the required airline schedule. End users can click the Submit button to send the information to the Jabber server. The Jabber server receives the information and sends a confirmation message to the Airline Reservation application, which appears on the Airline Reservation window.

◄ PREVIOUS NEXT ►

Creating the Personal Information Page

The UserInformation.html file creates the user interface to allow end users to enter personal information. Listing 6-1 shows the contents of the UserInformation.html file:

Listing 6-1: The UserInformation.html File

```
<HTML>
<HEAD>
<TITLE>New User </TITLE>
<script language="javascript">
function openwindow(pagename,windowname)
{
   if(document.profileenteryform.firstname.value=="")
   {
      alert('First name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.firstname.value))
      {
         alert('Please enter a valid value for first name.');
         return false;
      }
   }
   if(document.profileenteryform.lastname.value=="")
   {
      alert('Last Name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.lastname.value))
      {
         alert('Please enter a valid value for last name.');
         return false;
      }
   }
   if(document.profileenteryform.Address.value=="")
   {
      alert('Address is a required field.');
      return false;
   }
   if(document.profileenteryform.City.value=="")
   {
      alert('City is a required field.');
      return false;
   }
   if(document.profileenteryform.emailid.value=="")
   {
      alert('Email Id is a required field.');
      return false;
   }
   else
   {
   if(!isEmailAddr(document.profileenteryform.emailid.value))
   {
      alert('Please enter a valid value for email ID.');
      return false;
   }
}
function checkForInt(valuestring)
{
   var validcharstring=false;
   var intstring="0123456789!@#$%^&*()_-+="
   for(i=0;i<valuestring.length;i++)
   {
      if (intstring.indexOf(valuestring.charAt(i))!=-1)
      {
         return validcharstring;
      }
   }
   validcharstring=true;
   return validcharstring;
}
function isEmailAddr(email)
{
   var result = false;
   var theStr = new String(email);
   var index = theStr.indexOf("@");
   if (index > 0)
   {
      var pindex = theStr.indexOf(".",index);
```

```
      if ((pindex > index+1) && (theStr.length > pindex+1))
      result = true;
   }
   return result;
}
city=document.profileenteryform.City.value;
address=document.profileenteryform.Address.value;
fname=document.profileenteryform.firstname.value;
lname=document.profileenteryform.lastname.value;
emailid=document.profileenteryform.emailid.value;
pagename=pagename+"?city="+city+"&firstname="+fname+"&lastname="+lname+"&emailid="+emailid
+"&address"+address;
window.open(pagename, windowname, 'width=720, height=622, left=200, top=60,screenX=30, screenY=100')
}
</script>
</HEAD>
<BODY>
<FORM NAME="profileenteryform">
<TABLE WIDTH="100%">
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
<FONT SIZE="4" COLOR="">Airlines Enquiry System</FONT>
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
<FONT SIZE="4" COLOR="">Enter your personal information:</FONT>
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD>
<TABLE align="center" WIDTH="100%" border=0 cellspacing=0 cellpadding=0>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
First Name
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="firstname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Last Name
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="lastname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Address
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="Address" VALUE="" maxlength=50>
```

```
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="Address" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
City
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="City" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Email ID
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" width="100%" size="40%" NAME="emailid" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
 
</TD>
<TD WIDTH="50%" colspan=2>
<INPUT TYPE="BUTTON" NAME="ok" style="width:20%" VALUE="OK"
onclick="javascript:openwindow('Reservation.html', 'reservation');">
<INPUT TYPE="BUTTON" NAME="cancel" style="width:20%" VALUE="Cancel"
onclick="javascript:window.close();">
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Download this listing.

The UserInformation.html file creates the Personal Information page, as shown in Figure 6-2:

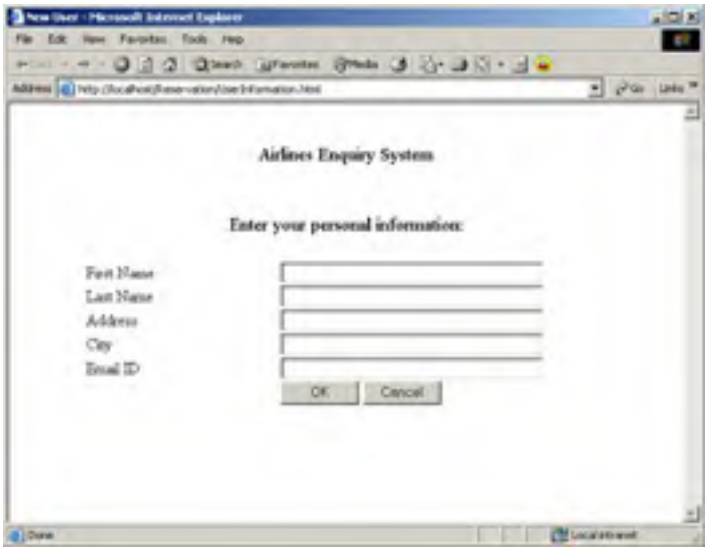


Figure 6-2: The Personal Information Page

◄ PREVIOUS NEXT ►

Sending Personal Information to the Reservation.html File

The Reservation.html file accepts an end user's personal information and calls the ReservationStatus.java file to create the user interface for the Airline Reservation application. Listing 6-2 shows the contents of the Reservation.html file:

Listing 6-2: The Reservation.html File

```
<HTML>
<HEAD>
<TITLE> Airline Reservation </TITLE>
<script>
function PageQuery(q)
{
   if(q.length > 1) this.q = q.substring(1, q.length);
   else this.q = null;
   this.keyValuePairs = new Array();
   if(q)
   {
      for(var i=0; i < this.q.split("&").length; i++)
      {
         this.keyValuePairs[i] = this.q.split("&")[i];
      }
   }
   this.getKeyValuePairs = function() { return this.keyValuePairs; }
   this.getValue = function(s) {
   for(var j=0; j < this.keyValuePairs.length; j++) {
   if(this.keyValuePairs[j].split("=")[0] == s)
   return this.keyValuePairs[j].split("=")[1];
   }
   return false;
}
this.getParameters = function() {
   var a = new Array(this.getLength());
   for(var j=0; j < this.keyValuePairs.length; j++) {
   a[j] = this.keyValuePairs[j].split("=")[0];
   }
   return a;
}
this.getLength = function() { return this.keyValuePairs.length; }
}
function queryString(key){
   var page = new PageQuery(window.location.search);
   return page.getValue(key);
   }
   function appletparam(key){
   if(queryString(key)=='false')
   {
      pagename='UserInformation.html';
      windowname='login';
      window.open(pagename,windowname,'width=400,height=200,left=300,top=100,screenX=500,screenY=100
   }
   else
   {
   return queryString(key);
   }
}
</script>
<SCRIPT LANGUAGE="JavaScript">
function writeAppletTag()
{
   document.writeln('<applet code="ReservationStatus" width="700" height="600">');
   document.writeln(buildParamTag('city',appletparam('city')));
   document.writeln(buildParamTag('firstname',appletparam('firstname')));
   document.writeln(buildParamTag('lastname',appletparam('lastname')));
   document.writeln(buildParamTag('emailid',appletparam('emailid')));
   document.writeln(buildParamTag('address',appletparam('address')));
   document.writeln('</APPLET>');
}
function buildParamTag(name, value) {
   return '<PARAM NAME="' + name + '" VALUE="' + value + '">';
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
writeAppletTag();
</SCRIPT>
</applet>
</BODY>
</HTML>
```

Download this listing.

In the above code, the Reservation.html file calls the ReservationStatus.java file to create the user interface for the Airline Reservation application. The ReservationStatus.java file passes an end user's personal information to the ReservationStatus.java file.

PREVIOUS NEXT

Creating the User Interface for the Airline Reservation Application

The ReservationStatus.java file creates the user interface for the Airline Reservation application. Listing 6-3 shows the contents of the ReservationStatus.java file:

Listing 6-3: The ReservationStatus.java File

```
/*Imports required swing classes*/
import javax.swing.*;
/*Imports required Applet class*/
import java.applet.Applet;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required i/o classes*/
import java.io.*;
/*Imports required net classes*/
import java.net.*;
/*Imports required util classes*/
import java.util.*;
/*Imports required swing classes*/
import javax.swing.table.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
/*Imports required Vector class*/
import java.util.Vector;
import javax.swing.table.AbstractTableModel;
import javax.swing.event.*;
/*
class ReservationStatus - This class is the main class of the application. This class initializes th
interface and loads all components like the button and textfields before displaying the result.
Constructor:
ReservationStatus-This constructor creates GUI.
Methods:
setTableContent - Sets search result into table.
validation - This method is used to validate user input.
*/
public class ReservationStatus extends JApplet  implements ActionListener
{
   /*Declare objects of JComboBox class.*/
   JComboBox fromcombobox;
   JComboBox tocombobox;
   JComboBox todaycombobox;
   JComboBox tomonthcombobox;
   JComboBox toyearcombobox;
   JComboBox fromdaycombobox;
   JComboBox frommonthcombobox;
   JComboBox fromyearcombobox;
   /*Declare objects of JLabel class.*/
   JLabel fromlabel;
   JLabel tolabel;
   JLabel fromdatelabel;
   JLabel todatelabel;
   /*Declare object of JButton class.*/
   JButton searchbutton;
   /*Declare object of Container class.*/
   Container contentpane;
   /*Declare object of String class.*/
   String TodayDate;
   /*Declare array of String class.*/
   String[] spleteddate;
   /*Declare objects of JRadioButton class.*/
   JRadioButton iradio;
   JRadioButton nriradio;
   /*Declare object of Vector class.*/
   Vector idvector;
   /*Declare object of String class.*/
   String servername="localhost";
   /*Declare object of SocketClass class.*/
   SocketClass socketclass;
   /*Declare objects of String class.*/
   String passportno="100001";
   String city;
   String username;
   String emailid;
   String address;
   /*Declare object of JButton class.*/
   JButton submitbutton;
   /*Declare object of JCheckBox class.*/
   JCheckBox roundtrip;
   /*Declare objects of Vector class.*/
   Vector evector;
   Vector gvector;
   /*Declare integer variable*/
```

```
int selectedrow;
/*Declare object of MyTableReservation class.*/
TableModelReservation mytable;
/*Declare object of ReservationResultTable class.*/
ReservationResultTable t;
/*
This method initializes the object variables in the applet.
*/
public void init()
{
contentpane=getContentPane();
/*Set the BorderLayout to the applet.*/
contentpane.setLayout(new BorderLayout());
/*
Initialize and set the Look and Feel of the application to Windows Look and Feel.
*/
try
{
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
   System.out.println("Unable to set WLF"+e);
}
/* Initialize the objects of Vector class. */
evector=new Vector();
gvector=new Vector();
city=getParameter("city");
username=getParameter("firstname")+getParameter("lastname");
emailid=getParameter("emailid");
address=getParameter("address");
/*
Declare and initialize the object of JPanel class.
*/
JPanel mainpanel=new JPanel();
/*
Declare and initialize the object of GridLayout class.
*/
GridLayout mainlayout=new GridLayout(2, 2, 5, 5);
/*Set the Gridlayout to mainpanel object.*/
mainpanel.setLayout(mainlayout);
/*
Declare and initialize the object of JPanel class.
*/
JPanel citypanel=new JPanel();
/*
Declare and initialize the object of GridLayout class.
*/
GridLayout citylayout=new GridLayout(4, 1, 15, 5);
/*Set the Gridlayout to citypanel object.*/
citypanel.setLayout(citylayout);
/*Initializes the object of JComboBox class.*/
fromcombobox=new JComboBox();
/*Add the items to the object of JComboBox class.*/
fromcombobox.addItem("Chicago");
fromcombobox.addItem("Boston");
fromcombobox.addItem("Denver");
/*Initializes the object of JLabel class.*/
fromlabel=new JLabel("From",JLabel.CENTER);
/*Sets the font to the object of JLabel class.*/
fromlabel.setFont(new Font("verdana", 1, 12));
/*
Adds the object of the JLabel class to the object JPanel class.
*/
citypanel.add(fromlabel);
/*
Declares and initializes the object of JPanel class.
*/
JPanel fromcombopanel=new JPanel();
/* Sets the size of the object of JComboBox class.*/
fromcombobox.setPreferredSize(new Dimension(200, 20));
/*
Adds the object of JComboBox class to object of JPanel class.
*/
fromcombopanel.add(fromcombobox);
/*
Adds the objects of JPanel class to object of JPanel class.
*/
citypanel.add(fromcombopanel);
mainpanel.add(citypanel);
/*
Declares and initializes the object of JPanel class.
*/
JPanel citypanelto=new JPanel(new GridLayout(4,1,15,5));
/*Initializes the object of JLabel class*/
tolabel=new JLabel("To",JLabel.CENTER);
/*Sets the font of the object of JLabel class.*/
```

```
    tolabel.setFont(new Font("verdana",1,12));
    /* Adds the object of JLabel class to the object of JPanel class.*/
    citypanelto.add(tolabel);
    /*Initializes the object of JComboBox class.*/
    tocombobox=new JComboBox();
    /*Add items to the object of JComboBox class.*/
    tocombobox.addItem("Chicago");
    tocombobox.addItem("Boston");
    tocombobox.addItem("Denver");
    /*
    Sets the preferred size of the object of JComboBox class.
    */
    tocombobox.setPreferredSize(new Dimension(200, 20));
    /*
    Declares and initializes the object of JPanel class.
    */
    JPanel tocombopanel=new JPanel();
    /*
    Adds the object of JCombobox class to the object of JPanel class.
    */
    tocombopanel.add(tocombobox);
    /*
    Adds the object of JPanel class to the object of JPanel class.
    */
    citypanelto.add(tocombopanel);
    /*Initializes the object of the JCheckBox class.*/
    roundtrip=new JCheckBox(" Round trip");
    /*Sets the font of the object of JCheckBox class.*/
    roundtrip.setFont(new Font("verdana",1,12));
    /*
    Adds the ItemListener to the object of JCheckBox class.
    */
    roundtrip.addItemListener(new ItemListener()
    {
       public void itemStateChanged(ItemEvent e)
       {
       if (e.getStateChange() == ItemEvent.SELECTED)
       {
          todaycombobox.setEnabled(true);
          tomonthcombobox.setEnabled(true);
          toyearcombobox.setEnabled(true);
       }
       else
       {
          todaycombobox.setEnabled(false);
          tomonthcombobox.setEnabled(false);
          toyearcombobox.setEnabled(false);
       }
    }
});
/*
Declares and Initializes the object of the JPanel class.
*/
JPanel roundtrippanel=new JPanel();
/*
Adds the object of the JCheckBox class to the object of JPanel class.
*/
roundtrippanel.add(roundtrip);
/*
Adds the object of JPanel class to the object of the JPanel class.
*/
citypanelto.add(roundtrippanel);
/*
Initialize the objects of JComboBox class.
*/
todaycombobox=new JComboBox();
tomonthcombobox=new JComboBox();
toyearcombobox=new JComboBox();
/*
Declares and Initializes the object of JPanel class.
*/
JPanel todaypanel=new JPanel();
todaycombobox.setEnabled(false);
tomonthcombobox.setEnabled(false);
toyearcombobox.setEnabled(false);
/*Sets the size of the object of JComboBox class.*/
todaycombobox.setPreferredSize(new Dimension(40, 20));
/*Calls the addDays() function.*/
addDays(todaycombobox);
/*Sets the size of the object of JComboBox class.*/
tomonthcombobox.setPreferredSize(new Dimension(70, 20));
/*Calls the addMonths() function.*/
addMonths(tomonthcombobox);
/*Sets the size of the object of JComboBox class.*/
toyearcombobox.setPreferredSize(new Dimension(50, 20));
```

```
/*Calls the addYears() function.*/
addYears(toyearcombobox);
/*
Adds the objects of JComboBox class to the object of JPanel class.
*/
todaypanel.add(todaycombobox);
todaypanel.add(tomonthcombobox);
todaypanel.add(toyearcombobox);
/*
Adds the object of JPanel class to the object of JPanel class.
*/
citypanelto.add(todaypanel);
/*
Adds the object of JPanel class to the object of JPanel class.
*/
mainpanel.add(citypanelto);
/*Initializes the object of the JLabel class.*/
fromdatelabel=new JLabel("Departure Date", JLabel.CENTER);
fromdatelabel.setFont(new Font("verdana", 1, 12));
/*
Adds the object of JLabel class to the object of JPanel class.
*/
/* Initializes the objects of JComboBox class.*/
citypanel.add(fromdatelabel);
fromdaycombobox=new JComboBox();
frommonthcombobox=new JComboBox();
fromyearcombobox=new JComboBox();
/*
Declares and Initializes the object of JPanel class.
*/
JPanel fromdaypanel=new JPanel();
/*Sets the size of the object of JComboBox class.*/
fromdaycombobox.setPreferredSize(new Dimension(40, 20));
/*Calls the addDays() function.*/
addDays(fromdaycombobox);
/*Sets the size of the object of JComboBox class.*/
frommonthcombobox.setPreferredSize(new Dimension(70, 20));
/*Calls the addMonths() function.*/
addMonths(frommonthcombobox);
/*Sets the size of the object of JComboBox class.*/
fromyearcombobox.setPreferredSize(new Dimension(50, 20));
/*Calls the addMonths() function.*/
addYears(fromyearcombobox);
/*
Adds the objects of JComboBox class to the object of JPanel class.
*/
fromdaypanel.add(fromdaycombobox);
fromdaypanel.add(frommonthcombobox);
fromdaypanel.add(fromyearcombobox);
citypanel.add(fromdaypanel);
/*
Declares and Initializes the object of JPanel class.
*/
JPanel nationalitypanel=new JPanel();
/*
Declares and Initializes the object of ButtonGroup class.
*/
ButtonGroup nationalitygroup = new ButtonGroup();
/*Initializes the objects of JRadioButton class.*/
iradio=new JRadioButton("US Citizen");
iradio.setSelected(true);
/*Initializes the objects of JRadioButton class.*/
nriradio=new JRadioButton("Foreigner");
/*
Declares and Initializes the object of JPanel class.
*/
JPanel subnationalitypanel=new JPanel(new GridLayout(3, 1, 0, 0));
/*
Declares and Initializes the object of JLabel class.
*/
JLabel nationalitylabel=new JLabel("Nationality");
/*Sets the font of the object of the JLabel class.*/
nationalitylabel.setFont(new Font("verdana", 1, 12));
/*
Adds the object of the JLabel class to the object of the JPanel class.
*/
subnationalitypanel.add(nationalitylabel);
/*
Adds the objects of the JRadioButton class to the object of ButtonGroup class.
*/
nationalitygroup.add(iradio);
nationalitygroup.add(nriradio);
/*
Adds the objects of the JRadioButton class to the object of JPanel class.
```

```
*/
subnationalitypanel.add(iradio);
subnationalitypanel.add(nriradio);
/*
Adds the objects of the JPanel class to the object of JPanel class.
*/
nationalitypanel.add(subnationalitypanel);
mainpanel.add(nationalitypanel);
/*
Adds the object of the JPanel class to the object of ContentPane object.
*/
contentpane.add(mainpanel);
/*Initializes the object of JButton class.*/
searchbutton=new JButton("Search");
/*
Sets the ActionCommand of the object of JButton class.
*/
searchbutton.setActionCommand("search");
searchbutton.setMnemonic('s');
/*
Adds the ActionListener to the object of the JButton class.
*/
searchbutton.addActionListener(this);
/*
Declares and Initializes the object of JPanel class.
*/
JPanel buttonpanel=new JPanel();
/*
Sets the Flowlayout of the object of JPanel class.
*/
buttonpanel.setLayout(new FlowLayout());
/*
Declares and Initializes the object of JButton class.
*/
JButton resetbutton=new JButton("Reset");
/*
Sets the ActionCommand of the object of JButton class.
*/
resetbutton.setActionCommand("reset");
resetbutton.setMnemonic('r');
/*
Adds the ActionListener to the object of the JButton class.
*/
resetbutton.addActionListener(this);
/*
Sets the size of the object of the JButton class
*/
resetbutton.setPreferredSize(new Dimension(80,20));
/*
Initializes the object of ReservationResultTable class.
*/
t=new ReservationResultTable();
/*
Sets the size of the object of the ReservationResultTable class.
*/
t.setPreferredSize(new Dimension(300, 100));
/*
Declares and Initializes the object of JPanel class.
*/
JPanel bpanel=new JPanel(new BorderLayout());
/*
Declares and Initializes the object of JPanel class.
*/
JPanel topsubmitbuttonpanel=new JPanel();
/*
Adds the objects of JButton class to the object of the JPanel class.
*/
topsubmitbuttonpanel.add(searchbutton);
topsubmitbuttonpanel.add(resetbutton);
bpanel.add(topsubmitbuttonpanel,BorderLayout.NORTH);
/*
Sets the size of the object of the JButton class.
*/
searchbutton.setPreferredSize(new Dimension(80, 20));
/*
Adds the object of ReservationResultTable class to the object of the JPanel class.
*/
bpanel.add(t,BorderLayout.CENTER);
/* Initializes the object of JButton class.*/
submitbutton=new JButton("Submit");
/*
Sets the ActionCommand of the object of JButton class.
```

```
*/
submitbutton.setActionCommand("submit");
submitbutton.setMnemonic('u');
/*
Adds the ActionListener to the object of the JButton class.
*/
submitbutton.addActionListener(this);
submitbutton.setEnabled(false);
/*
Adds the object of JButton class to the object of the JPanel class.
*/
bpanel.add(submitbutton,BorderLayout.SOUTH);
/*
Adds the object of JPanel class to the object of the ContentPane class.
*/
contentpane.add(bpanel,BorderLayout.SOUTH);
setSize(650,340);
/* Initializes the object of SocketClass class.*/
socketclass=new SocketClass("localhost", 5222, 1000,this);
}
public String getUserName()
{
   return username;
}
/*
setTableContent - Inserts the search result data into the table.
Parameters:messagebody - object of String class.
Return Value: NA
*/
public void setTableContent(String messagebody)
{
if (messagebody.trim().equals("No matching record found"))
{
   JOptionPane.showMessageDialog(null,messagebody,"Dialog box",JOptionPane.PLAIN_MESSAGE);
   submitbutton.setEnabled(false);
   return;
}
if(messagebody.indexOf("@")>0)
{
   String[] spletedschedule=messagebody.split("@");
   submitbutton.setEnabled(true);
   idvector=new Vector();
   Object[][] object=new Object[spletedschedule.length][];
   for (int i=0;i<spletedschedule.length;i++ )
   {
      String[] sp=spletedschedule[i].split("#");
      object[i]=new Object[sp.length-1];
      for (int j=0;j<sp.length-3;j++ )
      {
         String[] splitedvalue=sp[j].split("=");
         if (splitedvalue.length==2)
         {
            if (splitedvalue[1].equals("YES"))
            {
               object[i][j]=new Boolean(false);
            }
            else
            {
               if (splitedvalue[1].equals("NO"))
               {
                  object[i][j]="NA";
               }
               else
               {
                  object[i][j]=splitedvalue[1];
               }
            }
         }
         idvector.add(sp[sp.length-3]);
         evector.add(sp[sp.length-2]);
         gvector.add(sp[sp.length-1]);
      }
   }
   mytable= new TableModelReservation();
   mytable.setData(object);
   t.table.setModel(mytable);
}
}
/*
class TableModelReservation - This class is the main class of the application. This class initialize
the interface and loads all components like the button, textfields before displaying the result.
Methods:
setData - Sets Object array to table object.
getRowCount - This method returns the total number of rows.
getColumnName - This method returns the Column name.
getValueAt - This method returns the value at a particular cell location.
setValueAt - This method sets the value at a particular cell location.
```

```
*/
class TableModelReservation extends AbstractTableModel
{
   private String[] columnNames = {"Choice","Flight No.",
   "Aircraft type",
   "Origin",
   "No. of Stops",
   "Destination",
   "Dep.Date-time",
   "Arr.Date-time",
   "Executive Class",
   "Economy Class",
};
private Object[][] data = {};
public int getColumnCount()
{
   return columnNames.length;
}
public void setData(Object[][] dataarr)
{
   data=dataarr;
}
public int getRowCount()
{
   return data.length;
}
public String getColumnName(int col)
{
   return columnNames[col];
}
public Object getValueAt(int row, int col)
{
   return data[row][col];
}
public Class getColumnClass(int c)
{
   return getValueAt(0, c).getClass();
}
public boolean isCellEditable(int row, int col)
{
   if (col >=1&&col<=7)
   {
      return false;
   }
   else
   {
      return true;
   }
}
public void setValueAt(Object value, int row, int col)
{
   data[row][col] = value;
   fireTableCellUpdated(row, col);
   if (col==0&&value.toString().equals("true"))
   {
      int totalrows=getRowCount();
      for (int i=0;i<totalrows;i++ )
      {
         selectedrow=row;
         if (row!=i)
         {
            data[i][col] =new Boolean(false);
         }
      }
   }
   if ((col==9)&&value.toString().equals("true"))
   {
      data[row][col-1] = new Boolean(false);
      fireTableCellUpdated(row, col-1);
   }
   else
   {
      if (col==8&&value.toString().equals("true"))
      {
         data[row][col+1] =new Boolean(false);
         fireTableCellUpdated(row, col+1);
      }
   }
}
}
public void actionPerformed(ActionEvent ae)
{
   String journeytypestring="Oneway";
   String nationalitystring="US Citizen";
   String actioncommand=ae.getActionCommand();
   if (actioncommand.equals("search"))
```

```
    {
    String informationstring;
    String seachstring;
    if (validation())
    {
    if (iradio.isSelected())
    {
       nationalitystring=iradio.getText();
    }
    if (nriradio.isSelected())
    {
    nationalitystring=nriradio.getText();;
    }
    if (roundtrip.isSelected())
    {
       journeytypestring="roundtrip";
    }
    String dmonthpart="";
    String rmonthpart="";
    String rdaypart="";
    String ddaypart="";
    int dday=fromdaycombobox.getSelectedIndex()+1;
    int dmonth=frommonthcombobox.getSelectedIndex()+1;
    if (dday<10)
    {
       ddaypart="0";
    }
    if (dmonth<10)
    {
       dmonthpart="0";
    }
    int rmonth=tomonthcombobox.getSelectedIndex()+1;
    if (rmonth<10)
    {
       rmonthpart="0";
    }
    int rday=todaycombobox.getSelectedIndex()+1;
    if (rday<10)
    {
       rdaypart="0";
    }
    seachstring="#origin="+((String)fromcombobox.getSelectedItem());
    seachstring=seachstring+"#destination="+((String)tocombobox.getSelectedItem());
    seachstring=seachstring+"#ddate="+((String)fromyearcombobox.getSelectedItem())+dmonthpart+
    dmonth+dday
    part+dday;  seachstring=seachstring+"#rdate="+((String)toyearcombobox.getSelectedItem())+
    rmonthpart+rmonth+rdaypart+rday;
    seachstring=seachstring+"#nationality="+nationalitystring;
    seachstring=seachstring+"#typeofjurney="+journeytypestring;
    informationstring="<message to=\"user10@"+servername+"/airlines\" type=\"chat\">";
    informationstring=informationstring+"<body>";
    informationstring=informationstring+seachstring;
    informationstring=informationstring+"</body></message>";
    socketclass.sendXMLToJabber(informationstring);
}
}
if (actioncommand.equals("reset"))
{
    fromcombobox.setSelectedIndex(0);
    tocombobox.setSelectedIndex(0);
    todaycombobox.setSelectedIndex(0);
    tomonthcombobox.setSelectedIndex(0);
    toyearcombobox.setSelectedIndex(0);
    fromdaycombobox.setSelectedIndex(0);
    frommonthcombobox.setSelectedIndex(0);
    Object[][] emptyobject = {};
    TableModelReservation emptytable=new TableModelReservation();
    emptytable.setData(emptyobject);
    t.table.setModel(emptytable);
    submitbutton.setEnabled(false);
}
if (actioncommand.equals("submit"))
{
    String idstring=(String)idvector.get(selectedrow);
    String submitstring="";
    submitstring=submitstring+"<message to=\"user10@"+servername+"/airlines\"
    type=\"chat\"><body>";
    submitstring=submitstring+"#id"+idstring+"#username"+username+"#address="+address+"#city="+city+
    "#passportno"+passportno;
    submitstring=submitstring+"</body></message>";
    socketclass.sendXMLToJabber(submitstring);
    Object[][] emptyobject = {};
    TableModelReservation emptytable=new TableModelReservation();
    emptytable.setData(emptyobject);
    t.table.setModel(emptytable);
    JOptionPane.showMessageDialog(null,"Data has been submitted, for further details contact airlines
```

```
   administrator","Dialog box",JOptionPane.PLAIN_MESSAGE);
   submitbutton.setEnabled(false);
}
}
/*
validation - This method is used to validate user input.
Parameters:NA
Return Value: boolean
*/
public boolean validation()
{
   boolean valid=true;
   String validationmessage="";
   if ((String)fromcombobox.getSelectedItem()==(String)tocombobox.getSelectedItem())
   {
      validationmessage="Destination and origin should not be same.";
      valid=false;
   }
   if (Integer.parseInt(spleteddate[1])>(frommonthcombobox.getSelectedIndex()+1))
   {
      validationmessage=validationmessage+"\n"+"Departure date can not be less than today's date.";
      valid=false;
   }
   else
   {
      if (Integer.parseInt(spleteddate[1])==(frommonthcombobox.getSelectedIndex()+1))
      {
         String[] splitday=spleteddate[2].split(" ");
         if (Integer.parseInt(splitday[0])>(fromdaycombobox.getSelectedIndex()+1))
         {
            validationmessage=validationmessage+"\n"+"Departure date can not be less than today's
            date.";
            valid=false;
         }
      }
   }
   if (roundtrip.isSelected())
   {
      if (Integer.parseInt(spleteddate[1])>(tomonthcombobox.getSelectedIndex()+1))
      {
      validationmessage=validationmessage+"\n"+"Return date can not be less than today's date.";
      valid=false;
      }
      else
      {
         if (Integer.parseInt(spleteddate[1])==(tomonthcombobox.getSelectedIndex()+1))
         {
         String[] splitday=spleteddate[2].split(" ");
         if (Integer.parseInt(splitday[0])>(todaycombobox.getSelectedIndex()+1))
         {
            validationmessage=validationmessage+"\n"+"Return date can not be less than today's date.
            valid=false;
         }
         }
      }
   }
   if (roundtrip.isSelected())
   {
      if (Integer.parseInt((String)toyearcombobox.getSelectedItem())<Integer.parseInt((String)
      fromyearcombobox.getSelectedItem()))
      {
         validationmessage=validationmessage+"\n"+"Return date can not be less than Departure date."
         valid=false;
      }
      else
      {
         if ((frommonthcombobox.getSelectedIndex())>(tomonthcombobox.getSelectedIndex()))
         {
            validationmessage=validationmessage+"\n"+"Return date can not be less than Departur date
            valid=false;
         }
         else
         {
            if ((frommonthcombobox.getSelectedIndex())==(tomonthcombobox.getSelectedIndex()))
            {
               if ((fromdaycombobox.getSelectedIndex())>(todaycombobox.getSelectedIndex()))
               {
                  validationmessage=validationmessage+"\n"+"Return date can not be less than Departu
                  date.";
                  valid=false;
               }
            }
         }
      }
   }
   if (!validationmessage.equals(""))
   {
```

```
   JOptionPane.showMessageDialog(null,validationmessage,"Dialog box",JOptionPane.PLAIN_MESSAGE);
   }
   return valid;
}
/*
addDays - This method is used to fill days into fromdaycombobox.
Parameters:fromdaycombobox - Object of JComboBox class.
Return Value: NA
*/
public void addDays(JComboBox fromdaycombobox)
{
   for (int i=1;i<32 ;i++ )
   {
      fromdaycombobox.addItem(new Integer(i));
   }
}
/*
addMonths - This method is used to fill months into frommonthcombobox Combo.
Parameters:frommonthcombobox - Object of JComboBox class
Return Value: NA
*/
public void addMonths(JComboBox frommonthcombobox)
{
   frommonthcombobox.addItem("January");
   frommonthcombobox.addItem("February");
   frommonthcombobox.addItem("March");
   frommonthcombobox.addItem("April");
   frommonthcombobox.addItem("May");
   frommonthcombobox.addItem("June");
   frommonthcombobox.addItem("July");
   frommonthcombobox.addItem("August");
   frommonthcombobox.addItem("September");
   frommonthcombobox.addItem("October");
   frommonthcombobox.addItem("November");
   frommonthcombobox.addItem("December");
}
/*
addYears - This method is used to fill years into fromyearcombobox Combo.
Parameters:frommonthcombobox - Object of JComboBox class
Return Value: NA
*/
public void addYears(JComboBox fromyearcombobox)
{
   Calendar cal = Calendar.getInstance(TimeZone.getDefault());
   String DATE_FORMAT = "yyyy-MM-dd HH:mm:ss";
   java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat(DATE_FORMAT);
   sdf.setTimeZone(TimeZone.getDefault());
   TodayDate=sdf.format(cal.getTime());
   System.out.println(TodayDate);
   spleteddate=TodayDate.split("-");
   fromyearcombobox.addItem(spleteddate[0]);
   fromyearcombobox.addItem(new Integer(Integer.parseInt(spleteddate[0])+1));
}
}
```

Download this listing.

In the above listing, the ReservationStatus.java file creates the user interface by using the ReservationStatus class. The ReservationStatus.java file uses the TableModelReservation class to display the airline schedule in the user interface provided by the Airline Reservation application.

The methods defined in Listing 6-3 are:

- `init()`: Creates the user interface for the Airline Reservation application and calls the SocketClass class to establish a connection with the Jabber server.
- `getUserName()`: Returns the name of the end user.
- `setTableContent()`: Initializes the TableModelReservation class and displays the airline schedule in the user interface provided by the Airline Reservation application.
- `itemStateChanged()`: Activates the combo boxes for day, month, and year, if an end user selects the Round trip option.
- `addDays()`: Enters number of days in the combo box for the departure date and arrival date.
- `addMonths()`: Enters number of months in the combo box for the departure date and arrival date.
- `addYears()`: Enters number of years in the combo box for the departure date and arrival date.
- `setValueAt()`: Shows the airline schedule received from the Jabber server.
- `actionPerformed()`: Acts as an event listener and activates an appropriate class and method based on the action an end user performs.
- `getRowCount()`: Returns the total number of records in an airline schedule.

- `validation()`: Validates the information specified by the end user.

The ReservationStatus.java file creates the main window of the Airline Reservation application, as shown in Figure 6-3:

Figure 6-3: User Interface of the Airline Reservation Application

PREVIOUS NEXT

Creating the Socket Class to Send and Receive Messages

The SocketClass.java file opens the socket with the Jabber server to send and receive messages between end users. Listing 6-4 shows the contents of the SocketClass.java file:

Listing 6-4: The SocketClass.java File

```
/*Imports required java.util classes*/
import java.util.*;
/*Imports required java.io classes*/
import java.io.*;
/*Imports required java.net classes*/
import java.net.*;
/*Imports required javax.swing classes*/
import javax.swing.*;
/*
class SocketClass - This class is used for reading the input messages and writing output messages.
Constructor:
SocketClass - This constructor calls a method of SocketOpener class to open a client socket.
Methods:
*/
class SocketClass implements Runnable
{
   public boolean isConnected;
   /*Declare the object of the Vector class.*/
   public Vector tagvactor;
   /*
   Declare the objects of the PrintWriter class.
   */
   PrintWriter out = null;
   /*Declare the objects of the BufferedReader class.*/
   BufferedReader in = null;
   /*Declare the objects of the String class.*/
   String ipaddress="";
   String errortype;
   String resource="Airline";
   String username;
   String password;
   /*Declare the objects of the Socket class.*/
   Socket clientsocket;
   /*Declare the objects of the Thread class.*/
   Thread inputmessagethread;
   /*
   Declare the objects of the ReservationStatus class.
   */
   ReservationStatus reservation;
   int portno=5222;
   int wait=1000;
   public SocketClass(String ipaddress, int portno, int wait, ReservationStatus reservation)
   {
      this.ipaddress=ipaddress;
      this.portno=portno;
      this.wait=wait;
      this.reservation=reservation;
      /*Call openPort() method.*/
      openPort(ipaddress,portno,wait);
      if (clientsocket!=null)
      {
      isConnected=true;
      }
      try
      {
      /*
      Initializes the object of PrintWriter class.
      */
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      /*
      Initializes the object of PrintWriter class.
      */
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
      }
      catch(Exception e)
      {
      e.printStackTrace();
      }
      /*
      Declare the objects of the String class.
      */
      String sessionStratString;
      String registrationstring;
      String authentication;
      username=reservation.getUserName();
      sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
```

```
   sessionStratString=sessionStratString+" <stream:stream";
   sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
   sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   sendXMLToJabber(sessionStratString);
   registrationstring="<iq type=\"set\" id=\"1301\">";
   registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
   registrationstring=registrationstring+"<username>"+username+"</username>";
   registrationstring=registrationstring+"<password>"+username+"</password>";
   registrationstring=registrationstring+"</query></iq>";
   sendXMLToJabber(registrationstring);
   authentication="<iq type=\"set\" id=\"1301\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+" <username>"+username+"</username>";
   authentication=authentication+" <password>"+username+"</password>";
   authentication=authentication+" <resource>"+resource+"</resource> ";
   authentication=authentication+" </query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
   /*
   Initializes the object of Thread class.
   */
   inputmessagethread=new Thread(this);
   /*
   Calls start() method of the object of Thread class.
   */
   inputmessagethread.start();
}
/*
openPort: This method creates an object of SocketOpener class and class openSocket method of this
class.
Parameter: ipaddress - Object of String class,portno - int, timeinsec - int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   /*
   Declare the objects of the String class.
   */
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      /*
      Declare and initializes the objects of the SocketOpener class.
      */
      SocketOpener socketopener=new SocketOpener();
      /*
      Calls openSocket Method() of SocketOpener
      */
      clientsocket=socketopener.openSocket(ipaddress,portno,timeinsec);
   }
}
public void run()
{
int i=0;
String errororwarning="";
String errorstring="";
String inputstring="";
boolean starttag=false;
boolean endtag=false;
boolean startwritting=false;
String messagebody="";
String tagentity="";
Vector tagvactor;
tagvactor=new Vector();
int vactorindex=0;
int starttagsing=0;
while (true)
{
try
{
i=in.read();
if (i==-1)
{
   break;
}
else
{
inputstring=inputstring+(char)i;
if ((char)i=='<')
{
   starttag=true;
   starttagsing=1;
   tagentity="";
```

```
    }
    else
    {
    if ((char)i=='/' && starttag==true )
    {
       if (starttagsing==1)
       {
          starttag=false;
          endtag=true;
          reservation.setTableContent(messagebody);
          startwritting=false;
          messagebody="";
          vactorindex=vactorindex-1;
          if (vactorindex>=0)
          tagvactor.removeElementAt(vactorindex);
       }
    }
    else
    {
    starttagsing=0;
    if((char)i=='>')
    {
    if (starttag)
    {
       sendername(tagentity);
       errorstring=checkForError(tagentity);
       if (!errorstring.equals(""))
       {
       if (errortype=="major")
       {
          errororwarning="Error";
          JOptionPane.showMessageDialog(null,errorstring,errororwarning,JOptionPane.PLAIN_MESSAGE);
       }
       else
       {
          errororwarning="Warning";
          JOptionPane.showMessageDialog(null,errorstring,errororwarning,JOptionPane.PLAIN_MESSAGE);
       }
    }
    startwritting=true;
    tagvactor.insertElementAt(tagentity,vactorindex);
    vactorindex=vactorindex+1;
    starttag=false;
    }
}
else
{
   if (startwritting==true&&tagentity.trim().equals("body"))
   {
      messagebody=messagebody+(char)i;
   }
   if (starttag)
   {
      tagentity=tagentity+(char)i;
   }
}
}
}
}
}
catch(IOException ie)
{
}
}
isConnected=false;
}
/*
sendXMLToJabber: This method sends XML message string for the jabber server.
Parameter: outputmessage - object of String class
Return Value: N/A
*/
public void sendXMLToJabber(String outputmessage)
{
String[] tokenizerstring=outputmessage.split("\n");
for (int i=0;i<tokenizerstring.length;i++ )
{
   try
   {
      out.println(tokenizerstring[i]);
      out.flush();
   }
   catch(Exception e)
   {
      System.out.println("error");
      return;
   }
   out.flush();
   }
```

```
}
/*
getErrorString: This method checks input string for error code.
Parameter: codeid - object of String class
Return Value: String
*/
public String getErrorString(String codeid)
{
   if (codeid=="'401'")
   {
      errortype="minor";
      return "You have sent malformed syntax, which can not be understood by server.";
   }
   if (codeid=="'404'")
   {
      errortype="major";
      return "No User exist with this user id.";
   }
   if (codeid=="'405'")
   {
      errortype="minor";
      return "You have no right to send this command.";
   }
   if (codeid=="'409'")
   {
      errortype="major";
      return "A user with this jabber id is already exist. Please choose another user id.";
   }
   return "";
}
/*
checkForError: This method subtracts input string, retrieves errorcode and calls getErrorString()
Parameter: tagentity - object of String class
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if (tagentity.startsWith("error"))
   {
      if (tagentity.indexOf("code=")>0)
      {
         codeid=tagentity.substring(tagentity.indexOf("code=")+6,tagentity.indexOf("type="));
         error=getErrorString(codeid.trim());
      }
   }
   return error;
}
/*
sendername: This method extracts sender name from input message.
Parameter: tagentity - object of String class
Return Value: N/A
*/
public void sendername(String tagentity)
{
   String sendername="";
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("id=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("id="));
      }
   }
}
}
/*
class SocketOpener - This class is used to create an object of socket class.
Constructor:
openSocket
Methods:
getSocket: This method returns a handler of the object of socket class.
*/
class SocketOpener
{
   private Socket socket;
   public  Socket openSocket(String hostip,int portnumber,int timeinsec)
   {
      try
      {
         socket=new Socket(hostip,portnumber);
      }
      catch(IOException ie)
      {
      }
      return getSocket();
```

```
    }
    public SocketOpener()
    {
    }
    /*
    getSocket: This method returns a handler of the object of socket class.
    Parameter: N/A
    Return Value: Socket
    */
    public Socket getSocket()
    {
       return socket;
    }
}
```

Download this listing.

In the above code, the constructor of the SocketClass class takes an object of the ReservationStatus class, ipaddress, portno, and wait as input parameters. The object of the ReservationStatus class allows an end user to invoke the methods of the ReservationStatus class. The ipaddress string retrieves the ip address of the Jabber server to connect. The portno int retrieves the port number of the Jabber server to open the socket. The wait int retrieves the initial time to wait before connecting to the Jabber server.

The methods defined in Listing 6-4 are:

- `openPort()`: Creates an object of the SocketOpener class and calls the openSocket() method by using the object of the SocketOpener class to open a socket between the Airline Reservation application and the Jabber server.
- `sendXMLToJabber()`: Sends an XML message string to the Jabber server.
- `checkForError()`: Checks for errors in the information specified by end users and calls the getErrorString() method. The checkForError() method passes the error code to the getErrorString() method as an input parameter.
- `getErrorString()`: Shows an error message to end users based on the error code passed as an input parameter.
- `sendername()`: Retrieves the name of the sender from the input message and checks the identity of the sender.
- `run()`: Listens for the response sent by the Jabber server.
- `OpenSocket()`: Initializes the object of the SocketOpener class.
- `getSocket()`: Returns an object of the Socket class.

Searching for the Airline Schedules

The SocketConector.java file reads the airlines.xml file to search for airline schedules that match an end user's query. The SocketConector class opens the socket to receive end user's query from the Jabber server and sends the matched airline schedules to the Jabber server. Listing 6-5 shows the contents of the SocketConnector.java file:

Listing 6-5: The SocketConnector.java File

```
/*Imports required java.util classes*/
import java.util.*;
/*Imports required java.io classes*/
import java.io.*;
/*Imports required  java.net classes*/
import java.net.*;
/*Imports required swing classes*/
import javax.swing.*;
/*Imports required List class*/
import java.util.List;
/*Imports required Element class*/
import org.jdom.Element;
/*Imports required Iterator class*/
import java.util.Iterator;
/*
class SocketConnector - This class is used for reading the input messages and writing output message
Constructor:
SocketConnector-This constructor calls openPort() method to open a client socket.
Methods:
openPort
*/
class SocketConnector implements Runnable
{
/*Declare object of Vector class.*/
public Vector tagvactor;
/* Declare object of PrintWriter class.*/
PrintWriter out = null;
/*Declare object of BufferedReader class.*/
BufferedReader in = null;
/*Declare object of Socket class.*/
Socket clientsocket;
/*Declare object of Thread class.*/
Thread inputmessagethread;
/*Declare objects of String class.*/
String errortype;
String ipaddress="localhost";
String resource="airlines";
String username="user10";
String password="user10";
String sendernamestring;
int portno=5222;
int wait=1000;
public boolean isConnected;
/*
SocketConnector: This method call an openPort() method.
Parameter: ipaddress - object of String class, portno, wait
Return Value: boolean
*/
public SocketConnector(String ipaddress,int portno,int wait)
{
   this.ipaddress=ipaddress;
   this.portno=portno;
   this.wait=wait;
   openPort(ipaddress,portno,wait);
   if (clientsocket!=null)
   {
      isConnected=true;
   }
   else
      {
      System.err.println("Server not found.");
      return;
   }
   try
   {
      /*
      Initializes the object of PrintWriter class.
      */
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      /*
      Initializes the object of BufferedReader class.
      */
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
   }
```

```
   catch(Exception e)
   {
      e.printStackTrace();
   }
   String sessionStratString;
   String authentication;
   String registrationstring;
   sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
   sessionStratString=sessionStratString+" <stream:stream";
   sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
   sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   sendXMLToJabber(sessionStratString);
   authentication="<iq type=\"set\" id=\"1301\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+" <username>"+username+"</username>";
   authentication=authentication+" <password>"+password+"</password>";
   authentication=authentication+" <resource>"+resource+"</resource> ";
   authentication=authentication+" </query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
   /*
   Initializes the object of Thread class.
   */
   inputmessagethread=new Thread(this);
   /*
   Calls start method of inputmessagethread.
   */
   inputmessagethread.start();
}
public static void main(String args[])
{
   SocketConnector socketconnector=new SocketConnector("localhost",5222,1000);
}
/*
openPort: This method creates an object of SocketOpener class and class openSocket method of this
class.
Parameter: ipaddress - Object of String class,portno - int, timeinsec - int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      clientsocket=SocketOpenerClass.openSocket(ipaddress,portno,timeinsec);
   }
}
public void run()
{
   int i=0;
   String errororwarning="";
   String errorstring="";
   String inputstring="";
   boolean starttag=false;
   boolean endtag=false;
   boolean startwritting=false;
   String messagebody="";
   String tagentity="";
   Vector tagvactor;
   tagvactor=new Vector();
   int vactorindex=0;
   int starttagsing=0;
   while (true)
   {
   try
   {
   i=in.read();
   if (i==-1)
   {
      break;
   }
   else
   {
      inputstring=inputstring+(char)i;
      if ((char)i=='<')
      {
         starttag=true;
         starttagsing=1;
         tagentity="";
      }
      else
      {
         if ((char)i=='/' && starttag==true&&starttagsing==1)
```

```
            {
               starttag=false;
               endtag=true;
               if (!messagebody.trim().equals(""))
               {
                  writemessage(messagebody,sendernamestring);
               }
               startwritting=false;
               messagebody="";
               vactorindex=vactorindex-1;
               if (vactorindex>=0)
               tagvactor.removeElementAt(vactorindex);
            }
            else
            {
               starttagsing=0;
               if((char)i=='>')
               {
               if (starttag)
               {
                  sendername(tagentity);
                  errorstring=checkForError(tagentity);
                  if (!errorstring.equals(""))
                  {
                     if (errortype=="major")
                     {
                        errororwarning="Error";
                        JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                        JOptionPane.PLAIN_MESSAGE);
                     }
                     else
                     {
                        errororwarning="Warning";
                        JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                        JOptionPane.PLAIN_MESSAGE);
                     }
                  }
                  startwritting=true;
                  tagvactor.insertElementAt(tagentity,vactorindex);
                  vactorindex=vactorindex+1;
                  starttag=false;
                  }
               }
               else
               {
                  if (startwritting==true&&tagentity.trim().equals("body"))
                  {
                     messagebody=messagebody+(char)i;
                  }
                  if (starttag)
                  {
                     tagentity=tagentity+(char)i;
                  }
               }
            }
         }
      }
   }
}
catch(IOException ie)
{
}
}
isConnected=false;
try
{
   inputmessagethread.join();
}
catch(Exception e)
{
}
}
/*
writemessage: This method creates an object of XMLReader class, reads data from XML file, and send
result to the client.
Parameter: messagebody - Object of String class, sendername - Object of String class.
Return Value: N/A
*/
public void writemessage(String messagebody,String sendername)
{
String[] spletedmessage=messagebody.split("#");
if (spletedmessage[1].indexOf("id")>=0)
{
   return;
}
String origin=spletedmessage[1].substring(7);
String destination=spletedmessage[2].substring(12);
String ddate=spletedmessage[3].substring(6);
String rdate=spletedmessage[4].substring(6);
```

```
String nationality=spletedmessage[5].substring(12);
String typeofjurney=spletedmessage[6].substring(13);
java.util.List flightdescriptionlist;
XMLReader xmlreader=new XMLReader();
flightdescriptionlist=xmlreader.readFile("airlines.xml");
Iterator i = flightdescriptionlist.iterator();
String msgstr="";
String startmsgstr="@";
while (i.hasNext())
{
Element dateelement;
Element depdate=null;
Element arrdate=null;
Element eclass=null;
Element gclass=null;
Element eclassprice=null;
Element gclassprice=null;
Element arrtime=null;
Element dtime=null;
Element flightelement = (Element) i.next();
Element idelement=flightelement.getChild("id");
Element originelement=flightelement.getChild("origin");
Element flighnotelement=flightelement.getChild("flightno");
Element aircrafttypeelement=flightelement.getChild("aircrafttype");
Element noofstopelement=flightelement.getChild("noofstop");
Element destinationelement=flightelement.getChild("destination");
List executiveelement=flightelement.getChildren("class");
Iterator classiterator=executiveelement.iterator();
while (classiterator.hasNext())
{
   Element classelement=(Element) classiterator.next();
   eclass=classelement.getChild("executive");
   gclass=classelement.getChild("economy");
   eclassprice=classelement.getChild("executiveprice");
   gclassprice=classelement.getChild("economyprice");
}
Element economyelement=flightelement.getChild("economy");
List departurdatetimelist=flightelement.getChildren("departurdatetime");
List arrivaldatetimelist=flightelement.getChildren("arrivaldatetime");
Iterator idepartur=departurdatetimelist.iterator();
Iterator iarrivalarrival=arrivaldatetimelist.iterator();
while(idepartur.hasNext())
{
   Element departuredateelement=(Element) idepartur.next();
   depdate=departuredateelement.getChild("date");
   dtime=departuredateelement.getChild("dtime");
}
while(iarrivalarrival.hasNext())
{
   Element arrivaldateelement=(Element) iarrivalarrival.next();
   arrdate=arrivaldateelement.getChild("date");
   arrtime=arrivaldateelement.getChild("atime");
}
Element arrivaldateelement=flightelement.getChild("arrivaldatetime");
if (origin.equals(originelement.getTextTrim())&&destination.equals
(destinationelement.getTextTrim())&&ddate.equals(depdate.getTextTrim()))
{
   msgstr=msgstr+"choice=YES";
   msgstr=msgstr+"#flight="+flighnotelement.getTextTrim();
   msgstr=msgstr+"#aircrafttype="+aircrafttypeelement.getTextTrim();
   msgstr=msgstr+"#origin="+originelement.getTextTrim();
   msgstr=msgstr+"#noofstop="+noofstopelement.getTextTrim();
   msgstr=msgstr+"#destination="+destinationelement.getTextTrim();
   msgstr=msgstr+"#ddate="+depdate.getTextTrim()+" "+dtime.getTextTrim();
   msgstr=msgstr+"#adate="+arrdate.getTextTrim()+" "+arrtime.getTextTrim();
   msgstr=msgstr+"#executive="+eclass.getTextTrim();
   msgstr=msgstr+"#economy="+gclass.getTextTrim();
   msgstr=msgstr+"#id="+idelement.getTextTrim();
   msgstr=msgstr+"#eclassprice="+eclassprice.getTextTrim();
   msgstr=msgstr+"#gclassprice="+gclassprice.getTextTrim();
   msgstr=msgstr+startmsgstr;
}
if(typeofjurney.equals("roundtrip"))
{
   if (destination.equals(originelement.getTextTrim())&&origin.equals
   (destinationelement.getTextTrim())&&rdate.equals(depdate.getTextTrim()))
   {
      msgstr=msgstr+"choice=YES";
      msgstr=msgstr+"#flight="+flighnotelement.getTextTrim();
      msgstr=msgstr+"#aircrafttype="+aircrafttypeelement.getTextTrim();
      msgstr=msgstr+"#origin="+originelement.getTextTrim();
      msgstr=msgstr+"#noofstop="+noofstopelement.getTextTrim();
      msgstr=msgstr+"#destination="+destinationelement.getTextTrim();
      msgstr=msgstr+"#ddate="+depdate.getTextTrim()+" "+dtime.getTextTrim();
      msgstr=msgstr+"#adate="+arrdate.getTextTrim()+" "+arrtime.getTextTrim();
      msgstr=msgstr+"#executive="+eclass.getTextTrim();
      msgstr=msgstr+"#economy="+gclass.getTextTrim();
      msgstr=msgstr+"#id="+idelement.getTextTrim();
```

```
            msgstr=msgstr+"#eclassprice="+eclassprice.getTextTrim();
            msgstr=msgstr+"#gclassprice="+gclassprice.getTextTrim();
            msgstr=msgstr+startmsgstr;
         }
      }
      }
      if (msgstr.equals(""))
      {
         sendXMLToJabber("<message to='"+sendername+" type=\"chat\"><body
         >No matching record found</body></message>");
      }
      else
      {
         sendXMLToJabber("<message to='"+sendername+"
         type=\"chat\"><body>"+msgstr+"</body></message>");
      }
      }
      /*
      sendXMLToJabber: This method sends XML message string for the jabber server.
      Parameter: outputmessage - object of String class
      Return Value: N/A
      */
      public void sendXMLToJabber(String outputmessage)
      {
         String[] tokenizerstring=outputmessage.split("\n");
         System.out.println(outputmessage);
         for (int i=0;i<tokenizerstring.length;i++ )
         {
         try
         {
            out.println(tokenizerstring[i]);
            out.flush();
         }
         catch(Exception e)
         {
            System.out.println("error");
            return;
         }
         out.flush();
         }
      }
      /*
      getErrorString: This method checks input string for error code.
      Parameter: codeid - object of String class
      Return Value: String
      */
      public String getErrorString(String codeid)
      {
         if (codeid=="'401'")
         {
            errortype="minor";
            return "You have sent malformed syntax, which can not be understood by server.";
         }
         if (codeid=="'404'")
         {
            errortype="major";
            return "No User exist with this user id.";
         }
         if (codeid=="'405'")
         {
            errortype="minor";
            return "You have no right to send this command.";
         }
         if (codeid=="'409'")
         {
            errortype="major";
            return "A user with this jabber id is already exist. Please choose another user id.";
         }
         return "";
         }
         /*
         checkForError: This method substracts input string, retrieves errorcode, and calls getErrorString
         Parameter: tagentity - object of String class
         Return Value: String
         */
         public String checkForError(String tagentity)
         {
            String error="";
            String codeid="";
            if (tagentity.startsWith("error"))
            {
            if (tagentity.indexOf("code=")>0)
            {
               codeid=tagentity.substring(tagentity.indexOf("code=")+6,tagentity.indexOf("type="));
               error=getErrorString(codeid.trim());
            }
         }
```

```
   return error;
   }
   /*
   sendername: This method extracts sender name from input message.
   Parameter: tagentity - object of String class
   Return Value: N/A
   */
   public void sendername(String tagentity)
   {
   String sendername="";
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
      {
         sendernamestring=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to="))
      }
   }
}
}
/*
class SocketOpenerClass - This class is used to create an object of socket class.
Constructor:
openSocket
Methods:
getSocket: This method returns an handler of the object of socket class.
*/
class SocketOpenerClass implements Runnable
{
private String openerhost;
private int openerport;
private Socket socket;
public static Socket openSocket(String hostip,int portnumber,int timeinsec)
{
SocketOpenerClass opener=new SocketOpenerClass(hostip,portnumber);
Thread t=new Thread(opener);
t.start();
try
{
t.join(timeinsec);
}
catch(InterruptedException exception)
{
}
return opener.getSocket();
}
public SocketOpenerClass(String host,int port)
{
socket=null;
openerhost=host;
openerport=port;
}
public Socket getSocket()
{
return socket;
}
public void run()
{
try
{
socket=new Socket(openerhost,openerport);
}
catch(IOException ie)
{
}
}
}
```

Download this listing.

In the above code, the constructor of the SocketConnector class takes the ipaddress string, and portno and wait intergers as input parameters. The ipaddress string retrieves the ip address of the Jabber server to connect. The portno integer retrieves the port number of the Jabber server to open a socket and the wait integer retrieves the initial time to wait before connecting to the Jabber server.

The methods defined in Listing 6-5 are:

- `openPort()`: Opens a socket between the Jabber server and the Airline Reservation application.
- `writemessage()`: Creates an object of the XMLReader class to read airline schedules from the airlines.xml file. The writemessage() method calls the sendXMLToJabber() method and passes the matched airline schedules to the sendXMLToJabber() method.
- `sendXMLToJabber()`: Takes the matched airline schedule as an input parameter and sends the information to the Jabber server.
- `sendername()`: Retrieves the name of the sender from the input message and checks the identity of the sender.

- `run()`: Listens for the response sent by the Jabber server.
- `SocketOpenerClass()`: Initializes the object of the SocketOpenerClass class.
- `getSocket()`: Returns an object of the Socket class.

◄ PREVIOUS NEXT ►

Showing the Airline Schedules

The ReservationResultTable.java file shows the available airline schedules in the user interface of the Airline Reservation application. Listing 6-6 shows the contents of the ReservationResultTable.java file:

Listing 6-6: The ReservationResultTable.java File

```
/*
Imports required classes from javax.swing package
*/
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
/*
Imports required classes from javax.swing.table package.
*/
import javax.swing.table.*;
/*
Imports required classes from javax.swing.event package.
*/
import javax.swing.event.*;
/*
Imports required classes from javax.swing package.
*/
import javax.swing.*;
/*
Imports required classes from java.awt package.
*/
import java.awt.*;
/*
Imports required classes from java.awt.event package.
*/
import java.awt.event.*;
/*
class SocketClass - This class is used for reading the input messages and writing output messages.
Constructor:
SocketClass - This constructor calls a method of SocketOpener class to open a client socket.
Methods:
*/
public class ReservationResultTable extends JPanel
{
   /*
   Declare the object of the ReservationTableModel class.
   */
   ReservationTableModel tablemodel;
   /* Declare the object of the JTable class.*/
   JTable table;
   public ReservationResultTable()
   {
      super(new GridLayout(1,0));
      /*
      Initializes the object of ReservationTableModel class.
      */
      tablemodel=new ReservationTableModel();
      /*
      Initializes the object of JTable class.
      */
      table = new JTable(tablemodel);
      table.setPreferredScrollableViewportSize(new Dimension(600, 100));
      /* Declare and initialize the object of JScrollPane class.*/
      JScrollPane scrollPane = new JScrollPane(table);
      add(scrollPane);
      setPreferredSize(new Dimension(600, 100));
   }
   /*
   class ReservationTableModel - This class is used to insert data into table.
   Methods:
   getColumnCount - Returns number of column in the table.
   setData - Set data into table cell.
   getRowCount -gets number of rows in the table.
   isCellEditable - Set cell edit property.
   setValueAt - Set data into table cell.
   */
   class ReservationTableModel extends AbstractTableModel
   {
      private String[] columnNames = {"Choice","Flight No.",
      "Aircraft type",
      "Origin",
      "No. of Stops",
      "Destination",
      "Dep.Date-time",
      "Arr.Date-time",
      "Executive Class",
```

```
      "Economy Class",
   };
   private Object[][] data = {};
   /*
   getColumnCount - Returns number of column in the table.
   Parameter: N/A
   Return Value: int
   */
   public int getColumnCount()
   {
      return columnNames.length;
   }
   /*
   setData - Set data into table celles.
   Parameter: dataarr - array of Object Type.
   Return Value: int
   */
   public void setData(Object[][] dataarr)
   {
      data=dataarr;
   }
   /*
   getRowCount -gets number of rows in the table.
   Parameter: dataarr - array of Object Type.
   Return Value: int
   */
   public int getRowCount()
   {
      return data.length;
   }
   /*
   getColumnName - Returns Column name
   Parameter: col
   Return Value: String
   */
   public String getColumnName(int col)
   {
      return columnNames[col];
   }
   /*
   getValueAt - Returns value from a table cell.
   Parameter: row, col
   Return Value: Object
   */
   public Object getValueAt(int row, int col)
   {
      return data[row][col];
   }
   /*
   getColumnClass - Returns the Class of a table cell.
   Parameter: col
   Return Value: Class
   */
   public Class getColumnClass(int col)
   {
      return getValueAt(0, col).getClass();
   }
   /*
   isCellEditable - Set cell edit property.
   Parameter: row, col
   Return Value: boolean
   */
   public boolean isCellEditable(int row, int col)
   {
   if (col >=1&&col<=7)
   {
      return false;
   }
   else
   {
      return true;
   }
}
/*
setValueAt - Set data into table cell.
Parameter: value - Object of Object type, row, col
Return Value: N/A
*/
public void setValueAt(Object value, int row, int col)
{
   data[row][col] = value;
   fireTableCellUpdated(row, col);
}
}
}
```

Download this listing.

In the above code, the constructor of the ReservationResultTable class creates a new instance of the ReservationTableModel class. The ReservationTableModel class shows the matched airline schedules received from the Jabber server in a tabular format.

The methods defined in Listing 6-6 are:

- `getRowCount()`: Returns the total number of records present in the search result.
- `isCellEditable()`: Checks whether or not the information received from the Jabber server is editable.
- `setValueAt()`: Shows the matched airline schedules in the user interface of the Airline Reservation application.

The ReservationResultTable.java file shows the airline schedules in the main window of the Airline Reservation application, as shown in Figure 6-4:

Figure 6-4: Showing the Airline Schedules

A confirmation message appears when an end user clicks the Submit button, as shown in Figure 6-5:

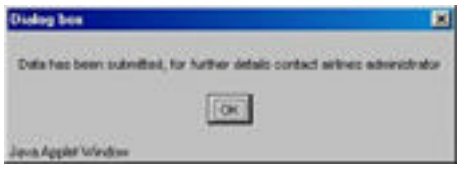


Figure 6-5: Confirmation Message

Reading XML Files

The XMLReader.java file reads the airlines.xml file that contains the information about available airline schedules. Listing 6-7 shows the contents of the XMLReader.java file:

Listing 6-7: The XMLReader.java File

```
/*Imports required File classes*/
import java.io.File;
/*Imports required IOException classes*/
import java.io.IOException;
/*Imports required PrintStream classes*/
import java.io.PrintStream;
/*Imports required Iterator classes*/
import java.util.Iterator;
/*Imports required List classes*/
import java.util.List;
/*Imports required Document classes*/
import org.jdom.Document;
/*Imports required Element classes*/
import org.jdom.Element;
/*Imports required JDOMException classes*/
import org.jdom.JDOMException;
/*Imports required SAXBuilder classes*/
import org.jdom.input.SAXBuilder;
/*Imports required XMLOutputter classes*/
import org.jdom.output.XMLOutputter;
/*
class XMLReader -This class is used to read XML file.
Methods:
readFile
*/
public class XMLReader
{
/*Declare object of String class.*/
String filename;
/*Declare object of Document class.*/
Document doc;
/*
readFile: This method call an openPort() method.
Parameter: fname - object of String class.
Return Value: List
*/
public List readFile(String fname)
{
   filename = fname;
   PrintStream out = System.out;
   SAXBuilder builder = new SAXBuilder();
   try
   {
      doc = builder.build(new File(filename));
   }
   catch(Exception je) {}
   Element root = doc.getRootElement();
   List childlist = root.getChildren("flightdescription");
   return childlist;
   }
}
```

Download this listing.

In the above listing, the XMLReader.java file contains the readFile() method. The readFile() method takes the location of the airlines.xml file as an input parameter and reads the file.

The airlines.xml File

The airlines.xml file contains the airline schedules in an XML format. Listing 6-8 shows the contents of the airlines.xml file:

Listing 6-8: The airlines.xml File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<airlines>
<flightdescription>
<id>
1
</id>
<flightno>
usa1000
</flightno>
<aircrafttype>
AB300
</aircrafttype>
<origin>Chicago</origin>
<noofstop>
0
</noofstop>
<destination>Boston</destination>
<departurdatetime>
<date>
20040829
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040829
</date>
<atime>
2145
</atime>
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
<flightdescription id="2">
<id>
2
</id>
<flightno>
usa10001
</flightno>
<aircrafttype>
AB3001
</aircrafttype>
<origin>Chicago</origin>
<noofstop>
0
</noofstop>
<destination>Boston</destination>
<departurdatetime>
<date>
20040829
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040829
</date>
<atime>
2145
</atime>
</arrivaldatetime>
```

```
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
<flightdescription id="3">
<id>
3
</id>
<flightno>
usa10301
</flightno>
<aircrafttype>
AB4001
</aircrafttype>
<origin>Boston</origin>
<noofstop>
0
</noofstop>
<destination>Chicago</destination>
<departurdatetime>
<date>
20040830
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040830
</date>
<atime>
2145
</atime>
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
<flightdescription id="4">
<id>
4
</id>
<flightno>
usa10301
</flightno>
<aircrafttype>
AB4001
</aircrafttype>
<origin>Chicago</origin>
<noofstop>
0
</noofstop>
<destination>Boston</destination>
<departurdatetime>
<date>
20040830
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040830
</date>
```

```
<atime>
2145
</atime>
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
</airlines>
```

Download this listing.

◄ PREVIOUS NEXT ►

Unit Testing

To test the Airline Reservation application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Download the free implementation of Java Document Object Model (JDOM) to read XML files from the following URL:

   http://www.jdom.org/

3. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

4. Set the classpath of the lib directory of J2SDK and the jdom.jar file by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;D:\jdom.jar;
   ```

5. Copy the ReservationStatus.java, SocketClass.java, SocketConnector.java, ReservationResultTable.java, and XMLReader.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder where you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

6. To run the Airline Reservation application, specify the following command at the command prompt:

   ```
   java SocketConnector
   ```

7. The user interface of the Personal Information page appears.

8. Enter information in the fields of the Personal Information page, as shown in Figure 6-6:

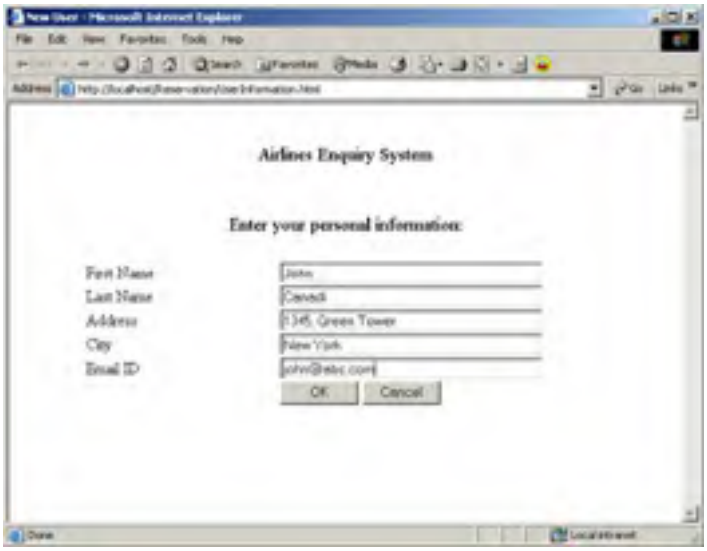


Figure 6-6: Entering Personal Information

9. Click OK to open the user interface of the Airline Reservation application.

10. Select Chicago in the From combo box and Boston in the To combo box. Specify the Departure Date and Nationality, as shown in Figure 6-7:

Figure 6-7: Searching for an Airline Schedule

11. Click the Search button to search for the specified airline schedule.
12. Select the flight number usa10001 and select the Economy check box to reserve the airline schedule, as shown in Figure 6-8:

Figure 6-8: Reserving an Airline Schedule

13. Click Submit to submit the reservation information. A confirmation dialog box appears.
14. Click OK to return to the Airline Reservation window.

Chapter 7: Creating a Contact List Application

Download CD Content

The Jabber protocol allows you to develop an application for creating and managing contact information by using the Jabber server. This chapter describes how to develop the Contact List application, which allows end users to manage contact information. By using the Contact List application, end users can add information to the contact list and check whether or not other end users are online. An end user can send messages to other end users added to the contact list.

Architecture of the Contact List Application

The Contact List application uses the following files:

- UserLogin.java: Creates the Login window for the Contact List application that allows end users to establish a connection with the Jabber server to communicate with other end users. This file allows end users to login as existing end users or register for a new account.
- ContactList.java: Shows a tree structure of the contact list created by end users. This file also allows end users to manage their contact list by adding, editing, or deleting contact information.
- UserInfo.java: Allows end users to view the information about other end users in the contact list.
- SocketClass.java: Opens a socket to send and receive messages through the Jabber server.
- EditDelete.java: Allows end users to edit or delete contact information from the contact list.
- DynamicTree.java: Creates and maintains a tree structure of the contact list. The tree structure is visible in the user interface of the Contact List application.
- ChatWindow.java: Creates a Chat window that allows end users to send messages to other end users present in the contact list.
- AddUser.java: Creates a user interface that allows end users to add other end users to the contact list.

Figure 7-1 shows the architecture of the Contact List application:

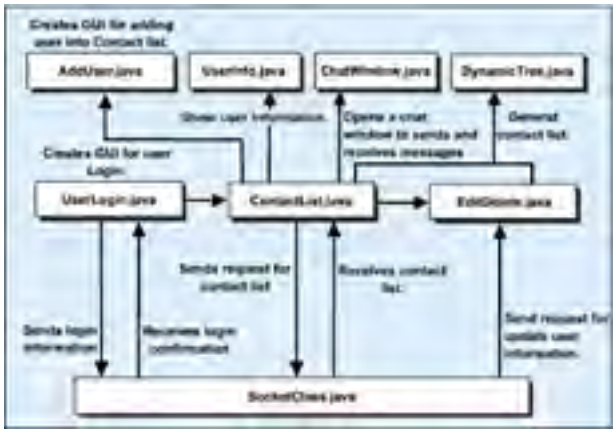


Figure 7-1: Architecture of the Contact List Application

The UserLogin.java file creates the Login window of the Contact List application that contains various labels, text boxes, and the Submit button. The UserLogin.java file allows end users to login as existing users or register for a new account.

When an end user enters the login information and clicks the Submit button on the Login window, the UserLogin.java file calls the ContactList.java file to create the user interface for the Contact List application. The ContactList.java file allows end users to add other end users and edit or delete the existing end users from the contact list. The ContactList.java file shows a tree structure of the contact list.

If an end user selects File-> Add User, the ContactList.java file calls the AddUser.java file to add other end users to the contact list. The AddUser.java file provides an interface with various labels, three text boxes, and two buttons, Add and Cancel.

If an end user selects File-> Manage User, the ContactList.java file calls the EditDelete.java file to edit or delete the existing end users from the contact list. The EditDelete.java file provides an interface containing two panes. The left pane shows a tree structure of the contact list and the right pane contains various labels, two text boxes, and three buttons: Up, Delete, and Cancel.

If an end user selects a specific end user name from the tree structure of the contact list, the ContactList.java file calls the ChatWindow.java file that allows end users to send messages to the selected end users. The ChatWindow.java file provides an interface with text area to view the text messages and the Send button to send the text messages.

The ChatWindow.java file calls the SocketClass.java file that opens a socket to send and receive messages with the help of the Jabber server.

When an end user right-clicks any user name in the contact list, the pop-up option, View Contact appears. If the end user selects the View Contact option, the ContactList.java file calls the UserInfo.java file to display the contact information of the selected end user.

PREVIOUS NEXT

Creating the Login Window

The UserLogin.java file creates the Login window for the Contact List application. Listing 7-1 shows the contents of the UserLogin.java file:

Listing 7-1: The UserLogin.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*
class UserLogin - This class is used to create a GUI for user login
Constructor:
UserLogin - This constructor creates GUI.
Methods:
actionPerformed
*/
public class UserLogin extends JFrame implements ActionListener
{
   /*Declares object of Container class.*/
   Container contentpane;
   /*Declares object of JTextField class.*/
   JTextField nametextfield;
   JTextField servernametextfield;
   /*Declares object of JPasswordField class.*/
   JPasswordField passwordtextfield;
   /*Declares object of JButton class.*/
   JButton submitbutton;
   /*Declares object of Socket class.*/
   Socket clientsocket;
   /*Declares object of String class.*/
   String servername;
   String username;
   String password;
   String resource;
   String registrationstring="";
   /*Declares object of JPanel class.*/
   JPanel note;
   JPanel combine;
   /*Declares object of SocketClass class.*/
   SocketClass socketclass;
   boolean submitclickd=false;
   public UserLogin ()
   {
      super("Login Window");
      this.clientsocket=clientsocket;
      contentpane=getContentPane();
      /*Initializes the objects of the JTextField class.*/
      nametextfield=new JTextField();
      servernametextfield=new JTextField();
      /*Initializes the object of the JPasswordField class.*/
      passwordtextfield=new JPasswordField();
      /*Initializes the object of the JPasswordField class.*/
      submitbutton=new JButton("Submit");
      /*Initializes the object of the JPanel class.*/
      combine=new JPanel();
      /*Initializes the object of JPanel class.*/
      note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
      /*Initializes the object of the SocketClass class.*/
      socketclass=new SocketClass();
      /*Sets action command for submitbutton.*/
      submitbutton.setActionCommand("submit");
      /*Adds action listener for submitbutton.*/
      submitbutton.addActionListener(this);
      submitbutton.setMnemonic('s');
      resource="Home";
      /*Declares and initializes the object of JPanel class.*/
      JPanel controlpanel=new JPanel();
      /*Declares and initializes the objects of JLabel class.*/
      JLabel room= new JLabel("User Name");
      JLabel name= new JLabel("Password ");
      JLabel serverlabel= new JLabel("Server IP");
      JLabel firstnode= new JLabel("* If you are not a registerd user your account will be ");
      JLabel secondnode= new JLabel("created automatically.");
      /*Declares and initializes the object of JPanel class.*/
      JPanel lable=new JPanel();
      /*Sets GridLayout of controlpanel.*/
      controlpanel.setLayout(new GridLayout(4, 2, 3, 3));
```

```
        controlpanel.setBorder(BorderFactory.createTitledBorder("Login Information"));
        /*Adds labels and textfields to controlpanel.*/
        controlpanel.add(room);
        controlpanel.add(nametextfield);
        controlpanel.add(name);
        controlpanel.add(passwordtextfield);
        controlpanel.add(serverlabel);
        controlpanel.add(servernametextfield);
        firstnode.setForeground(Color.red);
        secondnode.setForeground(Color.red);
        /*Adds labels to note.*/
        note.add(firstnode);
        note.add(secondnode);
        /*Declares and initializes the object of JLabel class.*/
        JLabel heading=new JLabel("Login Window", JLabel.CENTER);
        lable.add(heading,BorderLayout.NORTH);
        heading.setFont(new Font("verdana", 1, 12));
        contentpane.add(lable,BorderLayout.NORTH);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        contentpane.add(controlpanel,BorderLayout.CENTER);
        /*Declares and initializes the objects of JPanel class.*/
        JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
        JPanel bp=new JPanel(new FlowLayout());
        bp.add(submitbutton);
        buttonpanel.add(note);
        buttonpanel.add(bp);
        contentpane.add(buttonpanel, BorderLayout.SOUTH);
        setBounds(5, 5, 300, 240);
        submitbutton.addActionListener(this);
        show();
    }
    public void actionPerformed(ActionEvent ae)
    {
        String actioncommand=ae.getActionCommand();
        servername=servernametextfield.getText().trim();
        username=nametextfield.getText().trim();
        password=passwordtextfield.getText().trim();
        /*This block will be executed when an end user clicks the submit button.*/
        if (actioncommand.equals("submit"))
        {
            if (!submitclickd)
            {
                submitclickd=true;
                if (!socketclass.isConnected())
                {
                    socketclass.socketoOpenClass(servername, 5222, 1000);
                }
                socketclass.setUserNameAndPassword(username,password);
                if (socketclass.isConnected()&&(!socketclass.isWaitingForAuth()))
                {
                    socketclass.sendSessionStartMessage();
                    socketclass.sendAuthorized();
                    socketclass.setWaitForAuth(true);
                    socketclass.setUserLoginHandler(this);
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Server is not
                    found.", "wrring", JOptionPane.PLAIN_MESSAGE);
                }
        }
        else
        {
            submitclickd=false;
        }
    }
}
public static void main(String[] args)
{
try
{
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
    e.printStackTrace();
}
/*Declares and initializes the objects of UserLogin class.*/
UserLogin userlogin=new UserLogin();
}
}
```

Download this listing.

In the above code, the main() method creates an instance of the UserLogin class. The UserLogin class defines the actionPerformed() method. This method acts as an event listener and activates an appropriate method based on the action an end user performs.

The UserLogin.java file creates the Login window of the Contact List application, as shown in Figure 7-2:

Figure 7-2: The Login Window

◄ PREVIOUS NEXT ►

Creating the User Interface of the Contact List Application

The ContactList.java file creates the user interface for the Contact List application. Listing 7-2 shows the contents of the ContactList.java file:

Listing 7-2: The ContactList.java File

```
/*Imports required swing classes*/
import javax.swing.*;
/*Imports required awt classes*/
import java.awt.*;
/*Imports required tree classes*/
import javax.swing.tree.*;
/*Imports required event classes*/
import javax.swing.event.*;
import java.awt.event.*;
/*Imports required util classes*/
import java.util.Hashtable;
import java.util.Enumeration;
/*
class ContactList - This class is the main class for this application. It is used to create a tree
and a menubar.
Constructor:
ContactList - This constructor creates GUI.
Methods:
writemessage: This method is called to send input message to insertTextInToTextPane function.
eraseText: This method is called to send input message to insertTextInToTextPane function.
addToRosterTree: This method is called to add user name in to tree.
*/
public class  ContactList extends JFrame implements ActionListener, MouseListener
{
   /*Declares object of Container class.*/
   Container container = null;
   /*Declares objects of JMenuItem class.*/
   JMenuItem adduser=null;
   JMenuItem manageuser = null;
   /*Declares objects of DynamicTree class.*/
   DynamicTree tree;
   /*Declares objects of DynamicTree class.*/
   ContactListPopup popup=null;
   /*Declares objects of JPanel class.*/
   JPanel popuppanel=null;
   /*Declares objects of JLabel class.*/
   JLabel display=null;
   /*Declares objects of SocketClass class.*/
   SocketClass socketclass;
   /*Declares objects of JTree class.*/
   JTree rostertree;
   /*Declares objects of String class.*/
   String selecteditemvalue;
   /*Declares objects of Hashtable class.*/
   Hashtable hashnode;
   Hashtable chatwindowhash;
   /*Declares objects of AddUser class.*/
   AddUser add;
   boolean nouserexistsinlist;
   public ContactList(SocketClass socketclass)
   {
      super("Contact List");
      container = this.getContentPane();
      this.socketclass=socketclass;
      /*Initializes object of the JPanel class.*/
      popuppanel = new JPanel();
      /*Initializes objects of the Hashtable class.*/
      hashnode=new Hashtable();
      chatwindowhash=new Hashtable();
      /*Initializes objects of Hashtable class.*/
      display = new JLabel(" ");
      display.setOpaque(true);
      popuppanel.add(display);
      /*Sets window's default closing operation.*/
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      popuppanel.addMouseListener(this);
      display.addMouseListener(this);
      /*Initializes objects of JPanel class.*/
      popuppanel =  new JPanel(false);
      /*Declares and initializes objects of the JMenuBar class.*/
      JMenuBar menubar = new JMenuBar();
      /*Declares and initializes objects of the JMenu class.*/
      JMenu filemenu = new JMenu("File");
      /*Declares and initializes objects of the JMenuItem class.*/
      adduser= new JMenuItem("Add User");
      manageuser= new JMenuItem("Manage User");
      /*Adds filemenu to the menubar.*/
```

```
   menubar.add(filemenu);
   setJMenuBar(menubar);
   /*Adds menuitems to the filemenu.*/
   filemenu.add(adduser);
   filemenu.add(manageuser);
   /*Adds action listener to menu items.*/
   adduser.addActionListener(this);
   manageuser.addActionListener(this);
   /*Initializes objects of ContactListPopup class.*/
   popup = new ContactListPopup(this);
   /*Declares and initializes objects of DefaultMutableTreeNode class.*/
   DefaultMutableTreeNode root = new  DefaultMutableTreeNode("My Contact List");
   /*Initializes objects of DynamicTree class.*/
   tree=new DynamicTree(root);
   rostertree=tree.getTreeHandler();
   /*Adds action listener to the rostertree.*/
   rostertree.addMouseListener(this);
   /*Declares and initializes objects of JScrollPane class.*/
   JScrollPane scrollpane = new JScrollPane(tree);
   /*Adds mouse listener to the scrollpane.*/
   scrollpane.addMouseListener(this);
   /*Adds scrollpane to the container.*/
   container.add(scrollpane);
   setSize(300, 400);
   DefaultMutableTreeNode nousertreenode=new DefaultMutableTreeNode("No user exists in your conta
   list");
   tree.addObject(null,nousertreenode);
   nouserexistsinlist=true;
   show();
}
/*
writemessage: This method is called to send input message to insertTextInToTextPane function.
Parameter: pmessage - Object of  String class, psendername - Object of  String class,
preceivername - Object of  String class
Return Value: N/A
*/
public void writemessage(String pmessage,String psendername,String preceivername)
{
   String sendernameto=psendername;
   String[] splitedsendername=psendername.split("/");
   psendername="'"+splitedsendername[0]+"'";
   if (chatwindowhash.get(psendername)!=null)
   {
      ((ChatWindow)chatwindowhash.get(psendername)).insertTextInToTextPane(pmessage,sendernameto)
   }
   else
   {
      String[] nickname=psendername.split("@");
      ChatWindow chatwindow=new ChatWindow(nickname[0].substring(1, nickname[0].length()),
      psendername,socketclass);
      chatwindow.insertTextInToTextPane(pmessage,psendername);
      chatwindowhash.put(psendername,chatwindow);
   }
}
/*
eraseText: This method is called to send input message to insertTextInToTextPane function.
Parameter: pmessage - Object of  String class, psendername - Object of  String class,
preceivername - Object of  String class
Return Value: N/A
*/
public void eraseText(String sendername1)
{
   String sendername2=sendername1;
   String[] splitedsendername=sendername1.split("/");
   sendername1="'"+splitedsendername[0]+"'";
   if (chatwindowhash.get(sendername1)!=null)
   {
      ((ChatWindow)chatwindowhash.get(sendername1)).receivemessagetextarea.setText("");
      ((ChatWindow)chatwindowhash.get(sendername1)).setVisible(false);
   }
}
/*
addToRosterTree: This method is called to add user name in to tree.
Parameter: jid - Object of  String class, contactpersonname - Object of  String class.
Return Value: N/A
*/
public void addToRosterTree(String jid,String contactpersonname)
{
if (nouserexistsinlist)
{
   tree.clear();
}
else
{
   nouserexistsinlist=false;
}
contactpersonname=contactpersonname.substring(1,contactpersonname.length()-1);
if (!hashnode.containsKey(contactpersonname))
```

```
    {
       hashnode.put(contactpersonname,jid);
       DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(contactpersonname);
       tree.addObject(null,treenode);
    }
}
/*
removeFromRosterTree: This method is called to remove user from tree.
Parameter: username - Object of  String class.
Return Value: N/A
*/
public void removeFromRosterTree(String username)
{
   String[] splitedusername=username.split("/");
   String keyval=add.getNickName();
   String removestring;
   removestring="<iq type='set' id='1'>";
   removestring=removestring+"<query xmlns='jabber:iq:roster'>";
   removestring=removestring+"<item jid='"+splitedusername[0]+"' name='"+keyval+"'
   subscription='remove'/>";
   removestring=removestring+"</query></iq>";
   socketclass.sendXMLToJabber(removestring);
   hashnode.remove(keyval);
   Enumeration enum=hashnode.keys();
   tree.clear();
   while (enum.hasMoreElements())
   {
      DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(enum.nextElement());
      tree.addObject(treenode);
   }
}
/*
checkHash: This method is called to check that a user whose user name is in parameter one is exists
in hashtable or not.
Parameter: username - Object of String class.nname - Object of  String class.
Return Value: N/A
*/
public boolean checkHash(String uname,String nname)
{
   String tempuser="'"+uname+"'";
   String tempnname=nname;
   if (hashnode.containsKey(nname))
   {
      return true;
   }
   if (hashnode.containsValue(tempuser))
   {
      return true;
   }
   return false;
}

public void mousePressed(MouseEvent e)
{
   popup.setVisible(false);
}
public void mouseReleased(MouseEvent e)
{
   int selRow = rostertree.getRowForLocation(e.getX(), e.getY());
   TreePath selPath = rostertree.getPathForLocation(e.getX(), e.getY());
   TreePath path=rostertree.getSelectionPath();
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)rostertree.getLastSelectedPathComponent();
   if(node!=null)
   {
      if(!(node.toString().equals("My Contact List")))
      {
         if (e.getButton() == MouseEvent.BUTTON1)
         {
            if(selRow != -1)
            {
               if(e.getClickCount() == 2)
               {
                  selecteditemvalue=(String)hashnode.get(node.toString());
                  if (!chatwindowhash.containsKey(selecteditemvalue))
                  {
                     ChatWindow chatwindow=new ChatWindow(node.toString(), selecteditemvalue,
                     socketclass);
                     chatwindowhash.put(selecteditemvalue, chatwindow);
                  }
                  else
                  {
                     ((ChatWindow)chatwindowhash.get(selecteditemvalue)).show();
                  }
               }
            }
         }
         else if (e.getButton() == MouseEvent.BUTTON3)
         {
```

```
            if(selRow != -1)
            {
               if(e.getClickCount() ==1)
               {
                  popup.setVisible(false);
                  selecteditemvalue=(String)hashnode.get(node.toString());
                  popup = new ContactListPopup(this);
                  Point point=this.getLocation();
                  popup.setLocation(e.getX()+(int)point.getX(),e.getY()+(int)point.getY()+30);
                  popup.setVisible(true);
               }
            }
         }
      }
   }
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void actionPerformed(ActionEvent ae)
{
   if(ae.getSource() == adduser)
   {
      add =new AddUser(socketclass);
      add.show();
   }
   if(ae.getSource() == manageuser)
   {
      EditDelete editdelete = new EditDelete(hashnode,socketclass,tree);
      editdelete.show();
   }
}
/*
class ContactListPopup - This class is used to create a popup menu.
Constructor:
ContactListPopup-This constructor creates popup menu.
Methods:
actionPerformed
*/
public class ContactListPopup extends JPopupMenu implements ActionListener
{
   JMenuItem userinfomenuitem;
   ContactList useRightButton;
   String jabberid;
   public void setNodeName(String nodename)
   {
      jabberid=nodename;
   }
   public ContactListPopup(ContactList urb)
   {
      useRightButton = urb;
      userinfomenuitem = new JMenuItem("View Contact");
      userinfomenuitem.addActionListener(this);
      add(userinfomenuitem);
   }
   public void actionPerformed(ActionEvent e)
   {
      if (e.getSource() == userinfomenuitem)
      {
         String vacrstring="";
         vacrstring="<iq to="+useRightButton.selecteditemvalue+" type='get' id='1'>";
         vacrstring=vacrstring+"<vCard xmlns='vcard-temp'/></iq>";
         socketclass.sendXMLToJabber(vacrstring);
         popup.setVisible(false);
      }
   }
}
}
```

Download this listing.

In the above code, the constructor of the ContactList class takes an object of the SocketClass class as an input parameter. The object allows an end user to invoke methods of the SocketClass class.

The methods defined in Listing 7-2 are:

- `writemessage()`: Inserts the text message specified by an end user in the text area of the interface.
- `eraseText()`: Refreshes the text area every time an end user selects another end user's name from the contact list.
- `addToRosterTree()`: Adds other end users to the contact list.
- `removeFromRosterTree()`: Removes the existing end users from the contact list.
- `checkHash()`: Checks whether or not the specified end user already exists in the contact list.

- `ContactListPopup()`: Creates a popup menu to view the contact information of the selected end user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The ContactList.java file creates the main window of the Contact List application, as shown in Figure 7-3:

Figure 7-3: User Interface of the Contact List Application

Select the File menu to view the File menu options.

Figure 7-4 shows the File menu options of the Contact List application:

Figure 7-4: The File Menu of the Contact List Application

Select File-> Add User to add other end users to the contact list and send a welcome message to the added users.

Select File-> Manage User to edit or delete existing end users from the contact list.

Select any end user from the contact list to send messages to the selected user.

◄ PREVIOUS NEXT ►

Adding a Contact

The AddUser.java file creates the user interface to add other end users to the contact list and send a welcome message to the added end user. Listing 7-3 shows the contents of the AddUser.java file:

Listing 7-3: The AddUser.java File

```
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required javax.swing.event classes*/
import javax.swing.event.*;
/*Imports required java.awt classes*/
import java.awt.*;
/*mports required java.awt.event classes*/
import java.awt.event.*;
/*
class AddUser - This class is used to create a GUI for add a new user in user roster.
Constructor:
AddUser-This constructor creates GUI.
Methods:
getNickName: This method is called retrieve user nick name.
*/
public class  AddUser extends JFrame implements ActionListener
{
/*Declares object of Container class.*/
Container container;
/*Declares object of GridBagLayout class.*/
GridBagLayout gridbaglayout;
/*Declares object of JLabel class.*/
JLabel useridlabel;
JLabel nicknamelabel;
/*Declares object of JTextField class.*/
JTextField useridtext;
JTextField nicknametext;
/*Declares object of JTextPane class.*/
JTextPane messagetextarea;
/*Declares object of JTextPane class.*/
JButton addbutton;
JButton cancelbutton;
/*Declares object of SocketClass class.*/
SocketClass addsocketclass;
/*Declares object of String class.*/
String uid;
String nickname;
/*Declares object of JScrollPane class.*/
JScrollPane scpane;
public AddUser(SocketClass socketclass)
{
   super("Add Contact");
   container = this.getContentPane();
   addsocketclass=socketclass;
   /*Declares and initializes the objects of JLabel class.*/
   JPanel mainpane=new JPanel(new GridLayout(2,1));
   /*Declares and initializes the objects of JPanel class.*/
   JPanel labeltextpanel=new JPanel();
   container.add(new JLabel(" "),BorderLayout.NORTH);
   container.add(new JLabel(" "),BorderLayout.WEST);
   /*Declares and initializes the objects of GridLayout class.*/
   GridLayout labeltextgridlayout=new GridLayout(2, 2, 20, 0);
   /*Sets the layout of the labeltextpanel.*/
   labeltextpanel.setLayout(labeltextgridlayout);
   /*Initializes the objects of JLabel class.*/
   useridlabel=new JLabel("User ID");
   /*Adds the useridlabel to the labeltextpanel.*/
   labeltextpanel.add(useridlabel);
   /*Initializes the objects of JTextField class.*/
   useridtext=new JTextField();
   /*Sets the size of the useridtext.*/
   useridtext.setPreferredSize(new Dimension(150, 20));
   labeltextpanel.add(useridtext);
   /*Initializes the objects of JLabel class.*/
   nicknamelabel=new JLabel("User Name");
   labeltextpanel.add(nicknamelabel);
   /*Initializes the objects of JTextField class.*/
   nicknametext=new JTextField();
   /*Sets the size of the nicknametext.*/
   nicknametext.setPreferredSize(new Dimension(150, 20));
   labeltextpanel.add(nicknametext);
   /*Declares and initializes the objects of JPanel class.*/
   JPanel gridlayoutpanel=new JPanel(new GridLayout(1, 2, 0, 0));
   gridlayoutpanel.add(new JLabel(" "));
   /*initializes the objects of JPanel class.*/
   JPanel buttonpanel=new JPanel();
   buttonpanel.setPreferredSize(new Dimension(250, 30));
```

```
        addbutton=new JButton("Add");
        addbutton.addActionListener(this);
        addbutton.setPreferredSize(new Dimension(80, 25));
        addbutton.setActionCommand("add");
        cancelbutton=new JButton("Cancel");
        cancelbutton.setPreferredSize(new Dimension(80, 25));
        cancelbutton.addActionListener(this);
        cancelbutton.setActionCommand("cancel");
        buttonpanel.add(addbutton);
        buttonpanel.add(cancelbutton);
        gridlayoutpanel.add(buttonpanel);
        gridlayoutpanel.add(labeltextpanel);
        /*Declares and initializes the objects of JPanel class. */
        JPanel textareapanel=new JPanel(new GridLayout(1, 1, 20, 10));
        textareapanel.add(new JLabel("Message"));
        /*initializes the objects of JTextPane class. */
        messagetextarea=new JTextPane();
        /*initializes the objects of JScrollPane class.*/
        scpane=new JScrollPane(messagetextarea);
        /*Set the size of scpane.*/
        scpane.setPreferredSize(new Dimension(100, 30));
        textareapanel.add(scpane);
        mainpane.add(labeltextpanel);
        mainpane.add(textareapanel);
        container.add(mainpane);
        container.add(gridlayoutpanel, BorderLayout.SOUTH);
        setSize(420, 180);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String actionsource=ae.getActionCommand();
        if (actionsource.equals("add"))
        {
            String errmessage="";
            String uname=useridtext.getText().trim();
            String nname=nicknametext.getText().trim();
            if (uname.equals(""))
            {
                errmessage=" Jabber ID is a required field.\n";
            }
            else if (uname.indexOf("@")==-1)
            {
                errmessage=" Please enter a valid Jabber ID.\n";
            }
            if (nname.equals(""))
            {
                errmessage=errmessage+" User Name is a required field.";
            }
            if (!errmessage.equals(""))
            {
                JOptionPane.showMessageDialog(null,errmessage,"Error",JOptionPane.PLAIN_MESSAGE);
            }
            else
            {
                boolean checkforuser;
                checkforuser=addsocketclass.checkForRoster(uname,nname);
                if (!checkforuser)
                {
                    String msgstring="";
                    msgstring="<message type='chat' to='"+uname+"/Home'
                    from='"+addsocketclass.getUserName()+"@"+addsocketclass.getServerName()+"/Home'
                    ><body>";
                    msgstring=msgstring+messagetextarea.getText().trim();
                    msgstring=msgstring+"</body></message>";
                    addsocketclass.sendXMLToJabber(msgstring);
                    if (!addsocketclass.getUserInfo())
                    {
                        String addrosterstring="";
                        String servername=addsocketclass.getServerName();
                        nickname=nname;
                        uid=uname;
                        addrosterstring="<iq type='set'>";
                        addrosterstring=addrosterstring+"<query xmlns='jabber:iq:roster'>";
                        addrosterstring=addrosterstring+"<item jid='"+uname+"' subscription='to'
                        name='"+nname+"'/>";
                        addrosterstring=addrosterstring+"</query></iq>";
                        addsocketclass.sendXMLToJabber(addrosterstring);
                        addsocketclass.sendRosterRequest();
                        setVisible(false);
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null,"No User exist for this Jabber
                        ID.","Error",JOptionPane.PLAIN_MESSAGE);
                        addsocketclass.setUserInfo(false);
                    }
                }
```

```
            else
            {
               JOptionPane.showMessageDialog(null, "This Jabber ID or User Name has already present in
               your
               roster.", "Error", JOptionPane.PLAIN_MESSAGE);
            }
         }
      }
      else
      {
         JOptionPane.showMessageDialog(null, "errmessage", "Error", JOptionPane.PLAIN_MESSAGE);
         this.hide();
      }
}
/*
getNickName: This method is called retrieve user nick name.
Parameter: N/A
Return Value: String
*/
public String getNickName()
{
   return nickname;
}
}
```

Download this listing.

In the above code, the constructor of the AddUser class takes the object of the SocketClass class as an input parameter. This allows end users to invoke the methods of the SocketClass class.

The methods defined in Listing 7-3 are:

- `getNickName()`: Retrieves the user name of the added user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The AddUser.java file creates the window to add other end users to the contact list, as shown in Figure 7-5:

Figure 7-5: Adding End Users

PREVIOUS NEXT

Managing the Contact List

The EditDelete.java file creates the user interface to edit or delete existing end users from the contact list. Listing 7-4 shows the contents of the EditDelete.java file:

Listing 7-4: The EditDelete.java File

```
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required javax.swing.tree classes*/
import javax.swing.tree.*;
/*Imports required javax.swing.event classes*/
import javax.swing.event.*;
/*Imports required java.awt classes*/
import java.awt.*;
/*Imports required java.awt.event classes*/
import java.awt.event.*;
/* Imports required java.util classes*/
import java.util.*;
/*
class EditDelete - This class is used to create a GUI for maintaining Contact list.
Constructor:
EditDelete-This constructor creates GUI.
*/
public class  EditDelete extends JFrame implements ActionListener, MouseListener
{
/*Declares object of JPanel class.*/
JPanel combine;
/*Declares object of Container class.*/
Container container;
/*Declares object of GridBagLayout class.*/
GridBagLayout gridbaglayout;
/*Declares objects of JLabel class.*/
JLabel useridlabel = null;
JLabel nicknamelabel = null;
/*Declares objects of JTextField class.*/
JTextField useridtext = null;
JTextField nicknametext = null;
/*Declares objects of JButton class.*/
JButton editbutton = null;
JButton deletebutton = null;
JButton cancelbutton = null;
/*Declares objects of JPanel class.*/
JPanel buttonpanel = new JPanel();
/*Declares objects of JScrollPane class.*/
JScrollPane scpane = null;
JScrollPane treeView;
/*Declares object of JSplitPane class.*/
JSplitPane splitPane;
/*Declares object of Hashtable class.*/
Hashtable edithash;
/*Declares objects of String class.*/
String selecteduserid="";
String selectednickname="";
/*Declares object of JTree class.*/
JTree edittree;
/*Declares object of TreePath class.*/
TreePath path;
/*Declares object of DefaultMutableTreeNode class.*/
DefaultMutableTreeNode root ;
/*Declares object of DefaultTreeModel class.*/
DefaultTreeModel treeModel;
/*Declares objects of DynamicTree class.*/
DynamicTree tree;
DynamicTree dynamictree;
/*Declares object of SocketClass class.*/
SocketClass editsocketclass;
public EditDelete(Hashtable editdeletehash,SocketClass socketclass,DynamicTree dtree)
{
super("Edit/Delete Contact");
container = this.getContentPane();
edithash=editdeletehash;
dynamictree=dtree;
editsocketclass=socketclass;
/*Initializes the object of the JSplitPane class.*/
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
/*Initializes the object of the JPanel class.*/
combine=new JPanel(new BorderLayout());
/*Initializes the object of the DefaultMutableTreeNode class.*/
root = new  DefaultMutableTreeNode("My Contact List");
/*Initializes the object of the DynamicTree class.*/
tree=new DynamicTree(root);
edittree=tree.getTreeHandler();
/*Initializes the objects of the DefaultTreeModel class.*/
```

```
treeModel = new DefaultTreeModel(root);
/*Declares and initializes the object of the Enumeration class.*/
Enumeration editenumkey=edithash.keys();
while (editenumkey.hasMoreElements())
{
   DefaultMutableTreeNode childnode = new  DefaultMutableTreeNode(editenumkey.nextElement());
   tree.addObject(childnode);
}
edittree.addMouseListener(this);
splitPane.setDividerLocation(180);
treeView = new JScrollPane(tree);
splitPane.setTopComponent(treeView);
splitPane.setBottomComponent(combine);
/*Declares and initializes the object of the JPanel class.*/
JPanel jpanel = new JPanel();
/*Declares and initializes the object of the JLabel class.*/
JLabel labelheading =new JLabel("Edit/Delete Contact");
/*Sets the font of the labelheading.*/
labelheading.setFont(new Font("Verdana", Font.BOLD,12));
/*Adds labelheading to the jpanel*/
jpanel.add(labelheading);
/*Adds jpanel to the combine*/
combine.add(jpanel,BorderLayout.NORTH);
gridbaglayout=new GridBagLayout();
/*Declare and initializes the objects of the JPanel class.*/
JPanel userinfopanel = new JPanel(gridbaglayout);
/*Declare and initializes the objects of the GridBagConstraints class.*/
GridBagConstraints gridbagconstraints=new GridBagConstraints();
useridlabel = new JLabel("User Id");
nicknamelabel = new JLabel("User Name");
/*Initializes the objects of the JTextField class.*/
useridtext = new JTextField();
nicknametext = new JTextField();
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(useridlabel, gridbagconstraints);
/*Sets the size of the useridlabel.*/
useridlabel.setPreferredSize(new Dimension(80, 28));
/*Adds useridlabel to the userinfopanel.*/
userinfopanel.add(useridlabel);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(useridtext, gridbagconstraints);
/*Sets the size of the useridtext.*/
useridtext.setPreferredSize(new Dimension(80, 28));
/*Adds useridtext to the userinfopanel.*/
userinfopanel.add(useridtext);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=1;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(nicknamelabel,gridbagconstraints);
/*Sets the size of the nicknamelabel.*/
nicknamelabel.setPreferredSize(new Dimension(120, 28));
/*Adds nicknamelabel to the userinfopanel.*/
userinfopanel.add(nicknamelabel);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=1;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(nicknametext,gridbagconstraints);
/*Sets the size of the nicknametext.*/
nicknametext.setPreferredSize(new Dimension(120, 28));
/*Adds nicknametext to the userinfopanel.*/
userinfopanel.add(nicknametext);
/*Declare and initializes the objects of the JLabel class. */
```

```
JLabel blanklabel=new JLabel(" ");
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=2;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(blanklabel,gridbagconstraints);
/*Sets the size of the blanklabel.*/
blanklabel.setPreferredSize(new Dimension(120, 28));
/*Adds blanklabel to the userinfopanel.*/
userinfopanel.add(blanklabel);
/*Declare and initializes the objects of the JLabel class.*/
JLabel secondblanklabel=new JLabel(" ");
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=3;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(secondblanklabel,gridbagconstraints);
/*Sets the size of the secondblanklabel.*/
secondblanklabel.setPreferredSize(new Dimension(120, 28));
/*Adds secondblanklabel to the userinfopanel.*/
userinfopanel.add(secondblanklabel);
/*Declare and initializes the objects of the JLabel class.*/
JLabel thirdblanklabel=new JLabel(" ");
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=3;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(thirdblanklabel,gridbagconstraints);
thirdblanklabel.setPreferredSize(new Dimension(120, 28));
userinfopanel.add(thirdblanklabel);
/*Initializes the objects of the JButton class.*/
editbutton = new JButton("Update");
deletebutton = new JButton("Delete");
cancelbutton = new JButton("Cancel");
/*Adds the action listener to the objects of JButton class.*/
editbutton.addActionListener(this);
deletebutton.addActionListener(this);
cancelbutton.addActionListener(this);
/*Sets the action command to the objects of JButton class.*/
editbutton.setActionCommand("edit");
deletebutton.setActionCommand("delete");
cancelbutton.setActionCommand("cancel");
/*Sets the size of the objects of JButton class*/
editbutton.setPreferredSize(new Dimension(60, 22));
deletebutton.setPreferredSize(new Dimension(70, 22));
cancelbutton.setPreferredSize(new Dimension(70, 22));
/*Adds objects of JButton class to the buttonpanel.*/
buttonpanel.add(editbutton);
buttonpanel.add(deletebutton);
buttonpanel.add(cancelbutton);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=5;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(buttonpanel, gridbagconstraints);
userinfopanel.add(buttonpanel);
combine.add(userinfopanel);
container.add(splitPane);
setSize(495,230);
setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
   String actioncommand=ae.getActionCommand();
   /* This block will be executed when the end user clicks edit button.*/
   if (actioncommand.equals("edit"))
   {
      if ((!useridtext.getText().trim().equals(""))&&
      (!nicknametext.getText().trim().equals("")))
      {
         if ((!useridtext.getText().trim().equals(selecteduserid))&&
         (!nicknametext.getText().trim().equals(selectednickname)))
         {
            String editstring="<iq type='set' id='1'>";
```

```
            editstring=editstring+"<query xmlns='jabber:iq:roster'>";
            editstring=editstring+"<item jid="+selecteduserid+" name='"+selectednickname+"'
            subscription='remove'/>";
            editstring=editstring+"</query></iq>";
            editsocketclass.sendXMLToJabber(editstring);
            editstring="<iq type='set' >";
            editstring=editstring+"<query xmlns='jabber:iq:roster'>";
            editstring=editstring+"<item jid='"+useridtext.getText().trim()+"' subscription='to'
            name='"+nicknametext.getText().trim()+"' />";
            editstring=editstring+"</query></iq>";
            editsocketclass.sendXMLToJabber(editstring);
            edithash.remove(selectednickname);
            String jabbid="'"+useridtext.getText().trim()+"'";
            edithash.put(nicknametext.getText().trim(),jabbid);
            DefaultTreeModel model = (DefaultTreeModel)edittree.getModel();
            MutableTreeNode node = (MutableTreeNode)path.getLastPathComponent();
            dynamictree.clear();
            tree.clear();
            Enumeration enum=edithash.keys();
            while (enum.hasMoreElements())
            {
               String keyval=(String)enum.nextElement();
               DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(keyval);
               DefaultMutableTreeNode childnode = new  DefaultMutableTreeNode(keyval);
               dynamictree.addObject(treenode);
               tree.addObject(childnode);
            }
         }
      }
   }
   /* This block will be executed when the end user clicks delete button.*/
   else if (actioncommand.equals("delete"))
   {
      String editstring="<iq type='set' id='1'>";
      editstring=editstring+"<query xmlns='jabber:iq:roster'>";
      editstring=editstring+"<item jid="+selecteduserid+" name='"+selectednickname+"'
      subscription='remove'/>";
      editstring=editstring+"</query></iq>";
      editsocketclass.sendXMLToJabber(editstring);
      DefaultTreeModel model = (DefaultTreeModel)edittree.getModel();
      MutableTreeNode node = (MutableTreeNode)path.getLastPathComponent();
      model.removeNodeFromParent(node);
      useridtext.setText("");
      nicknametext.setText("");
      edithash.remove(selectednickname);
      dynamictree.clear();
      Enumeration enum=edithash.keys();
      while (enum.hasMoreElements())
      {
         DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(enum.nextElement());
         dynamictree.addObject(treenode);
      }
   }
   else
   {
   setVisible(false);
   }
}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e)
{
   int selRow = edittree.getRowForLocation(e.getX(), e.getY());
   TreePath selPath = edittree.getPathForLocation(e.getX(), e.getY());
   path=edittree.getSelectionPath();
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)edittree.getLastSelectedPathComponent();
   if(!(node.toString().equals("My Contact List")))
   {
      selecteduserid=(String)edithash.get(node.toString());
      selectednickname=node.toString();
      useridtext.setText(selecteduserid.substring(1,selecteduserid.length()-1));
      nicknametext.setText(selectednickname);
   }
}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
}
```

Download this listing.

In the above code, the constructor of the EditDelete class takes the object of the SocketClass class, DynamicTree class, and HashTable class as input parameters. The EditDelete class allows end users to invoke the methods of the SocketClass class, DynamicTree class, and HashTable class. The EditDelete class defines the actionPerformed() method, which acts as an event listener and activates an appropriate method based on the action an end user performs.

The EditDelete.java file creates the window to edit or delete the existing end users, as shown in Figure 7-6:

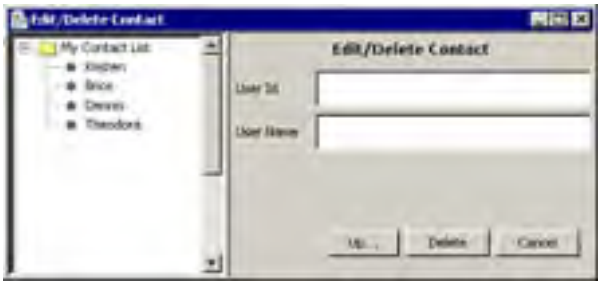


Figure 7-6: Managing the Contact List

PREVIOUS NEXT

Creating the Chat Window

The ChatWindow.java file creates a Chat window to send messages to a particular end user selected in the contact list. Listing 7-5 shows the contents of the ChatWindow.java file:

Listing 7-5: The ChatWindow.java File

```
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required java.awt classes*/
import java.awt.*;
/*Imports required javax.swing.event classes*/
import javax.swing.event.*;
/*Imports required java.awt.event classes*/
import java.awt.event.*;
/*Imports required javax.swing.text package classes*/
import javax.swing.text.*;
/*
class ChatWindow - This class is used to create GUI for chatting.
Constructor:
ChatWindow-This constructor creates GUI.
Methods:
writemessage: This method is called to send input message to insertTextInToTextPane function.
eraseText: This method is called to send input message to insertTextInToTextPane function.
addToRosterTree: This method is called to add user name in to tree.
*/
public class  ChatWindow extends JFrame implements ActionListener
{
/*Declares object of Container class.*/
Container container;
/*Declares object of GridBagLayout class.*/
GridBagLayout gridbaglayout;
/*Declares object of JLabel class.*/
JLabel fromuser = null;
JLabel touser = null;
JLabel fromusername = null;
JLabel tousername = null;
/*Declares object of JTextPane class.*/
JTextPane receivemessagetextarea = null;
/*Declares object of JTextField class.*/
JTextField sendmessagetext = null;
/*Declares object of JButton class.*/
JButton sendbutton = null;
/*Declares object of JButton class.*/
JScrollPane scpane;
/*Declares object of String class.*/
String username;
String usernickname;
String sendername;
/*Declares object of Document class.*/
Document contentmodel;
/*Declares object of MutableAttributeSet class.*/
MutableAttributeSet recervernameatrib, sendermessagattrib;
/*Declares object of SocketClass class.*/
SocketClass socketchatclass;
public ChatWindow(String nickname,String recevername, SocketClass socketclass)
{
   sendername=socketclass.getUserName();
   /*Initializes the objects of JLabel class.*/
   username=recevername;
   usernickname=nickname;
   socketchatclass=socketclass;
   fromuser = new JLabel("From :");
   touser = new JLabel("To :");
   fromusername=new JLabel(socketclass.getUserName());
   tousername = new JLabel(usernickname);
   /*Sets window title.*/
   setTitle("Chat Window: "+usernickname);
   container = this.getContentPane();
   /*Initializes the objects of GridBagLayout class.*/
   gridbaglayout=new GridBagLayout();
   /*Declare and initializes the objects of GridBagConstraints class.*/
   GridBagConstraints gridbagconstraints=new GridBagConstraints();
   /*Declare and initializes the objects of JPanel class.*/
   JPanel labelpanel = new JPanel(gridbaglayout);
   /*Sets constraints for gridbagconstraints.*/
   gridbagconstraints.fill=GridBagConstraints.BOTH;
   gridbagconstraints.insets=new Insets(5, 5, 0, 0);
   gridbagconstraints.gridx=0;
   gridbagconstraints.gridy=0;
   gridbagconstraints.weightx=1;
   gridbagconstraints.weighty=1;
   gridbagconstraints.anchor=GridBagConstraints.WEST;
   gridbaglayout.setConstraints(fromuser, gridbagconstraints);
```

```
/*Sets fromuser size.*/
fromuser.setPreferredSize(new Dimension(140, 28));
/*Adds label to labelpanel.*/
labelpanel.add(fromuser);
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 20);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.EAST;
gridbaglayout.setConstraints(fromusername, gridbagconstraints);
/*Sets fromusername size.*/
fromusername.setPreferredSize(new Dimension(20, 28));
/*Sets fromusername font.*/
fromusername.setFont(new Font("Verdana", Font.BOLD, 10));
/*Adds fromusername to labelpanel.*/
labelpanel.add(fromusername);
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=2;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.EAST;
gridbaglayout.setConstraints(touser, gridbagconstraints);
/*Sets touser size.*/
touser.setPreferredSize(new Dimension(140, 28));
/*Adds touser to labelpanel.*/
labelpanel.add(touser);
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=3;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.WEST;
gridbaglayout.setConstraints(tousername, gridbagconstraints);
/*Sets tousername size.*/
tousername.setPreferredSize(new Dimension(20, 28));
/*Sets tousername font.*/
tousername.setFont(new Font("Verdana", Font.BOLD, 10));
/*Adds tousername to labelpanel.*/
labelpanel.add(tousername);
/*Adds labelpanel to container.*/
container.add(labelpanel, BorderLayout.NORTH);
/*Initializes the objects of JTextPane class.*/
receivemessagetextarea = new JTextPane();
contentmodel=receivemessagetextarea.getDocument();
/*Initializes the objects of SimpleAttributeSet class.*/
recervernameatrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(recervernameatrib,"Verdana");
StyleConstants.setForeground(recervernameatrib, Color.black);
sendermessagattrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(sendermessagattrib,"Verdana");
StyleConstants.setForeground(sendermessagattrib,Color.red);
receivemessagetextarea.setPreferredSize(new Dimension(80, 20));
receivemessagetextarea.setEditable(false);
/*Initializes the objects of JScrollPane class.*/
scpane  = new JScrollPane(receivemessagetextarea, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
/*Adds scpane to container.*/
container.add(scpane,BorderLayout.CENTER);
/*Declare and initializes the objects of JPanel class.*/
JPanel sendmessagepanel = new JPanel(gridbaglayout);
/*Initializes the objects of JTextField class.*/
sendmessagetext = new JTextField();
/*Initializes the objects of JButton class.*/
sendbutton = new JButton("Send");
/*Adds action listener to sendbutton.*/
sendbutton.addActionListener(this);
/*Sets action command of sendbutton.*/
sendbutton.setActionCommand("send");
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(sendmessagetext, gridbagconstraints);
/*Sets tousername size.*/
sendmessagetext.setPreferredSize(new Dimension(120, 28));
/*Adds sendmessagetext to the sendmessagepanel*/
```

```
   sendmessagepanel.add(sendmessagetext);
   /*Sets constraints for gridbagconstraints.*/
   gridbagconstraints.fill=GridBagConstraints.BOTH;
   gridbagconstraints.insets=new Insets(5, 5, 0, 0);
   gridbagconstraints.gridx=1;
   gridbagconstraints.gridy=0;
   gridbagconstraints.weightx=0;
   gridbagconstraints.weighty=0;
   gridbagconstraints.anchor=GridBagConstraints.EAST;
   gridbaglayout.setConstraints(sendbutton, gridbagconstraints);
   /*Sets sendbutton size.*/
   sendbutton.setPreferredSize(new Dimension(60, 28));
   sendmessagepanel.add(sendbutton);
   container.add(sendmessagepanel, BorderLayout.SOUTH);
   setBounds(100, 100, 350, 280);
   show();
}
public void actionPerformed(ActionEvent ae)
{
   String actioncommand=ae.getActionCommand();
   String messagestring;
   if (actioncommand.equals("send"))
   {
      if (!sendmessagetext.getText().trim().equals(""))
      {
         String recevername=username.substring(1,username.length()-1);
         messagestring="<message type='chat' to='"+recevername+"/Home'
         from='"+sendername+"@"+socketchatclass.getServerName()+"/Home'>";
         messagestring=messagestring+"<body>";
         messagestring=messagestring+sendmessagetext.getText().trim();
         messagestring=messagestring+"</body></message>";
         try
         {
            contentmodel.insertString(contentmodel.getLength(),"\n"+sendername+":
            ",sendermessagattrib);
            contentmodel.insertString(contentmodel.getLength(), sendmessagetext.getText().trim(),
            recervernameatrib);
         }
         catch(BadLocationException ble)
         {
         }
         socketchatclass.sendXMLToJabber(messagestring);
         sendmessagetext.setText("");
         }
      }
   }
   /*
insertTextInToTextPane: This method is called to insert text message into textpane.
Parameter: pmessage - Object of  String class, psendername - Object of  String class.
Return Value: N/A
*/
public void insertTextInToTextPane(String pmessage,String psendername)
{
   try
   {
   contentmodel.insertString(contentmodel.getLength(), "\n"+usernickname+":
   "+pmessage,recervernameatrib);
   }
   catch(BadLocationException ble) {}
   }
}
```

Download this listing.

In the above code, the constructor of the ChatWindow class takes an object of the SocketClass class and two strings, userName and receivername as input parameters. The object of the SocketClass class allows end users to invoke the methods of the SocketClass class. The userName string retrieves the user name of the selected end user, and the receivername string retrieves the user ID of the selected end user.

The methods defined in Listing 7-5 are:

- `insertTextInToTextPane()`: Inserts the text messages specified by end users in the text area of the Chat window.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The ChatWindow.java file creates the Chat window to send messages to the selected end user, as shown in Figure 7-7:

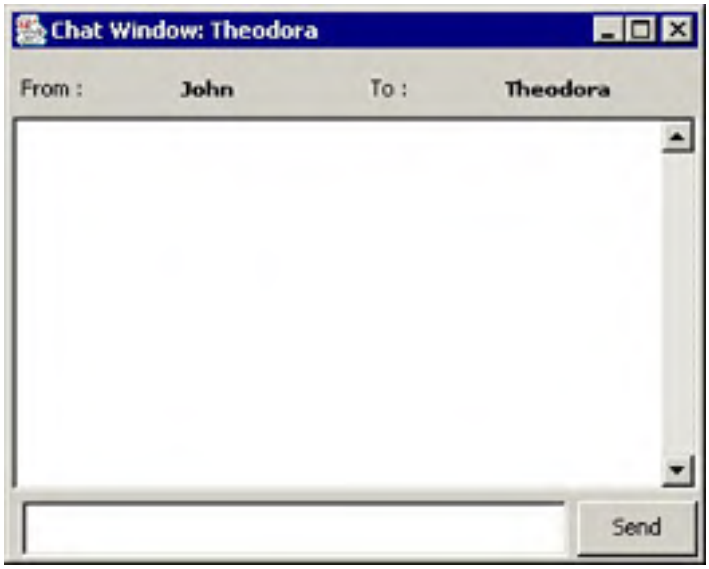


Figure 7-7: The Chat Window

PREVIOUS NEXT

Sending and Receiving Messages

The SocketClass.java file opens the socket to send and receive messages between end users with the help of a Jabber server. Listing 7-6 shows the contents of the SocketClass.java file:

Listing 7-6: The SocketClass.java File

```
/*Imports required java.util classes*/
import java.util.*;
/*Imports required java.io classes*/
import java.io.*;
/*Imports required java.net classes*/
import java.net.*;
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required DefaultMutableTreeNode class*/
import javax.swing.tree.DefaultMutableTreeNode;
/*
class SocketClass - This class is used for reading the input messages and writing output messages.
Methods:
isConnected: This method returns a boolean value.
isWaitingForAuth: This method returns a boolean value.
socketoOpenClass: This method returns a boolean value.
setUserLoginHandler: This method sets handler to the object of UserLogin class.
checkForRoster: This method calls another method of ContactList class.
getUserInfo: This method returns a boolean value.
setUserInfo: This method sets value of nouserexists variable.
sendRosterRequest: This method creates a Jabber message string for Roster and passes this string int
sendXMLToJabber method.
getUserName: This method returns user name.
getServerName: This method returns the server name.
setWaitForAuth: This method sets a boolean value for waitforauth.
sendXMLToJabber: This method sends XML message string for the jabber server.
checkForError: This method checks whether input message contains a error code or not.
sendername: This method extracts sender name from input message.
setUserNameAndPassword: This method stores the user name and password of the logged in end user.
sendSessionStartMessage: This method creates a Jabber message string for starting a jabber session a
passes this string into sendXMLToJabber method.
sendAuthorized: This method creates a Jabber message string for user authentication and passes this
string into sendXMLToJabber method.
sendRegistration: This method creates a Jabber message string for user registration and passes this
string into sendXMLToJabber method.
openPort: This method creates an object of SocketOpener class and class openSocket method of  this
class.
*/
class SocketClass implements Runnable
{
   boolean isConnected=false;
   /*Declares the object of Vector class.*/
   Vector tagvactor;
   /*Declares the object of PrintWriter class.*/
   PrintWriter out = null;
   /*Declares the object of BufferedReader class.*/
   BufferedReader in = null;
   /*Declares and initializes the object of Vector class.*/
   String servername;
   int portno;
   int wait;
   Socket clientsocket;
   /*Declares the object of Thread class.*/
   Thread inputmessagethread;
   /*Declares the object of String class.*/
   String errortype;
   String resource="Home";
   String username="";
   String password="";
   String auth="";
   String sendername="";
   String recevername="";
   String vcardString="";
   boolean waitforreg=false;
   boolean waitforauth=false;
   boolean waitForResult=false;
   boolean rosterflag=false;
   boolean nouserexists=false;
   boolean endofvcard=false;
   /*Declares the object of UserLogin class.*/
   UserLogin userloginhandler;
   /*Declares the object of ContactList class.*/
   ContactList contactlist;
   /*Declares the object of UserInfo class.*/
   UserInfo ui;
   public SocketClass() {}
   /*
```

```
isConnected: This method returns a boolean value.
Parameter: N/A
Return Value: boolean
*/
public boolean isConnected()
{
   return isConnected;
}
/*
isWaitingForAuth: This method returns a boolean value.
Parameter: N/A
Return Value: boolean
*/
public boolean isWaitingForAuth()
{
   return waitforauth;
}
/*
socketoOpenClass: This method returns a boolean value
Parameter: ipaddress - object of String class, portno, wait
Return Value: boolean
*/
public void socketoOpenClass(String ipaddress, int portno, int wait)
{
   this.servername=ipaddress;
   this.portno=portno;
   this.wait=wait;
   openPort(servername,portno,wait);
   if (clientsocket!=null)
   {
      isConnected=true;
      try
      {
         /*Initializes the object of PrintWriter class.*/
         out = new PrintWriter(clientsocket.getOutputStream(), true);
         /*Initializes the object of BufferedReader class.*/
         in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      /*Initializes the object of Thread class.*/
      inputmessagethread=new Thread(this);
      /*Calls start method of inputmessagethread.*/
      inputmessagethread.start();
      }
   }
   /*
   setUserLoginHandler: This method sets handler to the object of UserLogin class.
   Parameter: userlogin - object of UserLogin class
   Return Value: N/A
   */
   public void setUserLoginHandler(UserLogin userlogin)
   {
      userloginhandler=userlogin;
   }
   public void run()
   {
      int i=0;
      String errororwarning="";
      String errorstring="";
      String inputstring="";
      boolean starttag=false;
      boolean endtag=false;
      boolean startwritting=false;
      String messagebody="";
      String tagentity="";
      Vector tagvactor;
      tagvactor=new Vector();
      int vactorindex=0;
      int starttagsing=0;
      while (true)
      {
         try
         {
            i=in.read();
            if (i==-1)
            {
               break;
            }
            else
            {
            inputstring=inputstring+(char)i;
            if (endofvcard)
            {
               vcardString=vcardString+(char)i;
            }
```

```
if ((char)i=='<')
{
   starttag=true;
   starttagsing=1;
   tagentity="";
}
else
{
   if ((char)i=='/' && starttag==true )
   {
      if (starttagsing==1)
      {
         starttag=false;
         endtag=true;
         if (!messagebody.trim().equals("")&&(!nouserexists))
         {
            contactlist.writemessage(messagebody,sendername,recevername);
         }
         startwritting=false;
         messagebody="";
         vactorindex=vactorindex-1;
         if (vactorindex>=0)
         tagvactor.removeElementAt(vactorindex);
      }
      tagentity=tagentity+(char)i;
   }
   else
   {
   starttagsing=0;
   if((char)i=='>')
   {
      if (starttag)
      {
         if (tagentity.endsWith("/"))
         {
            tagentity=tagentity.substring(0,tagentity.length()-1);
         }
         if (tagentity.equals("PREF"))
         {
            String
            nodename=((DefaultMutableTreeNode)contactlist.rostertree.
            getLastSelectedPathComponent()).
            toString();
            String uid=(String)contactlist.hashnode.get(nodename);
            ui.setInfo(vcardString,nodename,uid);
            vcardString="";
         }
         sendername(tagentity);
         errorstring=checkForError(tagentity);
         if (errorstring.equals("unauthorized"))
         {
            sendRegistration();
            waitforreg=true;
            waitforauth=false;
         }
         if ((tagentity.indexOf("type='result'")>1)&&(waitforreg)&&
         (!errorstring.equals
         ("user exist")))
         {
            sendAuthorized();
            waitforauth=true;
            waitforreg=false;
         }
         if ((tagentity.indexOf("type='result'")>1)&&(waitforauth))
         {
            waitforauth=false;
         }
         if (tagentity.indexOf("item-not-found
         xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'")!=-1)
         {
            nouserexists=true;
            contactlist.removeFromRosterTree(sendername);
         }
         if (tagentity.indexOf("remote-server-not-found")!=-1)
         {
            nouserexists=true;
            contactlist.removeFromRosterTree(sendername);
         }
         if (rosterflag)
         {
            if (tagentity.indexOf("item")!=-1&&(!nouserexists))
            {
               String[] splitedsendername=sendername.split("/");
               String contactpersonname=tagentity.substring(tagentity.indexOf("name")
               tagentity.indexOf("jid")).trim();
               String
               jid=tagentity.substring(tagentity.indexOf("jid")+4,tagentity.length())
               if (!jid.equals("'"+splitedsendername[0]+"'"))
```

```
                {
                   contactlist.addToRosterTree(jid.trim(), contactpersonname.trim());
                }
             }
             else
             {
                if (nouserexists)
                {
                   contactlist.eraseText(sendername);
                   JOptionPane.showMessageDialog(null, "No User exists with this jabbe
                   ID.", "Error",JOptionPane.PLAIN_MESSAGE);
                   nouserexists=false;
                }
             }
             if (tagentity.indexOf("query")!=-1)
             {
                rosterflag=false;
             }
          }
          if (tagentity.indexOf("xmlns='vcard-temp'")>1&&(!endofvcard))
          {
             endofvcard=true;
          }
          if
          ((tagentity.indexOf("type='result'")>1)&&
          (!waitforauth)&&(!waitforreg))
          {
             waitForResult=true;
          }
          else
          {
             if (tagentity.indexOf("xmlns='jabber:iq:roster'")
             >1&&waitForResult)
             {
                rosterflag=true;
                waitForResult=false;
             }
             waitForResult=false;
          }
          if ((tagentity.indexOf("type='result'")>1)&&(!waitforauth))
          {
             if (contactlist==null)
             {
                contactlist=new ContactList(this);
                sendRosterRequest();
                userloginhandler.setVisible(false);
             }
          }
          if (errorstring.equals("user exixts")&&waitforreg)
          {
             JOptionPane.showMessageDialog(null, "A user with this user ID is already
             exists.Please enter
             other
             user ID.", errororwarning, JOptionPane.PLAIN_MESSAGE);
          }
          if (!errorstring.equals(""))
          {
             if (errortype=="major")
             {
                errororwarning="Error";
                JOptionPane.showMessageDialog(null,errorstring, errororwarning,
                JOptionPane.PLAIN_MESSAGE);
             }
             else
             {
                errororwarning="Warning";
                JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                JOptionPane.PLAIN_MESSAGE);
             }
          }
          startwritting=true;
          tagvactor.insertElementAt(tagentity,vactorindex);
          vactorindex=vactorindex+1;
          starttag=false;
          }
       }
       else
       {
          if (startwritting==true&&tagentity.trim().equals("body"))
          {
             messagebody=messagebody+(char)i;
          }
          if (starttag)
          {
             tagentity=tagentity+(char)i;
          }
       }
    }
```

```
                }
            }
         }
         catch(IOException ie)
         {
         }
   }
   isConnected=false;
}
/*
checkForRoster: This method calls another method of ContactList class.
Parameter: uname - object of String class,nname - object of String class
Return Value: boolean
*/
public boolean checkForRoster(String uname,String nname)
{
   return contactlist.checkHash(uname,nname);
}
/*
getUserInfo: This method returns a boolean value.
Parameter: N/A
Return Value: boolean
*/
public boolean getUserInfo()
{
   return nouserexists;
}
/*
setUserInfo: This method sets value of nouserexists variable.
Parameter: flag
Return Value: N/A
*/
public void setUserInfo(boolean flag)
{
   nouserexists=flag;
}
/*
sendRosterRequest: This method creates a Jabber message string  for Roster and passes this string in
sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendRosterRequest()
{
   String rosterstring="";
   rosterstring="<iq type=\"get\" id=\"1\">";
   rosterstring=rosterstring+"<query xmlns=\"jabber:iq:roster\"/></iq>";
   sendXMLToJabber(rosterstring);
}
/*
getUserName: This method returns user name.
Parameter: N/A
Return Value: String
*/
public String getUserName()
{
   return username;
}
/*
getServerName: This method returns the server name.
Parameter: N/A
Return Value: String
*/
public String getServerName()
{
   return servername;
}
/*
setWaitForAuth: This method sets a boolean value for waitforauth.
Parameter: flag - boolean
Return Value: N/A
*/
public void setWaitForAuth(boolean flag)
{
   waitforauth=flag;
}
/*
sendXMLToJabber: This method sends XML message string for the jabber server.
Parameter: outputmessage - object of String class
Return Value: N/A
*/
public void sendXMLToJabber(String outputmessage)
{
   if (outputmessage.indexOf("<vCard")!=-1)
   {
      ui=new UserInfo();
   }
   String[] tokenizerstring=outputmessage.split("\n");
```

```
   for (int i=0;i<tokenizerstring.length;i++ )
   {
      try
      {
         out.println(tokenizerstring[i]);
         out.flush();
      }catch(Exception e)
      {
         return;
      }
      out.flush();
   }
}
/*
checkForError: This method checks whether input message contains a error code or not.
Parameter: tagentity - object of String class
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if (tagentity.equals("error code='403' type='auth'"))
   {
      sendRegistration();
   }
   if ((tagentity.indexOf("code='401'")!=-1)||(tagentity.indexOf("code=\"401\"")!=-1))
   {
      return "unauthorized";
   }
   if ((tagentity.indexOf("code='409'")!=-1)||(tagentity.indexOf("code=\"409\"")!=-1))
   {
      return "user exixts";
   }
   return error;
}
/*
sendername: This method extracts sender name from input message.
Parameter: tagentity - object of String class
Return Value: N/A
*/
public void sendername(String tagentity)
{
   if (tagentity.startsWith("message"))
   {
      sendername="";
      recevername="";
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-2);
      }
      if (tagentity.indexOf("to=")>0&&tagentity.indexOf("type=")>0)
      {
         recevername=tagentity.substring(tagentity.indexOf("to=")+4,tagentity.indexOf("type=")-2);
      }
   }
}
public void setUserNameAndPassword(String username,String password)
{
   this.username=username;
   this.password=password;
}
/*
sendSessionStartMessage: This method creates a Jabber message string for starting a jabber session a
passes this string into sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendSessionStartMessage()
{
   String sessionStratString;
   sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>";
   sessionStratString=sessionStratString+"<stream:stream";
   sessionStratString=sessionStratString+" to= \""+servername +"\"";
   sessionStratString=sessionStratString+" xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   sendXMLToJabber(sessionStratString);
}
/*
sendAuthorized: This method creates a Jabber message string for user authentication and passes this
string into sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendAuthorized()
{
   String authentication;
```

```
   authentication="<iq type=\"set\" id=\"1\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+"<username>"+username+"</username>";
   authentication=authentication+"<password>"+password+"</password>";
   authentication=authentication+"<resource>"+resource+"</resource>";
   authentication=authentication+"</query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
}
/*
sendRegistration: This method creates a Jabber message string for user registration and passes this
string into sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendRegistration()
{
   String registrationstring;
   registrationstring="<iq type=\"set\" to=\""+username+"."+servername+"\" id=\"1\">";
   registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
   registrationstring=registrationstring+"<username>"+username+"</username>";
   registrationstring=registrationstring+"<password>"+password+"</password></
   query></iq>";
   sendXMLToJabber(registrationstring);
}
/*
openPort: This method creates an object of SocketOpener class and class openSocket method of this
class.
Parameter: ipaddress - Object of String class, portno - int, timeinsec - int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno, int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      SocketOpener socketopener=new SocketOpener();
      clientsocket=socketopener.openSocket(ipaddress, portno, timeinsec);
   }
}
}
/*
class SocketOpener - This class is used to create an object of socket class.
Constructor:
openSocket
Methods:
getSocket: This method returns a handler of the object of socket class.
*/
class SocketOpener
{
   private Socket socket;
   public Socket openSocket(String hostip, int portnumber, int timeinsec)
   {
      try
      {
         socket=new Socket(hostip,portnumber);
      }
      catch(IOException ie)
      {
         System.out.println("Exception occurred while creating a socket : " +ie);
      }
      return getSocket();
   }
   public SocketOpener()
   {
   }
   /*
   getSocket: This method returns a handler of the object of socket class.
   Parameter: N/A
   Return Value: Socket
   */
   public Socket getSocket()
   {
      return socket;
   }
}
```

Download this listing.

In the above code, the constructor of the SocketClass class creates a new instance of the SocketClass class. The object of the SocketClass class allows end users to invoke methods of the SocketClass class.

The methods defined in Listing 7-6 are:

- `isConnected()`: Checks whether or not an end user is connected to the Jabber server.
- `isWaitingForAuth()`: Checks whether or not an end user is authenticated with the Jabber server.
- `socketoOpenClass()`: Checks whether or not a socket is open.
- `setUserLoginHandler()`: Sets the handler to the object of the UserLogin class.
- `checkForRoster()`: Calls the checkHash() method of the ContactList class.
- `getUserInfo`: Checks whether or not information about an end user to be added to the contact list exists on the Jabber server.
- `setUserInfo()`: Sets the status as true if the specified end user information does not exist on the Jabber server.
- `sendRosterRequest`: Creates a message string and passes this message string to the sendXMLToJabber() method.
- `getUserName()`: Retrieves the name of an end user.
- `getServerName()`: Retrieves the name of the Jabber server.
- `setWaitForAuth()`: Sets the status as true if an end user is authenticated by the Jabber server.
- `sendXMLToJabber()`: Sends the message string to the Jabber server in XML format.
- `checkForError()`: Checks whether or not the input string received from the Jabber server contains an error message.
- `sendername()`: Retrieves the sender's name from the input message received from the Jabber server.
- setUserNameAndPassword(): Stores the user name and password of the logged on end user.
- `sendSessionStartMessage()`: Creates a message string in the XML format for starting a Jabber session and passes this message string to the sendXMLToJabber() method.
- `sendAuthorized()`: Creates a message string in XML format for user authentication and passes this message string to the sendXMLToJabber() method.
- `sendRegistration()`: Creates a message string in XML format for user registration and passes this message string to the sendXMLToJabber() method.
- `openPort`: Opens a client socket by using the IP address and port number of the Jabber server.
- `run()`: Listens for the response sent by the Jabber server.

PREVIOUS NEXT

Creating a Tree Structure of the Contact List

The DynamicTree.java file creates and maintains a tree structure of the contact list. Listing 7-7 shows the contents of the DynmaicTree.java file:

Listing 7-7: The DynamicTree.java File

```
/*Imports required GridLayout class*/
import java.awt.GridLayout;
/*Imports required Toolkit class*/
import java.awt.Toolkit;
/*Imports required JPanel class*/
import javax.swing.JPanel;
/*Imports required JScrollPane class*/
import javax.swing.JScrollPane;
/*Imports required JTree class*/
import javax.swing.JTree;
/*Imports required DefaultMutableTreeNode class*/
import javax.swing.tree.DefaultMutableTreeNode;
/* mports required DefaultTreeModel class*/
import javax.swing.tree.DefaultTreeModel;
/*Imports required MutableTreeNode class*/
import javax.swing.tree.MutableTreeNode;
/*Imports required TreePath class*/
import javax.swing.tree.TreePath;
/*Imports required TreeSelectionModel class*/
import javax.swing.tree.TreeSelectionModel;
/*Imports required TreeModelEvent class*/
import javax.swing.event.TreeModelEvent;
/*Imports required TreeModelListener class*/
import javax.swing.event.TreeModelListener;
/*
class DynamicTree - This class is used to create a tree.
Constructor:
DynamicTree-This constructor creates GUI.
Methods:
clear: This method is called remove all node from tree.
getTreeHandler: This method is called to get handler of the object of JTree class.
removeCurrentNode: This method is called to remove selected node from tree.
addObject: This method is called to add node into the tree.
*/
public class DynamicTree extends JPanel
{
   /*Declares object of DefaultMutableTreeNode class.*/
   DefaultMutableTreeNode rootNode;
   /*Declares object of DefaultTreeModel class.*/
   DefaultTreeModel treeModel;
   /*Declares object of JTree class.*/
   JTree tree;
   /*Declares and initializes object of Toolkit class.*/
   Toolkit toolkit = Toolkit.getDefaultToolkit();
   public DynamicTree(DefaultMutableTreeNode node)
   {
      super(new GridLayout(1, 0));
      /*initializes the object of DefaultMutableTreeNode class.*/
      rootNode = new DefaultMutableTreeNode("My Contact List");
      /*initializes the object of DefaultTreeModel class.*/
      treeModel = new DefaultTreeModel(rootNode);
      treeModel.addTreeModelListener(new RosterTreeModelListener());
      /*initializes the object of JTree class.*/
      tree = new JTree(treeModel);
      tree.setEditable(true);
      tree.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
      tree.setShowsRootHandles(true);
      /*Declares and initializes object of JScrollPane class.*/
      JScrollPane scrollPane = new JScrollPane(tree);
      add(scrollPane);
   }
   /*
   clear: This method is called remove all node from tree.
   Parameter: N/A
   Return Value: N/A
   */
   public void clear()
   {
      rootNode.removeAllChildren();
      treeModel.reload();
   }
   /*
   getTreeHandler: This method is called to get handler of the object of JTree class.
   Parameter: N/A
   Return Value: JTree
   */
```

```
public JTree getTreeHandler()
{
   return tree;
}
/*
removeCurrentNode: This method is called to remove selected node from tree.
Parameter: N/A
Return Value: N/A
*/
public void removeCurrentNode()
{
TreePath currentSelection = tree.getSelectionPath();
if (currentSelection != null)
{
   DefaultMutableTreeNode currentNode = (DefaultMutableTreeNode)
   (currentSelection.getLastPathComponent());
   MutableTreeNode parent = (MutableTreeNode)(currentNode.getParent());
   if (parent != null) {
      treeModel.removeNodeFromParent(currentNode);
      return;
      }
   }
   toolkit.beep();
}
/*
addObject: This method is called to add node into the tree.
Parameter: child - Object of Object class.
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(Object child)
{
   DefaultMutableTreeNode parentNode = null;
   TreePath parentPath = tree.getSelectionPath();
   if (parentPath == null)
   {
      parentNode = rootNode;
   }
   else
   {
      parentNode = (DefaultMutableTreeNode)(parentPath.getLastPathComponent());
   }
   return addObject(parentNode, child, true);
}
/*
addObject: This method is called to add child into the give tree node.
Parameter: parent - Object of DefaultMutableTreeNode class, child - Object of Object class.
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent,Object child)
{
   return addObject(parent, child, true);
}
/*
addObject: This method is called to add child into the give tree node.
Parameter: parent - Object of DefaultMutableTreeNode class, child - Object of Object
class, shouldBeVisible
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent, Object child, boolean
shouldBeVisible)
{
   DefaultMutableTreeNode childNode = new DefaultMutableTreeNode(child);
   if (parent == null)
   {
      parent = rootNode;
   }
   treeModel.insertNodeInto(childNode, parent, parent.getChildCount());
   if (shouldBeVisible)
   {
      tree.scrollPathToVisible(new TreePath(childNode.getPath()));
   }
   return childNode;
}
/*
class DynamicTree - This class implements TreeModelListener interface.
*/
class RosterTreeModelListener implements TreeModelListener
{
   public void treeNodesChanged(TreeModelEvent e)
   {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)(e.getTreePath().getLastPathComponent());
      try
      {
         int index = e.getChildIndices()[0];
         node = (DefaultMutableTreeNode)(node.getChildAt(index));
      }
      catch (NullPointerException exc) {}
```

```
        }
        public void treeNodesInserted(TreeModelEvent e) {}
        public void treeNodesRemoved(TreeModelEvent e) {}
        public void treeStructureChanged(TreeModelEvent e) {}
    }
}
```

Download this listing.

In the above code, the constructor of the DynamicTree class takes an object of the DefaultMutableTreeNode class and creates a tree structure of the contact list for the Contact List application.

The methods defined in Listing 7-7 are:

- `clear()`: Removes all end user names from the tree structure of the contact list.
- `getTreeHandler()`: Returns the object of the JTree class.
- `removeCurrentNode()`: Removes the selected end users from the tree structure of the contact list.
- `addObject()`: Adds other end users to the tree structure of the contact list.

« PREVIOUS NEXT »

Viewing Contact Information

The UserInfo.java file creates the user interface that allows end users to view the contact information of the selected end user from the tree structure of the Contact List application. Listing 7-8 shows the contents of the UserInfo.java file:

Listing 7-8: The UserInfo.java File

```
/*Import required javax.swing classes*/
import javax.swing.*;
/*Import required java.awt classes*/
import java.awt.*;
/*Import required java.awt.event. classes*/
import java.awt.event.*;
/*
class UserInfo - This class is used to show user profile which, is selected in the roster tree.
Constructor:
UserInfo-This constructor creates GUI.
Methods:
setInfo: This method is used to show user information text into labels.
*/
public class UserInfo extends JDialog  implements ActionListener
{
   /*Declares object of Container class.*/
   Container container;
   /*Declares objects of JLabel class.*/
   JLabel heading= null;
   JLabel userid = null;
   JLabel nickname = null;
   JLabel email = null;
   JLabel phone = null;
   JLabel address = null;
   JLabel useriddata = null;
   JLabel nicknamedata = null;
   JLabel emaildata = null;
   JLabel phonedata = null;
   JLabel addressdata = null;
   /*Declares objects of String class.*/
   String vcardxml;
   String fname="";
   String nname="";
   String addressinfo="";
   String telnumber="";
   /*Declares object of JButton class.*/
   JButton okbutton;
   public UserInfo()
   {
      container = this.getContentPane();
      /*Declares and initializes the object of JPanel class.*/
      JPanel toppanel = new JPanel();
      setTitle("Contact Information");
      /*Declares and initializes the objects of JLabel class.*/
      heading = new JLabel("Contact Information");
      userid = new JLabel("User Id:");
      nickname = new JLabel("User Name:");
      email=new JLabel("Email:");
      phone=new JLabel("Phone:");
      address=new JLabel("Address:");
      useriddata=new JLabel("");
      nicknamedata=new JLabel("");
      emaildata=new JLabel();
      phonedata=new JLabel("");
      addressdata=new JLabel("");
      /*Declares and initializes the objects of JButton class.*/
      okbutton=new JButton("OK");
      /*Sets the font of the heading.*/
      heading.setFont(new Font("Verdana", Font.BOLD, 12));
      /*Adds heading to the toppanel.*/
      toppanel.add(heading);
      container.add(toppanel,BorderLayout.NORTH);
      /*Declares and initializes the objects of JPanel class.*/
      JPanel userinfopanel = new JPanel(new GridLayout(5, 6, 10, 20));
      /*Adds userid to the userinfopanel.*/
      userinfopanel.add(userid);
      /*Sets the font of the useriddata.*/
      useriddata.setFont(new Font("Verdana", Font.BOLD, 10));
      /*Adds useriddata to the userinfopanel.*/
      userinfopanel.add(useriddata);
      /*Adds nickname to the userinfopanel.*/
      userinfopanel.add(nickname);
      /*Sets the font of the nicknamedata.*/
      nicknamedata.setFont(new Font("Verdana", Font.BOLD, 10));
      /*Adds nicknamedata to the userinfopanel.*/
      userinfopanel.add(nicknamedata);
      /*Adds email to the userinfopanel.*/
```

```
        userinfopanel.add(email);
        /*Sets the font of the emaildata.*/
        emaildata.setFont(new Font("Verdana", Font.BOLD,10));
        /*Adds emaildata to the userinfopanel.*/
        userinfopanel.add(emaildata);
        /*Adds phone to the userinfopanel.*/
        userinfopanel.add(phone);
        /*Sets the font of the phonedata.*/
        phonedata.setFont(new Font("Verdana", Font.BOLD, 10));
        /*Adds phonedata to the userinfopanel.*/
        userinfopanel.add(phonedata);
        /*Adds address to the userinfopanel.*/
        userinfopanel.add(address);
        /*Sets the font of the addressdata.*/
        addressdata.setFont(new Font("Verdana",Font.BOLD,10));
        /*Adds addressdata to the userinfopanel.*/
        userinfopanel.add(addressdata);
        container.add(userinfopanel);
        /*Declares and initializes the objects of JPanel class.*/
        JPanel buttonpanel = new JPanel();
        buttonpanel.add(okbutton);
        okbutton.addActionListener(this);
        container.add(buttonpanel,BorderLayout.SOUTH);
        setSize(350,300);
        setVisible(true);
    }
    /*
    setInfo: This method is used to show user information text into labels.
    Parameter: vcardString - Object of String class, nicknameto - Object of String class,
    UserID - Object of String class
    Return Value: String
    */
    public void setInfo(String vcardString,String nicknameto,String UserID)
    {
        vcardxml=vcardString;
        fname=vcardxml.substring(vcardxml.indexOf("<FN>")+4,vcardxml.indexOf("</FN>"));
        nname=vcardxml.substring(vcardxml.indexOf("<NICKNAME>")+10,vcardxml.indexOf
        ("</NICKNAME>"));
        if ((vcardxml.indexOf("<HOME/>")==-1)&&(vcardxml.indexOf("</HOME>")!=-1))
        {
            telnumber=vcardxml.substring(vcardxml.indexOf("<HOME>")+6,vcardxml.indexOf
            ("</HOME>"))+"\n";
        }
        if ((vcardxml.indexOf("<LOCALITY/>")==-1)&&(vcardxml.indexOf
        ("</LOCALITY>")!=-1))
        {
            addressinfo=addressinfo+vcardxml.substring(vcardxml.indexOf("<LOCALITY>")+10,
            vcardxml.indexOf("</LOCALITY>"));
        }
        if ((vcardxml.indexOf("<LOCALITY/>")==-1)&&
        (vcardxml.indexOf ("</LOCALITY>")!=-1))
        {
            addressinfo=addressinfo+vcardxml.substring(vcardxml.indexOf("<LOCALITY>")+10,
            vcardxml.indexOf("</LOCALITY>"));
        }
        useriddata.setText(UserID.substring(1,UserID.length()-1));
        nicknamedata.setText(nname);
        addressdata.setText(addressinfo);
        phonedata.setText(telnumber);
    }
    public void actionPerformed(ActionEvent ae)
    {
        if(ae.getSource() == okbutton)
        {
            this.dispose();
        }
    }
}
```

Download this listing.

In the above code, the constructor of the UserInfo class creates the user interface to view the contact information of the selected end user.

The methods defined in Listing 7-8 are:

- `setInfo()`: Shows the contact information text of the selected end user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.

The UserInfo.java file creates the user interface that allows an end user to view the contact information of the selected end user, as shown in Figure 7-8:

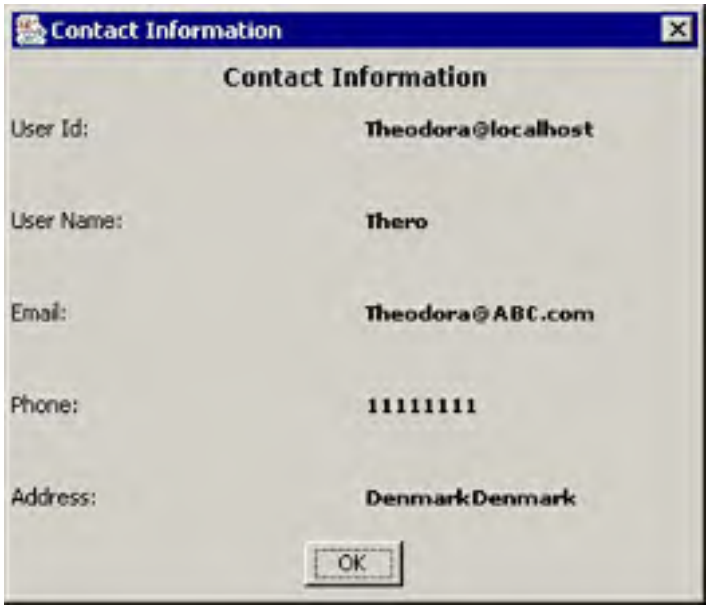


Figure 7-8: The Contact Information Window

PREVIOUS NEXT

Unit Testing

To test the Contact List application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the UserLogin.java, ContactList.java, UserInfo.java, AddUser.java, EditDelete.java, DynamicTree.java, ChatWindow.java, and SocketClass.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

5. To run the Contact List application, specify the following command at the command prompt:

   ```
   java ContactList
   ```

6. The Login window of the Contact List application appears. Enter the login information, as shown in Figure 7-9:

Figure 7-9: Specifying Login Information

7. Click the Submit button to send the login information to the Jabber Server for registered end users or open a new account in the Jabber server for unregistered end users.

8. The user interface of the Contact List application appears. Select File->Add User to add other end users to the contact list and send a welcome message to the added end users.

9. Enter the User ID, User Name, and a welcome text message to display to the new end users added to the contact list, as shown in Figure 7-10:

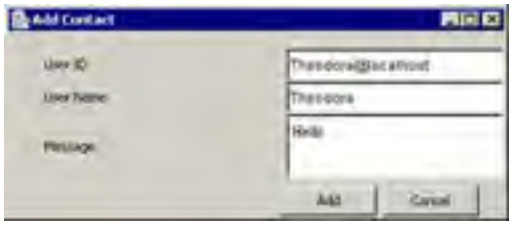


Figure 7-10: Adding an End User

10. Click the Add button to add the specified end user to the contact list and send the welcome text message to the added end user, as shown in Figure 7-11:

Figure 7-11: Updated Contact List

11. Select File->Manage User to manage the contact list.

12. Select any end user name from the tree structure of the of the contact list. The user information of the selected end user appears, as shown in Figure 7-12:

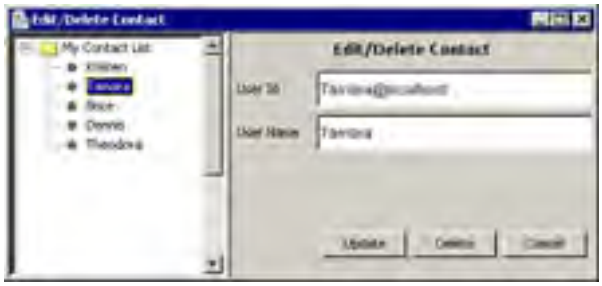


Figure 7-12: Managing a Contact

13. Modify the information and click the Update button. The information gets updated, as shown in Figure 7-13:

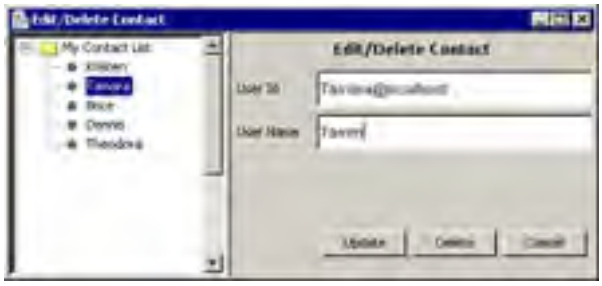


Figure 7-13: Updated Information in the Contact List

Index

A-I

airline schedules, Chapter 6: Creating an Airline Reservation Application
application, Chapter 7: Creating a Contact List Application
chat room, Chapter 3: Creating a Chat Room Application
Ejabberd, Chapter 2: Creating a Connector Application
Extensible Messaging and Presence Protocol, Chapter 2: Creating a Connector Application
Group Chatting application, Chapter 4: Creating a Group Chatting Application
IM, Chapter 1: Introduction
Instant Messaging, Chapter 1: Introduction

Index

J-P

Jabber protocol, Chapter 3: Creating a Chat Room Application, Chapter 4: Creating a Group Chatting Application

Jabber server, Chapter 1: Introduction, Chapter 2: Creating a Connector Application, Chapter 3: Creating a Chat Room Application, Chapter 4: Creating a Group Chatting Application, Chapter 7: Creating a Contact List Application

Jabber Software Foundation, Chapter 1: Introduction

JSF, Chapter 1: Introduction

LDAP, Chapter 2: Creating a Connector Application

Lightweight Directory Access Protocol, Chapter 2: Creating a Connector Application

online, Chapter 7: Creating a Contact List Application

protocol, Chapter 3: Creating a Chat Room Application, Chapter 6: Creating an Airline Reservation Application, Chapter 7: Creating a Contact List Application

Index

R-X

reservations, Chapter 6: Creating an Airline Reservation Application
server, Chapter 6: Creating an Airline Reservation Application
XML, Chapter 2: Creating a Connector Application
XMPP, Chapter 2: Creating a Connector Application

List of Figures

Chapter 2: Creating a Connector Application

Figure 2-1: Architecture of the Connector Application

Figure 2-2: User Interface of the Connector Application

Figure 2-3: File Menu of the Connector Application

Figure 2-4: The Logoff Confirmation Message

Figure 2-5: The Unsubscribe Confirmation Message

Figure 2-6: The Sign Up Window

Figure 2-7: The Login Window

Figure 2-8: The Message Window

Figure 2-9: The Enter Chat Room Window

Figure 2-10: Specifying Information in the Sign Up Window

Figure 2-11: The Signup Confirmation

Figure 2-12: Specifying Information in the Login Window

Figure 2-13: The Login Confirmation Message

Figure 2-14: Specifying the Message Text

Figure 2-15: Message Confirmation

Figure 2-16: Specifying the Chat Room Name

Figure 2-17: Enter Chat Room Confirmation

Chapter 3: Creating a Chat Room Application

Figure 3-1: Architecture of the Chat Room Application

Figure 3-2: The Login Window

Figure 3-3: User Interface of the Chat Room Application

Figure 3-4: The File Menu of the Chat Room Application

Figure 3-5: Joining a Chat Room

Figure 3-6: Creating a Chat Room

Figure 3-7: Creating a Group Chat Window

Figure 3-8: Specifying Login Information

Figure 3-9: Entering a Chat Room

Figure 3-10: The Group Chat Window of the Specified Chat Room

Figure 3-11: Entering Chat Room Name and User Name

Figure 3-12: The Group Chat Window of the New Chat Room

Figure 3-13: Sending a Message

Figure 3-14: Selecting an Existing Room

Figure 3-15: The Input Dialog Box

Figure 3-16: The Group Chat Window of the Selected Chat Room

Chapter 4: Creating a Group Chatting Application

Figure 4-1: Architecture of the Group Chatting Application

Figure 4-2: The Login Window

Figure 4-3: User Interface of the Group Chatting Application

Figure 4-4: The File Menu of the Group Chatting Application

Figure 4-5: Adding an End User

Figure 4-6: Renaming a Group

Figure 4-7: Private Chat Window

Figure 4-8: The Group Chat Window

Figure 4-9: Creating a Group Tree

Figure 4-10: Specifying Login Information

Figure 4-11: Adding an End User to a Group

Figure 4-12: Entering the Old and New Group Names

Figure 4-13: Renamed Group

Figure 4-14: Sending a Private Message

Figure 4-15: Sending a Group Message

Chapter 5: Creating an Instant Technical Support Application

Figure 5-1: Architecture of the Instant Technical Support Application

Figure 5-2: The Home Page

Figure 5-3: The Signup Page

Figure 5-4: The Login Page

Figure 5-5: User Interface of the Instant Technical Support Application

Figure 5-6: The Signup Information

Figure 5-7: Specifying the Query Text

Figure 5-8: Sending the Query

Figure 5-9: Receiving the Response for the Specified Query

Chapter 6: Creating an Airline Reservation Application

Figure 6-1: Architecture of the Airline Reservation Application

Figure 6-2: The Personal Information Page

Figure 6-3: User Interface of the Airline Reservation Application

Figure 6-4: Showing the Airline Schedules

Figure 6-5: Confirmation Message

Figure 6-6: Entering Personal Information

Figure 6-7: Searching for an Airline Schedule

Figure 6-8: Reserving an Airline Schedule

Chapter 7: Creating a Contact List Application

Figure 7-1: Architecture of the Contact List Application

Figure 7-2: The Login Window

Figure 7-3: User Interface of the Contact List Application

Figure 7-4: The File Menu of the Contact List Application

Figure 7-5: Adding End Users

Figure 7-6: Managing the Contact List

Figure 7-7: The Chat Window

Figure 7-8: The Contact Information Window

Figure 7-9: Specifying Login Information

Figure 7-10: Adding an End User

Figure 7-11: Updated Contact List

Figure 7-12: Managing a Contact

Figure 7-13: Updated Information in the Contact List

◄ PREVIOUS NEXT ►

List of Listings

Chapter 2: Creating a Connector Application

Listing 2-1: The JabberClient.java File

Listing 2-2: The SignUp.java File

Listing 2-3: The UserLogin.java File

Listing 2-4: The SocketClass.java File

Listing 2-5: The MessageClass.java File

Listing 2-6: The ChatRoom.java File

Chapter 3: Creating a Chat Room Application

Listing 3-1: The LoginGUI.java File

Listing 3-2: The MainGUI.java File

Listing 3-3: The ChatRoom.java File

Listing 3-4: The RoomInformation.java File

Listing 3-5: The SocketOpener.java File

Listing 3-6: The GroupChat.java File

Chapter 4: Creating a Group Chatting Application

Listing 4-1: The GroupLoginGUI.java File

Listing 4-2: The GroupList.java File

Listing 4-3: The AddUser.java File

Listing 4-4: The EditGroup.java File

Listing 4-5: The ChatWindow.java File

Listing 4-6: The GroupChat.java File

Listing 4-7: The SocketConnection.java File

Listing 4-8: The GroupTree.java File

Chapter 5: Creating an Instant Technical Support Application

Listing 5-1: The welcome.html File

Listing 5-2: The signuppage.html File

Listing 5-3: The loginpage.html File

Listing 5-4: The chathtml.html File

Listing 5-5: The ChatMainApplet.java File

Chapter 6: Creating an Airline Reservation Application

Listing 6-1: The UserInformation.html File

Listing 6-2: The Reservation.html File

Listing 6-3: The ReservationStatus.java File

Listing 6-4: The SocketClass.java File

Listing 6-5: The SocketConnector.java File

Listing 6-6: The ReservationResultTable.java File

Listing 6-7: The XMLReader.java File

Listing 6-8: The airlines.xml File

Chapter 7: Creating a Contact List Application

Listing 7-1: The UserLogin.java File

Listing 7-2: The ContactList.java File

Listing 7-3: The AddUser.java File

Listing 7-4: The EditDelete.java File

Listing 7-5: The ChatWindow.java File

Listing 7-6: The SocketClass.java File

Listing 7-7: The DynamicTree.java File

Listing 7-8: The UserInfo.java File

PREVIOUS NEXT

CD Content

Following are select files from this book's Companion CD-ROM. These files are copyright protected by the publisher, author, and/or other third parties. Unauthorized use, reproduction, or distribution is strictly prohibited.

File	Description	Size
All CD Content	Java InstantCode: Developing Applications Using Jabber	307,608
Chapter 2:	Creating a Connector Application	47,592
Chapter 3:	Creating a Chat Room Application	50,479
Chapter 4:	Creating a Group Chatting Application	62,588
Chapter 5:	Creating an Instant Technical Support Application	25,080
Chapter 6:	Creating an Airline Reservation Application	55,686
Chapter 7:	Creating a Contact List Application	66,293

Contents

Related Links:

InstantCode Series

Collections:

BusinessPro
ITPro
OfficeEssentials

Also Available:

ITPro

Java InstantCode: Developing Applications Using Jabber

SkillSoft Press © 2004

This code-rich reference includes many applications, such as instant reservation, group chatting, contact list, and chat room applications

Team LiB

Introduction

About InstantCode Books

The InstantCode series is designed to provide you - the developer - with code you can use for common tasks in the workplace. The goal of the InstantCode series is not to provide comprehensive information on specific technologies - this is typically well-covered in other books. Instead, the purpose of this series is to provide actual code listings that you can immediately put to use in building applications for your particular requirements.

How These Books are Structured

The underlying philosophy of the InstantCode series is to present code listings that you can download and apply to your own business needs. To support this, these books are divided into chapters, each covering an independent task.

Each chapter includes a brief description of the task, followed by an overview of the element of the book's subject technology that we will use to perform that task. Each section ends with a code listing: each of the individual code segments in the chapter is independently downloadable, as is the complete chapter code. You will be able to download source code files, as well as application files.

Who Should Read These Books

These books are written for software development professionals who have basic knowledge of the associated technology and want to develop customized technology solutions.

About the Book

Jabber is an open source set of streaming XML protocols used for messaging, such as Instant Messaging (IM), and remote system monitoring. Jabber allows real-time communication between two entities over the Internet via a proxy, known as the Jabber server. Jabber community uses Jabber protocol to develop IM based applications. Jabber Software Foundation (JSF) manages the Jabber protocol. Jabber Inc markets commercial solutions developed using Jabber. This book includes many application, which includes instant technical support, airline reservation, group chatting, contact list, and chat room applications.

About the Author

Anurag Sharma has 3 years of working experience in the field of Software Engineering. He has authored various books on JCA and JSAPI.

Credits

I would like to thank Radhika Chauhan and Gaurav Bathla for helping me complete the book on time and providing continuous support and encouragement.

Copyright

Java InstantCode: Developing Applications Using Jabber

Published by SkillSoft Corporation
20 Industrial Park Drive
Nashua, NH 03062
(603) 324-3000

information@skillsoft.com

Chapter 1: Introduction

Jabber is an open source set of streaming XML protocols for messaging, such as Instant Messaging (IM) and remote system monitoring that allow real-time communication between two entities over the Internet via a proxy, known as the Jabber server.

Jabber community uses Jabber protocol to develop IM based applications. Jabber Software Foundation (JSF) manages the Jabber protocol. Jabber Inc markets commercial solutions developed using Jabber.

Features of Jabber

Jabber follows the client-server architecture and provides the following features:

- Open Protocols: Provides streaming XML protocols known as Extensible Messaging and Presence Protocol (XMPP). The open-source community started Jabber. You can implement the Jabber protocols and may use any code license.
- XML Data Format: Supports XML, which is an integral part of Jabber. This makes Jabber flexible to transfer structured data.
- Standard-based addressing: Allows you to include new IM end users and identify individual entities. A Jabber server represents each end user as a different entity.
- Distributed network: Provides a client-server architecture similar to that of an e-mail. Jabber utilizes a distributed network of servers by using a common protocol to interact with clients and manage IM.
- Exchange Messages: Allows end users to send and receive instant messages in real-time environment. It provides both one-to-one chat and group chat.
- Roster: Maintains a server-side contact list called roster. An end user can manage the roster by creating groups and assigning contacts to these groups. End users can also update, add, and delete contact information from a roster.
- Presence: Sets indicators to inform whether or not an end user is present in the chat session.

Jabber Architecture

Jabber IM system work on a client-server architecture, which is similar to that of an e-mail. In a Jabber IM system, you can develop and run a customized Jabber server. For communication, every end user connects to a personal server, which maintains and receives information for the end user. The Jabber server uses two ports, 5222 and 5269. The 5222 port is used for communication between client and server applications and the 5269 port is used for server-to-server communication. The protocol used for communication between Jabber client and Jabber server or Jabber server and Jabber server is Transmission Control Protocol (TCP). To transmit messages, the Jabber server uses the push technique instead of the pull technique. This means that client does not ask for new messages but the server sends the messages to the client as soon as the server receives them.

Jabber Server

Jabber servers are modular and can communicate with any other Jabber server. A Jabber server handles all client connections, communicates directly with the Jabber clients, and maintains co-ordination between different server components. Each Jabber server contains definite code packages, which can handle services such as maintaining contact lists, registration and authentication of end users, and storage of offline messages. The basic functioning of the Jabber servers can be extended by using external components to handle advanced capabilities, such as providing gateways to other IM systems.

Jabber Client

A Jabber client allows you to:

- Communicate with Jabber servers by using the TCP sockets.
- Parse and comprehend XML data over an XML stream.
- Understand the main Jabber data types.

There are various Jabber client libraries, which handle the complex functions such as parsing of data at the client end. This allows you to concentrate on the user interface of the client application.

Note:

You can develop Jabber client in any language that supports XML messaging. The applications in this book use Java language to develop IM applications.

Jabber Identifier

Jabber Identifier (JID) provides a unique identification to different entities that communicate with each other by using Jabber protocols. The format of a JID is:

```
[node@]domain[/resource]
```

The elements in the JID are:

- The node identifier that signifies an end user. It is a mandatory element for a valid JID.
- The domain identifier that signifies the Jabber server. It is a mandatory element for a valid JID.
- The resource identifier signifies specific resources that belong to an end user. End users can maintain multiple simultaneous connections to the same Jabber server by using different resources.

Jabber XML Protocol

Jabber supports communication by using XML streams. The three core XML elements in the Jabber XML protocol are:

- <message/>: Contains messages send between two Jabber end users. Jabber supports various message types, such as normal messages, chat, group chat, headline, and error messages.
- <presence/>: Provides information about whether or not a Jabber entity, represented by JID, is available.
- <iq/>(Info/Query): Structures a basic conversation between two Jabber entities and allows the participating objects to exchange XML-formatted data.

Advantages of Jabber

Jabber allows you to develop products for real-time communication and presence-based services in which a Jabber server maintains chat sessions of end users. Some advantages of Jabber are:

- Open Source: Provides open-source protocols. Jabber also provides free and paid implementations for servers, clients, and components that can be integrated into your existing applications. The free implementation of servers includes jabberd version 1.0 onwards, OpenIM and ejabberd. The paid implementation of servers includes Antepo and Merak. The free implementation of client includes Agile, Colibri, and e4Applet. The paid implementation of client includes Akeni, Chatopus, and IMChat. The freeware components include oajabber. The paid compenents includes myMatrix and XMPP Com Library.
- XML Message Structure: Utilizes XML message structure for data streaming between end users.
- Decentralization: Allows you to develop and run your customized Jabber server due to similarity between Jabber network and e-mail architecture. This allows companies and individuals to run their customized IM services.
- Extensibility: Allows extending the basic XML format by using the XML and custom namespaces to customize the protocols for specialized applications.
- Diversity: Provides flexibility to build real-time applications and services apart from IM, such as file sharing, gaming, and remote system monitoring.

Future Enhancements

There is a scope for improvement to extend the functionality provided by Jabber. The Jabber community is currently working on enhancements that can be incorporated in future releases. The enhancements that Jabber server should support are:

- Support for enhanced basic functions, such as voice chat and whiteboard.
- Support for standard enterprise features, such as quality of service support.
- Support for improved security, and error-detection and correction mechanisms to formulate more advanced communication systems.

Chapter 2: Creating a Connector Application

Download CD Content

A Jabber session consists of two parallel XML streams, one from the client to the server and the other from the server to the client. The Connector application in this chapter uses the Jabber server, Ejabberd, to allow communication between end users.

Ejabberd is a free and open source distributed Jabber server. You can download Ejabberd from the following URL:

http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=10

Ejabberd is Extensible Messaging and Presence Protocol (XMPP) compliant and provides various features, such as built-in Web-based administration interface, support for multi-user chat sessions, and support for Lightweight Directory Access Protocol (LDAP) authentication.

This chapter describes how to develop the Connector application, which allows end users to send and receive messages from other end users connected to the Jabber server. To send and receive messages, end users need to first log on or sign up for a new account.

Architecture of the Connector Application

The Connector application provides an interface with two text areas, Client Response and Server Response. The application allows end users to view the XML streams sent and received by the Connector application in the Client Response and the Server Response text area respectively. The Client Response text area shows the request sent to the Jabber server. The Server Response text area shows the response received from the Jabber server. The Connector application allows end users to view XML streams for various tasks, such as sign up, log on, send message, enter chat room, and log off.

The Connector application uses the following files:

- JabberClient.java: Allows end users to establish a connection with the Jabber server and communicate with other end users connected to the server.
- ChatRoom.java: Allows end users to enter any chat room that exists on the Jabber server.
- MessageClass.java: Allows end users to send messages to other end users on the Internet.
- SignUp.java: Allows end users to sign up for a new account.
- UserLogin.java: Allows end users to log on as existing users.
- SocketClass.java: Opens a socket to send and receive messages with the help of the Jabber server.

Figure 2-1 shows the architecture of the Connector application:

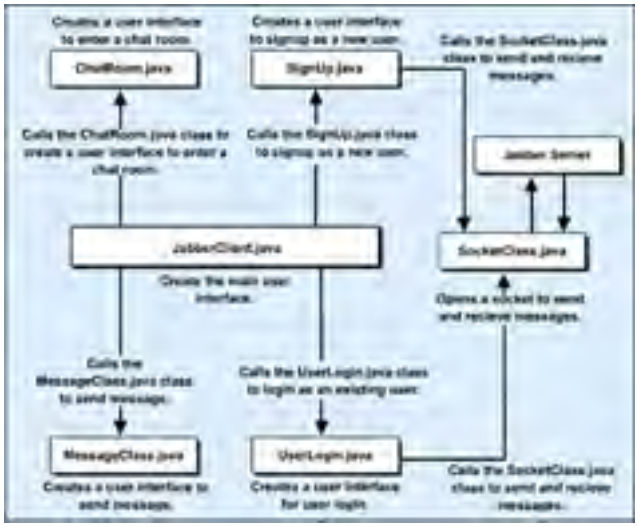


Figure 2-1: Architecture of the Connector Application

The JabberClient.java file creates the user interface of the Connector application that contains the File menu and two text areas, Client Request and Server Response. The File menu allows end users to sign up for a new account or log on as existing end users.

If the end user selects File-> Login, the JabberClient.java file calls the UserLogin.java file, which allows the existing end users to log on. The UserLogin.java file provides an interface with various labels, three text boxes, and the Submit button.

If the end user selects File-> Signup, the JabberClient.java file calls the SignUp.java file, which allows end users to sign up for a new account. The SignUp.java file provides an interface with various labels, text boxes, and two buttons, OK and Cancel.

If the end user selects File-> Send Message, the JabberClient.java file calls the MessageClass.java file, which allows end users to send a message to other end users connected to the Jabber server. The MessageClass.java file provides an interface with various labels, text boxes, and two buttons, Send Message and Close.

If the end user selects File-> Enter Chat Room, the JabberClient.java file calls the ChatRoom.java file, which allows end users to enter any particular chat room that exists on the Jabber server. The ChatRoom.java file provides an interface with a label, text box, and two buttons, OK and Cancel.

Team LiB

PREVIOUS NEXT

Creating the User Interface for the Connector Application

The JabberClient.java file creates the user interface for the Connector application. Listing 2-1 shows the contents of the JabberClient.java file:

Listing 2-1: The JabberClient.java File

```
/*Imports required swing classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*
class JabberClient-This class is the main class of the application. This class initializes the
interface and loads all components like the JTextPane, button before displaying the result.
Constructor:
JabberClient-This constructor creates GUI.
Methods:
getStatus: This method is called to retrieve user status.
setStatus: This method is called to set user status.
setStatusLabel: This method is called to show status in to label.
setServerResponse: This method is called to show server response into text pane.
setMessageInToTextBox: This method is called to retrieve server name.
getServerName: This method is called to retrieve server name.
getUserName: This method is called to retrieve user name.
setUserNameAndPassword: This method is called to sets user name, password, and server name.
getRoomName: This method is called to retrieve room name.
setRoomName: This method is called to set the name of the chat room specified by the end user.
disableAndEnableMenuItems: This method is called to sets enabling or disabling of menu items.
main - This method creates the object of JabberClient.
*/
public class  JabberClient extends JFrame implements ActionListener
{
   /*Declares object of String class.*/
   String username="";
   String password="";
   String servername="";
   String usermode="notconnected";
   String roomname="";
   /*Declares object of Container class.*/
   Container container = null;
   /*Declares object of JMenuBar class.*/
   JMenuBar menubar;
   /*Declares object of JMenu class.*/
   JMenu filemenu;
   /*Declares object of JMenuItem class.*/
   JMenuItem login=null;
   JMenuItem signup = null;
   JMenuItem logoff = null;
   JMenuItem unsubscribe = null;
   JMenuItem sendmessagemenuitem = null;
   JMenuItem enterroommenuitem = null;
   JMenuItem exit = null;
   /*
   Declares object of JCheckBoxMenuItem class.
   */
   JCheckBoxMenuItem available = null;
   JCheckBoxMenuItem unavailable = null;
   JCheckBoxMenuItem chat = null;
   JCheckBoxMenuItem away = null;
   JCheckBoxMenuItem dod = null;
   JCheckBoxMenuItem gfc = null;
   JCheckBoxMenuItem newstatus = null;
   /*Declares object of JTextPane class.*/
   JTextPane clienttextpane = null;
   JTextPane servertextpane = null;
   /*Declares object of JScrollPane class.*/
   JScrollPane scpane = null;
   /*Declares object of JLabel class.*/
   JLabel statuslabel = null;
   /*Declares object of JButton class.*/
   JButton submitbutton = null;
   /*Declares object of SocketClass class.*/
   SocketClass socketclass;
   public JabberClient()
   {
      super("Connector Application");
      /*
      Initializes and sets the look and feel of the application to windows look and feel.
      */
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

```
}
catch(Exception e)
{
   System.out.println("Unable to set WLF"+e);
}
container =  this.getContentPane();
/*
Declares and initializes object of Dimension class.*/
Dimension screendimension=Toolkit.getDefaultToolkit().getScreenSize();
/* Sets window's location on screen.*/
setLocation(screendimension.width/2- 200,screendimension.height/2-350);
/*
Declares and initializes object of JPanel class.
*/
JPanel panel = new JPanel(false);
/*
Initializes object of JMenuBar class.
*/
menubar = new JMenuBar();
/*Add menubar to panel.*/
panel.add(menubar);
/*Initializes object of JMenu class.*/ filemenu = new JMenu("File");
/*
Initializes object of JMenuItem class.
*/
login= new JMenuItem("Login");
signup= new JMenuItem("Signup");
logoff= new JMenuItem("Logoff");
unsubscribe= new JMenuItem("Unsubscribe");
sendmessagemenuitem=new JMenuItem("Send Message");
enterroommenuitem=new JMenuItem("Enter Chat Room");
exit= new JMenuItem("Exit");
/*Adds menu to the menubar.*/
menubar.add(filemenu);
/*Adds menuitems to the menu.*/
filemenu.add(login);
filemenu.add(signup);
filemenu.add(logoff);
filemenu.add(unsubscribe);
filemenu.add(sendmessagemenuitem);
filemenu.add(enterroommenuitem);
filemenu.add(exit);
setJMenuBar(menubar);
/*
Adds action listener to the menuitems.
*/
login.addActionListener(this);
signup.addActionListener(this);
logoff.addActionListener(this);
unsubscribe.addActionListener(this);
sendmessagemenuitem.addActionListener(this);
senterroommenuitem.addActionListener(this);
sendmessagemenuitem.setEnabled(false);
enterroommenuitem.setEnabled(false);
exit.addActionListener(this);
/*
Declares and initializes objects of the JPanel class.
*/
JPanel rootpanel =  new JPanel(new GridLayout(2, 1, 6, 6));
JPanel clientpanel = new JPanel(new BorderLayout());
/*
Initializes object of JTextPane class.
*/
clienttextpane = new JTextPane();
/*
Declares and initializes objects of the JLabel class.
*/
JLabel clientrequestlabel = new JLabel("Client Request");
/*
Sets font of the clientrequestlabel label.
*/
clientrequestlabel.setFont(newFont("Verdana", Font.BOLD, 10));
/*
Adds clientrequestlabel to the clientpanel panel.
*/
clientpanel.add(clientrequestlabel, BorderLayout.NORTH);
/*
Sets size of the clienttextpane textpane
*/
clienttextpane.setPreferredSize(new Dimension(180, 120));
/*
Initializes object of JScrollPane class.
*/
scpane = new JScrollPane(clienttextpane, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
/*Adds scrollpane to the panel.*/
clientpanel.add(scpane,BorderLayout.CENTER);
/*
```

```
   Declares and initializes objects of the JPanel class.
   */
   JPanel submitpanel=new JPanel(new FlowLayout());
   /*
   Initializes object of JButton class.
   */
   submitbutton = new JButton("Submit");
   /*Adds action listener to the button.*/
   submitbutton.addActionListener(this);
   /*Adds button to the panel.*/
   submitpanel.add(submitbutton);
   clientpanel.add(submitpanel, BorderLayout.SOUTH);
   /*
   Declares and initializes objects of the JPanel class.
   */
   JPanel serverresponsepanel = new JPanel(new BorderLayout());
   /*
   Initializes objects of the JTextPane class.
   */
   servertextpane = new JTextPane();
   /*
   Declares and initializes objects of the JLabel class.
   */
   JLabel serverresponselabel = new JLabel("Server Response");
   /*Sets font of label.*/
   serverresponselabel.setFont(new Font("Verdana", Font.BOLD,10));
   /* Adds label to panel.*/
   serverresponsepanel.add(serverresponselabel, BorderLayout.NORTH);
   /*Sets size of textpane.*/
   servertextpane.setPreferredSize(new Dimension(180, 120));
   /*
   Initializes object of JScrollPane class.
   */
   scpane  = new JScrollPane(servertextpane, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
   JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
   /*Adds scrollpane to the panel.*/
   serverresponsepanel.add(scpane, BorderLayout.CENTER);
   /*
   Declares and initializes objects of the JPanel class.
   */
   JPanel emptypanel=new JPanel(new FlowLayout());
   /*
   Declares and initializes objects of the JLabel class.
   */
   JLabel empty = new JLabel(" ");
   emptypanel.add(empty);
   serverresponsepanel.add(emptypanel,BorderLayout.SOUTH);
   /*Adds clientpanel to the rootpanel.*/
   rootpanel.add(clientpanel);
   /*
   Adds serverresponsepanel to the rootpanel.
   */
   rootpanel.add(serverresponsepanel);
   /*
   Initializes objects of the JLabel class.
   */
   statuslabel = new JLabel("Status : Not connected");
   /*Sets font of the statuslabel label.*/
   statuslabel.setFont(new Font("Verdana", Font.BOLD, 10));
   /*Initializes object of SocketClass.*/
   socketclass=new SocketClass(this);
   /*Adds rootpanel to the container.*/
   container.add(rootpanel);
   /*Adds statuslabel to the container.*/
   container.add(statuslabel,BorderLayout.SOUTH);
   /*Sets clienttextpane disable.*/
   clienttextpane.setEnabled(false);
   /*Sets servertextpane disable.*/
   servertextpane.setEnabled(false);
   /* Sets logoff disable.*/
   logoff.setEnabled(false);
   /*Sets unsubscribe disable.*/
   unsubscribe.setEnabled(false);
   setSize(420, 650);
   setVisible(true);
}
/*
disableAndEnableMenuItems: This method is called to sets enabling or disabling of menu items.
Parameter: flag
Return Value: N/A
*/
public void disableAndEnableMenuItems(boolean flag)
{
   signup.setEnabled(flag);
   login.setEnabled(flag);
   logoff.setEnabled(!flag);
   unsubscribe.setEnabled(!flag);
```

```
   sendmessagemenuitem.setEnabled(!flag);
   enterroommenuitem.setEnabled(!flag);
}
/*
setRoomName: This method is called to set the name of the chat room specified by the end user.
Parameter: room-Object of String class.
Return Value: N/A
*/
public void setRoomName(String room)
{
   roomname=room;
}
/*
getRoomName: This method is called to retrieve room name
Parameter: N/A
Return Value: String
*/
public String getRoomName()
{
   return roomname;
}
/*
setUserNameAndPassword: This method is called to sets user name, password, and server name.
Parameter: N/A
Return Value: N/A
*/
public void setUserNameAndPassword(String username, String password, String servername)
{
   this.username=username;
   this.password=password;
   this.servername=servername;
}
/*
getUserName: This method is called to retrieve user name.
Parameter: N/A
Return Value: String
*/
public String getUserName()
{
   return username;
}
/*
getServerName: This method is called to retrieve server name.
Parameter: N/A
Return Value: String
*/
public String getServerName()
{
   return servername;
}
/*
setMessageInToTextBox: This method is called to retrieve server name.
Parameter: messagestring - object of String class.
Return Value: N/A
*/
public void setMessageInToTextBox(String messagestring)
{
   clienttextpane.setText(messagestring);
}
/*
setServerResponse: This method is called to show server response into text pane.
Parameter: servermessage - object of String class.
Return Value: N/A
*/
public void setServerResponse(String servermessage)
{
   servertextpane.setText(servertextpane.getText()+"\n"+servermessage);
}
/*
setStatusLabel: This method is called to show status in to label.
Parameter: st - object of String class.
Return Value: N/A
*/
public void setStatusLabel(String st)
{
   statuslabel.setText("Status : "+st);
}
/*
setStatus: This method is called to set user status.
Parameter: status - object of String class.
Return Value: N/A
*/
public void setStatus(String status)
{
   usermode=status;
}
/*
getStatus: This method is called to retrieve user status.
```

```
   Parameter: N/A
   Return Value: String
   */
   public String getStatus()
   {
      return usermode;
   }
   public void actionPerformed(ActionEvent ae)
   {
      if(ae.getSource() == login)
      {
         UserLogin userlogin=new UserLogin(socketclass,this);
      }
      else if(ae.getSource() == signup)
      {
         SignUp sign =new SignUp(socketclass,this);
         sign.show();
      }
      else if(ae.getSource() == logoff)
      {
         clienttextpane.setText("</stream:stream>");
         setStatus("endofstream");
      }
      else if(ae.getSource() == unsubscribe)
      {
         String removesubscriptionstring="";
         removesubscriptionstring="<iq type='set' from='"+username+"@"+servername+"/Home'
         id='1'>";
         removesubscriptionstring=removesubscriptionstring+"<query xmlns='jabber:iq:register'>";
         removesubscriptionstring=removesubscriptionstring+"<remove/>";
         removesubscriptionstring=removesubscriptionstring+"</query>";
         removesubscriptionstring=removesubscriptionstring+"</iq>";
         clienttextpane.setText(removesubscriptionstring);
         setStatus("removereg");
      }
      else if(ae.getSource() == exit)
      {
         setVisible(false);
      }
      else if(ae.getSource() == sendmessagemenuitem)
      {
         MessageClass messageclass=new MessageClass(this);
      }
      else if(ae.getSource()== enterroommenuitem)
      {
         ChatRoom chatroom=new ChatRoom(this);
      }
      else if(ae.getSource() == submitbutton)
      {
         if (!clienttextpane.getText().trim().equals(""))
         {
            socketclass.sendXMLToJabber(clienttextpane.getText().trim());
            if (clienttextpane.getText().indexOf("<message")!=-1)
            {
               setStatusLabel(" Message has been send");
            }
            servertextpane.setText("");
            if (usermode.equals("waitforauth"))
            {
               setStatusLabel("Waiting for authentication");
            }
            if (usermode.equals("endofstream"))
            {
               setStatusLabel("Not Connected");
            }
            if (usermode.equals("removereg"))
            {
               setStatusLabel("Wait for remove registration");
            }
            System.out.println("submit");
         }
      }
   }
   public static void main(String args[])
   {
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      JabberClient jabclient = new JabberClient();
   }
}
```

Download this listing.

In the above code, the main() method creates an instance of the JabberClient class.

The methods defined in Listing 2-1 are:

- `getStatus()`: Retrieves the status of the end user, such as log on, log off, sign up, or unsubscribe.
- `setStatus()`: Sets the status of the end user.
- `setStatusLabel()`: Shows the status of the end user in the interface provided by the Connecter application.
- `setServerResponse()`: Shows the server response in the Server Response text area provided by the Connector application.
- `setMessageInToTextBox()`: Shows the message in the Client Response text area.
- `getServerName()`: Retrieves the name of the Jabber server.
- `getUserName()`: Retrieves the name of the end user currently logged in.
- `setUserNameAndPassword()`: Sets the end user name, password, and Jabber server name.
- `getRoomName()`: Retrieves the name of the chat room specified by the end user.
- `setRoomName()`: Sets the name of the chat room specified by the end user.
- `disableAndEnableMenuItems()`: Enables or disables File menu items depending on the options selected by the end user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the action the end user performs.

The JabberClient.java file generates the main window of the Connector application, as shown in Figure 2-2:

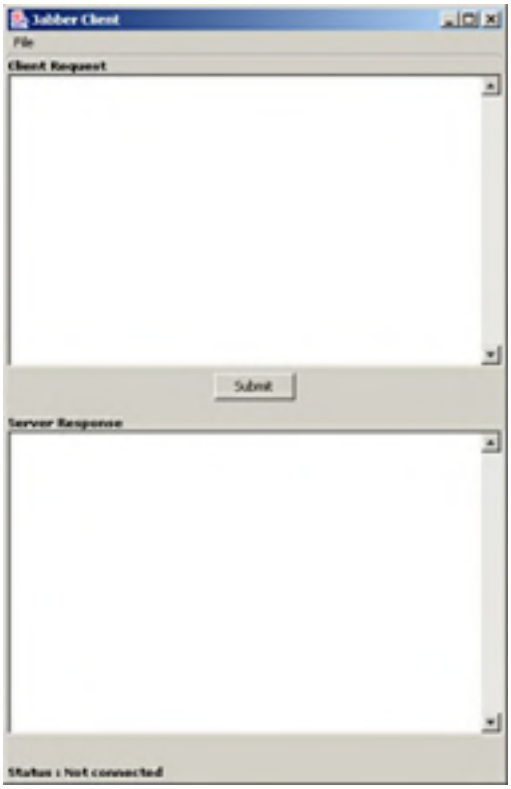


Figure 2-2: User Interface of the Connector Application

Select the File menu to view the File Menu options. Figure 2-3 shows the File menu options of the Connector application:

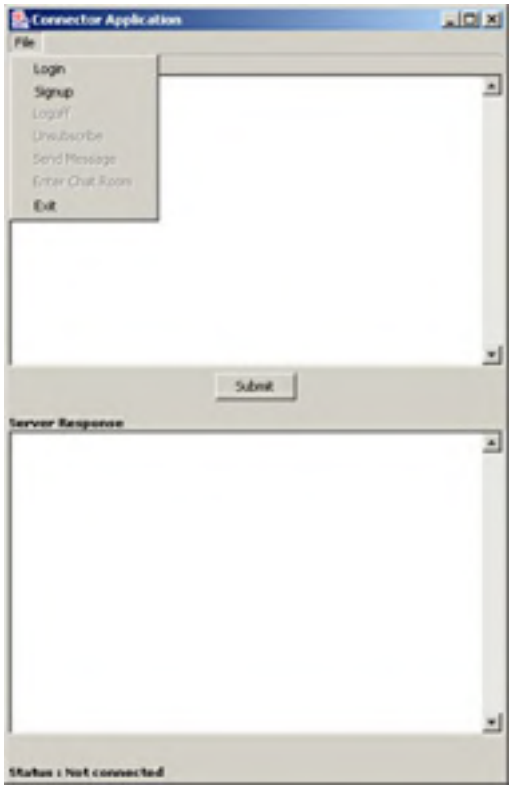


Figure 2-3: File Menu of the Connector Application

Select File-> Login to log on as an existing end user.

Select File-> Signup to sign up for a new account.

Select File-> Logoff and click the Submit button to log off from a chat session. The Connector application shows the confirmation message after logging out, as shown in Figure 2-4:

Figure 2-4: The Logoff Confirmation Message

Select File-> Unsubscribe and click Submit to unsubscribe from the Jabber server. The Connector application shows the confirmation message after unsubscribing an end user, as shown in Figure 2-5:

Figure 2-5: The Unsubscribe Confirmation Message

Select File-> Exit to close the application.

Creating the Sign Up Window

The SignUp.java file creates the user interface to allow an end user to sign up as a new user. Listing 2-2 shows the contents of the SignUp.java file:

Listing 2-2: The SignUp.java File

```
/*Imports required swing classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*
class SignUp - This class is used to create a GUI for signup a new user.
Constructor:
SignUp-This constructor creates GUI.
Methods:
actionPerformed - This method is called when end user clicks on send button.
checkfornumber -This function is used to check that supplied value contains a numeric value or not.
checkDateIsValid -This function is used to check that supplied date is a valid date.
isValidNumber -This function is used to check that supplied value is a valid number or not.
sendVCard -This function is used to create XML message of user profile.
*/
public class  SignUp extends JFrame implements ActionListener
{
   Container container;
   /*Declares objects of JLabel class.*/
   JLabel titlelabel;
   JLabel firstnamelabel = null;
   JLabel lastnamelabel = null;
   JLabel nicknamelabel=null;
   JLabel birthdaylabel=null;
   JLabel usernamelabel = null;
   JLabel passwordlabel = null;
   JLabel telephonelabel=null;
   JLabel confirmpassword = null;
   JLabel citylabel=null;
   JLabel countrylabel=null;
   JLabel addresslabel = null;
   /*Declares objects of JTextField class.*/
   JTextField firstnametext = null;
   JTextField lastnametext = null;
   JTextField nicknametext = null;
   JTextField usernametext = null;
   JPasswordField passwordtext = null;
   JPasswordField confirmpasswordtext = null;
   JTextField birthdaytext=null;
   JTextField telephonetext=null;
   JTextField addresstext = null;
   JTextField citytext=null;
   JTextField countrytext = null;
   JTextField titletext=null;
   /*Declares objects of JButton class.*/
   JButton okbutton = null;
   JButton cancelbutton = null;
   /*Declares objects of JPanel class.*/
   JPanel signuppanel;
   JPanel buttonpanel;
   /*Declares objects of SocketClass class.*/
   SocketClass socketclass;
   /*Declares objects of JabberClient class.*/
   JabberClient jabberclient;
   public SignUp(SocketClass socketclass,JabberClient jc)
   {
      super("Sign Up");
      container = this.getContentPane();
      this.socketclass=socketclass;
      jabberclient=jc;
      /*
      Declares and initializes the object of JPanel class.
      */
      JPanel jpanel = new JPanel();
      /*
      Declares and initializes the object of JLabel class.
      */
      JLabel labelheading =new JLabel("Sign Up");
      labelheading.setFont(new Font("Verdana", Font.BOLD,12));
      jpanel.add(labelheading);
      container.add(jpanel,BorderLayout.NORTH);
      /*
      Initializes the objects of JLabel class.*/
      titlelabel=new JLabel("Title");
      firstnamelabel = new JLabel("First Name :");
```

```
    lastnamelabel = new JLabel("Last Name :");
    nicknamelabel=new JLabel("Nick Name");
    birthdaylabel=new JLabel("DOB (dd/mm/yyyy)");
    usernamelabel = new JLabel("User Name :");
    passwordlabel = new JLabel("Password :");
    confirmpassword = new JLabel("Confirm Password :");
    telephonelabel=new JLabel("Telephone");
    addresslabel = new JLabel("Address :");
    citylabel=new JLabel("City");
    countrylabel=new JLabel("Country");
    /*
    Initializes the objects of JLabel class.*/
    titletext=new JTextField();
    firstnametext = new JTextField();
    lastnametext = new JTextField();
    nicknametext=new JTextField();
    usernametext = new JTextField();
    passwordtext = new JPasswordField();
    confirmpasswordtext = new JPasswordField();
    birthdaytext=new JTextField();
    telephonetext=new JTextField();
    addresstext =  new JTextField();
    citytext=new JTextField();
    countrytext =new JTextField();
    /*
    Initializes the objects of JPanel class.*/
    signuppanel = new JPanel(new GridLayout(12, 6, 7, 7));
    /*
    Adds labels and textfields to the panel.
    */
    signuppanel.add(titlelabel);
    signuppanel.add(titletext);
    signuppanel.add(firstnamelabel);
    signuppanel.add(firstnametext);
    signuppanel.add(lastnamelabel);
    signuppanel.add(lastnametext);
    signuppanel.add(nicknamelabel);
    signuppanel.add(nicknametext);
    signuppanel.add(birthdaylabel);
    signuppanel.add(birthdaytext);
    signuppanel.add(telephonelabel);
    signuppanel.add(telephonetext);
    signuppanel.add(addresslabel);
    signuppanel.add(addresstext);
    signuppanel.add(citylabel);
    signuppanel.add(citytext);
    signuppanel.add(countrylabel);
    signuppanel.add(countrytext);
    signuppanel.add(usernamelabel);
    signuppanel.add(usernametext);
    signuppanel.add(passwordlabel);
    signuppanel.add(passwordtext);
    signuppanel.add(confirmpassword);
    signuppanel.add(confirmpasswordtext);
    /*Sets the size of the addresstext.*/
    addresstext.setPreferredSize(new Dimension(120,28));
    /*Adds signuppanel to the container.*/
    container.add(signuppanel);
    /*
    Initializes the object of JPanel class.*/
    buttonpanel = new JPanel();
    /*
    Initializes the objects of JButton class.
    */
    okbutton = new JButton("OK");
    cancelbutton = new JButton("Cancel");
    /*
    Adds the ActionListener to the objects of the JButton class.
    */
    okbutton.addActionListener(this);
    cancelbutton.addActionListener(this);
    /*
    Adds the action command to the objects of the JButton class.
    */
    okbutton.setActionCommand("ok");
    cancelbutton.setActionCommand("cancel");
    buttonpanel.add(okbutton);
    buttonpanel.add(cancelbutton);
    container.add(buttonpanel, BorderLayout.SOUTH);
    setSize(350, 400);
    setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
    String actioncommand=ae.getActionCommand();
    String sessionstart="";
    if (actioncommand.equals("ok"))
    {
```

```
            if (!socketclass.isConnected)
            {
               socketclass.socketoOpenClass("localhost", 5222, 1000);
            }
            if (jabberclient.getStatus().equals("notconnected")||jabberclient.getStatus().
            equals("waitforreg")||jabberclient.getStatus().equals("waitforauth"))
            {
               sessionstart=socketclass.sendSessionStartMessage();
            }
            String firstname=firstnametext.getText().trim();
            String username=usernametext.getText().trim();
            String password=passwordtext.getText().trim();
            String confirmpassword=confirmpasswordtext.getText().trim();
            String dobstring=birthdaytext.getText().trim();
            String telephonestring=telephonetext.getText().trim();
            String lastnamestring=lastnametext.getText().trim();
            String errormessage="";
            if (firstname.equals(""))
            {
               errormessage="First Name is a required field.\n";
            }
            else
            {
               errormessage=checkfornumber(firstname, "First Name");
            }
            errormessage=errormessage+checkfornumber(lastnamestring, "Last Name");
            if (username.equals(""))
            {
               errormessage=errormessage+"User Name is a required field.\n";
            }
            if (password.equals(""))
            {
               errormessage=errormessage+"Password is a required field.\n";
            }
            if (!confirmpassword.equals(password))
            {
               errormessage=errormessage+"Confirm password and Password should be same.\n";
            }
            errormessage=errormessage+checkDateIsValid(dobstring,"DOB");
            errormessage=errormessage+isValidNumber(telephonestring, "Telephone");
            if (!errormessage.equals(""))
            {
               JOptionPane.showMessageDialog(null,errormessage, "Error", JOptionPane.PLAIN_MESSAGE);
               return ;
            }
            socketclass.setUserNameAndPassword(username, password, "localhost");
            jabberclient.setMessageInToTextBox(sessionstart+"\n"+socketclass.sendRegistration()+"\n"+
            sendVCard());
            jabberclient.setStatus("waitforreg");
            setVisible(false);
         }
         else if(actioncommand.equals("cancel"))
         {
            setVisible(false);
         }
      }
      /*
      checkfornumber-This function is used to check that supplied value contains a numeric value or not
      Parameters: value-object of String class, fieldname-object of String class.
      Return Value: String
      */
      public String checkfornumber(String value, String fieldname)
      {
         String message="";
         for(int i=0;i<value.length();i++)
         {
            if  ("1234567890".indexOf(value.charAt(i))!=-1)
            {
               message="Please enter a valid value for "+fieldname+".\n";
               return message;
            }
         }
         return "";
      }
      /*
      checkDateIsValid-This function is used to check that supplied date is a valid date.
      Parameters: datestring-object of String class, field-object of String class.
      Return Value: String
      */
      public String checkDateIsValid(String datestring,String field)
      {
         String[] spliteddate=datestring.split("/");
         String doberror="";
         int day;
         int month;
         int year;
         System.out.println(spliteddate.length);
         try
```

```
        {
           day=Integer.parseInt(spliteddate[0]);
           month=Integer.parseInt(spliteddate[1]);
           year=Integer.parseInt(spliteddate[2]);
        }
        catch(NumberFormatException nfe)
        {
           doberror="Please enter a valid value for "+field+".\n";
        return doberror;
        }
        if (spliteddate.length!=3)
        {
           doberror="Please enter a valid value for "+field+".\n";
        }
        else if (day>31||day<0||month>12||month<0||year>2000||year<1900)
        {
           doberror="Please enter a valid value for "+field+".\n";
        }
        return doberror;
     }
     /*
     isValidNumber-This function is used to check that supplied value is a valid number or not.
     Parameters: number-object of String class, field-object of String class.
     Return Value: String
     */
     public String isValidNumber(String number,String field)
     {
        String message="";
        for (int i=0;i<number.length();i++ )
        {
           if ("1234567890".indexOf(number.charAt(i))==-1)
           {
              message=message+"Please enter a valid value for "+field+".\n";
              return message;
           }
        }
        return "";
     }
     /*
     sendVCard -This function is used to create XML message of user profile.
     Parameters: N/A
     Return Value: String
     */
     public String sendVCard()
     {
        String vcardstring="";
        vcardstring="<iq type='set' id='1'>\n";
        vcardstring=vcardstring+"<vCard  xmlns='vcard-temp'>\n";
        vcardstring=vcardstring+"<FN>"+firstnametext.getText().trim()+"</FN>\n";
        vcardstring=vcardstring+"<N>\n";
        vcardstring=vcardstring+"<FAMILY>??</FAMILY>\n";
        vcardstring=vcardstring+"<GIVEN>"+firstnametext.getText().trim()+"</GIVEN>\n";
        vcardstring=vcardstring+"<MIDDLE/></N>\n";
        vcardstring=vcardstring+"<NICKNAME>"+nicknametext.getText().trim()+"</NICKNAME>\n";
        vcardstring=vcardstring+"<URL>??</URL>\n";
        vcardstring=vcardstring+"<BDAY>"+birthdaytext.getText().trim()+"</BDAY>\n";
        vcardstring=vcardstring+"<ORG>\n<ORGNAME>??</ORGNAME>\n";
        vcardstring=vcardstring+"<ORGUNIT/>\n";
        vcardstring=vcardstring+"</ORG>\n";
        vcardstring=vcardstring+"<TITLE>"+titletext.getText().trim()+"</TITLE>\n";
        vcardstring=vcardstring+"<ROLE>??</ROLE>\n";
        vcardstring=vcardstring+"<TEL>\n<VOICE/>\n<HOME>"+
        telephonetext.getText().trim()+ "</HOME>\n</TEL>\n";
        vcardstring=vcardstring+"<TEL>\n<FAX/>\n<HOME/>\n</TEL>\n";
        vcardstring=vcardstring+"<TEL>\n<MSG/>\n<HOME/>\n</TEL>\n";
        vcardstring=vcardstring+"<ADR>\n<HOME/>\n<EXTADD/>\n<STREET/>\n";
        vcardstring=vcardstring+"<LOCALITY>"+citytext.getText().trim()+"</LOCALITY>\n";
        vcardstring=vcardstring+"<REGION>??</REGION>\n";
        vcardstring=vcardstring+"<PCODE>??</PCODE>\n";
        vcardstring=vcardstring+"<COUNTRY>"+countrytext.getText().trim()+"</COUNTRY>\n";
         vcardstring=vcardstring+"</ADR>\n<TEL>\n<VOICE/>\n<WORK/>\n</
         TEL>\n<TEL>\n<FAX/>\n<WORK/>\n</TEL>\n";
        vcardstring=vcardstring+"<TEL>\n<MSG/>\n<WORK/>\n</TEL>\n";
        vcardstring=vcardstring+"<ADR>\n<WORK/>\n<EXTADD/>\n<STREET/>\n<
        LOCALITY>??</LOCALITY>\n";
        vcardstring=vcardstring+"<REGION>??</REGION>\n";
        vcardstring=vcardstring+"<PCODE>??</PCODE>\n";
        vcardstring=vcardstring+"<COUNTRY>??</COUNTRY>\n";
        vcardstring=vcardstring+"</ADR>\n";
        vcardstring=vcardstring+"<EMAIL>\n<INTERNET/>\n <PREF/>??</EMAIL>\n";
        vcardstring=vcardstring+"</vCard>\n</iq>\n";
        return vcardstring;
     }
}
```

Download this listing.

In the above code, the constructor of the SignUp class takes an object of the SocketClass class and an object of the JabberClient class as input parameters. This allows an end user to invoke the methods of the SocketClass and JabberClient classes.

The methods defined in Listing 2-2 are:

- actionPerformed: Acts as an event listener and activates an appropriate method based on the action the end user performs.
- Checkfornumber: Checks whether or not the information specified by the end user contains a numeric value.
- checkDateIsValid: Checks whether or not the date entered by the end user is a valid date.
- isValidNumber: Checks whether or not the value entered by the end user is numeric.
- sendVcard: Creates an XML message for the profile of an end user.

In Listing 2-2, the vCard format is set in the sendVcard method. The vCard format is a standard format for electronic business card data, which is useful for exchanging personal directory data across the Internet. The <vCard xmlns='vcard-temp'> XML tag sets the Vcard format for end users.

The SignUp.java file generates the Sign Up window, as shown in Figure 2-6:

Figure 2-6: The Sign Up Window

Creating the Login Window

The UserLogin.java file creates the user interface to log on as an existing user for the Connector application. Listing 2-3 shows the contents of the UserLogin.java file:

Listing 2-3: The UserLogin.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*
class UserLogin - This class is used to create a GUI for user login.
Constructor:
UserLogin-This constructor creates GUI.
Methods:
actionPerformed - This method is called when end user clicks on send button.
*/
public class UserLogin extends JDialog implements ActionListener
  {
      Container contentpane;
      /*
      Declares object of JPasswordField class.
      */
      JPasswordField passwordtextfield;
      /*
      Declares objects of JTextField class.
      */
      JTextField nametextfield;
      JTextField servernametextfield;
      /*Declares object of JButton class.*/
      JButton submitbutton;
      /*Declares objects of String class.*/
      String servername;
      String username;
      String password;
      String resource;
      String registrationstring="";
      /*Declares objects of JPanel class.*/
      JPanel note;
      JPanel combine;
      /*
      Declares objects of JabberClient class.
      */
      JabberClient jabberclient;
      /*
      Declares objects of SocketClass class.
      */
      SocketClass socketclass;
      boolean submitclickd=false;
      public UserLogin (SocketClass socketclass,JabberClient jabberhandler)
      {
         setTitle("Login Window");
         this.socketclass=socketclass;
         contentpane=getContentPane();
         jabberclient=jabberhandler;
         addWindowListener(new WindowAdapter()
         {
            public void windowClosing(WindowEvent we)
            {
               setVisible(false);
            }
         });
         /*
         Initializes the objects of JTextField class.
         */
         nametextfield=new JTextField();
         servernametextfield=new JTextField();
         /*
         Initializes the object of JPasswordField class.
         */
         passwordtextfield=new JPasswordField();
         /*
         Initializes the object of JButton class.
         */
         submitbutton=new JButton("Submit");
         /*
         Initializes the objects of JPanel class.
         */
```

```
    combine=new JPanel();
    note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
    submitbutton.setActionCommand("submit");
    submitbutton.addActionListener(this);
    submitbutton.setMnemonic('s');
    resource="Home";
    /*
    Declares and Initializes the objects of JPanel class.
    */
    JPanel controlpanel=new JPanel();
    JPanel lablepanel=new JPanel();
    /*
    Declares and Initializes the objects of JLabel class.
    */
    JLabel room= new JLabel("User Name");
    JLabel name= new JLabel("Password ");
    /*
    Sets the GridLayout of the controlpanel.
    */
    controlpanel.setLayout(new GridLayout(4, 2, 3, 3));
    controlpanel.setBorder(BorderFactory.createTitledBorder("Login Information"));
    /*
    Adds the labels and textfields to controlpanel.
    */
    controlpanel.add(room);
    controlpanel.add(nametextfield);
    controlpanel.add(name);
    controlpanel.add(passwordtextfield);
    controlpanel.add(serverl);
    controlpanel.add(servernametextfield);
    /*
    Declares and Initializes the objects of JLabel class.
    */
    JLabel heading=new JLabel("Login Window",JLabel.CENTER);
    /*Adds the heading to lablepanel.*/
    lablepanel.add(heading,BorderLayout.NORTH);
    /*Sets the font to the heading.*/
    heading.setFont(new Font("verdana", 1, 12));
    /*Adds the lablepanel to contentpane.*/
    contentpane.add(lablepanel,BorderLayout.NORTH);
    contentpane.add(controlpanel,BorderLayout.CENTER);
    /*
    Declares and Initializes the objects of JPanel class.
    */
    JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
    JPanel bp=new JPanel(new FlowLayout());
    bp.add(submitbutton);
    buttonpanel.add(note);
    buttonpanel.add(bp);
    contentpane.add(buttonpanel,BorderLayout.SOUTH);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(5, 5, 300, 240);
    submitbutton.addActionListener(this); show();
    getRootPane().setDefaultButton(submitbutton);
  }
public void actionPerformed(ActionEvent ae)
  {
    String actioncommand=ae.getActionCommand();
    servername=servernametextfield.getText().trim();
    username=nametextfield.getText().trim();
    password=passwordtextfield.getText().trim();
    if (username.equals(""))
    {
       JOptionPane.showMessageDialog(null, "User Name is a required
       field.", "Error", JOptionPane.PLAIN_MESSAGE);
       return;
    }
    if (password.equals(""))
    {
       JOptionPane.showMessageDialog(null,"Password is a required
       field.","Error",JOptionPane.PLAIN_MESSAGE);
       return;
    }
    if (actioncommand.equals("submit"))
    {
       String sessionstart="";
       if (!submitclickd)
       {
          submitclickd=true;
          if (servername.equals(""))
          { JOptionPane.showMessageDialog(null, "Server name is a required
          field.","Error",JOptionPane.PLAIN_MESSAGE);
          return;
       }
       if (jabberclient.getServerName().equals(servername))
       {
       }
       else
```

```
            {
               if (socketclass.isConnected)
               {
                  socketclass.closeConnection();
               }
               socketclass.socketoOpenClass(servername, 5222, 1000);
               if ((!socketclass.isConnected))
               {
                  setVisible(false);
                  return;
               }
            }
            if (jabberclient.getStatus().equals("notconnected")||jabberclient.getStatus().
            equals("waitforreg")||jabberclient.getStatus().equals("waitforauth"))
            {
               sessionstart=socketclass.sendSessionStartMessage();
            }
            jabberclient.setUserNameAndPassword(username,password,servername);
            socketclass.setUserNameAndPassword(username,password,servername);
            String authstring=socketclass.sendAuthorized();
            jabberclient.setMessageInToTextBox(sessionstart+"\n"+authstring);
            jabberclient.setStatus("waitforauth");
            setVisible(false);
            }
            else
            {
               submitclickd=false;
            }
         }
      }
   }
```

Download this listing.

In the above code, the constructor of the UserLogin class takes an object of the SocketClass class and an object of the JabberClient class as input parameters. This allows an end user to invoke the methods of the SocketClass and JabberClient classes. The UserLogin class defines the actionPerformed() method. The actionPerformed() method acts as an event listener and activates an appropriate method, based on the action an end user performs. End users invoke the actionPerformed() method by selecting any option from the User Login window.

Select File-> Login to log on as an existing end user. The UserLogin.java file generates the Login window, as shown in Figure 2-7:

Figure 2-7: The Login Window

PREVIOUS NEXT

Creating the Socket Class to Send and Receive Messages

The SocketClass.java file opens the socket to establish a connection with the Jabber server, to send and receive messages between an end user and the Jabber server. Listing 2-4 shows the contents of the SocketClass.java file:

Listing 2-4: The SocketClass.java File

```
/*Imports required util classes*/
import java.util.*;
/*Imports required io classes*/
import java.io.*;
/*Imports required net classes*/
import java.net.*;
/*Imports required swing classes*/
import javax.swing.*;
/*
class SocketClass-This class is used to open a socket and sends and receives Jabber messages.
Constructor:
SocketClass-This constructor creates GUI.
sendername: This method is called to substracte sender name from input message.
checkForError: This method is called to check whether the input string contains an error message or
not.
sendXMLToJabber: This method is called to send XML message to Jabber server.
closeConnection: This method is called to close client socket.
run: This method is called to listen server response sended by jabber server.
*/
class SocketClass implements Runnable
{
   /*Declares objects of Vector class.*/
   Vector tagvactor;
   /*Declares objects of PrintWriter class.*/
   PrintWriter out = null;
   /*
   Declares objects of BufferedReader class.
   */
   BufferedReader in = null;
   Socket clientsocket;
   Thread inputmessagethread;
   /*Declares objects of String class.*/
   String errortype;
   String resource="Home";
   String username="";
   String password="";
   String sendername="";
   String  recevername="";
   String registrationstring;
   String servermessage="";
   String ipaddress="";
   String endtagstr="";
   /*Declares object of UserLogin class.*/
   UserLogin userloginhandler;
   /*Declares object of JabberClient class.*/
   JabberClient jabberclient;
   int portno;
   int wait=1000;
   public boolean isConnected=false;
   public SocketClass(JabberClient jc)
   {
      jabberclient=jc;
   }
   /*
   isConnected: This method is called retrieve socket state.
   Parameter: N/A
   Return Value: boolean
   */
   public boolean isConnected()
   {
      return isConnected;
   }
   /*
   setUserLoginHandler: This method is called to set handler of the object of UserLogin class.
   Parameter: userlogin-Object of UserLogin  class.
   Return Value: N/A
   */
   public void setUserLoginHandler(UserLogin userlogin)
   {
      userloginhandler=userlogin;
   }
   /*
   socketoOpenClass: This method is called to open a socket and gets input and output streams.
   Parameter: ipaddress-Object of String class, portno, wait
   Return Value: N/A
   */
   public void socketoOpenClass(String ipaddress, int portno, int wait)
```

```
{
   this.ipaddress=ipaddress;
   this.portno=portno;
   this.wait=wait;
   openPort(ipaddress,portno,wait);
   if (clientsocket!=null)
   {
      isConnected=true;
   }
   else
   { JOptionPane.showMessageDialog(null, "Server not found.", "Error", JOptionPane.PLAIN_MESSAGE)
   return;
   }
   try
   {
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   inputmessagethread=new Thread(this);
   inputmessagethread.start();
}
/*
sendSessionStartMessage: This method is called to create a XML message string.
Parameter: N/A
Return Value: String
*/
public String sendSessionStartMessage()
{
   String sessionStratString;
   sessionStratString="<?xml version=\"1.0\"   encoding=\"UTF-8\" ?>\n";
   sessionStratString=sessionStratString+" <stream:stream";
   sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
   sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   return sessionStratString;
}
/*
sendAuthorized:  This method is called to create a XML message string.
Parameter: ipaddress - Object of String class, portno, wait
Return Value: N/A
*/
public String sendAuthorized()
{
   String authentication;
   authentication="<iq type=\"set\" id=\"1\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+" <username>"+username+"</username>";
   authentication=authentication+" <password>"+password+"</password>";
   authentication=authentication+" <resource>"+resource+"</resource> ";
   authentication=authentication+" </query> ";
   authentication=authentication+"</iq>";
   return authentication;
}
/*
sendRegistration:  This method is called to create a XML message string.
Parameter: ipaddress - Object of String class, portno, wait
Return Value: N/A
*/
public String sendRegistration()
{
   String registrationstr="";
   registrationstr="<iq type='set' to='"+username+".localhost' id='1'>";
   registrationstr=registrationstr+"<query xmlns='jabber:iq:register'>";
   registrationstr=registrationstr+"<username>"+username+"</username>";
   registrationstr=registrationstr+"<password>"+password+"</password>";
   registrationstr=registrationstr+"</query></iq>";
   return registrationstr;
}
/*
setUserNameAndPassword:  This method is called to set user name, password and servername
Parameter: username - Object of String class, password - Object of  String class, servername -
Object of  String class.
Return Value: N/A
*/
public void setUserNameAndPassword(String username, String password, String servername)
{
   this.username=username;
   this.password=password;
   this.ipaddress=servername;
}
/*
openPort:  This method is called to open a client socket.
Parameter: ipaddress - Object of String class, portno, timeinsec
```

```
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      SocketOpener sc=new SocketOpener(); Clientsocket=sc.openSocket(ipaddress,portno,timeinsec);
   }
}
/*
run:  This method is called to listen server   response sended by jabber server
Parameter: N/A
Return Value: N/A
*/
public void run()
{
   int i=0;
   String errororwarning="";
   String errorstring="";
   String inputstring="";
   boolean starttag=false;
   boolean endtag=false;
   boolean startwritting=false;
   String messagebody="";
   String tagentity="";
   Vector tagvactor;
   tagvactor=new Vector();
   int vactorindex=0;
   int starttagsing=0;
   while (true)
   {
      try
      {
         i=in.read();
         if (i==-1)
         {
            break;
         }
         else
         {
            inputstring=inputstring+(char)i;
            ervermessage=servermessage+(char)i;
            if ((char)i=='<')
            {
               starttag=true;
               starttagsing=1;
               stagentity="";
            }
            else
            {
               if ((char)i=='/' && starttag==true)
               {
                  if (starttagsing==1)
                  {
                     starttag=false;
                     endtag=true;
                     startwritting=false;
                     vactorindex=vactorindex-1;
                     if (vactorindex>=0) agvactor.removeElementAt(vactorindex);
                  }
                  tagentity=tagentity+(char)i;
               }
               else
               {
               starttagsing=0;
               if((char)i=='>')
            {
               jabberclient.setServerResponse(servermessage);
               servermessage="";
               if (endtag&&endtagstr.equals("stream:stream"))
               {
                  jabberclient.setStatusLabel("Logged Off");
                  jabberclient.setStatus("notconnected");
                  jabberclient.disableAndEnableMenuItems(true);
               }
               if (endtag)
               {
                  endtagstr="";
                  endtag=false;
               }
               if (starttag)
               {
                  if (tagentity.endsWith("/"))
```

```
                        { tagentity=tagentity.substring(0,tagentity.leng  th()-1);
                        }
                        sendername(tagentity);
                        if (tagentity.equals("remove"))
                        {jabberclient.setStatusLabel("Not Connected");
                        jabberclient.setStatus(""); jabberclient.disableAndEnableMenuItems(true);
                        } errorstring=checkForError(tagentity);
                        if (errorstring.equals("unauthorized"))
                        {
                           jabberclient.setStatus(""); JOptionPane.showMessageDialog(null, "User name o
                           password is invalid.", "Error",JOptionPane.PLAIN_MESSAGE);
                        }
                        if (errorstring.equals("user exists"))
                        {
                           if (jabberclient.getStatus().equals("waitforreg"))
                           {
                              jabberclient.setStatus("");
                              JOptionPane.showMessageDialog(null,"User ID already exists, Please choose
                              other User ID.", "Error", JOptionPane.PLAIN_MESSAGE);
                           }
                        }
                        if (errorstring.equals("User not exists.")&&messagebody.equals(""))
                        {
                           jabberclient.setStatusLabel(" Room does not exist");
                           JOptionPane.showMessageDialog(null,"Room does not
                           exist.","Error",JOptionPane.PLAIN_MESSAGE);
                        }
                     else if (errorstring.equals("User notexists.")&&(!messagebody.equals("")))
                     {
                        messagebody="";
                        JOptionPane.showMessageDialog(null,"User dose not
                        exists.","Error",JOptionPane.PLAIN_MESSAGE);
                     }
                     if (tagentity.equals("feature var='http://jabber.org/protocol/muc'"))
                     {
                        sendXMLToJabber("<presence from='"+username+"@conference."+ipaddress+"/Home'
                        to='"+jabberclient.getRoomName()+"@conference."+ipaddress+"/"+username+"'/>");
                     }
                     if (tagentity.indexOf("type='result'")!=-1)
                     {
                        String status=jabberclient.getStatus();
                        if (status.equals("waitforauth"))
                        {
                           jabberclient.setStatusLabel("User authenticated");
                           jabberclient.setStatus("auth");
                           jabberclient.disableAndEnableMenuItems(false);
                        }
                        if (status.equals("waitforreg"))
                        {jabberclient.setStatusLabel("Registered");
                        jabberclient.setStatus("reg");
                        }
                     }
                     startwritting=true;
                     tagvactor.insertElementAt(tagentity,vactorindex);
                     vactorindex=vactorindex+1;
                     starttag=false;
                     }
                  }
                  else
                  {
                     if (startwritting==true&&tagentity.trim().equals("body"))
                     {
                        messagebody=messagebody+(char)i;
                     }
                     if (starttag)
                     {
                        tagentity=tagentity+(char)i;
                     }
                     if (endtag)
                     {
                        endtagstr=endtagstr+(char)i;
                     }
                  }
               }
            }
         }
      }
     catch(IOException ie)
     {
     }
   }
   isConnected=false;
   }
/*
closeConnection: This method is called to close client socket.
Parameter: N/A
```

```
Return Value: N/A
*/
public void closeConnection()
{
   try
   {
      out.close();
      in.close();
      clientsocket.close();
   }
   catch (IOException e)
   {
      System.out.println(e);
   }
   isConnected=false;
}
/*
sendXMLToJabber: This method is called to send XML message to Jabber server.
Parameter: outputmessage - Object of String class.
Return Value: N/A
*/
public void sendXMLToJabber(String outputmessage)
{
   String[] tokenizerstring=outputmessage.split("\n");
   for (int i=0;i<tokenizerstring.length;i++ )
   {
      try
      {
         out.println(tokenizerstring[i]);
         out.flush();
      }
      catch(Exception e)
      {
         System.out.println("error");
         return;
      }
      out.flush();
   }
}
/*
checkForError:  This method is called to check whether the input string contains an error message or
not.
Parameter: tagentity - Object of String class.
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if ( tagentity.equals("error code='403' type='auth'"))
   {
      return "unauthorized";
   }
   if (tagentity.equals("error  code='404' type='cancel'"))
   {
      return "User not exists.";
   }
   if ((tagentity.indexOf("code='401'")>1)||(tagentity.indexOf("code=\"401\"")>1))
   {
      return "unauthorized";
   }
   if (tagentity.equals("error code='409' type='cancel'"))
   {
      return "user exists";
   }
   if ((tagentity.indexOf("code='409'")>1)||(tagentity.indexOf("code=\"409\"")>1))
   {
      return "user exists";
   }
   return error;
}
/*
sendername: This method is called to  substracte sender name from input message.
Parameter: tagentity - Object of String class.
Return Value: N/A
*/
public void sendername(String tagentity)
{
   if (tagentity.startsWith("message"))
   {
      sendername="";
      recevername=""; if(tagentity.indexOf("from=")>0&&tagentity.indxOf("to=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-2);
      }
      if (tagentity.indexOf("to=")>0&&tagentity.indexOf("type=")>0)
      { recevername=tagentity.substring(tagentity.indexOf("to=")+4,tagentity.indexOf("type=")-2);
      }
```

```
    }
  }
}
class SocketOpener
{
   private Socket socket;
   public Socket openSocket(String hostip,int portnumber, int timeinsec)
   {
      try
      {
         socket=new Socket(hostip,portnumber);
      }
      catch(IOException ie)
      {
         System.out.println("IE:::::::::: " +ie);
      }
      return getSocket();
   }
   public SocketOpener()
   {
   }
   public Socket getSocket()
   {
      return socket;
   }
};
```

Download this listing.

In the above code, the constructor of the SocketClass class takes the object of the JabberClient class to invoke the methods of the JabberClient class.

The methods defined in Listing 2-4 are:

- `sendername()`: Retrieves the sender's name from the input message received from the Jabber server.
- `checkForError()`: Checks whether or not the input string contains an error message.
- `sendXMLToJabber()`: Sends an XML message to the Jabber server.
- `closeConnection()`: Closes the client socket from the Jabber server.
- `run()`: Listens to the server response sent by the Jabber server.
- `isConnected()`: Determines whether or not a client socket is connected to the Jabber server.
- `setUserLoginHandler()`: Sets the handler of the object of the UserLogin class.
- `socketoOpenClass()`: Calls the openPort() method to open a client socket and gets input and output streams to send and receive messages.
- `sendSessionStartMessage()`: Creates an XML message string to initialize the session of an end user.
- `sendAuthorized()`: Creates an XML message string to authenticate the end users.
- `sendRegistration()`: Creates an XML message string to sign up for a new account with the Jabber server.
- `setUserNameAndPassword()`: Sets the user name, password, and Jabber server name.
- `openPort()`: Opens a client socket by using the IP address and port number of the Jabber server.

Sending Messages

The MessageClass.java file creates the user interface that allows the end user to specify the user ID and message. The MessageClass class sends the message to the user ID specified by the end user. Listing 2-5 shows the contents of the MessageClass.java file:

Listing 2-5: The MessageClass.java File

```
/*Imports required swing classes.*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes.*/
import java.awt.*;
import java.awt.event.*;
/*
class MessageClass-This class is used to create a GUI for sending messages.
Constructor:
MessageClass-This constructor creates GUI.
Methods:
actionPerformed-This method is called when end user clicks on send button.
*/
class MessageClass extends JFrame implements ActionListener
{
   /*Declares objects of JLabel class.*/
   JLabel useridlabel;
   JLabel messagelabel;
   JLabel lastmessagelabel;
   JLabel firstmessagelabel;
   /*Declares object of JTextField class.*/
   JTextField useridtext;
   /*Declares object of JTextArea class.*/
   JTextArea messagetextpane;
   /* Declares object of JScrollPane class.*/
   JScrollPane messagescrollpane;
   /*Declares objects of JButton class.*/
   JButton closebutton;
   JButton sendbutton;
   /* Declares objects of JPanel class.*/
   JPanel usermessagepanel;
   JPanel messagepanel;
   JPanel buttonpanel;
   /*Declares objects of JabberClient class.*/
   JabberClient jabberclient;
   public MessageClass(JabberClient jabberclient)
   {
      super("Message Window");
      Container contentpane=getContentPane();
      this.jabberclient=jabberclient;
      contentpane.add(new JLabel("  "),BorderLayout.WEST);
      /*
      Initializes the object of JLabel class
      */
      useridlabel=new JLabel("User ID");
      /*
      Initializes the object of JTextField class
      */
      useridtext=new JTextField();
      /*
      Initializes the object of JLabel class
      */
      messagelabel=new JLabel("Message");
      /*
      Initializes the object of JTextArea class
      */
      messagetextpane=new JTextArea();
      /*
      Initializes the object of JScrollPane class
      */
      messagescrollpane=new JScrollPane(messagetextpane, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
      JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
      /*
      Initializes the object of JPanel class
      */
      usermessagepanel=new JPanel();
      /*
      Initializes the objects of JButton class.
      */
      closebutton=new JButton("Close");
      sendbutton=new JButton("Send Message");
      /*
      Adds the ActionListener to the object of the JButton class.
      sendbutton.addActionListener(this);
      closebutton.addActionListener(this);
      /*
```

```
      Sets the ActionCommand of the object of JButton class.
      */
      sendbutton.setActionCommand("send");
      closebutton.setActionCommand("close");
      /*
      Declare and initialize the object of GridLayout class.
      */
      GridLayout usermessagegridlayout=new GridLayout(2, 2, 5, 5);
      /*
      Set the Gridlayout to usermessagepanel object.
      */
      usermessagepanel.setLayout(usermessagegridlayout);
      /*
      Initializes the objects of JLabel class.
      */
      firstmessagelabel=new JLabel("Message", JLabel.RIGHT);
      usermessagepanel.add(firstmessagelabel);
      /*
      Initializes the objects of JLabel class.
      */
      lastmessagelabel=newJLabel("Window", JLabel.LEFT);
      firstmessagelabel.setFont(newFont("Verdana", Font.BOLD, 12));
      lastmessagelabel.setFont(new Font("Verdana", Font.BOLD, 12));
      usermessagepanel.add(lastmessagelabel);
      /*
      Adds useridlabel to usermessagepanel
      */
      usermessagepanel.add(useridlabel);
      /*Adds useridtext to usermessagepanel*/
      usermessagepanel.add(useridtext);
      /*
      Initializes the objects of JPanel class.
      */
      messagepanel=new JPanel();
      /*
      Declare and initialize the object of GridLayout class.
      */
      GridLayout messagegridlayout=new GridLayout(1, 2, 5, 5);
      /*
      Set the Gridlayout to messagepanel object.
      */
      messagepanel.setLayout(messagegridlayout);
      /*Adds messagelabel to messagepanel*/
      messagepanel.add(messagelabel);
      /*
      Adds messagescrollpane to messagepanel
      */
      messagepanel.add(messagescrollpane);
      /*
      Declare and initialize the object of JPanel  class.
      */
      JPanel toppanel=new JPanel(new  GridLayout(2,1,5,5));
      /*Adds usermessagepanel to toppanel*/
      toppanel.add(usermessagepanel);
      /* Adds messagepanel to toppanel*/
      toppanel.add(messagepanel);
      /*Adds toppanel to contentpane*/contentpane.add(toppanel,BorderLayout.CENTER);
      buttonpanel=new JPanel(new GridLayout(1, 2, 0, 0));
      buttonpanel.add(new JLabel(" "));
      /*
      Declare and initialize the object of JPanel class.
      */
      JPanel subbuttonpanel=new JPanel();
      /* Adds buttons to subbuttonpanel*/
      subbuttonpanel.add(sendbutton);
      subbuttonpanel.add(closebutton);
      buttonpanel.add(subbuttonpanel);
      contentpane.add(buttonpanel,BorderLayout.SOUTH);
      useridtext.setPreferredSize(new  Dimension(80,20));
      setSize(400,200);
      setVisible(true);
   }
   public void actionPerformed(ActionEvent ae)
   {
      String actioncommand=ae.getActionCommand();
      String userid=useridtext.getText().trim();
      String message=messagetextpane.getText().trim();
      String errorstring="";
      if (actioncommand.equals("send"))
      {
         if (userid.equals(""))
         {
            errorstring="User ID is a required field.\n";
         }
         if (message.equals(""))
         {
            errorstring=errorstring+"Message field cannot be left blank.";
         }
```

```
            if (!errorstring.equals(""))
            {
               JOptionPane.showMessageDialog(null,errorstring,"Error",JOptionPane.PLAIN_MESSAGE);
               return;
            }
            String username=jabberclient.getUserName();
            String servername=jabberclient.getServerName();
            String sendmessagestring="";
            sendmessagestring="<message type='chat' to='"+userid+"/Home'
            from='"+username+"@"+servername+"/Home'>\n";
            sendmessagestring=sendmessagestring+"<body>\n"+message+"\n";
            sendmessagestring=sendmessagestring+"</body>\n</message>";
            jabberclient.setMessageInToTextBox(sendmessagestring);
         }
         setVisible(false);
      }
}
```

Download this listing.

In the above code, the constructor of the MessageClass class takes an object of the JabberClient class as an input parameter. This allows an end user to invoke the methods of the JabberClient class.

The methods defined in Listing 2-5 are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.
- `getText()`: Retrieves the text message specified by the end user.

Select File-> Send Message to send the message to the specified end user. The Message Window appears, as shown in Figure 2-8:

Figure 2-8: The Message Window

The MessageClass.java file allows end users to specify the User ID and Message in the Message Window.

Creating the Chat Room Interface

The ChatRoom.java file creates the user interface that allows end users to enter the chat room by specifying the name of the chat room for the Connector application. Listing 2-6 shows the contents of the ChatRoom.java file:

Listing 2-6: The ChatRoom.java File

```
/*Imports required swing classes*/
import javax.swing.*;
import javax.swing.event.*;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*
class ChatRoom-This class is used to create a GUI to enter in a existing room
Constructor:
ChatRoom-This constructor creates GUI.
Methods:
*/
class ChatRoom extends JFrame implements ActionListener
{
   /*Declares objects of JLabel class.*/
   JLabel roomnamelabel;
   JLabel pageheadelabel;
   /*Declares object of JTextField class.*/
   JTextField roomnametext;
   /* Declares objects of JPanel class.*/
   JPanel chatroompanel;
   JPanel buttonpanel;
   /* Declares objects of JButton class.*/
   JButton okbutton;
   JButton cancelbutton;
   /* Declares object of String class.*/
   String roomnamestring;
   /* Declares object of JabberClient class.*/
   JabberClient jabberclient;
   public ChatRoom(JabberClient jabberclient)
   {
      super("Enter Chat Room");
      Container contentpane=getContentPane();
      this.jabberclient=jabberclient;
      /* Initializes object of JLabel class.*/
      pageheadelabel=new JLabel("Enter Chat Room", JLabel.CENTER);
      /*
      Sets the font to the object of JLabel  class.
      */
      Pageheadelabel.setFont(new Font("Verdana", Font.BOLD, 12));
      /* Adds the label to the contentpane.*/
      contentpane.add(pageheadelabel, BorderLayout.NORTH);
      /*
      Initializes the object of JLabel  class.*/
      roomnamelabel=new JLabel("Room Name");
      /*
      Initializes the object of JTextField class.
      */
      roomnametext=new JTextField();
      /*
      Initializes the objects of JPanel class.*/
      chatroompanel=new JPanel();
      buttonpanel=new JPanel();
      /*
      Sets the size of the objects of JLabel class.
      */
      roomnamelabel.setPreferredSize(new Dimension(120, 23));
      roomnametext.setPreferredSize(new Dimension(170, 23));
      /*
      Declare and initialize the object of FlowLayout class.
      */
      FlowLayout chatroomflowlayout=new FlowLayout();
      /*
      Set the FlowLayout to chatroompanel object.
      */
      chatroompanel.setLayout(chatroomflowlayout);
      /*Adds the label to the panel.*/
      chatroompanel.add(roomnamelabel);
      /*Adds the roomnametext to the panel.*/
      chatroompanel.add(roomnametext);
      contentpane.add(chatroompanel);
      /*
      Initializes the objects of JButton class.*/
      okbutton=new JButton("OK");
      cancelbutton=new JButton("Cancel");
      /*
      Adds the okbutton to the buttonpanel.
```

```
        */
        buttonpanel.add(okbutton);
        /* Sets the size of the buttons.*/
        okbutton.setPreferredSize(new Dimension(80, 23));
        cancelbutton.setPreferredSize(new Dimension(80, 23));
        /*
        Adds the ActionListener to the buttons.
        */
        okbutton.addActionListener(this);
        cancelbutton.addActionListener(this);
        okbutton.setActionCommand("ok");
        cancelbutton.setActionCommand("cancel");
        buttonpanel.add(cancelbutton);
        contentpane.add(buttonpanel,BorderLayout.SOUTH);
        setSize(320,140);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String actioncommand=ae.getActionCommand();
        String roomquerystring="";
        if (actioncommand.equals("ok"))
        {
            roomnamestring=roomnametext.getText().trim();
            if (roomnamestring.equals(""))
            {
                JOptionPane.showMessageDialog(null,"Room name is a required
                field.","Error",JOptionPane.PLAIN_MESSAGE);
                return;
            }
            roomquerystring="<iq from='"+jabberclient.getUserName()+"@"+jabberclient.
            getServerName()+"/Home' id='1' to='"+roomnamestring+"@conference."+jabberclient.
            getServerName()+"' type='get'>";
            roomquerystring=roomquerystring+"<query
            xmlns='http://jabber.org/protocol/disco#info'/></iq>";
            jabberclient.setMessageInToTextBox(roomquerystring);
            jabberclient.setRoomName(roomnamestring);
        }
        setVisible(false);
    }
}
```

Download this listing.

In the above code, the constructor of the ChatRoom.java class takes an object of the JabberClient class as an input parameter to invoke the methods of the JabberClient class. The ChatRoom class allows you to create the Enter Chat Room window to enter the name of the chat room in the Connector application.

The methods defined in Listing 2-6 are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.
- `getText()`: Requests the end users to specify the name of the chat room.

Select File-> Enter Chat Room to enter the chat room. The ChatRoom.java generates the Enter Chat Room window, as shown in Figure 2-9:

Figure 2-9: The Enter Chat Room Window

Unit Testing

To test the Connector application:

1. Download and install the free implementation of Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Copy the JabberClient.java, ChatRoom.java, MessageClass.java, SignUp.java, User Login.java, and SocketClass.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder where you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

4. To run the Connector application, specify the following command at the command prompt:

   ```
   java JabberClient
   ```

5. The user interface of the Connector application appears. Select File-> Signup to sign up for a new account in the Connector application. The Sign Up window appears.

6. Fill up the fields of the Sign Up window, as shown in Figure 2-10:

Figure 2-10: Specifying Information in the Sign Up Window

7. Click OK to generate the XML message.

8. Click Submit to create a new account in the Jabber server. The Jabber server returns a confirmation response, as shown in Figure 2-11:

Figure 2-11: The Signup Confirmation

9. Select File-> Login to log on to the Connector application as an existing user. The Login window appears.
10. Fill up the login information, as shown in Figure 2-12:

Figure 2-12: Specifying Information in the Login Window

11. Click the Submit button. The confirmation XML message appears, as shown in Figure 2-13:

Figure 2-13: The Login Confirmation Message

The Client Request text area shows the request for logging on to the Jabber server in the form of an XML stream and the Server Response text area shows the XML stream after successfully logging on to the Jabber server

12. Select File-> Logoff and click the Submit button to log off from the Connector application. The Logoff confirmation XML message appears.
13. Select File-> Unsubscribe and click the Submit button to unsubscribe from the Jabber server. The Unsubscribe confirmation XML message appears.
14. Select File-> Send Message to send a message to another end user connected to the Jabber server. The Send Message window appears.
15. Specify the text for the message in the Send Message window, as shown in Figure 2-14:

Figure 2-14: Specifying the Message Text

16. Click the Submit button. The confirmation message appears, as shown in Figure 2-15:

Figure 2-15: Message Confirmation

The Client Request text area shows the request, which includes a text message and the Server Response text area shows the XML stream after successfully sending the message to the Jabber server

17. Select File-> Enter Chat Room to enter a particular chat room.
18. Fill the name of the chat room in the Enter Chat Room window, as shown in Figure 2-16:

Figure 2-16: Specifying the Chat Room Name

19. Click the Submit button. The confirmation message appears, as shown in Figure 2-17:

Figure 2-17: Enter Chat Room Confirmation

The Client Request text area shows the request to enter the chat room specified by an end user and the Server Response text area shows the XML stream after successfully entering the chat room.

◄ PREVIOUS NEXT ►

Chapter 3: Creating a Chat Room Application

Download CD Content

The Jabber protocol helps you develop chat room applications hosted on the Jabber server. End users can join a chat room to send and receive messages and view a list of other end users of that chat room.

This chapter describes how to develop the Chat Room application by using the Jabber protocol. The application allows end users to create or join a chat room and send messages to other end users of an active chat room.

Architecture of the Chat Room Application

The Chat Room application uses the following files:

- LoginGUI.java: Creates the Login window for the Chat Room application to allow end users to establish a connection with the Jabber server. This file allows end users to login as existing end users or create a new account.
- MainGUI.java: Allows end users to view a list of existing chat rooms and select any chat room from the given list.
- ChatRoom.java: Allows end users to enter a chat room by specifying a chat room name and a user name to participate in the group chat.
- RoomInformation.java: Allows end users to create a chat room by specifying a chat room name and a user name to initiate the group chat.
- GroupChat.java: Allows end users to participate in a group chat.
- SocketOpener.java: Opens a socket to send and receive messages through the Jabber server.

Figure 3-1 shows the architecture of the Chat Room application:

Figure 3-1: Architecture of the Chat Room Application

The LoginGUI.java file creates the Login window that contains various labels, text boxes, and the Submit button. This file allows end users to login as existing users or register for a new account.

When end users enter login information and click the Submit button on the Login window, the LoginGUI.java file calls the MainGUI.java file. The MainGUI.java file allows end users to create a chat room or join an existing chat room. The MainGUI.java file allows end users to select a chat room from the list of existing chat rooms provided by the user interface of the Chat Room application.

When end users select File->Create New Chat Room on the user interface of the Chat Room application, the MainGUI.java file calls the RoomInformation.java file to create a chat room. The RoomInformation.java file allows end users to specify a name for the chat room and a user name for the end user.

When end users select File->Go to Chat Room on the user interface of the Chat Room application, the MainGUI.java file calls the ChatRoom.java file to enter an existing chat room. The ChatRoom.java file allows end users to specify the name of the chat room and the user name to join the existing chat room.

When end users select any chat room from the list of existing chat rooms, the MainGUI.java file calls the SocketOpener.java file. The SocketOpener.java file opens the socket to send and receive messages between two end users. The SocketOpener.java file calls the GroupChat.java file to allow end users to participate in a group chat.

PREVIOUS NEXT

Creating the Login Window

The LoginGUI.java file creates the Login window for the Chat Room application. Listing 3-1 shows the contents of the LoginGUI.java file:

Listing 3-1: The LoginGUI.java File

```
/*Imports required swing package classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Imports required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*
Import required io and net classes
*/
import java.io.*;
import java.net.*;
/*
class LoginGUI - This class is used to create a GUI to get the information from user to login .
Constructor:
LoginGUI - This constructor creates GUI.
Methods:
getUserName:get the user name
getServerName:get the server name
*/
public class LoginGUI extends JFrame implements ActionListener, KeyListener
{
   /*Declare objects of Container class.*/
   Container c=null;
   /*Declare objects of JTextField class.*/
   JTextField namet=new JTextField();
   JTextField server=new JTextField();
   JPasswordField passwordt=new JPasswordField();
   /*Declare objects of JButton class.*/
   JButton b=new JButton("Submit");
   /*Declare objects of Socket class.*/
   Socket clientsocket;
   /*Declare String variable*/
   String servername;
   String username;
   String password;
   String resource="Home";
   String  recerverid="user5@localhost";
   String roomname;
   String nickname;
   String registrationstring="";
   String servererror="";
   /*
   Declare objects of MutableAttributeSet class.
   */
   MutableAttributeSet sendernameattrib;
   /*Declare objects of JPanel class.*/
   JPanel note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
   JPanel combine=new JPanel();
   /* Constructor for class*/
   public LoginGUI ()
   {
      /*set the title for window*/
      super("Chat Room Application");
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib,"Verdana");
      StyleConstants.setForeground(sendernameattrib,Color.blue);
      this.clientsocket=clientsocket;
      c=getContentPane();
      /*Declare objects of JPanel class.*/
      JPanel p=new JPanel();
      JLabel room= new JLabel("User Name");
      JLabel name= new JLabel("Password");
      JLabel serverl= new JLabel("Server IP");
      JLabel note11= new JLabel("* If you are not a registered user then, your account will be creat
      automatically.");
      JLabel note1= new JLabel("* If you are not a registerd user your account will be");
      JLabel note2= new JLabel("created automatically.");
      JPanel lable=new JPanel();
      /*set the layout for window*/
      p.setLayout(new GridLayout(4, 2, 3, 3));
      /*set the border for user*/
      p.setBorder(BorderFactory.createTitledBorder("Login Information"));
      /*Add the component in container*/
      p.add(room);
      p.add(namet);
      p.add(name);
      p.add(passwordt);
```

```
   p.add(serverl);
   p.add(server);
   /*set the color for label*/
   note1.setForeground(Color.red);
   note2.setForeground(Color.red);
   /*add the label to container*/
   note.add(note1);
   note.add(note2);
   /*Declare objects of JLabel class.*/
   JLabel heading=new JLabel("Login Window", JLabel.CENTER);
   lable.add(heading,BorderLayout.NORTH);
   /*set the heading for heading*/
   heading.setFont(new Font("verdana", 1, 12));
   c.add(lable,BorderLayout.NORTH);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   c.add(p,BorderLayout.CENTER);
   /*Declare objects of JPanel class.*/
   JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
   JPanel bp=new JPanel(new FlowLayout());
   bp.add(b);
   buttonpanel.add(note);
   buttonpanel.add(bp);
   c.add(buttonpanel,BorderLayout.SOUTH);
   /*set the bounds for window*/
   setBounds(5, 5, 371, 240);
   /*add the listener to button*/
   b.addActionListener(this);
   b.addKeyListener(this);
   /*called the function to show the window*/
   show();
}
/*function to handle the key event */
public void keyPressed(KeyEvent e)
{
   if(e.getKeyCode()==KeyEvent.VK_ENTER)
   {
      servername=server.getText();
      username=namet.getText();
      password=passwordt.getText();
      MainGUI mn=new MainGUI(username,password,servername,this);
   }
}
/*function to handle the key event*/
public void keyTyped(KeyEvent e)
{
}
/*function to handle the key event*/
public void keyReleased(KeyEvent e)
{
   if(e.getKeyCode()==KeyEvent.VK_ENTER)
   {
      servername=server.getText();
      username=namet.getText();
      password=passwordt.getText();
      MainGUI mn=new MainGUI(username,password,servername,this);
      servererror=mn.servererror;
   }
}
/* function to handle the  event*/
public void actionPerformed(ActionEvent e)
{
   servername=server.getText();
   username=namet.getText();
   password=passwordt.getText();
   try
   {
      MainGUI mn=new MainGUI(username,password,servername,this);
   }
   catch(Exception ee)
   {
      System.out.println("Server not found ::: "+ee);
   }
}
/*function to get username*/
public String getUserName()
{
   return username;
}
/*function to get password*/
public String getPassword()
{
   return password;
}
/* function to get servername*/
public String getServarName()
{
   return servername;
```

```
    }
    public static void main(String[] args)
    {
        /*
        Initialize and set the Look and Feel of the application to Windows Look and Feel.
        */
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        LoginGUI lgui=new LoginGUI();
    }
}
```

Download this listing.

In the above code, the main() method creates an instance of the LoginGUI class.

The methods defined in Listing 3-1 are:

- getUserName(): Retrieves the name of the end user currently logged in.
- getServerName(): Retrieves the name of the Jabber server specified by an end user.
- getPassword(): Retrieves the password specified by an end user.
- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the Login window.

The LoginGUI.java file creates the Login window of the Chat Room application, as shown in Figure 3-2:

Figure 3-2: The Login Window

Creating the User Interface of the Chat Room Application

The MainGUI.java file creates the user interface of the Chat Room application. Listing 3-2 shows the contents of the MainGUI.java file:

Listing 3-2: The MainGUI.java File

```
/*Imports required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net package classes*/
import java.io.*;
import java.net.*;
/*Import required util package classes*/
import java.util.*;
import java.lang.*;
/*Imports required swing package classes*/
import javax.swing.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import java.util.Vector;
/*
class MainGUI - This class is used to create a GUI to display the room list.
Constructor:
PortModem - This constructor creates GUI.
MainGUI:
makeTree - this function is used to create a tree for room list
*/
public class MainGUI extends JFrame implements ActionListener, Runnable, TreeSelectionListener
{
   /*Declare variables*/
   Socket clientsocket1;
   SocketOpener clientsocket;
   /*Declare objects of JMenuBar class*/
   JMenuBar jb=new JMenuBar();
   /*Declare objects of JMenu class*/
   JMenu menu=new JMenu("File");
   JMenuItem menuItem1=new JMenuItem("Go to Chat Room");
   JMenuItem menuItem2=new JMenuItem("Create New Chat Room");
   /*Declare the integer variable*/
   int portno=5222;
   int wait=1000;
   /*Declare the String variable*/
   String firstname="";
   String lastname="";
   String emailid="";
   String ipaddress;
   String errortype="";
   String servername;
   String username;
   String password;
   String resource="Home";
   String  recerverid="user5@manish";
   String roomname;
   String nickname;
   String servererror="";
   /*Declare the object of Button class*/
   Button sendbutton;
   Button closebutton;
   /*Declare the Boolean variable*/
   private  boolean isConnected=false;
   /*Declare objects of Thread class*/
   Thread inputmessagethread;
   /*Declare objects of MutableAttributeSet class*/
   MutableAttributeSet sendernameattrib,recervernameatrib, normaltextattrib;
   /* Declare objects of Document class*/
   Document contentmodel;
   /*Declare objects of JScrollPane class*/
   JScrollPane treeView;
   /*Declare objects of JTree class*/
   JTree tree;
   LoginGUI logingui;
   /*Function to make tree for room list*/
   public void makeTree()
   {
      tree=new JTree();
      DefaultMutableTreeNode top = new DefaultMutableTreeNode("Existing Rooms");
```

```
        DefaultMutableTreeNode regional = new DefaultMutableTreeNode("Regional ");
        DefaultMutableTreeNode culture = new DefaultMutableTreeNode("Culture");
        DefaultMutableTreeNode sports = new DefaultMutableTreeNode("Sports");
        DefaultMutableTreeNode education = new DefaultMutableTreeNode("Education");
        DefaultMutableTreeNode science = new DefaultMutableTreeNode("Science");
        DefaultMutableTreeNode software = new DefaultMutableTreeNode("Software");
        tree = new JTree(top);
        top.add(regional);
        top.add(culture);
        top.add(sports);
        top.add(education);
        top.add(science);
        top.add(software);
        tree.getSelectionModel().setSelectionMode
        (TreeSelectionModel.SINGLE_TREE_SELECTION);
        DefaultMutableTreeNode apolloNode =
        (DefaultMutableTreeNode)top.getFirstChild();
        TreeNode[] pathToRoot = top.getPath();
        TreePath path = new TreePath(pathToRoot);
        tree.expandPath(path);
        tree.addTreeSelectionListener(this);
        treeView = new JScrollPane(tree);
    }
    /*Constructor of class*/
    public MainGUI(String user, String password, String server, LoginGUI lgui)
    {
        super("Chat Room Application");
        /*
        Initialize and set the Look and Feel of the application to Windows Look and Feel.
        */
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        servername=server;
        username=user;
        this.password=password;
        ipaddress=server;
        logingui=lgui;
        SocketOpener.setMainPointer(this);
        SocketOpener.setLogin(logingui);
        jb.add(menu);
        menu.add(menuItem1);
        menu.add(menuItem2);
        this.setJMenuBar(jb);
        menuItem1.addActionListener(this);
        menuItem2.addActionListener(this);
        normaltextattrib=new SimpleAttributeSet();
        makeTree();
        getContentPane().add(treeView);
        setBounds(5,5,350,350);
        setVisible(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        start1();
    }
    public String getRoomName()
    {
        return roomname;
    }
    public String getNickName()
    {
        return nickname;
    }
    /*Function to handle the tree event*/
    public void valueChanged(TreeSelectionEvent e)
    {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode)
        tree.getLastSelectedPathComponent();
        if (node==null) return;
        roomname=node.toString().trim();
        if(!roomname.equals("Existing Rooms"))
        {
            nickname=JOptionPane.showInputDialog(this, "Please specify your user name for this room.");
            if(nickname!=null)
            {
                if(nickname.trim().equals(""))
                {
                    JOptionPane.showMessageDialog(this, "User name should not be blank", "Message",
                    JOptionPane.PLAIN_MESSAGE);
                    nickname=JOptionPane.showInputDialog(this, "Please specify your user name for this
                    room."
                );
                return;
            }
```

```
        }
        if(nickname==null)
        {
           return;
        }
        if((nickname.trim()!=null)||(!nickname.trim().equals("")))
        {
           String roomString="<presence from=\""+username.trim()+"@conference."+servername+"/Home\"
           to=\""+roomname.trim()+"@conference."+servername+"/"+nickname.trim()+"\"/>";
           SocketOpener.sendXMLToJabber(roomString);
        }
     }
  }
  /* Function to open a socket*/
  public void start1()
  {
     try
     {
        openPort(servername,5222,1000);
     }
     catch(Exception e)
     {
     }
     if (clientsocket1!=null)
     {
        isConnected=true;
     }
     Thread inputthread=new Thread(this);
     inputthread.start();
     try
     {
        String sessionStratString;
        String authentication;
        String registrationstring;
        sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
        sessionStratString=sessionStratString+" <stream:stream";
        sessionStratString=sessionStratString+" to= \""+servername+"\"";
        sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
        sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
        clientsocket.sendXMLToJabber(sessionStratString);
        if ((firstname.equals("false"))||(lastname.equals("false"))||(emailid.equals("false")))
        {
        }
        else
        {
           registrationstring="<iq type=\"set\" to=\""+username+"@localhost\" id=\"1001\">";
           registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
           registrationstring=registrationstring+"<username>"+username+"</username>";
           registrationstring=registrationstring+"<password>"+password+"</password>";
           registrationstring=registrationstring+"<first>"+firstname+"</first>";
           registrationstring=registrationstring+"<last>"+lastname+"</last>";
           registrationstring=registrationstring+"<email>"+emailid+"</email>";
           registrationstring=registrationstring+" </query> ";
           registrationstring=registrationstring+"</iq>";
        }
        authentication="<iq type=\"set\" id=\"1301\">";
        authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
        authentication=authentication+" <username>"+username+"</username>";
        authentication=authentication+" <password>"+password+"</password>";
        authentication=authentication+" <resource>"+resource+"</resource> ";
        authentication=authentication+" </query> ";
        authentication=authentication+"</iq>";
        clientsocket.sendXMLToJabber(authentication);
     }
     catch(Exception ie)
     {
     }
  }
  /*run function to handle the activity by thread*/
  public void run()
  {
     clientsocket.runinput();
  }
  /*Function to open a socket*/
  private void openPort(String ipaddress,int portno,int timeinsec)
  {
     String host;
     if (ipaddress.trim()=="")
     {
     }
     else
     {
        SocketOpener sc=new SocketOpener(ipaddress, portno);
        clientsocket=sc.openSocket(ipaddress, portno, timeinsec);
     }
  }
  /* Function to handle the event*/
```

```
public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==menuItem1)
   {
      ChatRoom r=new ChatRoom(clientsocket1, this);
   }
   if(e.getSource()==menuItem2)
   {
      RoomInformation r=new RoomInformation(clientsocket1, this);
   }
}
}
```

Download this listing.

In the above code, the constructor of the MainGUI class takes an object of the LoginGUI class and three strings as input parameters: user, password, and server. The object of the LoginGUI class allows an end user to invoke the methods of the LoginGUI class. The user string retrieves the user name of the end user. The password string retrieves the password of the end user, and the server string retrieves the ip address of the Jabber server to connect.

The methods defined in Listing 3-2 are:

- `makeTree()`: Maintains a tree structure of the chat rooms listed in the user interface.
- `valueChanged()`: Retrieves the name of the chat room selected by an end user from the various chat rooms listed in the user interface.
- `openPort ()`: Opens a socket to send and receive messages.
- `getRoomName()`: Retrieves the name of the chat room specified by an end user to participate in a group chat.
- `getNickName()`: Retrieves the name of an end user to initiate the chat.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by selecting any option from the user interface.

The MainGUI.java file creates the main window of the Chat Room application, as shown in Figure 3-3:

Figure 3-3: User Interface of the Chat Room Application

Select the File menu to view the File menu options.

Figure 3-4 shows the File menu options of the Chat Room application:

Figure 3-4: The File Menu of the Chat Room Application

Select File->Go to Chat Room to enter an existing chat room.

Select File->Create New Chat to create a new chat room.

Select any chat room from the Existing Rooms tree structure to initiate a chat.

PREVIOUS NEXT

Joining a Chat Room

The ChatRoom.java file creates the user interface to enter into an existing chat room to initiate a chat. Listing 3-3 shows the contents of the ChatRoom.java file:

Listing 3-3: The ChatRoom.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*
class ChatRoom-This class is used to create a GUI to take the room name and nick from user.
Constructor:
ChatRoom-This constructor creates GUI.
Methods:
getRoomName-to get room name
getNickName-to get nick name
*/
public class ChatRoom extends JDialog implements ActionListener
{
   /*Declare objects of Container class*/
   Container c=null;
   /*Declare objects of JTextField class*/
   JTextField roomt=new JTextField();
   JTextField namet=new JTextField();
   /*Declare objects of JButton class*/
   JButton b=new JButton("Submit");
   /*Declare objects of MainGUI class*/
   MainGUI maingui;
   /*Declare objects of Socket class*/
   Socket clientsocket;
   static Socket clientsocket1;
   /*Declare String variables*/
   String servername="localhost";
   String password="user4";
   String resource="Home";
   String  recerverid="user5@localhost";
   String roomname;
   String nickname;
   String username;
   /*Declare objects of MutableAttributeSet class*/
   MutableAttributeSet sendernameattrib;
   /*Constructor for class*/
   public ChatRoom(Socket clientsocket, MainGUI maingui)
   {
      setTitle("Chat Room Application");
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib,"Verdana");
      StyleConstants.setForeground(sendernameattrib,Color.blue);
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
      }
      this.clientsocket =clientsocket;
      c=getContentPane();
      this.maingui=maingui;
      /*Declare objects of JPanel class*/
      JPanel p = new JPanel();
      JLabel room = new JLabel("Room Name");
      JLabel name = new JLabel("User Name");
      JPanel lable = new JPanel();
      /*set the layout*/
      p.setLayout(new GridLayout(3, 2, 10, 10));
      /*set the border to panel*/
      p.setBorder(BorderFactory.createTitledBorder("Chat Room"));
      /*add the component to container*/
      p.add(room);
      p.add(roomt);
      p.add(name);
      p.add(namet);
      /*Declare objects of JLabel class*/
      JLabel heading=new JLabel("Chat Room", JLabel.CENTER);
      /*set the font heading*/
      heading.setFont(new Font("verdana", 1, 12));
      lable.add(heading, BorderLayout.NORTH);
```

```
        /*add panel to container*/
        c.add(lable,BorderLayout.NORTH);
        /*set default close operation for window*/
        setDefaultCloseOperation(1);
        /* set resizable true to show the window*/
        setResizable(false);
        c.add(p,BorderLayout.CENTER);
        /*Declare objects of JPanel class*/
        JPanel buttonpanel=new JPanel(new FlowLayout());
        /*add button to container*/
        buttonpanel.add(b);
        c.add(buttonpanel,BorderLayout.SOUTH);
        /*set bounds for window*/
        setBounds(5, 5, 300, 200);
        /*add action listener to button*/
        b.addActionListener(this);
        /*calls the show function to show the window*/
        show();
    }
    /*function to handle the event*/
    public void actionPerformed(ActionEvent e)
    {
        if((JButton)e.getSource()==b)
        {
            if(roomt.getText()==null||roomt.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please specify the room name." );
                return;
            }
            if(namet.getText()==null||namet.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please specify the user name." );
                return;
            }
            String dialog=null;
            roomname=roomt.getText();
            nickname=namet.getText();
            this.username=maingui.username;
            String roomString="<presence from=\""+username+"@conference.localhost/Home\"
            to=\""+roomname.trim()+"@conference."+servername+"/"+nickname.trim()+"\"/>";
            SocketOpener.sendXMLToJabber(roomString);
            SocketOpener.setChatPointer(this, maingui);
            return;
        }
    }
    /*function to get the room name*/
    public String getRoomName()
    {
        return roomname;
    }
    /*function to get the nick name*/
    public String getNickName()
    {
        return nickname;
    }
}
```

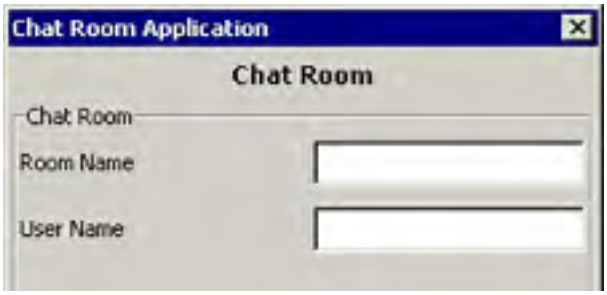
Download this listing.

In the above code, the constructor of the ChatRoom class takes an object of the MainGUI class and the Socket class as input parameters. The ChatRoom class allows end users to invoke the methods of the MainGUI class and the Socket class.

The methods defined in Listing 3-3 are:

- getRoomName(): Retrieves the name of the chat room specified by an end user.
- getNickName(): Retrieves the user name of an end user for the selected chat room.
- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the user interface.

The ChatRoom.java file creates the Chat Room window that allows end users to participate in a chat, as shown in Figure 3-5:



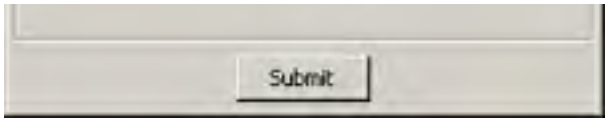


Figure 3-5: Joining a Chat Room

◄ PREVIOUS NEXT ►

Creating a Chat Room

The RoomInformation.java file allows end users to create a chat room. Listing 3-4 shows the contents of the RoomInformation.java file:

Listing 3-4: The RoomInformation.java File

```
/*Imports required swing package classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Imports required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required io and net package classes*/
import java.io.*;
import java.net.*;
/*
class RoomInformation - This class is used to create a GUI to take the room information from user.
Constructor:
RoomInformation - This constructor creates GUI.
Methods:
getRoomName - to get room name
getNickName - to get nick name
*/
public class RoomInformation extends JDialog implements ActionListener
{
   /*Declare objects of Container class*/
   Container c=null;
   /*Declare objects of JTextField class*/
   JTextField roomt=new JTextField();
   JTextField namet=new JTextField();
   /* Declare objects of JButton class*/
   JButton b=new JButton("Submit");
   /*Declare objects of MainGUI class*/
   MainGUI maingui;
   /*Declare objects of Socket class*/
   Socket clientsocket;
   static Socket clientsocket1;
   /*Declare String variables*/
   String servername="localhost";
   String password="user4";
   String resource="Home";
   String recerverid="user5@localhost";
   String roomname;
   String nickname;
   String username;
   /* Declare objects of MutableAttributeSet class*/
   MutableAttributeSet sendernameattrib;
   /*Constructor for class*/
   public RoomInformation(Socket clientsocket, MainGUI maingui)
   {
      setTitle("Chat Room Application");
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib, "Verdana");
      StyleConstants.setForeground(sendernameattrib, Color.blue);
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      this.clientsocket = clientsocket;
      c=getContentPane();
      this.maingui=maingui;
      /*Declare objects of JPanel class*/
      JPanel p=new JPanel();
      JLabel room = new JLabel("Room Name");
      JLabel name = new JLabel("User Name");
      JPanel lable = new JPanel();
      /*set the layout*/
      p.setLayout(new GridLayout(3, 2, 10, 10));
      /*set the border to panel*/
      p.setBorder(BorderFactory.createTitledBorder("Room Information"));
      /*add the component to container*/
      p.add(room);
      p.add(roomt);
      p.add(name);
      p.add(namet);
      /*Declare objects of JLabel class*/
      JLabel heading=new JLabel("Create Room", JLabel.CENTER);
      /*set the font heading*/
```

```
        heading.setFont(new Font("verdana", 1, 12));
        lable.add(heading, BorderLayout.NORTH);
        c.add(lable,BorderLayout.NORTH);
        /*set default close operation for window*/
        setDefaultCloseOperation(1);
        /*set resizable true to show the window*/
        setResizable(false);
        /*add panel to container*/
        c.add(p,BorderLayout.CENTER);
        /*Declare objects of JPanel class*/
        JPanel buttonpanel=new JPanel(new FlowLayout());
        /*add button to container*/
        buttonpanel.add(b);
        /*Add button panel to container*/
        c.add(buttonpanel,BorderLayout.SOUTH);
        /*set bounds for window*/
        setBounds(5, 5, 300, 200);
        /*add action listener to button*/
        b.addActionListener(this);
        /*calls the show function to show the window*/
        show();
    }
    /*function to handle the event*/
    public void actionPerformed(ActionEvent e)
    {
        if((JButton)e.getSource()==b)
        {
            if(roomt.getText()==null||roomt.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please Insert the room Name." );
                return;
            }
            if(namet.getText()==null||namet.getText().trim().equals(""))
            {
                JOptionPane.showMessageDialog(this, "Please specify the user name." );
                return;
            }
            String dialog=null;
            roomname=roomt.getText();
            nickname=namet.getText();
            this.username=maingui.username;
            String roomString="<presence from=\""+username+"@conference.localhost/Home\"
            to=\""+roomname.trim()+"@conference."+servername+"/"+nickname.trim()+"\"/>";
            SocketOpener.sendXMLToJabber(roomString);
            SocketOpener.setRoomPointer(this,maingui);
            return;
        }
    }
    /*function to get the room name*/
    public String getRoomName()
    {
        return roomname;
    }
    /*function to get the nick name*/
    public String getNickName()
    {
        return nickname;
    }
}
```

Download this listing.

In the above code, the constructor of the RoomInformation class takes an object of the MainGUI class and the Socket class as input parameters. The RoomInformation class allows end users to invoke the methods of the MainGUI class and the Socket class.

The methods defined in Listing 3-4 are:

- `getRoomName()`: Retrieves the name of the chat room specified by an end user to create a chat room.
- `getNickName()`: Retrieves the user name of an end user for the created chat room.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the user interface.

The RoomInformation.java file creates the Create Room window that allows end users to create a chat room, as shown in Figure 3-6:

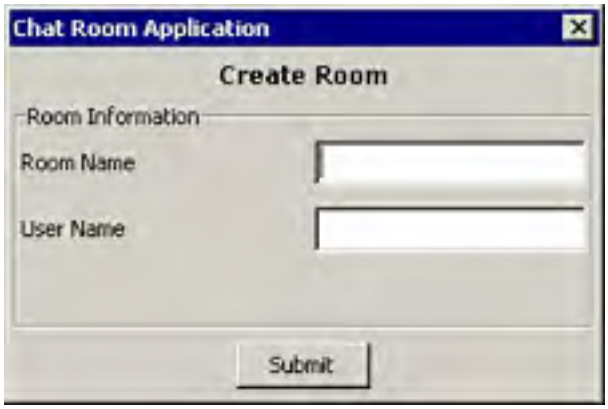


Figure 3-6: Creating a Chat Room

PREVIOUS NEXT

Creating the SocketOpener Class to Send and Receive Messages

The SocketOpener.java file creates a socket with the Jabber server to send and receive messages between end users. Listing 3-5 shows the contents of the SocketOpener.java file:

Listing 3-5: The SocketOpener.java File

```
/*Imports required swing package classes.*/
import javax.swing.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
/*Import required awt package classes.*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net package classes.*/
import java.io.*;
import java.net.*;
/*Import required util package classes.*/
import java.util.*;
import java.lang.*;
/*
class SocketOpener - This class is used for socket opening.
Constructor:
SocketOpener - This constructor accept host name and IP.
Methods:
openSocket - open the socket
getErrorString - get the error string send by server
writemessage-used to write the message in the textpane.
*/
public class SocketOpener
{
   /*Declare the int variables.*/
   private int openerport;
   /* Declare objects of Socket classes.*/
   static private Socket socket;
   static MutableAttributeSet sendernameattrib, nameattribut, informationattrib, welcomenameattrib;
   static Document contentmodel;
   /*Declare objects of Stream classes.*/
   static PrintWriter out = null;
   static BufferedReader in = null;
   /*Declare boolean variable.*/
   private  static boolean isConnected=false;
   /*Declare objects of Hasmap and vector class.*/
   Vector nicknamevector=new Vector();
   HashMap temphashmap=new HashMap();
   /*Declare objects of GroupChat class.*/
   static GroupChat gc;
   /*Declare objects of RoomInformation class.*/
   static RoomInformation socketroomgui;
   /*Declare objects of ChatRoom class.*/
   static ChatRoom socketchatmgui;
   /*Declare objects of MainGUI class.*/
   static MainGUI socketmaingui;
   /*Declare objects of LoginGUI class.*/
   static LoginGUI logingui;
   /*Declare the string variable*/
   static String errortype="";
   private String openerhost;
   String roomname="";
   String firstname="";
   String lastname="";
   String emailid="";
   String resource="Home";
   String temproomname="";
   Color textcolor;
   String nickname="";
   String servererror="";
   /* function to open socket*/
   public  SocketOpener openSocket(String hostip, int portnumber, int timeinsec)
   {
      sendernameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(sendernameattrib, "Verdana");
      StyleConstants.setForeground(sendernameattrib, Color.black);
      welcomenameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(welcomenameattrib, "Verdana");
      StyleConstants.setForeground(welcomenameattrib, Color.blue);
      nameattribut=new SimpleAttributeSet();
      StyleConstants.setFontFamily(nameattribut,"Verdana");
      StyleConstants.setForeground(nameattribut, Color.red);
      informationattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(informationattrib, "Verdana");
      StyleConstants.setItalic(informationattrib,true);
      StyleConstants.setForeground(informationattrib, Color.red);
      socketope();
```

```
   try
   {
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
   }
   catch(Exception exception)
   {
      JOptionPane.showMessageDialog(logingui, "Server not found", "Server not found", JOptionPane
      socketmaingui.servererror="Server error";
   }
   return getSocketClass();
}
/*Function to initialize the Socket class*/
public  void socketope()
{
   try
   {
      socket=new Socket(openerhost, openerport);
   }
   catch(IOException ie)
   {
   }
}
/*Constructor for SocketOpener class*/
public SocketOpener(String host, int port)
{
   socket=null;
   openerhost=host;
   openerport=port;
}
/*
Function returns the object of SocketOpener class
*/
public SocketOpener getSocketClass()
{
   return this;
}
/*Function to write a message*/
public void writemessage(String messagebody, MutableAttributeSet attribut)
{
   try
   {
      gc.writeintotextfield(messagebody, attribut);
   }
   catch(Exception ble)
   {
   }
}
/* Function returns the error string send by server*/
public static String getErrorString(String codeid)
{
   if (codeid=="'401'")
   {
      errortype="minor";
      return "You have sent malformed syntax which can't be understood by server.";
   }
   if (codeid=="'404'")
   {
      errortype="major";
      return "No Server exists.";
   }
   if (codeid=="'405'")
   {
      errortype="minor";
      return "You have no right to send this command.";
   }
   if (codeid=="'409'")
   {
      errortype="major";
      return "A user with this jabber id is already exist. Please choose another user id.";
   }
   if (codeid=="'403'")
   {
      errortype="auth";
      return "Invalid password";
   }
   return "";
}
public void sendername(String tagentity)
{
   String sendername="";
   String newuserid="";
   tagentity=tagentity.trim();
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("id=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("id="));
```

```
                gc.writeintotextfield("\n"+sendername,nameattribut);
            }
        }
        if (tagentity.startsWith("message"))
        {
            if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
            {
                sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to="));
                if(!nickname.trim().equalsIgnoreCase(sendername.trim()))
                {
                    gc.writeintotextfield("\n"+sendername.substring(sendername.indexOf("/")+1,
                    sendername.indexOf("'"))+":",nameattribut);
                }
                else
                {
                    gc.writeintotextfield("\n"+sendername.substring(sendername.indexOf("/")+1,
                    sendername.indexOf("'"))+": ",sendernameattrib);
                }
            }
        }
        if (tagentity.startsWith("presence"))
        {
            if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
            {
                nickname=tagentity.substring(tagentity.indexOf("from=")+6, tagentity.indexOf("to=")-1);
                if (tagentity.indexOf("type='unavailable'")>0||tagentity.indexOf("type=\"unavailable\"")
                {
                    gc.removeuser(nickname.substring(nickname.indexOf("/")+1, nickname.indexOf("'")));
                    String nickname1="\n"+nickname.substring(nickname.indexOf("/")+1,
                    nickname.indexOf("'"))+"  has left the room";
                    gc.writeintotextfield(nickname1, informationattrib);
                }
                else
                {
                if (gc!=null)
                {
                    gc.addNewUser(nickname.substring(nickname.indexOf("/")+1,nickname.indexOf("'")));
                    String nickname1="\n"+nickname.substring(nickname.indexOf("/")+1,
                    nickname.indexOf("'"))+"  has joined the room";
                    gc.writeintotextfield(nickname1, informationattrib);
                }
                else
                {
                    nicknamevector.add(nickname.substring(nickname.indexOf("/")+1,nickname.indexOf("'")))
                }
            }
        }
    }
    if (tagentity.startsWith("item"))
    {
        if (tagentity.indexOf("jid")<0&&(tagentity.indexOf("affiliation='none'")>0)
        &&(tagentity.indexOf("role='participant'")>0))
        {
            if (socketmaingui!=null)
            {
                temproomname=socketmaingui.getRoomName();
                nickname=socketmaingui.getNickName();
            }
            if (socketchatmgui!=null)
            {
                temproomname=socketchatmgui.getRoomName();
                nickname=socketchatmgui.getNickName();
                gc=new GroupChat(temproomname,nickname,openerhost);
                socketchatmgui.hide();
                gc.setUserName(logingui.getUserName());
                for (int i=0;i<nicknamevector.size();i++)
                {
                    gc.addNewUser((String)nicknamevector.get(i));
                }
                gc.hashmap.putAll(temphashmap);
                roomname=temproomname;
                gc.writeintotextfield("Welcome to " +roomname+ "  room ", welcomenameattrib);
            }
            if (temproomname.equals("Regional")||temproomname.equals("Culture")||
            temproomname.equals("Sports")||temproomname.equals("Education")||
            temproomname.equals("Science")||temproomname.equals("Software"))
            {
                if (gc==null)
                {
                gc=new GroupChat(temproomname, nickname, openerhost);
                gc.setUserName(logingui.getUserName());
                for (int i=0;i<nicknamevector.size();i++)
                {
                    gc.addNewUser((String)nicknamevector.get(i));
                }
                gc.hashmap.putAll(temphashmap);
                roomname=temproomname;
                gc.writeintotextfield("Welcome to " +roomname+ " room ", welcomenameattrib);
```

```
            }
            else
            {
               if (!gc.isVisible())
               {
                  gc.clearPane();
                  gc.writeintotextfield("Welcome to " +roomname+ " room ", welcomenameattrib);
                  gc.setVisible(true);
               }
            }
         }
          if(temproomname.equals("Regional")||temproomname.equals("Culture")||
          temproomname.equals("Sports")||temproomname.equals("Education")||
          temproomname.equals("Science")||
          temproomname.equals("Software"))
         {
         }
         else
         {
            temphashmap.clear();
            nicknamevector.clear();
            if (socketchatmgui==null)
            {
               JOptionPane.showMessageDialog(null, "Room already exists.");
            }
         }
         return;
      }
      if (tagentity.indexOf("jid=")>0&&tagentity.indexOf("affiliation=")>0)
      {
         newuserid=tagentity.substring(tagentity.indexOf("jid=")+5,tagentity.indexOf("affiliation=")-2)
         if(newuserid.equals(logingui.getUserName()+"@"+logingui.getServarName().trim()+"/Home"))
         {
            if (tagentity.indexOf("affiliation='owner'")>0||tagentity.indexOf
            ("affiliation=\"owner\"")>0)
            {
               if (socketroomgui!=null)
               {
                  socketroomgui.hide();
                  roomname=socketroomgui.getRoomName().trim();
                  nickname=socketroomgui.getNickName().trim();
               }
               else
               {
               try
               {
                  roomname=socketmaingui.getRoomName().trim();
                  nickname=socketmaingui.getNickName().trim();
               }
               catch(Exception e)
               {
                  JOptionPane.showMessageDialog(socketroomgui, "Room does not  exist");
                  return;
               }
            }
            gc=new GroupChat(roomname,nickname,openerhost);
            gc.setUserName(logingui.getUserName());
            for (int i=0;i<nicknamevector.size();i++)
            {
               gc.addNewUser((String)nicknamevector.get(i));
            }
            gc.hashmap.putAll(temphashmap);
            gc.writeintotextfield("Welcome to " +roomname+ "  room", welcomenameattrib);
            }
            else
            {
               if (roomname.equals("Regional")||roomname.equals("Culture")||roomname.equals
               ("Sports")||roomname.equals("Education")||roomname.equals("Science")
               ||roomname.equals("Software"))
               {
                  gc=new GroupChat(roomname, nickname, openerhost);
                  gc.setUserName(logingui.getUserName());
               }
               else
               {
                  if (gc!=null)
                  {
                  if (gc.isVisible())
                  {
                     gc.hide();
                     JOptionPane.showMessageDialog(null, "Room is already exist");
                     return;
                  }
               }
            }
            if (!socketroomgui.isVisible())
            {
               socketroomgui.setVisible(true);
```

```
            }
         }
      }
      if (tagentity.indexOf("role='none'")>0||tagentity.indexOf("role=\"none\"")>0)
      {
         if (!newuserid.equals(""))
         {
            gc.removeUserinHash(nickname,newuserid);
         }
      }
      else
      {
         if (!newuserid.equals(""))
         {
            if (gc!=null)
            {
               gc.addNewUserinHash(nickname,newuserid);
            }
            else
            {
               temphashmap.put(nickname,newuserid);
            }
         }
      }
   }
   }
   }
   /*
   Function to set the object of RoomInformation and MainGUI class
   */
   public static void setRoomPointer(RoomInformation room,MainGUI maingui)
   {
      socketroomgui=room;
      socketmaingui=maingui;
   }
   /*
   Function to set the object of ChatRoom and MainGUI class
   */
   public static void setChatPointer(ChatRoom chatroom,MainGUI maingui)
   {
      socketchatmgui=chatroom;
      socketmaingui=maingui;
   }
   /*Function to set the object of MainGUI class*/
   public static void setMainPointer(MainGUI maingui)
   {
      socketmaingui=maingui;
   }
   /*Function to set the object of LoginGUI class*/
   public static void setLogin(LoginGUI lgui)
   {
      logingui=lgui;
   }
   /*Function to check xml error*/
   public static String checkForError(String tagentity)
   {
      String error="";
      String codeid="";
      if (tagentity.indexOf("error")>0)
      {
      if (tagentity.indexOf("from=")>0)
      {
         codeid=tagentity.substring(tagentity.indexOf("id=")+3, tagentity.indexOf("type="));
         return "Invalid password.";
      }
      return "No user exists for this user ID.";
   }
   if (tagentity.indexOf("result")>0)
   {
      if(!socketmaingui.isVisible())
   {
      socketmaingui.setVisible(true);
      logingui.setVisible(false);
   }
   }
   return error;
   }
   /*Function to read the message from socket*/
   public void runinput()
   {
      int i=0;
      String errororwarning="";
      String errorstring="";
      String inputstring="";
      boolean starttag=false;
      boolean endtag=false;
```

```
boolean startwritting=false;
String messagebody="";
String tagentity="";
Vector tagvactor;
tagvactor=new Vector();
int vactorindex=0;
int starttagsing=0;
while (true)
{
   try
   {
      i=in.read();
      if (i==-1)
      {
         break;
      }
      else
      {
         inputstring=inputstring+(char)i;
         if ((char)i=='<')
         {
            starttag=true;
            starttagsing=1;
            tagentity="";
         }
         else
         {
            if ((char)i=='/'&& starttag==true &&starttagsing==1 )
            {
               starttag=false;
               endtag=true;
               if (!messagebody.trim().equals(""))
               {
                  writemessage(messagebody,sendernameattrib);
               }
               startwritting=false;
               messagebody="";
               vactorindex=vactorindex-1;
               if (vactorindex>=0)
               tagvactor.removeElementAt(vactorindex);
            }
            else
            {
            starttagsing=0;
            if((char)i=='>')
            {
               if (starttag)
               {
                  sendername(tagentity);
                  errorstring=checkForError(tagentity);
                  if (errorstring.equals("Invalid password."))
                  {
                     JOptionPane.showMessageDialog(logingui, "Invalid password", "Invalid
                     password",JOptionPane.ERROR_MESSAGE);
                     return;
                  }
                  if (!errorstring.equals(""))
                  {
                     String username=logingui.getUserName();
                     String password=logingui.getPassword();
                     String servarn=logingui.getServarName();
                     String registrationstring="<iq type=\"set\" to=\""+username+"@"+servarn+"\"
                     id=\"1001\">";
                     registrationstring=registrationstring+"<query
                     xmlns=\"jabber:iq:register\">";
                     registrationstring=registrationstring+"<username>"+username+
                     "</username>";registrationstring=registrationstring+"<password>
                     "+password+"</password>";
                      registrationstring=registrationstring+"<first>
                      "+firstname+"</first>";
                     registrationstring=registrationstring+"<last>"+lastname+"</last>";
                     registrationstring=registrationstring+"<email>"+emailid+"</email>";
                     registrationstring=registrationstring+" </query> ";
                     registrationstring=registrationstring+"</iq>";
                     String authentication="<iq type=\"set\" id=\"1301\">";
                     authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
                     authentication=authentication+" <username>"+username+"</username>";
                     authentication=authentication+" <password>"+password+"</password>";
                     authentication=authentication+" <resource>"+resource+"</resource>";
                     authentication=authentication+" </query> ";
                     authentication=authentication+"</iq>";
                     if(errortype=="auth")
                     {
                        errororwarning="Error";
                     }
                     if(errortype=="major")
                     {
                        errororwarning="Error";
```

```
                                JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                                JOptionPane.PLAIN_MESSAGE);
                                sendXMLToJabber(registrationstring);
                                sendXMLToJabber(authentication);
                                if (socketmaingui!=null)
                                {
                                   socketmaingui.hide();
                                }
                             }
                             else
                             {
                                errororwarning="Warning";
                                sendXMLToJabber(registrationstring);
                                sendXMLToJabber(authentication);
                                if (socketmaingui!=null)
                                {
                                   socketmaingui.hide();
                                }
                             }
                          }
                          startwritting=true;
                          tagvactor.insertElementAt(tagentity,vactorindex);
                          vactorindex=vactorindex+1;
                          starttag=false;
                       }
                    }
                    else
                    {
                       if (startwritting==true&&tagentity.trim().equals("body"))
                       {
                          messagebody=messagebody+(char)i;
                          messagebody=messagebody.trim();
                       }
                       if (starttag)
                       {
                          tagentity=tagentity+(char)i;
                       }
                    }
                 }
              }
           }
        }
     }
     catch(IOException ie)
     {
        JOptionPane.showMessageDialog(socketchatmgui, "Room not found", "Room not
        found", JOptionPane.ERROR_MESSAGE);
        return;
     }
}
isConnected=false;
}
/*Function to send xml to server*/
public static void sendXMLToJabber(String outputmessage)
{
   out.println(outputmessage);
   out.flush();
}
/*Function to set the object of GroupChat class*/
public static void setgc(GroupChat groupchat)
{
   gc=groupchat;
}
}
```

Download this listing.

In the above code, the constructor of the SocketOpener class takes host and port as input parameters. The host string retrieves the name of the Jabber server and the port integer retrieves the port number to open the socket with the Jabber server.

The methods defined in Listing 3-5 are:

- `openSocket()`: Opens the socket to send and receive messages.
- `getErrorString()`: Retrieves the error string sent by the Jabber server.
- `writemessage()`: Writes the text message in the text area of the Group Chat window.
- `sendXMLToJabber()`: Sends an XML (eXtensible Markup Language) message to the Jabber server.
- `setgc()`: Sets the object of the GroupChat class.
- `runinput()`: Listens for the server response sent by the Jabber server.
- `checkForError()`: Checks whether or not the input string contains an error message.
- `setLogin()`: Sets the object of the LoginGUI class.

- `setChatPointer()`: Sets the objects of the ChatRoom and MainGUI classes.
- `setRoomPointer()`: Sets the objects of the RoomInformation and MainGUI classes.
- `sendername()`: Retrieves the sender's name from the input message received from the Jabber server.
- `getErrorString()`: Retrieves the message sent by the Jabber server.
- `socketope()`: Initializes the Socket class to invoke the methods of the Socket class.

◄ PREVIOUS NEXT ►

Creating a Group Chat Window

The GroupChat.java file creates the Group Chat window of the chat room selected by an end user to send messages to all end users in the chat room. Listing 3-6 shows the contents of the GroupChat.java file:

Listing 3-6: The GroupChat.java File

```
/*Import required swing package classes*/
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.*;
/*Import required awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required util package classes*/
import java.util.*;
public class  GroupChat extends JFrame implements ActionListener, KeyListener
{
   /*Declare object of Container class.*/
   Container cont = null;
   /*Declare object of JMenuBar class.*/
   JMenuBar jb=new JMenuBar();
   JMenu menu=new JMenu("File");
   /*Declare object of JMenuItem class.*/
   JMenuItem menuItem1=new JMenuItem("Got To Chat Room");
   JMenuItem menuItem2=new JMenuItem("Create New Chat Room");
   /*Declare object of JPanel class.*/
   JPanel jp=new JPanel(new BorderLayout());
   /*Declare object of JButton class.*/
   JButton button=new JButton("Send");
   JButton colorb=new JButton("Text Color");
   /*Declare object of JTextField class.*/
   JTextField text=new JTextField();
   JList list;
   JColorChooser chooser=new JColorChooser();
   JDialog dialog;
   private DefaultListModel listModel;
   JTextPane je=new JTextPane();
   HashMap hashmap;
   /*Declare object of JScrollPane class.*/
   JScrollPane scrollPane3 = new JScrollPane(text);
   JScrollPane scrollPane1 = new JScrollPane(je);
   JScrollPane scrollPane;
   JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
   /*Declare the string variable*/
   String chatText=null;
   String roomname;
   String nickname;
   String servername;
   String leftmessage="";
   String username="";
   /*Declare object of Document class.*/
   Document contentmodel;
   /*Declare object of MutableAttributeSet class.*/
   MutableAttributeSet nicknameattrib;
   /*Function to clear the text from textpane*/
   public void clearPane()
   {
      je.setText("");
   }
   /* Constructor for class*/
   public GroupChat(String proomname,String pnickname,String pservername)
   {
      /*set the window title*/
      super(" Chat Room Application: "+proomname);
      cont = getContentPane();
      je.setEditable(false);
      nicknameattrib=new SimpleAttributeSet();
      /*set the attribute for text*/
      StyleConstants.setFontFamily(nicknameattrib,"Verdana");
      StyleConstants.setForeground(nicknameattrib,Color.black);
      contentmodel=je.getDocument();
      roomname=proomname;
      nickname=pnickname;
      servername=pservername;
      hashmap=new HashMap();
      listModel=new DefaultListModel();
      list=new JList(listModel);
      scrollPane= new JScrollPane(list);
      list.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
      list.setVisibleRowCount(5);
      jb.add(menu);
      menu.add(menuItem1);
      menu.add(menuItem2);
```

```
   Dimension minimumSize = new Dimension(300, 150);
   splitPane.setTopComponent(scrollPane1);
   splitPane.setBottomComponent(scrollPane);
   splitPane.setDividerLocation(400);
   splitPane.setPreferredSize(new Dimension(700, 200));
   list.setSelectionBackground(Color.cyan);
   Dimension d=button.getPreferredSize();
   button.setPreferredSize(new Dimension(130, 20));
   colorb.setPreferredSize(new Dimension(130, 20));
   text.setPreferredSize(new Dimension(400, 20));
   jp.add(text, BorderLayout.WEST);
   jp.add(button, BorderLayout.EAST);
   cont.add(jp, BorderLayout.SOUTH);
   cont.add(splitPane, BorderLayout.CENTER);
   setBounds(5, 5, 550, 550);
   getRootPane().show();
   menuItem1.addActionListener(this);
   menuItem2.addActionListener(this);
   button.addActionListener(this);
   button.addKeyListener(this);
   text.addKeyListener(this);
   colorb.addActionListener(this);
   this.addWindowListener(windowListener);
   this.show();
}
/*
Function to write the message in to the text pane
*/
public void writeintotextfield(String writeintotextfield, MutableAttributeSet psendernameattrib)
{
   try
   {
      contentmodel.insertString(contentmodel.getLength(),writeintotextfield,psendernameattrib);
   }
   catch(Exception e)
   {
   }
}
/*Inner class to handle window event*/
WindowListener windowListener = new WindowAdapter()
{
   public void windowClosing (WindowEvent w)
   {
      leftmessage="<presence type=\"unavailable\" from=\""+username+"@"+servername+"/Home\"
      to=\""+roomname+"@conference."+servername+"/"+nickname+"\"></presence>";
      SocketOpener.sendXMLToJabber(leftmessage);
      listModel.clear();
      clearPane();
      setVisible(false);
   }
   public void windowOpened(WindowEvent e)
   {
      leftmessage="<presence type=\"available\" from=\""+username+"@"+servername+"/Home\"
      to=\""+roomname+"@conference."+servername+"/"+nickname+"\"></presence>";
      SocketOpener.sendXMLToJabber(leftmessage);
      setVisible(true);
   }
};
/*Function to handle key event*/
public void keyPressed(KeyEvent e)
{
   if(e.getKeyCode()==KeyEvent.VK_ENTER)
   {
      String sendxmlstring="";
      chatText=text.getText();
      if (chatText.trim().equals(""))
      {
      }
      else
      {
         sendxmlstring="<message to=\""+roomname+"@conference."+servername+"\"
         from=\""+username+"@conference.localhost/Home\" type=\"groupchat\">";
         sendxmlstring=sendxmlstring+"<body>"+chatText+"</body></message>";
         SocketOpener.sendXMLToJabber(sendxmlstring);
      }
      text.setText("");
      }
   }
   /*Function to handle key event*/
   public void keyTyped(KeyEvent e)
   {
   }
   /*Function to handle key event*/
   public void keyReleased(KeyEvent e)
   {
   }
   /* Function to handle  event*/
```

```
        public void actionPerformed(ActionEvent e)
        {
           boolean modal = false;
           if(e.getSource()==button)
           {
              String sendxmlstring="";
              chatText=text.getText();
              if (chatText.trim().equals(""))
              {
              }
              else
              {
                 sendxmlstring="<message to=\""+roomname+"@conference."+servername+"\"
                 from=\""+username+"@conference.localhost/Home\" type=\"groupchat\">";
                 sendxmlstring=sendxmlstring+"<body>"+chatText+"</body></message>";
                 SocketOpener.sendXMLToJabber(sendxmlstring);
              }
              text.setText("");
           }
           if(e.getSource()==menuItem2)
           {
              String dialog=JOptionPane.showInputDialog(this, "Please Insert the New Room Name.");
              return;
           }
        }
        /*Function to add user in a list*/
        public void addNewUser(String newuser)
        {
           int index=0;
           String newusersearch="<html><body ><font color=red
           face=verdana>"+newuser+"</body></html>";
           String usersearch="<html><body><font color=black
           face=verdana>"+newuser+"</body></html>";
           if (!((listModel.contains((Object)newusersearch))||(listModel.contains((Object)usersearch))
           {
              if (newuser.equals(nickname))
              {
                 listModel.insertElementAt("<html><body ><font color=red
                 face=verdana>"+newuser+"</body></html>", index);
              }
              else
              {
                 listModel.insertElementAt("<html><body><font color=black
                 face=verdana>"+newuser+"</body></html>", index);
              }
           }
        }
        /* Function to remove user from the list*/
        public void removeuser(String username)
        {
        String rusername="<html><body><font color=black
        face=verdana>"+username+"</body></html>";
        if (listModel.contains((Object)rusername))
        {
           boolean n=listModel.removeElement((Object)rusername);
        }
     }
     /* Function to add user in hashmap*/
     public void addNewUserinHash(String nickname,String newuserid)
     {
        hashmap.put(nickname,newuserid);
     }
     /* Function to remove user in hashmap*/
     public void removeUserinHash(String nickname,String newuserid)
     {
        hashmap.remove(nickname);
     }
     /* Function to set username*/
     public void setUserName(String uname)
     {
        username=uname;
     }
}
```

Download this listing.

In the above code, the constructor of the GroupChat class takes three strings as input parameters: proomname, pnickname, and pservername. The proomname string retrieves the name of the chat room specified by an end user. The pnickname string retrieves the user name of an end user, and the pservername string retrieves the name of the Jabber server.

The methods defined in Listing 3-6 are:

- writeintotextfield(): Writes the message text in the text area of the Group Chat window.
- keyPressed(): Acts as a key listener and activates an appropriate method based on the key pressed by an end user.

- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs. End users invoke this method by clicking any button on the Group Chat window.
- clearPane(): Refreshes the text area every time an end user opens the Group Chat window of any chat room.
- addNewUser(): Adds a new end user to the list of existing end users in a chat room.
- removeuser(): Removes an end user from the list of existing end users.
- setUserName(): Sets the name of the end user who is currently logged in to the Chat Room application.

The GroupChat.java file allows end users to send messages to other end users in the chat room, as shown in Figure 3-7:

Figure 3-7: Creating a Group Chat Window

◄ PREVIOUS NEXT ►

Unit Testing

To test the Chat Room application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the LoginGUI.java, MainGUI.java, ChatRoom.java, RoomInformation.java, GroupChat.java, and SocketOpener.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

5. To run the Chat Room application, specify the following command at the command prompt:

   ```
   java LoginGUI
   ```

6. The Login window of the Chat Room application appears. Specify the login information, as shown in Figure 3-8:

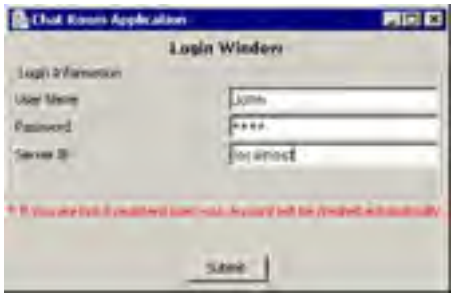


Figure 3-8: Specifying Login Information

7. Click the Submit button to send the login information to the Jabber server for registered end users or open a new account in the Jabber server for unregistered end users.

8. The user interface of the Chat Room application appears. Select File-> Go to Chat Room to specify the name of the chat room to join.

9. Enter the name of the chat room and the user name to join the chat room, as shown in Figure 3-9:

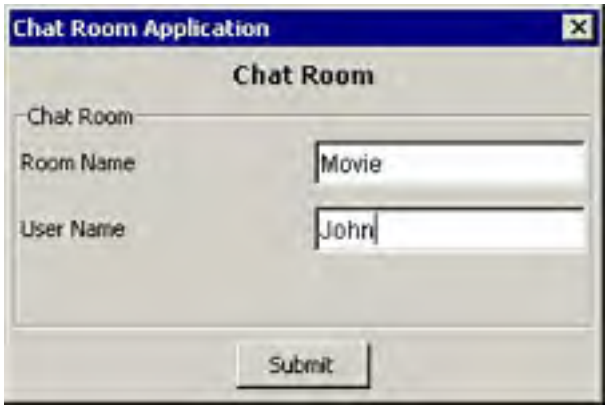


Figure 3-9: Entering a Chat Room

10. Click the Submit button to open the Group Chat window of the specified chat room, as shown in Figure 3-10:

Figure 3-10: The Group Chat Window of the Specified Chat Room

11. To create a chat room, select File-> Create New Chat Room from the user interface of the Chat Room application.
12. Enter the name of the chat room and the user name, as shown in Figure 3-11:

Figure 3-11: Entering Chat Room Name and User Name

13. Click the Submit button to create a new chat room and open the Group Chat window of the new chat room, as shown in Figure 3-12:

Figure 3-12: The Group Chat Window of the New Chat Room

14. Type the required message in the chat window and click the Send button to send the message to all end users logged in to the chat room, as shown in Figure 3-13:

Figure 3-13: Sending a Message

15. To initiate a group chat, select any chat room from the Existing Rooms tree structure of the Chat Room application, as shown in Figure 3-14:

Figure 3-14: Selecting an Existing Room

The Input Dialog box appears to specify the user name for the selected chat room, as shown in Figure 3-15:

Figure 3-15: The Input Dialog Box

16. Enter the user name and click the OK button. The Group Chat window for the selected room appears, as shown in Figure 3-16:

Figure 3-16: The Group Chat Window of the Selected Chat Room

PREVIOUS NEXT

Chapter 4: Creating a Group Chatting Application

Download CD Content

The Jabber protocol allows you to create and rename chat groups hosted on the Jabber server. End users can add user names to a particular chat group and initiate a private or group chat.

This chapter describes how to develop the Group Chatting application, which allows end users to send group messages to other end users connected to the Jabber server.

Architecture of the Group Chatting Application

The Group Chatting application uses the following files:

- GroupLoginGUI.java: Allows end users to establish a connection with the Jabber server to communicate with other end users connected to the server. This file allows end users to log in or register automatically to the Jabber server.
- GroupList.java: Shows the tree structure that contains the names of various chat groups and end users in each group. This file allows end users to select a group to start a chat session. End users can select any specific user within a group to start a private chat.
- AddUser.java: Allows end users to add other end users to a particular group. This file also allows end users to create a new group.
- ChatWindow.java: Allows end users to participate in a private chat.
- EditGroup.java: Allows end users to rename an existing chat group.
- GroupChat.java: Allows end users to participate in a group chat.
- SocketConnection.java: Opens a socket to send and receive messages through the Jabber server.
- GroupTree.java: Creates a tree structure of various groups in the Group Chatting application.

Figure 4-1 shows the architecture of the Group Chatting application:

Figure 4-1: Architecture of the Group Chatting Application

The GroupLoginGUI.java file creates the Login window of the Group Chatting application that contains various labels, text boxes, and the Submit button. The GroupLoginGUI.java file allows end users to log on as an existing user or register for a new account.

When an end user enters the login information and clicks the Submit button on the Login window, the GroupLoginGUI.java file calls the GroupList.java file to create the user interface for the Group Chatting application. The GroupList.java file allows end users to create a group or add other end users to a particular group. The GroupList.java file creates the tree structure to show the list of existing chat groups.

If an end user selects File-> Add User, the GroupList.java file calls the AddUser.java file to add other end users to an end user's personal list. The AddUser.java file also allows end users to create a chat group. The AddUser.java file provides an interface with various labels, two text boxes, and the Submit button.

If an end user selects File-> Edit Group, the GroupList.java file calls the EditGroup.java file to rename an existing group. The EditGroup.java file provides an interface with various labels, two text boxes, and the Submit button.

If an end user selects a specific end user name present in a particular group, the GroupList.java file calls the SocketConnection.java file that opens a socket to send and receive messages to the selected end user. The SocketConnection.java file calls the ChatWindow.java file, which allows an end user to start a private chat with the selected end user. The ChatWindow.java file provides an interface containing a text area to view text messages received from the Jabber server. The ChatWindow.java file also provides the Send button to send text messages to the Jabber server.

If an end user selects a chat group from the tree structure provided by the Group Chatting application, the GroupList.java file calls the SocketConnection.java file. This file opens a socket to send and receive messages to other end users present in the selected group. The SocketConnection.java file calls the GroupChat.java file to start a group chat session.

PREVIOUS NEXT

Creating the Login Window

The Group.java file creates the Login window for the Group Chatting application. Listing 4-1 shows the contents of the GroupLoginGUI.java file:

Listing 4-1: The GroupLoginGUI.java File

```
/*
Import required swing classes
*/
import javax.swing.*;
import javax.swing.text.*;
/*
Import required awt classes
*/
import java.awt.*;
import java.awt.event.*;
/*
Import required io and net classes
*/
import java.io.*;
import java.net.*;
/*
class GroupLoginGUI - This class is used to create a GUI to get the login information from user.
Constructor:
GroupLoginGUI - This constructor creates GUI.
Methods:
getUserName - Used to get the user name
getPassword - Used to get the password
getServarName - Used to get the servername
main - This method creates the main window of the application and displays all the components.
*/
public class GroupLoginGUI extends JFrame implements ActionListener, KeyListener
{
   /*Declare objects of Container class.*/
   Container container=null;
   /*Declare objects of JTextField class.*/
   JTextField namet=new JTextField();
   JTextField server=new JTextField();
   /*Declare objects of JPasswordField class.*/
   JPasswordField passwordt=new JPasswordField();
   /*Declare objects of JButton class.*/
   JButton submit=new JButton("Submit");
   /*Declare objects of SocketConnection class.*/
   Socket clientsocket;
   /*Declare string variable.*/
   String servername;
   String username;
   String password;
   /*Declare objects of JPanel class.*/
   JPanel note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
   JPanel panel=new JPanel();
   JPanel lable=new JPanel();
   /*Declare objects of JLabel class.*/
   JLabel room= new JLabel("User Name");
   JLabel name= new JLabel("Password ");
   JLabel serverl= new JLabel("Server IP");
   JLabel note1= new JLabel("*If you are not a registered user your account will be");
   JLabel note2= new JLabel("created automatically.");
   /*Constructor for class.*/
   public GroupLoginGUI ()
   {
      /*Set the window title*/
      super("Login Window");
      this.clientsocket=clientsocket;
      container=getContentPane();
      panel.setLayout(new GridLayout(4, 2, 3, 3));
      panel.setBorder(BorderFactory.createTitledBorder("Login Information"));
      panel.add(room);
      panel.add(namet);
      panel.add(name);
      panel.add(passwordt);
      panel.add(serverl);
      panel.add(server);
      note1.setForeground(Color.red);
      note2.setForeground(Color.red);
      note.add(note1);
      note.add(note2);
      JLabel heading=new JLabel("Login Window", JLabel.CENTER);
      lable.add(heading,BorderLayout.NORTH);
      heading.setFont(new Font("verdana", 1, 12));
      container.add(lable,BorderLayout.NORTH);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        container.add(panel,BorderLayout.CENTER);
        JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
        JPanel button=new JPanel(new FlowLayout());
        button.add(submit);
        buttonpanel.add(note);
        buttonpanel.add(button);
        container.add(buttonpanel,BorderLayout.SOUTH);
        setBounds(5, 5, 371, 240);
        /*Add the listener to the button*/
        submit.addActionListener(this);
        submit.addKeyListener(this);
        show();
    }
    public void keyPressed(KeyEvent e)
    {
        if(e.getKeyCode()==KeyEvent.VK_ENTER)
        {
        servername=server.getText();
        username=namet.getText();
        password=passwordt.getText();
        GroupList grp=new GroupList(this, servername, username, password);
        }
    }
    public void keyTyped(KeyEvent e)
    {
    }
    public void keyReleased(KeyEvent e)
    {
    }
    public void actionPerformed(ActionEvent e)
    {
        servername=server.getText();
        username=namet.getText();
        password=passwordt.getText();
        try
        {
            GroupList grp=new GroupList(this, servername, username, password);
            hide();
        }
        catch(Exception ee)
        {
        ee.printStackTrace();
        }
    }
    public String getUserName()
    {
        return username;
    }
    public String getPassword()
    {
        return password;
    }
    public String getServarName()
    {
        return servername;
    }
    public static void main(String[] args)
    {
        /*Set the window look and feel*/
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        GroupLoginGUI lgui=new GroupLoginGUI();
    }
}
```

Download this listing.

In the above code, the main() method creates an instance of the GroupLoginGUI class.

The methods defined in Listing 4-1 are:

- `getUserName()`: Retrieves the name of the end user currently logged in.
- `getPassword()`: Retrieves the password specified by the end user.
- `getServerName()`: Retrieves the name of the Jabber server.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs. End users invoke the actionPerformed() method by clicking the Submit button on the Login window.

The GroupLoginGUI.java file creates the Login window of the Group Chatting application, as shown in Figure 4-2:

Figure 4-2: The Login Window

◄ PREVIOUS NEXT ►

Creating the User Interface of the Group Chatting Application

The GroupList.java file creates the user interface of the Group Chatting application. Listing 4-2 shows the contents of the GroupList.java file:

Listing 4-2: The GroupList.java File

```
/*Import required  tree package classes*/
import javax.swing.tree.*;
import javax.swing.*;
/*Import required  awt package classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required  event package classes*/
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeModelEvent;
/*Import required  io package classes*/
import java.io.*;
/*Import required  net package classes*/
import java.net.*;
/*Import required  util package classes*/
import java.util.*;
/*
class GroupList - This class is used to create a GUI to show the group and user.
Constructor:
GroupList-This constructor creates GUI.
Methods:
openPort - Used to open the port
clearTree - Used to remove the tree node
addToTree-Used to set the group and user into tree
*/
public class GroupList extends JFrame implements TreeSelectionListener,ActionListener,Runnable
{
   /*Declare object of JMenuBar class.*/
   JMenuBar jb=new JMenuBar();
   /*Declare object of JMenu class.*/
   JMenu menu=new JMenu("File");
   /*Declare object of JMenuItem class.*/
   JMenuItem edit=new JMenuItem("Edit Group");
   JMenuItem addmember=new JMenuItem("Add User");
   /*Declare string variable.*/
   String firstname="";
   String lastname="";
   String emailid="";
   String ipaddress;
   String servername;
   String username;
   String password;
   String resource="Home";
   String errortype="";
   String roomname;
   String nickname;
   /*Declare integer*/
   int portno=5222;
   int wait=1000;
   /*Declare object of Hashtable class.*/
   Hashtable has=new Hashtable();
   Hashtable hasuser=new Hashtable();
   Hashtable edituser=new Hashtable();
   /*Declare object of Button class.*/
   Button sendbutton;
   Button closebutton;
   /*Declare the boolean variable.*/
   boolean existing;
   private boolean isConnected=false;
   /*Declare object of Thread class.*/
   Thread inputmessagethread;
   /*Declare object of JScrollPane class.*/
   JScrollPane treeView;
   /*Declare object of JTree class.*/
   JTree tree;
   /*Declare object of DefaultMutableTreeNode class.*/
   DefaultMutableTreeNode groupnode;
   DefaultMutableTreeNode groupnode1;
   DefaultMutableTreeNode top= new DefaultMutableTreeNode("Existing Category");
   DefaultTreeModel treeModel;
   /*Declare object of GroupLoginGUI class.*/
   GroupLoginGUI groupLogin;
   /*Declare object of Socket class.*/
   Socket clientsocket1;
   SocketConnection clientsocket;
   /*Declare object of SocketConnection class.*/
```

```
SocketConnection sc;
/*Declare object of GroupTree class.*/
GroupTree dynamic;
/*Declare object of EditGroup class.*/
EditGroup ag;
/*Declare object of ChatWindow class.*/
ChatWindow chat;
/*Constructor for class*/
public GroupList(GroupLoginGUI groupLogin, String server, String user, String password)
{
   super("Group Chatting Application");
   this.servername=server;
   this.username=user;
   this.password=password;
   this.groupLogin=groupLogin;
   this.password=password;
   start1();
   dynamic=new GroupTree(sc, groupLogin, this);
   jb.add(menu);
   menu.add(addmember);
   menu.add(edit);
   this.setJMenuBar(jb);
   edit.addActionListener(this);
   addmember.addActionListener(this);
   setBounds(5,5,350,350);
   setVisible(false);
   sc.sendRosterRequest();
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   getContentPane().add(dynamic);
   show();
}
/*
Function to get object of ChatWindow class
Parameter: N/A
Return Value: ChatWindow
*/
public ChatWindow getChat()
{
   return this.chat;
}
public void setChat(ChatWindow ch)
{
   this.chat=ch;
}
/*
Function to clear the tree
Parameter: N/A
Return Value: N/A
*/
public void clearTree()
{
   dynamic.clearTree();
   edituser.clear();
   has.clear();
   hasuser.clear();
}
public void valueChanged(TreeSelectionEvent e)
{
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)
   tree.getLastSelectedPathComponent();
   TreeNode parent =node.getParent();
   boolean b=node.isLeaf();
   if(b)
   {
      String user=node.toString();
   }
}
public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==edit)
   {
      ag=new EditGroup(this, dynamic,sc);
   }
   if(e.getSource()==addmember)
   {
      AddUser am=new AddUser(this, dynamic, sc);
   }
}
/*
Function to open the port and write the XML messages to server
Parameter: N/A
Return Value: N/A
*/
public void start1()
{
   try
   {
      openPort(servername, 5222, 1000);
```

```
    }
    catch(Exception e)
    {
    }
    if (clientsocket1!=null)
    {
       isConnected=true;
    }
    Thread inputthread=new Thread(this);
    inputthread.start();
    try
    {
       String sessionStratString;
       String authentication;
       String registrationstring;
       sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
       sessionStratString=sessionStratString+" <stream:stream";
       sessionStratString=sessionStratString+" to= \""+servername +"\"";
       sessionStratString=sessionStratString+" xmlns=\"jabber:client\"";
       sessionStratString=sessionStratString+"
       xmlns:stream=\"http://etherx.jabber.org/streams\">";
       sc.sendXMLToJabber(sessionStratString);
       System.out.println("sessionStratString"+sessionStratString);
       if ((firstname.equals("false"))||(lastname.equals("false"))||(emailid.equals("false")))
       {
          System.out.println("sessionStratString if"+sessionStratString);
       }
       else
       {
          registrationstring="<iq type=\"set\" to=\""+username+"@localhost\" id=\"1001\">";
          registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
          registrationstring=registrationstring+"<username>"+username+"</username>";
          registrationstring=registrationstring+"<password>"+password+"</password>";
          registrationstring=registrationstring+"<first>"+firstname+"</first>";
          registrationstring=registrationstring+"<last>"+lastname+"</last>";
          registrationstring=registrationstring+"<email>"+emailid+"</email>";
          registrationstring=registrationstring+" </query> ";
          registrationstring=registrationstring+"</iq>";
          System.out.println("registrationstring"+registrationstring);
       }
       authentication="<iq type=\"set\" id=\"1301\">";
       authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
       authentication=authentication+" <username>"+username+"</username>";
       authentication=authentication+" <password>"+password+"</password>";
       authentication=authentication+" <resource>"+resource+"</resource> ";
       authentication=authentication+" </query> ";
       authentication=authentication+"</iq>";
       sc.sendXMLToJabber(authentication);
       System.out.println("authentication"+authentication);
    }
    catch(Exception ie)
    {
    }
}
/*
Function to get the user name
Parameter: N/A
Return Value: String
*/
public String getUserName()
{
    String user=dynamic.getNodeName();
    return user;
}
public void run()
{
    System.out.println("Socket is opened runnnn"+sc);
    sc.runinput();
}
/*
Function to open port
Parameter: ipaddress-Object of string class
portno-int
timeinsec-int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
    String host;
    if (ipaddress.trim()=="")
    {
    }
    else
    {
       sc=new SocketConnection(ipaddress,portno,this,dynamic,groupLogin);
       clientsocket=sc.openSocket(ipaddress,portno,timeinsec);
       if(clientsocket==null)
       {
```

```
            System.out.println("Socket is opened null null"+clientsocket);
         }
      }
   }
   public static void main(String arg[])
   {
      /*Set the window look and feel*/
      try
      {
         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
   }
   /*
   Function to add the user list and user to the tree
   Parameter: groupname-Object of string class
   jid-Object of string class
   contactpersonname-Object of string class
   Return Value: N/A
   */
   public void addToTree(String groupname,String jid,String contactpersonname)
   {
      edituser.put(jid,groupname);
      boolean elementfound=false;
      groupnode=new DefaultMutableTreeNode(groupname);
      Enumeration e=has.elements();
      while(e.hasMoreElements())
      {
         String haselm=(String)e.nextElement();
         if(haselm.equals(groupname))
         {
            elementfound=true;
         }
      }
      if(!elementfound)
      {
         if(groupnode!=null)
         {
         if(!(hasuser.containsValue(groupnode)))
         {
            groupnode = dynamic.addObject(null, groupname);
            groupnode1=newDefaultMutableTreeNode(contactpersonname);
            groupnode.add(groupnode1);
            hasuser.put(groupname,groupnode);
            has.put(jid,groupname);
         }
      }
   }
   else
   {
      if(!hasuser.containsValue(contactpersonname.trim()))
      {
         DefaultMutableTreeNode d2=(DefaultMutableTreeNode)hasuser.get(groupname);
         dynamic.addObject(d2,contactpersonname);
         hasuser.put(jid,contactpersonname.trim());
         has.put(jid,groupname);
      }
   }
}
/*
Function to write the text into the textpane
Parameter:messagebody-Object of string class
user-Object of string class
Return Value: N/A
*/
public void insertTextInToTextPane(String messagebody,String user)
{
   String[] splitedusername=user.split("@");
   ChatWindow chatwindowobject=(ChatWindow)dynamic.mapObj.get(splitedusername[0]);
   if (chatwindowobject!=null)
   {
      chatwindowobject.insertTextInToTextPane(messagebody,splitedusername[0]);
   }
   else
   {
      ChatWindow ch=new ChatWindow(username,splitedusername[0],servername,sc);
      ch.insertTextInToTextPane(messagebody,splitedusername[0]);
      dynamic.mapObj.put(splitedusername[0],ch);
   }
}
}
```

Download this listing.

In the above code, the constructor of the GroupList class takes an object of the GroupLoginGUI class and three strings as input parameters: user, password, and server. The user string retrieves the user name specified by an end user. The password string retrieves the password specified by an end user, and the server string retrieves the name of the Jabber server specified by an end user. The object of the GroupLoginGUI class allows an end user to invoke the methods of the GroupLoginGUI class.

The methods defined in Listing 4-2 are:

- `openPort()`: Opens a client socket by using the Internet Protocol (IP) address and port number of the Jabber server.
- `clearTree()`: Refreshes the tree structure containing group and end user names.
- `addToTree()`: Adds the newly created group to the tree structure of the available groups.
- `getChat()`: Retrieves the object of the ChatWindow class.
- `insertTextInToTextPane()`: Inserts the text message specified by an end user into the text area of the chat window.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.

The GroupList.java file creates the main window of the Group Chatting application, as shown in Figure 4-3:

Figure 4-3: User Interface of the Group Chatting Application

Select the File menu to view the File menu options.

Figure 4-4 shows the File menu options of the Group Chatting application:

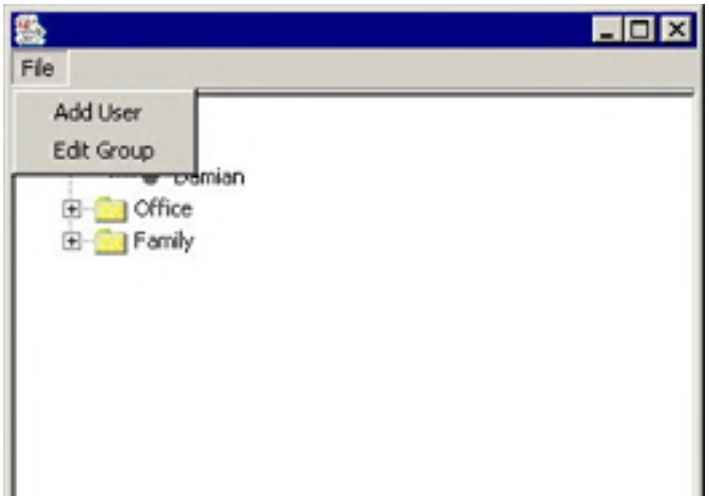


Figure 4-4: The File Menu of the Group Chatting Application

Select File-> Add User to add other end users to an end user's personal list of a particular chat group.

Select File-> Edit Group to rename an existing group.

Select any group or any specific end user from a particular group to initiate the chat.

◄ PREVIOUS NEXT ►

Adding Users

The AddUser.java file creates the user interface to add other end users to a particular group. Listing 4-3 shows the contents of the AddUser.java file:

Listing 4-3: The AddUser.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
import javax.swing.tree.*;
/*
class AddUser - This class is used to create a GUI to take the information from user.
Constructor:
AddUser-This constructor creates GUI.
*/
public class AddUser extends JFrame implements ActionListener
{
/*Declare object of Container class.*/
Container container=null;
/*Declare object of JPanel class.*/
JPanel panel=new JPanel();
JPanel lable=new JPanel();
JPanel buttonpanel=new JPanel(new GridLayout(1,1,5,5));
JPanel bp=new JPanel(new FlowLayout(FlowLayout.CENTER));
/*Declare object of JLabel class.*/
JLabel membername= new JLabel("User Name");
JLabel groupbername= new JLabel("Group Name");
JLabel heading=new JLabel("User Information",JLabel.CENTER);
/*Declare object of JTextField class.*/
JTextField membernamet=new JTextField();
JTextField groupnamet=new JTextField();
/*Declare object of JButton class.*/
JButton submit=new JButton("Submit");
/*Declare object of GroupList class.*/
GroupList group;
/*Declare object of GroupTree class.*/
GroupTree dynamic;
/*Declare object of SocketConnection class.*/
SocketConnection clientsocket;
/*Declare string variable.*/
String servername="localhost";
String friends=new String("Friends");
String family=new String("Family");
String office=new String("Office");
String colleague=new String("Colleague");
String selectedgroup="";
/*Constructor */
public AddUser (GroupList group,GroupTree dynamic,SocketConnection clientsocket)
{
   /*Set the window title*/
   super("Group Chatting Application");
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   this.group=group;
   this.dynamic=dynamic;
   this.clientsocket=clientsocket;
   container=getContentPane();
   /*set the panel's layout*/
   panel.setLayout(new GridLayout(2, 1, 1, 1));
   panel.setBorder(BorderFactory.createTitledBorder("Add User" ));
   /*Add the component to panel*/
   panel.add(membername);
   panel.add(membernamet);
   panel.add(groupbername);
   panel.add(groupnamet);
   /*Add the label to panel*/
   lable.add(heading,BorderLayout.NORTH);
   /*set the heading of font*/
   heading.setFont(new Font("verdana", 1, 12));
   /*Add the panel to container*/
```

```
    container.add(lable,BorderLayout.NORTH);
    /*Set the default close operation for window*/
    setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    /*Add the panel to container*/
    container.add(panel,BorderLayout.CENTER);
    /*Add the submit button to panel*/
    bp.add(submit);
    buttonpanel.add(bp);
    /*Add the panel to container*/
    container.add(buttonpanel,BorderLayout.SOUTH);
    setBounds(5,5,250,160);
    /*Add the listener to the button*/
    submit.addActionListener(this);
    /*Called the show function to display the window*/
    show();
}
/*Method to describe the event*/
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==submit)
    {
        String user=membernamet.getText();
        selectedgroup=groupnamet.getText();
        String from=dynamic.groupLogin.getUserName();
        String messagestring="<message type='chat'  from='"+from+"@localhost/Home'
        to='"+user+"@"+servername+"/Home' >";
        messagestring=messagestring+"<body>";
        messagestring=messagestring+"";
        messagestring=messagestring+"</body></message>";
        clientsocket.sendXMLToJabber(messagestring);
        String tag=clientsocket.tagentity;
        String userGroup="<iq type=\"set\" id=\"uniquevalue\">";
        userGroup=userGroup+"<query xmlns=\"jabber:iq:roster\">";
        userGroup=userGroup+"<item jid=\""+user+"@localhost/Home\"  name=\""+user+"\" ";
        userGroup=userGroup+"subscription=\"none\"  ask=\"subscribe\">";
        userGroup=userGroup+"<group>"+selectedgroup+"</group></item></
        query></iq>";
        System.out.println("UserGroup"+userGroup);
        clientsocket.createGroup(userGroup);
        clientsocket.sendRosterRequest();
        hide();
    }
}
}
```

Download this listing.

In the above code, the constructor of the AddUser class takes an object of the GroupList class, GroupTree class, and SocketConnection class as input parameters. This allows end users to invoke the methods of the GroupList class, GroupTree class, and SocketConnection class. The AddUser class defines the actionPerformed() method, which acts as an event listener and activates an appropriate method based on the action an end user performs.

The AddUser.java file creates the window to add other end users to a group, as shown in Figure 4-5:

Figure 4-5: Adding an End User

Renaming the Group

The EditGroup.java file creates the user interface to rename an existing chat group. Listing 4-4 shows the contents of the EditGroup.java file:

Listing 4-4: The EditGroup.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
import javax.swing.tree.*;
import java.util.*;
/*
class EditGroup - This class is used to create a GUI to take the information from user.
Constructor:
EditGroup-This constructor creates GUI.
Methods:
getOldGroupName:-get the froup name
main - This method creates the look and feel.
*/
public class EditGroup extends JFrame implements ActionListener
{
/*Declare objects of Container class.*/
Container container=null;
/*Declare objects of panel class.*/
JPanel panel=new JPanel();
JPanel lable=new JPanel();
JPanel buttonpanel=new JPanel(new GridLayout(1,1,5,5));
JPanel bp=new JPanel(new FlowLayout(FlowLayout.CENTER));
/*Declare objects of label  class.*/
JLabel oldgroupname= new JLabel("Old Group Name");
JLabel newgroupname= new JLabel("New Group Name");
JLabel heading=new JLabel("Rename Group", JLabel.CENTER);
JTextField oldgroupnamet=new JTextField();
JTextField newgroupnamet=new JTextField();
/*Declare objects of button class.*/
JButton submit=new JButton("Submit");
/*Declare objects of GroupList class.*/
GroupList group;
/*Declare objects of DefaultMutableTreeNode class.*/
DefaultMutableTreeNode newgroup;
/*Declare objects of GroupTree class.*/
GroupTree dynamic;
/*Declare objects of SocketConnection  class.*/
SocketConnection clientsocket;
/*Declare objects of string variable.*/
String userGroup="";
String ngroupname="";
String ogroupname="";
/*Constructor*/
public EditGroup (GroupList group, GroupTree dynamic, SocketConnection clientsocket)
{
   /*Set the window title*/
   super("Group Chatting Application");
   /*Set the window look and feel*/
   try
   {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
   this.group=group;
   this.dynamic=dynamic;
   this.clientsocket=clientsocket;
   container=getContentPane();
   /*Set the layout of panel*/
   panel.setLayout(new GridLayout(2, 1, 5, 5));
   /*Set the border of panel*/
   panel.setBorder(BorderFactory.createTitledBorder("Group Information"));
   /*Add the component to the panel*/
   panel.add(oldgroupname);
   panel.add(oldgroupnamet);
   panel.add(newgroupname);
   panel.add(newgroupnamet);
   lable.add(heading,BorderLayout.NORTH);
   /*Set the font of heading*/
```

```
   heading.setFont(new Font("verdana", 1, 12));
   container.add(lable,BorderLayout.NORTH);
   /*Set the default close operation*/
   setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
   container.add(panel,BorderLayout.CENTER);
   bp.add(submit);
   buttonpanel.add(bp);
   /*Add panel to the container*/
   container.add(buttonpanel,BorderLayout.SOUTH);
   /*Set the bounds for window*/
   setBounds(5, 5, 250, 160);
   /*Add the action listener to the button*/
   submit.addActionListener(this);
   /*Called the show function to display the window*/
   show();
}
/*
Function to get the groupname
Parameter: N/A
Return Value: N/A
*/
public String getOldGroupName()
{
   return ogroupname;
}
/* Method to describe the event*/
public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==submit)
   {
      ngroupname=newgroupnamet.getText();
      ogroupname=oldgroupnamet.getText();
      Vector vec=new Vector();
      Enumeration ee=group.edituser.keys();
      while(ee.hasMoreElements())
      {
         String haselm=(String)ee.nextElement();
         String usertoedit=haselm.substring(1,haselm.length());
         String spliteduser[]=usertoedit.split("@");
         Object nname=group.edituser.get(haselm);
         String nnamestr=nname.toString();
         if(nnamestr.equals(ogroupname))
         {
            vec.add(spliteduser[0]);
         }
      }
      for(int j=0;j<vec.size();j++)
      {
         userGroup="<iq type=\"set\" id=\"uniquevalue\">";
         userGroup=userGroup+"<query xmlns=\"jabber:iq:roster\">";
         userGroup=userGroup+"<item jid=\""+(String)vec.elementAt(j)+"@localhost/Home\"
         name=\""+(String)vec.elementAt(j)+"\" ";
         userGroup=userGroup+"subscription=\"none\"  ask=\"subscribe\">";
         userGroup=userGroup+"<group>"+ngroupname+"</group></item></
         query></iq>";
         clientsocket.createGroup(userGroup);
      }
      clientsocket.sendRosterRequest();
      }
   }
}
```

Download this listing.

In the above code, the constructor of the EditGroup class takes an object of the GroupList class, GroupTree class, and SocketConnection class as input parameters. The EditGroup class allows end users to invoke the methods of the GroupList class, GroupTree class, and SocketConnection class.

The methods defined in Listing 4-4 are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.
- `getOldGroupName()`: Retrieves the old name of a group after an end user renames the group.

The EditGroup.java file creates the Rename Group window to rename a group, as shown in Figure 4-6:

Figure 4-6: Renaming a Group

PREVIOUS NEXT

Initiating a Private Chat

The ChatWindow.java file creates the user interface that allows end users to initiate a private chat. Listing 4-5 shows the contents of the ChatWindow.java file:

Listing 4-5: The ChatWindow.java File

```
/*Import required swing classes*/
import javax.swing.*;
/*Import required swing event classes*/
import javax.swing.event.*;
/*Import required swing text classes*/
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
public class  ChatWindow extends JFrame implements ActionListener
{
   /*Declare object of Container class.*/
   Container container;
   GridBagLayout gridbaglayout;
   /*Declare object of JLabel class.*/
   JLabel fromuser = null;
   JLabel touser = null;
   JLabel fromusername = null;
   JLabel tousername = null;
   /*Declare object of JTextPane class.*/
   JTextPane textarea = null;
   JTextField textfield = null;
   /*Declare object of JButton class.*/
   JButton sendbutton = null;
   /*Declare object of JScrollPane class.*/
   JScrollPane scpane;
   /*Declare string variable.*/
   String user="";
   String receiver="";
   String servername="";
   String messagestring="";
   /*Declare object of SocketConnection class.*/
   SocketConnection clientsocket;
   Document contentmodel;
   MutableAttributeSet recervernameatrib;
   ChatWindow chat;
   /*
   class ChatWindow - This class is used to create a GUI for chatting.
   Constructor:
   ChatWindow-This constructor creates GUI.
   Methods:
   setObj - Used to set the use object of ChatWindow
   insertTextInToTextPane - Used to write the text into the textpane
   */
   /* Constructor for class*/
   public ChatWindow(String user, String receiver, String servername, SocketConnection clientsocket)
   {
      this.user=user;
      this.receiver=receiver;
      this.servername=servername;
      this.clientsocket=clientsocket;
      fromuser = new JLabel("From :");
      fromusername  = new JLabel(user);
      String touser1= receiver;
      touser = new JLabel("To :");
      tousername = new JLabel(touser1);
      /*Set the window title*/
      setTitle("Group Chatting Application: "+touser1);
      container = this.getContentPane();
      recervernameatrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(recervernameatrib,"Verdana");
      StyleConstants.setForeground(recervernameatrib,Color.red);
      gridbaglayout=new GridBagLayout();
      GridBagConstraints gridbagconstraints=new GridBagConstraints();
      /*Creating a panel with gridbaglayout*/
      JPanel jpanel1 = new JPanel(gridbaglayout);
      gridbagconstraints.fill=GridBagConstraints.BOTH;
      gridbagconstraints.insets=new Insets(5, 5, 0, 0);
      gridbagconstraints.gridx=0;
      gridbagconstraints.gridy=0;
      gridbagconstraints.weightx=1;
      gridbagconstraints.weighty=1;
      gridbagconstraints.anchor=GridBagConstraints.WEST;
      gridbaglayout.setConstraints(fromuser, gridbagconstraints);
      fromuser.setPreferredSize(new Dimension(140, 28));
      /*Adding component to panel*/
      jpanel1.add(fromuser);
```

```
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 20);
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(fromusername,gridbagconstraints);
    fromusername.setPreferredSize(new Dimension(20, 28));
    fromusername.setFont(new Font("Verdana", Font.BOLD, 10));
    /*Adding the component to panel*/
    jpanel1.add(fromusername);
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=2;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(touser,gridbagconstraints);
    touser.setPreferredSize(new Dimension(140, 28));
    /*Adding the component to panel*/
    jpanel1.add(touser);
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=3;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.WEST;
    gridbaglayout.setConstraints(tousername, gridbagconstraints);
    tousername.setPreferredSize(new Dimension(20,28));
    tousername.setFont(new Font("Verdana", Font.BOLD, 10));
    /*Adding component to panel*/
    jpanel1.add(tousername);
    /*Adding panel to container*/
    container.add(jpanel1,BorderLayout.NORTH);
    /*Creating a new panel*/
    JPanel jpanel2 = new JPanel();
    /*Creating new textpane*/
    textarea = new JTextPane();
    contentmodel=textarea.getDocument();
    /*Set the size of textpane*/
    textarea.setPreferredSize(new Dimension(80, 20));
    /*Creating a scrollpane and adding a textpane in to the scrollpane*/
    scpane  = new JScrollPane(textarea,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.
    HORIZONTAL_SCROLLBAR_AS_NEEDED);
    container.add(scpane,BorderLayout.CENTER);
    /*Creating a panel with gridbaglayout*/
    JPanel jpanel3 = new JPanel(gridbaglayout);
    textfield = new JTextField();
    /*Creating a object of send button*/
    sendbutton = new JButton("Send");
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=0;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=1;
    gridbagconstraints.weighty=1;
    gridbagconstraints.anchor=GridBagConstraints.NORTH;
    gridbaglayout.setConstraints(textfield, gridbagconstraints);
    textfield.setPreferredSize(new Dimension(120, 28));
    /*Adding a textfield into panel*/
    jpanel3.add(textfield);
    gridbagconstraints.fill=GridBagConstraints.BOTH;
    gridbagconstraints.insets=new Insets(5, 5, 0, 0);
    gridbagconstraints.gridx=1;
    gridbagconstraints.gridy=0;
    gridbagconstraints.weightx=0;
    gridbagconstraints.weighty=0;
    gridbagconstraints.anchor=GridBagConstraints.EAST;
    gridbaglayout.setConstraints(sendbutton, gridbagconstraints);
    sendbutton.setPreferredSize(new Dimension(60, 28));
    /*Adding a send button into panel*/
    jpanel3.add(sendbutton);
    /*Adding action listener in to sendbutton*/
    sendbutton.addActionListener(this);
    /*Adding action listener in to textfield*/
    textfield.addActionListener(this);
    /*Adding a panel to container*/
    container.add(jpanel3,BorderLayout.SOUTH);
    /*Set the bounds for window*/
    setBounds(100,100,350,280);
    /*Called the show method to display the window*/
    show();
}
/*Method to describe the event*/
```

```
   public void actionPerformed(ActionEvent e)
   {
      if(e.getSource()==sendbutton)
      {
         if (!textfield.getText().trim().equals(""))
         {
         String[] usersplited=user.split("@");
         messagestring="<message type='chat' from='"+usersplited[0]+"@localhost/Home'
         to='"+receiver+"@"+servername+"/Home' >";
         messagestring=messagestring+"<body>";
         messagestring=messagestring+textfield.getText().trim();
         messagestring=messagestring+"</body></message>";
         try
         {
            contentmodel.insertString(contentmodel.getLength(), "\n"+user+":
            "+textfield.getText(),recervernameatrib);
         }
         catch(Exception ble)
         {
         }
         clientsocket.sendXMLToJabber(messagestring);
         textfield.setText("");
      }
   }
   if(e.getSource()==textfield)
   {
   if (!textfield.getText().trim().equals(""))
   {
      String[] usersplited=user.split("@");
      messagestring="<message type='chat' from='"+usersplited[0]+"@localhost/Home'
      to='"+receiver+"@"+servername+"/Home' >";
      messagestring=messagestring+"<body>";
      messagestring=messagestring+textfield.getText().trim();
      messagestring=messagestring+"</body></message>";
      try
      {
         contentmodel.insertString(contentmodel.getLength(), "\n"+user+":
         "+textfield.getText(), recervernameatrib);
      }
      catch(Exception ble)
      {
      }
      clientsocket.sendXMLToJabber(messagestring);
      textfield.setText("");
      }
   }
}
/*
Function to set the use object of ChatWindow
Parameter: chat - Object of ChatWindow class.
Return Value: N/A
*/
public void setObj(ChatWindow chat)
{
   this.chat=chat;
}
/*
Used to write the text into the textpane
Parameter: message1 - Object of String class.
sendername1-Object of String class.
Return Value: N/A
*/
public void insertTextInToTextPane(String message1, String sendername1)
{
   try
   {
      contentmodel.insertString(contentmodel.getLength(), "\n"+sendername1+":
      "+message1,recervernameatrib);
   }
   catch(BadLocationException ble)
   {
   }
}
}
```

Download this listing.

In the above code, the constructor of the ChatWindow class takes the object of the SocketConnection class and three strings as input parameters: user, server, and receiver. The user string retrieves the name of an end user. The server string retrieves the name of the Jabber server and the receiver string retrieves the name of the end user at the receiving end. The object of the SocketConnection class helps in invoking the methods of the SocketConnection class.

The methods defined in Listing 4-5 are:

- `setObj()`: Sets the object of the ChatWindow class.

- `insertTextInToTextPane()`: Inserts the text message specified by an end user into the text area provided by the Group Chatting application.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The ChatWindow.java file creates a Private Chat window that allows the end user to initiate a private chat, as shown in Figure 4-7:

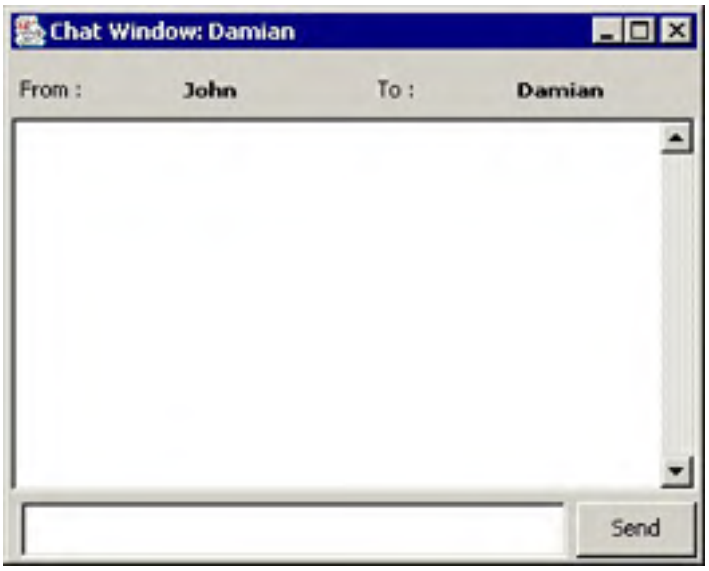


Figure 4-7: Private Chat Window

PREVIOUS NEXT

Initiating a Group Chat

The GroupChat.java file creates the user interface that allows end users to participate in a group chat. Listing 4-6 shows the contents of the GroupChat.java file:

Listing 4-6: The GroupChat.java File

```
/*Import required swing classes*/
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required util classes*/
import java.util.*;
/*
class GroupChat - This class is used to create a GUI for chatting.
Constructor:
GroupChat-This constructor creates GUI.
Methods:
clearPane - Used to clear the text
addElement - Used to add user to the list
writeintotextfield-Used to write text into textpane
*/
public class  GroupChat extends JFrame implements ActionListener
{
   /*Declare object of Container class.*/
   Container cont = null;
   /*Declare object of JPanel class.*/
   JPanel jp=new JPanel(new BorderLayout());
   /*Declare object of JButton class.*/
   JButton button=new JButton("Send");
   /*Declare object of JTextField class.*/
   JTextField text=new JTextField();
   /*Declare object of JList class.*/
   JList list;
   /*Declare object of DefaultListModel class.*/
   private DefaultListModel listModel;
   /*Declare object of JTextPane class.*/
   JTextPane  je=new JTextPane();
   /*Declare object of JScrollPane class.*/
   JScrollPane scrollPane1 = new JScrollPane(je);
   JScrollPane scrollPane ;
   /*Declare object of JSplitPane class.*/
   JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
   /*Declare the string variable*/
   String chatText=null;
   String roomname;
   String user;
   String username="";
   String servername;
   /*Declare object of Document class.*/
   Document contentmodel;
   /*Declare object of MutableAttributeSet class.*/
   MutableAttributeSet nicknameattrib;
   SocketConnection sc;
   /*Used to clear the text*/
   public void clearPane()
   {
      je.setText("");
   }
   public void addElement(String element)
   {
      listModel.addElement(element);
   }
   /*Constructor for class*/
   public GroupChat(String user, String servername, String roomname, SocketConnection clientsocket)
   {
      /*Set the title for window*/
      super(" Group Chatting Application: "+roomname);
      cont = getContentPane();
      /*Set the textpane noneditable*/
      je.setEditable(false);
      /*Set the attribute for the font*/
      nicknameattrib=new SimpleAttributeSet();
      StyleConstants.setFontFamily(nicknameattrib, "Verdana");
      StyleConstants.setForeground(nicknameattrib, Color.black);
      contentmodel=je.getDocument();
      this.roomname=roomname;
      this.user=user;
      this.servername=servername;
      /*Create the instance of DefaultListModel*/
      listModel=new DefaultListModel();
```

```
        String groupmessage="<presence from=\""+user+"@conference.localhost/Home\"
        to=\""+roomname.trim()+"@conference."+servername+"/"+user.trim()+"\"/>";
        sc=clientsocket;
        sc=sc.getSocketClass();
        sc.setGroupuser(this);
        /*Called the sendXMLToJabber function to send the message to jabber server*/
        sc.sendXMLToJabber(groupmessage);
        /*Add the listModel in to List */
        list=new JList(listModel);
        /*Add the list into JScrollPane*/
        scrollPane= new JScrollPane(list);
        list.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
        list.setVisibleRowCount(5);
        /*Set the scrollpane into splitpane*/
        splitPane.setTopComponent(scrollPane1);
        splitPane.setBottomComponent(scrollPane);
        /*Set the divider of splitpane*/
        splitPane.setDividerLocation(400);
        splitPane.setPreferredSize(new Dimension(700, 200));
        list.setSelectionBackground(Color.cyan);
        /*Set the size of button*/
        button.setPreferredSize(new Dimension(130, 20));
        /*Set the size of textfield*/
        text.setPreferredSize(new Dimension(400, 20));
        /*Add the text field in to panel*/
        jp.add(text,BorderLayout.WEST);
        /*Add the button field in to panel*/
        jp.add(button,BorderLayout.EAST);
        /*Add the panel in to container*/
        cont.add(jp,BorderLayout.SOUTH);
        cont.add(splitPane,BorderLayout.CENTER);
        /*Set bonds for window*/
        setBounds(5,5,550,550);
        getRootPane().show();
        /*Add the listener to the button*/
        button.addActionListener(this);
        /*Called the function to show the window*/
        this.show();
    }
    /*
    Used to write text into textpane
    Parameter: writeintotextfield-Object of String class
    psendernameattrib-Object of MutableAttributeSet class
    Return Value: N/A
    */
    public void writeintotextfield(String writeintotextfield, MutableAttributeSet psendernameattrib)
    {
        try
        {
            scontentmodel.insertString(contentmodel.getLength(), writeintotextfield, psendernameattrib)
        }
    catch(Exception e)
    {
    }
}
/*
Method to describe the event
Parameter: e-Object of ActionEvent class
Return Value: N/A
*/
public void actionPerformed(ActionEvent e)
{
    String sendxmlstring="<message to=\""+roomname+"@conference."+servername+"\"
    from=\""+user+"@conference.localhost/Home\" type=\"groupchat\">";
    sendxmlstring=sendxmlstring+"<body>"+text.getText()+"</body></message>";
    sc.sendXMLToJabber(sendxmlstring);
    text.setText("");
}
}
```

Download this listing.

In the above code, the constructor of the GroupChat class takes the object of the SocketConnection class and three strings as input parameters: user, server, and roomname. The user string retrieves the name of an end user. The server string retrieves the name of the Jabber server, and the roomname string retrieves the name of the chat room, which an end user selects. The object of the SocketConnection class helps in invoking the methods of the SocketConnection class.

The methods defined in Listing 4-6 are:

- clearPane(): Refreshes the text area every time an end user selects a particular group.
- addElement(): Adds other end users to the selected group.
- itemintotextfield(): Inserts the text message specified by an end user into the text area provided by the Group Chatting application.

- actionPerformed(): Acts as an event listener and activates an appropriate method based on the action an end user performs.

The GroupChat.java file creates the Group Chat window that allows end users to participate in a group chat, as shown in Figure 4-8:

Figure 4-8: The Group Chat Window

PREVIOUS NEXT

Creating the Socket Class to Send and Receive Messages

The SocketConnection.java file creates a socket with the Jabber server to send and receive messages between end users. Listing 4-7 shows the contents of the SocketConnection.java file:

Listing 4-7: The SocketConnection.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*Import required util classes*/
import java.util.*;
import java.lang.*;
import javax.swing.tree.*;
/*
class SocketConnection - This class is used to create a GUI to show the group and user.
Constructor:
SocketConnection-This constructor creates GUI.
Methods:
openSocket - Used to open the port.
socketope - Used to instantiate the socket class.
getErrorString-Used to get the error string.
sendername-Used to get the sendername.
checkForError-Used to check XML error.
runinput-Used to read the server output
*/
public  class  SocketConnection
{
   /*Declare the integer*/
   private int openerport;
   /*Declare object of Socket class.*/
   static private Socket socket;
   /*Declare object of PrintWriter class.*/
   static PrintWriter out = null;
   /*Declare object of BufferedReader class.*/
   static BufferedReader in = null;
   /*Declare object of MutableAttributeSet class.*/
   static MutableAttributeSet sendernameattrib,nameattribut,informationattrib,welcomenameattrib ;
   static Document contentmodel;
   /*Declare string variables.*/
   private String openerhost;
   static String errortype="";
   String roomname="";
   String firstname="";
   String lastname="";
   String emailid="";
   String resource="Home";
   String temproomname="";
   String username="";
   String password="";
   String groupname="";
   String nickname="";
   String servererror="";
   String chattype="";
   String contactpersonname="";
   String jid="";
   String sendername="";
   String tagentity="";
   /*Declare boolean variables*/
   private  static boolean isConnected=false;
   boolean waitForResult=false;
   boolean rosterflag=false;
   boolean waitforreg=false;
   boolean waitforauth=false;
   boolean waitforgroup=false;
   boolean groupnameb=false;
   boolean nouserexists=false;
   /*Declare object of GroupChat class.*/
   GroupChat groupuser;
   /*Declare object of GroupList class.*/
   GroupList group;
   /*Declare object of ChatWindow class.*/
   ChatWindow chat;
   /*Declare object of GroupTree class.*/
   GroupTree dynamic;
   /*Declare object of GroupLoginGUI class.*/
   GroupLoginGUI logingui;
```

```
/*Declare object of nicknamevector class.*/
Vector nicknamevector=new Vector();
/*Declare object of HashMap class.*/
HashMap temphashmap=new HashMap();
/*Declare object of Hashtable class.*/
Hashtable onlineuser=new Hashtable();
/*function to open socket*/
public  SocketConnection openSocket(String hostip,int portnumber,int timeinsec)
{
   sendernameattrib=new SimpleAttributeSet();
   StyleConstants.setFontFamily(sendernameattrib,"Verdana");
   StyleConstants.setForeground(sendernameattrib,Color.black);
   welcomenameattrib=new SimpleAttributeSet();
   StyleConstants.setFontFamily(welcomenameattrib,"Verdana");
   StyleConstants.setForeground(welcomenameattrib,Color.blue);
   nameattribut=new SimpleAttributeSet();
   StyleConstants.setFontFamily(nameattribut,"Verdana");
   StyleConstants.setForeground(nameattribut,Color.red);
   informationattrib=new SimpleAttributeSet();
   StyleConstants.setFontFamily(informationattrib,"Verdana");
   StyleConstants.setItalic(informationattrib,true);
   StyleConstants.setForeground(informationattrib,Color.red);
   socketope();
   try
   {
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(
      socket.getInputStream()));
   }
   catch(Exception exception)
   {
      JOptionPane.showMessageDialog(logingui,"Server not found","Server not
      found",JOptionPane.ERROR_MESSAGE);
   }
   return getSocketClass();
}
/*
Used to create instance of socket class
Parameter: N/A
Return Value: N/A
*/
public  void socketope()
{
   try
   {
      socket=new Socket(openerhost,openerport);
   }
   catch(IOException ie)
   {
   }
}
/* Constructor for class */
public SocketConnection(String host,int port,GroupList group,GroupTree dynamic,
GroupLoginGUI logingui)
{
   this.group=group;
   this.dynamic=dynamic;
   this.logingui=logingui;
   socket=null;
   openerhost=host;
   openerport=port;
}
/* Used to get the reference of socket class*/
public SocketConnection getSocketClass()
{
   return this;
}
/*
Function to write a get error
Parameter: codeid-Object of string class.
Return Value: String
*/
public static String getErrorString(String codeid)
{
   if (codeid=="'401'")
   {
      errortype="minor";
      return "You have sent malformed syntax, which can not be understood by server.";
   }
   if (codeid=="'404'")
   {
      errortype="major";
      return "No Server exist .";
   }
   if (codeid=="'405'")
   {
      errortype="minor";
      return "You have no right to send this command.";
```

```
            }
            if (codeid=="'409'")
            {
               errortype="major";
               return "A user with this jabber id is already exist. Please choose another user id.";
            }
            if (codeid=="'403'")
               {
               errortype="auth";
               return "Invalid password";
            }
            return "";
         }
         /*
         Function used to get the sendername
         Parameter: tagentity-Object of string class.
         Return Value: N/A.
         */
         public void sendername(String tagentity)
         {
            if (tagentity.startsWith("message"))
            {
               sendername="";
               if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
               {
                  sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-2);
               }
            }
         }
         /*
         Function to check xml error
         Parameter: tagentity-Object of string class
         Return Value: String
         */
         public static String checkForError(String tagentity)
         {
         String error="";
         String codeid="";
         if (tagentity.indexOf("error")>0)
         {
         if (tagentity.indexOf("from=")>0)
         {
            codeid=tagentity.substring(tagentity.indexOf("id=")+3,tagentity.indexOf("type="));
         }
      }
      if (tagentity.indexOf("result")>0)
      {
      }
      return error;
      }
      /*
      Used to set the object of GroupUser
      Parameter: gc-Object of GroupChat class
      Return Value: N/A
      */
      public void setGroupuser(GroupChat gc)
      {
         groupuser=gc;
      }
      /*
      Used to read the server output
      Parameter: N/A
      Return Value: N/A
      */
      public void runinput()
      {
         int i=0;
         String errororwarning="";
         String errorstring="";
         String inputstring="";
         boolean starttag=false;
         boolean endtag=false;
         boolean startwritting=false;
         String messagebody="";
         Vector tagvactor;
         tagvactor=new Vector();
         int vactorindex=0;
         int starttagsing=0;
         while (true)
         {
            try
            {
               i=in.read();
               if (i==-1)
               {
                  break;
               }
               else
```

```
{
   inputstring=inputstring+(char)i;
   if ((char)i=='<')
   {
      starttag=true;
      starttagsing=1;
      tagentity="";
   }
   else
   {
      if ((char)i=='/' &&starttag==true )
      {
         if(starttagsing==1)
         {
         starttag=false;
         endtag=true;
         waitforgroup=false;
         if (!groupname.trim().equals(""))
         {
            String contactperson="";
            contactperson=contactpersonname.substring(1,contactpersonname.length()-1);
            group.addToTree(groupname,jid,contactperson);
         }
         jid="";
         contactpersonname="";
         groupname="";
            if(!messagebody.trim().equals(""))
         {
         if (chattype.equals("chat"))
         {
            group.insertTextInToTextPane(messagebody,sendername);
         }
         else
         {
            String username=logingui.getUserName();
            String name[]=sendername.split("/");
            System.out.println("sender name "+name[1]);
            System.out.println("receiver name "+username);
            if(name[1].equals(username))
            {
               groupuser.writeintotextfield("\n"+name[1]+":"+messagebody, nameattribut);
            }
            else
            {
               groupuser.writeintotextfield("\n"+name[1]+":"+messagebody, sendernameattrib)
            }
         }
      }
      startwritting=false;
      messagebody="";
      vactorindex=vactorindex-1;
      if (vactorindex>=0)
      tagvactor.removeElementAt(vactorindex);
   }
   if (starttag)
   {
      stagentity=tagentity+(char)i;
   }
}
else
{
   starttagsing=0;
   if((char)i=='>')
   {
   if (starttag)
   {
   sendername(tagentity);
   errorstring=checkForError(tagentity);
   if((tagentity.indexOf("type='error'")>1) &&
   (tagentity.indexOf("id='1301'")>1))
   {
      String username=logingui.getUserName();
      String password=logingui.getPassword();
      String servarn=logingui.getServarName();
      String registrationstring="<iq type=\"set\" to=\""+username+"@"+servarn+"\"
      id=\"1001\">";
      registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
      registrationstring=registrationstring+"<username>"+username+"</username>";
      registrationstring=registrationstring+"<password>"+password+"</password>";
      registrationstring=registrationstring+"<first>"+firstname+"</first>";
      registrationstring=registrationstring+"<last>"+lastname+"</last>";
      registrationstring=registrationstring+"<email>"+emailid+"</email>";
      registrationstring=registrationstring+" </query> ";
      registrationstring=registrationstring+"</iq>";
      String authentication="<iq type=\"set\" id=\"1301\">";
      authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
      authentication=authentication+" <username>"+username+"</username>";
      authentication=authentication+" <password>"+password+"</password>";
```

```
            authentication=authentication+" <resource>"+resource+"</resource> ";
            authentication=authentication+" </query> ";
            authentication=authentication+"</iq>";
            sendXMLToJabber(authentication);
            sendXMLToJabber(registrationstring);
         }
         if (errorstring.equals("unauthorized"))
         {
            sendRegistration();
            waitforreg=true;
            waitforauth=false;
         }
         if (tagentity.startsWith("message"))
         {
            if (tagentity.indexOf("type='groupchat'")>0)
         {
            chattype="groupchat";
         }
         else
         {
            chattype="chat";
         }
      }
      if((tagentity.indexOf("type='result'")>1)&&(waitforreg)&&
      (!errorstring.equals("user exist")))
      {
         sendAuthorized();
         waitforauth=true;
         waitforreg=false;
      }
      if((tagentity.indexOf("type='result'")>1)&&(waitforauth))
      {
         waitforauth=false;
      }
      if(tagentity.startsWith("presence"))
      {
         if(tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
         {
         nickname=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-1);
         String loginuser[]=nickname.split("/");
         if (tagentity.indexOf("type='unavailable'")>0||tagentity.indexOf
         ("type=\"unavailable\"")>0)
         {
         }
         else
         {
            Enumeration e=group.hasuser.keys();
            while(e.hasMoreElements())
            {
               String haselm=(String)e.nextElement();
               String[] userid=nickname.split("/");
               Object nname=group.hasuser.get(haselm);
               String nnamestr=nname.toString();
               if(userid[1].equals(loginuser[1]))
               {
                  String loginusertoadd=loginuser[1].substring(0,loginuser[1].length()-1);
                  groupuser.addElement(loginusertoadd);
                  break;
               }
               if(userid[1].equals(nnamestr+"'"))
               {
                  groupuser.addElement(nnamestr);
               }
            }
         }
      }
   }
   if(tagentity.indexOf("item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'")!=-1)
   {
      String user=group.getUserName();
      String removeuser;
      removeuser="<iq type='set' id='1'>";
      removeuser=removeuser+"<query xmlns='jabber:iq:roster'>";
      removeuser=removeuser+"<item jid='"+user+"@localhost/Home' name='"+user+"'
      subscription='remove'/>";
      removeuser=removeuser+"</query></iq>";
      sendXMLToJabber(removeuser);
      String removestring;
      JOptionPane.showMessageDialog(group,"No user exists with this user
      id.",errororwarning,JOptionPane.PLAIN_MESSAGE);
      nouserexists=true;
      return;
   }
   if (rosterflag)
   {
   if (tagentity.indexOf("item")!=-1 &&
   (!nouserexists)&&(tagentity.indexOf("affiliation")==-1))
   {
```

```
            contactpersonname=tagentity.substring(tagentity.indexOf("name")+5,
            tagentity.indexOf("jid")).trim();
            jid=tagentity.substring(tagentity.indexOf("jid")+4,tagentity.length());
            waitforgroup=true;
         }
         if (tagentity.indexOf("query")!=-1)
         {
            rosterflag=false;
         }
      }
      if ((tagentity.indexOf("type='result'")>1)&&(!waitforauth)&&(!waitforreg))
      {
         waitForResult=true;
      }
      else
      {
         if (tagentity.indexOf("xmlns='jabber:iq:roster'")>1&&waitForResult)
         {
            rosterflag=true;
            waitForResult=false;
         }
         waitForResult=false;
      }
      if (errorstring.equals("user exist")&&waitforreg)
      {
         JOptionPane.showMessageDialog(null, "A user with this user ID is already exists.Please enter o
         user ID.", errororwarning, JOptionPane.PLAIN_MESSAGE);
      }
      if (!errorstring.equals(""))
      {
         if (errortype=="major")
         {
            errororwarning="Error";
            JOptionPane.showMessageDialog(null, errorstring,errororwarning, JOptionPane.PLAIN_MESSAGE);
         }
         else
         {
            errororwarning="Warning";
            JOptionPane.showMessageDialog(null, errorstring,errororwarning, JOptionPane.PLAIN_MESSAGE);
         }
      }
      startwritting=true;
      tagvactor.insertElementAt(tagentity, vactorindex);
      vactorindex=vactorindex+1;
      starttag=false;
   }
   }
   else
   {
      if (waitforgroup && tagentity.trim().equals("group"))
      {
         groupname=groupname+(char)i;
      }
      if(startwritting==true && tagentity.trim().equals("body"))
      {
         messagebody=messagebody+(char)i;
      }
      if (starttag)
      {
         tagentity=tagentity+(char)i;
      }
   }
   }
   }
   }
   }
   catch(IOException ie)
   {
   }
   }
   isConnected=false;
   }
   /*
   Function to set the object of ChatWindow
   Parameter:chat- Object of ChatWindow class
   Return Value: N/A
   */
   public void setObjChat(ChatWindow chat)
   {
   this.chat=chat;
   }
   public void sendAuthorized()
   {
      String authentication;
      authentication="<iq type=\"set\" id=\"1\">";
      authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
```

```
   authentication=authentication+"<username>"+username+"</username>";
   authentication=authentication+"<password>"+password+"</password>";
   authentication=authentication+"<resource>"+resource+"</resource>";
   authentication=authentication+"</query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
}
/*
Used to send the request to jabber server to get the user list
Parameter: N/A
Return Value: N/A
*/
public void sendRosterRequest()
{
   group.clearTree();
   String rosterstring="";
   rosterstring="<iq type=\"get\" id=\"1\">";
   rosterstring=rosterstring+"<query xmlns=\"jabber:iq:roster\"/></iq>";
   sendXMLToJabber(rosterstring);
}
/*
Used to create a group by sending a message to jabber server
Parameter:group- Object of string class
Return Value: N/A
*/
public void createGroup(String group)
{
   sendXMLToJabber(group);
}
public void sendRegistration()
{
   String registrationstring;
   registrationstring="<iq type=\"set\" to=\""+username+".localhost\" id=\"1\">";
   registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
   registrationstring=registrationstring+"<username>"+username+"</username>";
   registrationstring=registrationstring+"<password>"+password+"</password></
   query></iq>";
}
/*
Used to send the xml messages to jabber server
Parameter:outputmessage- Object of string class
Return Value: N/A
*/
public static void sendXMLToJabber(String outputmessage)
{
   out.println(outputmessage);
   out.flush();
}
}
```

Download this listing.

In the above code, the constructor of the SocketConnection class takes the integer string, the GroupTree class object of the GroupList class, and the GroupLoginGUI class as input parameters. The SocketConnection class allows end users to invoke the methods of the GroupList class, GroupTree class, and GroupLoginGUI class. The host string retrieves the name of the Jabber server and the port integer retrieves the port number of the Jabber server.

The methods defined in Listing 4-7 are:

- openSocket(): Opens a client socket by using the IP address and port number of the Jabber server.
- Socketope(): Instantiates the Socket class to invoke methods of the Socket class.
- getErrorString(): Retrieves the error string sent by the Jabber server.
- Sendername(): Retrieves the sender's name from the input message received from the Jabber server.
- checkForError(): Checks whether or not the input string contains an error message.
- runinput(): Reads the response sent by the Jabber server.

Creating a Group Tree

The GroupTree.java file creates and maintains the tree structure of the available groups and end users present in each group. Listing 4-8 shows the contents of the GroupTree.java file:

Listing 4-8: The GroupTree.java File

```
/*Import required awt classes*/
import java.awt.GridLayout;
import java.awt.Toolkit;
/*Import required swing classes*/
import javax.swing.JPanel;
import javax.swing.JScrollPane;
/*Import required Tree classes*/
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.MutableTreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;
/*Import required Tree event classes*/
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeSelectionEvent;
/*Import required util package classes*/
import java.util.*;
/*
class GroupTree - This class is used to create a dynamic tree
Constructor:
GroupTree-This constructor creates a tree.
Methods:
clearTree - Remove all nodes except the root node.
getNodeName - Used to get the  node
addObject-Use to add the node to the tree
*/
public class GroupTree extends JPanel implements TreeSelectionListener
{
/*Declare object of DefaultMutableTreeNode class.*/
protected DefaultMutableTreeNode rootNode;
protected DefaultMutableTreeNode parentNode = null;
protected DefaultTreeModel treeModel;
/*Declare object of JTree class.*/
protected JTree tree;
private Toolkit toolkit = Toolkit.getDefaultToolkit();
/*Declare object of SocketConnection class.*/
SocketConnection clientsocket;
/*Declare object of GroupLoginGUI class.*/
GroupLoginGUI groupLogin;
/*Declare object of GroupList class.*/
GroupList group;
/*Declare object of ChatWindow class.*/
ChatWindow ch;
/*Declare the string variable*/
String user="";
String servername="";
String resource="Home";
/*Declare object of HashMap class.*/
HashMap mapObj=new HashMap();
/*Declare object of Vector class.*/
Vector vec=new Vector();
/*Constructor for class*/
public GroupTree(SocketConnection clientsocket,GroupLoginGUI groupLogin,GroupList group)
{
   /*Set the window title*/
   super(new GridLayout(1,0));
   this. clientsocket=clientsocket;
   this.groupLogin=groupLogin;
   this.group=group;
   /*Create the object of DefaultMutableTreeNode to create root node*/
   rootNode = new DefaultMutableTreeNode("Group");
   /*Add the rootnode into treeModel*/
   treeModel = new DefaultTreeModel(rootNode);
   /*Add the treeModelListener to treemodel*/
   treeModel.addTreeModelListener(new MyTreeModelListener());
   /*Add the treeModel into the tree*/
   tree = new JTree(treeModel);
   /*Set the tree in editable mode*/
   tree.setEditable(true);
   tree.getSelectionModel().setSelectionMode
   (TreeSelectionModel.SINGLE_TREE_SELECTION);
   tree.setShowsRootHandles(true);
   tree.addTreeSelectionListener(this);
```

```
   /*Add the tree to the scrollpane*/
   JScrollPane scrollPane = new JScrollPane(tree);
   /*Add the scrollpane to container*/
   add(scrollPane);
}
/*
Remove all nodes except the root node.
Parameter: N/A
Return Value: N/A
*/
public void clearTree()
{
   rootNode.removeAllChildren();
   treeModel.reload();
}
/*
Function to get the username
Parameter: N/A
Return Value: String
*/
public String getNodeName()
{
   String user="";
   TreePath currentSelection = tree.getSelectionPath();
   if (currentSelection != null)
      {
         DefaultMutableTreeNode currentNode = (DefaultMutableTreeNode)
         (currentSelection.getLastPathComponent());
         MutableTreeNode parent = (MutableTreeNode)(currentNode.getParent());
         if (parent != null)
         {
            user=currentNode.toString();
         }
      }
      return user;
   }
/*
Used to add node to the tree
Parameter: child-Object of Object class
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(Object child)
{
DefaultMutableTreeNode parentNode = null;
TreePath parentPath = tree.getSelectionPath();
if (parentPath == null)
{
   parentNode = rootNode;
}
else
{
   parentNode = (DefaultMutableTreeNode)
   (parentPath.getLastPathComponent());
}
return addObject(parentNode, child, true);
}
/*
Used to add node to the tree
Parameter: parent-Object of DefaultMutableTreeNode class
child-Object of Object class
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent,Object child)
{
   return addObject(parent, child, true);
}
/*
Used to add node to the tree
Parameter: parent-Object of DefaultMutableTreeNode class
child-Object of Object class
shouldBeVisible-boolean
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent, Object child,
boolean shouldBeVisible)
{
   DefaultMutableTreeNode childNode =new DefaultMutableTreeNode(child);
   if (parent == null)
{
   parent = rootNode;
}
treeModel.insertNodeInto(childNode, parent,parent.getChildCount());
if(shouldBeVisible)
{
```

```
   tree.scrollPathToVisible(new TreePath(childNode.getPath()));
}
return childNode;
}
/* Inner class for event listener */
class MyTreeModelListener implements TreeModelListener
{
public void treeNodesChanged(TreeModelEvent e)
{
   DefaultMutableTreeNode node;
   node = (DefaultMutableTreeNode)(e.getTreePath().getLastPathComponent());
   try
   {
      int index = e.getChildIndices()[0];
      node = (DefaultMutableTreeNode)(node.getChildAt(index));
   }
   catch (NullPointerException exc)
   {
   }
}
public void treeNodesInserted(TreeModelEvent e)
{
}
public void treeNodesRemoved(TreeModelEvent e)
{
}
public void treeStructureChanged(TreeModelEvent e)
{
}
}
/*
Method to describe the event
Parameter: e-Object of TreeSelectionEvent class
Return Value:N/A
*/
public void valueChanged(TreeSelectionEvent e)
{
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)tree.getLastSelectedPathComponent();
   if (node!=null)
   {
      TreePath currentSelection = tree.getAnchorSelectionPath();
      TreeNode parent =node.getParent();
      TreeNode parentnode =node.getRoot();
      Enumeration enu=parentnode.children();
      boolean b=node.isLeaf();
      TreeNode root =node.getParent();
      if((b) &&!(parent.toString().equals("Group")))
      {
         String receiver=node.toString();
         user=groupLogin.getUserName();
         servername=groupLogin.getServarName();
         ch=new ChatWindow(user,receiver,servername,clientsocket);
         mapObj.put(receiver,ch);
      }
      else
      {
         if((root!=null) &&(parent.toString().equals("Group")))
         {
            DefaultMutableTreeNode groupuser = (DefaultMutableTreeNode)
            tree.getLastSelectedPathComponent();
            user=groupLogin.getUserName();
            servername=groupLogin.getServarName();
            String roomname=node.toString();
            GroupChat gp=new GroupChat(user,servername,roomname,clientsocket);
         }
      }
   }
}
/*
Used to set the ChatWindow Object
Parameter: ch-Object of ChatWindow class
Return Value:N/A
*/
public void setChat(ChatWindow ch)
{
   this.ch=ch;
}
/*
Used to get the ChatWindow Object
Parameter: N/A
Return Value:N/A
*/
public ChatWindow getChat()
{
   return this.ch;
}
}
```

Download this listing.

In the above code, the constructor of the GroupTree class takes an object of the GroupList class, GroupLoginGUI class, and SocketConnection class as input parameters. This allows end users to invoke the methods of the GroupList class, GroupLoginGUI class, and SocketConnection class.

The methods defined in Listing 4-8 are:

- clearTree(): Refreshes the tree structure containing group and end user names for each group.
- getNodeName(): Retrieves the name of the group selected by an end user.
- addObject(): Adds end users or groups to the tree structure.
- setChat(): Sets the object of the ChatWindow class.

The GroupTree.java file creates and maintains the tree structure of the available groups and end users in each group, as shown in Figure 4-9:

Figure 4-9: Creating a Group Tree

PREVIOUS NEXT

Unit Testing

To test the Group Chatting application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of Java 2 SDK (J2SDK) by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the GroupLoginGUI.java, GroupList.java, AddUser.java, GroupChat.java, EditGroup.java, GroupTree.java, ChatWindow.java, and SocketConnection.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the javac command, as shown:

   ```
   javac *.java
   ```

5. To run the Group Chatting application, specify the following command at the command prompt:

   ```
   java GroupLoginGUI
   ```

6. The Login window of the Group Chatting application appears. Specify the login information, as shown in Figure 4-10:

Figure 4-10: Specifying Login Information

7. Click the Submit button to send the login information to the Jabber Server.

8. The user interface of the Group Chatting application appears. Select File->Add User to add other end users to the selected group.

9. Enter the name of another end user and the group name to add the specified user to a group, as shown in Figure 4-11:

Figure 4-11: Adding an End User to a Group

10. Click the Submit button to add an end user to a group.

11. Select File->Edit Group to rename a group.

12. Enter the name of the old group and the new name to rename an existing group, as shown in Figure 4-12:

Figure 4-12: Entering the Old and New Group Names

13. Click the Submit button to rename the group. The group with the new name appears, as shown in Figure 4-13:

Figure 4-13: Renamed Group

14. Select an end user name from the tree structure of the available groups to initiate a private chat.
15. Type the message that you want to send and click the Send button to send the text message to the selected end user, as shown in Figure 4-14:

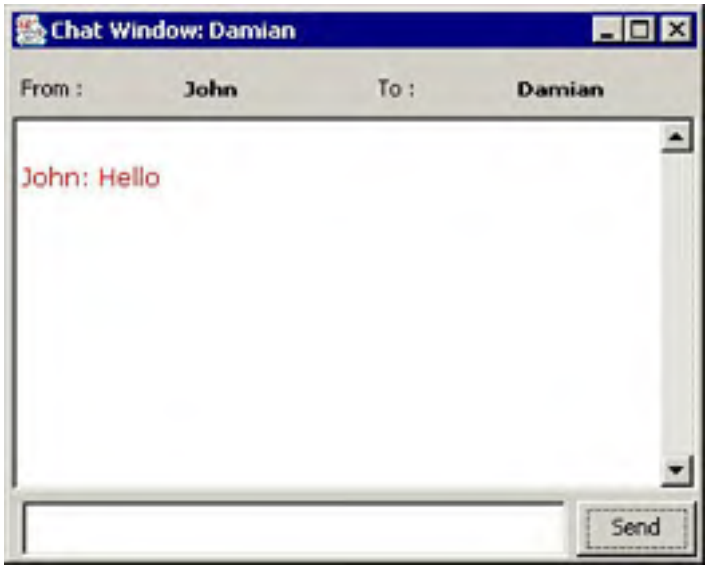


Figure 4-14: Sending a Private Message

16. Select a specific group from the tree structure of the available groups to participate in a group chat.
17. Type the required message and click the Send button to send the text message to all end users logged on to the selected group, as shown in Figure 4-15:

Figure 4-15: Sending a Group Message

PREVIOUS NEXT

Chapter 5: Creating an Instant Technical Support Application

Download CD Content

The Jabber protocol allows you to develop applications for instant messaging between end users through the Jabber server. This chapter describes how to develop an Instant Technical Support application that allows end users to send messages to a technical support executive for online help.

Architecture of the Instant Technical Support Application

The Instant Technical Support application provides a user interface to send and receive messages by using a Jabber server. The Instant Technical Support application allows end users to change font, style, and size of the text.

The Instant Technical Support application uses the following files:

- welcome.html: Creates the home page for the Instant Technical Support application.
- signuppage.html: Creates a Web interface that allows end users to sign up for a new account.
- loginpage.html: Creates a Web interface that allows end users to login.
- Chathtml.html: Sends the information specified by end users to the ChatMainApplet.java file.
- ChatMainApplet.java: Creates the user interface for the Instant Technical Support application that allows end users to establish a connection with the Jabber server. The ChatMainApplet.java file allows end users to communicate with the technical support executive connected to the Jabber server.

Figure 5-1 shows the architecture of the Instant Technical Support application:

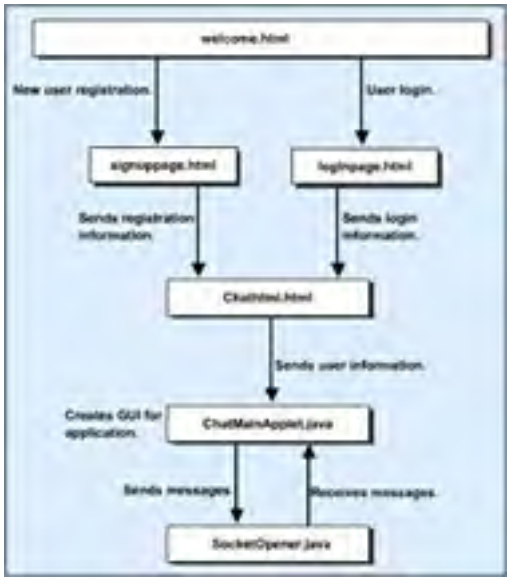


Figure 5-1: Architecture of the Instant Technical Support Application

If an end user clicks the Sign up button on the home page, the welcome.html file calls the signuppage.html file, which allows the end users to sign up for a new account. The signuppage.html file provides an interface with various labels, text boxes, and two buttons, OK and Cancel.

If an end user clicks the Click here button on the home page, the welcome.html page calls the loginpage.html file, which allows existing end users to login. The loginpage.html file provides an interface with various labels, two text boxes, and the Submit button.

Creating the Home Page

The welcome.html file creates the home page for the Instant Technical Support application. Listing 5-1 shows the contents of the welcome.html file:

Listing 5-1: The welcome.html File

```
<HTML>
<HEAD>
<TITLE>
Welcome to Instant Technical Support
</TITLE>
<SCRIPT language="javascript">
function openwindow(pagename,windowname)
{
   window.open(pagename, windowname, 'width=500, height=300, left=300, top=100, screenX=500,
   screenY=100');
}
</SCRIPT>
</HEAD>
<BODY>
<TABLE WIDTH="100%" CELLSPACING=0 CELLPADDING=0 BORDER=0>
<TR>
<TD>
<TABLE WIDTH="100%" CELLSPACING=0 CELLPADDING=0 BORDER=0>
<TR>
<TD ALIGN="CENTER">
<FONT SIZE="5" FACE="verdana" COLOR="">
<U>
Welcome to Instant Technical Support
</U>
</FONT>
</TD>
</TR>
</TABLE>
</TD>
</TR>
<TR>
<TD>
 
</TD>
</TR>
<TR>
<TD>
 
</TD>
</TR>
<TR>
<TD>
<TABLE WIDTH="100%" CELLSPACING=3 CELLPADDING=0 BORDER=0 ALIGN="CENTER" >
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="60%" ALIGN="CENTER" colspan=2>
<FONT SIZE="4" COLOR="">
</FONT>
</TD>
<TD WIDTH="40%" ALIGN="LEFT">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
```

```
 
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="LEFT">
Existing User
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
<INPUT TYPE="BUTTON" NAME="existinguser" style="width:100" VALUE="Click here"
onclick="javascript:openwindow('loginpage.html', 'LoginPage');">
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="LEFT">
New User
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
<INPUT TYPE="BUTTON" NAME="newuser" style="width:100" VALUE="Sign up"
onclick="javascript:openwindow('signuppage.html', 'NewUser');">
</TD>
<TD WIDTH="40%">
 
</TD>
</TR>
<TR>
<TD WIDTH="20%">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="30%" ALIGN="CENTER">
 
</TD>
<TD WIDTH="20%">
 
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Download this listing.

The welcome.html file creates the home page for the Instant Technical Support application, as shown in Figure 5-2:

Figure 5-2: The Home Page

Click the Click here button to login as an existing user.

Click the Sign up button to sign up for a new account.

Creating the Signup Page

The signuppage.html file creates the sign up page that allows end users to create a new account. Listing 5-2 shows the contents of the signuppage.html file:

Listing 5-2: The signuppage.html File

```
<HTML>
<HEAD>
<TITLE>Signup Page</TITLE>
<SCRIPT language="javascript">
function openwindow(pagename,windowname)
{
   if(document.profileenteryform.username.value=="")
   {
      alert('User name is a required field.');
      return false;
   }
   if(document.profileenteryform.password.value=="")
   {
      alert('Password is a required field.');
      return false;
   }
   if(document.profileenteryform.cpassword.value=="")
   {
      alert('Confirm Password is a required field.');
      return false;
   }
   if(document.profileenteryform.password.value!=document.profileenteryform.password.value)
   {
      alert('Password and Confirm Password must be same.');
      return false;
   }
   if(document.profileenteryform.firstname.value=="")
   {
      alert('First name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.firstname.value))
      {
         alert('Please enter a valid value for first name.');
         return false;
      }
   }
   if(document.profileenteryform.lastname.value=="")
   {
      alert('Last Name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.lastname.value))
      {
         alert('Please enter a valid value for last name.');
         return false;
      }
   }
   if(document.profileenteryform.emailid.value=="")
   {
      alert('Email Id is a required field.');
      return false;
   }
   else
   {
   if(!isEmailAddr(document.profileenteryform.emailid.value))
   {
      alert('Please enter a valid email ID.');
      return false;
   }
}
function checkForInt(valuestring)
{
   var validcharstring=false;
   var intstring="0123456789!@#$%^&*()_-+="
   for(i=0;i<valuestring.length;i++)
   {
      if (intstring.indexOf(valuestring.charAt(i))!=-1)
      {
         return validcharstring;
      }
   }
   validcharstring=true;
```

```
   return validcharstring;
}
function isEmailAddr(email)
{
   var result = false;
   var theStr = new String(email);
   var index = theStr.indexOf("@");
   if (index > 0)
   {
      var pindex = theStr.indexOf(".",index);
      if ((pindex > index+1) && (theStr.length > pindex+1))
      result = true;
   }
   return result;
}
user_name=document.profileenteryform.username.value;
pwd=document.profileenteryform.password.value;
fname=document.profileenteryform.firstname.value;
lname=document.profileenteryform.lastname.value;
emailid=document.profileenteryform.emailid.value;
pagename=pagename+"?username="+user_name+"&password="+pwd+"&firstname="+fname+"&lastname="
+lname+"&emailid="+emailid;
window.open(pagename, windowname,' width=470, height=515, left=300, top=100,screenX=30, screenY=100'
this.close();
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="profileenteryform">
<TABLE WIDTH="100%">
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
<FONT SIZE="4" COLOR="">
Signup for New Account
</FONT>
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD>
<TABLE ALIGN="CENTER" WIDTH="100%" >
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="20%">
User Name
</TD>
<TD WIDTH="40%">
<INPUT TYPE="INPUT" NAME="username" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Password
</TD>
<TD WIDTH="30%">
<INPUT TYPE="password" NAME="password" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
```

```
 
</TD>
<TD WIDTH="30%">
Confirm Password
</TD>
<TD WIDTH="30%">
<INPUT TYPE="PASSWORD" NAME="cpassword" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
First Name
</TD>
<TD WIDTH="30%">
<INPUT TYPE="INPUT" NAME="firstname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Last Name
</TD>
<TD WIDTH="30%">
<INPUT TYPE="INPUT" NAME="lastname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Email ID
</TD>
<TD WIDTH="30%">
<INPUT TYPE="INPUT" NAME="emailid" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
 
</TD>
<TD WIDTH="30%" colspan=2>
<INPUT TYPE="BUTTON" NAME="ok" style="width:30%" VALUE="OK"
onclick="javascript:openwindow('chathtml.html', 'chatpage');">
<INPUT TYPE="BUTTON" NAME="cancel"  style="width:30%" VALUE="Cancel"
onclick="javascript:window.close();">
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Download this listing.

The Signup.html page creates the signup page, as shown in Figure 5-3:

Figure 5-3: The Signup Page

◄ PREVIOUS NEXT ►

Creating the Login Page

The loginpage.html file creates the login page to help end users login as existing users. Listing 5-3 shows the contents of the loginpage.html file:

Listing 5-3: The loginpage.html File

```
<HTML>
<HEAD>
<TITLE>Login Page</TITLE>
<SCRIPT language="javascript">
function openwindow(pagename,windowname)
{
   if(document.loginform.username.value=="")
   {
      alert('User name is a required field.');
      return false;
   }
   if(document.loginform.username.value=="")
   {
      alert('Password is a required field.');
      return false;
   }
   user_name=document.loginform.username.value;
   pwd=document.loginform.password.value;
   pagename=pagename+"?username="+user_name+"&password="+pwd;
   window.open(pagename,windowname,'width=470,height=515,left=300,top=100,screenX=30,screenY=100');
   this.close();
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="loginform">
<TABLE WIDTH="100%" ALIGN="CENTER">
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%" colspan=2>
<FONT SIZE="4" COLOR="">
Enter your user name and password to login
</FONT>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
User Name
</TD>
<TD WIDTH="45%">
<INPUT TYPE="input" NAME="username" VALUE="" maxlength="100">
</TD>
<TD WIDTH="10%">
```

```
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
Password
</TD>
<TD WIDTH="45%">
<INPUT TYPE="password" NAME="password" VALUE="" maxlength="100">
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="45%">
 
</TD>
<TD WIDTH="45%">
<INPUT TYPE="BUTTON" NAME="okbutton" style="width:100" VALUE="OK"
onclick="javascript:openwindow('chathtml.html','chat');">
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Download this listing.

The loginpage.html page creates the login page, as shown in Figure 5-4:

Figure 5-4: The Login Page

Sending Login Information to the Jabber Server

The chathtml.html file calls the ChatMainApplet.java file to send the login information specified by an end user to the Jabber server. Listing 5-4 shows the contents of the chathtml.html file:

Listing 5-4: The chathtml.html File

```
<HTML>
<HEAD>
<TITLE>Instant Technical Support Application</TITLE>
<SCRIPT>
function PageQuery(q)
{
   if(q.length > 1)
   this.q = q.substring(1, q.length);
   else
   this.q = null;
   this.keyValuePairs = new Array();
   if(q)
   {
      for(var i=0; i <this.q.split("&").length; i++)
      {
      this.keyValuePairs[i] = this.q.split("&")[i];
      }
   }
   this.getKeyValuePairs = function()
   {
      return this.keyValuePairs;
   }
   this.getValue = function(s)
   {
      for(var j=0; j <this.keyValuePairs.length; j++)
         {
         if(this.keyValuePairs[j].split("=")[0] == s)
         return this.keyValuePairs[j].split("=")[1];
      }
      return false;
   }
   this.getParameters = function()
   {
      var a = new Array(this.getLength());
      for(var j=0; j <this.keyValuePairs.length; j++)
      {
         a[j] = this.keyValuePairs[j].split("=")[0];
      }
      return a;
   }
   this.getLength = function()
   {
      return this.keyValuePairs.length;
   }
}
function queryString(key)
{
   var page = new PageQuery(window.location.search);
   return page.getValue(key);
}
function appletparam(key)
{
if(queryString(key)=='false')
{
   pagename='loginpage.html';
   windowname='login';
   window.open(pagename, windowname,'width=400, height=200, left=300,top=100, screenX=500,
   screenY=100');
}
else
{
   return queryString(key);
}
}
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
function writeAppletTag()
{
   document.writeln('<applet code="ChatMainApplet" width="450" height="500">');
   document.writeln(buildParamTag('username', appletparam('username')));
   document.writeln(buildParamTag('password', appletparam('password')));
   document.writeln(buildParamTag('firstname', appletparam('firstname')));
   document.writeln(buildParamTag('lastname', appletparam('lastname')));
   document.writeln(buildParamTag('emailid', appletparam('emailid')));
   document.writeln('</APPLET>');
}
function buildParamTag(name, value)
```

```
{
   return '<PARAM NAME="' + name + '" VALUE="' + value + '">';
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
writeAppletTag();
</SCRIPT>
</BODY>
</HTML>
```

Download this listing.

In the above code, the writeAppletTag() function calls the ChatMainApplet.java file to create the user interface of the Instant Technical Support application.

◄ PREVIOUS NEXT ►

Creating the User Interface of the Instant Technical Support application

The ChatMainApplet.java file creates the user interface for the Instant Technical Support application that allows end users to exchange messages with the technical support executive through the Jabber server. Listing 5-5 shows the contents of the ChatMainApplet.java file:

Listing 5-5: The ChatMainApplet.java File

```
/*Imports required javax.swing package classes.*/
import javax.swing.*;
/*
Imports required javax.swing.event package classes
*/
import javax.swing.event.*;
/*Imports required javax.awt package classes.*/
import java.awt.*;
/*Imports required java.awt.event package classes.*/
import java.awt.event.*;
/*Imports required java.io package classes.*/
import java.io.*;
/*Imports required java.net classes.*/
import java.net.*;
/*Imports required io java.util classes.*/
import java.util.*;
/*Imports required javax.swing.text package classes.*/
import javax.swing.text.*;
/*
class ChatMainApplet - This class is the main class of the application. This class initializes
the interface and loads all components like the button, textfields before displaying the result.
Methods:
sendXMLToJabber - Send message to jabber server.
sendmessage - Convert message into XML format.
checkForError - Check for error message.
readDate - Read next message from socket.
*/
public class ChatMainApplet extends JApplet implements ItemListener,Runnable,ActionListener,KeyListe
{
/*Declare object of JTextField class.*/
JTextField inputtext;
/*Declare object of JTextPane class.*/
JTextPane output;
/*Declare object of JScrollPane class.*/
JScrollPane outputscrollpane;
/*Declare objects of JComboBox class.*/
private JComboBox fontChoice;
private JComboBox styleChoice;
private JComboBox sizeChoice;
/*Declare objects of JButton class.*/
Socket clientsocket;
/*Declare objects of JButton class.*/
Button sendbutton;
Button closebutton;
/*Declare object of PrintWriter class.*/
PrintWriter out = null;
BufferedReader in = null;
Thread inputmessagethread;
/*Declare object of Document class.*/
Document contentmodel;
/*Declares object of JPanel class.*/
JPanel panel;
JPanel bottompanel;
/*
Declare object of MutableAttributeSet class.
*/
MutableAttributeSet sendernameattrib,recervernameatrib,normaltextattrib;
/*Declare object of String class.*/
String ipaddress;
String recerverid;
String username;
String password;
String resource;
String firstname="";
String lastname="";
String emailid="";
String errortype="";
boolean isError=false;
private  boolean isConnected=false;
JLabel fontlabel;
JLabel fromlabel;
int portno;
```

```
int wait;
public void init()
{
/*
Initializes and sets the Look and Feel of the application to Windows Look and Feel.
*/
try
{
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
   System.out.println("Unable to set WLF"+e);
}
/*
Declares and initializes object of Container class.
*/
Container contenpane=getContentPane();
/*
Sets container layout as BorderLayout.
*/
contenpane.setLayout(new BorderLayout());
/*
Initializes object of JTextPane class.
*/
output = new JTextPane();
/* Sets background color to white.*/
output.setBackground(Color.white);
/*
Initializes object of the JScrollPane class.
*/
outputscrollpane=new JScrollPane(output,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
output.setEditable(false);
/*
Add outputscrollpane to contentpanel.
*/
contenpane.add(outputscrollpane, "Center");
/*
Initializes object of the JPanel class.
*/
panel = new JPanel();
/*
Initializes object of the JLabel class.
*/
fontlabel=new JLabel(" Select Font ");
/*
Add fontlabel to the panel.
*/
panel.add(fontlabel);
/*
Declares and initializes a String array containing the font list.
*/
String[] fontNames = getToolkit().getFontList();
String defaultFontName = getFont().getName();
/*
Initializes object of the JComboBox class.
*/
fontChoice = new JComboBox();
fontChoice.addItem( "" );
fontChoice.setSelectedIndex( 0 );
for ( int i = 0; i < fontNames.length; ++i )
{
   fontChoice.addItem( fontNames[i] );
   if ( fontNames[i].equals( defaultFontName ) )
   fontChoice.setSelectedIndex( i + 1 );
}
panel.add( fontChoice );
/*
Initializes object of the JComboBox class.
*/
styleChoice = new JComboBox();
/*
Add items to the styleChoice combobox.
*/
styleChoice.addItem( "Plain" );
styleChoice.addItem( "Bold" );
styleChoice.addItem( "Italic" );
styleChoice.addItem( "Bold Italic" );
styleChoice.setSelectedIndex(0);
/*Add object of JLabel to the panel.*/
panel.add(new JLabel("Select Style"));
/*
Add object of JComboBox to the panel.
*/
panel.add(styleChoice);
/*Add object of JLabel to the panel.*/
```

```
panel.add(new JLabel("Select size"));
/*
Initializes object of the JComboBox class.
*/
sizeChoice = new JComboBox();
/*Add items to the sizeChoice combobox.*/
sizeChoice.addItem( "6" );
sizeChoice.addItem( "8" );
sizeChoice.addItem( "10" );
sizeChoice.addItem( "12" );
sizeChoice.addItem( "15" );
sizeChoice.addItem( "20" );
sizeChoice.addItem( "25" );
sizeChoice.setSelectedIndex( 3 );
/*
Add object of JComboBox to the panel.
*/
panel.add(sizeChoice);
/*Add action listener to sizeChoice.*/
sizeChoice.addActionListener(this);
/*Add action command to sizeChoice.*/
sizeChoice.setActionCommand("size");
/*Add action listener to styleChoice.*/
styleChoice.addActionListener(this);
/*Add action command to styleChoice.*/
styleChoice.setActionCommand("style");
/*Add action listener to fontChoice.*/
fontChoice.addActionListener(this);
/*Add action command to fontChoice.*/
fontChoice.setActionCommand("font");
/*
Initializes object of JTextField class.
*/
inputtext = new JTextField();
/*Add key listener to the inputtext.*/
inputtext.addKeyListener(this);
/*Set button size.*/
inputtext.setPreferredSize(new Dimension(20, 30));
/*
Initializes object of the JPanel class.
*/
bottompanel=new JPanel();
/*Set panel size.*/
bottompanel.setPreferredSize(new Dimension(20, 120));
/*Initialize object of GridLayout.*/
GridLayout gridlayout=new GridLayout(2, 1, 0, 0);
/*Sets  BorderLayout layout to container.*/
bottompanel.setLayout(gridlayout);
/*Set panel size.*/
bottompanel.setPreferredSize(new Dimension(20, 120));
bottompanel.add(panel);
/*
Declare and initializes object of BorderLayout.
*/
BorderLayout bottomgridlayout=new BorderLayout();
/*
Declare and initializes object of JPanel.
*/
JPanel subbottompanel=new JPanel();
/*Set background color to white.*/
subbottompanel.setBackground(Color.WHITE);
/*Sets Layout as BorderLayout.*/
subbottompanel.setLayout(bottomgridlayout);
/*Sets panel size.*/
subbottompanel.setPreferredSize(new Dimension(20, 120));
/*
Add object of JLabel class to the panel.
*/
subbottompanel.add(new JLabel(" "), BorderLayout.NORTH);
/*
Add object of JTextField class to the panel.
*/
subbottompanel.add(inputtext,BorderLayout.CENTER);
/*
Declares and initializes object of GridLayout.
*/
JPanel sendbuttonpanel=new JPanel(new FlowLayout(1, 1, 3));
/*
Initializes object of Button class.
*/
sendbutton=new Button("Send");
/*Add action listener to button.*/
sendbutton.addActionListener(this);
/*Add action command to sendbutton.*/
```

```
sendbutton.setActionCommand("send");
sendbutton.setEnabled(true);
/* Sets button size.*/
sendbutton.setSize(new Dimension(10, 10));
/*
Add object of Button class to the JPanel.
*/
sendbuttonpanel.add(sendbutton);
/*Set background color to white.*/
sendbuttonpanel.setBackground(Color.WHITE);
/*
Add object of JPanel class to the JPanel.
*/
subbottompanel.add(sendbuttonpanel, BorderLayout.EAST);
/*
Add object of JLabel class to the JPanel.
*/
subbottompanel.add(new JLabel(" "), BorderLayout.SOUTH);
/*
Add object of JPanel class to the JPanel.
*/
bottompanel.add(subbottompanel);
/*
Add object of JPanel class to the ContentPane.
*/
contenpane.add(bottompanel,"South");
ipaddress="localhost";
portno=5222;
wait=5000;
resource="Home";
/*Sets jabber id.*/
recerverid="tcuser1@localhost";
username=getParameter("username");
password=getParameter("password");
firstname=getParameter("firstname");
lastname=getParameter("lastname");
emailid=getParameter("emailid");
/*
Initializes object of the SimpleAttributeSet class.
*/
recervernameatrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(recervernameatrib, "Verdana");
StyleConstants.setForeground(recervernameatrib, Color.red);
/*
Initializes object of SimpleAttributeSet class.
*/
normaltextattrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(normaltextattrib, "Verdana");
StyleConstants.setForeground(normaltextattrib, Color.black);
contentmodel=output.getDocument();
/*
Initializes object of SimpleAttributeSet class.
*/
sendernameattrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(sendernameattrib, "Verdana");
StyleConstants.setForeground(sendernameattrib, Color.blue);
String labelfromstring=" From: "+username;
String  labeltostring="To: Technical Support";
/*Initializes object of JLabel class.*/
fromlabel=new JLabel(labelfromstring,JLabel.LEFT);
fromlabel.setFont(new Font("verdana", 0, 12));
/*
Declares and initializes object of JLabel class.
*/
JLabel tolabel=new JLabel(labeltostring);
tolabel.setFont(new Font("verdana", 0, 12));
/*
Declares and initializes object of JPanel class.
*/
JPanel toppanel=new JPanel(new GridLayout(1, 2, 0, 10));
/*
Add object of JLabel class to the JPanel.
*/
toppanel.add(fromlabel);
/*
Add object of JLabel class to the JPanel.
*/
toppanel.add(tolabel);
/*
Add object of JPanel class to the ContenPane.
*/
contenpane.add(toppanel, BorderLayout.NORTH);
Font newfont=getNewFont();
inputtext.setFont(newfont);
startSession();
}
public void actionPerformed(ActionEvent ae)
{
```

```
   String source=ae.getActionCommand();
   if (source=="exit")
   {
      isConnected=false;
   }
   if (source=="send")
   {
      sendmessage();
   }
   if (source=="font"||source=="style"||source=="size")
   {
      inputtext.setFont(getNewFont());
   }
}
/*
sendmessage - This method is used to insert message into textpane and calls sendXMLToJabber function
Parameters: NA.
Return Value: NA.
*/
public void sendmessage()
{
   String outputstring;
   outputstring="<message to=\""+recerverid+"\">";
   outputstring=outputstring+"<body>";
   outputstring=outputstring+ inputtext.getText();
   outputstring=outputstring+"</body>";
   outputstring=outputstring+"</message>";
   try
   {
      contentmodel.insertString(contentmodel.getLength(), username+":",recervernameatrib);
   }
   catch(BadLocationException ble)
   {
   }
   output.setCharacterAttributes(normaltextattrib,true);
   try
   {
   if (((String)fontChoice.getSelectedItem()).trim().equals(""))
   {
      StyleConstants.setFontFamily(recervernameatrib,"Verdana");
      StyleConstants.setForeground(recervernameatrib,Color.red);
   }
   StyleConstants.setFontFamily(normaltextattrib, (String)fontChoice.getSelectedItem());
   StyleConstants.setFontSize(normaltextattrib,
   Integer.parseInt((String)sizeChoice.getSelectedItem()));
   if (styleChoice.getSelectedItem().equals("Bold"))
   {
      StyleConstants.setBold(normaltextattrib,true);
   }
   else if(styleChoice.getSelectedItem().equals("Italic"))
   {
      StyleConstants.setItalic(normaltextattrib,true);
   }
   else if(styleChoice.getSelectedItem().equals("Bold Italic"))
   {
      StyleConstants.setBold(normaltextattrib,true);
      StyleConstants.setItalic(normaltextattrib,true);
   }
   contentmodel.insertString(contentmodel.getLength(),inputtext.getText(), normaltextattrib);
}
catch(BadLocationException ble)
{
}
try
{
   contentmodel.insertString(contentmodel.getLength(), "\n",normaltextattrib);
}
catch(BadLocationException ble)
{
}
sendXMLToJabber(outputstring);
inputtext.setText("");
}
/*
sendXMLToJabber - This method is used to send xml message to jabber server.
Parameters: outputmessage - objcet of String class.
Return Value: NA
*/
public void sendXMLToJabber(String outputmessage)
{
   String[] tokenizerstring=outputmessage.split("\n");
   for (int i=0;i<tokenizerstring.length;i++ )
   {
      try
      {
         out.println(tokenizerstring[i]);
         out.flush();
      }
```

```
      catch(Exception e)
      {
         System.out.println("error");
         return;
      }
      out.flush();
   }
}
/*
startSession - This method is used to send xml message to jabber server.
Parameters: NA
Return Value: NA
*/
public void startSession()
{
   openPort(ipaddress, portno, wait);
   if (clientsocket!=null)
   {
      isConnected=true;
   }
   try
   {
      /*
      Initializes object of PrintWriter class.
      */
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      /*
      Initializes object of BufferedReader class.
      */
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
      /* Initializes object of Thread class.*/
      inputmessagethread=new Thread(this);
      /* Starts a new thread */
      inputmessagethread.start();
      /* Declare objects of String class.*/
      String sessionStratString;
      String authentication;
      String registrationstring="";
      sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
      sessionStratString=sessionStratString+" <stream:stream";
      sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
      sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
      sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
      sendXMLToJabber(sessionStratString);
      if ((firstname.equals("false"))||(lastname.equals("false"))||(emailid.equals("false")))
      {
      }
      else
      {
         registrationstring="<iq type=\"set\" to=\""+username+".localhost\" id=\"1001\">";
         registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
         registrationstring=registrationstring+"<username>"+username+"</username>";
         registrationstring=registrationstring+"<password>"+password+"</password>";
         registrationstring=registrationstring+"</query> ";
         registrationstring=registrationstring+"</iq>";
         sendXMLToJabber(registrationstring);
      }
      authentication="<iq type=\"set\" id=\"1001\">";
      authentication=authentication+"<query xmlns=\"jabber:iq:auth\">";
      authentication=authentication+"<username>"+username+"</username>";
      authentication=authentication+"<password>"+password+"</password>";
      authentication=authentication+"<resource>"+resource+"</resource> ";
      authentication=authentication+"</query> ";
      authentication=authentication+"</iq>";
      sendXMLToJabber(authentication);
   }
   catch(IOException ie)
   {
   }
}
/*
run - This method is called when a new thread is started.
Parameters: NA
Return Value: NA
*/
public void run()
{
   int i=0;
   String errororwarning="";
   String errorstring="";
   String inputstring="";
   String messagebody="";
   String tagentity="";
   boolean starttag=false;
   boolean endtag=false;
   boolean startwritting=false;
   Vector tagvactor=new Vector();
   int vactorindex=0;
```

```
int starttagsing=0;
while (true)
{
try
{
   i=in.read();
   if (i==-1)
   {
      break;
   }
   else
   {
   inputstring=inputstring+(char)i;
   if ((char)i=='<')
   {
      starttag=true;
      starttagsing=1;
      tagentity="";
   }
   else
   {
   if ((char)i=='/' && starttag==true )
   {
      if (starttagsing==1)
      {
         starttag=false;
         endtag=true;
         if (!messagebody.trim().equals(""))
         {
            writemessage(messagebody+"\n",normaltextattrib);
         }
         startwritting=false;
         messagebody="";
         vactorindex=vactorindex-1;
         if (vactorindex>=0)
         tagvactor.removeElementAt(vactorindex);
         }
      }
      else
      {
         starttagsing=0;
         if((char)i=='>')
         {
         if (starttag)
         {
            sendername(tagentity);
            errorstring=checkForError(tagentity);
            if (!errorstring.equals(""))
            {
               if (errortype=="major")
               {
                  errororwarning="Error";
                  JOptionPane.showMessageDialog(null,"No user exixts for this
                  userid.", errororwarning,JOptionPane.PLAIN_MESSAGE);
                  sendbutton.setEnabled(false);
                  fromlabel.setText(" From: ");
                  stop();
                  destroy();
               }
               else
               {
                  errororwarning="Warning";
                  JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                  JOptionPane.PLAIN_MESSAGE);
                  sendbutton.setEnabled(false);
                  fromlabel.setText(" From: ");
                  stop();
                  destroy();
               }
            }
            else
            {
            }
            startwritting=true;
            tagvactor.insertElementAt(tagentity,vactorindex);
            vactorindex=vactorindex+1;
            starttag=false;
            }
         }
         else
         {
         if (startwritting==true&&tagentity.trim().equals("body"))
         {
            messagebody=messagebody+(char)i;
         }
         if (starttag)
         {
            tagentity=tagentity+(char)i;
```

```
            }
         }
      }
   }
}
}
catch(IOException ie)
{
}
}
isConnected=false;
try
{
   inputmessagethread.join();
}
catch(Exception e)
{
}
}
/*
checkForError - This method is used to check error messages send by jabber server.
Parameters: tagentity - Object of String class
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if (!isError)
   {
      if (!firstname.equals("false"))
      {
      if (tagentity.indexOf("error")>0)
      {
         isError=true;
         return "User name is already exists. Please enter other User name.";
      }
   }
   if ((tagentity.indexOf("code='401'")>1)||(tagentity.indexOf("code=\"401\"")>1))
   {
      isError=true;
      return "User name or Password is invalid.";
   }
   if (tagentity.indexOf("error")>0)
   {
      if ((tagentity.equals("error code='409'"))||(tagentity.equals("error code=\"409\"")))
      {
         isError=true;
         return "User name is already exists.Please enter other User name.";
      }
   }
}
return error;
}
/*
sendername - This method is used to retrieve sender name from message.
Parameters: tagentity - Object of String class
Return Value: NA
*/
public void sendername(String tagentity)
{
   String sendername="";
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("id=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("id="));
         writemessage("Technical support: ",sendernameattrib);
      }
   }
}
/*
writemessage - This method is used to retrieve sender name from message.
Parameters: messagebody - Object of String class, attribut - Object of MutableAttributeSet.
Return Value: NA
*/
public void writemessage(String messagebody, MutableAttributeSet attribut)
{
   try
   {
      contentmodel.insertString(contentmodel.getLength(), messagebody,attribut);
   }
   catch(BadLocationException ble)
   {
   }
}
/*
openPort - This method is used to open a socket with the Jabber server.
```

```
Parameters: ipaddress - Object of String class
Return Value: NA
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      /*
      Declare and initialize object of SocketOpener class.
      */
      SocketOpener sc=new SocketOpener(ipaddress,portno);
      clientsocket=sc.openSocket(ipaddress,portno,timeinsec);
   }
}
public void itemStateChanged(ItemEvent itemevent)
{
   String eventsource=(String)itemevent.getSource();
}
/*
getNewFont - This method is used to get the font name selected by the end user.
Parameters: ipaddress - Object of String class
Return Value: Font
*/
public Font getNewFont()
{
   int fontsize=8;
   int fontstyle=0;
   String fontname="";
   Font font=null;
   if (sizeChoice.getSelectedIndex()!=-1)
   {
      fontsize=Integer.parseInt((String)sizeChoice.getSelectedItem());
   }
   if (styleChoice.getSelectedIndex()!=-1)
   {
      fontstyle=styleChoice.getSelectedIndex();
   }
   if (fontChoice.getItemCount()>0)
   {
      if (fontChoice.getSelectedItem()!="")
      {
         fontname=(String)fontChoice.getSelectedItem();
      }
      else
      {
         fontname="serif";
      }
   }
   font=new Font(fontname,fontstyle,fontsize);
   return font;
}
public boolean action( Event event, Object what )
{
   if( event.target == fontChoice || event.target == styleChoice ||
    event.target == sizeChoice )
   {
      inputtext.setFont(getNewFont());
      return true;
   }
   if (event.target==sendbutton)
   {
   }
   return false;
}
public void destroy()
{
   try
   {
   clientsocket.close();
   }
   catch(Exception e)
   {
   }
}
/*
readData - This method is used to read next message from socket.
Parameters: NA
Return Value: String
*/
private String readData() throws IOException
{
   StringBuffer s = new StringBuffer();
```

```
   int ch;
   int a=0;
   while ((ch = in.read()) > 0)
   {
      s.append((char)ch);
      a=a+1;
      if (a>10)
      {
      return ch == 0 ? s.toString() : null;
      }
   }
   return ch == 0 ? s.toString() : null;
}
/*
keyTyped - This method is used to handle the key typed event from the text field
Parameters: e - Object of KeyEvent class
Return Value: String
*/
public void keyTyped(KeyEvent e)
{
}
/*
keyPressed - This method is used to handle the key pressed event from the text field.
Parameters: e - Object of KeyEvent class
Return Value: NA
*/
public void keyPressed(KeyEvent e)
{
}
/*
keyReleased - This method is used to handle the key released event from the text field.
Parameters: e - Object of KeyEvent class
Return Value: NA
*/
public void keyReleased(KeyEvent e)
{
   int keyCode = e.getKeyCode();
   if (keyCode==13||keyCode==10)
   {
      sendmessage();
   }
}
};
/*
class SocketOpener - This class opens a socket.
the interface and loads all components like the button, textfields before displaying the result.
Methods:
openSocket - open a socket.
getSocket - Returns a socket objects.
*/
class SocketOpener
{
   private String openerhost;
   private int openerport;
   private Socket socket;
   public SocketOpener(String host,int port)
   {
      socket=null;
      openerhost=host;
      openerport=port;
   }
   /*
   openSocket - This method is used to open a socket.
   Parameters: hostip - Object of String class.portnumber - int,timeinsec - int
   Return Value: Socket
   */
   public  Socket openSocket(String hostip,int portnumber,int timeinsec)
   {
   try
   {
      socket=new Socket(openerhost,openerport);
   }
   catch(IOException ie)
   {
   }
   return getSocket();
}
/*
getSocket - This method is used to get object of Socket class.
Parameters: NA
Return Value: Socket
*/
public Socket getSocket()
{
   return socket;
}
};
```

Download this listing.

In the above code, the constructor of the ChatMainApplet class calls the init() method to create the user interface for the Instant Technical Support application. The init() method calls the startSession() method of the ChatMainApplet class to initialize the chat session with the Jabber server.

The methods defined in Listing 5-5 are:

- init(): Creates the user interface for the Instant Technical Support application.
- `sendmessage()`: Shows the query specified by an end user in the text area provided by the Instant Technical Support application. This method calls the sendXMLToJabber() method.
- `sendXMLToJabber()`: Sends the specified query to the Jabber server.
- `startSession()`: Sends the extensible markup language (XML) message to start the session.
- `checkForError()`: Checks the error messages sent by the Jabber server.
- `sendername()`: Retrieves the sender's name from the message received by the Jabber server.
- `openPort()`: Opens a socket to send and receive messages with the Jabber server.
- `getNewFont()`: Retrieves the font name selected by the end user.
- `readData()`: Reads the response to the specified query from the Jabber server.

The ChatMainApplet.java file creates the user interface of the Instant Technical Support application, as shown in Figure 5-5:

Figure 5-5: User Interface of the Instant Technical Support Application

Unit Testing

To test the Instant Technical Support application:

1. Download and install the free implementation of the Jabber server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the ChatMainApplet.java, welcome.html, signuppage.html, loginpage.html, and chathtml.html files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

5. Create a virtual directory named TechnicalSupport in a Web server.

6. To run the Instant Technical Support application, specify the following uniform resource locator (URL) in the Web browser:

   ```
   http://localhost/TechnicalSupport/welcome.html
   ```

7. The home page of the Instant Technical Support application appears. Click the Sign up button to signup for a new account in the Instant Technical Support application. The Signup page appears.

8. Enter data in the fields of the Signup page, as shown in Figure 5-6:

Figure 5-6: The Signup Information

9. Click OK to create a new account. The user interface of the Instant Technical Support application appears.

10. Specify the query text in the bottom text area provided by the Instant Technical Support application, as shown in Figure 5-7:

Figure 5-7: Specifying the Query Text

11. Click the Send button. The Instant Technical Support application sends the query to the technical support executive.

12. The specified query appears on the user interface in the upper text area, as shown in Figure 5-8:

Figure 5-8: Sending the Query

13. The technical support executive answers the query and sends the response back to the end user, as shown in Figure 5-9:

Figure 5-9: Receiving the Response for the Specified Query

Chapter 6: Creating an Airline Reservation Application

Download CD Content

The Jabber protocol allows you to develop an application to communicate with the Jabber server for accessing an Extensible Markup Language (XML) file that contains data. This chapter describes how to develop the Airline Reservation application that allows end users to search for airline schedules and make airline reservations by specifying the departure and arrival dates of the journey.

Architecture of the Airline Reservation Application

The Airline Reservation application uses the following files:

- UserInformation.html: Allows end users to specify personal information and shows the user interface of the Airline Reservation application.
- Reservation.html: Opens the user interface of the Airline Reservation application by calling the ReservationStatus.java file.
- ReservationStatus.java: Allows end users to select the starting place, destination, departure date, and arrival date of the journey. This file allows end users to submit the information to the Jabber server through the SocketClass.java file.
- SocketClass.java: Opens a socket and establishes a connection with the Jabber server to send and receive messages.
- SocketConnector.java: Reads the airlines.xml file based on the search information specified by end users. The SocketConnector.java file then connects with the Jabber server to send the airline schedules and confirmation message, written in the airlines.xml file, to the Airline Reservation application.
- ReservationResultTable.java: Allows end users to select the required schedule and send schedule information to the Jabber server.
- XMLReader.java: Allows the Airline Reservation application and the Jabber server to read the airlines.xml file.
- airlines.xml: Contains the information related to airline reservation. The Airline Reservation application reads this XML file to show the required information to end users.

Figure 6-1 shows the architecture of the Airline Reservation application:

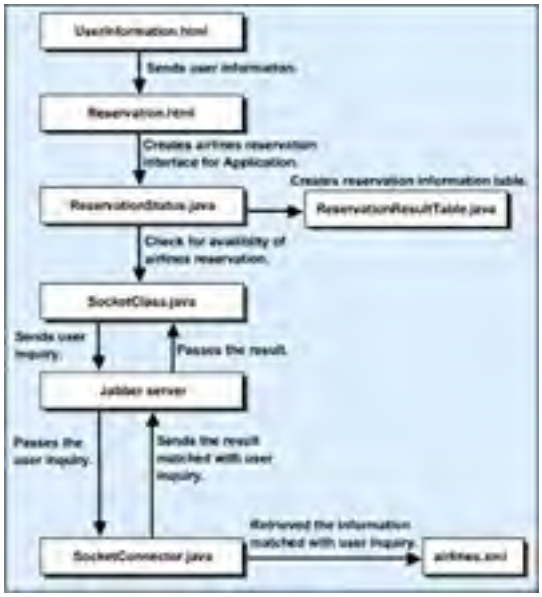


Figure 6-1: Architecture of the Airline Reservation Application

The UserInformation.html file creates the Web interface of Airlines Enquiry System that contains five labels, five text fields, and two command buttons. The UserInformation.html file allows end users to specify personal information.

If end users press the Cancel button on the Web interface, the Airline Enquiry System window closes. If an end user presses the OK button, the end user's information is submitted to the Reservation.html file. The Reservation.html file calls the ReservationStatus.java file.

The ReservationStatus.java file creates the user interface for the Airline Reservation application that contains combo boxes for starting place, destination, departure date, and arrival date. The user interface of the Airline Reservation application also contains two buttons, Reset and Search. When an end user clicks the Reset button, the Airline Reservation application refreshes the values stored in the combo boxes. When an end user clicks the Search button, the Airline Reservation application checks the existing airline schedule in the Jabber server based on the requirements specified by the end user and displays the schedule in the user interface.

The Jabber server passes an end user's query to the SocketConnector.java file. The SocketConnector.java file reads the airlines.xml file to search for the query specified by the end user. The airlines.xml file stores the database of existing airline schedules. If an end user's query matches any existing airline schedule, the SocketConnector.java file returns the information to the Jabber server. The Jabber server sends the information to the Airline Reservation application. The application shows the information received from the Jabber server in the Airline Reservation window by using the ReservationResultTable.java file.

End users can select any of the available airline schedules to reserve the required airline schedule. End users can click the Submit button to send the information to the Jabber server. The Jabber server receives the information and sends a confirmation message to the Airline Reservation application, which appears on the Airline Reservation window.

◄ PREVIOUS NEXT ►

Creating the Personal Information Page

The UserInformation.html file creates the user interface to allow end users to enter personal information. Listing 6-1 shows the contents of the UserInformation.html file:

Listing 6-1: The UserInformation.html File

```
<HTML>
<HEAD>
<TITLE>New User </TITLE>
<script language="javascript">
function openwindow(pagename,windowname)
{
   if(document.profileenteryform.firstname.value=="")
   {
      alert('First name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.firstname.value))
      {
         alert('Please enter a valid value for first name.');
         return false;
      }
   }
   if(document.profileenteryform.lastname.value=="")
   {
      alert('Last Name is a required field.');
      return false;
   }
   else
   {
      if(!checkForInt(document.profileenteryform.lastname.value))
      {
         alert('Please enter a valid value for last name.');
         return false;
      }
   }
   if(document.profileenteryform.Address.value=="")
   {
      alert('Address is a required field.');
      return false;
   }
   if(document.profileenteryform.City.value=="")
   {
      alert('City is a required field.');
      return false;
   }
   if(document.profileenteryform.emailid.value=="")
   {
      alert('Email Id is a required field.');
      return false;
   }
   else
   {
   if(!isEmailAddr(document.profileenteryform.emailid.value))
   {
      alert('Please enter a valid value for email ID.');
      return false;
   }
}
function checkForInt(valuestring)
{
   var validcharstring=false;
   var intstring="0123456789!@#$%^&*()_-+="
   for(i=0;i<valuestring.length;i++)
   {
      if (intstring.indexOf(valuestring.charAt(i))!=-1)
      {
         return validcharstring;
      }
   }
   validcharstring=true;
   return validcharstring;
}
function isEmailAddr(email)
{
   var result = false;
   var theStr = new String(email);
   var index = theStr.indexOf("@");
   if (index > 0)
   {
      var pindex = theStr.indexOf(".",index);
```

```
      if ((pindex > index+1) && (theStr.length > pindex+1))
      result = true;
   }
   return result;
}
city=document.profileenteryform.City.value;
address=document.profileenteryform.Address.value;
fname=document.profileenteryform.firstname.value;
lname=document.profileenteryform.lastname.value;
emailid=document.profileenteryform.emailid.value;
pagename=pagename+"?city="+city+"&firstname="+fname+"&lastname="+lname+"&emailid="+emailid
+"&address"+address;
window.open(pagename, windowname, 'width=720, height=622, left=200, top=60,screenX=30, screenY=100')
}
</script>
</HEAD>
<BODY>
<FORM NAME="profileenteryform">
<TABLE WIDTH="100%">
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
<FONT SIZE="4" COLOR="">Airlines Enquiry System</FONT>
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
<FONT SIZE="4" COLOR="">Enter your personal information:</FONT>
</TD>
</TR>
<TR>
<TD WIDTH="100%" ALIGN="CENTER">
 
</TD>
</TR>
<TR>
<TD>
<TABLE align="center" WIDTH="100%" border=0 cellspacing=0 cellpadding=0>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
First Name
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="firstname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Last Name
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="lastname" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Address
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="Address" VALUE="" maxlength=50>
```

```
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="Address" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
City
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" size="40%" width="100%" NAME="City" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
Email ID
</TD>
<TD WIDTH="50%">
<INPUT TYPE="INPUT" width="100%" size="40%" NAME="emailid" VALUE="" maxlength=50>
</TD>
<TD WIDTH="10%">
 
</TD>
</TR>
<TR>
<TD WIDTH="10%">
 
</TD>
<TD WIDTH="30%">
 
</TD>
<TD WIDTH="50%" colspan=2>
<INPUT TYPE="BUTTON" NAME="ok" style="width:20%" VALUE="OK"
onclick="javascript:openwindow('Reservation.html', 'reservation');">
<INPUT TYPE="BUTTON" NAME="cancel" style="width:20%" VALUE="Cancel"
onclick="javascript:window.close();">
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Download this listing.

The UserInformation.html file creates the Personal Information page, as shown in Figure 6-2:

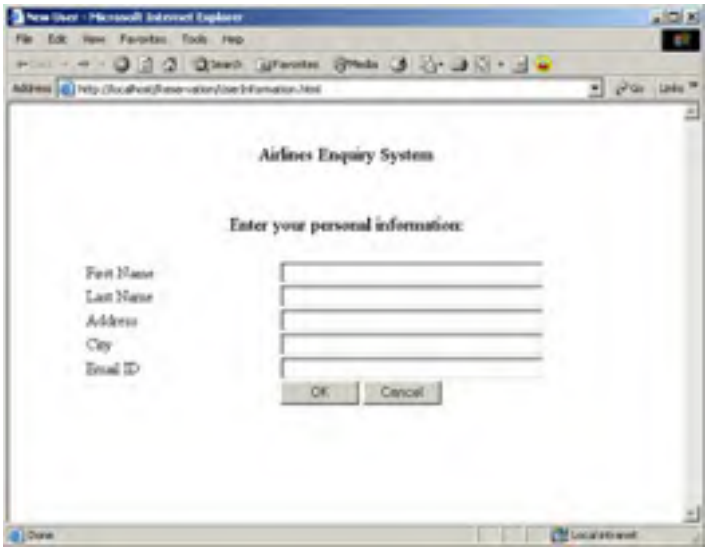


Figure 6-2: The Personal Information Page

◄ PREVIOUS NEXT ►

Sending Personal Information to the Reservation.html File

The Reservation.html file accepts an end user's personal information and calls the ReservationStatus.java file to create the user interface for the Airline Reservation application. Listing 6-2 shows the contents of the Reservation.html file:

Listing 6-2: The Reservation.html File

```
<HTML>
<HEAD>
<TITLE> Airline Reservation </TITLE>
<script>
function PageQuery(q)
{
   if(q.length > 1) this.q = q.substring(1, q.length);
   else this.q = null;
   this.keyValuePairs = new Array();
   if(q)
   {
      for(var i=0; i < this.q.split("&").length; i++)
      {
         this.keyValuePairs[i] = this.q.split("&")[i];
      }
   }
   this.getKeyValuePairs = function() { return this.keyValuePairs; }
   this.getValue = function(s) {
   for(var j=0; j < this.keyValuePairs.length; j++) {
   if(this.keyValuePairs[j].split("=")[0] == s)
   return this.keyValuePairs[j].split("=")[1];
   }
   return false;
}
this.getParameters = function() {
   var a = new Array(this.getLength());
   for(var j=0; j < this.keyValuePairs.length; j++) {
   a[j] = this.keyValuePairs[j].split("=")[0];
   }
   return a;
}
this.getLength = function() { return this.keyValuePairs.length; }
}
function queryString(key){
   var page = new PageQuery(window.location.search);
   return page.getValue(key);
   }
   function appletparam(key){
   if(queryString(key)=='false')
   {
      pagename='UserInformation.html';
      windowname='login';
      window.open(pagename,windowname,'width=400,height=200,left=300,top=100,screenX=500,screenY=100
   }
   else
   {
   return queryString(key);
   }
}
</script>
<SCRIPT LANGUAGE="JavaScript">
function writeAppletTag()
{
   document.writeln('<applet code="ReservationStatus" width="700" height="600">');
   document.writeln(buildParamTag('city',appletparam('city')));
   document.writeln(buildParamTag('firstname',appletparam('firstname')));
   document.writeln(buildParamTag('lastname',appletparam('lastname')));
   document.writeln(buildParamTag('emailid',appletparam('emailid')));
   document.writeln(buildParamTag('address',appletparam('address')));
   document.writeln('</APPLET>');
}
function buildParamTag(name, value) {
   return '<PARAM NAME="' + name + '" VALUE="' + value + '">';
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
writeAppletTag();
</SCRIPT>
</applet>
</BODY>
</HTML>
```

Download this listing.

In the above code, the Reservation.html file calls the ReservationStatus.java file to create the user interface for the Airline Reservation application. The ReservationStatus.java file passes an end user's personal information to the ReservationStatus.java file.

PREVIOUS NEXT

Creating the User Interface for the Airline Reservation Application

The ReservationStatus.java file creates the user interface for the Airline Reservation application. Listing 6-3 shows the contents of the ReservationStatus.java file:

Listing 6-3: The ReservationStatus.java File

```
/*Imports required swing classes*/
import javax.swing.*;
/*Imports required Applet class*/
import java.applet.Applet;
/*Imports required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Imports required i/o classes*/
import java.io.*;
/*Imports required net classes*/
import java.net.*;
/*Imports required util classes*/
import java.util.*;
/*Imports required swing classes*/
import javax.swing.table.*;
import javax.swing.JTextPane;
import javax.swing.text.*;
/*Imports required Vector class*/
import java.util.Vector;
import javax.swing.table.AbstractTableModel;
import javax.swing.event.*;
/*
class ReservationStatus - This class is the main class of the application. This class initializes th
interface and loads all components like the button and textfields before displaying the result.
Constructor:
ReservationStatus-This constructor creates GUI.
Methods:
setTableContent - Sets search result into table.
validation - This method is used to validate user input.
*/
public class ReservationStatus extends JApplet  implements ActionListener
{
   /*Declare objects of JComboBox class.*/
   JComboBox fromcombobox;
   JComboBox tocombobox;
   JComboBox todaycombobox;
   JComboBox tomonthcombobox;
   JComboBox toyearcombobox;
   JComboBox fromdaycombobox;
   JComboBox frommonthcombobox;
   JComboBox fromyearcombobox;
   /*Declare objects of JLabel class.*/
   JLabel fromlabel;
   JLabel tolabel;
   JLabel fromdatelabel;
   JLabel todatelabel;
   /*Declare object of JButton class.*/
   JButton searchbutton;
   /*Declare object of Container class.*/
   Container contentpane;
   /*Declare object of String class.*/
   String TodayDate;
   /*Declare array of String class.*/
   String[] spleteddate;
   /*Declare objects of JRadioButton class.*/
   JRadioButton iradio;
   JRadioButton nriradio;
   /*Declare object of Vector class.*/
   Vector idvector;
   /*Declare object of String class.*/
   String servername="localhost";
   /*Declare object of SocketClass class.*/
   SocketClass socketclass;
   /*Declare objects of String class.*/
   String passportno="100001";
   String city;
   String username;
   String emailid;
   String address;
   /*Declare object of JButton class.*/
   JButton submitbutton;
   /*Declare object of JCheckBox class.*/
   JCheckBox roundtrip;
   /*Declare objects of Vector class.*/
   Vector evector;
   Vector gvector;
   /*Declare integer variable*/
```

```
int selectedrow;
/*Declare object of MyTableReservation class.*/
TableModelReservation mytable;
/*Declare object of ReservationResultTable class.*/
ReservationResultTable t;
/*
This method initializes the object variables in the applet.
*/
public void init()
{
contentpane=getContentPane();
/*Set the BorderLayout to the applet.*/
contentpane.setLayout(new BorderLayout());
/*
Initialize and set the Look and Feel of the application to Windows Look and Feel.
*/
try
{
   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
   System.out.println("Unable to set WLF"+e);
}
/* Initialize the objects of Vector class. */
evector=new Vector();
gvector=new Vector();
city=getParameter("city");
username=getParameter("firstname")+getParameter("lastname");
emailid=getParameter("emailid");
address=getParameter("address");
/*
Declare and initialize the object of JPanel class.
*/
JPanel mainpanel=new JPanel();
/*
Declare and initialize the object of GridLayout class.
*/
GridLayout mainlayout=new GridLayout(2, 2, 5, 5);
/*Set the Gridlayout to mainpanel object.*/
mainpanel.setLayout(mainlayout);
/*
Declare and initialize the object of JPanel class.
*/
JPanel citypanel=new JPanel();
/*
Declare and initialize the object of GridLayout class.
*/
GridLayout citylayout=new GridLayout(4, 1, 15, 5);
/*Set the Gridlayout to citypanel object.*/
citypanel.setLayout(citylayout);
/*Initializes the object of JComboBox class.*/
fromcombobox=new JComboBox();
/*Add the items to the object of JComboBox class.*/
fromcombobox.addItem("Chicago");
fromcombobox.addItem("Boston");
fromcombobox.addItem("Denver");
/*Initializes the object of JLabel class.*/
fromlabel=new JLabel("From",JLabel.CENTER);
/*Sets the font to the object of JLabel class.*/
fromlabel.setFont(new Font("verdana", 1, 12));
/*
Adds the object of the JLabel class to the object JPanel class.
*/
citypanel.add(fromlabel);
/*
Declares and initializes the object of JPanel class.
*/
JPanel fromcombopanel=new JPanel();
/* Sets the size of the object of JComboBox class.*/
fromcombobox.setPreferredSize(new Dimension(200, 20));
/*
Adds the object of JComboBox class to object of JPanel class.
*/
fromcombopanel.add(fromcombobox);
/*
Adds the objects of JPanel class to object of JPanel class.
*/
citypanel.add(fromcombopanel);
mainpanel.add(citypanel);
/*
Declares and initializes the object of JPanel class.
*/
JPanel citypanelto=new JPanel(new GridLayout(4,1,15,5));
/*Initializes the object of JLabel class*/
tolabel=new JLabel("To",JLabel.CENTER);
/*Sets the font of the object of JLabel class.*/
```

```
    tolabel.setFont(new Font("verdana",1,12));
    /* Adds the object of JLabel class to the object of JPanel class.*/
    citypanelto.add(tolabel);
    /*Initializes the object of JComboBox class.*/
    tocombobox=new JComboBox();
    /*Add items to the object of JComboBox class.*/
    tocombobox.addItem("Chicago");
    tocombobox.addItem("Boston");
    tocombobox.addItem("Denver");
    /*
    Sets the preferred size of the object of JComboBox class.
    */
    tocombobox.setPreferredSize(new Dimension(200, 20));
    /*
    Declares and initializes the object of JPanel class.
    */
    JPanel tocombopanel=new JPanel();
    /*
    Adds the object of JCombobox class to the object of JPanel class.
    */
    tocombopanel.add(tocombobox);
    /*
    Adds the object of JPanel class to the object of JPanel class.
    */
    citypanelto.add(tocombopanel);
    /*Initializes the object of the JCheckBox class.*/
    roundtrip=new JCheckBox(" Round trip");
    /*Sets the font of the object of JCheckBox class.*/
    roundtrip.setFont(new Font("verdana",1,12));
    /*
    Adds the ItemListener to the object of JCheckBox class.
    */
    roundtrip.addItemListener(new ItemListener()
    {
       public void itemStateChanged(ItemEvent e)
       {
       if (e.getStateChange() == ItemEvent.SELECTED)
       {
          todaycombobox.setEnabled(true);
          tomonthcombobox.setEnabled(true);
          toyearcombobox.setEnabled(true);
       }
       else
       {
          todaycombobox.setEnabled(false);
          tomonthcombobox.setEnabled(false);
          toyearcombobox.setEnabled(false);
       }
    }
});
/*
Declares and Initializes the object of the JPanel class.
*/
JPanel roundtrippanel=new JPanel();
/*
Adds the object of the JCheckBox class to the object of JPanel class.
*/
roundtrippanel.add(roundtrip);
/*
Adds the object of JPanel class to the object of the JPanel class.
*/
citypanelto.add(roundtrippanel);
/*
Initialize the objects of JComboBox class.
*/
todaycombobox=new JComboBox();
tomonthcombobox=new JComboBox();
toyearcombobox=new JComboBox();
/*
Declares and Initializes the object of JPanel class.
*/
JPanel todaypanel=new JPanel();
todaycombobox.setEnabled(false);
tomonthcombobox.setEnabled(false);
toyearcombobox.setEnabled(false);
/*Sets the size of the object of JComboBox class.*/
todaycombobox.setPreferredSize(new Dimension(40, 20));
/*Calls the addDays() function.*/
addDays(todaycombobox);
/*Sets the size of the object of JComboBox class.*/
tomonthcombobox.setPreferredSize(new Dimension(70, 20));
/*Calls the addMonths() function.*/
addMonths(tomonthcombobox);
/*Sets the size of the object of JComboBox class.*/
toyearcombobox.setPreferredSize(new Dimension(50, 20));
```

```
/*Calls the addYears() function.*/
addYears(toyearcombobox);
/*
Adds the objects of JComboBox class to the object of JPanel class.
*/
todaypanel.add(todaycombobox);
todaypanel.add(tomonthcombobox);
todaypanel.add(toyearcombobox);
/*
Adds the object of JPanel class to the object of JPanel class.
*/
citypanelto.add(todaypanel);
/*
Adds the object of JPanel class to the object of JPanel class.
*/
mainpanel.add(citypanelto);
/*Initializes the object of the JLabel class.*/
fromdatelabel=new JLabel("Departure Date", JLabel.CENTER);
fromdatelabel.setFont(new Font("verdana", 1, 12));
/*
Adds the object of JLabel class to the object of JPanel class.
*/
/* Initializes the objects of JComboBox class.*/
citypanel.add(fromdatelabel);
fromdaycombobox=new JComboBox();
frommonthcombobox=new JComboBox();
fromyearcombobox=new JComboBox();
/*
Declares and Initializes the object of JPanel class.
*/
JPanel fromdaypanel=new JPanel();
/*Sets the size of the object of JComboBox class.*/
fromdaycombobox.setPreferredSize(new Dimension(40, 20));
/*Calls the addDays() function.*/
addDays(fromdaycombobox);
/*Sets the size of the object of JComboBox class.*/
frommonthcombobox.setPreferredSize(new Dimension(70, 20));
/*Calls the addMonths() function.*/
addMonths(frommonthcombobox);
/*Sets the size of the object of JComboBox class.*/
fromyearcombobox.setPreferredSize(new Dimension(50, 20));
/*Calls the addMonths() function.*/
addYears(fromyearcombobox);
/*
Adds the objects of JComboBox class to the object of JPanel class.
*/
fromdaypanel.add(fromdaycombobox);
fromdaypanel.add(frommonthcombobox);
fromdaypanel.add(fromyearcombobox);
citypanel.add(fromdaypanel);
/*
Declares and Initializes the object of JPanel class.
*/
JPanel nationalitypanel=new JPanel();
/*
Declares and Initializes the object of ButtonGroup class.
*/
ButtonGroup nationalitygroup = new ButtonGroup();
/*Initializes the objects of JRadioButton class.*/
iradio=new JRadioButton("US Citizen");
iradio.setSelected(true);
/*Initializes the objects of JRadioButton class.*/
nriradio=new JRadioButton("Foreigner");
/*
Declares and Initializes the object of JPanel class.
*/
JPanel subnationalitypanel=new JPanel(new GridLayout(3, 1, 0, 0));
/*
Declares and Initializes the object of JLabel class.
*/
JLabel nationalitylabel=new JLabel("Nationality");
/*Sets the font of the object of the JLabel class.*/
nationalitylabel.setFont(new Font("verdana", 1, 12));
/*
Adds the object of the JLabel class to the object of the JPanel class.
*/
subnationalitypanel.add(nationalitylabel);
/*
Adds the objects of the JRadioButton class to the object of ButtonGroup class.
*/
nationalitygroup.add(iradio);
nationalitygroup.add(nriradio);
/*
Adds the objects of the JRadioButton class to the object of JPanel class.
```

```
*/
subnationalitypanel.add(iradio);
subnationalitypanel.add(nriradio);
/*
Adds the objects of the JPanel class to the object of JPanel class.
*/
nationalitypanel.add(subnationalitypanel);
mainpanel.add(nationalitypanel);
/*
Adds the object of the JPanel class to the object of ContentPane object.
*/
contentpane.add(mainpanel);
/*Initializes the object of JButton class.*/
searchbutton=new JButton("Search");
/*
Sets the ActionCommand of the object of JButton class.
*/
searchbutton.setActionCommand("search");
searchbutton.setMnemonic('s');
/*
Adds the ActionListener to the object of the JButton class.
*/
searchbutton.addActionListener(this);
/*
Declares and Initializes the object of JPanel class.
*/
JPanel buttonpanel=new JPanel();
/*
Sets the Flowlayout of the object of JPanel class.
*/
buttonpanel.setLayout(new FlowLayout());
/*
Declares and Initializes the object of JButton class.
*/
JButton resetbutton=new JButton("Reset");
/*
Sets the ActionCommand of the object of JButton class.
*/
resetbutton.setActionCommand("reset");
resetbutton.setMnemonic('r');
/*
Adds the ActionListener to the object of the JButton class.
*/
resetbutton.addActionListener(this);
/*
Sets the size of the object of the JButton class
*/
resetbutton.setPreferredSize(new Dimension(80,20));
/*
Initializes the object of ReservationResultTable class.
*/
t=new ReservationResultTable();
/*
Sets the size of the object of the ReservationResultTable class.
*/
t.setPreferredSize(new Dimension(300, 100));
/*
Declares and Initializes the object of JPanel class.
*/
JPanel bpanel=new JPanel(new BorderLayout());
/*
Declares and Initializes the object of JPanel class.
*/
JPanel topsubmitbuttonpanel=new JPanel();
/*
Adds the objects of JButton class to the object of the JPanel class.
*/
topsubmitbuttonpanel.add(searchbutton);
topsubmitbuttonpanel.add(resetbutton);
bpanel.add(topsubmitbuttonpanel,BorderLayout.NORTH);
/*
Sets the size of the object of the JButton class.
*/
searchbutton.setPreferredSize(new Dimension(80, 20));
/*
Adds the object of ReservationResultTable class to the object of the JPanel class.
*/
bpanel.add(t,BorderLayout.CENTER);
/* Initializes the object of JButton class.*/
submitbutton=new JButton("Submit");
/*
Sets the ActionCommand of the object of JButton class.
```

```
*/
submitbutton.setActionCommand("submit");
submitbutton.setMnemonic('u');
/*
Adds the ActionListener to the object of the JButton class.
*/
submitbutton.addActionListener(this);
submitbutton.setEnabled(false);
/*
Adds the object of JButton class to the object of the JPanel class.
*/
bpanel.add(submitbutton,BorderLayout.SOUTH);
/*
Adds the object of JPanel class to the object of the ContentPane class.
*/
contentpane.add(bpanel,BorderLayout.SOUTH);
setSize(650,340);
/* Initializes the object of SocketClass class.*/
socketclass=new SocketClass("localhost", 5222, 1000,this);
}
public String getUserName()
{
   return username;
}
/*
setTableContent - Inserts the search result data into the table.
Parameters:messagebody - object of String class.
Return Value: NA
*/
public void setTableContent(String messagebody)
{
if (messagebody.trim().equals("No matching record found"))
{
   JOptionPane.showMessageDialog(null,messagebody,"Dialog box",JOptionPane.PLAIN_MESSAGE);
   submitbutton.setEnabled(false);
   return;
}
if(messagebody.indexOf("@")>0)
{
   String[] spletedschedule=messagebody.split("@");
   submitbutton.setEnabled(true);
   idvector=new Vector();
   Object[][] object=new Object[spletedschedule.length][];
   for (int i=0;i<spletedschedule.length;i++ )
   {
      String[] sp=spletedschedule[i].split("#");
      object[i]=new Object[sp.length-1];
      for (int j=0;j<sp.length-3;j++ )
      {
         String[] splitedvalue=sp[j].split("=");
         if (splitedvalue.length==2)
         {
            if (splitedvalue[1].equals("YES"))
            {
               object[i][j]=new Boolean(false);
            }
            else
            {
               if (splitedvalue[1].equals("NO"))
               {
                  object[i][j]="NA";
               }
               else
               {
                  object[i][j]=splitedvalue[1];
               }
            }
         }
         idvector.add(sp[sp.length-3]);
         evector.add(sp[sp.length-2]);
         gvector.add(sp[sp.length-1]);
      }
   }
   mytable= new TableModelReservation();
   mytable.setData(object);
   t.table.setModel(mytable);
}
}
/*
class TableModelReservation - This class is the main class of the application. This class initialize
the interface and loads all components like the button, textfields before displaying the result.
Methods:
setData - Sets Object array to table object.
getRowCount - This method returns the total number of rows.
getColumnName - This method returns the Column name.
getValueAt - This method returns the value at a particular cell location.
setValueAt - This method sets the value at a particular cell location.
```

```
*/
class TableModelReservation extends AbstractTableModel
{
   private String[] columnNames = {"Choice","Flight No.",
   "Aircraft type",
   "Origin",
   "No. of Stops",
   "Destination",
   "Dep.Date-time",
   "Arr.Date-time",
   "Executive Class",
   "Economy Class",
};
private Object[][] data = {};
public int getColumnCount()
{
   return columnNames.length;
}
public void setData(Object[][] dataarr)
{
   data=dataarr;
}
public int getRowCount()
{
   return data.length;
}
public String getColumnName(int col)
{
   return columnNames[col];
}
public Object getValueAt(int row, int col)
{
   return data[row][col];
}
public Class getColumnClass(int c)
{
   return getValueAt(0, c).getClass();
}
public boolean isCellEditable(int row, int col)
{
   if (col >=1&&col<=7)
   {
      return false;
   }
   else
   {
      return true;
   }
}
public void setValueAt(Object value, int row, int col)
{
   data[row][col] = value;
   fireTableCellUpdated(row, col);
   if (col==0&&value.toString().equals("true"))
   {
      int totalrows=getRowCount();
      for (int i=0;i<totalrows;i++ )
      {
         selectedrow=row;
         if (row!=i)
         {
            data[i][col] =new Boolean(false);
         }
      }
   }
   if ((col==9)&&value.toString().equals("true"))
   {
      data[row][col-1] = new Boolean(false);
      fireTableCellUpdated(row, col-1);
   }
   else
   {
      if (col==8&&value.toString().equals("true"))
      {
         data[row][col+1] =new Boolean(false);
         fireTableCellUpdated(row, col+1);
      }
   }
}
}
public void actionPerformed(ActionEvent ae)
{
   String journeytypestring="Oneway";
   String nationalitystring="US Citizen";
   String actioncommand=ae.getActionCommand();
   if (actioncommand.equals("search"))
```

```
    {
    String informationstring;
    String seachstring;
    if (validation())
    {
    if (iradio.isSelected())
    {
       nationalitystring=iradio.getText();
    }
    if (nriradio.isSelected())
    {
    nationalitystring=nriradio.getText();;
    }
    if (roundtrip.isSelected())
    {
       journeytypestring="roundtrip";
    }
    String dmonthpart="";
    String rmonthpart="";
    String rdaypart="";
    String ddaypart="";
    int dday=fromdaycombobox.getSelectedIndex()+1;
    int dmonth=frommonthcombobox.getSelectedIndex()+1;
    if (dday<10)
    {
       ddaypart="0";
    }
    if (dmonth<10)
    {
       dmonthpart="0";
    }
    int rmonth=tomonthcombobox.getSelectedIndex()+1;
    if (rmonth<10)
    {
       rmonthpart="0";
    }
    int rday=todaycombobox.getSelectedIndex()+1;
    if (rday<10)
    {
       rdaypart="0";
    }
    seachstring="#origin="+((String)fromcombobox.getSelectedItem());
    seachstring=seachstring+"#destination="+((String)tocombobox.getSelectedItem());
    seachstring=seachstring+"#ddate="+((String)fromyearcombobox.getSelectedItem())+dmonthpart+
    dmonth+dday
    part+dday;  seachstring=seachstring+"#rdate="+((String)toyearcombobox.getSelectedItem())+
    rmonthpart+rmonth+rdaypart+rday;
    seachstring=seachstring+"#nationality="+nationalitystring;
    seachstring=seachstring+"#typeofjurney="+journeytypestring;
    informationstring="<message to=\"user10@"+servername+"/airlines\" type=\"chat\">";
    informationstring=informationstring+"<body>";
    informationstring=informationstring+seachstring;
    informationstring=informationstring+"</body></message>";
    socketclass.sendXMLToJabber(informationstring);
}
}
if (actioncommand.equals("reset"))
{
    fromcombobox.setSelectedIndex(0);
    tocombobox.setSelectedIndex(0);
    todaycombobox.setSelectedIndex(0);
    tomonthcombobox.setSelectedIndex(0);
    toyearcombobox.setSelectedIndex(0);
    fromdaycombobox.setSelectedIndex(0);
    frommonthcombobox.setSelectedIndex(0);
    Object[][] emptyobject = {};
    TableModelReservation emptytable=new TableModelReservation();
    emptytable.setData(emptyobject);
    t.table.setModel(emptytable);
    submitbutton.setEnabled(false);
}
if (actioncommand.equals("submit"))
{
    String idstring=(String)idvector.get(selectedrow);
    String submitstring="";
    submitstring=submitstring+"<message to=\"user10@"+servername+"/airlines\"
    type=\"chat\"><body>";
    submitstring=submitstring+"#id"+idstring+"#username"+username+"#address="+address+"#city="+city+
    "#passportno"+passportno;
    submitstring=submitstring+"</body></message>";
    socketclass.sendXMLToJabber(submitstring);
    Object[][] emptyobject = {};
    TableModelReservation emptytable=new TableModelReservation();
    emptytable.setData(emptyobject);
    t.table.setModel(emptytable);
    JOptionPane.showMessageDialog(null,"Data has been submitted, for further details contact airlines
```

```
   administrator","Dialog box",JOptionPane.PLAIN_MESSAGE);
   submitbutton.setEnabled(false);
}
}
/*
validation - This method is used to validate user input.
Parameters:NA
Return Value: boolean
*/
public boolean validation()
{
   boolean valid=true;
   String validationmessage="";
   if ((String)fromcombobox.getSelectedItem()==(String)tocombobox.getSelectedItem())
   {
      validationmessage="Destination and origin should not be same.";
      valid=false;
   }
   if (Integer.parseInt(spleteddate[1])>(frommonthcombobox.getSelectedIndex()+1))
   {
      validationmessage=validationmessage+"\n"+"Departure date can not be less than today's date.";
      valid=false;
   }
   else
   {
      if (Integer.parseInt(spleteddate[1])==(frommonthcombobox.getSelectedIndex()+1))
      {
         String[] splitday=spleteddate[2].split(" ");
         if (Integer.parseInt(splitday[0])>(fromdaycombobox.getSelectedIndex()+1))
         {
            validationmessage=validationmessage+"\n"+"Departure date can not be less than today's
            date.";
            valid=false;
         }
      }
   }
   if (roundtrip.isSelected())
   {
      if (Integer.parseInt(spleteddate[1])>(tomonthcombobox.getSelectedIndex()+1))
      {
      validationmessage=validationmessage+"\n"+"Return date can not be less than today's date.";
      valid=false;
      }
      else
      {
         if (Integer.parseInt(spleteddate[1])==(tomonthcombobox.getSelectedIndex()+1))
         {
         String[] splitday=spleteddate[2].split(" ");
         if (Integer.parseInt(splitday[0])>(todaycombobox.getSelectedIndex()+1))
         {
            validationmessage=validationmessage+"\n"+"Return date can not be less than today's date.
            valid=false;
         }
         }
      }
   }
   if (roundtrip.isSelected())
   {
      if (Integer.parseInt((String)toyearcombobox.getSelectedItem())<Integer.parseInt((String)
      fromyearcombobox.getSelectedItem()))
      {
         validationmessage=validationmessage+"\n"+"Return date can not be less than Departure date."
         valid=false;
      }
      else
      {
         if ((frommonthcombobox.getSelectedIndex())>(tomonthcombobox.getSelectedIndex()))
         {
            validationmessage=validationmessage+"\n"+"Return date can not be less than Departur date
            valid=false;
         }
         else
         {
            if ((frommonthcombobox.getSelectedIndex())==(tomonthcombobox.getSelectedIndex()))
            {
               if ((fromdaycombobox.getSelectedIndex())>(todaycombobox.getSelectedIndex()))
               {
                  validationmessage=validationmessage+"\n"+"Return date can not be less than Departu
                  date.";
                  valid=false;
               }
            }
         }
      }
   }
   if (!validationmessage.equals(""))
   {
```

```
   JOptionPane.showMessageDialog(null,validationmessage,"Dialog box",JOptionPane.PLAIN_MESSAGE);
   }
   return valid;
}
/*
addDays - This method is used to fill days into fromdaycombobox.
Parameters:fromdaycombobox - Object of JComboBox class.
Return Value: NA
*/
public void addDays(JComboBox fromdaycombobox)
{
   for (int i=1;i<32 ;i++ )
   {
      fromdaycombobox.addItem(new Integer(i));
   }
}
/*
addMonths - This method is used to fill months into frommonthcombobox Combo.
Parameters:frommonthcombobox - Object of JComboBox class
Return Value: NA
*/
public void addMonths(JComboBox frommonthcombobox)
{
   frommonthcombobox.addItem("January");
   frommonthcombobox.addItem("February");
   frommonthcombobox.addItem("March");
   frommonthcombobox.addItem("April");
   frommonthcombobox.addItem("May");
   frommonthcombobox.addItem("June");
   frommonthcombobox.addItem("July");
   frommonthcombobox.addItem("August");
   frommonthcombobox.addItem("September");
   frommonthcombobox.addItem("October");
   frommonthcombobox.addItem("November");
   frommonthcombobox.addItem("December");
}
/*
addYears - This method is used to fill years into fromyearcombobox Combo.
Parameters:frommonthcombobox - Object of JComboBox class
Return Value: NA
*/
public void addYears(JComboBox fromyearcombobox)
{
   Calendar cal = Calendar.getInstance(TimeZone.getDefault());
   String DATE_FORMAT = "yyyy-MM-dd HH:mm:ss";
   java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat(DATE_FORMAT);
   sdf.setTimeZone(TimeZone.getDefault());
   TodayDate=sdf.format(cal.getTime());
   System.out.println(TodayDate);
   spleteddate=TodayDate.split("-");
   fromyearcombobox.addItem(spleteddate[0]);
   fromyearcombobox.addItem(new Integer(Integer.parseInt(spleteddate[0])+1));
}
}
```

Download this listing.

In the above listing, the ReservationStatus.java file creates the user interface by using the ReservationStatus class. The ReservationStatus.java file uses the TableModelReservation class to display the airline schedule in the user interface provided by the Airline Reservation application.

The methods defined in Listing 6-3 are:

- `init()`: Creates the user interface for the Airline Reservation application and calls the SocketClass class to establish a connection with the Jabber server.
- `getUserName()`: Returns the name of the end user.
- `setTableContent()`: Initializes the TableModelReservation class and displays the airline schedule in the user interface provided by the Airline Reservation application.
- `itemStateChanged()`: Activates the combo boxes for day, month, and year, if an end user selects the Round trip option.
- `addDays()`: Enters number of days in the combo box for the departure date and arrival date.
- `addMonths()`: Enters number of months in the combo box for the departure date and arrival date.
- `addYears()`: Enters number of years in the combo box for the departure date and arrival date.
- `setValueAt()`: Shows the airline schedule received from the Jabber server.
- `actionPerformed()`: Acts as an event listener and activates an appropriate class and method based on the action an end user performs.
- `getRowCount()`: Returns the total number of records in an airline schedule.

- `validation()`: Validates the information specified by the end user.

The ReservationStatus.java file creates the main window of the Airline Reservation application, as shown in Figure 6-3:

Figure 6-3: User Interface of the Airline Reservation Application

PREVIOUS NEXT

Creating the Socket Class to Send and Receive Messages

The SocketClass.java file opens the socket with the Jabber server to send and receive messages between end users. Listing 6-4 shows the contents of the SocketClass.java file:

Listing 6-4: The SocketClass.java File

```
/*Imports required java.util classes*/
import java.util.*;
/*Imports required java.io classes*/
import java.io.*;
/*Imports required java.net classes*/
import java.net.*;
/*Imports required javax.swing classes*/
import javax.swing.*;
/*
class SocketClass - This class is used for reading the input messages and writing output messages.
Constructor:
SocketClass - This constructor calls a method of SocketOpener class to open a client socket.
Methods:
*/
class SocketClass implements Runnable
{
   public boolean isConnected;
   /*Declare the object of the Vector class.*/
   public Vector tagvactor;
   /*
   Declare the objects of the PrintWriter class.
   */
   PrintWriter out = null;
   /*Declare the objects of the BufferedReader class.*/
   BufferedReader in = null;
   /*Declare the objects of the String class.*/
   String ipaddress="";
   String errortype;
   String resource="Airline";
   String username;
   String password;
   /*Declare the objects of the Socket class.*/
   Socket clientsocket;
   /*Declare the objects of the Thread class.*/
   Thread inputmessagethread;
   /*
   Declare the objects of the ReservationStatus class.
   */
   ReservationStatus reservation;
   int portno=5222;
   int wait=1000;
   public SocketClass(String ipaddress, int portno, int wait, ReservationStatus reservation)
   {
      this.ipaddress=ipaddress;
      this.portno=portno;
      this.wait=wait;
      this.reservation=reservation;
      /*Call openPort() method.*/
      openPort(ipaddress,portno,wait);
      if (clientsocket!=null)
      {
      isConnected=true;
      }
      try
      {
      /*
      Initializes the object of PrintWriter class.
      */
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      /*
      Initializes the object of PrintWriter class.
      */
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
      }
      catch(Exception e)
      {
      e.printStackTrace();
      }
      /*
      Declare the objects of the String class.
      */
      String sessionStratString;
      String registrationstring;
      String authentication;
      username=reservation.getUserName();
      sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
```

```
   sessionStratString=sessionStratString+" <stream:stream";
   sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
   sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   sendXMLToJabber(sessionStratString);
   registrationstring="<iq type=\"set\" id=\"1301\">";
   registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
   registrationstring=registrationstring+"<username>"+username+"</username>";
   registrationstring=registrationstring+"<password>"+username+"</password>";
   registrationstring=registrationstring+"</query></iq>";
   sendXMLToJabber(registrationstring);
   authentication="<iq type=\"set\" id=\"1301\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+" <username>"+username+"</username>";
   authentication=authentication+" <password>"+username+"</password>";
   authentication=authentication+" <resource>"+resource+"</resource> ";
   authentication=authentication+" </query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
   /*
   Initializes the object of Thread class.
   */
   inputmessagethread=new Thread(this);
   /*
   Calls start() method of the object of Thread class.
   */
   inputmessagethread.start();
}
/*
openPort: This method creates an object of SocketOpener class and class openSocket method of this
class.
Parameter: ipaddress - Object of String class,portno - int, timeinsec - int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   /*
   Declare the objects of the String class.
   */
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      /*
      Declare and initializes the objects of the SocketOpener class.
      */
      SocketOpener socketopener=new SocketOpener();
      /*
      Calls openSocket Method() of SocketOpener
      */
      clientsocket=socketopener.openSocket(ipaddress,portno,timeinsec);
   }
}
public void run()
{
int i=0;
String errororwarning="";
String errorstring="";
String inputstring="";
boolean starttag=false;
boolean endtag=false;
boolean startwritting=false;
String messagebody="";
String tagentity="";
Vector tagvactor;
tagvactor=new Vector();
int vactorindex=0;
int starttagsing=0;
while (true)
{
try
{
i=in.read();
if (i==-1)
{
   break;
}
else
{
inputstring=inputstring+(char)i;
if ((char)i=='<')
{
   starttag=true;
   starttagsing=1;
   tagentity="";
```

```
    }
    else
    {
    if ((char)i=='/' && starttag==true )
    {
       if (starttagsing==1)
       {
          starttag=false;
          endtag=true;
          reservation.setTableContent(messagebody);
          startwritting=false;
          messagebody="";
          vactorindex=vactorindex-1;
          if (vactorindex>=0)
          tagvactor.removeElementAt(vactorindex);
       }
    }
    else
    {
    starttagsing=0;
    if((char)i=='>')
    {
    if (starttag)
    {
       sendername(tagentity);
       errorstring=checkForError(tagentity);
       if (!errorstring.equals(""))
       {
       if (errortype=="major")
       {
          errororwarning="Error";
          JOptionPane.showMessageDialog(null,errorstring,errororwarning,JOptionPane.PLAIN_MESSAGE);
       }
       else
       {
          errororwarning="Warning";
          JOptionPane.showMessageDialog(null,errorstring,errororwarning,JOptionPane.PLAIN_MESSAGE);
       }
    }
    startwritting=true;
    tagvactor.insertElementAt(tagentity,vactorindex);
    vactorindex=vactorindex+1;
    starttag=false;
    }
}
else
{
   if (startwritting==true&&tagentity.trim().equals("body"))
   {
      messagebody=messagebody+(char)i;
   }
   if (starttag)
   {
      tagentity=tagentity+(char)i;
   }
}
}
}
}
}
catch(IOException ie)
{
}
}
isConnected=false;
}
/*
sendXMLToJabber: This method sends XML message string for the jabber server.
Parameter: outputmessage - object of String class
Return Value: N/A
*/
public void sendXMLToJabber(String outputmessage)
{
String[] tokenizerstring=outputmessage.split("\n");
for (int i=0;i<tokenizerstring.length;i++ )
{
   try
   {
      out.println(tokenizerstring[i]);
      out.flush();
   }
   catch(Exception e)
   {
      System.out.println("error");
      return;
   }
   out.flush();
   }
```

```
}
/*
getErrorString: This method checks input string for error code.
Parameter: codeid - object of String class
Return Value: String
*/
public String getErrorString(String codeid)
{
   if (codeid=="'401'")
   {
      errortype="minor";
      return "You have sent malformed syntax, which can not be understood by server.";
   }
   if (codeid=="'404'")
   {
      errortype="major";
      return "No User exist with this user id.";
   }
   if (codeid=="'405'")
   {
      errortype="minor";
      return "You have no right to send this command.";
   }
   if (codeid=="'409'")
   {
      errortype="major";
      return "A user with this jabber id is already exist. Please choose another user id.";
   }
   return "";
}
/*
checkForError: This method subtracts input string, retrieves errorcode and calls getErrorString()
Parameter: tagentity - object of String class
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if (tagentity.startsWith("error"))
   {
      if (tagentity.indexOf("code=")>0)
      {
         codeid=tagentity.substring(tagentity.indexOf("code=")+6,tagentity.indexOf("type="));
         error=getErrorString(codeid.trim());
      }
   }
   return error;
}
/*
sendername: This method extracts sender name from input message.
Parameter: tagentity - object of String class
Return Value: N/A
*/
public void sendername(String tagentity)
{
   String sendername="";
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("id=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("id="));
      }
   }
}
}
/*
class SocketOpener - This class is used to create an object of socket class.
Constructor:
openSocket
Methods:
getSocket: This method returns a handler of the object of socket class.
*/
class SocketOpener
{
   private Socket socket;
   public  Socket openSocket(String hostip,int portnumber,int timeinsec)
   {
      try
      {
         socket=new Socket(hostip,portnumber);
      }
      catch(IOException ie)
      {
      }
      return getSocket();
```

```
    }
    public SocketOpener()
    {
    }
    /*
    getSocket: This method returns a handler of the object of socket class.
    Parameter: N/A
    Return Value: Socket
    */
    public Socket getSocket()
    {
       return socket;
    }
}
```

Download this listing.

In the above code, the constructor of the SocketClass class takes an object of the ReservationStatus class, ipaddress, portno, and wait as input parameters. The object of the ReservationStatus class allows an end user to invoke the methods of the ReservationStatus class. The ipaddress string retrieves the ip address of the Jabber server to connect. The portno int retrieves the port number of the Jabber server to open the socket. The wait int retrieves the initial time to wait before connecting to the Jabber server.

The methods defined in Listing 6-4 are:

- `openPort()`: Creates an object of the SocketOpener class and calls the openSocket() method by using the object of the SocketOpener class to open a socket between the Airline Reservation application and the Jabber server.
- `sendXMLToJabber()`: Sends an XML message string to the Jabber server.
- `checkForError()`: Checks for errors in the information specified by end users and calls the getErrorString() method. The checkForError() method passes the error code to the getErrorString() method as an input parameter.
- `getErrorString()`: Shows an error message to end users based on the error code passed as an input parameter.
- `sendername()`: Retrieves the name of the sender from the input message and checks the identity of the sender.
- `run()`: Listens for the response sent by the Jabber server.
- `OpenSocket()`: Initializes the object of the SocketOpener class.
- `getSocket()`: Returns an object of the Socket class.

Searching for the Airline Schedules

The SocketConector.java file reads the airlines.xml file to search for airline schedules that match an end user's query. The SocketConector class opens the socket to receive end user's query from the Jabber server and sends the matched airline schedules to the Jabber server. Listing 6-5 shows the contents of the SocketConnector.java file:

Listing 6-5: The SocketConnector.java File

```
/*Imports required java.util classes*/
import java.util.*;
/*Imports required java.io classes*/
import java.io.*;
/*Imports required  java.net classes*/
import java.net.*;
/*Imports required swing classes*/
import javax.swing.*;
/*Imports required List class*/
import java.util.List;
/*Imports required Element class*/
import org.jdom.Element;
/*Imports required Iterator class*/
import java.util.Iterator;
/*
class SocketConnector - This class is used for reading the input messages and writing output message
Constructor:
SocketConnector-This constructor calls openPort() method to open a client socket.
Methods:
openPort
*/
class SocketConnector implements Runnable
{
/*Declare object of Vector class.*/
public Vector tagvactor;
/* Declare object of PrintWriter class.*/
PrintWriter out = null;
/*Declare object of BufferedReader class.*/
BufferedReader in = null;
/*Declare object of Socket class.*/
Socket clientsocket;
/*Declare object of Thread class.*/
Thread inputmessagethread;
/*Declare objects of String class.*/
String errortype;
String ipaddress="localhost";
String resource="airlines";
String username="user10";
String password="user10";
String sendernamestring;
int portno=5222;
int wait=1000;
public boolean isConnected;
/*
SocketConnector: This method call an openPort() method.
Parameter: ipaddress - object of String class, portno, wait
Return Value: boolean
*/
public SocketConnector(String ipaddress,int portno,int wait)
{
   this.ipaddress=ipaddress;
   this.portno=portno;
   this.wait=wait;
   openPort(ipaddress,portno,wait);
   if (clientsocket!=null)
   {
      isConnected=true;
   }
   else
      {
      System.err.println("Server not found.");
      return;
   }
   try
   {
      /*
      Initializes the object of PrintWriter class.
      */
      out = new PrintWriter(clientsocket.getOutputStream(), true);
      /*
      Initializes the object of BufferedReader class.
      */
      in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
   }
```

```
   catch(Exception e)
   {
      e.printStackTrace();
   }
   String sessionStratString;
   String authentication;
   String registrationstring;
   sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
   sessionStratString=sessionStratString+" <stream:stream";
   sessionStratString=sessionStratString+" to= \""+ipaddress +"\"";
   sessionStratString=sessionStratString+"  xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   sendXMLToJabber(sessionStratString);
   authentication="<iq type=\"set\" id=\"1301\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+" <username>"+username+"</username>";
   authentication=authentication+" <password>"+password+"</password>";
   authentication=authentication+" <resource>"+resource+"</resource> ";
   authentication=authentication+" </query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
   /*
   Initializes the object of Thread class.
   */
   inputmessagethread=new Thread(this);
   /*
   Calls start method of inputmessagethread.
   */
   inputmessagethread.start();
}
public static void main(String args[])
{
   SocketConnector socketconnector=new SocketConnector("localhost",5222,1000);
}
/*
openPort: This method creates an object of SocketOpener class and class openSocket method of this
class.
Parameter: ipaddress - Object of String class,portno - int, timeinsec - int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno,int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      clientsocket=SocketOpenerClass.openSocket(ipaddress,portno,timeinsec);
   }
}
public void run()
{
   int i=0;
   String errororwarning="";
   String errorstring="";
   String inputstring="";
   boolean starttag=false;
   boolean endtag=false;
   boolean startwritting=false;
   String messagebody="";
   String tagentity="";
   Vector tagvactor;
   tagvactor=new Vector();
   int vactorindex=0;
   int starttagsing=0;
   while (true)
   {
   try
   {
   i=in.read();
   if (i==-1)
   {
      break;
   }
   else
   {
      inputstring=inputstring+(char)i;
      if ((char)i=='<')
      {
         starttag=true;
         starttagsing=1;
         tagentity="";
      }
      else
      {
         if ((char)i=='/' && starttag==true&&starttagsing==1)
```

```
            {
               starttag=false;
               endtag=true;
               if (!messagebody.trim().equals(""))
               {
                  writemessage(messagebody,sendernamestring);
               }
               startwritting=false;
               messagebody="";
               vactorindex=vactorindex-1;
               if (vactorindex>=0)
               tagvactor.removeElementAt(vactorindex);
            }
            else
            {
               starttagsing=0;
               if((char)i=='>')
               {
               if (starttag)
               {
                  sendername(tagentity);
                  errorstring=checkForError(tagentity);
                  if (!errorstring.equals(""))
                  {
                     if (errortype=="major")
                     {
                        errororwarning="Error";
                        JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                        JOptionPane.PLAIN_MESSAGE);
                     }
                     else
                     {
                        errororwarning="Warning";
                        JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                        JOptionPane.PLAIN_MESSAGE);
                     }
                  }
                  startwritting=true;
                  tagvactor.insertElementAt(tagentity,vactorindex);
                  vactorindex=vactorindex+1;
                  starttag=false;
                  }
               }
               else
               {
                  if (startwritting==true&&tagentity.trim().equals("body"))
                  {
                     messagebody=messagebody+(char)i;
                  }
                  if (starttag)
                  {
                     tagentity=tagentity+(char)i;
                  }
               }
            }
         }
      }
   }
}
catch(IOException ie)
{
}
}
isConnected=false;
try
{
   inputmessagethread.join();
}
catch(Exception e)
{
}
}
/*
writemessage: This method creates an object of XMLReader class, reads data from XML file, and send
result to the client.
Parameter: messagebody - Object of String class, sendername - Object of String class.
Return Value: N/A
*/
public void writemessage(String messagebody,String sendername)
{
String[] spletedmessage=messagebody.split("#");
if (spletedmessage[1].indexOf("id")>=0)
{
   return;
}
String origin=spletedmessage[1].substring(7);
String destination=spletedmessage[2].substring(12);
String ddate=spletedmessage[3].substring(6);
String rdate=spletedmessage[4].substring(6);
```

```
String nationality=spletedmessage[5].substring(12);
String typeofjurney=spletedmessage[6].substring(13);
java.util.List flightdescriptionlist;
XMLReader xmlreader=new XMLReader();
flightdescriptionlist=xmlreader.readFile("airlines.xml");
Iterator i = flightdescriptionlist.iterator();
String msgstr="";
String startmsgstr="@";
while (i.hasNext())
{
Element dateelement;
Element depdate=null;
Element arrdate=null;
Element eclass=null;
Element gclass=null;
Element eclassprice=null;
Element gclassprice=null;
Element arrtime=null;
Element dtime=null;
Element flightelement = (Element) i.next();
Element idelement=flightelement.getChild("id");
Element originelement=flightelement.getChild("origin");
Element flighnotelement=flightelement.getChild("flightno");
Element aircrafttypeelement=flightelement.getChild("aircrafttype");
Element noofstopelement=flightelement.getChild("noofstop");
Element destinationelement=flightelement.getChild("destination");
List executiveelement=flightelement.getChildren("class");
Iterator classiterator=executiveelement.iterator();
while (classiterator.hasNext())
{
   Element classelement=(Element) classiterator.next();
   eclass=classelement.getChild("executive");
   gclass=classelement.getChild("economy");
   eclassprice=classelement.getChild("executiveprice");
   gclassprice=classelement.getChild("economyprice");
}
Element economyelement=flightelement.getChild("economy");
List departurdatetimelist=flightelement.getChildren("departurdatetime");
List arrivaldatetimelist=flightelement.getChildren("arrivaldatetime");
Iterator idepartur=departurdatetimelist.iterator();
Iterator iarrivalarrival=arrivaldatetimelist.iterator();
while(idepartur.hasNext())
{
   Element departuredateelement=(Element) idepartur.next();
   depdate=departuredateelement.getChild("date");
   dtime=departuredateelement.getChild("dtime");
}
while(iarrivalarrival.hasNext())
{
   Element arrivaldateelement=(Element) iarrivalarrival.next();
   arrdate=arrivaldateelement.getChild("date");
   arrtime=arrivaldateelement.getChild("atime");
}
Element arrivaldateelement=flightelement.getChild("arrivaldatetime");
if (origin.equals(originelement.getTextTrim())&&destination.equals
(destinationelement.getTextTrim())&&ddate.equals(depdate.getTextTrim()))
{
   msgstr=msgstr+"choice=YES";
   msgstr=msgstr+"#flight="+flighnotelement.getTextTrim();
   msgstr=msgstr+"#aircrafttype="+aircrafttypeelement.getTextTrim();
   msgstr=msgstr+"#origin="+originelement.getTextTrim();
   msgstr=msgstr+"#noofstop="+noofstopelement.getTextTrim();
   msgstr=msgstr+"#destination="+destinationelement.getTextTrim();
   msgstr=msgstr+"#ddate="+depdate.getTextTrim()+" "+dtime.getTextTrim();
   msgstr=msgstr+"#adate="+arrdate.getTextTrim()+" "+arrtime.getTextTrim();
   msgstr=msgstr+"#executive="+eclass.getTextTrim();
   msgstr=msgstr+"#economy="+gclass.getTextTrim();
   msgstr=msgstr+"#id="+idelement.getTextTrim();
   msgstr=msgstr+"#eclassprice="+eclassprice.getTextTrim();
   msgstr=msgstr+"#gclassprice="+gclassprice.getTextTrim();
   msgstr=msgstr+startmsgstr;
}
if(typeofjurney.equals("roundtrip"))
{
   if (destination.equals(originelement.getTextTrim())&&origin.equals
   (destinationelement.getTextTrim())&&rdate.equals(depdate.getTextTrim()))
   {
      msgstr=msgstr+"choice=YES";
      msgstr=msgstr+"#flight="+flighnotelement.getTextTrim();
      msgstr=msgstr+"#aircrafttype="+aircrafttypeelement.getTextTrim();
      msgstr=msgstr+"#origin="+originelement.getTextTrim();
      msgstr=msgstr+"#noofstop="+noofstopelement.getTextTrim();
      msgstr=msgstr+"#destination="+destinationelement.getTextTrim();
      msgstr=msgstr+"#ddate="+depdate.getTextTrim()+" "+dtime.getTextTrim();
      msgstr=msgstr+"#adate="+arrdate.getTextTrim()+" "+arrtime.getTextTrim();
      msgstr=msgstr+"#executive="+eclass.getTextTrim();
      msgstr=msgstr+"#economy="+gclass.getTextTrim();
      msgstr=msgstr+"#id="+idelement.getTextTrim();
```

```
            msgstr=msgstr+"#eclassprice="+eclassprice.getTextTrim();
            msgstr=msgstr+"#gclassprice="+gclassprice.getTextTrim();
            msgstr=msgstr+startmsgstr;
         }
      }
      }
      if (msgstr.equals(""))
      {
         sendXMLToJabber("<message to='"+sendername+" type=\"chat\"><body
         >No matching record found</body></message>");
      }
      else
      {
         sendXMLToJabber("<message to='"+sendername+"
         type=\"chat\"><body>"+msgstr+"</body></message>");
      }
      }
      /*
      sendXMLToJabber: This method sends XML message string for the jabber server.
      Parameter: outputmessage - object of String class
      Return Value: N/A
      */
      public void sendXMLToJabber(String outputmessage)
      {
         String[] tokenizerstring=outputmessage.split("\n");
         System.out.println(outputmessage);
         for (int i=0;i<tokenizerstring.length;i++ )
         {
         try
         {
            out.println(tokenizerstring[i]);
            out.flush();
         }
         catch(Exception e)
         {
            System.out.println("error");
            return;
         }
         out.flush();
         }
      }
      /*
      getErrorString: This method checks input string for error code.
      Parameter: codeid - object of String class
      Return Value: String
      */
      public String getErrorString(String codeid)
      {
         if (codeid=="'401'")
         {
            errortype="minor";
            return "You have sent malformed syntax, which can not be understood by server.";
         }
         if (codeid=="'404'")
         {
            errortype="major";
            return "No User exist with this user id.";
         }
         if (codeid=="'405'")
         {
            errortype="minor";
            return "You have no right to send this command.";
         }
         if (codeid=="'409'")
         {
            errortype="major";
            return "A user with this jabber id is already exist. Please choose another user id.";
         }
         return "";
         }
         /*
         checkForError: This method substracts input string, retrieves errorcode, and calls getErrorString
         Parameter: tagentity - object of String class
         Return Value: String
         */
         public String checkForError(String tagentity)
         {
            String error="";
            String codeid="";
            if (tagentity.startsWith("error"))
            {
            if (tagentity.indexOf("code=")>0)
            {
               codeid=tagentity.substring(tagentity.indexOf("code=")+6,tagentity.indexOf("type="));
               error=getErrorString(codeid.trim());
            }
         }
```

```
   return error;
   }
   /*
   sendername: This method extracts sender name from input message.
   Parameter: tagentity - object of String class
   Return Value: N/A
   */
   public void sendername(String tagentity)
   {
   String sendername="";
   if (tagentity.startsWith("message"))
   {
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
      {
         sendernamestring=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to="))
      }
   }
}
}
/*
class SocketOpenerClass - This class is used to create an object of socket class.
Constructor:
openSocket
Methods:
getSocket: This method returns an handler of the object of socket class.
*/
class SocketOpenerClass implements Runnable
{
private String openerhost;
private int openerport;
private Socket socket;
public static Socket openSocket(String hostip,int portnumber,int timeinsec)
{
SocketOpenerClass opener=new SocketOpenerClass(hostip,portnumber);
Thread t=new Thread(opener);
t.start();
try
{
t.join(timeinsec);
}
catch(InterruptedException exception)
{
}
return opener.getSocket();
}
public SocketOpenerClass(String host,int port)
{
socket=null;
openerhost=host;
openerport=port;
}
public Socket getSocket()
{
return socket;
}
public void run()
{
try
{
socket=new Socket(openerhost,openerport);
}
catch(IOException ie)
{
}
}
}
```

Download this listing.

In the above code, the constructor of the SocketConnector class takes the ipaddress string, and portno and wait intergers as input parameters. The ipaddress string retrieves the ip address of the Jabber server to connect. The portno integer retrieves the port number of the Jabber server to open a socket and the wait integer retrieves the initial time to wait before connecting to the Jabber server.

The methods defined in Listing 6-5 are:

- `openPort()`: Opens a socket between the Jabber server and the Airline Reservation application.
- `writemessage()`: Creates an object of the XMLReader class to read airline schedules from the airlines.xml file. The writemessage() method calls the sendXMLToJabber() method and passes the matched airline schedules to the sendXMLToJabber() method.
- `sendXMLToJabber()`: Takes the matched airline schedule as an input parameter and sends the information to the Jabber server.
- `sendername()`: Retrieves the name of the sender from the input message and checks the identity of the sender.

- `run()`: Listens for the response sent by the Jabber server.
- `SocketOpenerClass()`: Initializes the object of the SocketOpenerClass class.
- `getSocket()`: Returns an object of the Socket class.

◄ PREVIOUS NEXT ►

Showing the Airline Schedules

The ReservationResultTable.java file shows the available airline schedules in the user interface of the Airline Reservation application. Listing 6-6 shows the contents of the ReservationResultTable.java file:

Listing 6-6: The ReservationResultTable.java File

```
/*
Imports required classes from javax.swing package
*/
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
/*
Imports required classes from javax.swing.table package.
*/
import javax.swing.table.*;
/*
Imports required classes from javax.swing.event package.
*/
import javax.swing.event.*;
/*
Imports required classes from javax.swing package.
*/
import javax.swing.*;
/*
Imports required classes from java.awt package.
*/
import java.awt.*;
/*
Imports required classes from java.awt.event package.
*/
import java.awt.event.*;
/*
class SocketClass - This class is used for reading the input messages and writing output messages.
Constructor:
SocketClass - This constructor calls a method of SocketOpener class to open a client socket.
Methods:
*/
public class ReservationResultTable extends JPanel
{
   /*
   Declare the object of the ReservationTableModel class.
   */
   ReservationTableModel tablemodel;
   /* Declare the object of the JTable class.*/
   JTable table;
   public ReservationResultTable()
   {
      super(new GridLayout(1,0));
      /*
      Initializes the object of ReservationTableModel class.
      */
      tablemodel=new ReservationTableModel();
      /*
      Initializes the object of JTable class.
      */
      table = new JTable(tablemodel);
      table.setPreferredScrollableViewportSize(new Dimension(600, 100));
      /* Declare and initialize the object of JScrollPane class.*/
      JScrollPane scrollPane = new JScrollPane(table);
      add(scrollPane);
      setPreferredSize(new Dimension(600, 100));
   }
   /*
   class ReservationTableModel - This class is used to insert data into table.
   Methods:
   getColumnCount - Returns number of column in the table.
   setData - Set data into table cell.
   getRowCount -gets number of rows in the table.
   isCellEditable - Set cell edit property.
   setValueAt - Set data into table cell.
   */
   class ReservationTableModel extends AbstractTableModel
   {
      private String[] columnNames = {"Choice","Flight No.",
      "Aircraft type",
      "Origin",
      "No. of Stops",
      "Destination",
      "Dep.Date-time",
      "Arr.Date-time",
      "Executive Class",
```

```
      "Economy Class",
   };
   private Object[][] data = {};
   /*
   getColumnCount - Returns number of column in the table.
   Parameter: N/A
   Return Value: int
   */
   public int getColumnCount()
   {
      return columnNames.length;
   }
   /*
   setData - Set data into table celles.
   Parameter: dataarr - array of Object Type.
   Return Value: int
   */
   public void setData(Object[][] dataarr)
   {
      data=dataarr;
   }
   /*
   getRowCount -gets number of rows in the table.
   Parameter: dataarr - array of Object Type.
   Return Value: int
   */
   public int getRowCount()
   {
      return data.length;
   }
   /*
   getColumnName - Returns Column name
   Parameter: col
   Return Value: String
   */
   public String getColumnName(int col)
   {
      return columnNames[col];
   }
   /*
   getValueAt - Returns value from a table cell.
   Parameter: row, col
   Return Value: Object
   */
   public Object getValueAt(int row, int col)
   {
      return data[row][col];
   }
   /*
   getColumnClass - Returns the Class of a table cell.
   Parameter: col
   Return Value: Class
   */
   public Class getColumnClass(int col)
   {
      return getValueAt(0, col).getClass();
   }
   /*
   isCellEditable - Set cell edit property.
   Parameter: row, col
   Return Value: boolean
   */
   public boolean isCellEditable(int row, int col)
   {
   if (col >=1&&col<=7)
   {
      return false;
   }
   else
   {
      return true;
   }
}
/*
setValueAt - Set data into table cell.
Parameter: value - Object of Object type, row, col
Return Value: N/A
*/
public void setValueAt(Object value, int row, int col)
{
   data[row][col] = value;
   fireTableCellUpdated(row, col);
}
}
}
```

Download this listing.

In the above code, the constructor of the ReservationResultTable class creates a new instance of the ReservationTableModel class. The ReservationTableModel class shows the matched airline schedules received from the Jabber server in a tabular format.

The methods defined in Listing 6-6 are:

- `getRowCount()`: Returns the total number of records present in the search result.
- `isCellEditable()`: Checks whether or not the information received from the Jabber server is editable.
- `setValueAt()`: Shows the matched airline schedules in the user interface of the Airline Reservation application.

The ReservationResultTable.java file shows the airline schedules in the main window of the Airline Reservation application, as shown in Figure 6-4:

Figure 6-4: Showing the Airline Schedules

A confirmation message appears when an end user clicks the Submit button, as shown in Figure 6-5:

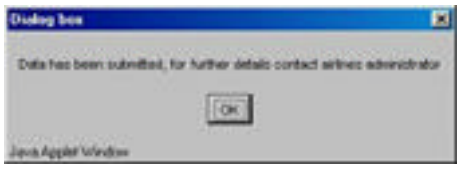


Figure 6-5: Confirmation Message

Reading XML Files

The XMLReader.java file reads the airlines.xml file that contains the information about available airline schedules. Listing 6-7 shows the contents of the XMLReader.java file:

Listing 6-7: The XMLReader.java File

```
/*Imports required File classes*/
import java.io.File;
/*Imports required IOException classes*/
import java.io.IOException;
/*Imports required PrintStream classes*/
import java.io.PrintStream;
/*Imports required Iterator classes*/
import java.util.Iterator;
/*Imports required List classes*/
import java.util.List;
/*Imports required Document classes*/
import org.jdom.Document;
/*Imports required Element classes*/
import org.jdom.Element;
/*Imports required JDOMException classes*/
import org.jdom.JDOMException;
/*Imports required SAXBuilder classes*/
import org.jdom.input.SAXBuilder;
/*Imports required XMLOutputter classes*/
import org.jdom.output.XMLOutputter;
/*
class XMLReader -This class is used to read XML file.
Methods:
readFile
*/
public class XMLReader
{
/*Declare object of String class.*/
String filename;
/*Declare object of Document class.*/
Document doc;
/*
readFile: This method call an openPort() method.
Parameter: fname - object of String class.
Return Value: List
*/
public List readFile(String fname)
{
   filename = fname;
   PrintStream out = System.out;
   SAXBuilder builder = new SAXBuilder();
   try
   {
      doc = builder.build(new File(filename));
   }
   catch(Exception je) {}
   Element root = doc.getRootElement();
   List childlist = root.getChildren("flightdescription");
   return childlist;
   }
}
```

Download this listing.

In the above listing, the XMLReader.java file contains the readFile() method. The readFile() method takes the location of the airlines.xml file as an input parameter and reads the file.

The airlines.xml File

The airlines.xml file contains the airline schedules in an XML format. Listing 6-8 shows the contents of the airlines.xml file:

Listing 6-8: The airlines.xml File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<airlines>
<flightdescription>
<id>
1
</id>
<flightno>
usa1000
</flightno>
<aircrafttype>
AB300
</aircrafttype>
<origin>Chicago</origin>
<noofstop>
0
</noofstop>
<destination>Boston</destination>
<departurdatetime>
<date>
20040829
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040829
</date>
<atime>
2145
</atime>
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
<flightdescription id="2">
<id>
2
</id>
<flightno>
usa10001
</flightno>
<aircrafttype>
AB3001
</aircrafttype>
<origin>Chicago</origin>
<noofstop>
0
</noofstop>
<destination>Boston</destination>
<departurdatetime>
<date>
20040829
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040829
</date>
<atime>
2145
</atime>
</arrivaldatetime>
```

```
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
<flightdescription id="3">
<id>
3
</id>
<flightno>
usa10301
</flightno>
<aircrafttype>
AB4001
</aircrafttype>
<origin>Boston</origin>
<noofstop>
0
</noofstop>
<destination>Chicago</destination>
<departurdatetime>
<date>
20040830
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040830
</date>
<atime>
2145
</atime>
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
<flightdescription id="4">
<id>
4
</id>
<flightno>
usa10301
</flightno>
<aircrafttype>
AB4001
</aircrafttype>
<origin>Chicago</origin>
<noofstop>
0
</noofstop>
<destination>Boston</destination>
<departurdatetime>
<date>
20040830
</date>
<dtime>
1945
</dtime>
</departurdatetime>
<arrivaldatetime>
<date>
20040830
</date>
```

```
<atime>
2145
</atime>
</arrivaldatetime>
<class>
<executive>
YES
</executive>
<executiveprice>
700
</executiveprice>
<economy>
YES
</economy>
<economyprice>
500
</economyprice>
</class>
</flightdescription>
</airlines>
```

Download this listing.

◄ PREVIOUS NEXT ►

Unit Testing

To test the Airline Reservation application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Download the free implementation of Java Document Object Model (JDOM) to read XML files from the following URL:

   http://www.jdom.org/

3. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

4. Set the classpath of the lib directory of J2SDK and the jdom.jar file by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;D:\jdom.jar;
   ```

5. Copy the ReservationStatus.java, SocketClass.java, SocketConnector.java, ReservationResultTable.java, and XMLReader.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder where you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

6. To run the Airline Reservation application, specify the following command at the command prompt:

   ```
   java SocketConnector
   ```

7. The user interface of the Personal Information page appears.

8. Enter information in the fields of the Personal Information page, as shown in Figure 6-6:

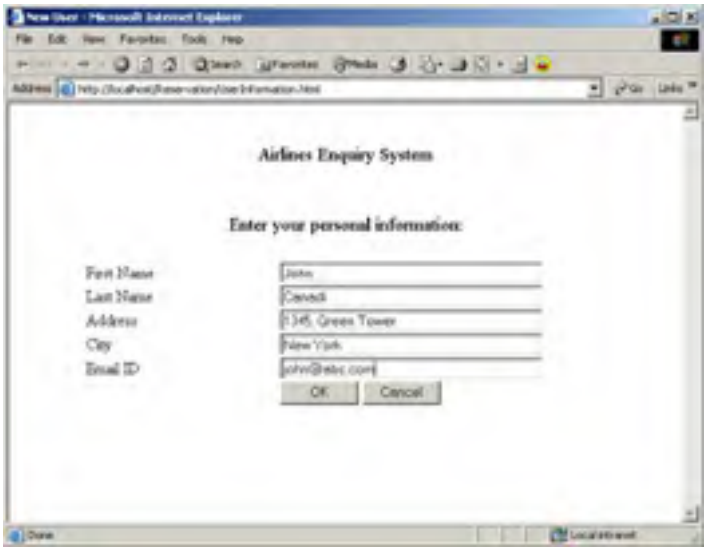


Figure 6-6: Entering Personal Information

9. Click OK to open the user interface of the Airline Reservation application.

10. Select Chicago in the From combo box and Boston in the To combo box. Specify the Departure Date and Nationality, as shown in Figure 6-7:

Figure 6-7: Searching for an Airline Schedule

11. Click the Search button to search for the specified airline schedule.
12. Select the flight number usa10001 and select the Economy check box to reserve the airline schedule, as shown in Figure 6-8:

Figure 6-8: Reserving an Airline Schedule

13. Click Submit to submit the reservation information. A confirmation dialog box appears.
14. Click OK to return to the Airline Reservation window.

Chapter 7: Creating a Contact List Application

Download CD Content

The Jabber protocol allows you to develop an application for creating and managing contact information by using the Jabber server. This chapter describes how to develop the Contact List application, which allows end users to manage contact information. By using the Contact List application, end users can add information to the contact list and check whether or not other end users are online. An end user can send messages to other end users added to the contact list.

Architecture of the Contact List Application

The Contact List application uses the following files:

- UserLogin.java: Creates the Login window for the Contact List application that allows end users to establish a connection with the Jabber server to communicate with other end users. This file allows end users to login as existing end users or register for a new account.
- ContactList.java: Shows a tree structure of the contact list created by end users. This file also allows end users to manage their contact list by adding, editing, or deleting contact information.
- UserInfo.java: Allows end users to view the information about other end users in the contact list.
- SocketClass.java: Opens a socket to send and receive messages through the Jabber server.
- EditDelete.java: Allows end users to edit or delete contact information from the contact list.
- DynamicTree.java: Creates and maintains a tree structure of the contact list. The tree structure is visible in the user interface of the Contact List application.
- ChatWindow.java: Creates a Chat window that allows end users to send messages to other end users present in the contact list.
- AddUser.java: Creates a user interface that allows end users to add other end users to the contact list.

Figure 7-1 shows the architecture of the Contact List application:

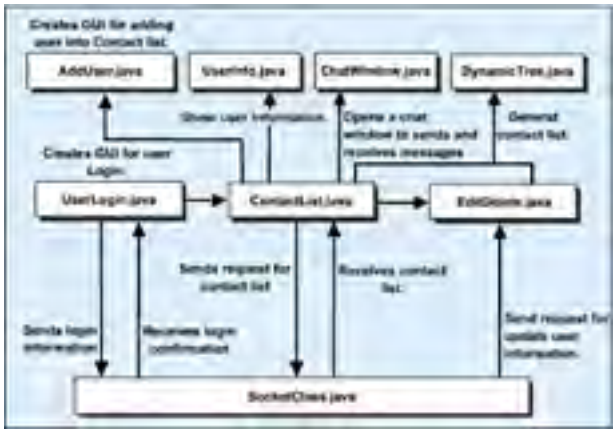


Figure 7-1: Architecture of the Contact List Application

The UserLogin.java file creates the Login window of the Contact List application that contains various labels, text boxes, and the Submit button. The UserLogin.java file allows end users to login as existing users or register for a new account.

When an end user enters the login information and clicks the Submit button on the Login window, the UserLogin.java file calls the ContactList.java file to create the user interface for the Contact List application. The ContactList.java file allows end users to add other end users and edit or delete the existing end users from the contact list. The ContactList.java file shows a tree structure of the contact list.

If an end user selects File-> Add User, the ContactList.java file calls the AddUser.java file to add other end users to the contact list. The AddUser.java file provides an interface with various labels, three text boxes, and two buttons, Add and Cancel.

If an end user selects File-> Manage User, the ContactList.java file calls the EditDelete.java file to edit or delete the existing end users from the contact list. The EditDelete.java file provides an interface containing two panes. The left pane shows a tree structure of the contact list and the right pane contains various labels, two text boxes, and three buttons: Up, Delete, and Cancel.

If an end user selects a specific end user name from the tree structure of the contact list, the ContactList.java file calls the ChatWindow.java file that allows end users to send messages to the selected end users. The ChatWindow.java file provides an interface with text area to view the text messages and the Send button to send the text messages.

The ChatWindow.java file calls the SocketClass.java file that opens a socket to send and receive messages with the help of the Jabber server.

When an end user right-clicks any user name in the contact list, the pop-up option, View Contact appears. If the end user selects the View Contact option, the ContactList.java file calls the UserInfo.java file to display the contact information of the selected end user.

PREVIOUS NEXT

Creating the Login Window

The UserLogin.java file creates the Login window for the Contact List application. Listing 7-1 shows the contents of the UserLogin.java file:

Listing 7-1: The UserLogin.java File

```
/*Import required swing classes*/
import javax.swing.*;
import javax.swing.text.*;
/*Import required awt classes*/
import java.awt.*;
import java.awt.event.*;
/*Import required io and net classes*/
import java.io.*;
import java.net.*;
/*
class UserLogin - This class is used to create a GUI for user login
Constructor:
UserLogin - This constructor creates GUI.
Methods:
actionPerformed
*/
public class UserLogin extends JFrame implements ActionListener
{
   /*Declares object of Container class.*/
   Container contentpane;
   /*Declares object of JTextField class.*/
   JTextField nametextfield;
   JTextField servernametextfield;
   /*Declares object of JPasswordField class.*/
   JPasswordField passwordtextfield;
   /*Declares object of JButton class.*/
   JButton submitbutton;
   /*Declares object of Socket class.*/
   Socket clientsocket;
   /*Declares object of String class.*/
   String servername;
   String username;
   String password;
   String resource;
   String registrationstring="";
   /*Declares object of JPanel class.*/
   JPanel note;
   JPanel combine;
   /*Declares object of SocketClass class.*/
   SocketClass socketclass;
   boolean submitclickd=false;
   public UserLogin ()
   {
      super("Login Window");
      this.clientsocket=clientsocket;
      contentpane=getContentPane();
      /*Initializes the objects of the JTextField class.*/
      nametextfield=new JTextField();
      servernametextfield=new JTextField();
      /*Initializes the object of the JPasswordField class.*/
      passwordtextfield=new JPasswordField();
      /*Initializes the object of the JPasswordField class.*/
      submitbutton=new JButton("Submit");
      /*Initializes the object of the JPanel class.*/
      combine=new JPanel();
      /*Initializes the object of JPanel class.*/
      note=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
      /*Initializes the object of the SocketClass class.*/
      socketclass=new SocketClass();
      /*Sets action command for submitbutton.*/
      submitbutton.setActionCommand("submit");
      /*Adds action listener for submitbutton.*/
      submitbutton.addActionListener(this);
      submitbutton.setMnemonic('s');
      resource="Home";
      /*Declares and initializes the object of JPanel class.*/
      JPanel controlpanel=new JPanel();
      /*Declares and initializes the objects of JLabel class.*/
      JLabel room= new JLabel("User Name");
      JLabel name= new JLabel("Password ");
      JLabel serverlabel= new JLabel("Server IP");
      JLabel firstnode= new JLabel("* If you are not a registerd user your account will be ");
      JLabel secondnode= new JLabel("created automatically.");
      /*Declares and initializes the object of JPanel class.*/
      JPanel lable=new JPanel();
      /*Sets GridLayout of controlpanel.*/
      controlpanel.setLayout(new GridLayout(4, 2, 3, 3));
```

```
        controlpanel.setBorder(BorderFactory.createTitledBorder("Login Information"));
        /*Adds labels and textfields to controlpanel.*/
        controlpanel.add(room);
        controlpanel.add(nametextfield);
        controlpanel.add(name);
        controlpanel.add(passwordtextfield);
        controlpanel.add(serverlabel);
        controlpanel.add(servernametextfield);
        firstnode.setForeground(Color.red);
        secondnode.setForeground(Color.red);
        /*Adds labels to note.*/
        note.add(firstnode);
        note.add(secondnode);
        /*Declares and initializes the object of JLabel class.*/
        JLabel heading=new JLabel("Login Window", JLabel.CENTER);
        lable.add(heading,BorderLayout.NORTH);
        heading.setFont(new Font("verdana", 1, 12));
        contentpane.add(lable,BorderLayout.NORTH);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        contentpane.add(controlpanel,BorderLayout.CENTER);
        /*Declares and initializes the objects of JPanel class.*/
        JPanel buttonpanel=new JPanel(new GridLayout(2, 1, 5, 5));
        JPanel bp=new JPanel(new FlowLayout());
        bp.add(submitbutton);
        buttonpanel.add(note);
        buttonpanel.add(bp);
        contentpane.add(buttonpanel, BorderLayout.SOUTH);
        setBounds(5, 5, 300, 240);
        submitbutton.addActionListener(this);
        show();
    }
    public void actionPerformed(ActionEvent ae)
    {
        String actioncommand=ae.getActionCommand();
        servername=servernametextfield.getText().trim();
        username=nametextfield.getText().trim();
        password=passwordtextfield.getText().trim();
        /*This block will be executed when an end user clicks the submit button.*/
        if (actioncommand.equals("submit"))
        {
            if (!submitclickd)
            {
                submitclickd=true;
                if (!socketclass.isConnected())
                {
                    socketclass.socketoOpenClass(servername, 5222, 1000);
                }
                socketclass.setUserNameAndPassword(username,password);
                if (socketclass.isConnected()&&(!socketclass.isWaitingForAuth()))
                {
                    socketclass.sendSessionStartMessage();
                    socketclass.sendAuthorized();
                    socketclass.setWaitForAuth(true);
                    socketclass.setUserLoginHandler(this);
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Server is not
                    found.", "wrring", JOptionPane.PLAIN_MESSAGE);
                }
        }
        else
        {
            submitclickd=false;
        }
    }
}
public static void main(String[] args)
{
try
{
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
}
catch(Exception e)
{
    e.printStackTrace();
}
/*Declares and initializes the objects of UserLogin class.*/
UserLogin userlogin=new UserLogin();
}
}
```

Download this listing.

In the above code, the main() method creates an instance of the UserLogin class. The UserLogin class defines the actionPerformed() method. This method acts as an event listener and activates an appropriate method based on the action an end user performs.

The UserLogin.java file creates the Login window of the Contact List application, as shown in Figure 7-2:

Figure 7-2: The Login Window

◄ PREVIOUS NEXT ►

Creating the User Interface of the Contact List Application

The ContactList.java file creates the user interface for the Contact List application. Listing 7-2 shows the contents of the ContactList.java file:

Listing 7-2: The ContactList.java File

```
/*Imports required swing classes*/
import javax.swing.*;
/*Imports required awt classes*/
import java.awt.*;
/*Imports required tree classes*/
import javax.swing.tree.*;
/*Imports required event classes*/
import javax.swing.event.*;
import java.awt.event.*;
/*Imports required util classes*/
import java.util.Hashtable;
import java.util.Enumeration;
/*
class ContactList - This class is the main class for this application. It is used to create a tree
and a menubar.
Constructor:
ContactList - This constructor creates GUI.
Methods:
writemessage: This method is called to send input message to insertTextInToTextPane function.
eraseText: This method is called to send input message to insertTextInToTextPane function.
addToRosterTree: This method is called to add user name in to tree.
*/
public class  ContactList extends JFrame implements ActionListener, MouseListener
{
   /*Declares object of Container class.*/
   Container container = null;
   /*Declares objects of JMenuItem class.*/
   JMenuItem adduser=null;
   JMenuItem manageuser = null;
   /*Declares objects of DynamicTree class.*/
   DynamicTree tree;
   /*Declares objects of DynamicTree class.*/
   ContactListPopup popup=null;
   /*Declares objects of JPanel class.*/
   JPanel popuppanel=null;
   /*Declares objects of JLabel class.*/
   JLabel display=null;
   /*Declares objects of SocketClass class.*/
   SocketClass socketclass;
   /*Declares objects of JTree class.*/
   JTree rostertree;
   /*Declares objects of String class.*/
   String selecteditemvalue;
   /*Declares objects of Hashtable class.*/
   Hashtable hashnode;
   Hashtable chatwindowhash;
   /*Declares objects of AddUser class.*/
   AddUser add;
   boolean nouserexistsinlist;
   public ContactList(SocketClass socketclass)
   {
      super("Contact List");
      container = this.getContentPane();
      this.socketclass=socketclass;
      /*Initializes object of the JPanel class.*/
      popuppanel = new JPanel();
      /*Initializes objects of the Hashtable class.*/
      hashnode=new Hashtable();
      chatwindowhash=new Hashtable();
      /*Initializes objects of Hashtable class.*/
      display = new JLabel(" ");
      display.setOpaque(true);
      popuppanel.add(display);
      /*Sets window's default closing operation.*/
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      popuppanel.addMouseListener(this);
      display.addMouseListener(this);
      /*Initializes objects of JPanel class.*/
      popuppanel =  new JPanel(false);
      /*Declares and initializes objects of the JMenuBar class.*/
      JMenuBar menubar = new JMenuBar();
      /*Declares and initializes objects of the JMenu class.*/
      JMenu filemenu = new JMenu("File");
      /*Declares and initializes objects of the JMenuItem class.*/
      adduser= new JMenuItem("Add User");
      manageuser= new JMenuItem("Manage User");
      /*Adds filemenu to the menubar.*/
```

```
   menubar.add(filemenu);
   setJMenuBar(menubar);
   /*Adds menuitems to the filemenu.*/
   filemenu.add(adduser);
   filemenu.add(manageuser);
   /*Adds action listener to menu items.*/
   adduser.addActionListener(this);
   manageuser.addActionListener(this);
   /*Initializes objects of ContactListPopup class.*/
   popup = new ContactListPopup(this);
   /*Declares and initializes objects of DefaultMutableTreeNode class.*/
   DefaultMutableTreeNode root = new  DefaultMutableTreeNode("My Contact List");
   /*Initializes objects of DynamicTree class.*/
   tree=new DynamicTree(root);
   rostertree=tree.getTreeHandler();
   /*Adds action listener to the rostertree.*/
   rostertree.addMouseListener(this);
   /*Declares and initializes objects of JScrollPane class.*/
   JScrollPane scrollpane = new JScrollPane(tree);
   /*Adds mouse listener to the scrollpane.*/
   scrollpane.addMouseListener(this);
   /*Adds scrollpane to the container.*/
   container.add(scrollpane);
   setSize(300, 400);
   DefaultMutableTreeNode nousertreenode=new DefaultMutableTreeNode("No user exists in your conta
   list");
   tree.addObject(null,nousertreenode);
   nouserexistsinlist=true;
   show();
}
/*
writemessage: This method is called to send input message to insertTextInToTextPane function.
Parameter: pmessage - Object of  String class, psendername - Object of  String class,
preceivername - Object of  String class
Return Value: N/A
*/
public void writemessage(String pmessage,String psendername,String preceivername)
{
   String sendernameto=psendername;
   String[] splitedsendername=psendername.split("/");
   psendername="'"+splitedsendername[0]+"'";
   if (chatwindowhash.get(psendername)!=null)
   {
      ((ChatWindow)chatwindowhash.get(psendername)).insertTextInToTextPane(pmessage,sendernameto)
   }
   else
   {
      String[] nickname=psendername.split("@");
      ChatWindow chatwindow=new ChatWindow(nickname[0].substring(1, nickname[0].length()),
      psendername,socketclass);
      chatwindow.insertTextInToTextPane(pmessage,psendername);
      chatwindowhash.put(psendername,chatwindow);
   }
}
/*
eraseText: This method is called to send input message to insertTextInToTextPane function.
Parameter: pmessage - Object of  String class, psendername - Object of  String class,
preceivername - Object of  String class
Return Value: N/A
*/
public void eraseText(String sendername1)
{
   String sendername2=sendername1;
   String[] splitedsendername=sendername1.split("/");
   sendername1="'"+splitedsendername[0]+"'";
   if (chatwindowhash.get(sendername1)!=null)
   {
      ((ChatWindow)chatwindowhash.get(sendername1)).receivemessagetextarea.setText("");
      ((ChatWindow)chatwindowhash.get(sendername1)).setVisible(false);
   }
}
/*
addToRosterTree: This method is called to add user name in to tree.
Parameter: jid - Object of  String class, contactpersonname - Object of  String class.
Return Value: N/A
*/
public void addToRosterTree(String jid,String contactpersonname)
{
if (nouserexistsinlist)
{
   tree.clear();
}
else
{
   nouserexistsinlist=false;
}
contactpersonname=contactpersonname.substring(1,contactpersonname.length()-1);
if (!hashnode.containsKey(contactpersonname))
```

```
    {
       hashnode.put(contactpersonname,jid);
       DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(contactpersonname);
       tree.addObject(null,treenode);
    }
}
/*
removeFromRosterTree: This method is called to remove user from tree.
Parameter: username - Object of  String class.
Return Value: N/A
*/
public void removeFromRosterTree(String username)
{
   String[] splitedusername=username.split("/");
   String keyval=add.getNickName();
   String removestring;
   removestring="<iq type='set' id='1'>";
   removestring=removestring+"<query xmlns='jabber:iq:roster'>";
   removestring=removestring+"<item jid='"+splitedusername[0]+"' name='"+keyval+"'
   subscription='remove'/>";
   removestring=removestring+"</query></iq>";
   socketclass.sendXMLToJabber(removestring);
   hashnode.remove(keyval);
   Enumeration enum=hashnode.keys();
   tree.clear();
   while (enum.hasMoreElements())
   {
      DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(enum.nextElement());
      tree.addObject(treenode);
   }
}
/*
checkHash: This method is called to check that a user whose user name is in parameter one is exists
in hashtable or not.
Parameter: username - Object of String class.nname - Object of  String class.
Return Value: N/A
*/
public boolean checkHash(String uname,String nname)
{
   String tempuser="'"+uname+"'";
   String tempnname=nname;
   if (hashnode.containsKey(nname))
   {
      return true;
   }
   if (hashnode.containsValue(tempuser))
   {
      return true;
   }
   return false;
}

public void mousePressed(MouseEvent e)
{
   popup.setVisible(false);
}
public void mouseReleased(MouseEvent e)
{
   int selRow = rostertree.getRowForLocation(e.getX(), e.getY());
   TreePath selPath = rostertree.getPathForLocation(e.getX(), e.getY());
   TreePath path=rostertree.getSelectionPath();
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)rostertree.getLastSelectedPathComponent();
   if(node!=null)
   {
      if(!(node.toString().equals("My Contact List")))
      {
         if (e.getButton() == MouseEvent.BUTTON1)
         {
            if(selRow != -1)
            {
               if(e.getClickCount() == 2)
               {
                  selecteditemvalue=(String)hashnode.get(node.toString());
                  if (!chatwindowhash.containsKey(selecteditemvalue))
                  {
                     ChatWindow chatwindow=new ChatWindow(node.toString(), selecteditemvalue,
                     socketclass);
                     chatwindowhash.put(selecteditemvalue, chatwindow);
                  }
                  else
                  {
                     ((ChatWindow)chatwindowhash.get(selecteditemvalue)).show();
                  }
               }
            }
         }
         else if (e.getButton() == MouseEvent.BUTTON3)
         {
```

```
            if(selRow != -1)
            {
               if(e.getClickCount() ==1)
               {
                  popup.setVisible(false);
                  selecteditemvalue=(String)hashnode.get(node.toString());
                  popup = new ContactListPopup(this);
                  Point point=this.getLocation();
                  popup.setLocation(e.getX()+(int)point.getX(),e.getY()+(int)point.getY()+30);
                  popup.setVisible(true);
               }
            }
         }
      }
   }
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void actionPerformed(ActionEvent ae)
{
   if(ae.getSource() == adduser)
   {
      add =new AddUser(socketclass);
      add.show();
   }
   if(ae.getSource() == manageuser)
   {
      EditDelete editdelete = new EditDelete(hashnode,socketclass,tree);
      editdelete.show();
   }
}
/*
class ContactListPopup - This class is used to create a popup menu.
Constructor:
ContactListPopup-This constructor creates popup menu.
Methods:
actionPerformed
*/
public class ContactListPopup extends JPopupMenu implements ActionListener
{
   JMenuItem userinfomenuitem;
   ContactList useRightButton;
   String jabberid;
   public void setNodeName(String nodename)
   {
      jabberid=nodename;
   }
   public ContactListPopup(ContactList urb)
   {
      useRightButton = urb;
      userinfomenuitem = new JMenuItem("View Contact");
      userinfomenuitem.addActionListener(this);
      add(userinfomenuitem);
   }
   public void actionPerformed(ActionEvent e)
   {
      if (e.getSource() == userinfomenuitem)
      {
         String vacrstring="";
         vacrstring="<iq to="+useRightButton.selecteditemvalue+" type='get' id='1'>";
         vacrstring=vacrstring+"<vCard xmlns='vcard-temp'/></iq>";
         socketclass.sendXMLToJabber(vacrstring);
         popup.setVisible(false);
      }
   }
}
}
```

Download this listing.

In the above code, the constructor of the ContactList class takes an object of the SocketClass class as an input parameter. The object allows an end user to invoke methods of the SocketClass class.

The methods defined in Listing 7-2 are:

- `writemessage()`: Inserts the text message specified by an end user in the text area of the interface.
- `eraseText()`: Refreshes the text area every time an end user selects another end user's name from the contact list.
- `addToRosterTree()`: Adds other end users to the contact list.
- `removeFromRosterTree()`: Removes the existing end users from the contact list.
- `checkHash()`: Checks whether or not the specified end user already exists in the contact list.

- `ContactListPopup()`: Creates a popup menu to view the contact information of the selected end user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The ContactList.java file creates the main window of the Contact List application, as shown in Figure 7-3:

Figure 7-3: User Interface of the Contact List Application

Select the File menu to view the File menu options.

Figure 7-4 shows the File menu options of the Contact List application:

Figure 7-4: The File Menu of the Contact List Application

Select File-> Add User to add other end users to the contact list and send a welcome message to the added users.

Select File-> Manage User to edit or delete existing end users from the contact list.

Select any end user from the contact list to send messages to the selected user.

◄ PREVIOUS NEXT ►

Adding a Contact

The AddUser.java file creates the user interface to add other end users to the contact list and send a welcome message to the added end user. Listing 7-3 shows the contents of the AddUser.java file:

Listing 7-3: The AddUser.java File

```
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required javax.swing.event classes*/
import javax.swing.event.*;
/*Imports required java.awt classes*/
import java.awt.*;
/*mports required java.awt.event classes*/
import java.awt.event.*;
/*
class AddUser - This class is used to create a GUI for add a new user in user roster.
Constructor:
AddUser-This constructor creates GUI.
Methods:
getNickName: This method is called retrieve user nick name.
*/
public class  AddUser extends JFrame implements ActionListener
{
/*Declares object of Container class.*/
Container container;
/*Declares object of GridBagLayout class.*/
GridBagLayout gridbaglayout;
/*Declares object of JLabel class.*/
JLabel useridlabel;
JLabel nicknamelabel;
/*Declares object of JTextField class.*/
JTextField useridtext;
JTextField nicknametext;
/*Declares object of JTextPane class.*/
JTextPane messagetextarea;
/*Declares object of JTextPane class.*/
JButton addbutton;
JButton cancelbutton;
/*Declares object of SocketClass class.*/
SocketClass addsocketclass;
/*Declares object of String class.*/
String uid;
String nickname;
/*Declares object of JScrollPane class.*/
JScrollPane scpane;
public AddUser(SocketClass socketclass)
{
   super("Add Contact");
   container = this.getContentPane();
   addsocketclass=socketclass;
   /*Declares and initializes the objects of JLabel class.*/
   JPanel mainpane=new JPanel(new GridLayout(2,1));
   /*Declares and initializes the objects of JPanel class.*/
   JPanel labeltextpanel=new JPanel();
   container.add(new JLabel(" "),BorderLayout.NORTH);
   container.add(new JLabel(" "),BorderLayout.WEST);
   /*Declares and initializes the objects of GridLayout class.*/
   GridLayout labeltextgridlayout=new GridLayout(2, 2, 20, 0);
   /*Sets the layout of the labeltextpanel.*/
   labeltextpanel.setLayout(labeltextgridlayout);
   /*Initializes the objects of JLabel class.*/
   useridlabel=new JLabel("User ID");
   /*Adds the useridlabel to the labeltextpanel.*/
   labeltextpanel.add(useridlabel);
   /*Initializes the objects of JTextField class.*/
   useridtext=new JTextField();
   /*Sets the size of the useridtext.*/
   useridtext.setPreferredSize(new Dimension(150, 20));
   labeltextpanel.add(useridtext);
   /*Initializes the objects of JLabel class.*/
   nicknamelabel=new JLabel("User Name");
   labeltextpanel.add(nicknamelabel);
   /*Initializes the objects of JTextField class.*/
   nicknametext=new JTextField();
   /*Sets the size of the nicknametext.*/
   nicknametext.setPreferredSize(new Dimension(150, 20));
   labeltextpanel.add(nicknametext);
   /*Declares and initializes the objects of JPanel class.*/
   JPanel gridlayoutpanel=new JPanel(new GridLayout(1, 2, 0, 0));
   gridlayoutpanel.add(new JLabel(" "));
   /*initializes the objects of JPanel class.*/
   JPanel buttonpanel=new JPanel();
   buttonpanel.setPreferredSize(new Dimension(250, 30));
```

```
        addbutton=new JButton("Add");
        addbutton.addActionListener(this);
        addbutton.setPreferredSize(new Dimension(80, 25));
        addbutton.setActionCommand("add");
        cancelbutton=new JButton("Cancel");
        cancelbutton.setPreferredSize(new Dimension(80, 25));
        cancelbutton.addActionListener(this);
        cancelbutton.setActionCommand("cancel");
        buttonpanel.add(addbutton);
        buttonpanel.add(cancelbutton);
        gridlayoutpanel.add(buttonpanel);
        gridlayoutpanel.add(labeltextpanel);
        /*Declares and initializes the objects of JPanel class. */
        JPanel textareapanel=new JPanel(new GridLayout(1, 1, 20, 10));
        textareapanel.add(new JLabel("Message"));
        /*initializes the objects of JTextPane class. */
        messagetextarea=new JTextPane();
        /*initializes the objects of JScrollPane class.*/
        scpane=new JScrollPane(messagetextarea);
        /*Set the size of scpane.*/
        scpane.setPreferredSize(new Dimension(100, 30));
        textareapanel.add(scpane);
        mainpane.add(labeltextpanel);
        mainpane.add(textareapanel);
        container.add(mainpane);
        container.add(gridlayoutpanel, BorderLayout.SOUTH);
        setSize(420, 180);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String actionsource=ae.getActionCommand();
        if (actionsource.equals("add"))
        {
            String errmessage="";
            String uname=useridtext.getText().trim();
            String nname=nicknametext.getText().trim();
            if (uname.equals(""))
            {
                errmessage=" Jabber ID is a required field.\n";
            }
            else if (uname.indexOf("@")==-1)
            {
                errmessage=" Please enter a valid Jabber ID.\n";
            }
            if (nname.equals(""))
            {
                errmessage=errmessage+" User Name is a required field.";
            }
            if (!errmessage.equals(""))
            {
                JOptionPane.showMessageDialog(null,errmessage,"Error",JOptionPane.PLAIN_MESSAGE);
            }
            else
            {
                boolean checkforuser;
                checkforuser=addsocketclass.checkForRoster(uname,nname);
                if (!checkforuser)
                {
                    String msgstring="";
                    msgstring="<message type='chat' to='"+uname+"/Home'
                    from='"+addsocketclass.getUserName()+"@"+addsocketclass.getServerName()+"/Home'
                    ><body>";
                    msgstring=msgstring+messagetextarea.getText().trim();
                    msgstring=msgstring+"</body></message>";
                    addsocketclass.sendXMLToJabber(msgstring);
                    if (!addsocketclass.getUserInfo())
                    {
                        String addrosterstring="";
                        String servername=addsocketclass.getServerName();
                        nickname=nname;
                        uid=uname;
                        addrosterstring="<iq type='set'>";
                        addrosterstring=addrosterstring+"<query xmlns='jabber:iq:roster'>";
                        addrosterstring=addrosterstring+"<item jid='"+uname+"' subscription='to'
                        name='"+nname+"'/>";
                        addrosterstring=addrosterstring+"</query></iq>";
                        addsocketclass.sendXMLToJabber(addrosterstring);
                        addsocketclass.sendRosterRequest();
                        setVisible(false);
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null,"No User exist for this Jabber
                        ID.","Error",JOptionPane.PLAIN_MESSAGE);
                        addsocketclass.setUserInfo(false);
                    }
                }
```

```
            else
            {
               JOptionPane.showMessageDialog(null, "This Jabber ID or User Name has already present in
               your
               roster.", "Error", JOptionPane.PLAIN_MESSAGE);
            }
         }
      }
      else
      {
         JOptionPane.showMessageDialog(null, "errmessage", "Error", JOptionPane.PLAIN_MESSAGE);
         this.hide();
      }
}
/*
getNickName: This method is called retrieve user nick name.
Parameter: N/A
Return Value: String
*/
public String getNickName()
{
   return nickname;
}
}
```

Download this listing.

In the above code, the constructor of the AddUser class takes the object of the SocketClass class as an input parameter. This allows end users to invoke the methods of the SocketClass class.

The methods defined in Listing 7-3 are:

- `getNickName()`: Retrieves the user name of the added user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The AddUser.java file creates the window to add other end users to the contact list, as shown in Figure 7-5:

Figure 7-5: Adding End Users

PREVIOUS NEXT

Managing the Contact List

The EditDelete.java file creates the user interface to edit or delete existing end users from the contact list. Listing 7-4 shows the contents of the EditDelete.java file:

Listing 7-4: The EditDelete.java File

```
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required javax.swing.tree classes*/
import javax.swing.tree.*;
/*Imports required javax.swing.event classes*/
import javax.swing.event.*;
/*Imports required java.awt classes*/
import java.awt.*;
/*Imports required java.awt.event classes*/
import java.awt.event.*;
/* Imports required java.util classes*/
import java.util.*;
/*
class EditDelete - This class is used to create a GUI for maintaining Contact list.
Constructor:
EditDelete-This constructor creates GUI.
*/
public class  EditDelete extends JFrame implements ActionListener, MouseListener
{
/*Declares object of JPanel class.*/
JPanel combine;
/*Declares object of Container class.*/
Container container;
/*Declares object of GridBagLayout class.*/
GridBagLayout gridbaglayout;
/*Declares objects of JLabel class.*/
JLabel useridlabel = null;
JLabel nicknamelabel = null;
/*Declares objects of JTextField class.*/
JTextField useridtext = null;
JTextField nicknametext = null;
/*Declares objects of JButton class.*/
JButton editbutton = null;
JButton deletebutton = null;
JButton cancelbutton = null;
/*Declares objects of JPanel class.*/
JPanel buttonpanel = new JPanel();
/*Declares objects of JScrollPane class.*/
JScrollPane scpane = null;
JScrollPane treeView;
/*Declares object of JSplitPane class.*/
JSplitPane splitPane;
/*Declares object of Hashtable class.*/
Hashtable edithash;
/*Declares objects of String class.*/
String selecteduserid="";
String selectednickname="";
/*Declares object of JTree class.*/
JTree edittree;
/*Declares object of TreePath class.*/
TreePath path;
/*Declares object of DefaultMutableTreeNode class.*/
DefaultMutableTreeNode root ;
/*Declares object of DefaultTreeModel class.*/
DefaultTreeModel treeModel;
/*Declares objects of DynamicTree class.*/
DynamicTree tree;
DynamicTree dynamictree;
/*Declares object of SocketClass class.*/
SocketClass editsocketclass;
public EditDelete(Hashtable editdeletehash,SocketClass socketclass,DynamicTree dtree)
{
super("Edit/Delete Contact");
container = this.getContentPane();
edithash=editdeletehash;
dynamictree=dtree;
editsocketclass=socketclass;
/*Initializes the object of the JSplitPane class.*/
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
/*Initializes the object of the JPanel class.*/
combine=new JPanel(new BorderLayout());
/*Initializes the object of the DefaultMutableTreeNode class.*/
root = new  DefaultMutableTreeNode("My Contact List");
/*Initializes the object of the DynamicTree class.*/
tree=new DynamicTree(root);
edittree=tree.getTreeHandler();
/*Initializes the objects of the DefaultTreeModel class.*/
```

```
treeModel = new DefaultTreeModel(root);
/*Declares and initializes the object of the Enumeration class.*/
Enumeration editenumkey=edithash.keys();
while (editenumkey.hasMoreElements())
{
   DefaultMutableTreeNode childnode = new  DefaultMutableTreeNode(editenumkey.nextElement());
   tree.addObject(childnode);
}
edittree.addMouseListener(this);
splitPane.setDividerLocation(180);
treeView = new JScrollPane(tree);
splitPane.setTopComponent(treeView);
splitPane.setBottomComponent(combine);
/*Declares and initializes the object of the JPanel class.*/
JPanel jpanel = new JPanel();
/*Declares and initializes the object of the JLabel class.*/
JLabel labelheading =new JLabel("Edit/Delete Contact");
/*Sets the font of the labelheading.*/
labelheading.setFont(new Font("Verdana", Font.BOLD,12));
/*Adds labelheading to the jpanel*/
jpanel.add(labelheading);
/*Adds jpanel to the combine*/
combine.add(jpanel,BorderLayout.NORTH);
gridbaglayout=new GridBagLayout();
/*Declare and initializes the objects of the JPanel class.*/
JPanel userinfopanel = new JPanel(gridbaglayout);
/*Declare and initializes the objects of the GridBagConstraints class.*/
GridBagConstraints gridbagconstraints=new GridBagConstraints();
useridlabel = new JLabel("User Id");
nicknamelabel = new JLabel("User Name");
/*Initializes the objects of the JTextField class.*/
useridtext = new JTextField();
nicknametext = new JTextField();
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(useridlabel, gridbagconstraints);
/*Sets the size of the useridlabel.*/
useridlabel.setPreferredSize(new Dimension(80, 28));
/*Adds useridlabel to the userinfopanel.*/
userinfopanel.add(useridlabel);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(useridtext, gridbagconstraints);
/*Sets the size of the useridtext.*/
useridtext.setPreferredSize(new Dimension(80, 28));
/*Adds useridtext to the userinfopanel.*/
userinfopanel.add(useridtext);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=1;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(nicknamelabel,gridbagconstraints);
/*Sets the size of the nicknamelabel.*/
nicknamelabel.setPreferredSize(new Dimension(120, 28));
/*Adds nicknamelabel to the userinfopanel.*/
userinfopanel.add(nicknamelabel);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=1;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(nicknametext,gridbagconstraints);
/*Sets the size of the nicknametext.*/
nicknametext.setPreferredSize(new Dimension(120, 28));
/*Adds nicknametext to the userinfopanel.*/
userinfopanel.add(nicknametext);
/*Declare and initializes the objects of the JLabel class. */
```

```
JLabel blanklabel=new JLabel(" ");
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=2;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(blanklabel,gridbagconstraints);
/*Sets the size of the blanklabel.*/
blanklabel.setPreferredSize(new Dimension(120, 28));
/*Adds blanklabel to the userinfopanel.*/
userinfopanel.add(blanklabel);
/*Declare and initializes the objects of the JLabel class.*/
JLabel secondblanklabel=new JLabel(" ");
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=3;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(secondblanklabel,gridbagconstraints);
/*Sets the size of the secondblanklabel.*/
secondblanklabel.setPreferredSize(new Dimension(120, 28));
/*Adds secondblanklabel to the userinfopanel.*/
userinfopanel.add(secondblanklabel);
/*Declare and initializes the objects of the JLabel class.*/
JLabel thirdblanklabel=new JLabel(" ");
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=3;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(thirdblanklabel,gridbagconstraints);
thirdblanklabel.setPreferredSize(new Dimension(120, 28));
userinfopanel.add(thirdblanklabel);
/*Initializes the objects of the JButton class.*/
editbutton = new JButton("Update");
deletebutton = new JButton("Delete");
cancelbutton = new JButton("Cancel");
/*Adds the action listener to the objects of JButton class.*/
editbutton.addActionListener(this);
deletebutton.addActionListener(this);
cancelbutton.addActionListener(this);
/*Sets the action command to the objects of JButton class.*/
editbutton.setActionCommand("edit");
deletebutton.setActionCommand("delete");
cancelbutton.setActionCommand("cancel");
/*Sets the size of the objects of JButton class*/
editbutton.setPreferredSize(new Dimension(60, 22));
deletebutton.setPreferredSize(new Dimension(70, 22));
cancelbutton.setPreferredSize(new Dimension(70, 22));
/*Adds objects of JButton class to the buttonpanel.*/
buttonpanel.add(editbutton);
buttonpanel.add(deletebutton);
buttonpanel.add(cancelbutton);
/*Sets the constraints for the gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=5;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(buttonpanel, gridbagconstraints);
userinfopanel.add(buttonpanel);
combine.add(userinfopanel);
container.add(splitPane);
setSize(495,230);
setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
   String actioncommand=ae.getActionCommand();
   /* This block will be executed when the end user clicks edit button.*/
   if (actioncommand.equals("edit"))
   {
      if ((!useridtext.getText().trim().equals(""))&&
      (!nicknametext.getText().trim().equals("")))
      {
         if ((!useridtext.getText().trim().equals(selecteduserid))&&
         (!nicknametext.getText().trim().equals(selectednickname)))
         {
            String editstring="<iq type='set' id='1'>";
```

```
            editstring=editstring+"<query xmlns='jabber:iq:roster'>";
            editstring=editstring+"<item jid="+selecteduserid+" name='"+selectednickname+"'
            subscription='remove'/>";
            editstring=editstring+"</query></iq>";
            editsocketclass.sendXMLToJabber(editstring);
            editstring="<iq type='set' >";
            editstring=editstring+"<query xmlns='jabber:iq:roster'>";
            editstring=editstring+"<item jid='"+useridtext.getText().trim()+"' subscription='to'
            name='"+nicknametext.getText().trim()+"' />";
            editstring=editstring+"</query></iq>";
            editsocketclass.sendXMLToJabber(editstring);
            edithash.remove(selectednickname);
            String jabbid="'"+useridtext.getText().trim()+"'";
            edithash.put(nicknametext.getText().trim(),jabbid);
            DefaultTreeModel model = (DefaultTreeModel)edittree.getModel();
            MutableTreeNode node = (MutableTreeNode)path.getLastPathComponent();
            dynamictree.clear();
            tree.clear();
            Enumeration enum=edithash.keys();
            while (enum.hasMoreElements())
            {
               String keyval=(String)enum.nextElement();
               DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(keyval);
               DefaultMutableTreeNode childnode = new  DefaultMutableTreeNode(keyval);
               dynamictree.addObject(treenode);
               tree.addObject(childnode);
            }
         }
      }
   }
   /* This block will be executed when the end user clicks delete button.*/
   else if (actioncommand.equals("delete"))
   {
      String editstring="<iq type='set' id='1'>";
      editstring=editstring+"<query xmlns='jabber:iq:roster'>";
      editstring=editstring+"<item jid="+selecteduserid+" name='"+selectednickname+"'
      subscription='remove'/>";
      editstring=editstring+"</query></iq>";
      editsocketclass.sendXMLToJabber(editstring);
      DefaultTreeModel model = (DefaultTreeModel)edittree.getModel();
      MutableTreeNode node = (MutableTreeNode)path.getLastPathComponent();
      model.removeNodeFromParent(node);
      useridtext.setText("");
      nicknametext.setText("");
      edithash.remove(selectednickname);
      dynamictree.clear();
      Enumeration enum=edithash.keys();
      while (enum.hasMoreElements())
      {
         DefaultMutableTreeNode treenode=new DefaultMutableTreeNode(enum.nextElement());
         dynamictree.addObject(treenode);
      }
   }
   else
   {
   setVisible(false);
   }
}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e)
{
   int selRow = edittree.getRowForLocation(e.getX(), e.getY());
   TreePath selPath = edittree.getPathForLocation(e.getX(), e.getY());
   path=edittree.getSelectionPath();
   DefaultMutableTreeNode node = (DefaultMutableTreeNode)edittree.getLastSelectedPathComponent();
   if(!(node.toString().equals("My Contact List")))
   {
      selecteduserid=(String)edithash.get(node.toString());
      selectednickname=node.toString();
      useridtext.setText(selecteduserid.substring(1,selecteduserid.length()-1));
      nicknametext.setText(selectednickname);
   }
}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
}
```

Download this listing.

In the above code, the constructor of the EditDelete class takes the object of the SocketClass class, DynamicTree class, and HashTable class as input parameters. The EditDelete class allows end users to invoke the methods of the SocketClass class, DynamicTree class, and HashTable class. The EditDelete class defines the actionPerformed() method, which acts as an event listener and activates an appropriate method based on the action an end user performs.

The EditDelete.java file creates the window to edit or delete the existing end users, as shown in Figure 7-6:

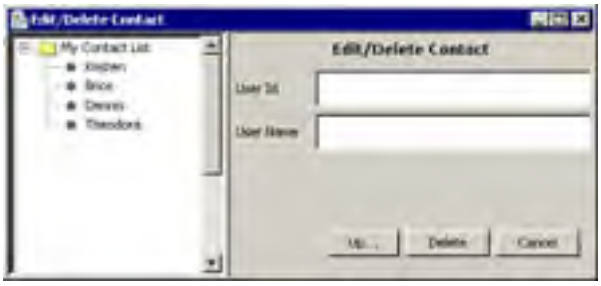


Figure 7-6: Managing the Contact List

PREVIOUS NEXT

Creating the Chat Window

The ChatWindow.java file creates a Chat window to send messages to a particular end user selected in the contact list. Listing 7-5 shows the contents of the ChatWindow.java file:

Listing 7-5: The ChatWindow.java File

```
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required java.awt classes*/
import java.awt.*;
/*Imports required javax.swing.event classes*/
import javax.swing.event.*;
/*Imports required java.awt.event classes*/
import java.awt.event.*;
/*Imports required javax.swing.text package classes*/
import javax.swing.text.*;
/*
class ChatWindow - This class is used to create GUI for chatting.
Constructor:
ChatWindow-This constructor creates GUI.
Methods:
writemessage: This method is called to send input message to insertTextInToTextPane function.
eraseText: This method is called to send input message to insertTextInToTextPane function.
addToRosterTree: This method is called to add user name in to tree.
*/
public class  ChatWindow extends JFrame implements ActionListener
{
/*Declares object of Container class.*/
Container container;
/*Declares object of GridBagLayout class.*/
GridBagLayout gridbaglayout;
/*Declares object of JLabel class.*/
JLabel fromuser = null;
JLabel touser = null;
JLabel fromusername = null;
JLabel tousername = null;
/*Declares object of JTextPane class.*/
JTextPane receivemessagetextarea = null;
/*Declares object of JTextField class.*/
JTextField sendmessagetext = null;
/*Declares object of JButton class.*/
JButton sendbutton = null;
/*Declares object of JButton class.*/
JScrollPane scpane;
/*Declares object of String class.*/
String username;
String usernickname;
String sendername;
/*Declares object of Document class.*/
Document contentmodel;
/*Declares object of MutableAttributeSet class.*/
MutableAttributeSet recervernameatrib, sendermessagattrib;
/*Declares object of SocketClass class.*/
SocketClass socketchatclass;
public ChatWindow(String nickname,String recevername, SocketClass socketclass)
{
   sendername=socketclass.getUserName();
   /*Initializes the objects of JLabel class.*/
   username=recevername;
   usernickname=nickname;
   socketchatclass=socketclass;
   fromuser = new JLabel("From :");
   touser = new JLabel("To :");
   fromusername=new JLabel(socketclass.getUserName());
   tousername = new JLabel(usernickname);
   /*Sets window title.*/
   setTitle("Chat Window: "+usernickname);
   container = this.getContentPane();
   /*Initializes the objects of GridBagLayout class.*/
   gridbaglayout=new GridBagLayout();
   /*Declare and initializes the objects of GridBagConstraints class.*/
   GridBagConstraints gridbagconstraints=new GridBagConstraints();
   /*Declare and initializes the objects of JPanel class.*/
   JPanel labelpanel = new JPanel(gridbaglayout);
   /*Sets constraints for gridbagconstraints.*/
   gridbagconstraints.fill=GridBagConstraints.BOTH;
   gridbagconstraints.insets=new Insets(5, 5, 0, 0);
   gridbagconstraints.gridx=0;
   gridbagconstraints.gridy=0;
   gridbagconstraints.weightx=1;
   gridbagconstraints.weighty=1;
   gridbagconstraints.anchor=GridBagConstraints.WEST;
   gridbaglayout.setConstraints(fromuser, gridbagconstraints);
```

```
/*Sets fromuser size.*/
fromuser.setPreferredSize(new Dimension(140, 28));
/*Adds label to labelpanel.*/
labelpanel.add(fromuser);
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 20);
gridbagconstraints.gridx=1;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.EAST;
gridbaglayout.setConstraints(fromusername, gridbagconstraints);
/*Sets fromusername size.*/
fromusername.setPreferredSize(new Dimension(20, 28));
/*Sets fromusername font.*/
fromusername.setFont(new Font("Verdana", Font.BOLD, 10));
/*Adds fromusername to labelpanel.*/
labelpanel.add(fromusername);
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=2;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.EAST;
gridbaglayout.setConstraints(touser, gridbagconstraints);
/*Sets touser size.*/
touser.setPreferredSize(new Dimension(140, 28));
/*Adds touser to labelpanel.*/
labelpanel.add(touser);
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=3;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.WEST;
gridbaglayout.setConstraints(tousername, gridbagconstraints);
/*Sets tousername size.*/
tousername.setPreferredSize(new Dimension(20, 28));
/*Sets tousername font.*/
tousername.setFont(new Font("Verdana", Font.BOLD, 10));
/*Adds tousername to labelpanel.*/
labelpanel.add(tousername);
/*Adds labelpanel to container.*/
container.add(labelpanel, BorderLayout.NORTH);
/*Initializes the objects of JTextPane class.*/
receivemessagetextarea = new JTextPane();
contentmodel=receivemessagetextarea.getDocument();
/*Initializes the objects of SimpleAttributeSet class.*/
recervernameatrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(recervernameatrib,"Verdana");
StyleConstants.setForeground(recervernameatrib, Color.black);
sendermessagattrib=new SimpleAttributeSet();
StyleConstants.setFontFamily(sendermessagattrib,"Verdana");
StyleConstants.setForeground(sendermessagattrib,Color.red);
receivemessagetextarea.setPreferredSize(new Dimension(80, 20));
receivemessagetextarea.setEditable(false);
/*Initializes the objects of JScrollPane class.*/
scpane  = new JScrollPane(receivemessagetextarea, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
/*Adds scpane to container.*/
container.add(scpane,BorderLayout.CENTER);
/*Declare and initializes the objects of JPanel class.*/
JPanel sendmessagepanel = new JPanel(gridbaglayout);
/*Initializes the objects of JTextField class.*/
sendmessagetext = new JTextField();
/*Initializes the objects of JButton class.*/
sendbutton = new JButton("Send");
/*Adds action listener to sendbutton.*/
sendbutton.addActionListener(this);
/*Sets action command of sendbutton.*/
sendbutton.setActionCommand("send");
/*Sets constraints for gridbagconstraints.*/
gridbagconstraints.fill=GridBagConstraints.BOTH;
gridbagconstraints.insets=new Insets(5, 5, 0, 0);
gridbagconstraints.gridx=0;
gridbagconstraints.gridy=0;
gridbagconstraints.weightx=1;
gridbagconstraints.weighty=1;
gridbagconstraints.anchor=GridBagConstraints.NORTH;
gridbaglayout.setConstraints(sendmessagetext, gridbagconstraints);
/*Sets tousername size.*/
sendmessagetext.setPreferredSize(new Dimension(120, 28));
/*Adds sendmessagetext to the sendmessagepanel*/
```

```
   sendmessagepanel.add(sendmessagetext);
   /*Sets constraints for gridbagconstraints.*/
   gridbagconstraints.fill=GridBagConstraints.BOTH;
   gridbagconstraints.insets=new Insets(5, 5, 0, 0);
   gridbagconstraints.gridx=1;
   gridbagconstraints.gridy=0;
   gridbagconstraints.weightx=0;
   gridbagconstraints.weighty=0;
   gridbagconstraints.anchor=GridBagConstraints.EAST;
   gridbaglayout.setConstraints(sendbutton, gridbagconstraints);
   /*Sets sendbutton size.*/
   sendbutton.setPreferredSize(new Dimension(60, 28));
   sendmessagepanel.add(sendbutton);
   container.add(sendmessagepanel, BorderLayout.SOUTH);
   setBounds(100, 100, 350, 280);
   show();
}
public void actionPerformed(ActionEvent ae)
{
   String actioncommand=ae.getActionCommand();
   String messagestring;
   if (actioncommand.equals("send"))
   {
      if (!sendmessagetext.getText().trim().equals(""))
      {
         String recevername=username.substring(1,username.length()-1);
         messagestring="<message type='chat' to='"+recevername+"/Home'
         from='"+sendername+"@"+socketchatclass.getServerName()+"/Home'>";
         messagestring=messagestring+"<body>";
         messagestring=messagestring+sendmessagetext.getText().trim();
         messagestring=messagestring+"</body></message>";
         try
         {
            contentmodel.insertString(contentmodel.getLength(),"\n"+sendername+":
            ",sendermessagattrib);
            contentmodel.insertString(contentmodel.getLength(), sendmessagetext.getText().trim(),
            recervernameatrib);
         }
         catch(BadLocationException ble)
         {
         }
         socketchatclass.sendXMLToJabber(messagestring);
         sendmessagetext.setText("");
         }
      }
   }
   /*
insertTextInToTextPane: This method is called to insert text message into textpane.
Parameter: pmessage - Object of  String class, psendername - Object of  String class.
Return Value: N/A
*/
public void insertTextInToTextPane(String pmessage,String psendername)
{
   try
   {
   contentmodel.insertString(contentmodel.getLength(), "\n"+usernickname+":
   "+pmessage,recervernameatrib);
   }
   catch(BadLocationException ble) {}
   }
}
```

Download this listing.

In the above code, the constructor of the ChatWindow class takes an object of the SocketClass class and two strings, userName and receivername as input parameters. The object of the SocketClass class allows end users to invoke the methods of the SocketClass class. The userName string retrieves the user name of the selected end user, and the receivername string retrieves the user ID of the selected end user.

The methods defined in Listing 7-5 are:

- `insertTextInToTextPane()`: Inserts the text messages specified by end users in the text area of the Chat window.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action an end user performs.

The ChatWindow.java file creates the Chat window to send messages to the selected end user, as shown in Figure 7-7:

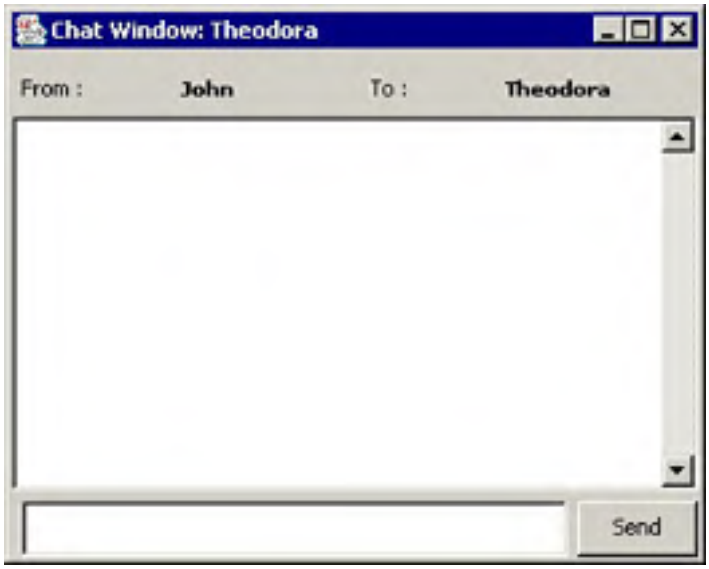


Figure 7-7: The Chat Window

PREVIOUS NEXT

Sending and Receiving Messages

The SocketClass.java file opens the socket to send and receive messages between end users with the help of a Jabber server. Listing 7-6 shows the contents of the SocketClass.java file:

Listing 7-6: The SocketClass.java File

```
/*Imports required java.util classes*/
import java.util.*;
/*Imports required java.io classes*/
import java.io.*;
/*Imports required java.net classes*/
import java.net.*;
/*Imports required javax.swing classes*/
import javax.swing.*;
/*Imports required DefaultMutableTreeNode class*/
import javax.swing.tree.DefaultMutableTreeNode;
/*
class SocketClass - This class is used for reading the input messages and writing output messages.
Methods:
isConnected: This method returns a boolean value.
isWaitingForAuth: This method returns a boolean value.
socketoOpenClass: This method returns a boolean value.
setUserLoginHandler: This method sets handler to the object of UserLogin class.
checkForRoster: This method calls another method of ContactList class.
getUserInfo: This method returns a boolean value.
setUserInfo: This method sets value of nouserexists variable.
sendRosterRequest: This method creates a Jabber message string for Roster and passes this string int
sendXMLToJabber method.
getUserName: This method returns user name.
getServerName: This method returns the server name.
setWaitForAuth: This method sets a boolean value for waitforauth.
sendXMLToJabber: This method sends XML message string for the jabber server.
checkForError: This method checks whether input message contains a error code or not.
sendername: This method extracts sender name from input message.
setUserNameAndPassword: This method stores the user name and password of the logged in end user.
sendSessionStartMessage: This method creates a Jabber message string for starting a jabber session a
passes this string into sendXMLToJabber method.
sendAuthorized: This method creates a Jabber message string for user authentication and passes this
string into sendXMLToJabber method.
sendRegistration: This method creates a Jabber message string for user registration and passes this
string into sendXMLToJabber method.
openPort: This method creates an object of SocketOpener class and class openSocket method of  this
class.
*/
class SocketClass implements Runnable
{
   boolean isConnected=false;
   /*Declares the object of Vector class.*/
   Vector tagvactor;
   /*Declares the object of PrintWriter class.*/
   PrintWriter out = null;
   /*Declares the object of BufferedReader class.*/
   BufferedReader in = null;
   /*Declares and initializes the object of Vector class.*/
   String servername;
   int portno;
   int wait;
   Socket clientsocket;
   /*Declares the object of Thread class.*/
   Thread inputmessagethread;
   /*Declares the object of String class.*/
   String errortype;
   String resource="Home";
   String username="";
   String password="";
   String auth="";
   String sendername="";
   String recevername="";
   String vcardString="";
   boolean waitforreg=false;
   boolean waitforauth=false;
   boolean waitForResult=false;
   boolean rosterflag=false;
   boolean nouserexists=false;
   boolean endofvcard=false;
   /*Declares the object of UserLogin class.*/
   UserLogin userloginhandler;
   /*Declares the object of ContactList class.*/
   ContactList contactlist;
   /*Declares the object of UserInfo class.*/
   UserInfo ui;
   public SocketClass() {}
   /*
```

```
isConnected: This method returns a boolean value.
Parameter: N/A
Return Value: boolean
*/
public boolean isConnected()
{
   return isConnected;
}
/*
isWaitingForAuth: This method returns a boolean value.
Parameter: N/A
Return Value: boolean
*/
public boolean isWaitingForAuth()
{
   return waitforauth;
}
/*
socketoOpenClass: This method returns a boolean value
Parameter: ipaddress - object of String class, portno, wait
Return Value: boolean
*/
public void socketoOpenClass(String ipaddress, int portno, int wait)
{
   this.servername=ipaddress;
   this.portno=portno;
   this.wait=wait;
   openPort(servername,portno,wait);
   if (clientsocket!=null)
   {
      isConnected=true;
      try
      {
         /*Initializes the object of PrintWriter class.*/
         out = new PrintWriter(clientsocket.getOutputStream(), true);
         /*Initializes the object of BufferedReader class.*/
         in = new BufferedReader(new InputStreamReader(clientsocket.getInputStream()));
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
      /*Initializes the object of Thread class.*/
      inputmessagethread=new Thread(this);
      /*Calls start method of inputmessagethread.*/
      inputmessagethread.start();
      }
   }
   /*
   setUserLoginHandler: This method sets handler to the object of UserLogin class.
   Parameter: userlogin - object of UserLogin class
   Return Value: N/A
   */
   public void setUserLoginHandler(UserLogin userlogin)
   {
      userloginhandler=userlogin;
   }
   public void run()
   {
      int i=0;
      String errororwarning="";
      String errorstring="";
      String inputstring="";
      boolean starttag=false;
      boolean endtag=false;
      boolean startwritting=false;
      String messagebody="";
      String tagentity="";
      Vector tagvactor;
      tagvactor=new Vector();
      int vactorindex=0;
      int starttagsing=0;
      while (true)
      {
         try
         {
            i=in.read();
            if (i==-1)
            {
               break;
            }
            else
            {
            inputstring=inputstring+(char)i;
            if (endofvcard)
            {
               vcardString=vcardString+(char)i;
            }
```

```
if ((char)i=='<')
{
   starttag=true;
   starttagsing=1;
   tagentity="";
}
else
{
   if ((char)i=='/' && starttag==true )
   {
      if (starttagsing==1)
      {
         starttag=false;
         endtag=true;
         if (!messagebody.trim().equals("")&&(!nouserexists))
         {
            contactlist.writemessage(messagebody,sendername,recevername);
         }
         startwritting=false;
         messagebody="";
         vactorindex=vactorindex-1;
         if (vactorindex>=0)
         tagvactor.removeElementAt(vactorindex);
      }
      tagentity=tagentity+(char)i;
   }
   else
   {
   starttagsing=0;
   if((char)i=='>')
   {
      if (starttag)
      {
         if (tagentity.endsWith("/"))
         {
            tagentity=tagentity.substring(0,tagentity.length()-1);
         }
         if (tagentity.equals("PREF"))
         {
            String
            nodename=((DefaultMutableTreeNode)contactlist.rostertree.
            getLastSelectedPathComponent()).
            toString();
            String uid=(String)contactlist.hashnode.get(nodename);
            ui.setInfo(vcardString,nodename,uid);
            vcardString="";
         }
         sendername(tagentity);
         errorstring=checkForError(tagentity);
         if (errorstring.equals("unauthorized"))
         {
            sendRegistration();
            waitforreg=true;
            waitforauth=false;
         }
         if ((tagentity.indexOf("type='result'")>1)&&(waitforreg)&&
         (!errorstring.equals
         ("user exist")))
         {
            sendAuthorized();
            waitforauth=true;
            waitforreg=false;
         }
         if ((tagentity.indexOf("type='result'")>1)&&(waitforauth))
         {
            waitforauth=false;
         }
         if (tagentity.indexOf("item-not-found
         xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'")!=-1)
         {
            nouserexists=true;
            contactlist.removeFromRosterTree(sendername);
         }
         if (tagentity.indexOf("remote-server-not-found")!=-1)
         {
            nouserexists=true;
            contactlist.removeFromRosterTree(sendername);
         }
         if (rosterflag)
         {
            if (tagentity.indexOf("item")!=-1&&(!nouserexists))
            {
               String[] splitedsendername=sendername.split("/");
               String contactpersonname=tagentity.substring(tagentity.indexOf("name")
               tagentity.indexOf("jid")).trim();
               String
               jid=tagentity.substring(tagentity.indexOf("jid")+4,tagentity.length())
               if (!jid.equals("'"+splitedsendername[0]+"'"))
```

```
                {
                   contactlist.addToRosterTree(jid.trim(), contactpersonname.trim());
                }
             }
             else
             {
                if (nouserexists)
                {
                   contactlist.eraseText(sendername);
                   JOptionPane.showMessageDialog(null, "No User exists with this jabbe
                   ID.", "Error",JOptionPane.PLAIN_MESSAGE);
                   nouserexists=false;
                }
             }
             if (tagentity.indexOf("query")!=-1)
             {
                rosterflag=false;
             }
          }
          if (tagentity.indexOf("xmlns='vcard-temp'")>1&&(!endofvcard))
          {
             endofvcard=true;
          }
          if
          ((tagentity.indexOf("type='result'")>1)&&
          (!waitforauth)&&(!waitforreg))
          {
             waitForResult=true;
          }
          else
          {
             if (tagentity.indexOf("xmlns='jabber:iq:roster'")
             >1&&waitForResult)
             {
                rosterflag=true;
                waitForResult=false;
             }
             waitForResult=false;
          }
          if ((tagentity.indexOf("type='result'")>1)&&(!waitforauth))
          {
             if (contactlist==null)
             {
                contactlist=new ContactList(this);
                sendRosterRequest();
                userloginhandler.setVisible(false);
             }
          }
          if (errorstring.equals("user exixts")&&waitforreg)
          {
             JOptionPane.showMessageDialog(null, "A user with this user ID is already
             exists.Please enter
             other
             user ID.", errororwarning, JOptionPane.PLAIN_MESSAGE);
          }
          if (!errorstring.equals(""))
          {
             if (errortype=="major")
             {
                errororwarning="Error";
                JOptionPane.showMessageDialog(null,errorstring, errororwarning,
                JOptionPane.PLAIN_MESSAGE);
             }
             else
             {
                errororwarning="Warning";
                JOptionPane.showMessageDialog(null, errorstring, errororwarning,
                JOptionPane.PLAIN_MESSAGE);
             }
          }
          startwritting=true;
          tagvactor.insertElementAt(tagentity,vactorindex);
          vactorindex=vactorindex+1;
          starttag=false;
          }
       }
       else
       {
          if (startwritting==true&&tagentity.trim().equals("body"))
          {
             messagebody=messagebody+(char)i;
          }
          if (starttag)
          {
             tagentity=tagentity+(char)i;
          }
       }
    }
```

```
                }
            }
         }
         catch(IOException ie)
         {
         }
   }
   isConnected=false;
}
/*
checkForRoster: This method calls another method of ContactList class.
Parameter: uname - object of String class,nname - object of String class
Return Value: boolean
*/
public boolean checkForRoster(String uname,String nname)
{
   return contactlist.checkHash(uname,nname);
}
/*
getUserInfo: This method returns a boolean value.
Parameter: N/A
Return Value: boolean
*/
public boolean getUserInfo()
{
   return nouserexists;
}
/*
setUserInfo: This method sets value of nouserexists variable.
Parameter: flag
Return Value: N/A
*/
public void setUserInfo(boolean flag)
{
   nouserexists=flag;
}
/*
sendRosterRequest: This method creates a Jabber message string  for Roster and passes this string in
sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendRosterRequest()
{
   String rosterstring="";
   rosterstring="<iq type=\"get\" id=\"1\">";
   rosterstring=rosterstring+"<query xmlns=\"jabber:iq:roster\"/></iq>";
   sendXMLToJabber(rosterstring);
}
/*
getUserName: This method returns user name.
Parameter: N/A
Return Value: String
*/
public String getUserName()
{
   return username;
}
/*
getServerName: This method returns the server name.
Parameter: N/A
Return Value: String
*/
public String getServerName()
{
   return servername;
}
/*
setWaitForAuth: This method sets a boolean value for waitforauth.
Parameter: flag - boolean
Return Value: N/A
*/
public void setWaitForAuth(boolean flag)
{
   waitforauth=flag;
}
/*
sendXMLToJabber: This method sends XML message string for the jabber server.
Parameter: outputmessage - object of String class
Return Value: N/A
*/
public void sendXMLToJabber(String outputmessage)
{
   if (outputmessage.indexOf("<vCard")!=-1)
   {
      ui=new UserInfo();
   }
   String[] tokenizerstring=outputmessage.split("\n");
```

```
   for (int i=0;i<tokenizerstring.length;i++ )
   {
      try
      {
         out.println(tokenizerstring[i]);
         out.flush();
      }catch(Exception e)
      {
         return;
      }
      out.flush();
   }
}
/*
checkForError: This method checks whether input message contains a error code or not.
Parameter: tagentity - object of String class
Return Value: String
*/
public String checkForError(String tagentity)
{
   String error="";
   String codeid="";
   if (tagentity.equals("error code='403' type='auth'"))
   {
      sendRegistration();
   }
   if ((tagentity.indexOf("code='401'")!=-1)||(tagentity.indexOf("code=\"401\"")!=-1))
   {
      return "unauthorized";
   }
   if ((tagentity.indexOf("code='409'")!=-1)||(tagentity.indexOf("code=\"409\"")!=-1))
   {
      return "user exixts";
   }
   return error;
}
/*
sendername: This method extracts sender name from input message.
Parameter: tagentity - object of String class
Return Value: N/A
*/
public void sendername(String tagentity)
{
   if (tagentity.startsWith("message"))
   {
      sendername="";
      recevername="";
      if (tagentity.indexOf("from=")>0&&tagentity.indexOf("to=")>0)
      {
         sendername=tagentity.substring(tagentity.indexOf("from=")+6,tagentity.indexOf("to=")-2);
      }
      if (tagentity.indexOf("to=")>0&&tagentity.indexOf("type=")>0)
      {
         recevername=tagentity.substring(tagentity.indexOf("to=")+4,tagentity.indexOf("type=")-2);
      }
   }
}
public void setUserNameAndPassword(String username,String password)
{
   this.username=username;
   this.password=password;
}
/*
sendSessionStartMessage: This method creates a Jabber message string for starting a jabber session a
passes this string into sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendSessionStartMessage()
{
   String sessionStratString;
   sessionStratString="<?xml version=\"1.0\" encoding=\"UTF-8\" ?>";
   sessionStratString=sessionStratString+"<stream:stream";
   sessionStratString=sessionStratString+" to= \""+servername +"\"";
   sessionStratString=sessionStratString+" xmlns=\"jabber:client\"";
   sessionStratString=sessionStratString+" xmlns:stream=\"http://etherx.jabber.org/streams\">";
   sendXMLToJabber(sessionStratString);
}
/*
sendAuthorized: This method creates a Jabber message string for user authentication and passes this
string into sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendAuthorized()
{
   String authentication;
```

```
   authentication="<iq type=\"set\" id=\"1\">";
   authentication=authentication+" <query xmlns=\"jabber:iq:auth\">";
   authentication=authentication+"<username>"+username+"</username>";
   authentication=authentication+"<password>"+password+"</password>";
   authentication=authentication+"<resource>"+resource+"</resource>";
   authentication=authentication+"</query> ";
   authentication=authentication+"</iq>";
   sendXMLToJabber(authentication);
}
/*
sendRegistration: This method creates a Jabber message string for user registration and passes this
string into sendXMLToJabber method.
Parameter: N/A
Return Value: N/A
*/
public void sendRegistration()
{
   String registrationstring;
   registrationstring="<iq type=\"set\" to=\""+username+"."+servername+"\" id=\"1\">";
   registrationstring=registrationstring+"<query xmlns=\"jabber:iq:register\">";
   registrationstring=registrationstring+"<username>"+username+"</username>";
   registrationstring=registrationstring+"<password>"+password+"</password></
   query></iq>";
   sendXMLToJabber(registrationstring);
}
/*
openPort: This method creates an object of SocketOpener class and class openSocket method of this
class.
Parameter: ipaddress - Object of String class, portno - int, timeinsec - int
Return Value: N/A
*/
private void openPort(String ipaddress,int portno, int timeinsec)
{
   String host;
   if (ipaddress.trim()=="")
   {
      System.out.println("No IP Address Specified.");
   }
   else
   {
      SocketOpener socketopener=new SocketOpener();
      clientsocket=socketopener.openSocket(ipaddress, portno, timeinsec);
   }
}
}
/*
class SocketOpener - This class is used to create an object of socket class.
Constructor:
openSocket
Methods:
getSocket: This method returns a handler of the object of socket class.
*/
class SocketOpener
{
   private Socket socket;
   public Socket openSocket(String hostip, int portnumber, int timeinsec)
   {
      try
      {
         socket=new Socket(hostip,portnumber);
      }
      catch(IOException ie)
      {
         System.out.println("Exception occurred while creating a socket : " +ie);
      }
      return getSocket();
   }
   public SocketOpener()
   {
   }
   /*
   getSocket: This method returns a handler of the object of socket class.
   Parameter: N/A
   Return Value: Socket
   */
   public Socket getSocket()
   {
      return socket;
   }
}
```

Download this listing.

In the above code, the constructor of the SocketClass class creates a new instance of the SocketClass class. The object of the SocketClass class allows end users to invoke methods of the SocketClass class.

The methods defined in Listing 7-6 are:

- `isConnected()`: Checks whether or not an end user is connected to the Jabber server.
- `isWaitingForAuth()`: Checks whether or not an end user is authenticated with the Jabber server.
- `socketoOpenClass()`: Checks whether or not a socket is open.
- `setUserLoginHandler()`: Sets the handler to the object of the UserLogin class.
- `checkForRoster()`: Calls the checkHash() method of the ContactList class.
- `getUserInfo`: Checks whether or not information about an end user to be added to the contact list exists on the Jabber server.
- `setUserInfo()`: Sets the status as true if the specified end user information does not exist on the Jabber server.
- `sendRosterRequest`: Creates a message string and passes this message string to the sendXMLToJabber() method.
- `getUserName()`: Retrieves the name of an end user.
- `getServerName()`: Retrieves the name of the Jabber server.
- `setWaitForAuth()`: Sets the status as true if an end user is authenticated by the Jabber server.
- `sendXMLToJabber()`: Sends the message string to the Jabber server in XML format.
- `checkForError()`: Checks whether or not the input string received from the Jabber server contains an error message.
- `sendername()`: Retrieves the sender's name from the input message received from the Jabber server.
- setUserNameAndPassword(): Stores the user name and password of the logged on end user.
- `sendSessionStartMessage()`: Creates a message string in the XML format for starting a Jabber session and passes this message string to the sendXMLToJabber() method.
- `sendAuthorized()`: Creates a message string in XML format for user authentication and passes this message string to the sendXMLToJabber() method.
- `sendRegistration()`: Creates a message string in XML format for user registration and passes this message string to the sendXMLToJabber() method.
- `openPort`: Opens a client socket by using the IP address and port number of the Jabber server.
- `run()`: Listens for the response sent by the Jabber server.

PREVIOUS NEXT

Creating a Tree Structure of the Contact List

The DynamicTree.java file creates and maintains a tree structure of the contact list. Listing 7-7 shows the contents of the DynmaicTree.java file:

Listing 7-7: The DynamicTree.java File

```
/*Imports required GridLayout class*/
import java.awt.GridLayout;
/*Imports required Toolkit class*/
import java.awt.Toolkit;
/*Imports required JPanel class*/
import javax.swing.JPanel;
/*Imports required JScrollPane class*/
import javax.swing.JScrollPane;
/*Imports required JTree class*/
import javax.swing.JTree;
/*Imports required DefaultMutableTreeNode class*/
import javax.swing.tree.DefaultMutableTreeNode;
/* mports required DefaultTreeModel class*/
import javax.swing.tree.DefaultTreeModel;
/*Imports required MutableTreeNode class*/
import javax.swing.tree.MutableTreeNode;
/*Imports required TreePath class*/
import javax.swing.tree.TreePath;
/*Imports required TreeSelectionModel class*/
import javax.swing.tree.TreeSelectionModel;
/*Imports required TreeModelEvent class*/
import javax.swing.event.TreeModelEvent;
/*Imports required TreeModelListener class*/
import javax.swing.event.TreeModelListener;
/*
class DynamicTree - This class is used to create a tree.
Constructor:
DynamicTree-This constructor creates GUI.
Methods:
clear: This method is called remove all node from tree.
getTreeHandler: This method is called to get handler of the object of JTree class.
removeCurrentNode: This method is called to remove selected node from tree.
addObject: This method is called to add node into the tree.
*/
public class DynamicTree extends JPanel
{
   /*Declares object of DefaultMutableTreeNode class.*/
   DefaultMutableTreeNode rootNode;
   /*Declares object of DefaultTreeModel class.*/
   DefaultTreeModel treeModel;
   /*Declares object of JTree class.*/
   JTree tree;
   /*Declares and initializes object of Toolkit class.*/
   Toolkit toolkit = Toolkit.getDefaultToolkit();
   public DynamicTree(DefaultMutableTreeNode node)
   {
      super(new GridLayout(1, 0));
      /*initializes the object of DefaultMutableTreeNode class.*/
      rootNode = new DefaultMutableTreeNode("My Contact List");
      /*initializes the object of DefaultTreeModel class.*/
      treeModel = new DefaultTreeModel(rootNode);
      treeModel.addTreeModelListener(new RosterTreeModelListener());
      /*initializes the object of JTree class.*/
      tree = new JTree(treeModel);
      tree.setEditable(true);
      tree.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
      tree.setShowsRootHandles(true);
      /*Declares and initializes object of JScrollPane class.*/
      JScrollPane scrollPane = new JScrollPane(tree);
      add(scrollPane);
   }
   /*
   clear: This method is called remove all node from tree.
   Parameter: N/A
   Return Value: N/A
   */
   public void clear()
   {
      rootNode.removeAllChildren();
      treeModel.reload();
   }
   /*
   getTreeHandler: This method is called to get handler of the object of JTree class.
   Parameter: N/A
   Return Value: JTree
   */
```

```
public JTree getTreeHandler()
{
   return tree;
}
/*
removeCurrentNode: This method is called to remove selected node from tree.
Parameter: N/A
Return Value: N/A
*/
public void removeCurrentNode()
{
TreePath currentSelection = tree.getSelectionPath();
if (currentSelection != null)
{
   DefaultMutableTreeNode currentNode = (DefaultMutableTreeNode)
   (currentSelection.getLastPathComponent());
   MutableTreeNode parent = (MutableTreeNode)(currentNode.getParent());
   if (parent != null) {
      treeModel.removeNodeFromParent(currentNode);
      return;
      }
   }
   toolkit.beep();
}
/*
addObject: This method is called to add node into the tree.
Parameter: child - Object of Object class.
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(Object child)
{
   DefaultMutableTreeNode parentNode = null;
   TreePath parentPath = tree.getSelectionPath();
   if (parentPath == null)
   {
      parentNode = rootNode;
   }
   else
   {
      parentNode = (DefaultMutableTreeNode)(parentPath.getLastPathComponent());
   }
   return addObject(parentNode, child, true);
}
/*
addObject: This method is called to add child into the give tree node.
Parameter: parent - Object of DefaultMutableTreeNode class, child - Object of Object class.
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent,Object child)
{
   return addObject(parent, child, true);
}
/*
addObject: This method is called to add child into the give tree node.
Parameter: parent - Object of DefaultMutableTreeNode class, child - Object of Object
class, shouldBeVisible
Return Value: DefaultMutableTreeNode
*/
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode parent, Object child, boolean
shouldBeVisible)
{
   DefaultMutableTreeNode childNode = new DefaultMutableTreeNode(child);
   if (parent == null)
   {
      parent = rootNode;
   }
   treeModel.insertNodeInto(childNode, parent, parent.getChildCount());
   if (shouldBeVisible)
   {
      tree.scrollPathToVisible(new TreePath(childNode.getPath()));
   }
   return childNode;
}
/*
class DynamicTree - This class implements TreeModelListener interface.
*/
class RosterTreeModelListener implements TreeModelListener
{
   public void treeNodesChanged(TreeModelEvent e)
   {
      DefaultMutableTreeNode node;
      node = (DefaultMutableTreeNode)(e.getTreePath().getLastPathComponent());
      try
      {
         int index = e.getChildIndices()[0];
         node = (DefaultMutableTreeNode)(node.getChildAt(index));
      }
      catch (NullPointerException exc) {}
```

```
        }
        public void treeNodesInserted(TreeModelEvent e) {}
        public void treeNodesRemoved(TreeModelEvent e) {}
        public void treeStructureChanged(TreeModelEvent e) {}
    }
}
```

Download this listing.

In the above code, the constructor of the DynamicTree class takes an object of the DefaultMutableTreeNode class and creates a tree structure of the contact list for the Contact List application.

The methods defined in Listing 7-7 are:

- `clear()`: Removes all end user names from the tree structure of the contact list.
- `getTreeHandler()`: Returns the object of the JTree class.
- `removeCurrentNode()`: Removes the selected end users from the tree structure of the contact list.
- `addObject()`: Adds other end users to the tree structure of the contact list.

« PREVIOUS NEXT »

Viewing Contact Information

The UserInfo.java file creates the user interface that allows end users to view the contact information of the selected end user from the tree structure of the Contact List application. Listing 7-8 shows the contents of the UserInfo.java file:

Listing 7-8: The UserInfo.java File

```
/*Import required javax.swing classes*/
import javax.swing.*;
/*Import required java.awt classes*/
import java.awt.*;
/*Import required java.awt.event. classes*/
import java.awt.event.*;
/*
class UserInfo - This class is used to show user profile which, is selected in the roster tree.
Constructor:
UserInfo-This constructor creates GUI.
Methods:
setInfo: This method is used to show user information text into labels.
*/
public class UserInfo extends JDialog  implements ActionListener
{
   /*Declares object of Container class.*/
   Container container;
   /*Declares objects of JLabel class.*/
   JLabel heading= null;
   JLabel userid = null;
   JLabel nickname = null;
   JLabel email = null;
   JLabel phone = null;
   JLabel address = null;
   JLabel useriddata = null;
   JLabel nicknamedata = null;
   JLabel emaildata = null;
   JLabel phonedata = null;
   JLabel addressdata = null;
   /*Declares objects of String class.*/
   String vcardxml;
   String fname="";
   String nname="";
   String addressinfo="";
   String telnumber="";
   /*Declares object of JButton class.*/
   JButton okbutton;
   public UserInfo()
   {
      container = this.getContentPane();
      /*Declares and initializes the object of JPanel class.*/
      JPanel toppanel = new JPanel();
      setTitle("Contact Information");
      /*Declares and initializes the objects of JLabel class.*/
      heading = new JLabel("Contact Information");
      userid = new JLabel("User Id:");
      nickname = new JLabel("User Name:");
      email=new JLabel("Email:");
      phone=new JLabel("Phone:");
      address=new JLabel("Address:");
      useriddata=new JLabel("");
      nicknamedata=new JLabel("");
      emaildata=new JLabel();
      phonedata=new JLabel("");
      addressdata=new JLabel("");
      /*Declares and initializes the objects of JButton class.*/
      okbutton=new JButton("OK");
      /*Sets the font of the heading.*/
      heading.setFont(new Font("Verdana", Font.BOLD, 12));
      /*Adds heading to the toppanel.*/
      toppanel.add(heading);
      container.add(toppanel,BorderLayout.NORTH);
      /*Declares and initializes the objects of JPanel class.*/
      JPanel userinfopanel = new JPanel(new GridLayout(5, 6, 10, 20));
      /*Adds userid to the userinfopanel.*/
      userinfopanel.add(userid);
      /*Sets the font of the useriddata.*/
      useriddata.setFont(new Font("Verdana", Font.BOLD, 10));
      /*Adds useriddata to the userinfopanel.*/
      userinfopanel.add(useriddata);
      /*Adds nickname to the userinfopanel.*/
      userinfopanel.add(nickname);
      /*Sets the font of the nicknamedata.*/
      nicknamedata.setFont(new Font("Verdana", Font.BOLD, 10));
      /*Adds nicknamedata to the userinfopanel.*/
      userinfopanel.add(nicknamedata);
      /*Adds email to the userinfopanel.*/
```

```
        userinfopanel.add(email);
        /*Sets the font of the emaildata.*/
        emaildata.setFont(new Font("Verdana", Font.BOLD,10));
        /*Adds emaildata to the userinfopanel.*/
        userinfopanel.add(emaildata);
        /*Adds phone to the userinfopanel.*/
        userinfopanel.add(phone);
        /*Sets the font of the phonedata.*/
        phonedata.setFont(new Font("Verdana", Font.BOLD, 10));
        /*Adds phonedata to the userinfopanel.*/
        userinfopanel.add(phonedata);
        /*Adds address to the userinfopanel.*/
        userinfopanel.add(address);
        /*Sets the font of the addressdata.*/
        addressdata.setFont(new Font("Verdana",Font.BOLD,10));
        /*Adds addressdata to the userinfopanel.*/
        userinfopanel.add(addressdata);
        container.add(userinfopanel);
        /*Declares and initializes the objects of JPanel class.*/
        JPanel buttonpanel = new JPanel();
        buttonpanel.add(okbutton);
        okbutton.addActionListener(this);
        container.add(buttonpanel,BorderLayout.SOUTH);
        setSize(350,300);
        setVisible(true);
    }
    /*
    setInfo: This method is used to show user information text into labels.
    Parameter: vcardString - Object of String class, nicknameto - Object of String class,
    UserID - Object of String class
    Return Value: String
    */
    public void setInfo(String vcardString,String nicknameto,String UserID)
    {
        vcardxml=vcardString;
        fname=vcardxml.substring(vcardxml.indexOf("<FN>")+4,vcardxml.indexOf("</FN>"));
        nname=vcardxml.substring(vcardxml.indexOf("<NICKNAME>")+10,vcardxml.indexOf
        ("</NICKNAME>"));
        if ((vcardxml.indexOf("<HOME/>")==-1)&&(vcardxml.indexOf("</HOME>")!=-1))
        {
            telnumber=vcardxml.substring(vcardxml.indexOf("<HOME>")+6,vcardxml.indexOf
            ("</HOME>"))+"\n";
        }
        if ((vcardxml.indexOf("<LOCALITY/>")==-1)&&(vcardxml.indexOf
        ("</LOCALITY>")!=-1))
        {
            addressinfo=addressinfo+vcardxml.substring(vcardxml.indexOf("<LOCALITY>")+10,
            vcardxml.indexOf("</LOCALITY>"));
        }
        if ((vcardxml.indexOf("<LOCALITY/>")==-1)&&
        (vcardxml.indexOf ("</LOCALITY>")!=-1))
        {
            addressinfo=addressinfo+vcardxml.substring(vcardxml.indexOf("<LOCALITY>")+10,
            vcardxml.indexOf("</LOCALITY>"));
        }
        useriddata.setText(UserID.substring(1,UserID.length()-1));
        nicknamedata.setText(nname);
        addressdata.setText(addressinfo);
        phonedata.setText(telnumber);
    }
    public void actionPerformed(ActionEvent ae)
    {
        if(ae.getSource() == okbutton)
        {
            this.dispose();
        }
    }
}
```

Download this listing.

In the above code, the constructor of the UserInfo class creates the user interface to view the contact information of the selected end user.

The methods defined in Listing 7-8 are:

- `setInfo()`: Shows the contact information text of the selected end user.
- `actionPerformed()`: Acts as an event listener and activates an appropriate method based on the action the end user performs.

The UserInfo.java file creates the user interface that allows an end user to view the contact information of the selected end user, as shown in Figure 7-8:

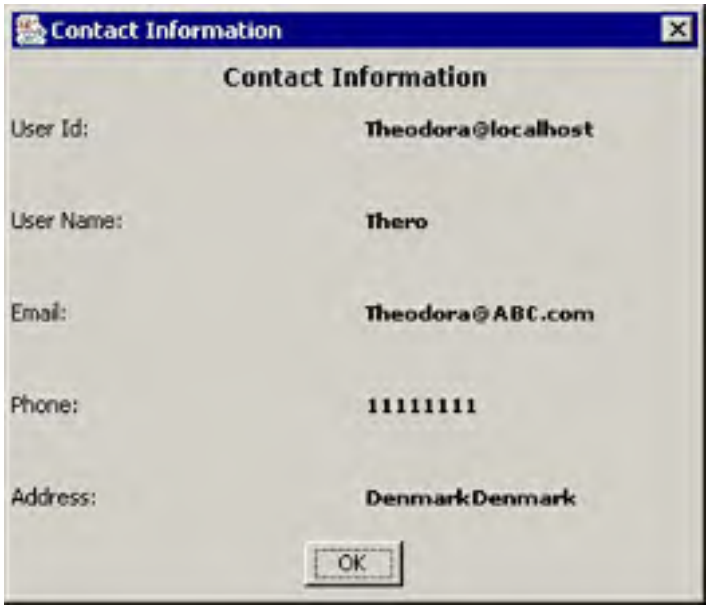


Figure 7-8: The Contact Information Window

PREVIOUS NEXT

Unit Testing

To test the Contact List application:

1. Download and install the free implementation of the Jabber Server from the following URL:

   http://www.jabberstudio.org/projects/ejabberd/releases/view.php?id=610

2. Set the path of the bin directory of J2SDK by executing the following command at the command prompt:

   ```
   set path=%path%;D:\j2sdk1.4.0_02\bin;
   ```

3. Set the classpath of the lib directory of J2SDK by executing the following command at the command prompt:

   ```
   set classpath=%classpath%;D:\j2sdk1.4.0_02\lib;
   ```

4. Copy the UserLogin.java, ContactList.java, UserInfo.java, AddUser.java, EditDelete.java, DynamicTree.java, ChatWindow.java, and SocketClass.java files to a folder on your computer. Use the cd command at the command prompt to move to the folder in which you have copied the Java files. Compile the files by using the following javac command, as shown:

   ```
   javac *.java
   ```

5. To run the Contact List application, specify the following command at the command prompt:

   ```
   java ContactList
   ```

6. The Login window of the Contact List application appears. Enter the login information, as shown in Figure 7-9:

Figure 7-9: Specifying Login Information

7. Click the Submit button to send the login information to the Jabber Server for registered end users or open a new account in the Jabber server for unregistered end users.

8. The user interface of the Contact List application appears. Select File->Add User to add other end users to the contact list and send a welcome message to the added end users.

9. Enter the User ID, User Name, and a welcome text message to display to the new end users added to the contact list, as shown in Figure 7-10:

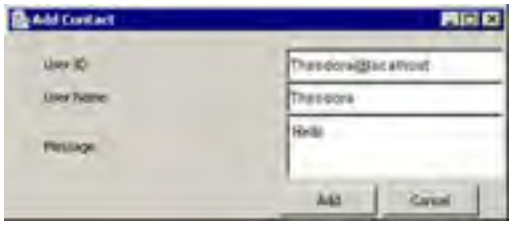


Figure 7-10: Adding an End User

10. Click the Add button to add the specified end user to the contact list and send the welcome text message to the added end user, as shown in Figure 7-11:

Figure 7-11: Updated Contact List

11. Select File->Manage User to manage the contact list.

12. Select any end user name from the tree structure of the of the contact list. The user information of the selected end user appears, as shown in Figure 7-12:

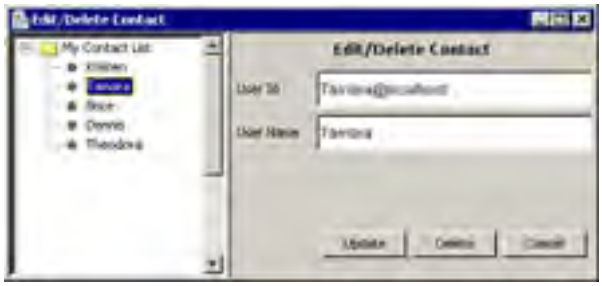


Figure 7-12: Managing a Contact

13. Modify the information and click the Update button. The information gets updated, as shown in Figure 7-13:

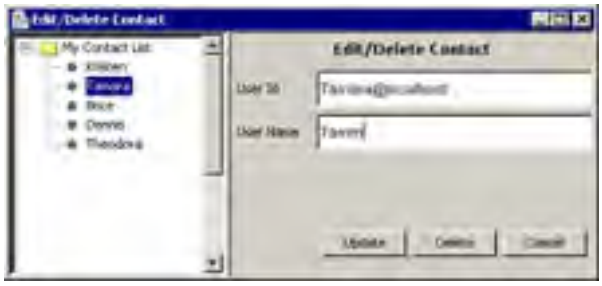


Figure 7-13: Updated Information in the Contact List

Index

A-I

airline schedules, Chapter 6: Creating an Airline Reservation Application
application, Chapter 7: Creating a Contact List Application
chat room, Chapter 3: Creating a Chat Room Application
Ejabberd, Chapter 2: Creating a Connector Application
Extensible Messaging and Presence Protocol, Chapter 2: Creating a Connector Application
Group Chatting application, Chapter 4: Creating a Group Chatting Application
IM, Chapter 1: Introduction
Instant Messaging, Chapter 1: Introduction

Index

J-P

Jabber protocol, Chapter 3: Creating a Chat Room Application, Chapter 4: Creating a Group Chatting Application

Jabber server, Chapter 1: Introduction, Chapter 2: Creating a Connector Application, Chapter 3: Creating a Chat Room Application, Chapter 4: Creating a Group Chatting Application, Chapter 7: Creating a Contact List Application

Jabber Software Foundation, Chapter 1: Introduction

JSF, Chapter 1: Introduction

LDAP, Chapter 2: Creating a Connector Application

Lightweight Directory Access Protocol, Chapter 2: Creating a Connector Application

online, Chapter 7: Creating a Contact List Application

protocol, Chapter 3: Creating a Chat Room Application, Chapter 6: Creating an Airline Reservation Application, Chapter 7: Creating a Contact List Application

Index

R-X

reservations, Chapter 6: Creating an Airline Reservation Application
server, Chapter 6: Creating an Airline Reservation Application
XML, Chapter 2: Creating a Connector Application
XMPP, Chapter 2: Creating a Connector Application

List of Figures

Chapter 2: Creating a Connector Application

Figure 2-1: Architecture of the Connector Application

Figure 2-2: User Interface of the Connector Application

Figure 2-3: File Menu of the Connector Application

Figure 2-4: The Logoff Confirmation Message

Figure 2-5: The Unsubscribe Confirmation Message

Figure 2-6: The Sign Up Window

Figure 2-7: The Login Window

Figure 2-8: The Message Window

Figure 2-9: The Enter Chat Room Window

Figure 2-10: Specifying Information in the Sign Up Window

Figure 2-11: The Signup Confirmation

Figure 2-12: Specifying Information in the Login Window

Figure 2-13: The Login Confirmation Message

Figure 2-14: Specifying the Message Text

Figure 2-15: Message Confirmation

Figure 2-16: Specifying the Chat Room Name

Figure 2-17: Enter Chat Room Confirmation

Chapter 3: Creating a Chat Room Application

Figure 3-1: Architecture of the Chat Room Application

Figure 3-2: The Login Window

Figure 3-3: User Interface of the Chat Room Application

Figure 3-4: The File Menu of the Chat Room Application

Figure 3-5: Joining a Chat Room

Figure 3-6: Creating a Chat Room

Figure 3-7: Creating a Group Chat Window

Figure 3-8: Specifying Login Information

Figure 3-9: Entering a Chat Room

Figure 3-10: The Group Chat Window of the Specified Chat Room

Figure 3-11: Entering Chat Room Name and User Name

Figure 3-12: The Group Chat Window of the New Chat Room

Figure 3-13: Sending a Message

Figure 3-14: Selecting an Existing Room

Figure 3-15: The Input Dialog Box

Figure 3-16: The Group Chat Window of the Selected Chat Room

Chapter 4: Creating a Group Chatting Application

Figure 4-1: Architecture of the Group Chatting Application

Figure 4-2: The Login Window

Figure 4-3: User Interface of the Group Chatting Application

Figure 4-4: The File Menu of the Group Chatting Application

Figure 4-5: Adding an End User

Figure 4-6: Renaming a Group

Figure 4-7: Private Chat Window

Figure 4-8: The Group Chat Window

Figure 4-9: Creating a Group Tree

Figure 4-10: Specifying Login Information

Figure 4-11: Adding an End User to a Group

Figure 4-12: Entering the Old and New Group Names

Figure 4-13: Renamed Group

Figure 4-14: Sending a Private Message

Figure 4-15: Sending a Group Message

Chapter 5: Creating an Instant Technical Support Application

Figure 5-1: Architecture of the Instant Technical Support Application

Figure 5-2: The Home Page

Figure 5-3: The Signup Page

Figure 5-4: The Login Page

Figure 5-5: User Interface of the Instant Technical Support Application

Figure 5-6: The Signup Information

Figure 5-7: Specifying the Query Text

Figure 5-8: Sending the Query

Figure 5-9: Receiving the Response for the Specified Query

Chapter 6: Creating an Airline Reservation Application

Figure 6-1: Architecture of the Airline Reservation Application

Figure 6-2: The Personal Information Page

Figure 6-3: User Interface of the Airline Reservation Application

Figure 6-4: Showing the Airline Schedules

Figure 6-5: Confirmation Message

Figure 6-6: Entering Personal Information

Figure 6-7: Searching for an Airline Schedule

Figure 6-8: Reserving an Airline Schedule

Chapter 7: Creating a Contact List Application

Figure 7-1: Architecture of the Contact List Application

Figure 7-2: The Login Window

Figure 7-3: User Interface of the Contact List Application

Figure 7-4: The File Menu of the Contact List Application

Figure 7-5: Adding End Users

Figure 7-6: Managing the Contact List

Figure 7-7: The Chat Window

Figure 7-8: The Contact Information Window

Figure 7-9: Specifying Login Information

Figure 7-10: Adding an End User

Figure 7-11: Updated Contact List

Figure 7-12: Managing a Contact

Figure 7-13: Updated Information in the Contact List

◄ PREVIOUS NEXT ►

List of Listings

Chapter 2: Creating a Connector Application

Listing 2-1: The JabberClient.java File

Listing 2-2: The SignUp.java File

Listing 2-3: The UserLogin.java File

Listing 2-4: The SocketClass.java File

Listing 2-5: The MessageClass.java File

Listing 2-6: The ChatRoom.java File

Chapter 3: Creating a Chat Room Application

Listing 3-1: The LoginGUI.java File

Listing 3-2: The MainGUI.java File

Listing 3-3: The ChatRoom.java File

Listing 3-4: The RoomInformation.java File

Listing 3-5: The SocketOpener.java File

Listing 3-6: The GroupChat.java File

Chapter 4: Creating a Group Chatting Application

Listing 4-1: The GroupLoginGUI.java File

Listing 4-2: The GroupList.java File

Listing 4-3: The AddUser.java File

Listing 4-4: The EditGroup.java File

Listing 4-5: The ChatWindow.java File

Listing 4-6: The GroupChat.java File

Listing 4-7: The SocketConnection.java File

Listing 4-8: The GroupTree.java File

Chapter 5: Creating an Instant Technical Support Application

Listing 5-1: The welcome.html File

Listing 5-2: The signuppage.html File

Listing 5-3: The loginpage.html File

Listing 5-4: The chathtml.html File

Listing 5-5: The ChatMainApplet.java File

Chapter 6: Creating an Airline Reservation Application

Listing 6-1: The UserInformation.html File

Listing 6-2: The Reservation.html File

Listing 6-3: The ReservationStatus.java File

Listing 6-4: The SocketClass.java File

Listing 6-5: The SocketConnector.java File

Listing 6-6: The ReservationResultTable.java File

Listing 6-7: The XMLReader.java File

Listing 6-8: The airlines.xml File

Chapter 7: Creating a Contact List Application

Listing 7-1: The UserLogin.java File

Listing 7-2: The ContactList.java File

Listing 7-3: The AddUser.java File

Listing 7-4: The EditDelete.java File

Listing 7-5: The ChatWindow.java File

Listing 7-6: The SocketClass.java File

Listing 7-7: The DynamicTree.java File

Listing 7-8: The UserInfo.java File

PREVIOUS NEXT

CD Content

Following are select files from this book's Companion CD-ROM. These files are copyright protected by the publisher, author, and/or other third parties. Unauthorized use, reproduction, or distribution is strictly prohibited.

File	Description	Size
All CD Content	Java InstantCode: Developing Applications Using Jabber	307,608
Chapter 2:	Creating a Connector Application	47,592
Chapter 3:	Creating a Chat Room Application	50,479
Chapter 4:	Creating a Group Chatting Application	62,588
Chapter 5:	Creating an Instant Technical Support Application	25,080
Chapter 6:	Creating an Airline Reservation Application	55,686
Chapter 7:	Creating a Contact List Application	66,293

Java InstantCode: Developing Applications Using Jabber

SkillSoft Press © 2004

This code-rich reference includes many applications, such as instant technical support, airline reservation, group chatting, contact list, and chat room applications.

Table of Contents

Introduction
Copyright
Chapter 1 - Introduction
Chapter 2 - Creating a Connector Application
Chapter 3 - Creating a Chat Room Application
Chapter 4 - Creating a Group Chatting Application
Chapter 5 - Creating an Instant Technical Support Application
Chapter 6 - Creating an Airline Reservation Application
Chapter 7 - Creating a Contact List Application
Index
List of Figures
List of Listings
CD Content