
**Java InstantCode: Developing Applications Using Java NIO**

SkillSoft Press © 2004

Learn about the New I/O (NIO) API introduced in J2SDK v 1.4 that provides new features and improved performance in the areas of polling, buffer management, character converters, and much more.

**Table of Contents**

# 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

This book describes the New I/O (NIO) API introduced in J2SDK v 1.4 that provides new features and improved performance in the areas of polling, buffer management, scalable network and advanced file system I/O, character converters, and regular-expression matching. It also explains how the NIO API supplements the existing I/O facilities. In addition, it describes how to implement these new features to improve the efficiency of Java applications.

## About the Author

Vishal Jayaswal holds a Bachelor's degree in IT Engineering. He is proficient in technologies such as C++, Java, HTML, DHTML, J2EE, EJB, JNDI, JMS, and RMI and has been involved in technical writing for NIIT in the same areas. In addition, he has written books on Data Mining and Java NIO. He has a sound knowledge of Adobe Photoshop and DreamWeaver.

## Credits

I would like to thank Gaurav Bhatla, Reena Roy, and S. Sripriya, for helping me complete the book on time. I also thank the editors and the quality assurance team for their timely help.

# Copyright

Java InstantCode: Developing Applications Using Java NIO


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

information@skillsoft.com

# Chapter 1: Introduction to New Input/Output API

Java2 Platform, Standard Edition (J2SE) provides a New Input/Output (NIO) API that provides features for polling, buffer management, scalable network and advanced I/O file system, character conversion, and regular-expression matching. Development of NIO API was proposed in Java Specification Request (JSR) 51. JSR contains description of the proposal to develop new specifications for the for the Java platform. Using NIO API, you can create applications that can search for regular expressions in a file, read and write data in a file, transfer data over a network, and encode and decode characters. In addition, NIO API provides support for scalable I/O operations for sockets and files, such as file locking and memory mapping.

NIO API consists of the java.nio, java.nio.channels, java.nio.channels.spi, java.nio.charset, and java.nio.charset.spi packages.

This chapter explains the packages provided by NIO API, along with the classes included in each package. This chapter also explains the commonly used methods provided by each class.

## The java.nio Package

The java.nio package contains various classes and methods for buffer management. You can use the java.nio classes to store the content of a file in a buffer in the form of bytes and characters. Buffer is a container to store fixed amount of specific primitive type data, such as byte, character, integer, and float.

> **Note** To learn more about java.nio package, see: http://java.sun.com/j2se/1.4.2/docs/api/java/nio/package-summary.html.

The various attributes of a buffer are:

- `capacity`: Represents the number of elements stored in a buffer.
- `limit`: Represents the index of the first element stored in the buffer. You cannot perform read and write operation on the limit. The limit of a buffer is always positive and greater than the capacity.
- `position`: Represents the valid index position in the buffer for performing read and write operation. The position is always positive and greater than the limit.

Figure 1-1 shows the class diagram for the java.nio package:

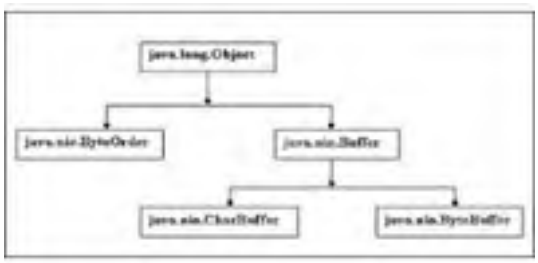

**Figure 1-1:** The java.nio Package

### Buffer

The Buffer class is an abstract class that provides common methods to all its subclasses, such as ByteBuffer, ByteOrder, and CharBuffer. The various methods declared in the Buffer class are:

- `capacity()`: Returns an integer that specifies the storage capacity of the buffer.
- `clear()`: Clears the buffer. This method makes the buffer ready for a new sequence of channel read or put operation. The clear() method also returns a reference to buffer objects.
- `flip()`: Sets the buffer to empty state from a fill state and makes a buffer ready for channel-write or get operation.
- `hasRemaining()`: Returns a Boolean value to indicate whether or not any element exists between the current position and the limit. The method returns true if any element exists in the buffer.
- `isReadOnly()`: Returns a Boolean value that indicates whether or not the buffer is read-only. The method returns true if the buffer is read-only.
- `limit()`: Returns an integer value that indicates the buffer limit. This method throws the IllegalArgumentException exception that occurs when an illegal argument is passed within the limit() method.
- `mark()`: Sets the mark of a buffer at its position. A mark is the index to which the position of a buffer is reset.
- `position()`: Returns an integer that indicates the current position of the buffer.
- `remaining()`: Returns an integer that indicates the number of elements between the current position and limit of the buffer.
- `reset()`: Resets the current position of a buffer to the previously-marked position. This method throws the InvalidMarkException exception that occurs when the mark is not defined in the buffer.
- `rewind()`: Reverses the order of the data in the buffer. The position is set to zero and discards the mark of the

buffer.

### ByteBuffer

The ByteBuffer class represents a byte buffer. This is a subclass of the Buffer class. The most commonly used methods in the ByteBuffer class are:

- `allocate()`: Returns the ByteBuffer class object. The allocate() method allocates a new byte buffer. You need to specify the size of the new byte buffer.
- `array()`: Returns a byte array. The returned array backs up the buffer. If you change the content of the buffer, the content in the returned array is also changed. This method throws two exceptions, ReadOnlyBufferException and UnsupportedOperationException. The ReadOnlyBufferException exception occurs when the put() or compact() method is invoked on a read-only buffer. The UnsupportedOperationException exception occurs when the requested operation is not supported by the Java Virtual Machine (JVM).
- `arrayOffset()`: Returns an integer value that specifies the offset within the backup array of the buffer. This method also throws two exceptions, ReadOnlyBufferException and UnsupportedOperationException.
- `asCharBuffer()`: Returns an object of the CharBuffer class. The asCharBuffer() method creates a view of the byte buffer as a char buffer.
- `asDoubleBuffer()`: Returns an object of the DoubleBuffer class. This method creates a view of the byte buffer as a double buffer.
- `asFloatBuffer()`: Returns an object of the FloatBuffer class. This method creates a view of the byte buffer as a float buffer.
- `asIntBuffer()`: Returns an object of the IntBuffer class. The asIntBuffer() method creates a view of the byte buffer as int buffer.
- `asLongBuffer()`: Returns an object of the LongBuffer class. This method creates a view of the byte buffer as a long buffer.
- `asReadOnlyBuffer()`: Returns an object of the ByteBuffer class. This method creates a new read-only byte buffer that shares the content of the byte buffer with another byte buffer.
- `asShortBuffer()`: Returns an object of the ShortBuffer class. This method creates a view of the byte buffer as a short buffer.
- `compact()`: Returns a ByteBuffer object. The compact() method compacts the buffer by copying the bytes between the current position and limits to the beginning of the buffer. This method throws a ReadOnlyBufferException, if the buffer is read-only.
- `compareTo()`: Compares the object of the ByteBuffer class to another object of another buffer and returns an integer value: positive, negative, or zero. A positive integer indicates that the buffer is greater than the specified buffer. A negative integer indicates that the buffer is less than the specified buffer whereas zero indicates that the buffer and the specified buffer are equal. The compareTo() method throws a ClassCastException if the object passed is not a byte buffer.
- `duplicate()`: Returns an object ByteBuffer class. The duplicate() method creates a new byte buffer and shares the content of the buffer with the newly created byte buffer.
- `getChar()`: Returns a char type value at the current position of the buffer. The getChar() method reads the next two bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by two. The getChar() method throws the BufferUnderflowException exception if there are less than two bytes in the buffer.
- `getDouble()`: Returns a double value at the current position of the buffer. The getDouble() method reads the next eight bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by eight. The getDouble() method throws the BufferUnderflowException exception if there are less than eight bytes in the buffer.
- `getFloat()`: Returns a float type value at the current position of the buffer. The getFloat() method reads the next four bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by four. The getFloat() method throws the BufferUnderflowException exception if there are less than four bytes in the buffer.
- `getInt()`: Returns an int type value at the current position of the buffer. The getInt() method reads the next four bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by four. The getInt() method throws the BufferUnderflowException exception if there are less than four bytes in the buffer.
- `getLong()`: Returns a long type value at the current position of the buffer. The getLong() method reads the next eight bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by eight. The getLong() method throws the BufferUnderflowException exception if there are less than 8 bytes in the buffer.
- `getShort()`: Returns a short type value at the current position of the buffer. The getShort() method reads the next two bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by two. The getShort() method throws the BufferUnderflowException exception if there are less than two bytes in the buffer.
- `putChar()`: Returns an object of the ByteBuffer class. The putChar() method writes two bytes at the current

position into the buffer and then increments the position by two. The putChar() method throws the BufferOverflowException exception if there are less than two bytes left in the byte buffer. In addition, the putChar() method throws the ReadOnlyBufferException exception if the buffer is read-only.

- `putDouble()`: Returns an object of the ByteBuffer class. The putDouble() method writes eight bytes that contain the specified char value into the buffer at the current position, and then increments the position by eight. The putDouble() method throws the BufferOverflowException exception if there are less than 8 bytes left in the byte buffer. In addition, the putDouble() method throws the ReadOnlyBufferException exception if the buffer is read-only.
- `putFloat()`: Returns an object of the ByteBuffer class. The putFloat() method writes four bytes that contain the specified char value into the buffer at the current position, and then increments the position by four. The putFloat() method throws the BufferOverflowException exception if there are less than four bytes left in the byte buffer. In addition, the putFloat() method throws a ReadOnlyBufferException, if the buffer is read-only.
- `putInt()`: Returns an object of the ByteBuffer class. The putInt() method writes four bytes that contain the specified char value into the buffer at the current position, and then increments the position by four. The putInt() method throws the BufferOverflowException exception if there are less than four bytes left in the byte buffer. In addition, the putInt() method throws a ReadOnlyBufferException, if the buffer is read-only.

### ByteOrder

The ByteOrder class defines the constants that determine the type of byte order to use when storing or retrieving byte values from a buffer. This class acts as a type-safe enumeration classes. Enumerations help create named values. The ByteOrder class defines two static values that denote byte order:

- `BIG_ENDIAN`: Denotes the BIG_ENDIAN byte order. The bytes of a multibyte value are ordered from the most significant to least significant.
- `LITTLE_ENDIAN`: Denotes the LITTLE_ENDIAN byte order. The bytes of a multibyte value are ordered from the least significant to most significant.

The most commonly used method in the ByteOrder class is the nativeOrder() method. This method returns the native byte order of the hardware on which JVM is running. Using the nativeOrder() method, the performance-sensitive Java code can allocate direct buffers with the same byte order as the hardware. The nativeOrder() method is a static method.

### CharBuffer

The CharBuffer class represents a character buffer. The CharBuffer class provides methods to copy the String type variables into CharBuffer. All these methods return the CharBuffer object. The most commonly used methods in the CharBuffer class are:

- `allocate()`: Creates a buffer object and allocates space to hold data elements. You also need to specify the size of the buffer to hold data elements when you invokes the allocate() method. This method throws two exceptions, ReadOnlyBufferException and UnsupportedOperationException.
- `wrap()`: Wraps a character array into a buffer.
- `slice()`: Creates a new character buffer. The content of the new buffer starts from the current position of the buffer.
- `duplicate()`: Creates a new buffer. The new buffer shares the content of the buffer.
- `asReadOnlyBuffer()`: Creates a new read-only character buffer. The new character buffer shares the content of the buffer.
- `get()`: Reads the specified character from the current position of the buffer, and then increments the position by one. The method throws the BufferUnderFlowException exception if the current position of the buffer is smaller than the limit of the buffer.
- `put()`: Writes the specified character into the current position of the buffer, and then increments the position by one. This method throws two exceptions, BufferOverflowException and ReadOnlyBufferException.
- `hasArray()`: Returns a Boolean value. The hasArray() method indicates whether or not the given buffer is copied by the character array.

# The java.nio.channels Package

The java.nio.channels package defines channels. The channels represent connections to entities that can perform I/O operations. The various entities are files, sockets server socket, and selectors. The entities use the channels to perform I/O operations, such as read and write. In addition, this package helps you create a non-blocking server. A non-blocking server handles multiple client requests without blocking them. A non-blocking server is created using the selector class of NIO. A selector is a selectable channel, which is a special type of channel that can be put in a non-blocking mode. The selectors process all the client requests and send the requests to the server.

**Note** To learn more about java.nio.channels package, see:
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/package-summary.html

Figure 1-2 shows the class diagram for the java.nio.channels package:

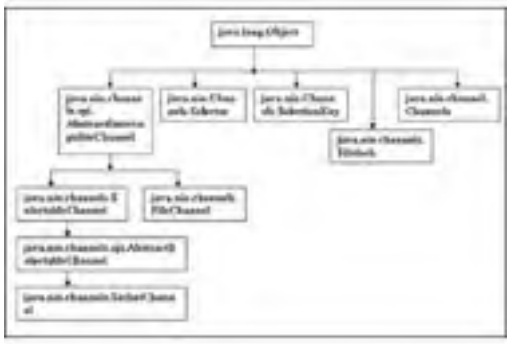

**Figure 1-2:** The java.nio.channels Package

### Channel

The Channel class contains utility methods to work with channels and streams. The most commonly used methods in the Channel class are:

- `newChannel()`: Returns a new readable byte channel value. The newChannel() method constructs a channel that can read bytes from a given stream.
- `newInputStream()`: Returns the InputStream object. The newInputStream() method constructs a stream that can read bytes from a given channel.
- `newReader()`: Returns the Reader object. The newReader() method constructs a reader that decodes the bytes. The reader uses a decoder to decode bytes from a given channel.
- `newWriter()`: Returns the Writer object. The newWriter() method constructs a writer that encodes characters. The writer uses an encoder to encode the characters and writes the resulting bytes to a given channel.

### FileChannel

The FileChannel class provides a channel to read, write, map, and manipulate a file. The FileChannel class defines methods that can read and write bytes from a specific location in a file. The most commonly used methods in the FileChannel class are:

- `force()`: Updates the files that use the channel. The force() method also writes a back up file to the storage device. The force() method throws the ClosedChannelException exception if the channel is closed. In addition, the force() method throws the IOException exception if some other types of input/output errors occur.
- `lock()`: Returns an object of the FileLock class. The lock() method acquires a lock on a specific portion of a file or a complete file that uses the channel.
- `map()`: Returns an object of the MappedByteBuffer class. The map() method maps an area of a file of the channel into the memory. The mapping can be read-only, read/write, or private. The method throws the NonReadableChannelException exception if the source channel is not open to read but the mode is READ_ONLY. The method throws the NonWritableChannelException exception if the channel is not open to write and the mode is READ_WRITE or PRIVATE.
- `read()`: Returns an integer value that indicates the number of bytes read. The read() method reads bytes from the channel into a given buffer. In addition, the read() method throws the NonReadableChannelException exception if the channel is not open to read. The method may throw the ClosedChannelException exception if the channel is closed.
- `size()`: Returns a long value that indicates the size of the file in the channel. The size() method throws the ClosedChannelException exception if the channel is closed. In addition, the size() method throws the IOException exception if some input/output error occurs.
- `transferFrom()`: Returns a long type value. The transferFrom() method transfers bytes into a file of the channel. In addition, the transferFrom() method throws a NonReadableChannelException exception if the source channel is not open to read. The method may throw the NonWritableChannelException exception if the channel is not open for performing write operation.

- `transferTo()`: Returns a long type value. The transferTo() method transfer bytes from a file of the channel to a given writable byte channel. In addition, the transferTo() method throws the NonReadableChannelException exception if the source channel is not open to read. The method may throw the NonWritableChannelException exception if the channel is not open for performing write operation.
- `write()`: Returns an int value that indicates the number of bytes written to the specified file. The bytes are written to the file specified The write() method writes bytes from the channel into a given buffer. In addition, the write() method throws the NonWriteableChannelException exception if the channel is not open to write. The method can also throw the ClosedChannelException exception if the channel is closed.

### FileLock

The FileLock class contains methods that perform file lock operations on a file. This class works as a token that represents a lock on a portion of a file. You can acquire a lock on a file in exclusive or shared mode. The most commonly used methods in the FileLock class are:

- `channel()`: Returns an object of the FileChannel class.
- `isShared()`: Returns a Boolean value that indicates whether or not the type of lock is shared.
- `isValid()`: Returns a Boolean value that indicates whether or not the lock is valid.
- `position()`: Returns a long value that indicates the first byte of the locked region.
- `release()`: Releases a lock from the file.
- `size()`: Returns a long value that indicates the total number of bytes in the locked region.

### Selector

The Selector class represents a multiplexor of the SelectableChannel class object. Moreover, the Selector class manages the information about the registered channels and their respective states. When a channel registers with the selector, the selector is responsible to update the state of a channel. The Selector class uses the service provider classes of the java.nio.channels.spi package to create a new selector. The most commonly used methods of the Selector class are:

- `isOpen()`: Returns a Boolean value that indicates whether or not the selector is open. The isOpen() method returns true if the selector is open.
- `keys()`: Returns an object of the Set interface that indicates the key set of the selector. The Set interface represents a collection that contains no duplicate elements. This method throws the ClosedSelectorException exception. This exception occurs when the selector on which you perform I/O operation is closed.
- `open()`: Returns an object of the Selector class. The open() method opens a selector. In addition, the open() method throws the IOException exception if an input/output error occurs.
- `provider()`: Returns an object of the SelectorProvider class. The SelectorProvider class is a provider that creates the channel.

### SocketChannel

The SocketChannel class represents a selectable channel. The selectable channel is a type of channel that connects a selector to a stream-oriented socket using TCP/IP. The SocketChannel class contains methods to work with the socket channel. The most commonly used methods in the SocketChannel class are:

- `connect()`: Returns a Boolean value that indicates whether or not a connection is established. In addition, the connect() method returns true if a connection exists, otherwise the method returns false. The connect() method throws the ClosedChannelException exception if the channel is closed. In addition, the connect() method throws the AlreadyConnectedException exception if the channel is already connected.
- `finishConnect()`: Returns a Boolean value that indicates whether or not the process to connect a socket channel is complete. In addition, the finishConnect() method returns true if the socket of the channel is connected. The method throws a ClosedChannelException exception if the channel is closed. In addition, the finishConnect() method throws an IOException exception if any input/output error occurs.
- `isConnected()`: Returns a Boolean value that indicates whether or not the network socket of a channel is connected to the server. In addition, the isConnected() method returns true if the network socket of a channel is connected, otherwise the method returns false.
- `open()`: Returns an object of the SocketChannel class. In addition, the connect() method opens a socket channel. This method also throws an IOException exception, if any input/output error occurs.
- `read()`: Returns an int value that indicates the number of bytes read. The `read()` method reads a series of bytes from the channel in a buffer. This method also throws an IOException exception.
- `socket()`: Returns an object of the Socket class. The socket() method retrieves a socket from the associated channel.
- `write()`: Returns an int value that indicates the number of bytes written. The `write()` method writes a series of bytes to the channel from a given buffer. In addition, the write() method throws the NotYetConnectedException exception if the channel is not connected. The write() method throws the IOException exception if any input/output error occurs.

### SelectionKey

The SelectionKey class creates and manages a selection key. A selection key is a token that represents the registration of the SelectableChannel class with the Selector class. This token is created each time the objects of a channel get registered with a selector. The most commonly used methods in the SelectionKey class are:

- `attach()`: Returns an object of the Object class. The `attach()` method attaches an object to the selection key.
- `attachment()`: Returns an object of the Object class. The attachment() method retrieves the object that is currently attached to the selection key.
- `channel()`: Returns an object of the SelectChannel() class that indicates a selector for the selection key.
- `isAcceptable()`: Returns a Boolean value that indicates whether or not the selection key is ready to accept a new socket connection. The isAcceptable() method returns true if the readyOps() method and OP_ACCEPT field are greater than zero.
- `isReadable()`: Returns a Boolean value that indicates whether or not the selection key is ready to read. The isReadable() method returns true if readyOps() method and OP_READ field are greater than zero.
- `isValid()`: Returns a Boolean value that indicates whether or not the selection key is valid. The isValid() method returns true if the selection key is valid.
- `isWritable()`: Returns a Boolean value that indicates whether or not the selection key is ready to write. The isWritable() method returns true if readyOps() method and OP_WRITE field are greater than zero.

The SelectionKey class also defines four static variables:

- `OP_ACCEPT`: Represents the operation-set bit for socket-accept operations.
- `OP_CONNECT`: Represents the operation-set bit for socket-connect operations.
- `OP_READ`: Represents the operation-set bit for read operations.
- `OP_WRITE`: Represents the operation-set bit for write operations.



◄ PREVIOUS NEXT ►

# The java.nio.channels.spi Package

The java.nio.channels.spi package contains various service-provider classes and methods to create a new selector provider class.

**Note** To learn more about java.nio.channels.spi package, see:
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/spi/package-summary.html

The commonly used classes in the java.nio.channels.spi package are:

- `SelectProvider`
- `AbstractSelectot`

### SelectorProvider

The SelectorProvider class defines various methods to create datagram channel, socket channel, and selectable channels. The most commonly used methods in the SelectorProvider class are:

- `openDatagramChannel()`: Returns an object of the DatagramChannel class. The openDatagramChannel() method opens a datagram channel. The datagram channel is a type of channel that connects the selector to a stream-oriented socket using User Datagram Protocol (UDP).
- `openSelectors()`: Returns an object of the AbstractSelector class. The openSelectors() method opens a selector.
- `openSocketChannel()`: Returns an object of the SocketChannel class. The openSocketChannel() method opens a socket channel.
- `provider()`: Returns an object of the SelectorProvider class that contains a default selector provider. The provider() method is a static method.

### AbstractSelector

The AbstractSelector class works as a base implementation class for selectors. The AbstractSelector class defines methods that work with selectors. The most commonly used methods in the AbstractSelector class are:

- `begin()`: Identifies the beginning of the input/output operation.
- `end()`:Identifies the end of the input/output operation.
- `isOpen()`: Returns a Boolean value that indicates whether the selector is open or close. The isOpen() method returns true if the selector is open.
- `provider()`: Returns an object of the SelectorProvider class that creates the channel.
- `register()`: Returns an object of the SelectionKey class. The register() method registers the given channel with the selector.

# The java.nio.charset Package

The java.nio.charset package contains various classes and methods for character set conversion, regular expression matching, and for encoding and decoding data. This package provides the charset, encoder, and decoder classes to convert data between bytes and coded characters. The character data is encoded for transmission over a network or for storage in a file.

A character set is a set of characters, such as alphabets from A to Z, a to z, and special characters. The coded character set is an assignment of numeric value to each character in the character set using the standard encoding scheme. The encoding scheme is a process of mapping the coded character set to a sequence of Octets. A collection of 8 bytes is known as Octet.

The charset are the combination of coded character and encoding schemes. The various standard charsets are US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, and UTF-16.

> **Note** To learn more about java.nio.charset package, see: http://java.sun.com/j2se/1.4.2/docs/api/java/nio/charset/package-summary.html

Figure 1-3 shows the class diagram for the java.nio.charset package:

**Figure 1-3:** The java.nio.charset Package

### Charset

The Charset class defines methods to retrieve various names associated with a character set. In addition, the Charset class defines methods to create encoders and decoders. The most commonly used methods of the Charset class are:

- `aliases()`: Returns an object of the Set interface that indicates the aliases of a charset.
- `availableCharsets()`: Returns an object of the SortedMap interface that contains one entry for each charset supported by the current JVM. The availableCharsets() method develops a sorted map from canonical charset names to charset objects. The map contains one entry for each charset, which the current JVM supports.
- `decode()`: Returns an object of the CharBuffer class. The decode() method decodes bytes in the charset into Unicode characters.
- `displayName()`: Returns a string that represents a readable name.
- `encode()`: Returns an object of the ByteBuffer class. The encode() method encodes Unicode characters into bytes in the charset.
- `forName()`: Returns an object of the Charset class for the standard charset. In addition, the forName() method throws the IllegalCharsetNameException exception if the given charset name is illegal and the UnsupportedCharsetException exception if there is no support for the named charset.
- `hashCode()`: Returns an int value that indicates the calculated hashcode for the charset. The hashcode is an object that contains the hash id and hash state of the hash table.
- `newDecoder()`: Returns an object of the CharsetDecoder class. The newDecoder() method constructs a new decoder for the charset.
- `newEncoder()`: Returns an object of the CharsetEncoder class. The newEncoder() method constructs a new encoder for the charset.

### CharsetDecoder

The CharsetDecoder class defines methods that transform a sequence of bytes into a sequence of 16-bit Unicode characters. The source buffer is a byte buffer and the resultant buffer is a character buffer. The most commonly used methods in the CharsetDecoder class are:

- `charset()`: Returns an object of the Charset class that creates the decoder.
- `decode()`: Returns an object of the CoderResult class. The decode() method decodes the bytes from the source buffer and writes the resultant character in the target buffer. In addition, the decode() method throws the IllegalStateException exception if a decode operation is already in progress.
- `decodeLoop()`: Returns an object of the CoderResult class. The decodeLoop() method decodes one or more bytes into one or more characters.
- `flush()`: Returns an object of the CoderResult class. The flush() method clears the decoder.
- `isAutoDetecting()`: Returns a Boolean value. The isAutoDetecting() method indicates whether the decoder implements an auto-detecting charset.
- `isCharsetDetected()`: Returns a Boolean value that indicates whether or not the decoder has detected a

charset.

- `replacement()`: Returns a string value that indicates the replacement value of the decoder.
- `reset()`: Returns an object of the CharsetDecoder class. The reset() method resets the decoder.

### CharsetEncoder

The CharsetEncoder class defines methods that transform a sequence of 16-bit Unicode characters into a sequence of bytes. The source buffer is a character buffer, and the resultant buffer is a byte buffer. The most commonly used methods in the CharsetDecoder class are:

- `canEncode()`: Returns a Boolean value that indicates whether the encoder can encode the given character.
- `charset()`: Returns an object of the Charset class that creates the encoder.
- `encode()`: Returns an object of the CoderResult class. The encode() method encodes the characters from the source buffer and writes the resultant bytes in the target buffer. In addition, the encode() method throws the IllegalStateException exception if an encode operation is already in progress.
- `encodeLoop()`: Returns an object of the CoderResult class. The encodeLoop() method encodes one or more bytes into one or more characters.
- `flush()`: Returns an object of the CoderResult class. The flush() method flushes the encoder.
- `replacement()`: Returns a string value that indicates the replacement value of the encoder.
- `reset()`: Returns an object of CharsetEncoder class. The reset() method resets the encoder.

### CoderResult

The CoderResult class defines the result state of a coder. A coder consumes bytes or characters from an input buffer, translates them, and writes the resulting characters or bytes to an output buffer. The translation process can be stopped due to various reasons, such as underflow, overflow, malformed-input error, and unmappable-character error. A coder can be an encoder or decoder. The most commonly used methods in the CoderResult class are:

- `isError()`: Returns a string value that describes a coder result.
- `isMalformed()`: Returns a Boolean value that indicates whether the object describes a malformed-input error. The malformed-input error occurs when an input byte sequence is not legal for a specified charset. The isMalformed() method returns true if an object indicates malformed-input error else returns false.
- `isOverFlow()`: Returns a Boolean value that indicates whether the object describes overflow condition. An overflow condition occurs when the buffer that stores the decoded data reaches the buffer limit. The isOverFlow() method returns true if the object indicates overflow, otherwise the method returns false.
- `isUnderFlow()`: Returns a Boolean value that indicates whether the object describes underflow condition. An underflow condition occurs when the buffer that stores the decoded data is empty. The isUnderFlow() method returns true if the object indicates underflow, otherwise the method returns false.
- `isUnmappable()`: Returns a Boolean value that indicates whether the object describes an unmappable-character error. The unmappable-character error occurs when an input character or byte sequence is valid but cannot map to an output byte or character sequence. The isUnmappable() method returns true if the object indicates unmappable-character error, otherwise the method returns false.



PREVIOUS NEXT

# The java.nio.charset.spi Package

The java.nio.charset.spi package contains a service-provider class to create a new charset class. This package contains only one class, CharSetProvider.

**Note** To learn more about java.nio.charset.spi package, see:
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/charset/spi/package-summary.html

### CharSetProvider

The CharSetProvider class is a subclass of the Charset class. This class takes no arguments in the default constructor. You can look up the Charset provider with the help of the loaded context class of the current thread. The methods used in the CharSetProvider class are:

- `charsetForName()`: Returns an object of the Charset class. The charsetForName() method takes charsetName of String type as an input parameter.
- `charsets()`: Returns a collection of objects of the Charset class supported by the provider.

# Chapter 2: Creating a Chat Application

The New Input/Output (NIO) Application Programming Interface (API) supports the java.nio and java.nio.channels packages for handling advance I/O file system and sockets, multiplexing, and non-blocking data transfer between the client and the server. The java.nio package contains ByteBuffer classes that allow you to store data in bytes. The java.nio.channels package provides various classes, such as FileChannel, SelectionKey, Selector, ServerSocketChannel, and SocketChannel. You can use these classes to initialize a socket, connect the socket to the server, and transfer data using this socket.

This chapter explains how to develop a Chat application using the java.nio and java.nio.channels packages.

## Architecture of the Chat Application

Using the Chat application, an end user can send private and broadcast messages. A private message is sent to an individual end user, while a broadcast message is sent to all the end users who are connected to the chat server.

The Chat application contains two folders, Server and Client.

The Server folder contains the following files:

- `ChatServer.java`: Initializes all _ the chat server classes. This is the main class of the chat server.
- `UAServer_Socket.java`: Creates, _ registers, and maps a socket to broadcast a message to all the end users connected in a chat session.
- PRServer_Socket.java: Creates, registers, and maps a socket to send private messages to a specified end user.
- `Msgbroadcast.java`: Broadcasts messages to all the end users connected in a chat session.
- `SocketCallback.java`: Uses sockets to communicate with the chat client.
- `AppendUserList.java`: Stores the names of all the end users connected in a chat session.

The Client folder contains the following files:

- `ChatLogin.java`: Creates the Chat Login window for the chat application.
- `ChatClient.java`: Creates the main window of the chat application. This window contains a text pane and a list box that display the messages and the user list.
- `CClient.java`: Implements the methods that are declared or called in the chat application client. This file connects the chat client to the chat server.
- `Messager.java`: Defines an abstract _ method to read and write the message to the text pane of the chat application client.

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

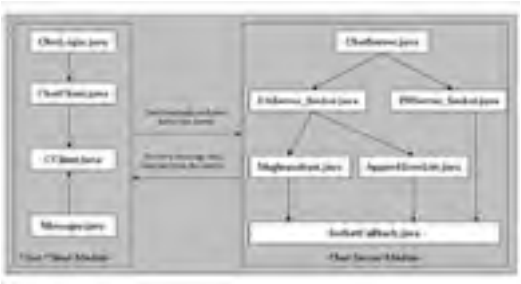

**Figure 2-1:** Architecture of the Chat Application

In the Chat application, there are two separate modules, chat server and chat client. In the chat server, the ChatServer.java file calls the UAServer_Socket.java file and PRServer_Socket.java file to create sockets to send the private and broadcast messages. The UAServer_Socket.java file calls the Msgbroadcast.java and AppendUserList.java files. Next, the Msgbroadcast.java file calls the SocketCallback.java file to broadcast the message to all the chat users. The AppendUserList.java file calls the SocketCallback.java file to add a new user to the user list. The PRServer_Socket.java file again calls the SocketCallback.java file to send the private message to a specific chat user.

In the chat client, the ChatLogin.java file calls the ChatClient.java file to open the chat client window. Next, ChatClient.java calls the CClient.java and Messager.java files. The CClient.java file establishes a connection between the client and the server to send and receive messages. The Messager.java file provides an abstract method, message(), for chat users.

# Creating the Chat Server

To create the chat server, you need to create the following files:

- ChatServer.java file
- UAServer_Socket.java file
- PRServer_Socket.java file
- Msgbroadcast.java file
- AppendUserList.java file
- SocketCallback.java file

### Creating the ChatServer.java File

The ChatServer.java file helps create a Command Line Interface (CLI)-based chat server. The chat server creates sockets to send and receive private and broadcast messages. To allow multiple users to connect to the chat server at the same time, you need to configure the chat server as non blocking.

Listing 2-1 shows the contents of the ChatServer.java file:

**Listing 2-1: The ChatServer.java File**

```
/*Imports java.net package class.*/
import java.net.*;
/Imports java.nio package class.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
/*Imports java.util package class.*/
import java.util.*;
/*Imports java.io package class.*/
import java.io.*;
/*
class ChatServer - Initializes all the chat server classes. This is the main class of the chat _
server.
   Field:
   SendAllPort: Stores the default port number.
   SendPrivatePort: Stores the default port number.
   UserList: Stores the list of users.
   Method:
   start_server(): Creates the objects of socket classes and starts the thread to run _
   the server.
   main(): Creates the object of the ChatServer class and call the Start_Server() _
   method.
*/
public class ChatServer
{
   /*Declares the port numbers.*/
   private int SendAllPort=9999;
   private int SendPrivatePort=8888;
   /*Creates and initializes the object of the ArrayList class.*/
   ArrayList UserList=new ArrayList();
   /*Defines the default constructor.*/
   public ChatServer(){}
   /*
   start_server() - This method is called to start the chat server.
   Parameters:   NA
   Return Value: NA
   */
   public void start_server()
   {
      /*Creates the object of the UAServer_Socket class.*/
      UAServer_Socket ua=new UAServer_Socket(SendAllPort,UserList);
      /*Creates the object of the PRServer_Socket class.*/
      PRServer_Socket p=new PRServer_Socket(SendPrivatePort,UserList);
      /*Initializes and starts a new thread to create the socket for broadcasting.*/
      new Thread(ua).start();
      /*Initializes and starts a new thread to create the socket for private messaging.*/
      new Thread(p).start();
      System.out.println("Server is started. Now waiting for client requests...");
   }
   /*
      main() - This method creates the main window of the user interface and displays it.
      Parameters:
      args[] - Contains any command line arguments passed.
      Return Value: NA
   */
   public static void main(String[] args)
   {
      /*Creates and initializes the objects of the ChatServer class.*/
```

```
      ChatServer cs=new ChatServer();
      /* Calls the Start_Server() method. */
      cs.start_server();
   }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the ChatServer class and calls the start_server() method. The start_server() method creates the object of the UAServer_Socket and PRServer_Socket classes. The start_server() method then initializes and starts a new thread to create the socket to send and receive private and broadcast messages.

### Creating the UAServer_Socket.java File

The UAServer_Socket.java file creates the socket connection to broadcast the messages. The UAServer_Socket class initializes and registers the broadcast socket to the chat server.

Listing 2-2 shows the contents of the UAServer_Socket.java file:

**Listing 2-2: The UAServer_Socket.java File**

```
/*Imports the java.net package class.*/
import java.net.*;
/*Imports the java.net package class.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
/*Imports the java.util package class.*/
import java.util.*;
/*Imports the java.io package class.*/
import java.io.*;
/*
Class UAServer_Socket - Creates the socket, registers and maps the socket to broadcast the _
message to all the end users connected in a chat session.
   Fields:
      Port - Contains the port number.
      srvr_sckt_chnel - Creates the server socket channel.
      sckt_manager - Manages the various sockets.
      selkey - Selects the appropriate socket.
      tdata - Stores the array of bytes.
      broadcastMessage - Stores the messages.
   Method:
      init_socket() - Initializes the private messaging socket.
      register_server() - Registers the server.
      accept_connection() - Accepts the connections from the server.
      read_Message() - Reads the message from the server socket.
      closeRemoteChannel() - Closes all the remote channels.
      write_Message() - Writes the messages to the server socket.
      got_connection() - Retrieves the connection.
*/
public class UAServer_Socket implements Runnable
{
   /*Declares the port for broadcasting.*/
   int port=9999;
   /*Declares the object of the ServerSocketChannel class.*/
   ServerSocketChannel srvr_sckt_chnel=null;
   /*Declares the object of the Selector class.*/
   Selector sckt_manager=null;
   /*Declares the object of the Iterator class.*/
   Iterator searcher=null;
   /*Declares the object of the SelectionKey class.*/
   SelectionKey selkey=null;
   /*Declares the objects of the ArrayList class.*/
   ArrayList broadcastMessage=new ArrayList();
   /*Declares the object of the ByteBuffer class and allocate the size to the byte buffer.*/
   ByteBuffer buffer=ByteBuffer.allocateDirect(1024);
   int buflen=0;
   byte[] tdata=null;
   int conId=0;
   int kichek =0;
   /*
   Defines the constructor that provides the port number as input.
   */
   public UAServer_Socket(int port)
   {
      this.port=port;
   }
   /* Defines the constructor that provides the port number and user list as input. */
   public UAServer_Socket(int port, ArrayList conlist)
   {
      this.port=port;
      broadcastMessage=conlist;
   }
   /* Defines the constructor that provides the user list as input. */
   public UAServer_Socket(ArrayList conlist)
```

```
{
   broadcastMessage=conlist;
}
/*
init_socket() - This method initializes the chat server.
   Parameter: NA
   Return Value: NA
*/
public void init_socket() throws Exception
{
   /*Opens the selector*/
   sckt_manager=SelectorProvider.provider().openSelector();
   /*Opens the server socket channel.*/
   srvr_sckt_chnel=ServerSocketChannel.open();
   /*Configures the server as NON Blocking.*/
   srvr_sckt_chnel.configureBlocking(false);
   /*Creates and initialize the object of the InetSocketAddress class.*/
   InetSocketAddress fulnetaddr=new InetSocketAddress(port);
   /*Binds the server socket.*/
   srvr_sckt_chnel.socket().bind(fulnetaddr);
   /*Calls the register()method.*/
   register_server(srvr_sckt_chnel,SelectionKey.OP_ACCEPT);
}
/*
   register_server() - This method registers the chat server.
   Parameter:
       Ssc: Represents the object of the ServerSocketChannel class.
       selectionkey_ops: Integer type that determines the selection option.
    Return Value: NA
*/
public void register_server(ServerSocketChannel ssc,int selectionkey_ops)throws Exception
{
   ssc.register(sckt_manager,selectionkey_ops);
}
/*
run() - This method calls the methods to initialize the socket and accept the connection of _
the sockets.
   Parameters: NA
   Return Value: NA
*/
public void run()
{
   try
   {
      /*Calls the initialization method.*/
      init_socket();
      /*Calls the method to accept the connection.*/
      accept_connection();
   }
   catch(Exception ej)
   {
      ej.printStackTrace();
   }
}
/*
accept_connection() - This method accepts the connection from the server.
Parameter: NA
Return Value: NA
*/
public void accept_connection()throws Exception
{
   while (true)
      {
         /*Calls the select() method of the Selector class.*/
         kichek=sckt_manager.select();
         if(kichek>=0)
         {
            /*
            Initializes the object of the Iterator class by calling the selectedKeys() methodof t
            */
            searcher=sckt_manager.selectedKeys().iterator();
            while(searcher.hasNext())
            {
               /*Gets the next element.*/
               selkey=(SelectionKey)searcher.next();
               /*Removes the item from the searcher.*/
               searcher.remove();
               /*When the message provider gets new connection from user, this section is _
               executed.*/
               if(selkey.isAcceptable())
               {
                  /*Creates and initializes the object of the ServerSocketChannel class to _
                  get the channel withthe help of the channel() method.*/
                  ServerSocketChannel server=(ServerSocketChannel)selkey.channel();
                  /* Calls the got_connection() method.*/
                  got_connection(server);
                  System.out.println("connection establish");
            }
```

```
/* When the selector provider gets the stream for reading by the server from end user, this _
section is executed.*/
else if(selkey.isReadable())
            {
               /*Calls the read_Message() method.*/
               read_Message((SocketCallback)selkey.attachment());
               }

/*When the selector Provider gets the stream for writing to user, this section is executed.*/
else if(selkey.isWritable())
               {
                  /*Creates the object of the SocketCallback class and call the attachment() _
                  method of the SelectionKey class.*/
                  SocketCallback  Vsc=(SocketCallback)selkey.attachment();
                  String m_essage="";
                  if (Vsc.getString().length()>0)
                  m_essage =Vsc.getString();
                  if(Vsc.get_broad_msg().length()>0)
                  m_essage +=Vsc.get_broad_msg();
                  /* Calls the write_Message(). */
                  write_Message(Vsc,m_essage);
                  }
                  }
               }
            else
            break;
                     }
               }
   /*
   read_Message() - This method reads the messages from the server socket.
      Parameter:
       sc: Represents the object of the SocketCallback class.
       Return Value:
       String: Returns a string value.
   */
   public String read_Message(SocketCallback sc)
   {
      try
      {
         if (sc.isValidUser())
         {
               /* Clears the buffer. */
            buffer.clear();
/*Reads the bytes from the channel. */
int nbyte=sc.getChannel().read(buffer);
            if(nbyte==-1)
            {
               /*Closes the channel. */
               closeRemoteChannel(sc);
               sc.getChannel().close();
               return "";
            }
            if (nbyte>0)
            {
               /*Flips the buffer.*/
               buffer.flip();
               /*Initializes the object of the Byte class.*/
               tdata=new byte[nbyte];
               /*Retrieves the data from the buffer.*/
               buffer.get(tdata,0,nbyte);
               /*Initializes the object of the Inner_checker class.*/
               String s=new String(tdata);
               if(tdata[0]==31)
               {
                  /*Initializes the object of the AppendUserList class.*/
                  new AppendUserList(sc,broadcastMessage);
                  return "";
               }
               else
               {
                  if (s.length()>0)
                  {
                  sc.addString(s);
                  /*Initializes the object of the Msgbroadcast class.*/
                  new Msgbroadcast(broadcastMessage,s,sc.getUserId());
                  return s;
                  }
               }
            }
         }
         else
         {
            /*Clears the buffer.*/
            buffer.clear();

            /*Reads the bytes from the channel.*/
            int
            nbyte=sc.getChannel().read(buffer);
```

```
            if(nbyte==-1)
            {
               /*Closes the channel.*/
               closeRemoteChannel(sc);
               sc.getChannel().close();
               return "";
            }
            if (nbyte>0)
              {
               /*Flips the buffer.*/
               buffer.flip();
               /*Initializes the object of the Byte class.*/
                           tdata=new byte[nbyte];
               /*Retrieves the data from the buffer.*/
               buffer.get(tdata,0,nbyte);
               /*Initializes the object of the Inner_checker class.*/
               new UAinnerChecker(sc,tdata,nbyte);
                           }
                     }
            }
      catch(IOException ex)
      {
         try
         {
            /*Calls the closeRemoteChannel() method.*/
                        closeRemoteChannel(sc);
                        /* Closes the channel. */
                        sc.getChannel().close();
                     }
                     catch(IOException ec)
                     {
                        ec.printStackTrace();
                     }
                  }
                  /*Returns a null string.*/
      return "";
      }
      /*
         closeRemoteChannel() - This method close the remote channel.
            Parameter:
            sCb: Represents the object of the SocketCallback class.
            Return Value: NA
   */
      public void closeRemoteChannel(SocketCallback sCb)
   {
      int i = broadcastMessage.indexOf(sCb);
      if (i >= 0)
      {
         /*Removes the broadcast message from the list.*/
         broadcastMessage.remove(i);
      }
   }
   /*
      write_Message() - This method writes the messages to the server socket.
         Parameter:
         sc: Represents the object of the SocketCallback class.
         msg: Stores the message.
         Return Value: NA
   */
   public void write_Message(SocketCallback sc,String msg)throws Exception
   {
      if (msg.length()>0)
      {
         /*Calls the doBroadcast() method.*/
         sc.doBroadcast();
         /*Creates the object of the ByteBuffer class. Next, wraps the bytes that are retrieved
         from the message.*/
         ByteBuffer b=ByteBuffer.wrap(msg.getBytes());
         /*Writes the byte buffer to the server socket.*/
         sc.getChannel().write(b);
         }
      }
   /*
   got_connection() - This method gets the connection from the server.
      Parameter:
      server: Represents the object of the ServerSocketChannel class.
      Return Value: NA
   */
   private void got_connection(ServerSocketChannel servr)throws Exception
   {
      /*Initializes the object of the SocketChannel class to accept the connection.*/
      SocketChannel sc=servr.accept();
      /*Sets the server to Non Blocking server.*/
      sc.configureBlocking(false);
      /*Registers the server socket with the selector.*/
      SelectionKey skey=sc.register(sckt_manager,SelectionKey.OP_READ|SelectionKey.OP_WRITE);
      /*Creates the object of the SocketCallback class.*/
      SocketCallback callbk=new SocketCallback(sc);
```

```
   skey.attach(callbk);
   }
/*
   finalize() - This method is invoked when message is read or sent.
   Parameter: NA
   Return Value: NA
*/
public void finalize() throws IOException
{
   /*Closes the server socket channel.*/
            srvr_sckt_chnel.close();
            /* Closes the selector. */
            sckt_manager.close();
}

/*
Inner Class: UAinnerChecker - This class checks the user validity.
   Methods:
   run(): Runs the started threads.
   Chk_Validity(): Checks the user validity.
   Chk_Valid_User(): Checks user validation.
*/
class UAinnerChecker extends Thread
{
   /* eclares the objects of the SocketCallback class.*/
   SocketCallback u=null;
   byte[] b=null;
   int len=0;
   /*Defines the constructor that takes the SocketCallback class object and integer type
   value as parameters.*/
   UAinnerChecker(SocketCallback call,byte[] b,int len)
   {
      this.u=call;
      this.b=b;
      this.len=len;
      start();
   }
   /*
      run() - This method is called when the thread is started to call the thread methods.
         Parameters:   NA
         Return Value: NA
   */
   public void run()
   {
      /*Calls the Chk_Validity() method.*/
      Chk_Validity();
   }
   /*
   Chk_Validity() - This method check the user validity.
      Parameter: NA
      Return Value: NA
   */
   private void Chk_Validity()
   {
      /*Declares the objects of the String class.*/
      String u_id="";
      String pwd="";
      String info="";
      /* Declares the FLAGS. */
      boolean id_flag=true;
      boolean pwd_flag=true;
      try
      {
         for (int x=0;x<len;x++ )
         {
            if(id_flag)
            {
               /*Checks the user name separator "28" in the string.*/
               if (b[x]==28)
               {
               id_flag=false;
               }
               else
               u_id+=(char)b[x];
            }
            else if(pwd_flag)
            {
               /*Checks the password separator "29" in the string.*/
               if (b[x]==29)
               {
               pwd_flag=false;
               }
               else
               pwd+=(char)b[x];
           }
           else
           info+=(char)b[x];
         }
```

```
/*Checks the chat user is valid or not.*/
if(Chk_Valid_User(u_id, pwd))
{
   /*Calls the setUserId() method of the SocketCallback class.*/
   u.setUserId(u_id);
   /*Calls the setPwd() method of the SocketCallback class.*/
   u.setPwd(pwd);
   /*Calls the setUinfo() method of the SocketCallback class.*/
   u.setUinfo(info);
   /*Calls the ValidUser() method of the SocketCallback class.*/
   u.ValidUser(true);
   /*Increments the counter by one. */
   conId++;
   /*Calls the setConid() method of the SocketCallback class.*/
   u.setConid(conId);
   broadcastMessage.add(u);
}
else
{
   /*Closes the channel.*/
   u.getChannel().close();
}
}
catch(Exception e3)
         {
            e3.printStackTrace();
         }
      }
      /*
         Chk_Valid_User() - This method checks the user is valid or not.
            Parameter:
            Uid: Stores the user name.
            pass: Stores the password.
            Return Value: NA
      */
public boolean Chk_Valid_User(String Uid,String pass)
      {
           return true;
        }
   }
}
```

Download this Listing.

In the above code, the UAServer_Socket class creates a socket on port 9999 that allows end users to send and receive broadcast messages and an updated user list. The methods defined in this code are:

- `run()`: Calls the init_socket() method to initialize the socket and calls the accept_connection() method to accept the connection from the client.
- `init_socket()`: Opens the selector using the openSelector() method of the Selector class and the server socket channel using the open() method of the ServerSocketChannel class. This method then configures the server as non blocking using the configureBlocking() method of the ServerSocketChannel class. The init_socket() method creates the object of the InetSocketAddress class at port 9999 and binds the socket to the channel at that server IP. Finally, this method calls the register() method of the ServerSocketChannel to register the server.
- `accept_connection()`: Calls the select() method of the Selector class when the connection is established. This method then initializes the object of the Iterator class by calling the selectedKeys() method of the Selector class. If the message provider gets a new connection from the end user, the accept_connection() method creates and initializes the object of the ServerSocketChannel class to retrieve the channel using the getChannel() method. Next, the accept_connection() method calls the got_connection() method to get the connection. If the selector provider gets a stream to read the message from the chat user, the accept_connection() method calls the read_Message() method to read the message. If the selector provider gets the stream to write a message to the chat user, the accept_connection() method creates the object of the SocketCallback class and calls the attachment() method of the SelectionKey class. Finally, the accept_connection() method calls the write_Message() to write the message.
- `got_connection()`: Initializes the object of the SocketChannel class to accept the connection and sets the server to non blocking using the configureBlocking() method. The got_connection() method then registers the server socket with the selector, creates the object of the SocketCallback class, and calls the attach() method of the SelectionKey class.
- `read_Message()`: Checks end user validity using the isValidUser() method of the SocketCallback class. If the end user is valid, the read_Message() method clears the buffer and reads the bytes from the channel to the buffer. The read_Message() method then retrieves the data from the buffer and separates the user name and broadcast message from the incoming data. If the data received from the server is an instance of the user name, the read_Message() method initializes the object of the AppendUserList class to add the end user to the user list. If the data received from the server is an instance of a message, the read_Message() method initializes the object of the Msgbroadcast class. If the end user is not valid, the read_Message() method clears the buffer, calls the closeRemoteChannel() method, and initializes the object of the UAinnerChecker class.
- `write_Message()`: Calls the doBroadcast() method of the SocketCallback class. The write_Message() method then creates the object of the ByteBuffer class and wraps the message bytes. Finally, the write_Message() method writes the byte buffer to the server socket.
- closeRemoteChannel(): Removes all the messages stored in the broadcastMessage array list.

- `finalize()`: Calls the close() method to close the server socket channel and selector.

The UAServer_Socket class consists of an inner class, UainnerChecker, which allows you to validate the user name and password. The methods defined in this inner class are:

- `run()`: Calls the Chk_Validity() method. This method is invoked when the constructor of the UAinnerChecker class is executed.
- `Chk_Validity()`: Sets the id_flag and pwd_flag for the user id and password and checks the end user validity using the Chk_Valid_User() method. After validation, the Chk_Validity() method calls the setConid() method of the SocketCallback class.
- `Chk_Valid_User()`: Returns true if the user name and password you specify are valid.

### Creating the PRServer_Socket.java File

The PRServer_Socket.java file creates the socket connection to send and receive private messages. The PRServer_Socket class initializes and registers the socket to the chat server.

Listing 2-3 shows the contents of the PRServer_Socket.java file:

**Listing 2-3: The PRServer_Socket.java File**

```
/*Imports the java.net package class.*/
import java.net.*;
/*Imports the java.net package class.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
/*Imports the java.util package class.*/
import java.util.*;
/*Imports the java.io package class.*/
import java.io.*;
/*
Class PRServer_Socket - Creates the socket, registers and maps the socket to send private _
messages to the specified end user.
      Fields:
      port: Contains the port number.
      srvr_sckt_chnel: Creates the server socket channel.
      sckt_manager: Manages the various sockets.
      selkey: Selects the appropriate socket.
      tdata: Stores the array of the bytes.
      userName: Stores the name of the private message user.
      privateMessage - Stores the messages.
      Methods:
      init_socket(): Initializes the private messaging socket.
      register_server(): Registers the server.
      run(): Runs the started threads.
            accept_connection() - Accepts the connections from the server.
            read_Message() - Reads the message from the server socket.
            closeRemoteChannel() - Closes all the remote channels.
            write_Message() - Writes the messages to the server socket.
            got_connection() - Retrieves the connection.
            finalize() - Finalizes the server connection.
*/
public class PRServer_Socket implements Runnable
{
   /*Declares the port for private messaging.*/
   int port=8888;
   /*Declares the object of the ServerSocketChannel class.*/
   ServerSocketChannel srvr_sckt_chnel=null;
   /*Declares the object of the Selector class.*/
   Selector sckt_manager=null;
   /*Declares the object of the Iterator class.*/
   Iterator searcher=null;
   /*Declares the object of the SelectionKey class.*/
   SelectionKey selkey=null;
   /*Declares the object of the ByteBuffer class and allocate the size to byte buffer.*/
   ByteBuffer buffer=ByteBuffer.allocateDirect(1024);
   /*Declares the objects of the ArrayList class.*/
   ArrayList userName=new ArrayList();
   ArrayList privateMessage = new ArrayList();
   int conId=0;
   int buflen=0;
   byte[] tdata=null;
   int kichek =0;
   /* Defines the constructor that provides the port number as input. */
   public PRServer_Socket(int port)
   {
      this.port=port;
   }
   /*Defines the constructor that provides the port number and user list as input.*/
   public PRServer_Socket(int port, ArrayList conlist)
   {
      this.port=port;
      userName=conlist;
   }
```

```
   /*Defines the constructor that provides the user list as input.*/
   public PRServer_Socket(ArrayList conlist)
   {
      userName=conlist;
   }
   /*
   init_socket() - This method initializes the chat server.
   Parameter: NA
   Return Value: NA
   */
   public void init_socket() throws Exception
   {
      /*Opens the selector.*/
      sckt_manager=SelectorProvider.provider().openSelector();
      /*Opens the server socket channel.*/
      srvr_sckt_chnel=ServerSocketChannel.open();
      /* Configures the server as NON Blocking.*/
      srvr_sckt_chnel.configureBlocking(false);
      /*Creates and initializes the object of the InetSocketAddress class.*/
      InetSocketAddress fulnetaddr=new InetSocketAddress(port);
      /*Binds the server socket.*/
      srvr_sckt_chnel.socket().bind(fulnetaddr);
      /*Calls the register()method.*/
      register_server(srvr_sckt_chnel,SelectionKey.OP_ACCEPT);
   }
   /*
      register_server() - This method register the chat server.
         Parameter:
         ssc: Represents the object of the ServerSocketChannel class.
         selectionkey_ops: Integer type that determines the selection option.
         Return Value: NA
   */
public void register_server(ServerSocketChannel ssc, int selectionkey_ops)throws Exception
   {
      /*Calls the register() method of the ServerSocketChannel class*/
      ssc.register(sckt_manager,selectionkey_ops);
   }
   /*
      run() - This method is called when the thread is started.
         Parameters:   NA
         Return Value: NA
   */
   public void run()
   {
      try
      {
         /*Calls the initialization method.*/
         init_socket();
         /*Calls the method to accept the connection.*/
         accept_connection();
      }
      catch(Exception ej)
      {
         ej.printStackTrace();
      }
   }
   /*
      accept_connection() - This method accepts the connection from the server.
         Parameter: NA
         Return Value: NA
   */
   public void accept_connection()throws Exception
   {
      while (true)
        {
            /*Calls the select() method of the Selector class.*/
         kichek = sckt_manager.select();
         if(kichek >= 0)
         {
            /*Initializes the object of the Iterator class by calling the selectedKeys() method
            of the Selector class.*/
            searcher = sckt_manager.selectedKeys().iterator();
            while(searcher.hasNext())
            {
               /*Gets the next element.*/
               selkey = (SelectionKey)searcher.next();
               /*Removes the item from the searcher.*/
               searcher.remove();
               /*When the message provider gets new connection from the end user, this section
               is executed.*/
               if(selkey.isAcceptable())
               {
                  /*Creates and initializes the object of the ServerSocketChannel class to get _
                  the channel using the channel() method.*/
                  ServerSocketChannel server=(ServerSocketChannel)selkey.channel();
                  /*Calls the got_connection() method.*/
                  got_connection(server);
                  System.out.println("connection establish");
```

```
                }
/*When the selector provider gets the stream for reading by the server from the end user, this _
section is executed.*/
else if(selkey.isReadable())
                {
                   /*Calls the read_Message() method*/
                   read_Message((SocketCallback)selkey.attachment());
                }
/* When selector Provider gets stream for writing to the end user, then this section is executed. */
else if(selkey.isWritable())
                {
                   /* Creates the object of the SocketCallback class and calls the attachment() _
                   method of the SelectionKey class. */
                   SocketCallback  vsc =(SocketCallback)selkey.attachment();
                   String m_essage="";
                   if (vsc.getString().length()>0)
                   m_essage =vsc.getString();
                   if(vsc.get_broad_msg().length()>0)
                   m_essage +=vsc.get_broad_msg();
                   /* Calls the write_Message(). */
                   write_Message(vsc,m_essage);
                }
             }
          }
          else
          break;
       }
    }
    /*
read_Message() - This method reads the messages from the server socket.
       Parameter:
sc: Represents the object of the SocketCallback class
       Return Value:
       String: Returns a string value.
    */
    public String read_Message(SocketCallback sc)
    {
       try
       {
          /*Clears the buffer.*/
          buffer.clear();
/*Reads the bytes from the channel.*/
int nbyte=sc.getChannel().read(buffer);
          if(nbyte==-1)
          {
             /*Closes the channel.*/
             sc.getChannel().close();
             return "";
          }
          if (nbyte>0)
          {
             /* Flips the buffer. */
             buffer.flip();
             /*Initializes the object of the Byte class.*/
             tdata=new byte[nbyte];
             /*Retrieves the data from the buffer. */
             buffer.get(tdata,0,nbyte);
             /*Initializes the object of the Inner_checker class.*/
             new Inner_checker(sc,tdata,tdata.length);
          }
        }
       catch(IOException ex)
       {
          try
          {
             /*Calls the closeRemoteChannel() method.*/
             closeRemoteChannel(sc);
             /*Closes the channel.*/
             sc.getChannel().close();
          }
          catch(IOException ec)
          {
             ec.printStackTrace();
          }
       }
       /*Returns a null string.*/
       return "";
    }
    /*
       closeRemoteChannel() - This method close the remote channel
          Parameter:
       sCb: Represents the object of the SocketCallback class.
          Return Value: NA
    */
       public void closeRemoteChannel(SocketCallback sCb)
          {
             int i = userName.indexOf(sCb);
             if (i >= 0)
```

```
            {
            /*Removes the user name from the list.*/
                     userName.remove(i);
                  }
                  int j = privateMessage.indexOf(sCb);
                  if (j >= 0)
                  {
            /*Removes the private message from the user list.*/
                     privateMessage.remove(j);
      }
   }
   /*
write_Message() - This method write the messages to the server socket
      Parameter:
sc: Represents the object of the SocketCallback class.
      msg: Stores the message.
      Return Value: NA
   */
public void write_Message(SocketCallback sc,String msg)throws Exception
   {
      if (msg.length()>0)
      {
         /*Calls the doBroadcast() method.*/
         sc.doBroadcast();
         /*Creates the object of the ByteBuffer class. Next, wraps the bytes that are retrieved
         from the message. */
         ByteBuffer b=ByteBuffer.wrap(msg.getBytes());
         /*Writes the byte buffer to the server socket.*/
         sc.getChannel().write(b);
      }
   }
   /*
got_connection() - This method gets the connection from the server.
      Parameter:
server: Represents the object of the ServerSocketChannel class.
      Return Value: NA
   */
private void got_connection(ServerSocketChannel servr)throws Exception
   {
      /*Initializes the object of the SocketChannel class to accept the connection.*/
      SocketChannel sc=servr.accept();
      /*Sets the server to Non Blocking server.*/
      sc.configureBlocking(false);
      /Registers the server socket with the selector.*/
      SelectionKey skey=sc.register(sckt_manager,SelectionKey.OP_READ|SelectionKey.OP_WRITE);
      /*Creates the object of the SocketCallback class.*/
      SocketCallback callbk=new SocketCallback(sc);
      privateMessage.add(callbk);
      skey.attach(callbk);
   }
   /*
finalize() - This method is invoked when message is read or sent.
      Parameter: NA
      Return Value: NA
   */
   public void finalize() throws IOException
   {
      /*Closes the server socket channel.*/
      srvr_sckt_chnel.close();
      /*Closes the selector.*/
      sckt_manager.close();
   }
   /*
      Inner Class: Inner_checker - This class checks the end user validity.
            Methods:
            run(): Runs the started threads.
            readName(): Reads the user name.
      storeMessage_on_Database(): Stores the messages.
   */
   class Inner_checker extends Thread
   {
      /*Declares the objects of the SocketCallback class.*/
      SocketCallback to_m=null;
      SocketCallback from=null;
      byte ib[]=null;
      int blen=0;
      /*Defines the constructor that takes the SocketCallback class object and integer type _
      value as parameters.*/
      public Inner_checker(SocketCallback isc,byte[] msgbyte,int mlen)
            {
               from=isc;
               ib=msgbyte;
               blen=mlen;
               /*Calls the start() method.*/
               start();
            }
      /*
         run() - This method is called when the thread is started.
```

```
            Parameters:   NA
            Return Value: NA
      */
      public void run()
      {
         boolean V_U_chk=false;
         /*Initializes the object of the Iterator class that gets the user name.*/
         Iterator chkval=userName.iterator();
         /*Gets the user id.*/
         String fromName = "" + from.getUserId();
         String[] s=null;
         if(fromName.length()<1)
         {
            /*Calls the readName() method.*/
            s=readName(ib,blen,1);
            return;
         }
         else
         {
            /*Calls the readName() method.*/
            s=readName(ib,blen,0);
            /*Gets the user id.*/
            fromName=from.getUserId();
         }
           while(chkval.hasNext())
         {
            to_m=(SocketCallback)chkval.next();
            if(fromName.equals(to_m.getUserId()))
            {
                  V_U_chk=true;
                  break;
            }
         }
            if(!V_U_chk)
         {
            try
            {
            from.getChannel().close();
            int iobj = privateMessage.indexOf(from);
            if (iobj >= 0)
               {
                  /*Removes private message from the privateMessage array list.*/
                  privateMessage.remove(iobj);
               }
            return;
            }
            catch(Exception ec)
            {
               ec.printStackTrace();
            }
         }
/*Initiates the object of the Iterator class to iterate the private message.*/
Iterator n=privateMessage.iterator();
         SocketCallback msgTo=null;
         while(n.hasNext())
         {
            msgTo=(SocketCallback)n.next();
            if(msgTo.getUserId().equals(s[0]))
            {
               /*Calls the append_broad_msg() message of the SocketCallback class.*/
               msgTo.append_broad_msg(from.getUserId()+ ":"+s[1]);
               /*Calls the storeMessage_on_Database() method.*/
               storeMessage_on_Database(s[1],s[0],fromName);
               break;
            }
         }
      }
      /*
         readName() - This method reads the name of the user.
            Parameter:
            b: Represents a byte array.
            len: Contains the length.
            chk: Represents the chk integer.
            Return Value: NA
      */
private String[] readName(byte[] b,int len,int chk)
      {
         String tmp="";
         int sindex=0;
         byte t[]=null;
         String tmpStr[]=new String[2];
         String usrNAME="";
         for(int x=0;x<len;x++)
         {
            /*Checks for read name separator.*/
            if(b[x]==3)
            {
               sindex=x;
```

```
                break;
             }
             else
             tmp+=(char)b[x];
          }
                if(chk==1)
          {
             for(int x=sindex+1;x<len;x++)
             {
                if(b[x]==3)
                {
                   sindex=x;
                   break;
                }
                else
/*Reads from the end user.*/
                usrNAME+=(char)b[x];
             }
             /*Sets user id.*/
             from.setUserId(usrNAME);
          }
             int tmpval=len-sindex;
             t=new byte[tmpval];
          /*Copies the array.*/
          System.arraycopy(b,sindex+1,t,0,tmpval-1);
          tmpStr[0]=tmp;
          /*Creates and initializes the object of the String class.*/
          tmpStr[1]=new String(t);
          /*Returns the user name array string.*/
          return tmpStr;
       };
       /*
          storeMessage_on_Database() - This method stores the messages on database.
             Parameter:
             u_msg: Contains the message.
             to_clientID: Contains the sender user name.
             from_id: Contains the receiver user name.
             Return Value: NA
       */
public void storeMessage_on_Database(String u_msg,String to_clientID,String from_id)
       {
          /*Inserts code here to implement this method.*/
       };

    };
}
```

Download this Listing.

In the above code, the PRServer_Socket class creates a socket on port 8888 to send and receive private messages. The methods defined in this code are:

- `run()`: Calls the init_socket() method to initialize the socket and calls the accept_connection() method to accept the connection.
- `init_socket()`: Opens the selector using the openSelector() method of the Selector class and opens the server socket channel using the open() method of the ServerSocketChannel class. The init_socket() method then configures the server as non blocking using the configureBlocking() method of the ServerSocketChannel class, and creates the object of the InetSocketAddress class at port 9999. Finally, this method binds the socket to the channel at that server IP and calls the register_server() method.
- `register_server()`: Calls the register() method of the ServerSocketChannel to register the server.
- `accept_connection()`: Calls the select() method of the Selector class and checks the value of a variable, kichek. This method then initializes the object of the Iterator class by calling the selectedKeys() method of the Selector class. If the searcher variable has more elements, the accept_connection() method gets the next element and removes the item from the searcher. If the message provider gets a new connection from the end user, this method creates and initializes the object of the ServerSocketChannel class to get the channel using the channel() method. Next, the accept_connection() method calls the got_connection() method to get the connection. If the selector provider gets the stream to read the message from the end user, the accept_connection() method calls the read_Message() method to read the message. If the selector Provider gets the stream to write the message that is to be sent by the chat server to the end user, the accept_connection() method creates the object of the SocketCallback class and calls the attachment() method of the SelectionKey class. This method then calls the write_Message() method to write the message.
- `read_Message()`: Reads the bytes from the channel to the buffer and flips the buffer. This method then retrieves the data from the buffer and initializes the object of the Inner_checker class.
- `closeRemoteChannel()`: Removes all the messages stored in the privateMessage array list and the user names stored in the userName array list.
- `write_Message()`: Calls the doBroadcast() method of the SocketCallback class and creates the object of the ByteBuffer class. The write_Message() method then wraps the message bytes and writes the byte buffer to the server socket.

- `got_connection()`: Initializes the object of the SocketChannel class to accept the connection. This method then sets the server to a non blocking server using the configureBlocking() method and registers the server socket with the selector. The got_connection() method creates the object of the SocketCallback class and calls the attach() method of the SelectionKey class.
- `finalize()`: Calls the close() method to close the server socket channel and selector.

The above code uses an inner class, named Inner_checker, to send private messages. The methods defined in this inner class are:

- `run()`: Sends and stores private messages in the database. This method calls the readName() method to read the name of the sender. If the names of the sender and receiver are identical, the run() method calls the break method. Otherwise, the run() method initializes the object of the Iterator class to iterate the private message and calls the append_broad_msg() message of the SocketCallback class to send the private message. The run() method calls the storeMessage_on_Database() method to store the messages in a database.
- `readName()`: Reads the data string from the server and separates the user name from this string using a separator, 3. This method reads the name and calls the setUserId() method of the SocketCallback class.

**Note** The storeMessage_on_Database() method stores the messages in the database, but this method is not implemented in the above code. You can implement this method for future enhancements.

### Creating the Msgbroadcast.java File

The Msgbroadcast.java file sends the message to all the end users connected to the chat server.

Listing 2-4 shows the contents of the Msgbroadcast.java file:

**Listing 2-4: The Msgbroadcast.java File**

```
/*Imports the java.util package class.*/
import java.util.*;
/*
   Class Msgbroadcast - Broadcasts the messages to all the end users connected in a chat session.
   storer - Stores the user list.
   sc - Stores the object of the SocketCallback class.
   next - Contains the Iterator object.
   msg - Contains the message.
   userId - contains the name of the end user.
*/
public class Msgbroadcast extends Thread
{
   /*Declares the object of the ArrayList class.*/
   ArrayList storer = null;
   /*Declares the object of the SocketCallback class.*/
   SocketCallback sc = null;
   /*Declares the object of the Iterator class.*/
   Iterator next = null;
   /* Declares the object of the String class.*/
   String msg = "";
   String userId = null;
   /*Defines the default constructor of the Msgbroadcast class.*/
   Msgbroadcast(ArrayList store,String msg,String userId)
   {
      storer = store;
      this.msg = msg;
      this.userId = userId;
      /*Calls the start() method of the Thread class. */
      start();
   }
   /*
      run() - This method is called when the thread is started.
         Parameters:   NA
         Return Value: NA
   */
   public void run()
   {
      /* Initializes the object of the Iterator class to get the user id from the user list. */
      next=storer.iterator();
      while(next.hasNext())
      {
         /* Gets the user id from the socket. */
         sc=(SocketCallback)next.next();
         /* Broadcasts the messages to all end users. */
         sc.append_broad_msg(userId + ":" + msg);
      }
   }
}
```

Download this Listing.

In the above code:

- The constructor of the Msgbroadcast class calls the start() method of the Thread class to invoke the run() method.

- The run() method initializes the object of the Iterator class to get the user name from the user list. The run() method also retrieves the user name from the server socket and calls the append_broad_msg() method of the SocketCallback class.

### Creating the AppendUserList.java File

The AppendUserList.java file retrieves the user list from the server, adds a new end user to it, and returns the updated user list to the server.

Listing 2-5 shows the contents of the AppendUserList.java file:

**Listing 2-5: The AppendUserList.java File**

```
/* Imports the java.util package class. */
import java.util.*;
/*
   Class AppendUserList - This class stores the name of all the end users connected in the chat sess
   Field:
   sc: Contains the object of the SocketCallback class.
   uL: Stores the user list in an array list.
   Method:
   run() - This method is called by the start() method of Thread class. This method adds the
   user id.
*/
public class AppendUserList extends Thread
{
   /* Declares the objects of the SocketCallback class. */
   SocketCallback csc=null;
   SocketCallback sc=null;
   /* Declares the object of the ArrayList class. */
   ArrayList uL=null;
   /* Defines default constructor of the AppendUserList class. */
   public AppendUserList(SocketCallback csc,ArrayList UL)
   {
      this.csc=csc;
      this.uL=UL;
      /* Calls the start() method of the Thread class. */
      start();
   }
   /*
run() - This method is called when the thread is started.
      Parameters:   NA
      Return Value: NA
   */
   public void run()
{
   /*Creates and initializes the object of the Iterator class to get the user id from the user list.
   Iterator next=uL.iterator();
   /*Creates a string that contains the user id separator "31".*/
   String s=""+(char)31;
   while(next.hasNext())
   {
      /* Gets the user id from the socket. */
      sc=(SocketCallback)next.next();
      s=s+sc.getUserId()+(char)31;
   }
   /*Adds a user id to the user list.*/
   this.csc.append(s);
   }
}
```

Download this Listing.

In the above code:

- The constructor of the AppendUserList class calls the start() method of the Thread class that invokes the run() method.
- The run() method creates and initializes the object Iterator class to get the user name from the user list.
- The run() method also creates a string that contains the user id with separator, 31.
- The run() method retrieves the user id from the socket and adds the user name to the user list.

### Creating the SocketCallback.java File

The SocketCallback.java file contains the methods to provide connectivity between the client and the server. The SocketCallback class establishes a connection between the chat server and the chat client and gets the file channel to send messages to all the users connected to the chat server.

Listing 2-6 shows the contents of the SocketCallback.java file:

**Listing 2-6: The SocketCallback.java File**

```
/* Imports the java.nio package class. */
import java.nio.*;
import java.nio.channels.*;
/*
Class PRServer_Socket - Uses the sockets to communicate with the chat client.
   Fields:
   soket: Contains the object of the SocketChannel class.
   UserId: Contains the user id.
   Pwd: Contains the password.
   Uinfo: Contains the user information
   broadcast_msg: Contains the broadcasting message.
   Methods:
   isValidUser(): Checks the end user is valid or not.
   ValidUser(): Returns the valid user.
   getChannel(): Returns the socket channel.
   setUserId(): Sets the user id.
   getUserId(): Retrieves the user id.
   setPwd(): Sets the password.
   getPwd(): Retrieves the password.
   setUinfo(): Sets the user information.
   getUinfo(): Retrieves the user information.
   append(): Appends the message.
   append_broad_msg(): Appends the broadcasting messages.
   get_broad_msg(): Retrieves the broadcast messages.
   addString(): Adds message string.
   getString(): Retrieves the messages string.
   getConid(): Returns the connection id.
   doBroadcast(): Broadcasts the message to all user.
   isBroadcast(): Checks message is broadcasted or not.
   setConid(): Sets the connection id.
*/
public class SocketCallback
{
   /*Declares the object of the SocketChannel class.*/
   private SocketChannel soket=null;
   /*Declares the objects of the String class.*/
   private String str="";
   private String UserId="";
   private String Pwd="";
   private String Uinfo="";
   private int cid=0;
   private boolean isvalid_user=false;
   private boolean userMark=false;
   private boolean msgbroadcast=false;
   String broadcast_msg="";
   /*Defines the default constructor.*/
   public SocketCallback(SocketChannel sc)
   {
      soket=sc;
   }
   /* The isValidUser() implementation. */
   public boolean isValidUser()
   {
      return isvalid_user;
   }
   /*The ValidUser() implementation.*/
   public void ValidUser(boolean b)
   {
      isvalid_user=b;
   }
   /*The getChannel() implementation.*/
   public SocketChannel getChannel()
   {
      return soket;
   }
   /*The setUserId() implementation.*/
   public void setUserId(String u)
   {
      UserId=u;
   }
   /*The getUserId() implementation.*/
   public String getUserId()
   {
      return UserId;
   }
   /*The setPwd() implementation.*/
   public void setPwd(String u)
   {
      Pwd=u;
   }
   /* The getPwd() implementation. */
   public String getPwd()
   {
      return Pwd;
   }
   /* The setUinfo() implementation. */
```

```
    public void setUinfo(String u)
    {
       Uinfo=u;
    }
    /* The getUinfo() implementation. */
    public String getUinfo()
    {
       return Uinfo;
    }
    /* The append() implementation. */
    public void append(String s)
    {
       broadcast_msg=s+broadcast_msg;
    }
    /*The append_broad_msg() implementation.*/
    public void append_broad_msg(String msg)
    {
       if (msg.length()>0)
    {
       broadcast_msg+=msg;
    }
    }
    /*The get_broad_msg() implementation.*/
    public String get_broad_msg()
    {
       return broadcast_msg;
    }
    /*The addString() implementation.*/
    public void addString(String s)
    {
       str=s;
    }
    /*The getString() implementation.*/
    public String getString()
    {
       String str1=new String(str);
       str="";
       return str1;
    }
    /*The getConid() implementation.*/
    public int getConid()
    {
       return cid;
    }
    /*The doBroadcast() implementation.*/
    public void doBroadcast(boolean chk)
    {
       msgbroadcast=chk;
    }
    /*The doBroadcast() implementation.*/
    public void doBroadcast()
    {
       broadcast_msg="";
    }
    /* The isBroadcast() implementation. */
    public boolean isBroadcast()
    {
       return msgbroadcast;
    }
    /* The setConid() implementation. */
    public void setConid(int id)
    {
       cid=id;
    }
}
```

Download this Listing.

In the above code, the SocketCallback class implements all the methods that are invoked in the server classes. The methods defined in this code are:

- `isValidUser()`: Checks whether or not the end user is valid. This method returns true if the end user is valid.
- `getChannel()`: Returns the socket channel.
- `setUserId()`: Sets the user name.
- `getUserId()`: Retrieves the user name.
- `setPwd()`: Sets the password.
- `getPwd()`: Retrieves the password.
- `setUinfo()`: Sets the user information.
- `getUinfo()`: Retrieves the user information.

- `append()`: Appends the message.
- `append_broad_msg()`: Appends the broadcasting messages.
- `get_broad_msg()`: Retrieves the broadcast messages.
- `addString()`: Adds message string.
- `getString()`: Retrieves the message string.
- `getConid()`: Returns the connection id.
- `doBroadcast()`: Broadcasts the message to all the end users.
- `isBroadcast()`: Checks whether or not the message is broadcasted.
- `setConid()`: Sets the connection id.



◄ PREVIOUS NEXT ►

# Creating the Chat Client

To create the chat client, you need to create the following files:

- ChatLogin.java file
- ChatClient.java file
- CClient.java file
- Messager.java file

### Creating ChatLogin.java File

The ChatLogin.java file helps create a login window. In the login window, an end user enters the server IP address, user name, and password to connect to the chat server.

Listing 2-7 shows the contents of the ChatLogin.java file:

**Listing 2-7: The ChatLogin.java File**

```
/* Imports the javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
import javax.swing.JPasswordField;
import javax.swing.BorderFactory;
/*Imports the java.awt package classes.*/
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
import java.awt.Font;
/*Imports the javax.swing.event package classes.*/
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JOptionPane;
import javax.swing.UIManager;
import javax.swing.JSeparator;
/*
class ChatLogin - Creates the Chat Login window for the chat application.
      Method:
      - actionPerformed()
      - main()
*/
public class ChatLogin extends JDialog implements ActionListener
{
   JPanel panel;
   JPanel pane;
   /* Declares the objects of the JLabel class. */
   JLabel titleLabel;
   JLabel nameLabel;
   JLabel pwdLabel;
   JLabel ipLabel;
   /* Declares the objects of JTextField class */
   JTextField nameText;
   JPasswordField pwdText;
   JTextField ipText;
   /*Declares the objects of the JButton class.*/
   JButton connect;
   JButton cancel;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   int value;
   /* Defines the default constructor.*/
   public ChatLogin()
   {
      /*Sets the title of the Font dialog box.*/
      setTitle("Chat Login Window");
      /*Sets the size of Font dialog box.*/
      setSize(280, 170);
      /*Sets resizable button to FALSE.*/
      setResizable(false);
      /*Initializes the object of the GridBagLayout class*/
      gbl = new GridBagLayout();
      /* Sets the Layout. */
      getContentPane().setLayout(gbl);
      /*Creates an object of the GridBagConstraints class.*/
```

```
        gbc = new GridBagConstraints();
        /*
           Initializes the title label object and adds it to the 1, 1, 2, 1 position with WEST
           alignment.
        */
           gbc.gridx = 1;
           gbc.gridy = 1;
           gbc.gridwidth = 2;
           gbc.gridheight = 1;
           gbc.anchor = GridBagConstraints.CENTER;
           pane = new JPanel();
        pane.setLayout(new FlowLayout());
        titleLabel = new JLabel("Chat Login Window");
        titleLabel.setFont(new Font("Verdana",Font.BOLD,20));
        pane.add(titleLabel);
        getContentPane().add(pane, gbc);
        gbc.gridx = 1;
        gbc.gridy = 2;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        getContentPane().add(new JSeparator(), gbc);
        /*
           Initializes the IP address label object and adds it to the 1, 3, 1, 1 position with
           EAST alignment
        */
        gbc.gridx = 1;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.EAST;
        ipLabel = new JLabel("Server IP: ");
        getContentPane().add(ipLabel, gbc);
        /*
Initializes the IP address text field object and adds it to the 2, 3, 1, 1 position with WEST alignm
        */
        gbc.gridx = 2;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        ipText = new JTextField("192.168.0.36", 15);
        ipText.setFont(new Font("Verdana",Font.PLAIN,12));
        getContentPane().add(ipText, gbc);
        gbc.gridx = 1;
        gbc.gridy = 4;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        getContentPane().add(new JSeparator(), gbc);
        /*
Initializes the user name label object and adds it to the 1, 5, 1, 1 position with EAST alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 5;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.EAST;
        nameLabel = new JLabel("User Name: ");
        getContentPane().add(nameLabel, gbc);
        /*
           Initializes the user name text field object and adds it to the 2, 5, 1, 1 position with
           WEST alignment.
        */
        gbc.gridx = 2;
        gbc.gridy = 5;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        nameText = new JTextField(15);
        nameText.setFont(new Font("Verdana",Font.PLAIN,12));
        getContentPane().add(nameText, gbc);
        gbc.gridx = 1;
        gbc.gridy = 6;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        getContentPane().add(new JSeparator(), gbc);
        /*
           Initializes the password label object and adds it to the 1, 7, 1, 1 position with EAST
           alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 7;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.EAST;
        pwdLabel = new JLabel("Password: ");
```

```
      getContentPane().add(pwdLabel, gbc);
      /*
         Initializes the password text field object and adds it to the 2, 7, 1, 1 position with
         WEST alignment.
      */
      gbc.gridx = 2;
      gbc.gridy = 7;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      pwdText = new JPasswordField(15);
      pwdText.setFont(new Font("Verdana",Font.PLAIN,12));
      pwdText.addActionListener(this);
      getContentPane().add(pwdText, gbc);
      gbc.gridx = 1;
      gbc.gridy = 8;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      getContentPane().add(new JSeparator(), gbc);
      /*
         Initializes the OK and Cancel button. Adds the button to the 2, 9, 1, 1 position with
         WEST alignment.
      */
      gbc.gridx = 2;
      gbc.gridy = 9;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      panel = new JPanel();
      panel.setLayout(new FlowLayout(FlowLayout.LEFT));
      connect = new JButton("Connect");
      connect.addActionListener(this);
      cancel = new JButton("Cancel");
      cancel.addActionListener(this);
      panel.add(connect);
      panel.add(cancel);
      getContentPane().add(panel, gbc);

      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent we)
         {
            System.exit(0);
         }
      });
   }
   /*
actionPerformed() - This method is called when the user clicks the any button.
Parameters: ae: Represents an object of the ActionEvent class that contains the details of the event
   Return Value: NA
   */
   public void actionPerformed(ActionEvent ae)
   {
      if(ae.getSource() == connect)
      {
/*Creates an object of the ChatClient class.*/
         ChatClient tcc = new ChatClient();
/*Calls the con() method to connect to the server.*/
tcc.con(nameText.getText(), pwdText.getText(), ipText.getText());
         this.setVisible(false);
      }
      else if(ae.getSource() == cancel)
      {
         System.exit(0);
      }
      else if(ae.getSource() == pwdText)
      {
         /*Creates an object of the ChatClient class.*/
         ChatClient tcc = new ChatClient();
         /*Calls the con() method to connect to the server.*/
         tcc.con(nameText.getText(), pwdText.getText(), ipText.getText());
         this.setVisible(false);
         }
         }
         /*Main method.*/
         public static void main(String args[])
         {
            try
            {
               /*Sets the window look and feel to the application.*/
               UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
            }
            catch(Exception e){}
            ChatLogin cl = new ChatLogin();
            cl.show();
      }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the ChatLogin class that allows you to open the Chat Login window, as shown in Figure 2-2:


**Figure 2-2:** The Chat Login Window

After an end user specifies the server IP address, user name, and password and clicks any button on the Chat Login window, the chat application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate class and method, based on the button the end user clicks. If the end user clicks the Connect button or presses the Enter key, the actionPerformed() method creates an object of the ChatClient class and calls the con() method of the ChatClient class.

### Creating ChatClient.java File

The ChatClient.java file helps create a user interface for the chat client. End users can use this interface to display the user list of the connected users. The chat application user interface contains a text pane that displays the messages sent by other chat users.

Listing 2-8 shows the contents of the ChatClient.java file:

**Listing 2-8: The ChatClient.java File**

```
/*Imports the java.awt package classes.*/
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.List;
import java.awt.Point;
/*Imports the java.awt.event package classes.*/
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
/*Imports java.io package classes.*/
import java.io.PrintStream;
/*Imports java.util package classes.*/
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util.Vector;
/*Imports the javax.swing package classes.*/
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.JOptionPane;
import javax.swing.UIManager;
/*Imports the java.swing.text package classes.*/
import javax.swing.text.AttributeSet;
import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;
/*
Class ChatClient - Creates the main window of the chat application. This window contains a text
pane to display the messages and list box to display the user list.
   Methods:
      - prepareAllStyles()
      - run()
      - itemStateChanged()
```

```
      - actionPerformed()
      - con()
      - AddUser()
   Class
      - ListofUser
      - PrivateChat
      - BroadcastChat
*/
public class ChatClient extends JFrame implements ActionListener, Runnable, ItemListener
{
   /*Declares the AWT and Swing components.*/
   JPanel paneMain;
   JPanel paneLeft;
   JPanel paneRight;
   JPanel paneBottom;
   JLabel welcomeLabel;
   JLabel listLabel;
   JLabel pMsgLabel;
   JLabel bMsgLabel;
   JTextField msgText;
   JTextArea pArea;
   JTextField empty;
   List clientList;
   JScrollPane pAreaScroll;
   JScrollPane listScroll;
   JButton sendButton;
   JButton privateButton;
   JButton logoutButton;
   JTextField listText;
   SimpleAttributeSet aset;
   AttributeSet current;
   Hashtable ht = null;
   DefaultStyledDocument doc = null;
   JTextPane tp = null;
   /*Declares the object of the Thread class.*/
   Thread thread;
   Thread t;
   Thread th;
   /*Declares the static string to set the text pane attributes.*/
   final String NORMALBLUE = "NormalBlue";
   final String BOLDBLUE = "BoldBlue";
   final String NORMALBLACK = "NormalBlack";
   final String BOLDGREEN = "BoldGreen";
   final String NORMALRED = "NormalRed";
   final String ITALICRED = "ItalicRed";
   /*Declares the objects of the String class.*/
   String nameStr1 = null;
   String msgStr1 = null;
   String nameStr2 = null;
   String msgStr2 = null;
   String str3 = null;
   String str4 = null;
   String str5 = null;
   String str6 = null;
   String msg;
   String user;
   int value;
   int count=0;
   int i;
   /*Creates the object of the PrivateChat class.*/
   PrivateChat Pchat = new PrivateChat();
   /*Creates the object of the BroadcastChat class.*/
   BroadcastChat Achat = new BroadcastChat();
   /*Creates the object of the ListofUser class.*/
   ListofUser UL=new ListofUser();
   /*Creates the object of the CClient class.*/
   CClient CC = null;
   /*Defines the default constructor.*/
   public ChatClient()
   {
      setSize(700, 400);
      setTitle("Chat Client");
      setResizable(false);
      paneMain = new JPanel();
      paneMain.setLayout(new BorderLayout());
      getContentPane().add(paneMain);
      paneLeft = new JPanel();
      paneLeft.setLayout(new BorderLayout());
      doc = new DefaultStyledDocument();
      ht = new Hashtable();
      tp = new JTextPane(doc)
      {
         public void paintComponent(Graphics g)
         {
         super.paintComponent(g);
         }
         public void paint(Graphics g)
         {
```

```
                super.paint(g);
            }
        };
        tp.setFont(new Font("Verdana", Font.PLAIN, 12));
        tp.setEditable(false);
        pAreaScroll = new JScrollPane(tp);
        paneLeft.add(pAreaScroll, BorderLayout.CENTER);
        paneMain.add(paneLeft, BorderLayout.CENTER);
        paneRight = new JPanel();
        paneRight.setLayout(new BorderLayout());
        listLabel = new JLabel("User List");
        listText = new JTextField(10);
        listText.setVisible(false);
        clientList = new List(3);
        clientList.addActionListener(this);
        clientList.addItemListener(this);
        clientList.select(0);
        listScroll = new JScrollPane(clientList);
        paneRight.add(listText, BorderLayout.SOUTH);
        paneRight.add(listLabel, BorderLayout.NORTH);
        paneRight.add(listScroll, BorderLayout.CENTER);
        paneMain.add(paneRight, BorderLayout.EAST);
        paneBottom = new JPanel();
        paneBottom.setLayout(new FlowLayout());
        msgText = new JTextField(33);
        msgText.requestFocus();
        msgText.addActionListener(this);
        msgText.setFont(new Font("Verdana",Font.PLAIN,14));
        sendButton = new JButton("Send");
        privateButton = new JButton("Private Message");
        logoutButton = new JButton("Logout");
        empty = new JTextField(10);
        paneBottom.add(msgText);
        paneBottom.add(sendButton);
        paneBottom.add(privateButton);
        paneBottom.add(logoutButton);
        paneBottom.add(empty);
        paneMain.add(paneBottom, BorderLayout.SOUTH);
        privateButton.addActionListener(this);
        sendButton.addActionListener(this);
        logoutButton.addActionListener(this);
        prepareAllStyles();
        t = new Thread(this);
        /*
            addWindowListener - It contains a windowClosing() method.
            windowClosing: It is called when the user clicks the cancel button of the Window. It _
            closes the main window.
                Parameter: we- Object of WindowEvent class.
                Return Value: NA
        */
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                value = JOptionPane.showConfirmDialog(null, "Are you sure you want to close the chat
                application?", "Close", JOptionPane.YES_NO_OPTION);
                if(value == 0)
                {
                    try
                    {
                        /*Calls the sendBroadMsg() method to inform all the users.*/
                        CC.sendBroadMsg(user + "* has left the chat session." + "+REMOVE");
                        System.exit(0);
                    }
                    catch(Exception e){}
                }
                else if(value == 1)
                {
                }
            }
        });
    }
/*Defines the prepareAllStyles() method that set the text attribute values.*/
public void prepareAllStyles()
    {
        aset = new SimpleAttributeSet();
        StyleConstants.setForeground(aset,Color.blue);
        StyleConstants.setFontSize(aset,12);
        StyleConstants.setFontFamily(aset,"Verdana");
        ht.put(NORMALBLUE,aset);
        aset = new SimpleAttributeSet();
        StyleConstants.setForeground(aset,Color.blue);
        StyleConstants.setFontSize(aset,12);
        StyleConstants.setFontFamily(aset,"Verdana");
        StyleConstants.setBold(aset, true);
        ht.put(BOLDBLUE,aset);
        aset = new SimpleAttributeSet();
        StyleConstants.setForeground(aset,Color.black);
```

```
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      ht.put(NORMALBLACK,aset);
      aset = new SimpleAttributeSet();
      StyleConstants.setForeground(aset, new Color(0, 128, 0));
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      StyleConstants.setBold(aset, true);
      ht.put(BOLDGREEN,aset);
      aset = new SimpleAttributeSet();
      StyleConstants.setForeground(aset,Color.red);
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      ht.put(NORMALRED,aset);
      aset = new SimpleAttributeSet();
      StyleConstants.setForeground(aset,Color.red);
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      StyleConstants.setItalic(aset, true);
      ht.put(ITALICRED,aset);
   }
   /*
itemStateChanged() - This method is called when the end user selects the user name from the user lis
   Parameters: ae: Represent an object of the ActionEvent class that contains the details of the eve
      Return Value: NA
   */
   public void itemStateChanged(ItemEvent ie)
   {
      if(ie.getSource() == clientList)
      {
listText.setText(clientList.getSelectedItem());
         i = clientList.getSelectedIndex();
      }
   }
   /*
      actionPerformed() - This method is called when the end user clicks any button.
         Parameters: ae: Represent an object of the ActionEvent class that contains the details
      of the event.
      Return Value: NA
   */
   public void actionPerformed(ActionEvent ae)
   {
      /*
         This section is executed when the end user clicks the Private Message button of the applica
      */
      if(ae.getSource() == privateButton)
      {
         try
         {
if(listText.getText().equals("*" + user))
            {
               JOptionPane.showMessageDialog(null, "You cannot send private message to _
               yourself!", "Error",
               JOptionPane.ERROR_MESSAGE);
            }
            else
            {
               current = (AttributeSet)ht.get(BOLDBLUE);
               doc.insertString(doc.getLength(), "To " + listText.getText() + ": " ,current);
               current = (AttributeSet)ht.get(NORMALBLACK);
               doc.insertString(doc.getLength(), msgText.getText() +  "\n" ,current);
               Point pt1=tp.getLocation();
               Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
               pAreaScroll.getViewport().setViewPosition(pt2);
            }
         }
catch(Exception e) {}
if(msgText.getText().trim().equals(""))
         {
         }
         else
         {
            /*Calls the sendPrivateMessage() method to send the private message.*/
            CC.sendPrivateMessage(msgText.getText().trim(),listText.getText());
            msgText.setText("");
         }
      }
      /*
This section is executed, when the end user clicks the Send button of the application.
      */
      else if(ae.getSource() == sendButton)
      {
         if(msgText.getText().trim().equals(""))
         {
         }
         else
         {
            /*Calls the sendBroadMsg() method to send the broadcast message.*/
```

```
            CC.sendBroadMsg(msgText.getText().trim() + "+MSG");
            msgText.setText("");
         }
      }
      /*
This section is executed, when the end user clicks the Logout button of the application.
      */
      else if(ae.getSource() == logoutButton)
      {
         value = JOptionPane.showConfirmDialog(null, "Are you sure you want to logout?", "Logout",
         JOptionPane.YES_NO_OPTION);
         if(value == 0)
         {
         try
         {
         /*Calls the sendBroadMsg() method to inform the end user has left the chat session.*/
         CC.sendBroadMsg(user + "* has left the chat session." + "+REMOVE");
         System.exit(0);
            }
            catch(Exception e){}
         }
         else if(value == 1)
         {
         }
      }
      /*
This section is executed, when the end user enter the sending text and press the enter button
from the keyboard.
      */
      else if(ae.getSource() == msgText)
      {
/*Calls the sendBroadMsg() method to send the broadcast message.*/
CC.sendBroadMsg(msgText.getText().trim() + "+MSG");
         msgText.setText("");
      }
   }
/*Implementation of con() method. This method establishes the connection between client and server.*
public void con(String userName, String pwd, String ipAdd)
   {
      user = userName;
      CC = new CClient();
      /*Calls the addMessager() method of the CClient class.*/
      CC.addMessager(UL , Achat, Pchat);
      try
      {
         /*Calls the connect() method of the CClient class.*/
         CC.connect(userName, pwd, ipAdd);
      }
      catch(Exception e)
      {
         JOptionPane.showMessageDialog(null, "Unable to connect the server!", "Error",
         JOptionPane.ERROR_MESSAGE);
         ChatLogin cl = new ChatLogin();
         cl.show();
         this.setVisible(false);
         return;
      }
setTitle("Chat Application - " + userName);
/*Starts the thread to send to inform all the users.*/
      try
      {
         count = 1;
         t = new Thread(this);
         t.setPriority(Thread.MAX_PRIORITY);
         t.sleep(100);
         t.start();
      }
      catch(Exception e){};
/*Starts the thread to display the user list.*/
      try
      {
         count = 2;
         thread = new Thread(this);
         thread.sleep(100);
         thread.start();
      }
      catch(Exception e){}
      this.show();
      tp.repaint();
   }
/*Creates the class to display the user list.*/
   class ListofUser implements Messager
   {
      public void message(String m)
      {
         clientList.clear();
         AddUser(m);
         clientList.select(i);
```

```
      }
   };
/*Creates the class to send the private message.*/
class PrivateChat implements Messager
   {
      public void message(String m)
      {
         try
         {
            /*Inserts text in the text pane with the specified attributes.*/
            m=m.substring(0,m.length()-1);
            StringTokenizer st1 = new StringTokenizer(m, ":");
            nameStr1 = st1.nextToken();
            msgStr1 = st1.nextToken();
            current = (AttributeSet)ht.get(BOLDBLUE);
            doc.insertString(doc.getLength(), "From " + nameStr1 + ": ", current);
            current = (AttributeSet)ht.get(NORMALBLACK);
            doc.insertString(doc.getLength(), msgStr1 + "\n" , current);
            Point pt1=tp.getLocation();
            Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
            pAreaScroll.getViewport().setViewPosition(pt2);
         }
         catch(Exception e){}
         tp.repaint();
      }
   };
/*Creates the class to send the broadcast message.*/
   class BroadcastChat implements Messager
   {
      public void message(String m)
      {
         try
         {
            /*Inserts text in the text pane with the specified attributes.*/
            StringTokenizer st2 = new StringTokenizer(m, ":");
            nameStr2 = st2.nextToken();
            msgStr2 = st2.nextToken();
            StringTokenizer st3 = new StringTokenizer(msgStr2, "+");
            str3 = st3.nextToken();
            str4 = st3.nextToken();
            if(str4.equals("ADD"))
            {
               StringTokenizer st4 = new StringTokenizer(str3, "*");
               str5 = st4.nextToken();
               str6 = st4.nextToken();
               if(str5.equals(user))
               {
                  try
               {
                  msg = "Welcome to chat session!";
                  current = (AttributeSet)ht.get(BOLDBLUE);
                  tp.setCharacterAttributes(current,true);
                  doc.insertString(doc.getLength(),  msg + "\n",current);
               }
               catch(Exception e){}
               }
               else
               {
                  try
                  {
                     current = (AttributeSet)ht.get(ITALICRED);
                     doc.insertString(doc.getLength(), str5 + str6 + "\n",current);
                     Point pt1=tp.getLocation();
                     Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                     pAreaScroll.getViewport().setViewPosition(pt2);
                  }
               catch(Exception e){}
               }
            }
else if(str4.equals("REMOVE"))
            {
               StringTokenizer st4 = new StringTokenizer(str3, "*");
               str5 = st4.nextToken();
               str6 = st4.nextToken();
               if(str5.equals(user))
               {
                  try
                  {
                     msg = "You have successfully logged out from this chat session.";
                     current = (AttributeSet)ht.get(ITALICRED);
                     doc.insertString(doc.getLength(), msg + "\n",current);
                     Point pt1=tp.getLocation();
                     Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                     pAreaScroll.getViewport().setViewPosition(pt2);
                  }
               catch(Exception e){}
               }
```

```
                else
                {
                   try
                   {
                      current = (AttributeSet)ht.get(ITALICRED);
                      doc.insertString(doc.getLength(), str5 + str6 + "\n",current);
                      Point pt1=tp.getLocation();
                      Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                      pAreaScroll.getViewport().setViewPosition(pt2);
                   }
                catch(Exception e){}
                }
             }
             else if (str4.equals("MSG"))
             {
                try
                {
                   current = (AttributeSet)ht.get(BOLDGREEN);
                   doc.insertString(doc.getLength(), nameStr2 + ": " , current);
                   current = (AttributeSet)ht.get(NORMALBLUE);
                   doc.insertString(doc.getLength(), str3 + "\n",current);
                   Point pt1=tp.getLocation();
                   Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                   pAreaScroll.getViewport().setViewPosition(pt2);
                }
                catch(Exception e){}
             }
          }
          catch (Exception ex) {}
          tp.repaint();
       }
    };
/*Creates the class to add a new end user into the user list.*/
private void AddUser(String s)
    {
       String tmp="";
       byte b[]=s.getBytes();
       if(s.length()>1)
        {
          for(int x=1;x<b.length;x++)
          {
             if(b[x]==31)
             {
                if(tmp.equals(user))
                {
                   clientList.add("*"+tmp);
                }
                else
                {
                   clientList.add(tmp);
                }
                tmp="";
             }
             else
             tmp+=(char)b[x];
          }
       }
    };
/*This method is invoked when thread is started.*/
public void run()
    {
       if(count == 1)
       {
          /*Calls the sendBroadMsg() to inform all the end users.*/
          CC.sendBroadMsg(user + "* has joined the chat session." + "+ADD");
       }
       else if(count == 2)
       {
          while(true)
          {
             try
             {
                /*Calls the sendListMsg() method to display the list of users.*/
                CC.sendListMsg("");
                Thread.sleep(5000);
             }
catch(Exception ee){ee.printStackTrace();}
          }
       }
    }
}
```

Download this Listing.

In the above code, the constructor of the ChatClient class creates the user interface for the chat client window, as shown in Figure 2-3:

**Figure 2-3:** The Chat Client Window

The text pane displays the messages the end users send or receive. The user interface allows an end user to send private and broadcast messages to other end users connected with the chat server. To send a public or broadcast message to all the users, the end user needs to type the message in the text box in the bottom pane and click the Send button. To send a private message to a particular chat client, the end user needs to select the client's user name from the user list, type the message in the text box, and click the Private Message button.

The inner classes defined in the above code are:

- `ListofUser`: Implements the Messager interface and defines the abstract method, message(). The message() method calls the clear() method to clear the user list and the AddUser() method to add the end user.
- `PrivateChat`: Implements the Messager interface and defines the abstract method, message(). The message() method tokenizes the message to retrieve the user name and the message. The message() method then sets the attributes of the document and calls the insertString() method to insert the message in the text pane.
- `BroadcastChat`: Implements the Messager interface and defines the abstract method, message(). The message() method tokenizes the message to retrieve the user name and the message. Next, the message() method again tokenizes the message to check the message type, such as new end user connected message, user left message, or simple broadcast message. The message() method then sets the attributes of the document and calls the insertString() method to insert the message in the text pane.

The methods defined in the above code are:

- `con()`: Accepts the user name, password, and IP address and calls the addMessager() method of the CClient class. The con() method also calls the connect() method of the CClient class to connect the end user to the chat server.
- `run()`: Checks the value of the variable count. If the count value is 1, this method calls the sendBroadMsg() method of the CClient class to send a message to all the users that a new end user has joined the chat session. If the count value is 2, the run() method calls the sendListMsg() method of the CClient class to display the user list.
- `prepareAllStyles()`: Sets the text attribute values. This method initializes the object of the SimpleAttributeSet class and calls the setForeground(), setFontSize(), setFontFamily(), setBold(), and setItalic() methods of the StyleConstants class to set the document styles.
- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method, based on the button the end user clicks on the Chat Client window. If the end user clicks the Send button of the Chat application, the actionPerformed() method calls the sendBroadMsg() method to send the message to all the users connected to the chat server. If the end user clicks the Private Message button of the Chat application, the actionPerformed() method calls the sendPrivateMessage() method to send the message to a specified chat user. If an end user clicks the Logout button of the Chat application, the actionPerformed() method calls the sendBroadMsg() method to send a message to all the users that this end user has left the chat session. The actionPerformed() method then calls the System.exit(0) method to close the Chat application window.
- `itemStateChanged()`: Acts as an event listener and activates an appropriate method, based on the item the end user selects from the user list box.
- `AddUser()`: Adds the user to the user list. This method separates the user name from the message sent by the chat server to chat client.

### Creating CClient.java File

The CClient.java file implements the core functionality of the chat client module. This file reads the server IP address, user name, and password from the Chat Login window and connects the end user to the chat server. An end user can use this file to send and receive messages from the chat server.

Listing 2-9 shows the contents of the CClient.java file:

**Listing 2-9: The CClient.java File**

```
/* Imports the java packages. */
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
import java.util.*;
import java.io.*;
import java.net.*;
```

```
/*
Class CClient - Implements the methods declared or called in the chat application client. This _
file connects the chat client to chat server.
   Method:
      - addMessager()
      - getBroadCastSender()
      - getPrivateSender()
      - connect()
      - runClient()
      - run()
      - sendBroadMsg()
      - sendListMsg()
      - sendPrivateMessage()

*/
public class CClient extends Thread
{
   /* Declares the objects and initializes the variables. */
   private String userId="";
   private String pwd="";
   ByteBuffer buffer=ByteBuffer.allocateDirect(1024);
   SocketChannel Psc=null;
   SocketChannel Asc=null;
   private InetSocketAddress AAddr=null;
   private InetSocketAddress PAddr=null;
   private int APort=9999;
   private int PPort=8888;
   boolean connectStatus=false;
   private ArrayList Lmsg=null;
   private boolean Validity_Checker=false;
   public Messager usrlist=null;
   public Messager bcmsg=null;
   public Messager prmsg=null;
   boolean chkfirst=true;
   /* Defines the default constructor. */
   public CClient()
   {
   }
   /* Implements the addMessager() method. */
   public void addMessager(Messager usrlist, Messager bcmsg, Messager prmsg)
   {
      this.usrlist=usrlist;
      this.bcmsg=bcmsg;
      this.prmsg=prmsg;
   }
   /*Implements the connect() method. This method connects the specified user to the server.*/
   public void connect(String UserId,String Password,String HostAddress) throws
   NotYetConnectedException
   {
      this.userId=UserId;
      this.pwd=Password;
      try
      {
      Asc=SocketChannel.open(new InetSocketAddress(HostAddress,APort));
      Psc=SocketChannel.open(new InetSocketAddress(HostAddress,PPort));
      while(!Asc.finishConnect())
      {
      }
      while(!Psc.finishConnect())
      {
      }
      runClient();
      sendBroadMsg("");
      sendPrivateMessage(" "," ");
      }
      catch(Exception e1)
      {
         throw new NotYetConnectedException();
      }
   }
/* Implements the runClient() method. This method runs the client. */
   public void runClient() throws Exception
   {
      this.start();
      new PrivateReceiver(Psc);
   }
/*Implements the run() method. This method is invoked when thread is started.*/
public void run()
   {
      byte[] Tdata=null;
      while(true)
      {
         try
         {
            buffer.clear();
            int nbyte=Asc.read(buffer);
            if(nbyte==-1)
            {
```

```
                bcmsg.message("Connection Closed");
                Asc.close();
                return;
             }
             if (nbyte>0)
             {
/*Reads the messages from the chat server.*/
buffer.flip();
Tdata=new byte[nbyte];
buffer.get(Tdata,0,nbyte);
if(Tdata[0]!=31)
             {
                String s=new String(Tdata);
                bcmsg.message(s);
             }
             else
             {
                String s=new String(Tdata);
                usrlist.message(s);
             }
          }
          catch(IOException ex)
          {
             try
             {
                Asc.close();
             }
             catch(IOException ec)
             {
                ec.printStackTrace();
             }
          }
       }
    }
/*Implements the sendBroadMsg() method. This method broadcast the message to all the end users.*/
    public void sendBroadMsg(String msg)
    {
       try
       {
          if(!connectStatus)
          {
             String tmps=userId+(char)28+pwd+(char)29+"";
             ByteBuffer b=ByteBuffer.wrap(tmps.getBytes());
             Asc.write(b);
             connectStatus=true;
          }
          else
          {
             ByteBuffer b=ByteBuffer.wrap(msg.getBytes());
             Asc.write(b);
          }
       }
       catch(Exception sb)
       {
          sb.printStackTrace();
       }
    }
/*Implements the sendListMsg() method. This method sends the user list to the end user.*/
public void sendListMsg(String msg)
    {
       try
       {
          String tmps=""+(char)31+"";
          ByteBuffer b=ByteBuffer.wrap(tmps.getBytes());
          Asc.write(b);
       }
       catch(Exception sl)
       {
          sl.printStackTrace();
       }
    }
/*Implements the sendPrivateMessage() method.*/
public void sendPrivateMessage(String Msg,String to)
    {
       try
       {
          String Ts=null;
          if(chkfirst)
          {
             Ts=to+(char)3+userId+(char)3+Msg;
             chkfirst=false;
          }
          else
             Ts=to+(char)3+Msg;
             ByteBuffer b=ByteBuffer.wrap(Ts.getBytes());
             Psc.write(b);
```

```
        }
        catch(Exception se)
        {
           se.printStackTrace();
        }
     }
     /*Creates the PrivateReceiver class.*/
     class PrivateReceiver extends Thread
     {
        SocketChannel P_channel=null;
        ByteBuffer buff=ByteBuffer.allocateDirect(1024);
        byte[] Tdata=null;
        /*Defines the default constructor.*/
        PrivateReceiver(SocketChannel chan )
        {
            this.P_channel=chan;
           start();
        }
        /*Implements the run() method.*/
        public void run()
        {
           while(true)
           {
              try
              {
                 buff.clear();
                 int nbyte=P_channel.read(buff);
                 if(nbyte==-1)
                 {
                    prmsg.message("Connection Closed");
                    P_channel.close();
                    return;
                 }
                 if (nbyte>0)
                 {
                    /* Reads the data. */
                    buff.flip();
                    Tdata=new byte[nbyte];
                    buff.get(Tdata,0,nbyte);
                    String s=new String(Tdata);
                    prmsg.message(s);
                 }
              }
              catch(IOException ex)
              {
                 try
                 {
                 P_channel.close();
                 }
                 catch(IOException ec)
                 {
                    ec.printStackTrace();
                 }
              }
           }
         }
     };
}
```

Download this Listing.

In the above code, the CClient class defines the default contractor. The methods defined in this code are:

- `addMessager()`: Retrieves the user list, broadcast message, and private message.
- `connect()`: Opens two sockets to send private and broadcast messages. This method then calls the runClient(), sendBroadMsg(), and sendPrivateMsg() methods.
- `runClient()`: Starts a thread, initializes the object of the PrivateReceiver class, and invokes the run() method.
- `run()`: Reads the bytes from the socket to the buffer and separates the user name and broadcast message from the received message. The run() method then calls the message() method to send the broadcast message and the user list.
- `sendBroadMsg()`: Checks the connection status, wraps the message to the byte buffer, and writes the buffer to the server socket.
- `sendListMsg()`: Wraps the message to the byte buffer and writes the buffer to the server socket.
- `sendPrivateMessage()`: Checks the destination user id, wraps the message to the byte buffer, and writes the buffer to the server socket.

The CClient class contains a sub class, PrivateReceiver, which receives the message from the server. This class defines the default constructor that calls the start() method of the Thread class to invoke the run() method. The run() method clears the buffer and reads the bytes from the channel to the buffer. This method then flips the buffer and calls the message() method to send a private message.

### Creating Messager.java File

The Messager.java file creates an interface that declares the message() method. This method helps display the message in the text pane of the chat client window.

Listing 2-10 shows the contents of the Messager.java file:

**Listing 2-10: The Messager.java File**

```
/*Defines an abstract method to read and write the message to the text pane of the chat _
application client.*/
public interface Messager
{
   /*Declares the message() method.*/
   public void message(String msg);
}
```

Download this Listing.

In the above code, the Messager interface declares an abstract message() method. This method is defined later, where this interface is implemented to send or receive message from the chat server..

## Unit Testing

To test the Chat application:

1. 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;
   ```
2. 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;
   ```
3. Copy the ChatServer.java, UAServer_Socket.java, PRServer_Socket.java, AppendUserList.java, Msgbroadcast.java, and SocketCallback.java files to a folder on the server computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. Copy the ChatLogin.java, ChatClient.java, CClient.java, and Messager.java files to a folder on the client computer. Compile the files using the following javac command:
   ```
   javac *.java
   ```
5. Run the Chat application server using the following command at the command prompt:
   ```
   java ChatServer
   ```
6. Run the Chat application login window using the following command at the command prompt:
   ```
   java ChatLogin
   ```
   The Chat Login window appears, as shown in Figure 2-2.
7. Specify the server IP address, user name, and password. Click the Connect button. The Chat Application - user1 window appears, as shown in Figure 2-4:

**Figure 2-4:** Welcome Message in the Chat Application - user1 Window

An Error dialog box appears, if the process to establish a connection between the client and the server fails, as shown in Figure 2-5:

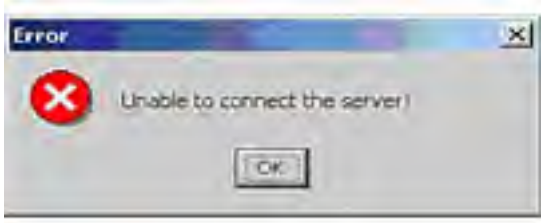


**Figure 2-5:** The Error Dialog Box

8. Run the Chat application login window using the following command at the command prompt from another client computer:
   ```
   java ChatLogin
   ```
   The Chat Login window appears, as shown in Figure 2-2.
9. Specify the server IP address, user name, and password. Click the Connect button. The Chat Application - user2 window appears, as shown in Figure 2-6:

**Figure 2-6:** The Chat Application - user2 Window with List of Users

When user2 joins the chat session, a message, user2 has joined the chat session appears in the text pane of the user1 chat window, as shown in Figure 2-7:

**Figure 2-7:** The Chat Application - user1 Window with user2 in Chat Session

10. Type Hi all in the text box of the Chat application window of user1 and click the Send button. The message is broadcasted to all the users, as shown in Figure 2-8:

**Figure 2-8:** Receiving Message in the Chat Application - user2 Window

Figure 2-9 shows the broadcast message that appears in the user2 client window:

**Figure 2-9:** Receiving Message in the Chat Application - user2 Window

11. Type Hi user2! in the text box of the user1 Chat application window, select user2 from the User List, and click the Private Message button. The message is sent to the user2 only, as shown in Figure 2-10:

**Figure 2-10:** Receiving Message in the Chat Application - user1 Window

Figure 2-11 shows the private message that appears in the user2 client window:

**Figure 2-11:** Private Message in the Chat Application - user2 Window

12. Click the Logout button of the Chat Application - user2 window. A Confirm dialog box appears, as shown in Figure 2-12:

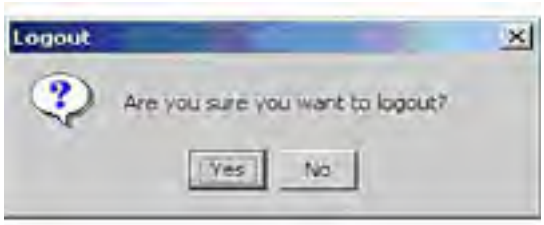


**Figure 2-12:** The Confirm Dialog Box

13. Click Yes on the Confirm dialog box. A message, user2 has left the chat session appears in the text pane of all the chat users, as shown in Figure 2-13:

**Figure 2-13:** The Chat Application - user1 Window

# Chapter 3: Creating a File Download Application

The New Input/Output (NIO) API supports the java.nio and java.nio.channels packages for buffer management and advance I/O file system. The NIO API allows you to use the java.rmi package for file transfer across the Java Virtual Machine (JVM). The java.nio package contains the ByteBuffer classes, which you can use to store the content of a file. The java.nio.channels package provides the FileChannel class, which you can use to read and write the file. The java.rmi package provides remote interface to call the methods from the remote client.

This chapter explains how to develop a File Download application. The application uses the above-mentioned NIO and Remote Method Invocation (RMI) packages to download files from the remote machines.

## Architecture of the File Download Application

The File Download application allows an end user to view a list of files stored at a specific location on the file server. The end user can select a file from the list and download it at the specified location.

The File Download application uses the following files:

- `FileRemote.java`: Creates a remote interface that declares the remote method for the application.
- `FileRemoteImpl.java`: Creates an implementation file that defines the remote methods declared in the remote interface.
- `FileInfo.java`: Creates a class that contains details, such as name, path, and size, of a particular file.
- `FileServer.java`: Creates a file server that binds the remote objects to the RMI registry. As a result, a client can invoke the object from the remote location.
- `FileClient.java`: Creates a user interface for the File Download application. The user interface helps an end user display a list of files stored in the file server in a tabular form. This interface also allows the end user to select and download a particular file.
- `ProgressTest.java`: Creates a Download Status dialog box that contains a progress bar. The progress bar indicates the percentage of file that has been downloaded.

Figure 3-1 shows the architecture of the File Download application:

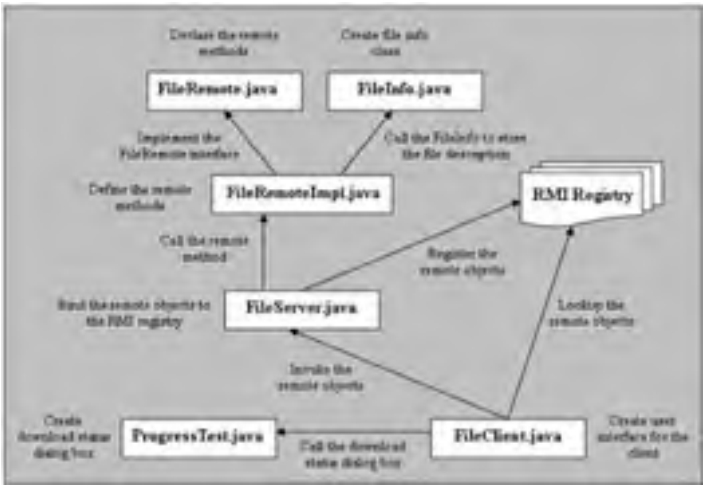


**Figure 3-1:** Architecture of the File Download Application

In the File Download application, the FileClient.java file calls the FileServer.java file to invoke the remote methods to display a file list and download a specified file. The FileServer.java file calls the FileRemoteImpl.java file, which finds and returns a list of files available at the specified location to the FileClient.java file. To download a particular file, the FileRemoteImpl.java file reads and transfers that file to the client computer. The FileRemoteImpl.java file implements the FileRemote.java file, which creates a remote interface to declare the remote method. The FileRemoteImpl.java file calls the FileInfo.java file to store the file description. The FileClient.java file calls the ProgressTest.java file to open the Download Status dialog box.

# Creating the Remote Interface

To create the client-server application using RMI, you need to create a remote interface that declares the business logic of the application. The FileRemote.java file is a remote interface for the File Download application that defines the prototypes of the methods implemented by the FileRemoteImpl.java file.

Listing 3-1 shows the contents of the FileRemote.java file:

**Listing 3-1: The FileRemote.java File**

```
/* Imports java.rmi package classes. */
import java.rmi.Remote;
import java.rmi.RemoteException;
/* Imports java.util package classes. */
import java.util.Vector;
/*
Interface FileRemote - This interface declares the remote methods.
Methods:
downloadFile() - Downloads the file from a remote location.
displayList() - Displays the list of files stored in the server to the client.
*/
public interface FileRemote extends Remote
{
   /* Declares downloadFile() method. */
   public byte[] downloadFile(String filename) throws RemoteException;
   /* Declares displayList() method. */
   public Vector displayList() throws RemoteException;
}
```

Download this Listing.

In the above code:

- The FileRemote interface defines the methods to perform various operations of the File Download application. These methods include displayList() and downloadFile().
- The displayList() method displays the file list to the client machine.
- The downloadFile() method downloads the selected file from the remote machine.

# Creating File Descriptor

The FileInfo.java file creates a file descriptor class that stores the description of a particular file.

Listing 3-2 shows the content of the FileInfo.java file:

**Listing 3-2: The FileInfo.java File**

```
/* Imports the java.io package class. */
import java.io.Serializable;
/*
class FileInfo - This class stores the file description, such as file index, file name, and file siz
Fields:
fileIndex - Contains the file name.
fileName - Contains the file path.
fileSize - Contains the file size.
*/
public class FileInfo implements Serializable
{
   /* Declares the objects of String class. */
   String fileIndex;
   String fileName;
   String fileSize;
   /* Defines default constructors. */
   public FileInfo(String fileIndex, String fileName, String fileSize)
   {
      this.fileIndex = fileIndex;
      this.fileName = fileName;
      this.fileSize = fileSize;
   }
}
```

Download this Listing.

In the above code, the FileInfo() constructor creates the file descriptor class. The fileName, filePath, and fileSize strings contain the file name, file location, and file size, respectively.

# Creating the Implementation File

The implementation file implements all the business methods that are declared in the remote interface. FileRemoteImpl.java is an implementation file that defines all the methods that are declared in the remote interface.

Listing 3-3 shows the content of the FileRemoteImpl.java file:

**Listing 3-3: The FileRemoteImpl.java**

```
/* Imports java.io package classes.*/
import java.io.*;
import java.io.File;
/* Imports java.rmi package classes. */
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
/* Imports java.nio package classes. */
import java.nio.*;
import java.nio.charset.*;
import java.nio.channels.*;
/* Imports java.util package classe. */
import java.util.Vector;
/*
Class FileRemoteImpl - This is the implementation class that implements all
the methods declared in the remote interface.
Fields:
name - Stores the name.
fi - Represents the object of FileInfo class.
Methods:
downloadFile() - This method is called when the end user clicks the window close button.
displayList() - This method is invoked when an end user selects any command from the menu bar.
*/
public class FileRemoteImpl extends UnicastRemoteObject implements FileRemote
{
   private String name;
   public FileInfo fi;
   /* Defines default constructor. */
   public FileRemoteImpl(String str) throws RemoteException
   {
      super();
      name = str;
      }
      /*
      downloadFile() - Defines the downloadFile() method.
      Parameter: filename - Represents the name of the file.
      Return Value: bufferFile[] - Represents the byte array that contains the file data.
      */
         public byte[] downloadFile(String filename)
      {
         /* Declares object of FileInputStream class. */
         FileInputStream fin;
         /* Declares object of FileChannel class. */
         FileChannel fchan;
         /* Declares object of ByteBuffer class. */
         ByteBuffer buff;
         long fsize;
         String str;
         byte bufferFile[] = null;
         try
         {
            /* Creates instance of FileInputStream class. */
            fin = new FileInputStream(filename);
            /* Gets the channel from file input stream. */
            fchan = fin.getChannel();
            /* Retrieves the size of the file channel. */
            fsize = fchan.size();
            /* Allocates the size of byte buffer. */
            buff = ByteBuffer.allocate((int)fsize);
            /* Reads the file from the channel to buffer. */
            fchan.read(buff);
            /* Rewinds the buffer. */
            buff.rewind();
            /* Creates a byte buffer of size equals to the file size. */
            bufferFile = new byte[(int)fsize];
            for(int i=0; i<(int)fsize; i++)
            {
               /* Stores the data into a byte array. */
               bufferFile[i] = buff.get();
            }
         }
         catch(Exception e)
         {
            System.out.println("Error: " + e);
          }
         /* Returns the byte array. */
```

```
            return(bufferFile);
         }
      /*
      displayList() - Defines the displayList() method.
      Parameter: NA
      Return Value: v - Represents a vector that contains the list of files and their details.
      */
      public Vector displayList() throws RemoteException
      {
         /*
         Creates and initializes the object of the Vector class.
         */
         Vector v = new Vector();
         /* Creates objects of the String class. */
         String SNo = "";
         String fileName = "";
         String size = "";
         /* Creates an object of the String array. */
         String files[] = null;
         int count =1;
         try
         {
            /* Creates and initialize the object of the File class. */
            File folder = new File("C:/CodeBook/NIO/FileTransfer/Server");
            /* Stores the list of files in the string array. */
            files = folder.list();
            for (int i=0; i<files.length; i++)
            {
               /* Creates and initialize the object of the File class. */
               File file = new File(folder.getAbsolutePath() + File.separator + files[i]);
               /*
               Checks whether the file object is a File type or a Directory type.
               */
               if (file.isFile())
               {
                  /* Stores the serial number. */
                  SNo = String.valueOf((count));
                  /* Stores the file name. */
                  fileName = file.getName();
                  /* Stores the file size. */
                  size = String.valueOf(file.length());
                  /* Initializes the object of the FileInfo class. */
                  fi = new FileInfo(SNo, fileName, size);
                  /*
                  Adds the object of the FileInfo class at the end of the vector.
                  */
                  v.addElement(fi);
                  /* Increments the counter by 1. */
                  count++;
               }
            }
         }
         catch(Exception e)
         {
            e.printStackTrace();
            System.out.println("Error! " + e);
         }
/* Returns the vector. */
         return v;
      }
}
```

Download this Listing.

In the above code, the FileRemoteImpl class creates an instance of the FileInfo class to store the file description. The FileRemoteImpl class defines the following remote methods:

- `downloadFile()`: Uses the filename as a parameter and returns the byte array of that file. This method then creates the instance of the FileInputStream, FileChannel, and ByteBuffer classes. Next, the downloadFile() method initializes the instance of the FileInputStream class and calls the getChannel() method of the FileChannel class. The downloadFile() method then initializes the object of the ByteBuffer class and allocates the size of the byte buffer. Next, the downloadFile() method reads the bytes from the file channel to the byte buffer and rewinds the bytes in the byte buffer. Finally, the downloadFile() method initializes a byte array called bufferFile, puts the values from the byte buffer into bufferFile, and returns the bufferFile byte array.
- `displayList()`: Returns the list of files in the form of a vector. This method creates and initializes an object of the Vector class. The displayList() method then initializes the SNo, fileName, and size string variables and creates a file object using the specified directory location. Next, this method calls the list() method of the File class to retrieve a list of files and subdirectories available within the specified directory location. The displayList() method then stores this list in the files[] String array. The displayList() method checks whether the file object is of File type or Directory type. If the file object is of File type, the displayList() method retrieves the name, path, and size from that file. The displayList() method then initializes an object of the FileInfo class using the SNo, fileName, and size variables. Next, this method adds the file descriptor object to a vector and increments the counter by one. Finally, the displayList() method returns the file list vector to the client machine.

# Creating the File Server

The FileServer.java file contains the main function of the file server. The file server binds the remote object to the RMI registry. So, any authorized client can access the remote objects.

Listing 3-4 shows the content of the FileServer.java file:

**Listing 3-4: The FileServer.java**

```
/* Imports java.io package classes. */
import java.io.*;
/* Imports java.rmi package classes. */
import java.rmi.*;
/*
class FileServer - Creates the RMI server that registers all the remote objects.
Method:
main() - Starts the RMI server and registers the remote objects. */
public class FileServer
{
   public static void main(String args[])
   {
      /* Sets the RMI security manager. */
      System.setSecurityManager(new RMISecurityManager());
      try
      {
         /* Creates the remote object. */
         FileRemote f = new FileRemoteImpl("FServer");
         /* Binds the remote object to the RMI registry. */
         Naming.rebind("FServer", f);
         System.out.println("Object is registered.");
         System.out.println("Now server is waiting for client request");
        }
      catch(Exception e)
      {
        e.printStackTrace();
        System.out.println("FileServer: " + e);
      }
   }
}
```

Download this Listing.

In the above code:

- The main() method creates an instance of the RMISecurityManager class. This method sets the security manager to the File Download application using the setSecurityManager() method.
- The main() method also creates an object of the FileRemoteImpl class and binds the remote object to the RMI registry using the rebind() method.

# Creating the User Interface for the File Download Application

The FileClient.java file helps create a user interface that contains a set of labels and buttons. This interface also has an empty tabular space where the file lists are displayed. End users can select a file from the list and download it.

Listing 3-5 shows the contents of the FileClient.java file:

**Listing 3-5: The FileClient.java File**

```
/* Imports java.io package class. */
import java.io.*;
import java.io.File;
/* Imports java.rmi package class. */
import java.rmi.*;
/* Imports java.nio package class. */
import java.nio.*;
import java.nio.charset.*;
import java.nio.channels.*;
/* Imports java.util package class. */
import java.util.Vector;
import java.util.StringTokenizer;
/* Imports java.awt package classes. */
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.Color;
import java.awt.*;
/* Imports java.awt.event package classes. */
import java.awt.event.*;
/* Imports javax.swing package classes. */
import javax.swing.*;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.LookAndFeel;
import javax.swing.UIManager;
/* Imports javax.swing.table package classes. */
import javax.swing.table.*;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.DefaultTableColumnModel;
/*
Class FileClient - Initializes the interface and loads all components like table,
label and button on the main window. This class is the main class of the application.
   Fields:
title - Contains the content of the title label.
panel - Contains the swing components.
scrollpane - Contains the table component.
table - Contains the list of files.
model - Represents the default table model.
listButton - Creates the Get File List button.
downloadButton - Creates the Download File button.
fileName - Stores the file name.
filePath - Stores the file path.
thread - Creates a progress bar thread.
th - Creates a downloading thread.
fName - Stores the file name.
fExt - Stores the file extension.
fout - Contains the file output stream.
fchan - Contains the file channel.
Methods:
addWindowListener() - This method is called when an end user clicks the window close button to close
actionPerformed() - This method is invoked when an end user selects any command from the menu bar.
display() - This method is called when an end user clicks the Get File List button to display the fi
download() - This method is called when an end user clicks the Download File button to download the
main() - This method creates the main window of the application and displays it.
*/
public class FileClient extends JFrame implements ActionListener , Runnable
{
/* Declares the object of the JLabel class. */
   JLabel title;
   /* Declares the object of the JPanel class. */
   JPanel panel;
   JPanel pan1;
   JPanel pan2;
```

```
JPanel pan3;
/* Declares the object of the JScrollPane class. */
JScrollPane scrollpane;
/* Declares the object of the JTable class. */
JTable table;
/* Create and initialize the object of the Vector class. */
Vector vRow = new Vector();
Vector vCol = new Vector();
/* Declares the object of the DefaultTableModel class. */
DefaultTableModel model;
/* Declare the objects of the JButton class. */
JButton listButton;
JButton downloadButton;
/* Declares the object of the GridBagLayout class. */
GridBagLayout gbl;
/* Declares the object of the GridBagConstraints class. */
 GridBagConstraints gbc;
static String str1, str2;
/* Declares the objects of the String class. */
String fileName;
String filePath;
Thread thread;
Thread th;
String fName;
String fExt;
String fname = null;
String fpath = null;
String path = null;
static String ipAdd;
/* Declares the object of the ProgressTest class. */
public ProgressTest pt;
/* Declares the object of the FileOutputStream class. */
FileOutputStream fout;
/* Declares the object of the FileChannel class. */
FileChannel fchan;
int count = 0;
int filesize, fs;
int value;
/* Declares the default constructor of the FileClient class. */
public FileClient()
{
   try
   {
      /* Sets windows look and feel to the application. */
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
   }
   catch(Exception e){}
   /* Sets the size of the window. */
   setSize(470, 515);
   /* Sets the reliability of the progress dialog box to false. */
   setResizable(false);
   /* Sets the title of the progress dialog box. */
   setTitle("File Download Application");
   /* Initializes the object of the GridBagLayout class. */
   gbl = new GridBagLayout();
   /* Sets the Layout */
   getContentPane().setLayout(gbl);
   /* Creates an object of the GridBagConstraints class. */
   gbc = new GridBagConstraints();
   /*
   Initializes the title label object and adds it to the 1,1,1,1 position with CENTER alignment.
   */
   gbc.gridx = 1;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   pan1 = new JPanel();
   pan1.setLayout(new BorderLayout());
   title = new JLabel(" File Download Application ");
   title.setFont(new Font("Verdana",Font.BOLD,20));
   pan1.add(title, BorderLayout.CENTER);
   getContentPane().add(pan1, gbc);
   /*
   Initializes listButton and downloadButton, adds these buttons to the panel. Next,
   adds it to the 1,2,1,1 position with CENTER alignment.
   */
   gbc.gridx = 1;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   pan2 = new JPanel();
   pan2.setLayout(new FlowLayout());
   listButton = new JButton("Get File List");
   listButton.addActionListener(this);
   pan2.add(listButton);
   downloadButton = new JButton("Download the File");
```

```
    downloadButton.setEnabled(false);
    downloadButton.addActionListener(this);
    pan2.add(downloadButton);
    getContentPane().add(pan2, gbc);
    /*
    Initializes the table label object and adds it to the 1,3,1,1 position with CENTER alignment.
    */
    gbc.gridx = 1;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan3 = new JPanel();
    pan3.setLayout(new BorderLayout());
    /*
    Adds the elements at the end of the column vector.
    */
    vCol.addElement("S. No.");
    vCol.addElement("File Name");
    vCol.addElement("Size (Bytes)");
    /* Initializes the object of the DefaultTableModel class to create a table model. */
    model = new DefaultTableModel(vRow, vCol);
    /* Initializes the object of JTable */
    table = new JTable(model);
    /* Sets single selection on the table */
    table.setSelectionMode(0);
    /*
    Creates instance of TableColumn class and set the width of the 0th column.
    */
    int vColIndex = 0;
    TableColumn col = table.getColumnModel().getColumn(vColIndex);
    int width = 50;
    col.setPreferredWidth(width);
    /*
    Creates instance of TableColumn class and set the width of the 1st column.
    */
    vColIndex = 1;
    col = table.getColumnModel().getColumn(vColIndex);
    width = 300;
    col.setPreferredWidth(width);
    /*
    Creates instance of TableColumn class and sets the width of the 2nd column.
    */
    vColIndex = 2;
    col = table.getColumnModel().getColumn(vColIndex);
    width = 100;
    col.setPreferredWidth(width);
    scrollpane = new JScrollPane(table);
    pan3.add(scrollpane, BorderLayout.CENTER);
    getContentPane().add(pan3, gbc);
    /*
    addWindowListener - Contains a windowClosing() method.
    windowClosing: Closes the main window It is called when the end user
    clicks the cancel button of the Window.
    Parameter: we- Object of the WindowEvent class.
    Return Value: NA
    */
    addWindowListener(new WindowAdapter()
    {
    public void windowClosing(WindowEvent we)
       {
          System.exit(0);
       }
    });
}
/*
actionPerformed() - This method is called when the end user selects any menu item from the menu b
Parameters:   ae - An ActionEvent object containing the details of the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
    /*
    This is executed when end user clicks the Get File List button.
    */
    if(ae.getSource() == listButton)
       {
          /* Calls the display method. */
          display();
          /* Sets the caption of the listButton to "Refresh". */
          listButton.setText("Refresh");
          /* Enables the download button. */
          downloadButton.setEnabled(true);
    }
    /*
    This is executed when end user clicks the Download File button.
    */
    else if(ae.getSource() == downloadButton)
```

```
        {
           int n = table.getSelectedRow();
           if ( n != -1)
           {
              /* Retrieves the value of specified column. */
              Object obj1 = table.getValueAt(n, 1);
              Object obj2 = table.getValueAt(n, 2);
              fname = (obj1).toString();
              fs = Integer.parseInt(obj2.toString());
              filesize = fs;
              /* Opens a Confirm dialog box. */
              value = JOptionPane.showConfirmDialog(null, "Are you sure you want to download the file?
              "Confirm", JOptionPane.YES_NO_OPTION);
              if(value == 0)
              {
                 count = 1;
                 /* Initializes a new thread. */
                 th = new Thread(this);
                 /* Starts the thread. */
                 th.start();
              }
              else if(value == 1)
              {
              }
           }
           else
           {
              /* Opens the error messages. */
              JOptionPane.showMessageDialog(null, "You must select a file before downloading!",
              "Message", JOptionPane.ERROR_MESSAGE);
           }
        }
     }
     /*
     display() - This method is called when the end user clicks the List the
     File button of the File Download Application window.
     Parameters:   NA
     Return Value: NA
     */
     public void display()
      {
        /* Creates and initialize the object of the vector class. */
        Vector vec = new Vector();
        /* Sets the number of rows in a table model to ZERO. */
        model.setRowCount(0);
        /* Creates the object of the FileInfo class. */
        FileInfo obj;
        try
        {
           String str = "rmi://"+ipAdd+"/FServer";
           /*
           Creates an instance of the FileRemote interface that looks up the remote
           object from the specified location.
           */
           FileRemote f = (FileRemote)Naming.lookup(str);
           /*
           Calls the displayList() method that returns the list of files stored in the File server.
           */
           vec = f.displayList();
           if((vec!=null) && (vec.size()>0))
           {
           for(int i=0; i<vec.size(); i++)
              {
                 /* Initializes the vRow object of the Vector class. */
                 vRow = new Vector();
                 /* Retrieves the value from the ith location. */
                 obj = (FileInfo)vec.elementAt(i);
                 /* Adds the fileIndex at the end of the vector. */
                 vRow.addElement(obj.fileIndex);
                 /* Adds the fileName at the end of the vector. */
                 vRow.addElement(obj.fileName);
                 /* Adds the fileSize at the end of the vector. */
                 vRow.addElement(obj.fileSize);
                 /* Adds the row in the table model. */
                 model.addRow(vRow);
              }
           }
        }
        catch(Exception e)
        {
           System.out.println("Vector Error:" + e);
        }
     }
  /*
  download() - This method is called when the end user clicks the Download
  File button of the File Download Application window.
  Parameters:
  fileName - Contains the name of the file.
```

```
filePath - Contains the path of the file.
Return Value: NA
*/
public void download(String fileName, String filePath)
    {
      /* Declares the object of the ByteBuffer class. */
      ByteBuffer buff;
      long fsize;
      try
      {
         String str = "rmi://"+ipAdd+"/FServer";
         /*
         Creates an instance of the FileRemote interface that looks up the remote
         object from the specified location.
         */
         FileRemote f = (FileRemote)Naming.lookup(str);
         /*
         Creates and initialize the object of the StringTokenizer class.
         */
         StringTokenizer st = new StringTokenizer(fileName, ".");
         /*
         Retrieves the value of the string before the delimiter "."
         */
         fName = st.nextToken();
         /*
         Retrieves the value of the string after the delimiter ".".
         */
         fExt = st.nextToken();
         path = filePath + "." + fExt;
         count = 2;
         /* Initializes the object of the Thread class. */
         thread = new Thread(this);
         /* Starts the thread. */
         thread.start();
         /*
         Calls the downloadFile() method that returns the content of the file.
         */
         byte[] data = f.downloadFile(fileName);
         try
         {
            /* Initializes the object of the FileOutputStream class. */
            fout = new FileOutputStream(path);
            /* Gets the channel from the output stream. */
            fchan = fout.getChannel();
            /* Allocates the size of the buffer. */
            buff = ByteBuffer.allocateDirect((int)(data.length));
            for(int i=0; i<data.length; i++)
            /* Reads the data from the byte array to the buffer. */
            buff.put(data[i]);
            /* Rewinds the buffer. */
            buff.rewind();
            /* Writes the buffer to the channel. */
            fchan.write(buff);
            /* Closes the channel. */
            fchan.close();
            /* Closes the output stream. */
            fout.close();
            }
         catch(Exception e)
         {
         }
      }
      catch(Exception e)
      {
         e.printStackTrace();
System.out.println("Error: " + e);
      }
   }
/*
run() - This method is called when the thread is started.
Parameters:   NA
Return Value: NA
*/
   public void run()
   {
      if(count == 1)
      {
         try
         {
            /* Creates and initialize the object of the JFileChooser class. */
            JFileChooser jfc = new JFileChooser();
            /* Sets the selection mode of the file chooser. */
            jfc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
            /* Displays the Save dialog box. */
            value = jfc.showSaveDialog(this);
            if(value == JFileChooser.APPROVE_OPTION)
            {
               try
```

```
                {
                   /* Selects the file. */
                   File file = jfc.getSelectedFile();
                   /* Retrieves the file path. */
                   fpath = file.getAbsolutePath();
                   /* Calls the downlaod() method. */
                   download(fname, fpath);
                }
                catch(Exception e)
                {
                   e.printStackTrace();
                   System.out.println("Error" + e);
                }
             }
          }
          catch(Exception e)
          {
             e.printStackTrace();
             System.out.println("Error" + e);
          }
       }
       else if(count == 2)
       {
          /* Initializes the object of ProgressTest class. */
          pt = new ProgressTest(filesize, path, th);
          /* Sets the title. */
          pt.setTitle("Download Complete.");
          /* Sets the label text. */
          pt.label.setText("Download Complete.");
          /* Sets the button caption. */
          pt.ok.setText(" OK ");
          /* Hides the progress bar. */
          pt.bar.setVisible(false);
       }
    }
    /*
    main() - This method creates the main window of the user interface and displays it.
    Parameters:
    args[] - Contains any command line arguments passed.
    Return Value: NA
    */
    public static void main(String args[])
    {
       ipAdd = args[0];
       /*
       Creates and initializes the object of the FileClient class.
       */
       FileClient fc = new FileClient();
       fc.show();
       }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the FileClient class. This class generates the main window of the File Download application, as shown in Figure 3-2:

**Figure 3-2:** The File Download Application User Interface

The application interface contains a tabular space where the list of files is displayed. The various methods defined in the above listing are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the button the end user clicks. If the end user clicks the Get File List button, the actionPerformed() method calls a display() method to display the list of files stored in the server. The actionPerformed() method also changes the button caption to Get File List to Refresh and enables the Download the File button. If an end user clicks the Download the File button, the actionPerformed() method retrieves the file name and size from the selected row of the table and opens a Confirm dialog box. After the end user clicks OK in the Confirm dialog box, the actionPerformed() method sets the value of the count variable to 1 and starts the newly initialized thread.
- `display()`: Creates and initializes the object of the vector class. This method then sets the row count of the table model to 0 and creates an instance of the FileInfo object to retrieve the description of a file. The display() method looks up the remote object from the specified location and calls the displayList() method that returns the list of files stored in the file server. Next, the display() method retrieves the value from the file list vector and adds the element to the table row, vRow vector. Finally, the display() method adds the row vector to the table model.
- `download()`: Takes the file name and file path as input parameters to download the specified file and declares the object of the ByteBuffer class. Next, the download() method looks up the remote object from the specified location and creates and initializes the object of the StringTokenizer class to get the file name extension. This method then sets the counter value to 2 and starts another new thread. The download() method also calls the downloadFile() method that returns the file in the form of a byte array. The download() method then initializes the object of the FileOutputStream class to write a file at a specified location and calls the getChannel() method of the FileChannel class. Next, this method initializes the object of the ByteBuffer class, allocates the size of the byte buffer, reads the bytes from the byte array to the buffer, and rewinds the byte buffer. Finally, the download() method writes the byte buffer to the file channel and close the file channel and output stream.
- `run()`: Runs the threads that are started in the application and checks the count value. If the value is 1, the run() method creates and initializes the object of the JFileChooser class to open the Save dialog box. This method then initializes the File object and gets the file using the getSelectedFile() method and the path using the getAbsolutePath() method of the File class. Next, the run() method calls the download() method. If the count value is 2, the run() method initializes the object of the ProgressTest class and opens the Download Status dialog box. After downloading, the run() method changes the title, label caption, and button caption to indicate that the file has been downloaded.



◄ PREVIOUS NEXT ►

# Creating a Download Status Dialog Box

The ProgressTest.java file helps create a Download Status dialog box for the File Dialog application. This dialog box displays a progress bar and a label to show the download status.

Listing 3-6 shows the contents of the ProgressTest.java file:

**Listing 3-6: The ProgressTest.java File**

```
/* Imports java.util package class. */
import java.util.*;
/* Imports java.awt package classes. */
import java.awt.*;
/* Imports java.awt.event package class. */
import java.awt.event.*;
/* Imports javax.swing package classes. */
import javax.swing.*;
/* Imports javax.swing.event package class. */
import javax.swing.event.*;
/* Imports java.io package class. */
import java.io.File;
/*
class ProgressTest - Creates a Download Status dialog box that indicates how much data is downloaded
Fields:
label - Contains the content of title label.
ok - Creates an OK button.
size - Contains the size of the file.
path - Contains the path where the end user wants to save the file.
th - Contains the instance of the downloading thread.
Method:
performTask() - This method is called when the ProgressBar is painted.
actionPerformed() - This method is invoked when the end user clicks the OK or Cancel button.
*/
public class ProgressTest extends JDialog implements ActionListener
{
   /* Declares the objects of the JLabel class. */
   JLabel label;
   JLabel label_1;
   JLabel label_2;
   /* Declares the object of the JButton class. */
   JButton ok;
   /* Declares the object of the JProgressbar class. */
   JProgressBar bar;
   /* Declares the object of the GridBagLayout class. */
   GridBagLayout gbl;
   /* Declare the object of the GridBagConstraints class. */
   GridBagConstraints gbc;
   /* Declares the objects of the string class. */
   String str;
   String path;
   /* Declares and initialize the size as integer. */
   int size = 100;
   /* Declares the object of the Thread class. */
   Thread th;
   int n = 0;
   /*
   ProgressTest() - This is the default constructor of the ProgressTest class.
   Parameter:
   size - Represents the file size.
   path - Represents the file path where the end user saves the file.
   th - Represents the instance of the Thread class.
   */
   public ProgressTest(int size, String path, Thread th)
   {
      this.size = size;
      this.path = path;
      this.th = th;
      /* Sets the size of the Download Status dialog box. */
      setSize(270, 150);
      /* Sets the visibility of the Download Status dialog box. */
      setVisible(true);
      /*
      Sets the reliability of the Download Status dialog box to false.
      */
      setResizable(false);
      /* Sets the title of the Download Status dialog box. */
      setTitle("Download Status");
      /* Initializes the object of the G   ridBagLayout class. */
      gbl = new GridBagLayout();
      /* Sets the Layout. */
      getContentPane().setLayout(gbl);
      /* Creates an object of the GridBagConstraints class. */
      gbc = new GridBagConstraints();
      /*
```

```
Initializes the label object and add it to the 1,1,1,1 position with WEST alignment.
*/
gbc.gridx = 1;
gbc.gridy = 1;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
label = new JLabel(" File is downloading");
getContentPane().add(label, gbc);
/*
Initializes a blank label object and add it to the 1,2,1,1 position with WEST alignment.
*/
gbc.gridx = 1;
gbc.gridy = 2;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
label_1 = new JLabel(" ");
getContentPane().add(label_1, gbc);
/*
Initializes an object of the JprogressBar class and adds it to the 1,3,1,1 position with CENTE
*/
gbc.gridx = 1;
gbc.gridy = 3;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
bar = new JProgressBar();
/* Sets the size of the progress bar. */
bar.setPreferredSize(new Dimension(225, 20));
bar.setMinimum( 0 );
bar.setMaximum( size );
bar.setValue( 0 );
bar.setBorderPainted(true);
bar.setStringPainted(true);
getContentPane().add(bar, gbc);
/*
Initializes a blank label object and adds it to the 1,4,1,1 position with WEST alignment.
*/
gbc.gridx = 1;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
label_2 = new JLabel(" ");
getContentPane().add(label_2, gbc);
/*
Initializes an object of the Button class and adds it to the 1,5,1,1 position with CENTER alig
*/
gbc.gridx = 1;
gbc.gridy = 5;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
ok = new JButton(" Cancel ");
ok.addActionListener(this);
getContentPane().add(ok, gbc);
n = 100;
/*
Starts the progress bar until the file is downloaded.
*/
for( int iCtr = 1; iCtr < size+1; iCtr +=n )
{
   /*
   Calls the performTask() method to insert a time delay.
   */
   performTask( iCtr );
   /* Updates the progress indicator and label. */
   label.setText( "Downloading "+ iCtr + " bytes of " + size + " bytes.");
   /* Creates an object of the Rectangle class that gets the label bound. */
   Rectangle labelRect = label.getBounds();
   labelRect.x = 0;
   labelRect.y = 0;
   /* Paints the label. */
   label.paintImmediately( labelRect );
   /* Sets the value to the progress bar. */
   bar.setValue( iCtr );
   /*
   Creates an object of the Rectangle class that gets the progress bar bound.
   */
   Rectangle progressRect = bar.getBounds();
   progressRect.x = 0;
   progressRect.y = 0;
   /* Paints the progress bar. */
   bar.paintImmediately( progressRect );
}
```

```
    }
    /*
    performTask() - This method provides a time delay for the progress bar creating loop.
    Paramter: ictr
    Return Value: NA
    */
    public void performTask( int iCtr )
    {
       Random random = new Random( iCtr );
       for( int i = 0; i < random.nextFloat() * 1000; i++ )
       {
          /* Runs the loop */
       }
    }
    /*
    actionPerformed() - This method is invoked when the end user clicks the button.
    Parameter: ae - an ActionEvent object containing the details of the event.
    Returns Value: NA
    */
    public void actionPerformed(ActionEvent ae)
    {
       String arg = (String)ae.getActionCommand();
       if(arg.equals(" Cancel "))
       {
          /* Stops the downloading thread. */
          th.stop();
          try
          {
             /* Creates instance of the File class. */
             File f = new File(path);
             /* Performs deletion until the file is deleted. */
             while(f.delete())
             {
                /* Delete the file. */
                f.delete();
             }
             /* Hides the Download Status dialog box. */
             this.setVisible(false);
          }
          catch(Exception e)
          {
             System.out.println("Error in I/O");
          }
       }
       else if(arg.equals(" OK "))
       {
          /* Hides the Download Status dialog box. */
          this.setVisible(false);
       }
    }
}
```

Download this Listing.

In the above code, the ProgressTest() constructor creates the Download Status dialog box, as shown in Figure 3-3:


**Figure 3-3:** The Download Status Dialog Box

If an end user clicks the Cancel button on the Download Status dialog box, the actionPerformed() method is invoked. This method stops the downloading thread. The actionPerformed() method then creates an instance of the File class and deletes the file that is being downloaded.

## Unit Testing

To test the File Download application:

1. 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;
   ```
2. 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;
   ```
3. Copy the FileRemote.java, FileRemoteImpl.java, FileInfo.java, FileServer.java, FileClient.java, and ProgressTest.java files to a folder on your computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. Generate the stub and skeleton files for the File Download application using the following command:
   ```
   rmic FileRemoteImpl
   ```
5. Start the RMI registry using the following command:
   ```
   start rmiregistry
   ```
6. Copy the FileRemote.class, FileRemoteImpl.class, FileInfo.class, FileServer.class, FileRemoteImpl_Stub.class, and FileRemoteImpl_Skel.class files to the folder, named Server, where your server machine is hosted. On the command prompt, use the cd command to move to the folder where you have copied the class files.
7. Run the server of the File Download application using the following command at the command prompt:
   ```
   java FileServer
   ```
8. Copy the FileRemoteImpl.class, FileInfo.class, FileRemoteImpl_Stub.class, FileClient.class, and ProgressTest.class files to the folder, named Client, where your client machine is hosted. On the command prompt, use the cd command to move to the folder where you have copied the class files.
9. Create a java.policy file to authenticate the client to access the remote object using the following command:
   ```
   Policytool
   ```
10. Run the client of the File Download application. You need to specify the IP address of the server on the command line. To execute the client, specify the following command at the command prompt:
    ```
    java FileClient 192.168.0.36
    ```
11. Click the Get File List button on the File Download Application window to display the list of files that are available for download. The File Download Application window with the file list appears, as shown in Figure 3-4:


**Figure 3-4:** Displaying File Download Application

12. Click the Download the File button without selecting any file from the table. An error message appears, as shown in Figure 3-5:


**Figure 3-5:** Error Dialog Box

13. Select a file that you want to download.
14. Click the Download the File button to start the file download process. A Confirm dialog box appears, as shown in Figure 3-6:


**Figure 3-6:** Confirm Dialog Box

15. Click OK in the Confirm dialog box. A Save dialog box opens, as shown in Figure 3-7:

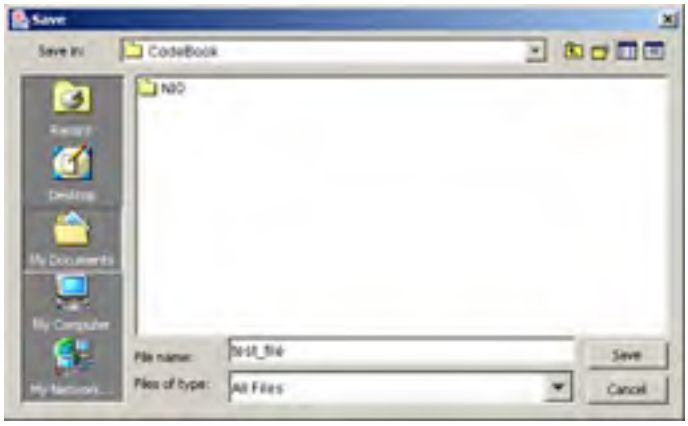


**Figure 3-7:** Save Dialog Box

16. Click the Save button of the Save dialog box. A Download Status dialog box opens, as shown in Figure 3-8:


**Figure 3-8:** Download Status Dialog Box

To stop the download, the end user can click Cancel. After download, a Download Complete dialog box opens, as shown in Figure 3-9:

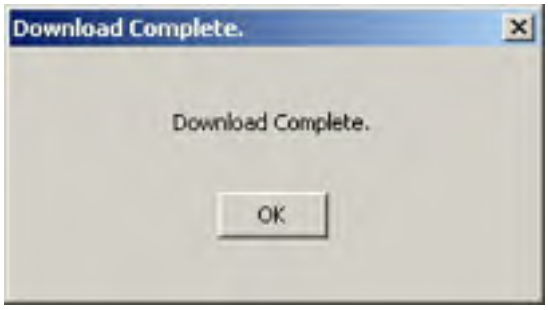


**Figure 3-9:** Download Complete Dialog Box



PREVIOUS NEXT

# Chapter 4: Creating a File Search Application

The New Input/Output (NIO) API supports the java.nio, java.nio.channels, and java.util.regex packages for buffer management, character conversion, and regular expression matching. The java.nio and java.nio.channels packages contain the File, FileChannel, and CharBuffer classes to read and buffer the contents of a file. The java.util.regex package provides the Pattern and Matcher classes to match regular expressions.

This chapter explains how to develop a File Search application, which uses the above-mentioned NIO packages to search for specific files.

## Architecture of the File Search Application

The File Search application enables an end user to search for files that are available at a specific location and contain a specific text pattern. The application also searches for all the instances of the specified text pattern in the files and displays the start position of the pattern to end users.

The File Search application uses the following files:

- `Search.java`: Creates a user interface that helps an end user specify the criteria to search for specific files and view the retrieved result
- `FileList.java`: Searches for files that are available at the specified location and contain the specified text pattern
- `Help.java`: Creates a window that lists the steps to use the File Search application

Figure 4-1 shows the architecture of the File Search application:

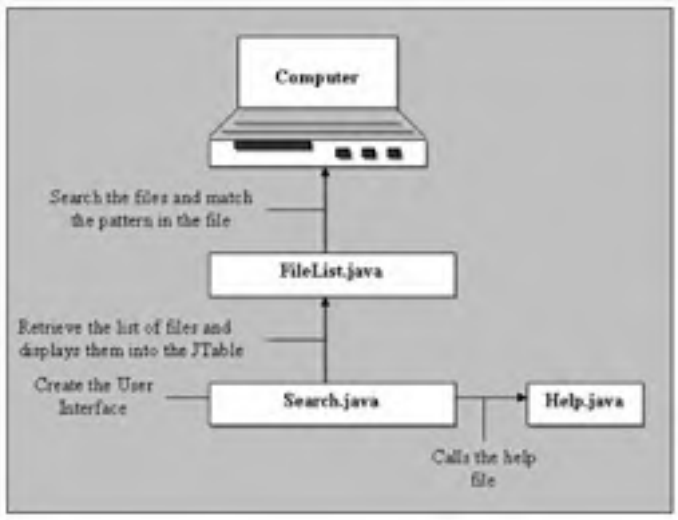


**Figure 4-1:** Architecture of the File Search Application

In the File Search application, the Search.java file calls the FileList.java file to search for files that are present at the specified location and contain the specified text pattern. The FileList.java file searches for these files and passes the search results to the Search.java file. The Search.java file then displays these results to the end user in a tabular format. Before starting the file search process, if an end user opts to view the application help file, the Search.java file calls the Help.java file to display the steps to use the File Search application.

# Creating the User Interface for the File Search Application

The Search.java file helps you create a user interface with a set of text boxes, labels, and buttons for the File Search application. End users can use this interface to specify a location and text pattern to search for specific files. The right pane of the File Search application interface contains an empty space where the file search results are displayed to the end user.

Listing 4-1 shows the contents of the Search.java file:

**Listing 4-1: The Search.java File**

```
/* Imports the required I/O classes. */
import java.io.File;
/* Imports the required Util classes. */
import java.util.Vector;
/* Imports the required AWT classes */
import java.awt.event.*;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.Color;
import java.awt.*;
/* Imports the required Swing classes */
import javax.swing.*;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.LookAndFeel;
import javax.swing.UIManager;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
/*
class Search - This is the main class of the application. This class initializes the
interface and loads all components, such as table, before displaying the result.
Methods:
listFiles() - This method searches for files that meet the specified search criteria.
Usage() - This method displays an error message.
Main() - This method creates the main window of the application and displays all the components and
*/
public class Search extends JFrame implements ActionListener, Runnable
{
   /* Declare objects of the JLabel class. */
   JLabel labelText;
   JLabel labelLook;
   JLabel labelSearch;
   JLabel labelResult;
   /* Declare objects of the JTextField class. */
   JTextField textText;
   JTextField textLook;
   /* Declare objects of the JButton class. */
   JButton search;
   JButton cancel;
   JButton browse;
   JButton help;
   /* Declare objects of the JPanel class. */
   JPanel panel;
   JPanel paneLeft;
   JPanel paneRight;
   JPanel pane1;
   JPanel pane2;
   JPanel pane3;
   JPanel pane4;
   JPanel pane5;
   JPanel p1;
   JPanel p2;
   JPanel p3;
   /* Declare object of the JScrollPane class. */
   JScrollPane scrollpane;
   /* Declare String objects. */
   static String strText;
   static String strLook;
   /* Declare an object of the JTable class. */
   JTable table;
   /* Declare and initialize a counter.*/
   int count = 0;
   int FLAG = 0;
   /*
```

```
Declare and initialize an object of the Object class.
*/
Vector vRow = new Vector();
Vector vCol = new Vector();
public FileList fl = new FileList(this);
public Help h;
/* Declare arrays of the String class.*/
DefaultTableModel model;
Thread thread;
int countFile;
/* Implement a constructor of the Search class. */
public Search()
{
   try
   {
      /*
      Initialize and set the look and feel of the application to Windows look and feel.
      */
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
   }
   catch(Exception e)
   {
      /*
      If an error occurs while loading the Windows look and feel, an error is
      displayed and the application is closed.
      */
      usage();
   }
   /* Set the size of the application frame. */
   setSize(800, 600);
   /* Set the Title of the application frame. */
   setTitle("File Search Utility");
   /*
   Initialize a panel, containing two subpanels, for the application frame.
   */
   panel = new JPanel();
   /* Set the layout of the frame as Grid layout. */
   panel.setLayout(new GridLayout(0, 2));
   /* Add the panel to the frame. */
   getContentPane().add(panel);
   /*
   Initialize the other two subpanels.
   Add the first subpanel (paneLeft) to the left side of the main panel.
   Add the second subpanel (paneRight) to the right side of the main panel.
   */
   paneLeft = new JPanel();
   paneRight = new JPanel();
   panel.add(paneLeft);
   panel.add(paneRight);
   /*
   Set the layout of the left subpanel as Border layout.
   Initialize a new Grid panel.
   Set the layout of the new panel as Grid layout.
   Add this panel to the left subpanel.
   */
   paneLeft.setLayout(new BorderLayout());
   pane5 = new JPanel();
   pane5.setLayout(new FlowLayout(FlowLayout.RIGHT));
   help = new JButton("Help");
   help.addActionListener(this);
   pane5.add(help);
   paneLeft.add(pane5, BorderLayout.SOUTH);
   /*
   Set the layout of the left subpanel as Border layout.
   Initialize a new Grid panel.
   Set the layout of the new panel as Grid Layout.
   Add this panel to the left subpanel.
   */
   pane3 = new JPanel();
   pane3.setLayout(new GridLayout(6,0));
   paneLeft.add(pane3, BorderLayout.NORTH);
   /*
   Initialize a new panel.
   Set the layout of this panel as Flow layout.
   Initialize a new label and add this label to the panel.
   Add this panel to the first grid of the new Grid panel.
   */
   p1 = new JPanel();
   p1.setLayout(new FlowLayout(FlowLayout.LEFT));
   labelLook = new JLabel("Search for file(s) in:");
   p1.add(labelLook);
   labelLook.setFont(new Font("Verdana",Font.BOLD,12));
   pane3.add(p1);
   /*
   Initialize a new panel.
   Set the layout of this panel as Flow layout.
   Initialize a text field and add it to the panel.
   Initialize a button and add it to the panel.
```

```
Add this panel to the second grid of the new Grid panel.
*/
panel = new JPanel();
panel.setLayout(new FlowLayout(FlowLayout.LEFT));
textLook = new JTextField(25);
browse = new JButton("Browse");
browse.addActionListener(this);
panel.add(textLook);
panel.add(browse);
textLook.setFont(new Font("Verdana",Font.PLAIN,12));
pane3.add(panel);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize a new label and add this label to the panel.
Add this panel to the third grid of the new Grid panel.
*/
p2 = new JPanel();
p2.setLayout(new FlowLayout(FlowLayout.LEFT));
labelText = new JLabel("Containing text:");
p2.add(labelText);
labelText.setFont(new Font("Verdana",Font.BOLD,12));
pane3.add(p2);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize a text field and add it to the panel.
Add this panel to the fourth grid of the new Grid panel.
*/
p3 = new JPanel();
p3.setLayout(new FlowLayout(FlowLayout.LEFT));
textText = new JTextField(33);
p3.add(textText);
textText.setFont(new Font("Verdana",Font.PLAIN,12));
pane3.add(p3);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize buttons and add these buttons to the panel.
Add this panel to the fifth grid of the new Grid panel.
*/
pane2 = new JPanel();
pane2.setLayout(new FlowLayout(FlowLayout.LEFT));
search = new JButton("Search");
cancel = new JButton("Cancel");
pane2.add(search);
search.addActionListener(this);
pane2.add(cancel);
cancel.addActionListener(this);
pane3.add(pane2);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize buttons and add these buttons to the panel.
Add this panel to the fifth grid of the new Grid panel.
*/
pane4 = new JPanel();
pane4.setLayout(new FlowLayout(FlowLayout.LEFT));
labelSearch = new JLabel("Search Result(s): ");
labelSearch.setFont(new Font("Verdana",Font.BOLD,12));
labelResult = new JLabel("0 files.");
labelResult.setFont(new Font("Verdana",Font.BOLD,12));
pane4.add(labelSearch);
pane4.add(labelResult);
pane3.add(pane4);
/*
Set the layout of the right subpanel as Border layout.
Initialize an object of the table.
Initialize an object of the scroll panel.
Add the table to the scroll panel.
Add this scroll panel to the right subpanel.
*/
paneRight.setLayout(new BorderLayout());
model = new DefaultTableModel();
vCol.addElement("File Name");
vCol.addElement("Start Position");
vCol.addElement("Path");
vCol.addElement("Size");
model = new DefaultTableModel(vRow, vCol);
table = new JTable(model);
scrollpane = new JScrollPane(table);
paneRight.add(scrollpane, BorderLayout.CENTER);
/* Pack the components of the frame. */
doLayout();
pack();
/*
addWindowListener: It contains the windowClosing() method.
windowClosing: It is called when the end user clicks the Cancel button on the window.
```

```
   It closes the main window.
   we parameter: An object of the WindowEvent class.
   Return Value: NA
   */
   addWindowListener(new WindowAdapter()
   {
      public void windowClosing(WindowEvent we)
      {
         System.exit(0);
      }
   });
}
/*
actionPerformed(): This method is called when the end user clicks the Browse,
Search, or Cancel button.
ae parameter: An ActionEvent object that contains information about the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   /*
   This is executed when an end user clicks the Search button.
   */
   if(ae.getSource() == search)
   {
      countFile = 0;
      count = 0;
      FLAG = 0;
      model.setNumRows(0);
      strText = textText.getText();
      strLook = textLook.getText();
      /*
      If both the text field have the required text, the actionPerformed() method creates
      a new instance of the Thread class, which further calls the listFiles() method.
      */
      if(strText.equals("") == false && strLook.equals("") == false)
      {
         try
         {
            thread = new Thread(this);
            thread.start();
         }
         catch(Exception e)
         {
      System.out.println("Error!" + e);
         }
      }
      /*
      Otherwise, display an Error message box.
      The Error message box is displayed by calling the showMessageDialog()
      method of the JOptionPane class.
      showMessageDialog(): It is called by the system automatically, when required.
      Parameter:
      Parent Name: Name of the parent frame.
      Display Message: Text to be displayed on the message box.
      Title: Sets the title of the message box.
      Message Box Option: Type of message box.
      Return Value: NA
      */
      else
      {
         JOptionPane.showMessageDialog(this, "You must enter the searching text.",
         "Alert Message", JOptionPane.WARNING_MESSAGE);
      }
   }
   /*
   This is executed when the end user clicks the Search button.
   */
   else if (ae.getSource() == browse)
   {

      try
      {
         /* Initialize an object of the JFileChooser class. */
         JFileChooser jfc = new JFileChooser();
         /* Set the mode of the FileChooser box. */
         jfc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
         jfc.showOpenDialog(this);
         /*
         Get the selected file using the getSelectedFile() method.
         */
         File file = jfc.getSelectedFile();
         /*
         Retrieve the value of the absolute path of the file. Store the value into a string.
         */
         String str = file.getAbsolutePath();
         textLook.setText(str);
      }
```

```
            catch(Exception e)
            {
               +System.out.println("Error" + e);
            }
         }
         /*
         This is executed when an end user clicks the Search button.
         */
         else if (ae.getSource() == cancel)
         {
            /*
            This hides and closes all the components of the application.
            */
            search.setEnabled(true);
            FLAG = 1;
         }
         /*
         This is executed when an end user clicks the Help button.
         */
         else if (ae.getSource() == help)
         {
            /* It shows the help utility. */
            h = new Help();
            h.show();
         }
      }
      public void run()
      {
         fl.listFiles(strLook);
      }
      /*
      This is the main method that creates an instance of the Search class and shows the main frame.
      */
      public static void main(String[] args)
      {
              Search s = new Search();
         s.show();
      }
      /*
      A utility method to display the invocation syntax and exit.
      */
      public static void usage()
      {
           System.err.println("Error");
           System.exit(1);
       }
}
```

Download this Listing.

In the above listing, the main() method creates an instance of the Search class. This class generates the main window of the File Search application, as shown in Figure 4-2:

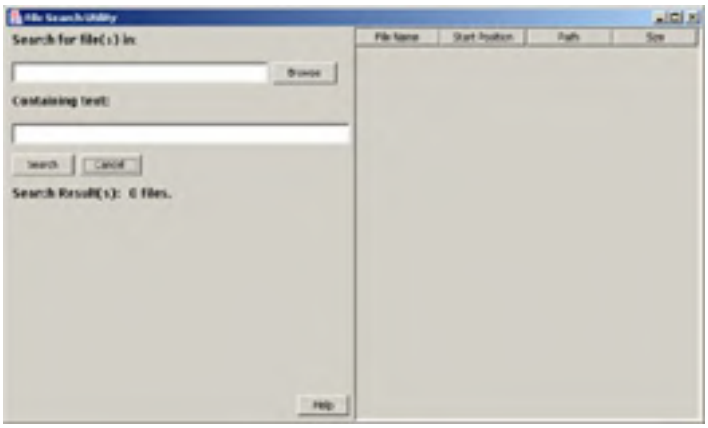

**Figure 4-2:** The File Search Application User Interface

The text boxes in the above figure enable an end user to specify the location and the text pattern to search for specific files. The empty space in the right pane of the window displays the results of the file search.

When an end user clicks any button on the File Search Utility window, the File Search application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate class, based on the button that the end user clicks.

When an end user clicks the Browse button adjacent to the Search for file(s) in text box, the actionPerformed() method creates an instance of the JFileChooser class to open the File dialog box. This dialog box enables the end user to browse to a specific location, where the File Search application should search for the desired files.

After specifying the search criteria, when an end user clicks the Search button, the actionPerformed() method creates an instance of the Thread class and calls the start() method of this class. The start() method further calls the run() method of the Thread class, which calls the listFiles() method to search for files that meet the specified criteria.

The Search.java file declares an object of the JTable class. This object creates a tabular format to display the search results, returned by the listFiles() method, to the end user.

The Cancel button on the File Search Utility window helps an end user stop the file search process before it is complete. When an end user clicks the Cancel button, the actionPerformed() method sets the value of the FLAG variable, declared in the Search class, to 1 and stops the running instance of the Thread class.

If an end user clicks the Help button, the actionPerformed() method creates an instance of the Help class and calls the show() method of this class. The show() method opens the Help window, which displays the sequence of steps to search for files using the File Search application.



PREVIOUS NEXT

# Implementing the Search Functionality

The FileList.java file implements the core functionality of the File Search application. This file reads the location and text pattern that the end user specifies on the File Search Utility window and retrieves information, such as the number of files meeting the search criteria, file names, start position of the specified text pattern within these files, and file sizes. Finally, the FileList.java file passes this information to the Search.java file, which displays this search information to the end user in a tabular format.

Listing 4-2 shows the contents of the FileList.java file:

**Listing 4-2: The FileList.java File**

```
/* Imports the required I/O classes. */
import java.io.FileInputStream;
import java.io.File;
import java.io.IOException;
/* Imports the required NIO classes. */
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.charset.Charset;
import java.nio.charset.CharsetDecoder;
import java.nio.channels.FileChannel;
import java.nio.charset.UnsupportedCharsetException;
/* Imports the required Util classes. */
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;
import java.util.Vector;
/* Imports the required Swing classes */
import javax.swing.JOptionPane;
/*
class FileList - This class is the subclass of the application. It displays the list of files in a t
*/
public class FileList
{
   /* Initialize the object of Search class. */
   Search se;
   int option;
   int count = 1;
   /*
   Declare the default constructor of the FileList class.
   */
   public FileList(Search se)
   {
      this.se = se;
   }
   /*
   listFiles() - This method is called when the end user clicks the Search button of the application
   Parameter:
   dirname: It is a string type that contains the name of the directory.
   Return Value - NA
   */
   public void listFiles(String dirname)
   {
      String encodingName = "UTF-8";
      /* Set the flag value to CASE INSENSITIVE. */
      int flags = Pattern.CASE_INSENSITIVE;
      try
      {
         /*
         Initialize the instance of the Charset class and register the class with the
         encoding name.
         */
         Charset charset = Charset.forName(encodingName);
         /*
         Initialize the pattern object and compile it with the search pattern specified
         by the end user.
         */
         Pattern pattern = Pattern.compile(se.strText, flags);
         try
         {
            /*
            Create a file object that contains the names of all the files and directories.
            */
            File folder = new File(dirname);
            /*
            Store the list of files and directories in a String array.
            */
            String files[] = folder.list();
            for (int i=0; i<files.length-1; i++)
            {
               File file = new File(folder.getAbsolutePath() + File.separator + files[i]);
               /*
               Check if the file object is a file or a directory.
               */
```

```
            if (se.FLAG == 0)
            {
               if (file.isFile())
               {
                  /* This will store the complete text of the file. */
                  CharBuffer chars;
                  try
                  {
                     /*
                     Handle the errors of each file locally and open a file channel to the named
                     */
                     FileInputStream stream = new FileInputStream(file);
                     FileChannel f = stream.getChannel();
                     /*
                     Calls the map() method of the FileChannel class, which returns an
                     object of the ByteBuffer class. The map() method uses the
                     MapMode.READ_ONLY attribute of the FileChannel class to open the files.
                     */
                     ByteBuffer bytes = f.map(FileChannel.MapMode.READ_ONLY, 0, f.size( ));
                     /*
                     We can close the file once it is mapped to the memory.
                     Closing the stream closes the file channel also.
                     */
                     stream.close( );
                     /*
                     Decode the entire ByteBuffer into one big CharBuffer.
                     */
                     chars = charset.decode(bytes);
                  }
                  catch(IOException e)
                  {
                     /*
                     File not found. Print an error message. Move to the next file.
                     */
                     System.err.println(e);
                     continue;
                  }
                  /*
                  A Matcher holds the state of a given pattern and text. Start matching the text.
                  */
                  Matcher matcher = pattern.matcher(chars);
                  while(matcher.find())
                  {
                     try
                     {
                        se.vRow = new Vector();
                        /*
                        Add elements to the row vector.
                        */
                        se.vRow.addElement(file.getName());
                        se.vRow.addElement(Integer.toString(matcher.start()));
                        se.vRow.addElement(file.getParent());
                        se.vRow.addElement(Long.toString(file.length()) + " bytes");
                        /*
                        Add a row to the table model.
                        */
                        se.model.addRow(se.vRow);
                        se.countFile = 1;
                        se.labelResult.setText(Integer.toString(se.model.getRowCount()) + "file(s
               }
               catch(Exception e)
               {
                  System.out.println("Error!" + e);
               }
            }
            }
            else
            /*
            If the file object is a directory, it calls the listFiles() method again.
            */
            listFiles(file.getAbsolutePath());
         }
         else if (se.FLAG == 1)
         {
            break;
         }
      }
   }
   catch (Exception ex)
   {
      /*  Print Error message. Return. */
      ex.printStackTrace();
      return;
      }
   }
   catch(UnsupportedCharsetException e)
   {
      /*
```

```
            Bad encoding name. Print exception name.
            */
            System.err.println("Unknown encoding: " + encodingName);
         }
         catch(PatternSyntaxException e)
         {
            /*
            Bad pattern-matching. Print exception name.
            */
            System.err.println("Syntax error in search pattern: " + e.getMessage());
         }
         if(se.countFile == 0)
         {
            JOptionPane.showMessageDialog(null, "No file found!", "Alert Message",
            JOptionPane.WARNING_MESSAGE);
            se.countFile = 1;
         }
      }
}
```

Download this Listing.

The above listing creates an instance of the FileList class, which accepts the Search class as an input parameter. When an end user clicks the Search button on the File Search Utility window, the listFiles() method is invoked. The dirname parameter of this method stores the file location specified by the end user. The listFiles() method creates a new instance of the Charset class and registers it with Unicode Transformation Format-8 (UTF-8). Next, the method creates an instance of the Pattern class, initializes the pattern object, and stores the text pattern specified by the end user in this pattern object. The listFiles() method creates a file object using the directory location specified by the end user. Next, the listFiles() method calls the list() method of the File class to retrieve a list of files and subdirectories available within the specified directory location and stores this list in the files[ ] String array.

The listFiles() method now iterates through the files[ ] String array to check for the type of files. If the file is of the type File class, the listFiles() method creates a new instance of the FileInputStream class and calls the getChannel() method of this class. The getChannel() method returns an object of the FileChannel class. Next, the listFiles() method calls the map() method of the FileChannel class, which returns an object of the ByteBuffer class. The listFiles() method then calls the close() method of the FileInputStream class to close the file after it is mapped to the memory.

After mapping the files available at the specified location and containing the specified text pattern to the memory, the listFiles() method calls the matcher() method of the Pattern class. The Pattern class contains the text pattern specified by the end user. The matcher() method returns an object of the Matcher class. The listFiles() method calls the find() method of the Matcher class to obtain information about the files that contain the specified text pattern. The search result, which contains the total number of files along with file information, such as file name and file size, is then returned to the Search class to be displayed to the end user.

# Creating the Help File

The Help.java file enables you to create the Help window of the File Search application. This window displays the steps to use the File Search application to the end user.

Listing 4-3 shows the contents of the Help.java file:

**Listing 4-3: The Help.java File**

```
/*
Import the packages to use their classes in this class.
*/
import java.awt.event.*;
import java.awt.FlowLayout;
import java.awt.Font;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.JDialog;
/* A class for the Help file. */
public class Help extends JDialog implements ActionListener
{
   JLabel label1;
   JTextArea area;
   JScrollPane sp;
   JPanel pane;
   String str;
   String text;
   JButton ok;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   /*
   Define the default constructor of the Help class.
   */
   public Help()
   {
      /*
      Set the size. Initialize the panel and set the layout. Add this panel to the frame.
      */
      setTitle("Help");
      setSize(460, 260);
      setVisible(true);
      setResizable(false);
      gbl = new GridBagLayout();
      getContentPane().setLayout(gbl);
      gbc = new GridBagConstraints();
      /*
      Initialize the Label. Add the label to the panel.
      */
      gbc.gridx = 1;
      gbc.gridy = 1;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.CENTER;
      label1 = new JLabel("File Search Utility Help");
      Font f = new Font("Veradan",Font.BOLD,14);
      /*
      Set the Font style and size of the label caption.
      */
      label1.setFont(f);
      getContentPane().add(label1, gbc);
      /*
      Create a text area and set the text.
      Set the font as Verdana in the text area.
      Set editable false in the text area.
      Add this text area to the scroll pane.
      */
      gbc.gridx = 1;
      gbc.gridy = 2;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.CENTER;
      area = new JTextArea(10, 40);
      text = "The steps to search for a file are:\n\n"
      + "1. Click the Browse button to locate a directory.\n"
      + "2. Enter the text that you want to search in the Containing text text box.\n"
      + "3. Click the Search button to start the search process.\n"
      + "4. Click the Cancel button to stop the search process before it is complete.";
      Font f1 = new Font("Veradan",Font.PLAIN,12);
```

```
        area.setFont(f1);
        area.setText(text);
        area.setLineWrap(true);
        area.setWrapStyleWord(true);
        area.setEditable(false);
        sp = new JScrollPane(area);
        getContentPane().add(sp, gbc);
        /*
        Initialize the OK button. Add action listener.
        */
        gbc.gridx = 1;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        ok = new JButton(" OK ");
        ok.addActionListener(this);
        getContentPane().add(ok, gbc);

    }
    /*
    actionPerformed() This method is called when the end user clicks the OK button.
    Parameters:
    ae - An ActionEvent object containing information about the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent ae)
    {
        /*
        This is executed when the end user clicks the Search button.
        */
        if(ae.getSource() == ok)
        {
            this.setVisible(false);
        }
    }
}
```

Download this Listing.

In the above listing, the Help() constructor creates the Help window, as shown in Figure 4-3:


**Figure 4-3:** The Help Window

When an end user clicks the OK button on the Help window, the actionPerformed() method is invoked. This method sets the visibility of the Help window to False and closes the window.



PREVIOUS NEXT

## Unit Testing

To test the File Search application:

1. 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;
   ```
2. 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;
   ```
3. Copy the Search.java, FileList.java, and Help.java files to a folder on your computer. Next, compile the files using the javac command as follows:
   ```
   javac *.java
   ```
4. To run the File Search application, specify the following command at the command prompt:
   ```
   java Search
   ```
5. Click the Browse button on the File Search Utility window to specify the location where you want to search for files.
6. Specify a text string in the Containing text box to limit your search to those files that contain this text pattern.
7. Click the Search button to start the file search process. The File Search Utility window with the search results appears, as shown in Figure 4-4:

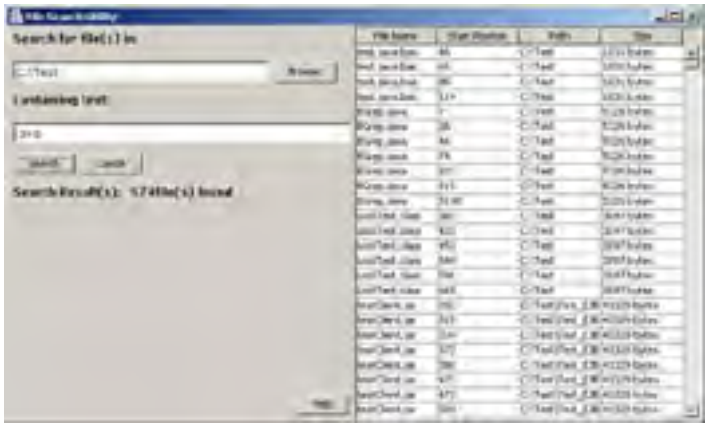

**Figure 4-4:** The File Search Utility Window with File Search Results

The right pane of the window shows the name and size of 974 files that are located within the C:\Test folder and contain the text string, java. The window also displays the start positions of the text pattern, java, in these 974 files.

# Chapter 5: Creating a Printer Management Application

The Java New Input/Output (NIO) API provides the java.nio, java.nio.channels, and java.awt.print packages, which you can use to perform various printer management functions. The java.nio package provides two classes, ByteBuffer and CharBuffer. You can use these classes to buffer a file. To read and write text and image files, you can use two classes provided by java.nio.channels, File and FileChannels. The java.awt.print package provides the PrinterJob and PageFormat classes to specify the properties of a file and print it. You can also use this package to manage the printers connected on the network.

This chapter describes how to develop a Printer Management application using the java.nio, java.nio.channels, and java.awt.print packages. It also explains how to read, write, buffer, format, and print text and image files. The application manages the printers that are connected to the computers that have Windows operating system.

## Architecture of the Printer Management Application

The Printer Management application provides an interface that allows end users to create a book, add a document to the book, specify the document page setup, and simultaneously print the required number of copies of that book. The book interface also allows end users to customize the print settings and use the application to manage the network printers. The application provides a Print dialog box that displays the status of the printer.

The Printer Management application uses the following files:

- `PrintFile.java`: Creates a user interface for the Printer Management application. This class also allows the end user to browse and open a file.
- `CreateBookInterface.java`: Creates a book interface that helps the end user to add documents that are to be printed in a book.
- `PrintComp.java`: Creates the Print dialog box to print the currently open document.

Figure 5-1 shows the architecture of the Printer Management application:

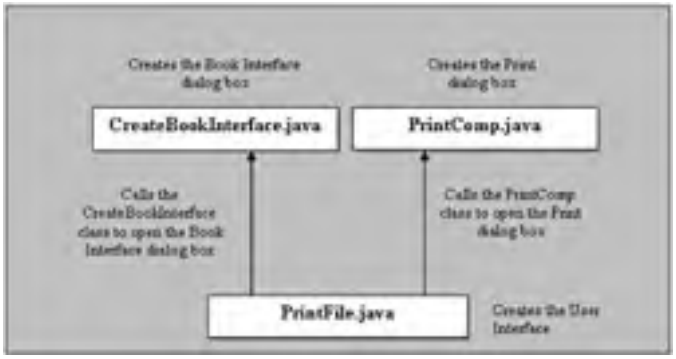


**Figure 5-1:** Architecture of the Printer Management Application

The PrintFile.java file is the main class of the Printer Management application. The File menu of the PrintFile.java file allows end users to open a document, specify its page setup, and print it. If they select the File-> Print->Print Default or File->Page Setup & Print menu item, the PrintFile.java file calls the PrintComp.java file, which prints the currently open text or image file.

If end users select the File-> Print Custom menu item, the PrintFile.java file calls the CreateBookInterface.java file, which creates a book that can be printed. End users can specify the page format and the number of copies of each page they want to print. All the pages added to the book can be printed using one print command.

# Creating the User Interface for the Printer Management Application

You can use the PrintFile.java file to create a user interface that has a menu and a tabbed pane. You can click the Print Text tab from the tabbed pane to open and print a text document. To open and print an image, click the Print Image tab from the tabbed pane. The user interface provides a set of menu items to open a document, specify the page setup, and print that document.

Listing 5-1 shows the contents of the PrintFile.java file:

**Listing 5-1: The PrintFile.java File**

```
/* Imports required swing classes. */
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTabbedPane;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JMenuItem;
import javax.swing.UIManager;
/* Imports required io classes. */
import java.io.File;
import java.io.FileInputStream;
/* Imports required nio classes. */
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
/* Imports required awt classes. */
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowListener;
import java.awt.event.WindowAdapter;
/*
class PrintFile - This class is the main class of the application. This class creates the user inter
Methods:
actionPerformed() - This method is invoked when an end user clicks any button.
main() - This method creates the main window of the application and displays it.
*/
class PrintFile extends JFrame implements ActionListener, Runnable
{
/* Declares object of JTabbedPane class. */
 JTabbedPane tabbedpane;
/* Declares object of Container class. */
 Container container;
/* Declares object of JMenuBar class. */
 JMenuBar menubar;
/* Declares object of JMenu class. */
 JMenu file_menu;
 JMenu print_menuitem;
/* Declares object of JMenuItem class. */
JMenuItem setup_menuitem, exit_menuitem, open_menuitem,prin_default,prin_custom;
/* Declares object of JLabel class. */
 JLabel label;
/* Declares objects of JScrollPane class. */
 JScrollPane image_scrollpane;
 JScrollPane text_scrollpane;
/* Declares object of JTextArea class. */
 JTextArea textarea;
/* Declares objects of JFileChooser class. */
 JFileChooser textfilechooser, imgfilechooser;
/* Declares object of CreateBookInterface class. */
 CreateBookInterface bIn;
 Thread thread1, thread2;
 int count = 0;
/* Implements the constructor of PrintFile class. */
 public PrintFile(String title)
 {
 super(title);
/*
Sets the look and feel of the application to windows.
*/
 try
 {
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
 }
```

```
 catch(Exception e)
 {
System.out.println("Unknown Look and Feel." + e);
 }
/*Sets the size of user interface. */
 setSize(600,400);
/* Sets the resizable property of Jframe to false. */
 setResizable(false);
 bIn = new CreateBookInterface();
/* Adds window listener to the JFrame. */
 addWindowListener(new WindowAdapter()
 {
public void windowClosing(WindowEvent we)
 {
 System.exit(0);
 }
 });
 container = getContentPane();
/*Sets the layout of the container to BorderLayout. */
 container.setLayout(new BorderLayout());
/* Creates objects of JFileChooser class. */
 textfilechooser = new JFileChooser();
 imgfilechooser = new JFileChooser();
/*
Creates and sets the new file filter for selecting the text files.
*/
textfilechooser.setFileFilter(new javax.swing.filechooser.FileFilter()
 {
 public boolean accept(File f)
 {
 if(f.isDirectory())
 {
 return true;
 }
 String name=f.getName();
if ((name.endsWith(".jpg")) ||(name.endsWith(".gif")) || (name.endsWith(".GIF")) || (name.endsWith("
 {
 return false;
 }
 return true;
 }
 public String getDescription(){
 return "Text Files";
 }
 });
/*
Creates and set the new file filter for selecting the image files.
*/
imgfilechooser.setFileFilter(new javax.swing.filechooser.FileFilter ()
 {
/*
This method lets the end user select only JPG and GIF files from the file chooser.
*/
 public boolean accept(File f)
 {
 if (f.isDirectory())
 {
 return true;
 }
 String name = f.getName();
if (name.endsWith(".jpg") || name.endsWith(".gif") || name.endsWith(".GIF") || name.endsWith(".JPG")
 {
 return true;
 }
 return false;
 }
 public String getDescription() {
 return ".jpg, .gif";
 }
 });
/*
Initializes the menu bar. Sets menu bar to the frame.
*/
 menubar = new JMenuBar();
 setJMenuBar(menubar);
/* Initializes the menu. Adds menus to the menu bar. */
 file_menu = new JMenu("File");
 print_menuitem = new JMenu("Print");
 menubar.add(file_menu);
/*
Initializes the menu items and adds the menu item to the particular menu.
*/
 open_menuitem = new JMenuItem("Open");
setup_menuitem = new JMenuItem("Page Setup & Print");
```

```
 exit_menuitem = new JMenuItem("Exit");
prin_default=new JMenuItem("Print Default");
prin_custom=new JMenuItem("Print Custom");
 file_menu.add(open_menuitem);
 file_menu.add(setup_menuitem);
 print_menuitem.add(prin_default);
 print_menuitem.add(prin_custom);
 file_menu.add(print_menuitem);
 file_menu.add(exit_menuitem);
/*Adds the action listener with each menu item.*/
 open_menuitem.addActionListener(this);
 setup_menuitem.addActionListener(this);
 prin_default.addActionListener(this);
 prin_custom.addActionListener(this);
 exit_menuitem.addActionListener(this);
/* Creates an object of JTabbedPane class.*/
 tabbedpane = new JTabbedPane();
/* Creates a new JScrollPane for the text file.*/
text_scrollpane = new JScrollPane(textarea = new JTextArea());
/* Sets the text wrap style of the JTextArea class.*/
 textarea.setLineWrap(true);
 textarea.setWrapStyleWord(true);
/* Adds the JScrollPane to JTabbedPane.*/
tabbedpane.add("Print Text", text_scrollpane );
/* Creates an object of JScrollPane class for the image file.*/
image_scrollpane = new JScrollPane(label =new JLabel());
/* Adds the JScrollPane to JTabbedPane*/
tabbedpane.add("Print Image", image_scrollpane);
/* Adds the JTabbedPane to container.*/
container.add(tabbedpane, BorderLayout.CENTER);
 setVisible(true);
 }
/*
actionPerformed() - This method is called when the end user clicks any button.
Parameters: ae - an ActionEvent object containing the details of the event.
Return Value: NA
*/
 public void actionPerformed(ActionEvent ae)
 {
/*
This section is executed when the end user selects the File-> Page Setup & Print menu item.
*/
 if (ae.getSource()==setup_menuitem)
 {
 count =1;
 thread1 = new Thread(this);
 thread1.start();
 }
/*
This section is executed when the end user selects the File -> Print Default menu item.
*/
 if (ae.getSource()==prin_default)
 {
 count =2;
 thread2 = new Thread(this);
 thread2.start();
 }
/*
This section is executed when the end user selects the File -> Print Custom menu item.
*/
 if(ae.getSource()==prin_custom)
 {
 if (tabbedpane.getSelectedIndex()==0)
 bIn.setComponent(textarea);
 else
 bIn.setComponent(label);
 bIn.setVisible(true);
 }
/*
This section is executed when the end user selects the File->Exit menu item.
*/
 if (ae.getSource()==exit_menuitem)
 {
/* Terminates the application. */
 System.exit(0);
 }
/*
This section is executed when the end user selects the File -> Open menu item.
*/
 if (ae.getSource()==open_menuitem)
 {
/* Checks whether the text's tab is selected in the tabbed pane. */
 if (tabbedpane.getSelectedIndex()==0)
 {
int returnVal = textfilechooser.showOpenDialog(this);
if(returnVal == JFileChooser.APPROVE_OPTION)
```

```
 {
 try
 {
/*
Reads the contents of opened file and sets the contents of the opened file in the text area.
*/
FileInputStream inputStream = new FileInputStream(textfilechooser.getSelectedFile().getAbsolutePath(
FileChannel fileIn=inputStream.getChannel();
long fsize=fileIn.size();
ByteBuffer buffin=ByteBuffer.allocate((int)fsize);
fileIn.read(buffin);
buffin.rewind();
String readLine=new String(buffin.array());
textarea.setText(readLine);
fileIn.close();
inputStream.close();
 }
 catch(Exception ex)
 {}
bIn=new CreateBookInterface();
 }
 }
/* Checks whether the image's tab is selected in the tabbed pane. */
else if (tabbedpane.getSelectedIndex()==1)
 {
int returnVal = imgfilechooser.showOpenDialog(this);
if(returnVal == JFileChooser.APPROVE_OPTION)
 {
 try
 {
label.setIcon(new ImageIcon(imgfilechooser.getSelectedFile().getAbsolutePath()));
 }
 catch(Exception ex)
 {}
 }
 }
 }
 }
public void run()
 {
 if(count == 1)
 {
 if (tabbedpane.getSelectedIndex()==0)
new PrintComp(textarea).pageSetupAndPrint();
 else
new PrintComp(label).pageSetupAndPrint();
 }
 else if(count == 2)
 {
if (tabbedpane.getSelectedIndex()==0)
 PrintComp.printComponent(textarea);
 else
 PrintComp.printComponent(label);
 this.setVisible(true);
 }
 }
/*
main() - This method creates the main window of the user interface and displays it.
Parameters:
args[] - Contains any command line arguments passed.
Return Value: NA
*/
 public static void main(String[] args)
 {
PrintFile frame = new PrintFile("Printer Management Application");
 }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the PrintFile.java class. This class generates the main window of the Printer Management application, as shown in Figure 5-2:

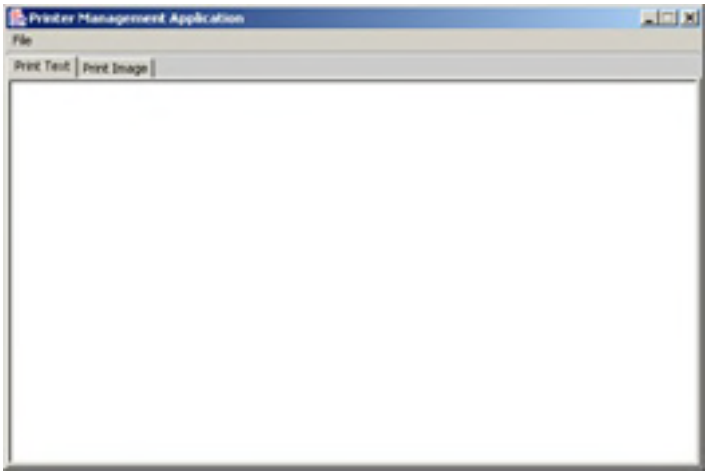


**Figure 5-2:** The Printer Management Application User Interface

When the end user selects the File menu, the options associated with it appear, as shown in Figure 5-3:

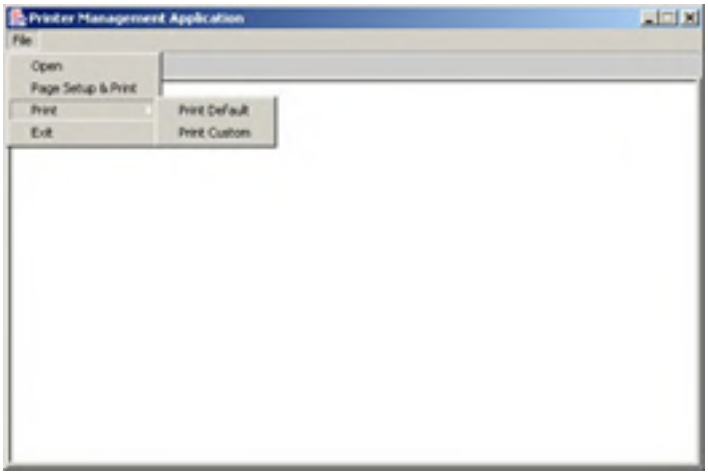


**Figure 5-3:** The File Menu of the Printer Management Application

To open a file, end users need to select the File-> Open menu option. The Open dialog box appears. When an end user selects the file and clicks the Open button of Open dialog box, the content of the selected file is displayed in the text area.

If end users select the File -> Page Setup & Print menu option, the actionPerformed() method sets the count value to 1 and calls the start() method of the Thread class to start the thread, thread1.

If the File->Print->Print Default menu option is selected, the actionPerformed() method sets the count value to 2 and calls the start() method of the Thread class to start the thread, thread2.

When the end user selects the File->Print->Print Custom menu option, the actionPerformed() method calls the setComponent() method of the CreateBookInterface class and opens a Book Interface dialog box.

The application terminates if the end user selects the File->Exit menu option.

The methods defined in the above listing are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the menu option the end user selects. This method is invoked when an end user selects any option from the File menu.
- `run()`: Checks the count value. If the count value is 1, this method creates an instance of the PrintComp class and calls the pageSetupAndPrint() method of the PrintComp class. If the count value is 2, the run() method creates an instance of PrintComp class.


◄ PREVIOUS NEXT ►

# Implementing the Print Functionality

The PrintComp.java file implements the Printable interface of Java. This file allows an end user to print a document through the network printer.

Listing 5-2 shows the contents of the PrintComp.java file:

**Listing 5-2: The PrintComp.java File**

```
/* Imports required awt classes. */
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;
import java.awt.print.PageFormat;
/* Imports required swing classes. */
import javax.swing.JComponent;
/*
class PrintComp - This class is the subclass of the application. This class implements the Printable
Methods:
printComponent() - Invokes the print() method of PrintComp class and prints a component.
pageSetupAndPrint() - Allows the end user to specify the page setup and print the page.
print() - Prints the page.
print()-Prints a graphic at the specified page index in the specified page format.
*/
class PrintComp implements Printable
{
/* Declares object of the JComponent class.*/
 private JComponent component;
/* Declares object of the PrinterJob class.*/
PrinterJob printJob = PrinterJob.getPrinterJob();
/* Declares object of the PageFormat class*/
 PageFormat pageFormat= printJob.defaultPage();
/*
printComponent - This method is called when the end user clicks the Print Default menu item in the u
Parameters:
c - An object of the JComponent class.
Return Value: NA
*/
 public static void printComponent(JComponent c)
 {
 new PrintComp(c).print();
 }
/* Implements the constructor of the PrintComp class. */
 public PrintComp(JComponent component)
 {
 this.component= component;
 }
/*
pageSetupAndPrint - This method is called when the end user clicks the Page Setup and Print button.
Parameters: NA
Return Value: NA
*/
 public void pageSetupAndPrint()
 {
/* Initializes the object of the PageFormat class. */
 pageFormat = printJob.pageDialog(pageFormat);
/* Sets the object of the PrinterJob class to printable. */
 printJob.setPrintable(this, pageFormat);
 if (printJob.printDialog())
 {
 try
 {
/* Prints the document. */
 printJob.print();
 }
 catch(PrinterException pe)
 {
System.out.println("Error in printing !!! " + pe);
 }
 }
 }
/*
print - This method is called when the end user clicks the Print button.
Parameters: NA
Return Value: NA
*/
 public void print()
 {
/* Sets the object of the PrinterJob class to printable. */
```

```
 printJob.setPrintable(this, pageFormat);
/* Checks the return value of PrintDialog. */
 if (printJob.printDialog())
 {
 try
 {
/* Prints the document. */
 printJob.print();
 }
 catch(PrinterException pe)
 {
System.out.println("Error in printing !!! " + pe);
 }
 }
 }
/*
print() - This method defines a Printable interface.
Parameters:
g - Represents the object of the Graphics class.
pf - Represents the object of the PageFormat class.
index - Represents an index of the page.
Return Value: int PAGE_EXIST
*/
public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
 {
/*
Creates an object of the Graphics2D class and convert simple graphics to 2D graphics.
*/
 Graphics2D g2 = (Graphics2D)g;
/*Gets size of document. */
 Dimension d = component.getSize();
/* Gets the width of document in pixels. */
 double componentWidth = d.width;
/* Gets the height of document in pixels. */
 double componentHeight = d.height;
/* Gets the height of printer page. */
double pageHeight = pf.getImageableHeight();
/* Get the width of printer page*/
double pageWidth = pf.getImageableWidth();
/*
Sets the value of scale for the document to be printed.
*/
 double scale = pageWidth/componentWidth;
int pages = (int)Math.ceil(scale * componentHeight / pageHeight);
/* Does not print empty pages. */
 if(pageIndex >= pages)
 {
 return Printable.NO_SUCH_PAGE;
 }
/*
Shifts the graphic to line up with the beginning of print-imageable region.
*/
g2.translate(pf.getImageableX(), pf.getImageableY());
/*
Shifts the graphic to line up with the beginning of the next page to print.
*/
 g2.translate(0f, -pageIndex*pageHeight);
/* Scales the page so that the width fits. */
 g2.scale(scale, scale);
/* Repaints the page. */
 component.paint(g2);
 return Printable.PAGE_EXISTS;
 }
}
```

Download this Listing.

In the above code, the constructor of the PrintComp class uses the object of the JComponent class to load the text or image file. The methods defined in the listing are:

- `printComponent()`: Calls the print() method to print the document.
- `pageSetupAndPrint()`: Initializes the object of the PageFormat class and sets the object of the PrinterJob class to printable. This method then calls the print() method of the PrinterJob class.
- `print()`: Calls the print() method of the PrinterJob class.
- `print(Graphics g, PageFormat pf, int pageIndex)`: Creates an object of the Graphics2D class and converts simple graphics to 2D ones. This method then gets the size of the document and sets its scale value. Next, the method calls the translate() and scale() methods of the Graphics2D class. Finally, the print() method calls the paint() method of the Component class to paint the document with new dimensions and returns Printable.PAGE_EXISTS.

# Creating the Book Interface

The CreateBookInterface.java file helps you to create a book interface for the document that is to be printed.

Listing 5-3 shows the contents of the CreateBookInterface.java file:

**Listing 5-3: The CreateBookInterface.java File**

```
/* Imports java.awt package classes. */
import java.awt.Graphics;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowListener;
import java.awt.event.WindowAdapter;
/* Imports awt.print package classes. */
import java.awt.print.Book;
import java.awt.print.PrinterJob;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
/* Imports javax.swing package classes. */
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JCheckBox;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.DefaultListModel;
import javax.swing.ButtonGroup;
/* Imports java.util package classes. */
import java.util.Enumeration;
import java.util.Vector;
/*
class
CreateBookInterface - This class is the subclass of the application. This class creates a book for t

Methods:
actionPerformed() - This method is invoked when an end user clicks any button.
setComponent() - This method sets the component to the currently opened document.
createBook() - This method is called when the end user clicks the Print Custom menu option.
addPagesInBook()- This method is called when the end user clicks the Add Pages In Book button.
printBook()- This method is called when the end user clicks the Print button.
*/
public class CreateBookInterface extends JFrame implements ActionListener
{
/* Declares object of JComponent class. */
 JComponent component;
/* Declares object of Book class. */
 Book book = null;
/* Declares object of DefaultListModel class. */
 DefaultListModel dlm;
/* Declares object of PrinterJob class. */
 PrinterJob printJob = null;
/* Declares object of ButtonGroup class.*/
 ButtonGroup bg;
/* Declares object of JCheckBox class.*/
 JCheckBox lscape = null;
 JCheckBox potrait=null;
/* Declares object of JTextField class. */
 JTextField pageNo = null;
/* Declares object of JList class. */
 JList bookItemList = null;
/* Declares object of JButton class. */
 JButton add, print, reset;
 String format;
 int pageNum;
/* Declares object of BookPrintable class. */
 BookPrintable bookPrintable;
/*
Implements the constructor of the CreateBookInterface class.
*/
 CreateBookInterface()
 {
/*Sets the title of the CreateBookInterface class. */
```

```
 setTitle("Book Interface");
/* Sets the resizable property to false. */
 setResizable(false);
/*
Adds the window closing event to the CreateBookInterface class.
*/
 addWindowListener(new WindowAdapter()
 {
public void windowClosing(WindowEvent we)
 {
 setVisible(false);
 }
 });
/* Sets the size of the CreateBookInterface class. */
 setSize(400,250);
 setLocation(50,50);
/* Initializes the object of the BookPrintable class. */
 bookPrintable = new BookPrintable();
/* Creates a new ButtonGroup object. */
 bg = new ButtonGroup();
/* Creates a new object of the DefaultListModel class. */
 dlm = new DefaultListModel();
/* Creates a new object of the JPanel class. */
 JPanel selectPanel = new JPanel();
/* Sets the layout of the Jpanel class. */
selectPanel.setLayout(new BorderLayout());
/* Creates a new object of the JPanel class. */
 JPanel formatPanel = new JPanel();
/* Sets the layout of the Jpanel class. */
formatPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
/* Adds a new JLabel to the panel. */
formatPanel.add(new JLabel("Page Format : "));
/* Creates new JCheckBox objects. */
 lscape=new JCheckBox("Landscape",true);
 potrait=new JCheckBox("Potrait",false);
/* Adds the JCheckBox objects to ButtonGroup. */
 bg.add(lscape);
 bg.add(potrait);
/*
Adds the objects of JCheckBox to the JPanel, formatPanel.
*/
 formatPanel.add(lscape);
 formatPanel.add(potrait);
/* Adds formatPanel to the panel. */
 selectPanel.add("North", formatPanel);
/* Creates a new object of the JPanel class. */
 JPanel pageNoPanel = new JPanel();
/* Sets the layout of the JPanel class. */
pageNoPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
/* Adds a new JLabel to the JPanel,pageNoPanel. */
pageNoPanel.add(new JLabel("Number of Pages : "));
/* Creates a new object of the JTextField class. */
 pageNo = new JTextField(5);
/* Adds JTextBox to the JPanel,pageNoPanel. */
 pageNoPanel.add(pageNo);
/*
Adds the pageNoPanel to the selectPanel panel.
*/
 selectPanel.add("Center", pageNoPanel);
 add = new JButton("Add in the Book");
/* Adds the action listener to the add button. */
 add.addActionListener(this);
/* Adds the add button to the selectPanel panel. */
 selectPanel.add("South", add);
getContentPane().add("North", selectPanel);
/* Creates an object of the JList class. */
 bookItemList = new JList(dlm);
/* Sets the background color of the JList object to gray. */
 bookItemList.setBackground(Color.gray);
getContentPane().add("Center", new JScrollPane(bookItemList));
/* Creates a new JPanel, ctrlPanel. */
 JPanel ctrlPanel = new JPanel();
/*
Sets the layout of the ctrlPanel JPanel to GridLayout.
*/
 ctrlPanel.setLayout(new GridLayout(1,2));
 print = new JButton("Print");
 print.setEnabled(false);
/* Adds action listener to the print JButton. */
 print.addActionListener(this);
 ctrlPanel.add(print);
 reset = new JButton("Reset");
 reset.setEnabled(false);
/* Adds action listener to the reset JButton. */
 reset.addActionListener(this);
 ctrlPanel.add(reset);
 getContentPane().add("South", ctrlPanel);
/* Invokes the createBook() method*/
```

```
 createBook();
 setVisible(false);
 }
/*
actionPerformed() - This method is called when the end user clicks any button.
Parameters: ae - An ActionEvent object containing details of the event.
Return Value: NA
*/
 public void actionPerformed(ActionEvent ae)
 {
/*
This section is executed when the end user clicks the Add in the Book button.
*/
if(ae.getActionCommand().equals("Add in the Book"))
 {
/*
Gets all the elements added to the ButtonGroup, bg in an object of the Enumeration class.
*/
 Enumeration enum=bg.getElements();
/*
The while loop runs until there is any element in the enumeration.
*/
 while(enum.hasMoreElements())
 {
/*
Retrieves the next element of the enumeration in an object of the JCheckBox class.
*/
JCheckBox button=(JCheckBox)enum.nextElement();
/* Checks if the check box is selected.*/
 if (button.isSelected())
 {
/* Gets the text of the check box in a string.*/
format=button.getText();
 }
 }
 try
 {
pageNum = Integer.parseInt(pageNo.getText().trim());
 }
 catch (Exception ex)
 {
System.out.println("Error : Invalid entry of page number.");
 return;
 }
 print.setEnabled(true);
/* Adds the currently open document to the book.*/
 addPagesInBook(format, pageNum);
 }
/*
This section is executed when the end user clicks the Print button.
*/
else if(ae.getActionCommand().equals("Print"))
 {
 reset.setEnabled(true);
 print.setEnabled(false);
 setComponent(null);
/* Invokes the printBook() method.*/
 printBook();
 }
/*
This section is executed when the end user clicks the Reset button.
*/
else if(ae.getActionCommand().equals("Reset"))
 {
 print.setEnabled(true);
 reset.setEnabled(false);
/* Clears the JList.*/
 dlm.clear();
/* Clears the book. */
 book = null;
/* Invokes the createBook() method*/
 createBook();
 }
 }
/*
setComponent() - This method sets the value of the object of the JComponent class.
Parameters: component - An object of the JComponent class.
Return Value: NA
*/
 void setComponent(JComponent component)
 {
 this.component= component;
 }
/*
createBook() - This method creates a book to add the document to be printed.
Parameters: NA
```

```
Return Value: NA
*/
 void createBook()
 {
 book = new Book();
 printJob = PrinterJob.getPrinterJob();
 }
/*
createBook() - This method is called when the end user clicks the Add in the Book button.
Parameters:
format- A string that represents the format in which the document to be printed is added to the book
pageNumber- An integer representing the number of pages to be added to the book.
Return Value: NA
*/
void addPagesInBook(String format, int pageNumber)
 {
/* Creates an object of PageFormat class. */
PageFormat pageFormat = printJob.defaultPage();
/* Checks if the Potrait check box is checked. */
 if(format.equals("Potrait"))
 {
pageFormat.setOrientation (PageFormat.PORTRAIT);
 }
/* Checks if the Landscape check box is checked. */
 else if(format.equals("Landscape"))
 {
pageFormat.setOrientation (PageFormat.LANDSCAPE);
 }
/* Appends the document to be printed to the book. */
book.append(bookPrintable,pageFormat, pageNumber);
String temporarySt=new String("Book Item Added : Page Format : "+format+" : "+" Number of Pages : "+
/*
Adds the details of the document to be printed to a JList.
*/
 dlm.addElement(temporarySt);
 }
/*
printBook() - This method is called when the end user clicks the Print button.
Parameters: NA
Return Value: NA
*/
 void printBook()
 {
 printJob.setPageable(book);
 try
 {
/* Calls the print() method to print a book. */
 printJob.print();
 }
 catch(PrinterException pe)
 {
System.out.println("Error in printing !!! " + pe);
 }
 }
/*
class BookPrintable - This class is the subclass of the application. This class implements the Print
Methods:
print()-This method is called when the end user prints a document.
*/
 class BookPrintable implements Printable
 {
/*
print() - This method defines the Printable interface.
Parameters:
g - Represents the object of the Graphics class.
pf - Represents the object of the PageFormat class.
index - Represents an index of a page.
Return Value: int PAGE_EXIST
*/
public int print(Graphics g, PageFormat pf, int pageIndex)
 {
/* Implements print code here. */
 return Printable.PAGE_EXISTS;
 }
 }
}
```

Download this Listing.

In the above code, the CreateBookInterface class allows you to create a user interface to create and print a book, as shown in Figure 5-4:

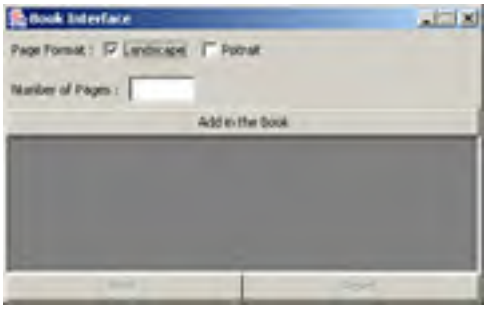


**Figure 5-4:** The Book Interface Window

The methods defined in the above code are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the button the end user clicks in the Book Interface dialog box. If the end user clicks the Add in the Book button, the actionPerformed() method calls the addPagesInBook() method. If the Print button is clicked, the actionPerformed() method calls the printBook() method. When the end user clicks the Reset button, the actionPerformed() method calls the clear() method of the JComponent class and then calls the createBook() method of Book class.
- `createBook()`: Initializes the object of the Book class and gets the object of the PrinterJob class using the getPrinterJob() method.
- `addPagesInBook()`: Uses format and paneNumber as input parameters. The addPagesInBook() method creates an object of the PageFormat class. This method then checks the page format style and sets the appropriate page orientation. Next, the addPagesInBook() method appends to the book the document that is to be printed. Finally, the addPageInBook() method adds the elements to the list box.
- `printBook()`: Calls the setPageable() method of the PrinterJob class to set the page. The printBook() method calls the print() method to print the file.

The CreateBookInterface class contains a sub class called BookPrintable, which implements the Printable interface of the book. The BookPrintable class calls the print() method to print the file. You can define the print() method to print a customized book.



◄ PREVIOUS NEXT ►

# Unit Testing

To test the Printer Management application:

1. 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;
   ```
2. 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;
   ```
3. Copy the PrintFile.java, PrintComp.java, and CreateBookInterface.java files to a folder on your computer. On the command prompt, use the cd command to move to that folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Printer Management application, specify the following command at the command prompt:
   ```
   java PrintFile
   ```
5. Click the Print Text tab in the user interface of the Printer Management application.
6. Select the File->Open command from the File menu of the Printer Management Application. Browse and open the text file that is to be printed using the Open dialog box. The selected file appears in JTabbedPane, as shown in Figure 5-5:

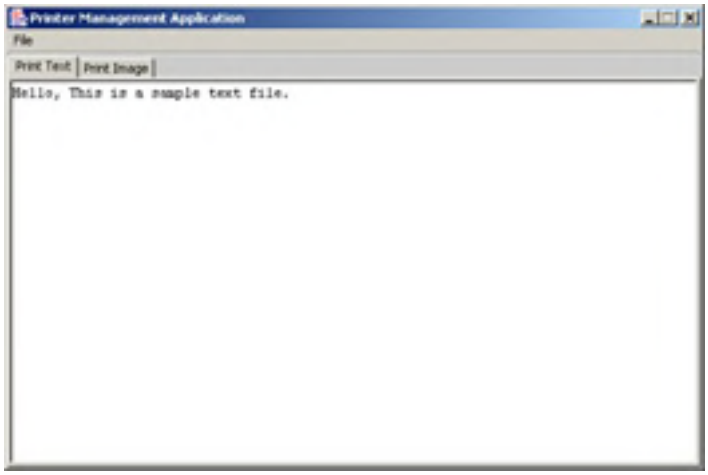


**Figure 5-5:** Printer Management Application Window

7. Select the File->Page Setup & Print command. The Page Setup dialog box opens, as shown in Figure 5-6:


**Figure 5-6:** Page Setup Dialog Box

8. Click the Printer button on the Page Setup dialog box. A new Page Setup dialog box appears, which displays the name, status, type and location of the current printer, as shown in Figure 5-7:

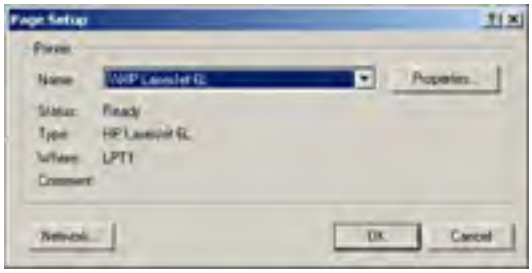


**Figure 5-7:** Displaying the Printer Status

9. Click the Network button to displays all the printers available on the network, as shown in Figure 5-8:

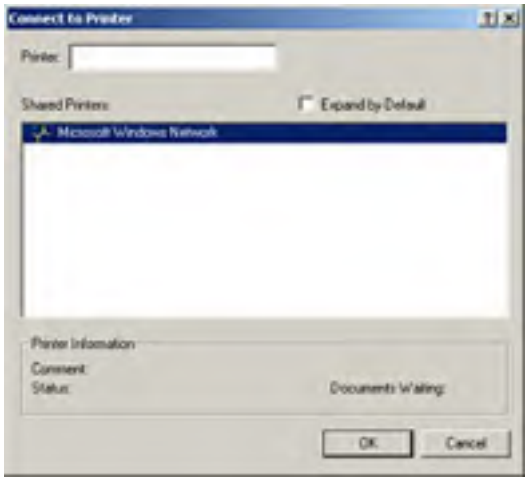


**Figure 5-8:** Connect to Printer Dialog Box

10. Click the OK button on the Page Setup dialog box. A Print dialog box opens, as shown in Figure 5-9:

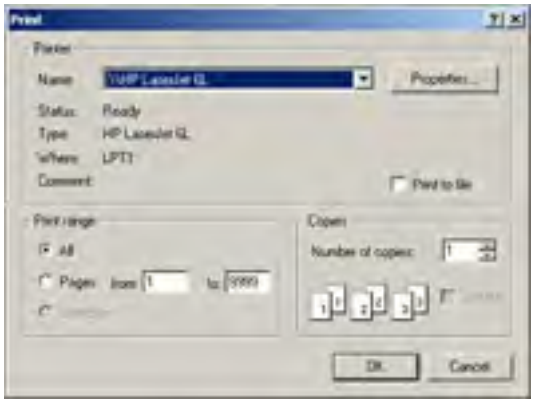


**Figure 5-9:** Print Dialog Box

11. Click OK to print the current file.
12. Click the Print Image tab. The image view of the Printer Management application appears, as shown in Figure 5-10:

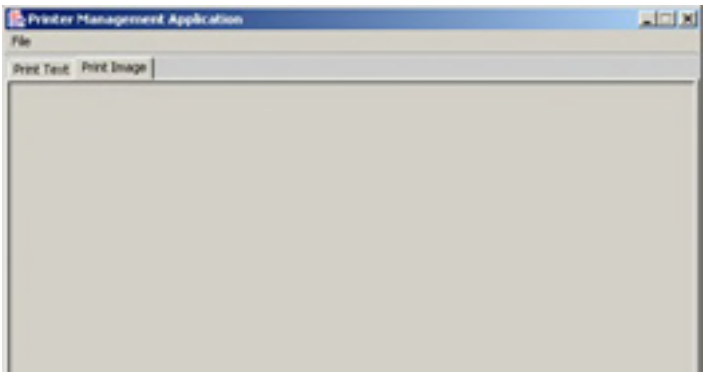

**Figure 5-10:** Printer Management Application with Print Image Tabbed Pane

13. Select File->Open to open an image in the Print Image tabbed pane, as shown in Figure 5-11:

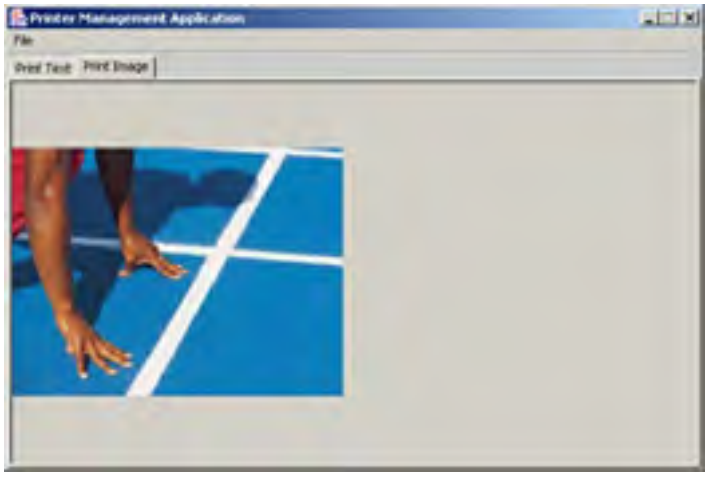


**Figure 5-11:** Displaying an Image File in the Print Image Tabbed Pane

14. Select File->Print->Print Default to print the current page. Select File->Print->Print Custom to print a customized book, as shown in Figure 5-4. When the end user selects the page format using the Page Format check box and specifies the number of pages in Number of Pages test box of the Book Interface dialog box. Now, clicks the Add in the Book button in the Book Interface dialog box. The results are displayed in the list box of the Book Interface, as shown in Figure 5-12:

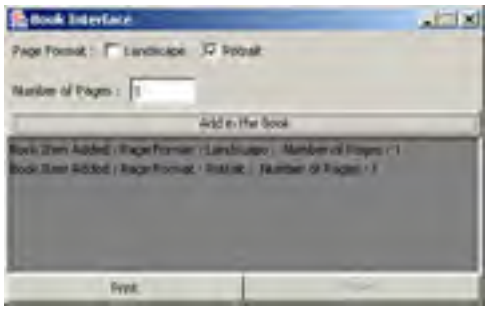


**Figure 5-12:** Displaying a Book Interface dialog

15. Click the Print button to print the book items.

# Chapter 6: Creating a Text Editor Application

The New Input/Output (NIO) API provides the java.nio and java.nio.channels packages for buffer management, advance I/O file system, and file locking mechanism. The java.nio package contains the ByteBuffer classes that allow you to store the contents of a text file. To read, write, and lock the text files, you can use the FileLock and FileChannel classes available in the java.nio.channels package.

This chapter explains how to develop a Text Editor application using the java.nio and java.nio.channels packages and create, edit, print, and save text files.

## Architecture of the Text Editor Application

The Text Editor application allows you to read, write, print, and lock a specific text file. You can also change the color and font of the text file. The Text Editor application provides various editing and formatting features, such as cut, copy, paste, undo, redo, select all, find, and word wrap.

The Text Editor application uses the following files:

- `Editor.java`: Creates a user interface that contains a menu bar and a text area. There are five menus on the menu bar: File, Edit, Format, Locks, and Help. Each menu contains several menu items. You can use these items to perform various tasks, such as opening, saving, or printing a file.
- `ActionPerform.java`: Implements the commands to create, open, save, lock, or copy a file. It also implements commands to find a word in a file. In addition, this class also implements the commands to cut, copy, paste, undo, redo, set font, set text color, and insert date in the file.
- `FontClass.java`: Creates a font dialog box that provides options to change the type, size, and style of the font in the text file.
- `ColorClass.java`: Creates a color dialog box with sliders. You can use this dialog box to specify the Red Green Blue (RGB) value and change the color of the file text.
- `PrintClass`: Opens a print dialog box that allows you to print a document. You can also use this dialog box to change the properties of the document, such as page layout and paper quality.
- `Help.java`: Creates a help dialog box that lists the steps to use the Text Editor.

Figure 6-1 shows the architecture of the Text Editor application:

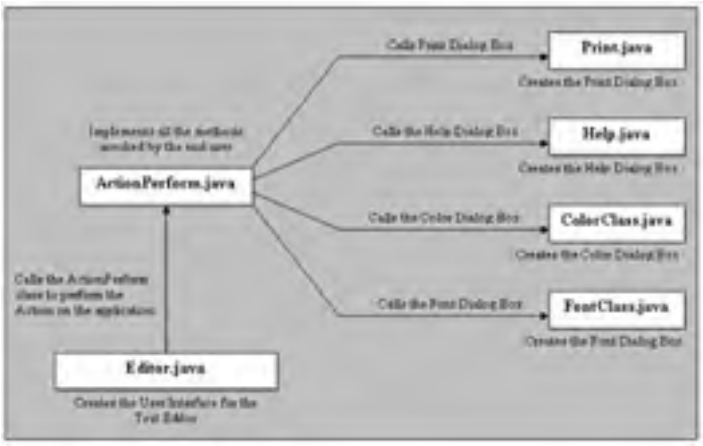


**Figure 6-1:** Architecture of the Text Editor Application

The Editor.java file calls ActionPerform.java to perform the file edit, format, and lock operations. After the ActionPerform.java file implements these operations, it returns the results to the Editor.java file. The Editor.java file then displays the results to the end user. The ActionPerform.java file calls the PrintClass.java file to print a file. To change the font type, size, and style of the text file, the ActionPerform.java file calls the FontClass.java file. The ActionPerform.java file calls the ColorClass.java file to change the color of the file text.

If an end user selects the help file option, the Editor.java file calls the ActionPerform.java file, which, in turn, calls the Help.java file. The Help.java file lists the steps required to use the Text Editor.

# Creating the User Interface for the Text Editor Application

The Editor.java file helps create a user interface with a menu bar and a text area for the Text Editor application. Using this interface, the end user can create, open, save, lock, edit, or format a new file or the existing file. The user interface also contains a text area where the contents of the new file or existing file are displayed to the end user.

Listing 6-1 shows the contents of the Editor.java file:

**Listing 6-1: The Editor.java File**

```
/* Imports the java.net package class. */
import java.net.URL;
/* Imports javax.swing.undo package classes. */
import javax.swing.undo.UndoManager;
import javax.swing.undo.CannotUndoException;
import javax.swing.undo.CannotRedoException;
/* Imports javax.swing package classes. */
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.UIManager;
import javax.swing.Action;
import javax.swing.KeyStroke;
import javax.swing.AbstractAction;
/* Imports java.awt package class. */
import java.awt.BorderLayout;
/* Imports java.awt.event package classes. */
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
/* Imports javax.swing.event package classes. */
import javax.swing.event.UndoableEditListener;
import javax.swing.event.UndoableEditEvent;
/*
Class Editor - This class is the main class of the application. This class initializes
the interface and loads all components like text area and menu bar on the main window.
Fields:
   menu - Creates the menu bar.
   file - Creates the File menu.
   edit - Creates the Edit menu.
   format - Creates the Format menu.
   lock - Creates the Lock menu.
   help - Creates the Help menu.
   newfile - Creates the New menu item.
   open - Creates the Open menu item.
   print - Creates the Print menu item.
   save - Creates the Save menu item.
   saveas - Creates the Save As menu item.
   exit - Creates the Open Exit item.
   cut - Creates the Cut menu item.
   copy - Creates the Copy menu item.
   paste - Creates the Paste menu item.
   find - Creates the Find menu item.
   selectAll - Creates the Select All menu tem.
   datetime - Creates the Date/Time menu item.
   font - Creates the Open Font item.
   color - Creates the Open Color item.
   about - Creates the Open Help item.
   area - Creates an object the JTextArea class that contains the file contents.
   panel - Creates the panel menu item that contains all the AWT and swing components.
   scrollpane - Creates an object of the JScrollPane class that is added to the text area.
   str - Creates the String object.
   wordWrap - Creates the Word Wrap menu item.
   exclusiveLock - Creates the Exclusive menu item.
   shareLock - Creates the Share menu item.
Methods:
   addWindowListener() - This method is called when end user clicks the window close button.
   actionPerformed() - This method is invoked when an end user selects any command from the menu bar
   undoableEditHappened() - This method is called when an end user performs undo and redo operation.
   main() - This method creates the main window of the application and displays it.
*/
public class Editor extends JFrame implements ActionListener, UndoableEditListener, Runnable
{
   /* Creates an object of the ActionPerform class. */
   public ActionPerform action = new ActionPerform(this);
```

```
/* Creates an object of the UndoManager class. */
UndoManager undoManager = new UndoManager();
/* Creates an instance of the UndoAction class. */
UndoAction undoAction = new UndoAction();
/* Creates an instance of the RedoAction class. */
RedoAction redoAction = new RedoAction();
/* Declares an object of the JMenuBar class. */
JMenuBar menu;
/* Declares objects of the JMenu class. */
JMenu file;
JMenu edit;
JMenu format;
JMenu lock;
JMenu help;
/* Declares objects of the JMenuItem class. */
JMenuItem newfile;
JMenuItem open;
JMenuItem print;
JMenuItem save;
JMenuItem saveas;
JMenuItem exit;
JMenuItem cut;
JMenuItem copy;
JMenuItem paste;
JMenuItem find;
JMenuItem selectAll;
JMenuItem datetime;
JMenuItem font;
JMenuItem color;
JMenuItem about;
/* Declares an object of the JTextArea class. */
JTextArea area;
/* Declares an object of the JPanel class. */
JPanel panel;
/* Declares an object of the JScrollPane class. */
JScrollPane scrollpane;
/* Declares an object of the String class. */
String str;
/* Declares objects of the JCheckBoxMenuItem class. */
JCheckBoxMenuItem wordWrap;
JCheckBoxMenuItem exclusiveLock;
JCheckBoxMenuItem shareLock;
/* Declares objects of the Thread class. */
Thread t1;
Thread t2;
Thread t3;
int act;
/* Defines the constructor of the Editor class. */
public Editor()
{
   /*
   Initializes the menu bar and sets the menu bar to the frame.
   */
   menu = new JMenuBar();
   setJMenuBar(menu);
   setTitle("Text Editor");
   /*
   Initializes the menu and adds the menus to the menu bar.
   */
   file = new JMenu("File");
   menu.add(file);
   edit = new JMenu("Edit");
   menu.add(edit);
   format = new JMenu("Format");
   menu.add(format);
   lock = new JMenu("Locks");
   menu.add(lock);
   help = new JMenu("Help");
   menu.add(help);
   /* Sets the mnemonic to the menu. */
   file.setMnemonic('f');
   edit.setMnemonic('e');
   lock.setMnemonic('l');
   format.setMnemonic('o');
   help.setMnemonic('h');
   /*
   Initializes the menu items.
   Adds the menu item to the particular menu.
   Adds the action listener with each menu item.
   Sets accelerator to each menu item.
   */
   /* File->New */
   newfile = new JMenuItem("New");
   newfile.addActionListener(this);
   newfile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, ActionEvent.CTRL_MASK));
   file.add(newfile);
   /* File->Open */
   open = new JMenuItem("Open...");
```

```
open.addActionListener(this);
open.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, ActionEvent.CTRL_MASK));
file.add(open);
/* File->Save */
save = new JMenuItem("Save");
save.addActionListener(this);
save.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, ActionEvent.CTRL_MASK));
file.add(save);
/* File->Save As */
saveas = new JMenuItem("Save As...");
saveas.addActionListener(this);
file.add(saveas);
/* File->Print */
print = new JMenuItem("Print");
print.addActionListener(this);
print.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P, ActionEvent.CTRL_MASK));
file.add(print);
/* File->Exit */
exit = new JMenuItem("Exit");
exit.addActionListener(this);
exit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F4, ActionEvent.CTRL_MASK));
file.add(exit);
/* Inserts separators at the 4th and 6th positions. */
file.insertSeparator(4);
file.insertSeparator(6);
/* Edit->Undo */
edit.add(undoAction);
/* Edit->Redo */
edit.add(redoAction);
/* Edit->Cut */
cut = new JMenuItem("Cut");
edit.add(cut);
cut.addActionListener(this);
cut.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK));
/* Edit->Copy */
copy = new JMenuItem("Copy");
edit.add(copy);
copy.addActionListener(this);
copy.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C, ActionEvent.CTRL_MASK));
/* Edit->Paste */
paste = new JMenuItem("Paste");
edit.add(paste);
paste.addActionListener(this);
paste.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_V, ActionEvent.CTRL_MASK));
/* Edit->Find */
find = new JMenuItem("Find");
edit.add(find);
find.addActionListener(this);
find.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F, ActionEvent.CTRL_MASK));
/* Edit->Select All */
selectAll = new JMenuItem("Select All");
edit.add(selectAll);
selectAll.addActionListener(this);
selectAll.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A, ActionEvent.CTRL_MASK));
/* Edit->Date/Time */
datetime = new JMenuItem("Date/Time");
edit.add(datetime);
datetime.addActionListener(this);
datetime.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F5, ActionEvent.CTRL_MASK));
/* Inserts separators at the 2nd, 6th, and 9th positions. */
edit.insertSeparator(2);
edit.insertSeparator(6);
edit.insertSeparator(9);
/* Format->Font */
font = new JMenuItem("Font");
format.add(font);
font.addActionListener(this);
/* Format->Color */
color = new JMenuItem("Color");
format.add(color);
color.addActionListener(this);
/* Format->Word Wrap */
wordWrap = new JCheckBoxMenuItem("Word Wrap");
format.add(wordWrap);
wordWrap.addActionListener(this);
/* Lock->Exclusive Lock */
exclusiveLock = new JCheckBoxMenuItem("Exclusive Lock");
exclusiveLock.setEnabled(false);
lock.add(exclusiveLock);
exclusiveLock.addActionListener(this);
/* Lock->Share Lock */
shareLock = new JCheckBoxMenuItem("Share Lock");
shareLock.setEnabled(false);
lock.add(shareLock);
shareLock.addActionListener(this);
/* Help->Help Topics */
about = new JMenuItem("Help Topics");
help.add(about);
```

```
about.addActionListener(this);
/* Initializes the panel. */
panel = new JPanel();
/* Sets the panel layout as BorderLayout. */
panel.setLayout(new BorderLayout());
/* Adds the panel to the frame. */
getContentPane().add(panel, BorderLayout.CENTER);
/* Initializes the object of text area */
area = new JTextArea(25, 65);
/*
Adds UndoableEditListener to the area for undo and redo operations.
*/
area.getDocument().addUndoableEditListener(this);
/* Sets line wrap to false. */
area.setLineWrap(false);
/* Sets word wrap style to false. */
area.setWrapStyleWord(false);
/*
Initializes the object of the Scrollpane class and adds the text area to the scroll pane.
*/
scrollpane = new JScrollPane(area);
/* Adds the scroll pane to the panel. */
panel.add(scrollpane);
/* Sets default close operation to false. */
setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
/*
addWindowListener - It contains a windowClosing() method.
windowClosing: It is called when the user clicks the cancel button of the Window.
It closes the main window.
Parameter: we- Represents an object of the WindowEvent class.
Return Value: NA
*/
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
       System.exit(0);
    }
 });
}
/*
undoableEditHappened() - This method is called when the user wants to perform undo/redo.
Parameters: e - Represents an object of UndoableEditEvent class that contains
the details of the undo/redo events.
Return Value: NA
*/
public void undoableEditHappened(UndoableEditEvent e)
{
   /*
   Inserts an Edit at indexOfNextAdd and removes any old edits that were at indexOfNextAdd or
   */
    undoManager.addEdit(e.getEdit());
   /* Calls the update() method to perform undo operations. */
    undoAction.update();
   /* Calls the update() method to perform redo operations. */
    redoAction.update();
}
/*
actionPerformed() - This method is called when the end user selects any menu item from the men
Parameters: ae - Represents an object of the ActionEvent class that contains the details of th
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   /*
   This section is executed when the end user selects the File->New command to the menu bar.
   */
    if(ae.getSource() == newfile)
    {
      /* Calls the newFile() method. */
       exclusiveLock.setEnabled(false);
       shareLock.setEnabled(false);
       action.newFile();
    }
   /*
   This section is executed when the end user selects the File->Open command to the menu bar.
   */
    else if(ae.getSource() == open)
    {
      /* Calls the openFile() method. */
       action.openFile();
    }

   /*
   This section is executed when the end user selects the File->Save command to the menu bar.
   */
    else if(ae.getSource() == save)
    {
```

```
   /* Calls the saveFile() method. */
    action.saveFile();
 }
/*
This section is executed when the end user selects the File->Save As command to the menu ba
*/
 else if(ae.getSource() == saveas)
 {
   /* Calls the saveAsFile() method. */
    action.saveAsFile();
 }
/*
This section is executed when the end user selects the File->Print command to the menu bar.
*/
 else if(ae.getSource() == print)
 {
   /* Calls the printFile() method. */
    act = 1;
    t1 = new Thread(this);
    t1.start();
 }
/*
This section is executed when the end user selects the Edit->Exit command to the menu bar.
*/
 else if(ae.getSource() == exit)
 {
   /* Calls the exitFile() method. */
    action.exitFile();
 }
/*
This section is executed when the end user selects the Edit->Cut command to the menu bar.
*/
 else if(ae.getSource() == cut)
 {
   /* Calls the cutFile() method. */
    action.cutFile();
 }
/*
This section is executed when the end user selects the Edit->Copy command to the menu bar.
*/
 else if(ae.getSource() == copy)
 {
   /* Calls the copyFile() method. */
    action.copyFile();
 }
/*
This section is executed when the end user selects the Edit->Paste command to the menu bar.
*/
 else if(ae.getSource() == paste)
 {
      /* Calls the pasteFile() method. */
       action.pasteFile();
 }
/*
This section is executed when the end user selects the Edit->Date/Time command to the menu
*/
 else if(ae.getSource() == datetime)
 {
   /* Calls the dateTime() method. */
    action.dateTime();
 }
/*
This section is executed when the end user selects the Edit->Find command to the menu bar.
*/
 else if(ae.getSource() == find)
 {
   /* Calls the findFile() method. */
    action.findFile();
 }
/*
This section is executed when the end user selects the Edit->Select All command to the menu
*/
 else if(ae.getSource() == selectAll)
 {
   /* Calls the selectAllFile() method. */
    action.selectAllFile();
 }
/*
This section is executed when the end user selects the Format->Font command to the menu bar
*/
 else if(ae.getSource() == font)
 {
   /* Calls the fontFile() method. */
    action.fontFile();
 }
/*
This section is executed when the end user selects the Format->Color command to the menu ba
```

```
      */
       else if(ae.getSource() == color)
       {
         /* Calls the colorFile() method. */
          action.colorFile();
       }
      /*
      This section is executed when the end user selects the Format->Word Wrap command to the men
      */
       else if(ae.getSource() == wordWrap)
       {
         /* Calls the warpFile() method. */
          action.wrapFile();
       }
      /*
      This section is executed when the end user selects the Lock->Exclusive command to the menu
      */
       else if(ae.getSource() == exclusiveLock)
       {
          act = 2;
          t2 = new Thread(this);
          t2.start();
       }
      /*
      This section is executed when the end user selects the Lock->Share command to the menu bar.
      */
       else if(ae.getSource() == shareLock)
       {
          act = 3;
          t3 = new Thread(this);
          t3.start();
       }
      /*
      This section is executed when the end user selects the Help->Help Topics command to the men
      */
       else if(ae.getSource() == about)
       {
         /* Calls the aboutFile() method. */
          action.aboutFile();
       }
    }
    public void run()
    {
       if(act==1)
       {
          action.printFile();
    }
    else if(act==2)
    {
      /* Calls the exLockFile() method. */
       action.exLockFile();
    }
    else if(act==3)
    {
      /* Calls the shLockFile() method. */
       action.shLockFile();
    }
 }
/*
main() - This method creates the main window of the user interface and displays it.
Parameters:
args[] - Contains any command line arguments passed.
Return Value: NA
*/
 public static void main(String args[])
 {
    try
    {
      /*
      Initializes and sets the look and feel of the application to Windows look and feel.
      */
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    }
    catch(Exception e)
    {
      System.out.println("Unknown Look and Feel." + e);
    }
   /* Creates an object of the Editor class. */
    Editor ed = new Editor();
   /* Sets the size of the main window. */
    ed.setSize(725, 460);
   /* Sets the visibility of the window to TRUE. */
    ed.setVisible(true);
   /* Displays the main window. */
    ed.show();
 }
/*
Class UndoAction - Extends the AbstractAction class to perform undo operations.
```

```
    Method:
    UndoAction() - Implements the default constructor of the UndoClass.
    actionPerformed() - Performs the action event.
    update() - Updates the change made by the end user. */
     class UndoAction extends AbstractAction
     {
       /* Defines the default constructor. */
        public UndoAction()
     {
        super("Undo");
        setEnabled(false);
     }
    /*
    actionPerformed() - This method is called when the end user select the Edit->Undo command from th
    Parameters: e - Represent an object of the ActionEvent class that contains details of the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent e)
     {
        try
        {
           undoManager.undo();
        }
        catch (Exception ex)
        {
          System.out.println("Error: " + ex);
        }
        update();
        redoAction.update();
     }
    /*
    update() - This method performs undo operations.
    Parameters: NA
    Return Value: NA
    */
     protected void update()
     {
        if(undoManager.canUndo())
        {
             setEnabled(true);
             putValue("Undo", undoManager.getUndoPresentationName());
        }
        else
        {
          setEnabled(false);
          putValue(Action.NAME, "Undo");
        }
     }
 }
/*
Class UndoAction - Extends the AbstractAction class to perform redo operations.
Method:
RedoAction() - Implements the default constructor of UndoClass.
actionPerformed() - Performs the action event.
update() - Updates the changes made by the end user.
*/
 class RedoAction extends AbstractAction
 {
   /* Defines the default constructor. */
    public RedoAction()
    {
       super("Redo");
       setEnabled(false);
    }
      /*
      actionPerformed() - This method is called when the end user selects the
      Edit->Redo command from the menu bar.
      Parameters: e - Represents an object of the ActionEvent class that contains
      the details of the event.
      Return Value: NA
      */
      public void actionPerformed(ActionEvent e)
      {
         try
         {
             undoManager.redo();
         }
          catch (Exception ex)
          {
            System.out.println("Error: " + ex);
          }
          update();
          undoAction.update();
       }
      /*
      update() - This method performs the redo operations.
      Parameters: NA
```

```
        Return Value: NA
        */
        protected void update()
        {
            if(undoManager.canRedo())
            {
                 setEnabled(true);
                 putValue("Redo", undoManager.getRedoPresentationName());
            }
            else
            {
               setEnabled(false);
              putValue(Action.NAME, "Redo");
            }
          }
      }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the Editor class. This class generates the main window of the Text Editor, as shown in Figure 6-2:

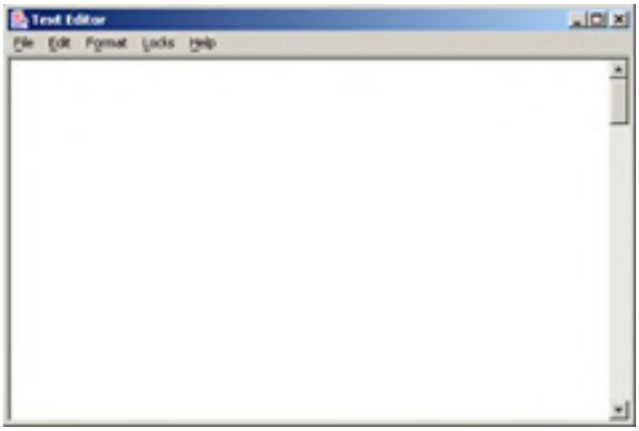


**Figure 6-2:** The Text Editor User Interface

When an end user clicks a menu item from the menu bar, the Text Editor invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate class or method, based on the selected menu item. The menu options in the Text Editor application are:

- File Menu: Contains the following menu items:
  - New: Calls a newFile() method to create a new text file.
  - Open: Calls an openFile() method to open an existing text file.
  - Save: Calls a saveFile() method to save an existing text file.
  - Save As: Calls a saveAsFile() method to save a newly created text file.
  - Print: Calls the printFile() method to display the Print dialog box, based on the value assigned to the act variable.
  - Exit: Calls an exitFile() method to close the Text Editor application.
- Edit Menu: Contains the following menu items:
  - Undo: Calls an undo() method to undo the last operation performed on the file.
  - Redo: Calls a redo() method to redo the last undo operation performed on the file.
  - Cut: Calls a cutFile() method to cut a selected text from the text file.
  - Copy: Calls a copyFile() method to copy a selected text from the text file.
  - Paste: Calls a pasteFile() method to paste the cut or copied text at a specific position.
  - Find: Calls a findFile() method to find a specific word in the file.
  - Select All: Calls a selectAllFile() method to select all the contents of the file.
  - Date/Time: Calls a dateTime() method to insert the current date and time in the text file.

- Format Menu: Contains the following menu items:
  - Font: Calls a fontFile() method to display the Font dialog box.
  - Color: Calls a colorFile() method to display the Color dialog box.
  - Word Wrap: Calls a wrapFile() method to wrap the text style in the text area.
- Locks Menu: Contains the following menu items:
  - Exclusive Lock: Calls the exLockFile() method based on the value assigned to the act variable. The exLockFile() method locks the file with an exclusive lock.
  - Share Lock: Calls the shLockFile() method based on the value assigned to the act variable. The shLockFile() method locks the file with the share lock.
- Help Menu: Contains one menu item, Help Topics. This menu item calls an aboutFile() method to display the Help dialog box.

The Editor.java file creates objects of the UndoManager, UndoAction, and RedoAction classes, which perform the undo and redo operations on the file. The UndoAction and RedoAction classes contain protected update() methods that enable or disable the Undo and Redo menu items of the Edit menu, respectively.



◄ PREVIOUS NEXT ►

# Implementing the Text Editor Application

The ActionPerform.java file implements the core functionality of the Text Editor application. This file defines the methods, such as newFile(), openFile(), saveFile(), printFile(), cutFile(), exLockFile(), or shLockFile() that are invoked when an end user clicks a specific menu item from the menu bar.

Listing 6-2 shows the contents of the ActionPerform.java file:

**Listing 6-2: The ActionPerform.java File**

```
/* Imports the java.util package classes. */
import java.util.Date;
import java.util.StringTokenizer;
/* Imports the java.io package classes. */
import java.io.File;
import java.io.RandomAccessFile;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
/* Imports the javax.swing package classes. */
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import java.awt.Font;
/* Imports the java.awt.event package classes. */
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
/* Imports the java.nio package classes. */
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.channels.FileLock;
/*
The class ActionPerform - This class implements the core functionality of the Editor.
This class provides the implementation of methods that are called in the Editor class.
Fields:
   value - Contains the return value of the showDialog() method.
   option - Contains 0 or 1 value.
   content - Contains the content of a file.
   name - Contains the file name.
   word - Contains the word that to be searched.
   str - Represents a string variable.
Methods:
   newFile() - This method opens a new file.
   openFile() - This method calls the method to open an existing file.
   saveFile() - This method calls the method to save the existing file.
   saveAsFile() - This method calls the method to save the new file.
   printFile() - This method enables you to print the file.
   exitFile() - This method closes the application.
   cutFile() - This method cuts the selected text.
   copyFile() - This method copies the selected text.
   pasteFile() - This method pastes the cut or copied text.
   dateTime() - This method inserts date and time in the file.
   exLockFile() - This method locks the file with exclusive lock.
   shLockFile() - This method locks the file with share lock.
   findFile() - This method finds a specified word in the file.
   wrapFile() - This method wraps the text line in the file.
   selectAllFile() - This method selects all the contents of the file.
   aboutFile() -This method opens the help file.
   colorFile() - This method opens a Color dialog box.
   fontFile() - This method opens a Font dialog box.
   open() - This method opens a new file.
   save() - This method saves the existing file.
   saveAs() - This method saves the newly created file.
*/
public class ActionPerform
{
   int value;
   int option;
   String content = null;
   String name = null;
   String word;
   String str;
   /* Creates an instance of the JFileChooser() class. */
   JFileChooser jfc = new JFileChooser(".");
   /* Creates an instance of the Editor class. */
   public Editor ed;
   /* Creates an object of the FontClass() method. */
   public FontClass font = new FontClass();
   /* Creates an object of the ColorClass() method. */
   public ColorClass cc = new ColorClass();
   /* Creates an object of the Help() method. */
   public Help h = new Help();
   /* Creates an object of the PrintClass() method. */
   public PrintClass pc;
```

```
/*
Defines default constructor of the ActionPerform class.
*/
public ActionPerform(Editor ed)
{
   this.ed = ed;
}
/* Implementation of the newFile() method. */
 public void newFile()
{
   if(!ed.area.getText().equals("") && !ed.area.getText().equals(content))
   {
      if(name == null)
      {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the saveAs() method. */
             saveAs();
            /* Sets the text area to be NULL. */
             ed.area.setText("");
          }

          if(option == 1)
          {
             ed.area.setText("");
          }
       }
       else
       {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the save() method. */
            save();
            ed.area.setText("");
          }
          if(option == 1)
          {
             ed.area.setText("");
          }
       }
    }
    else
    {
       ed.area.setText("");
    }
   /* Sets the title of the main window. */
    ed.setTitle("Untitled - Text Editor");
   }
   /* Implementation of openFile() method. */
   public void openFile()
   {
   if(!ed.area.getText().equals("") && !ed.area.getText().equals(content))
   {
      if(name == null)
      {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the saveAs() method. */
            saveAs();
            open();
         }
         if(option == 1)
         {
            open();
         }
       }
       else
       {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the save() method. */
            save();
            /* Calls the open() method. */
            open();
          }
          if(option == 1)
         {
            /* Calls the save() method. */
            open();
          }
```

```
        }
      }
      else
      {
        /* Calls the open() method. */
         open();
      }
   }
  /* Implementation of the saveFile() method. */
   public void saveFile()
   {
      if(name == null)
      {
        /* Calls the saveAs() method. */
         saveAs();
      }
      else
      {
        /* Calls the save() method. */
         save();
      }
   }
  /* Implementation of the saveAsFile() method. */
   public void saveAsFile()
   {
     /* Calls the saveAs() method. */
      saveAs();
   }
  /* Implementation of printFile() method. */
   public void printFile()
   {
        /*
        Creates an instance of the PrintClass that inputs the ed.area component and ed
        object of the Editor class as parameters.
        */
         pc = new PrintClass(ed.area, ed);
        /* Calls the print() method. */
         pc.print();
   }
  /* Implementation of the exitFile() method. */
   public void exitFile()
   {
     if(!ed.area.getText().equals("") && !ed.area.getText().equals(content))
      {
         if(name == null)
        {
           /* Shows a Confirm dialog box. */
           option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
           if(option == 0)
           {
              /* Calls the saveAs() method. */
              saveAs();
              /* Closes the main application. */
              System.exit(0);
           }
           if(option == 1)
           {
              System.exit(0);
           }
        }
        else
        {
           /* Shows a Confirm dialog box. */
           option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
           if(option == 0)
           {
              /* Calls the save() method. */
              save();
              System.exit(0);
           }
           if(option == 1)
           {
              System.exit(0);
           }
        }
     }
     else
     {
         System.exit(0);
     }
  }
  /* Implementation of the cutFile() method. */
   public void cutFile()
   {
     /* Cuts the selected area from the text area. */
      ed.area.cut();
   }
     /* Implementation of the copyFile() method. */
```

```
   public void copyFile()
   {
      /* Copies the selected area from the text area. */
      ed.area.copy();
   }
   /* Implementation of the pasteFile() method. */
   public void pasteFile()
   {
      /* Pastes the cut/copied area to the text area. */
       ed.area.paste();
   }
   /* Implementation of the selectAllFile() method. */
   public void selectAllFile()
   {
      /* Selects all the content of the text area. */
       ed.area.selectAll();
    }
   /* Implementation of the findFile() method. */
    public void findFile()
    {
       try
      {
         /* Shows a word input dialog box. */
         word = JOptionPane.showInputDialog("Type the word to find");
         while(ed.area.getText().indexOf(word) == -1)
         {
         /* Shows a message dialog box. */
         JOptionPane.showMessageDialog(null,"Word not found!","No
         match",JOptionPane.WARNING_MESSAGE);
         word = JOptionPane.showInputDialog(" Type the word to find");
    }
    /* Selects the word in the text area. */
    ed.area.select(ed.area.getText().indexOf(word),
    ed.area.getText().indexOf(word) + word.length());
    }
    catch(Exception ex)
    {
      /* Showss an error message dialog box. */
      JOptionPane.showMessageDialog
      (null,"Search canceled","Abourted",JOptionPane.WARNING_MESSAGE);
    }
 }
/* Implementation of the dateTime() method. */
 public void dateTime()
 {
   /* Creates an object of the Date class. */
    Date d = new Date();
   /* Converts time to string and display it. */
    String str = d.toLocaleString();
   /* Appends the date and time in the text area. */
    ed.area.append(str);
 }
/* Implementation of the fontFile() method. */
 public void fontFile()
 {
   /* Shows the Font dialog box. */
   font.setVisible(true);
   font.pack();
   /* Calls the getOk() method of FontClass. */
   font.getOk().addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         /*
         Sets the font name, size, and style of the file text.
         */
         ed.area.setFont(font.font());
         font.label.setFont(new Font("Arial",Font.PLAIN,15));
         font.setVisible(false);
      }
   });
   /* Calls the getCancel() method of FontClass. */
   font.getCancel().addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         /* Hides the Font dialog box. */
         font.label.setFont(new Font("Arial",Font.PLAIN,15));
         font.setVisible(false);
      }
   });
}
/* Implementation of the colorFile() method. */
public void colorFile()
{
   /* Shows the Color dialog box. */
   cc.setVisible(true);
```

```
      cc.pack();
      /* Calls the getOk() method of ColorClass. */
      cc.getOk().addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent ae)
         {
            /* Sets the color of the file text. */
            ed.area.setForeground(cc.color());
            cc.setVisible(false);
         }
      });
      /* Calls the getCancel() method of ColorClass. */
      cc.getCancel().addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent ae)
         {
            /* Hides the Color dialog box. */
            cc.setVisible(false);
         }
      });
   }
   /* Implementation of the wrapFile() method. */
   public void wrapFile()
   {
      if(ed.wordWrap.getState() == true)
      {
         /* Sets the line and word wrap style to TRUE. */
         ed.area.setLineWrap(true);
         ed.area.setWrapStyleWord(true);
      }
      else
      {
         /* Sets the line and word wrap style to FALSE. */
         ed.area.setLineWrap(false);
         ed.area.setWrapStyleWord(false);
      }
   }
   /* Implementation of the aboutFile() method. */
   public void aboutFile()
   {
      /* Shows the Help dialog box. */
      h.setVisible(true);
      /* Calls the getOk() method of Help. */
      h.getOk().addActionListener(new ActionListener()
      {
      public void actionPerformed(ActionEvent ae)
      {
         /* Hides the Help dialog box. */
         h.setVisible(false);
      }
   });
   }
   /* Implementation of exLockFile() method. */
   public void exLockFile()
   {
      try
      {
         /* Creates an object of the File class. */
         File file = new File(str);
         /* Creates an object of the RandomAccessFile class. */
         RandomAccessFile raf = new RandomAccessFile(file, "rw");
         /* Creates an object of the FileChannel class. */
         FileChannel channel = raf.getChannel();
         if(ed.exclusiveLock.getState() == true)
         {
            /*
            Uses the file channel to create an exclusive lock on the file.
            The lock() method blocks until it can retrieve the lock.
            */
            FileLock lock = channel.lock(0, Long.MAX_VALUE, false);
         }
         else if(ed.exclusiveLock.getState() == false)
         {
            /* Closes the channel. */
            channel.close();
         }
      }
      catch (Exception e)
      {
         System.out.println("Error: " + e);
      }
   }
   /* Implementation of the shLockFile() method. */
   public void shLockFile()
   {
      try
      {
         /* Creates an object of the File class. */
```

```
            File file = new File(str);
            /* Create an object of the RandomAccessFile class. */
            RandomAccessFile raf = new RandomAccessFile(file, "rw");
            /* Creates an object of the FileChannel class. */
            FileChannel channel = raf.getChannel();
            if(ed.shareLock.getState() == true)
            {
               /*
               Uses the file channel to create a share lock on the file.
               The lock() method blocks until it can retrieve the lock.
               */
               FileLock lock = channel.lock(0, Long.MAX_VALUE, true);
            }
            else if(ed.shareLock.getState() == false)
            {
               /* Closes the channel. */
               channel.close();
            }
         }
         catch (Exception e)
         {
            System.out.println("Error:" + e);
         }
      }
      /* Implementation of open() method. */
      public void open()
      {
         value = jfc.showOpenDialog(ed);
         if(value == JFileChooser.APPROVE_OPTION)
         {
            ed.area.setText(null);
            ed.exclusiveLock.setEnabled(true);
            ed.shareLock.setEnabled(true);
            try
            {
               /* Gets the file name. */
               name = jfc.getSelectedFile().getPath();
               /* Creates an object of FileInputStream class. */
               FileInputStream fin = new FileInputStream(jfc.getSelectedFile());
               /*
               Creates a channel and get the channel from the FileInputStream
               */
               FileChannel fchan = fin.getChannel();
               /* Stores the size of file in long variable */
               long fsize = fchan.size();
               /*
               Creates an object of the Bytebuffer class and allocate the size of this byte buffer
               */
               ByteBuffer buff = ByteBuffer.allocate((int)fsize);
               /* Reads the bytes from channel to byte buffer*/
               fchan.read(buff);
               /* Rewinds the byte buffer */
               buff.rewind();
               /*
               Returns the byte buffer into an array and converts this array into string
               */
               String str = new String(buff.array());
               /* Appends this string to the text area */
               ed.area.append(str);
               content = ed.area.getText();
               /* Closes the channel */
               fchan.close();
               /* Closes the input stream */
               fin.close();
            }
            catch(IOException ioe)
            {
               System.err.println("I/O Error on Open");
            }
            /* Sets the title of the main window */
            ed.setTitle(jfc.getSelectedFile().getAbsolutePath()+ " - Text Editor");
            str = jfc.getSelectedFile().getAbsolutePath();
         }
      }
      /* Implementation of save() method. */
      public void save()
      {
         if(value == JFileChooser.APPROVE_OPTION)
         {
            try
            {
               /* Creates an object of FileOutputStream class */
               FileOutputStream fout = new FileOutputStream(jfc.getSelectedFile());
               /* Creates an object of PrintWriter class */
               PrintWriter pw = new PrintWriter(fout);
               content = ed.area.getText();
               /* Creates an object of StringTokenizer class */
               StringTokenizer st=new StringTokenizer(content,System.getProperty("line.separator"));
```

```
            while(st.hasMoreTokens())
            {
               pw.println(st.nextToken());
            }
            /* Closes print writer */
            pw.close();
            /* Closes output stream */
            fout.close();
         }
         catch(IOException ioe)
         {
            System.err.println("I/O Error on Save");
         }
         /* Sets the title of the window */
         ed.setTitle(jfc.getSelectedFile().getName()+ " - Text Editor");
         str = jfc.getSelectedFile().getAbsolutePath();
      }
   }
   /* Implementation of saveAs() method. */
   public void saveAs()
   {
      jfc.setDialogTitle("Save As");
      value = jfc.showSaveDialog(ed);
      if(value == JFileChooser.APPROVE_OPTION)
      {
         try
         {
            /* Creates an object of FileOutputStream class */
            FileOutputStream fout = new FileOutputStream(jfc.getSelectedFile() + ".txt");
            /* Creates an object of PrintWriter class */
            PrintWriter pw = new PrintWriter(fout);
            content = ed.area.getText();
            name = jfc.getSelectedFile().getPath();
            /* Creates an object of StringTokenizer class */
            StringTokenizer st=new StringTokenizer(content,System.getProperty("line.separator"));
            while(st.hasMoreTokens())
            {
               pw.println(st.nextToken());
            }
            /* Closes print writer */
            pw.close();
            /* Closes output stream */
            fout.close();
         }
         catch(IOException ioe)
         {
            System.err.println ("I/O Error on Save");
         }
         /* Sets the title of the window */
         ed.setTitle(jfc.getSelectedFile().getAbsolutePath() + " - Text Editor");
         str = jfc.getSelectedFile().getAbsolutePath();
      }
   }
}
```

Download this Listing.

In the above code, the ActionPerform class creates instances of the FontClass, PrintClass, Help, PrintClass, Editor, and JFileChooser classes. The ActionPerform() constructor accepts the Editor class as an input parameter. The methods defined in the above listing are:

- `newFile()`: Checks whether or not the text area contains text. If the text area contains text, the newFile() method checks the file name using the name variable. If the name returns a null file name, the newFile() method calls the saveAs() method to save the file as a new one; otherwise the newFile() method calls the save() method to save the existing file.
- `openFile()`: Checks whether or not the text area contains text. If the text area contains text, the openFile() method checks the file name using the name variable. If the name returns a null file name, the openFile() method calls the saveAs() and open() methods to open a file; otherwise the openFile() method calls the save() and open() methods to open a file.
- `saveFile()`: Checks the file name using the name variable. If the name returns a null file name, the saveFile() method calls the saveAs() method to save the file as a new one; otherwise the saveFile() method calls the save() method to save the existing file.
- `saveAsFile()`: Calls the saveAs() method to save the file.
- `printFile()`: Creates an instance of the PrintClass class. The printFile() method calls the print() method to print the file.
- `exitFile()`: Checks whether or not the text area contains text. If the text area contains text, the exitFile() method checks the file name using the name variable. If the name returns a null file name, the exitFile() method calls the saveAs() and System.exit(0) method to close the file; otherwise the exitFile() method calls the save() and System.exit(0) methods to close the file.
- `cutFile()`: Calls the cut() method to cut the selected area of the text area.

- `copyFile()`: Calls the copy() method to copy the selected area of the text area.
- `pasteFile()`: Calls the paste() method to paste the copied or cut text at the specific position in the text area.
- `selectAllFile()`: Calls the selectAllFile() method to select all the contents of the text area.
- `dateTime()`: Creates an instance of the Date class and inserts the current date and time at the curser position in the text area.
- `findFile()`: Opens an Input dialog box to input the word that is to be searched in the file. If the findFile() method finds the specified word in the file, it selects the word in the text area; otherwise it opens a Message dialog box.
- `fontFile()`: Opens the Font dialog box and calls the getOk() method to set the font of the text that is displayed in the text area. The fontFile() method calls the getCancel() method to close the Font dialog box.
- `colorFile()`: Opens the Color dialog box and calls the getOk() method to set the color of the text that is displayed in the text area. The fontFile() method calls the getCancel() method to close the Color dialog box.
- `wordWrap()`: Checks the status of the Word Wrap check box. If the Word Wrap check box returns true, the wordWrap() method sets the line and word wrap styles to true; otherwise it sets the line and word wrap styles to false.
- `exLockFile()`: Creates an instance of the File class. The exLockFile() method creates an instance of the RandomAccessFile class and calls the getChannel() method of the FileChannel class. The exLockFile() method then checks the status of the Exclusive Lock check box. If the Exclusive Lock check box returns true, the exLockFile() method locks the currently opened file with an exclusive lock.
- `shLockFile()`: Creates an instance of the File class. The exLockFile() creates an instance of RandomAccessFile class and calls the getChannel() method of the FileChannel class. The shLockFile() method then checks the status of the Share Lock check box. If the Share Lock check box returns true, the shLockFile() method locks the currently opened file with a share lock.
- `aboutFile()`: Opens a Help dialog box and calls the getOk) method to close the Help dialog box.
- `open()`: Displays an Open dialog box that is to open a file from a specific location. The open() method creates an instance of the FileInputStream class and calls the getChannel() method of the FileChannel class. The open() method then stores the size of the file in a variable of long type, creates an object of the ByteBuffer class, and allocates the size to the byte buffer. Next, the open() method reads the bytes from the channel to the byte buffer and rewinds the byte buffer. Finally, the open() method returns the byte buffer into an array, converts this array into a string, and appends this string to the text area.
- `save()`: Creates an instance of the FileOutputStream class. The save() method creates an instance of the PrintWriter class and an object of the StringTokenizer class. This method then writes the file at a specified location using the PrintWriter and StringTokenizer class instances.
- `saveAs()`: Displays a Save As dialog box that is to save a new file at a specific location using save() method.



PREVIOUS NEXT

# Creating the Print Dialog Box

You can use the PrintClass.java file to open the default Print dialog box for the Text Editor application. In this dialog box, you can specify the print range, number of copies, and page property before you print the file.

Listing 6-3 shows the contents of the PrintClass.java file:

**Listing 6-3: The PrintClass.java File**

```
/* Imports java.awt.print package class. */
import java.awt.print.PrinterJob;
import java.awt.print.PrinterException;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
/* Imports java.awt package classes. */
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Graphics2D;
/* Imports javax.swing package class. */
import javax.swing.RepaintManager;
/*
class PrintClass - This class implements Printable interface and enable the end user to print the do
Fields:
   compPrint - Contains the document that to be printed.
Methods:
   print() - This method prints the current document opened in the text editor window
*/
public class PrintClass implements Printable
{
   /* Declares the object of Component class. */
   private Component compPrint;
   public Editor editor;
   /* Defines the default constructor. */
   public PrintClass(Component compPrint, Editor editor)
   {
      this.compPrint = compPrint;
      this.editor = editor;
   }
   /*
   print() - This method prints the document
   Parameters: lse - NA
   Return Value: NA
   */
   public void print()
   {
      /* Creates an object of the PrinterJob class */
      PrinterJob printer = PrinterJob.getPrinterJob();
      /* Sets the object of PrinterJob class to printable */
      printer.setPrintable(this);
      if(printer.printDialog())
      {
         try
         {
            /* Prints the document */
            printer.print();
         }
         catch(PrinterException pe)
         {
            System.out.println("Error: " + pe);
         }
      }
      editor.show();
   }
   /*
   print() - This method opens a Print dialog box
   Parameters:
   g - Represents the object of Graphics class
   format - Represents the object of PageFormat class
   index - Represents an index of page
   Return Value: int PAGE_EXIST
   */
   public int print(Graphics g, PageFormat format, int index)
   {
      if(index > 0)
      {
         return(NO_SUCH_PAGE);
      }
      else
      {
         /*
         Creates an object of the Graphics2D class and converts simple graphics to 2D graphics
         */
         Graphics2D g2d = (Graphics2D)g;
         /*
```

```
            Translates the origin of the Graphics2D context to the point (x, y)
            in the current coordinate system
            */
            g2d.translate(format.getImageableX(), format.getImageableY());
            /* Creates the object of the RepaintManager class */
            RepaintManager manager1 = RepaintManager.currentManager(compPrint);
            /* Sets the double buffer to FALSE */
            manager1.setDoubleBufferingEnabled(false);
            /* Paints the component */
            compPrint.paint(g2d);
            /* Creates the object of RepaintManager class */
            RepaintManager manager2 = RepaintManager.currentManager(compPrint);
            /* Sets the double buffer to TRUE */
            manager2.setDoubleBufferingEnabled(true);
            /* Returns the PAGE_EXIST value */
            return(PAGE_EXISTS);
        }
    }
}
```

Download this Listing.

In the above code, the PrintClass() constructor of the PrintClass takes two arguments, compPrint and editor. The compPrint argument is an object of the Component class that contains the printing content. The editor argument is an object of the Editor class.

The print() method creates an object of the PrinterJob class. This method then sets the printable property on the object of the PrinterClass to open the Print dialog box, as shown in Figure 6-3:

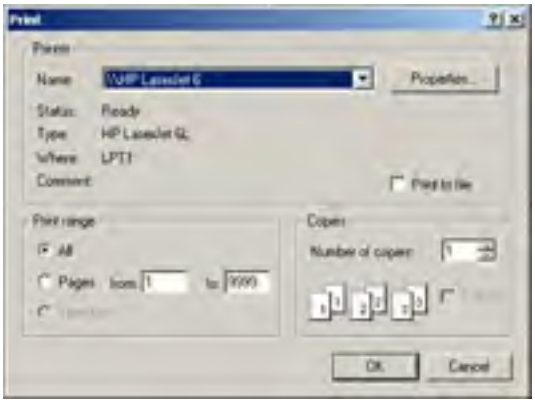


**Figure 6-3:** The Print Dialog Box

The Printer panel in the Print dialog box displays the status of the printer selected by end users in the Name combo box. The Properties button in this panel helps set the layout and quality of the page. In the Print range panel, end users can select a radio button to specify the print range. The Copies panel provides a list box that allows end users to specify the number of copies of a document they want to print.

When end users click the OK button in the Print dialog box, the print() method of PrinterJob class is invoked. This method prints the file.

If end users click the Cancel button of the Print Dialog box, the editor.show() method is invoked. This method closes the Print dialog box and sets the focus to the main window of the Text Editor application.

If end users click the Properties button of the Help dialog box, the print(Graphics g, PageFormat format, int index) method is invoked. This method uses three objects as parameters: g object of the Graphics class, format object of the PageFormat class, and index object of the Integer class. The print() method creates the object of the Graphics2D class. The object of the Graphics2D class translates the origin of the Graphics2D context to the point x, y in the current coordinate system. The print() method then creates the object of the RepaintManager class and sets the double buffer property to false. Next, the print() method paints the component using the paint() method. The print() method again creates the object of the RepaintManager class and sets the double buffer property to TRUE. Finally, the print() method returns the PAGE_EXISTS variable.

# Creating the Font Dialog Box

The FontClass.java file allows you to create the Font dialog box for the Text Editor application. End users can use this dialog box to specify the font type, size, and style to change the font of the text file.

Listing 6-4 shows the contents of the FontClass.java file:

**Listing 6-4: The FontClass.java File**

```
/* Imports javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
/* Imports java.awt package classes. */
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
import java.awt.Font;
/* Imports javax.swing.event package classes. */
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
/*
class FontClass - This class creates a Font dialog box that enables the
end user to change the font, size, and type of the text.
Fields:
   fontLabel - Contains the content of Font label.
   sizeLabel - Contains the content of Size label.
   typeLabel - Contains the content of Type label.
   previewLabel - Contains the content of preview.
   label - Contains the preview contents.
   fontText - Contains the selected font name.
   typeText - Contains the selected font type name.
   sizeText - Contains the selected font size name.
   fontScroll - Contains the Font list.
   typeScroll - Contains the Type list.
   sizeScroll - Contains the Size list.
   fontList - Contains all the available font.
   typeList - Contains all the available types of font style.
   sizeList - Contains all the available font size.
   ok - Creates an OK button.
   cancel - Creates a cancel button.
Methods:
   getOK() - This method returns the OK button object
   getCancel() - This method returns the Cancel button object
   valueChanged() - This method is invoked when an end user select the item from the List box.
   font() - This method returns the font
*/
public class FontClass extends JDialog implements ListSelectionListener
{
   /* Declares the objects of the JPanel class. */
   JPanel pan1;
   JPanel pan2;
   JPanel pan3;
   /* Declares the objects of the JLabel class. */
   JLabel fontLabel;
   JLabel sizeLabel;
   JLabel typeLabel;
   JLabel previewLabel;
   /* Declares the objects of the JTextField class. */
   JTextField label;
   JTextField fontText;
   JTextField sizeText;
   JTextField typeText;
   /* Declares the objects of the JScrollPane class. */
   JScrollPane fontScroll;
   JScrollPane typeScroll;
   JScrollPane sizeScroll;
   /* Declares the objects of the JList class. */
   JList fontList;
   JList sizeList;
   JList typeList;
   /* Declares the objects of the JButton class. */
   JButton ok;
   JButton cancel;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   /* Defines the default constructor. */
   public FontClass()
```

```
{
   /* Sets the title of the Font dialog. */
   setTitle("Font Dialog");
   /* Sets the size of Font dialog. */
   setSize(300, 400);
   /* Sets resizable button to false. */
   setResizable(false);
   /* Initializes the object of the GridBagLayout class. */
   gbl = new GridBagLayout();
   /* Sets the Layout. */
   getContentPane().setLayout(gbl);
   /* Creates an object of GridBagConstraints class. */
   gbc = new GridBagConstraints();
   /*
   Initializes the Font label object and add it to the 1, 1, 1, 1 positions with WEST alignment.
   */
   gbc.gridx = 1;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontLabel = new JLabel("Fonts: ");
   getContentPane().add(fontLabel, gbc);
   /*
   Initializes the Size label object and adds it to the 2, 1, 1, 1 positions with WEST alignment.
   */
   gbc.gridx = 2;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeLabel = new JLabel("Sizes: ");
   getContentPane().add(sizeLabel, gbc);
   /*
   Initializes the Types label object and adds it to the 3, 1, 1, 1 positions with WEST alignment
   */
   gbc.gridx = 3;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeLabel = new JLabel("Types: ");
   getContentPane().add(typeLabel, gbc);
   /*
   Initializes the Font text field object and adds it to the 1, 2, 1, 1 positions with WEST align
   */
   gbc.gridx = 1;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontText = new JTextField("Arial", 12);
   getContentPane().add(fontText, gbc);
   /*
   Initializes the Size text field object and adds it to the 2, 2, 1, 1 positions with WEST align
   */
   gbc.gridx = 2;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeText = new JTextField("8", 4);
   getContentPane().add(sizeText, gbc);
   /*
   Initializes the Types text field object and adds it to the 3, 2, 1, 1 positions with WEST alig
   */
   gbc.gridx = 3;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeText = new JTextField("Regular", 6);
   getContentPane().add(typeText, gbc);
   /*
   Initializes the Font list object and add it to the Font scroll pane object.
   Adds this scroll pane object to 1, 3, 1, 1 positions with WEST alignment
   */
   gbc.gridx = 1;
   gbc.gridy = 3;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames
   fontList = new JList(fonts);
   fontList.setFixedCellWidth(110);
   fontList.addListSelectionListener(this);
   fontList.setSelectedIndex(0);
   fontScroll = new JScrollPane(fontList);
```

```
        getContentPane().add(fontScroll, gbc);
        /*
        Initializes the Size list object and add it to the Size scroll pane object.
        Adds this scroll pane object to 2, 3, 1, 1 positions with WEST alignment
        */
        gbc.gridx = 2;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        String[] sizes = {"8","10","12","14","16","18","20","24","28","32","48","72"};
        sizeList = new JList(sizes);
        sizeList.setSelectedIndex(0);
        sizeList.setFixedCellWidth(20);
        sizeList.addListSelectionListener(this);
        sizeScroll = new JScrollPane(sizeList);
        getContentPane().add(sizeScroll, gbc);
        /*
        Initializes the Types list object and adds it to the Types scroll pane object.
        Next, adds this scroll pane object to 3, 3, 1, 1 positions with WEST alignment
        */
        gbc.gridx = 3;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        String[] types = {"Regular", "Bold", "Italic", "Bold Italic"};
        typeList = new JList(types);
        typeList.setFixedCellWidth(60);
        typeList.addListSelectionListener(this);
        typeList.setSelectedIndex(0);
        typeScroll = new JScrollPane(typeList);
        getContentPane().add(typeScroll, gbc);
        /*
        Initializes the preview label and adds it to 1,4,3,1 positions with CENTER alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 4;
        gbc.gridwidth = 3;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        pan1 = new JPanel();
        pan1.setLayout(new FlowLayout());
        previewLabel = new JLabel("Preview:");
        pan1.add(previewLabel);
        getContentPane().add(pan1, gbc);
        /*
        Initializes the preview text field and adds it to 1,5,3,1 positions with CENTER alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 5;
        gbc.gridwidth = 3;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        pan2 = new JPanel();
        pan2.setLayout(new FlowLayout());
        label = new JTextField("AaBaCcDdeEfFgGhHjJ", 15);
        label.setEditable(false);
        label.setBorder(BorderFactory.createEtchedBorder());
        label.setFont(new Font("Arial",Font.PLAIN,20));
        pan2.add(label);
        getContentPane().add(pan2, gbc);
        /*
        Initializes the OK and Cancel button and adds these two buttons to the panel.
        Sets layout of the panel to FlowLayout. Now add this panel to the 1, 6, 4, 1
        positions with CENTER alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 6;
        gbc.gridwidth = 3;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        pan3 = new JPanel();
        pan3.setLayout(new FlowLayout());
        ok = new JButton("OK");
        cancel = new JButton("Cancel");
        pan3.add(ok);
        pan3.add(cancel);
        getContentPane().add(pan3, gbc);
    }
    /*
    valueChanged() - This method is called when the user selects any menu item from the menu bar.
    Parameters: lse - a ListSelectionEvent object containing details of the event.
    Return Value: NA
    */
    public void valueChanged(ListSelectionEvent lse)
```

```
    {
       try
       {
          /*
          This section is executed, when end user selects the item from Font list.
          */
          if(lse.getSource() == fontList)
          {
             Font f1 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
             Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
             fontText.setText(String.valueOf(fontList.getSelectedValue()));
             label.setFont(f1);
          }
          /*
          This section is executed, when end user selects the item from Size list.
          */
          else if(lse.getSource() == sizeList)
          {
             Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
             Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
             sizeText.setText(String.valueOf(sizeList.getSelectedValue()));
             label.setFont(f2);
          }
          /*
          This section is executed, when end user selects the item from Type list.
          */
          else if(lse.getSource() == typeList)
          {
             Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
             Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
             typeText.setText(String.valueOf(typeList.getSelectedValue()));
             label.setFont(f2);
          }
       }
       catch(Exception nfe){}
    }
    /*
    font() - This method is set the font of the file text.
    Parameters: NA
    Return Value: font
    */
    public Font font()
    {
       /* Creates an object of the Font class. */
       Font font = new Font(String.valueOf(fontList.getSelectedValue()), typeList.getSelectedIndex(),
       Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
       /* Returns the font object */
       return font;
    }
    /*
    getOk() method - This method is invoked when end user click the OK button of the Font dialog box
    parameter - NA
    return value - ok
    */
    public JButton getOk()
    {
       return ok;
    }
    /*
    getCancel() method - This method is invoked when end user click the Cancel button of the Font dia
    parameter - NA
    return value - cancel
    */
    public JButton getCancel()
    {
       return cancel;
    }
}
```

Download this Listing.

In the above code, the FontClass() constructor creates the Font dialog box, as shown in Figure 6-4:


**Figure 6-4:** The Font Dialog Box

When an end user selects any item from the list box of the Font dialog box, the Font class invokes the valueChanged() method. This method acts as an event listener and activates the appropriate method to set the font of the preview label.

The FontClass defines two methods, getOK() and getCancel(), to perform the OK and Cancel operations. These methods return the object of the Button class.

When an end user clicks the OK button in the Font dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Font dialog box to false and changes the font of the file text, based on the selected font type, size, and style.

If an end user clicks the Cancel button in the Font dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Font dialog box to false and closes the Font dialog box.



◄ PREVIOUS NEXT ►

# Creating the Color Dialog Box

You can create the Color dialog box for the Text Editor application using the ColorClass.java file. This dialog box displays a set of labels, sliders, and buttons using which you can specify the RGB values to change the color of the text file.

Listing 6-5 shows the contents of the ColorClass.java file:

**Listing 6-5: The ColorClass.java File**

```
/* Imports javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JSlider;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
/* Imports java.awt package classes. */
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.Color;
/* Imports javax.swing.event package classes. */
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
/*
class ColorClass - This class creates a Color dialog box that enables the end user to
change the color of the file text.
Fields:
   panel - Contains all the components of the Color dialog.
   redLabel - Contains the content of Red label.
   greenLabel - Contains the content of Green label.
   blueLabel - Contains the content of Blue label.
   previewLabel - Contains the content of Preview label.
   redSlider - Enables the end user to select the red value.
   greenSlider - Enables the end user to select the green value.
   blueSlider - Enables the end user to select the blue value.
   labelText - Contains the content of preview text.
   ok - Creates an OK button.
   cancel - Creates a cancel button.
   r - Stores the Red value.
   g - Stores the Green value.
   b - Stores the Blue value.
Methods:
   getOK() - This method returns the OK button object.
   getCancel() - This method returns the Cancel button object.
   stateChanged() - This method is invoked when an end user slide slider.
   color() - This method returns the color.
   */
public class ColorClass extends JDialog implements ChangeListener
{
   /* Declares the object of the JPanel class. */
   JPanel panel;
   /* Declares the objects of the JLabel class. */
   JLabel redLabel;
   JLabel greenLabel;
   JLabel blueLabel;
   JLabel previewLabel;
   /* Declares the objects of the JSlider class */
   JSlider redSlider;
   JSlider greenSlider;
   JSlider blueSlider;
   /* Declares the object of the JTextField class */
   JTextField labelText;
   /* Declares the objects of the JButton class */
   JButton ok;
   JButton cancel;
   /* Declares the integer for storing the RGB values */
   int r = 0;
   int g = 0;
   int b = 0;
   public Editor ed;
   /* Defines the default constructor of the ColorClass class. */
   public ColorClass()
   {
      /* Sets the title of the Font dialog. */
      setTitle("Color Dialog");
      /* Sets resizable button to false. */
      setResizable(false);
      /* Initializes the object of the JPanel class. */
      panel = new JPanel();
      /* Sets the Layout as GridLayout.*/
      panel.setLayout(new GridLayout(5,2,1,1));
      /* Adds the panel to Color dialog frame */
      getContentPane().add(panel);
```

```
        /* Initializes and adds Red label to the panel. */
        redLabel = new JLabel("Red: ");
        panel.add(redLabel);
        /* Initializes and adds Red slider to the panel. */
        redSlider = new JSlider(0, 255, 1);
        panel.add(redSlider);
        /* Sets Border to the Red slider. */
        redSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Adds state change listener to Red slider. */
        redSlider.addChangeListener(this);
        /* Initializes and adds Green label to the panel. */
        greenLabel = new JLabel("Green: ");
        panel.add(greenLabel);
        /* Initializes and adds Green slider to the panel. */
        greenSlider = new JSlider(0, 255, 1);
        panel.add(greenSlider);
        /* Sets Border to the Green slider. */
        greenSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Adds state change listener to Green slider. */
        greenSlider.addChangeListener(this);
        /* Initializes and adds Blue label to the panel. */
        blueLabel = new JLabel("Blue: ");
        panel.add(blueLabel);
        /* Initializes and adds Blue slider to the panel. */
        blueSlider = new JSlider(0, 255, 1);
        panel.add(blueSlider);
        /* Sets Border to the Blue slider. */
        blueSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Adds state change listener to Blue slider. */
        blueSlider.addChangeListener(this);
        /* Initializes and add Preview label to the panel. */
        previewLabel = new JLabel("Preview: ");
        panel.add(previewLabel);
        /*
        Initializes and adds Preview text field to the panel.
        */
        labelText = new JTextField(10);
        panel.add(labelText);
        /* Initializes and adds OK button to the panel. */
        ok = new JButton("OK");
        panel.add(ok);
        /* Initializes and adds Cancel button to the panel. */
        cancel = new JButton("Cancel");
        panel.add(cancel);
    }
    /*
    stateChanged() - This method is called when the user slides any slider of the Color dialog box.
    Parameters: ce - Represents an object of the ChangeEvent class that contains the details of the e
    Return Value: NA
    */
    public void stateChanged(ChangeEvent ce)
    {
        /*
        This section is executed when end user slides the Red slider.
        */
        if(ce.getSource() == redSlider)
        {
            r = redSlider.getValue();
        }
        /*
        This section is executed, when end user slides the Green slider.
        */
        else if(ce.getSource() == greenSlider)
        {
            g = greenSlider.getValue();
        }
        /*
        This section is executed, when end user slides the Blue slider.
        */
        else if(ce.getSource() == blueSlider)
        {
            b = blueSlider.getValue();
        }
        /* Creates the object of the Color class. */
        Color c = new Color(r, g, b);
        /*
        Sets the background color of the preview text field.
        */
        labelText.setBackground(c);
    }
    /*
    color() - This method is set color of the file text.
    Parameters: NA
    Return Value: color
    */
    public Color color()
    {
        Color color = new Color(redSlider.getValue(), greenSlider.getValue(), blueSlider.getValue());
```

```
        return color;
    }
    /*
    getOk() method - This method is invoked when end user click the OK button of the Color dialog box
    parameter - NA
    return value - ok
    */
    public JButton getOk()
    {
        return ok;
    }
    /*
    getCancel() method - This method is invoked when end user click the Cancel
    button of the Color dialog box.
    parameter - NA
    return value - cancel
    */
    public JButton getCancel()
    {
        return cancel;
    }
}
```

Download this Listing.

In the above code, the ColorClass() constructor creates the Color dialog box, as shown in Figure 6-5:


**Figure 6-5:** The Color Dialog Box

When the end user slides any slider in the Color dialog box, the ColorClass class invokes the stateChanged() method. This method acts as an event listener and activates the appropriate method to set the color of the preview label.

The ColorClass class defines two methods, getOk() and getCancel(), to perform the OK and Cancel operations. These methods return the object of the Button class.

When the end user clicks the OK button on the Color dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Color dialog box to false and changes the color of the text file based on the selected RGB values.

If the end user clicks the Cancel button on the Color dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Color dialog box to false and closes the Color dialog box.



◄ PREVIOUS NEXT ►

# Creating the Help File

You can to create the Help dialog box of the Text Editor application using the Help.java file. This dialog box displays a list box and a text area. When you select a help topic name from the list box, information covered under that topic is displayed in the text area.

Listing 6-6 shows the contents of the Help.java file:

**Listing 6-6: The Help.java File**

```
/* Imports javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JList;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
/* Imports java.awt package classes. */
import java.awt.FlowLayout;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
/* Imports javax.swing.event package classes. */
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
/*
class Help - This class creates an Help dialog that enables the end user to read the user manual of
Fields:
   panel - Contains the document that to be printed.
   title - Contains the content of Title label.
   label1 - Contains the content of Help Topic label.
   label2 - Contains the content of Topic Details label.
   area - Displays the detail of sleeted topic.
   ok - Creates an OK button.
Methods:
valueChanged() - This method is invoked when an end user select the item from the List box.
*/
public class Help extends JDialog implements ListSelectionListener
{
   /* Declares the object of the JPanel class. */
   JPanel panel;
   /* Declares the objects of the JLabel class. */
   JLabel title;
   JLabel label1;
   JLabel label2;
   /* Declares the objects of the JScrollPane class. */
   JScrollPane listScroll;
   JScrollPane areaScroll;
   /* Declares the object of the JList class. */
   JList list;
   /* Declares the object of the JTextArea class. */
   JTextArea area;
   /* Declares the object of the JButton class. */
   JButton ok;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   String str = null;
   /* Defines the default constructor. */
   public Help()
   {
      /* Sets the title of the Help dialog. */
      setTitle("Help");
      /* Sets the size of the Help dialog. */
      setSize(600, 300);
      /* Sets resizable button to false. */
      setResizable(false);
      /* Initializes the object of the GridBagLayout class. */
      gbl = new GridBagLayout();
      /* Sets the Layout. */
      getContentPane().setLayout(gbl);
      /* Creates an object of GridBagConstraints class. */
      gbc = new GridBagConstraints();
      /*
      Initializes the Help Topic label object and adds it to the 1, 1, 1, 1 positions with WEST alig
      */
      gbc.gridx = 1;
      gbc.gridy = 1;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      label1 = new JLabel("Help Topics:");
      getContentPane().add(label1, gbc);
      /*
      Initializes the Topic Details label object and adds it to the 2, 1, 1, 1 positions with WEST a
```

```
      */
      gbc.gridx = 2;
      gbc.gridy = 1;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      label2 = new JLabel("Topic Details:");
      getContentPane().add(label2, gbc);
      /*
      Initializes the Help Topic List object and adds it to the Help Topic scroll pane object.
      Next, adds this scroll pane object to 1, 2, 1, 1 positions with WEST alignment.
      */
      gbc.gridx = 1;
      gbc.gridy = 2;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      String[] listString = {"Change the text color",
      "Cut, Copy and Paste the text",
      "Find a word in the file",
      "Insert date and time in the file",
      "Open a file",
      "Print a file",
      "Save a file",
      "Change the text font",
      "Using file locks",
      "Wrap the file text"};
      list = new JList(listString);
      /* Sets the height of the list box. */
      list.setFixedCellHeight(25);
      list.addListSelectionListener(this);
      listScroll = new JScrollPane(list);
      getContentPane().add(listScroll, gbc);
      /*
      Initializes the text area object and adds it to the text area scroll pane object.
      Next, adds this scroll pane objects to 2, 2, 1, 1 positions with WEST alignment.
      */
      gbc.gridx = 2;
      gbc.gridy = 2;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      area = new JTextArea(12, 60);
      /*
      Sets the editable property of the text area to false.
      */
      area.setEditable(false);
      /* Sets the Line and word wrap style to TRUE */
      area.setLineWrap(true);
      area.setWrapStyleWord(true);
      areaScroll = new JScrollPane(area);
      getContentPane().add(areaScroll, gbc);
      /*
      Initializes the OK button and adds this OK buttons to the panel. Next, sets the layout
      of the panel to FlowLayout. Now, adds this panel to the 1, 3, 2, 1 positions with CENTER align
      */
      gbc.gridx = 1;
      gbc.gridy = 3;
      gbc.gridwidth = 2;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.CENTER;
      panel = new JPanel();
      panel.setLayout(new FlowLayout());
      ok = new JButton("OK");
      panel.add(ok);
      getContentPane().add(panel, gbc);
   }
   /*
   valueChanged() - This method is called when the user selects any help topic from the list
   Parameters: lse - Represents an object of the ListSelectionEvent class that contains the details
   Return Value: NA
   */
   public void valueChanged(ListSelectionEvent lse)
   {
      try
      {
         /*
         This section is executed, when end user selects the item from Help Topic list.
         */
         if(lse.getSource() == list)
         {
            /*
            When end user selects "Change the text color" from the List.
            */
            if(list.getSelectedIndex() == 0)
            {
               str = "The steps to change the font color are:\n\n" + " 1.
               Select Format->Color command from the menu bar.\n\n" + " 2.
```

```
   Scroll the Red slider, Green slider, and Blue sliders to specify the
   R, G, B values for the color.\n\n" + " 3. Click OK button.";
}
/*
When end user selects "Cut, Copy, and Paste the text" from the List.
*/
else if(list.getSelectedIndex() == 1)
{
   str = "The steps to cut, copy, and paste the text are:\n\n" + " 1.
   Select the Edit->Cut command from the menu bar to cut the selected text.
   \n\n" + " 2. Select the Edit->Copy command from the menu bar to copy
   the selected text.\n\n" + " 3. Select the Edit-Paste command from the menu
   bar to paste the cut or copied text to the specified location.";
}
/*
When end user selects "Find a word in the file" from the List.
*/
else if(list.getSelectedIndex() == 2)
{
   str = "The steps to find a specific character or word are:\n\n" + " 1.
   Select the Edit->Find command from the menu bar.\n\n" + " 2.
   Specify the character or word in the Type the Word to find text box.\n\n" +
   " 3. Click the OK button.\n\n" + "
   4. Select the Edit->Find Next command from the menu bar to find the next word.";
}
/*
When end user selects "Insert date and time in the file" from the List.
*/
else if(list.getSelectedIndex() == 3)
{
   str = "The step to insert date and time in the document is:\n\n" + " 1.
   Select Edit->Date/Time command from the menu bar to insert the
   current date and time in the file.";
}
/*
When end user selects "Open a file" from the List.
*/
else if(list.getSelectedIndex() == 4)
{
   str = "The steps to open a file are:\n\n" + " 1. Select File->
   Open command from the menu bar.\n\n" + " 2. Browse the File that
   you want to open.\n\n" + " 3. Select the appropriate file in
   File name text box.\n\n" + " 4. Click OK button.";
}
/*
When end user selects "Print a file" from the List.
*/
else if(list.getSelectedIndex() == 5)
{
   str = "The steps to print the document are:\n\n" + " 1. Select File->
   Print command from the menu bar.\n\n" + " 2. Specify the print range.\n\n" + "
   3. Specify the copies in Number of copies text box.\n\n" + " 4. Click OK button.";
}
/*
When end user selects "Save a file" from the List.
*/
else if(list.getSelectedIndex() == 6)
{
   str = "The steps to save a new document are:\n\n" + " 1. Select File->
   Save As command from the menu bar.\n\n" + " 2. Browse the location where
   you want to save the file.\n\n" + " 3. Specify the name of the file in
   File name text box.\n\n" + " 4. Click OK button.\n\n" + "
   5. Select File->Save command from the menu bar to save the existing file.";
}
/*
When end user selects "Change the text font" from the List.
*/
else if(list.getSelectedIndex() == 7)
{
   str = "The steps to change the font are:\n\n" + " 1. Select the Format->
   Font command from the menu bar.\n\n" + " 2. Select the font from the
   Fonts list box.\n\n" + " 3. Select the font size from the Size list box.\n\n" + "
   4. Select the font style from the Style list box.";
}
/*
When end user selects "Using file locks" from the List.
*/
else if(list.getSelectedIndex() == 8)
{
   str = "The steps to lock the files are:\n\n" + " 1. Select Lock->
   Exclusive command from the menu bar to lock the file with exclusive lock.\n\n" + "
   2. Select Lock->Share command from the menu bar to lock the file with share lock.";
}
/*
When end user selects "Wrap the file text" from the List.
*/
else if(list.getSelectedIndex() == 9)
{
```

```
                str = "The step to wrap a text in the window is:\n\n" + "
                1. Select Format->Word Wrap command from the menu bar.";
             }
             /* Sets the text to the text area. */
             area.setText(str);
             }
          }
          catch(Exception nfe)
          {
       }
    }
    /*
    getOk() method - This method is invoked when end user click the OK button of the Font dialog box.
    parameter - NA
    return value - ok
    */
    public JButton getOk()
    {
       return ok;
    }
}
```

Download this Listing.

In the above code, the Help() constructor creates the Help dialog box, as shown in Figure 6-6:

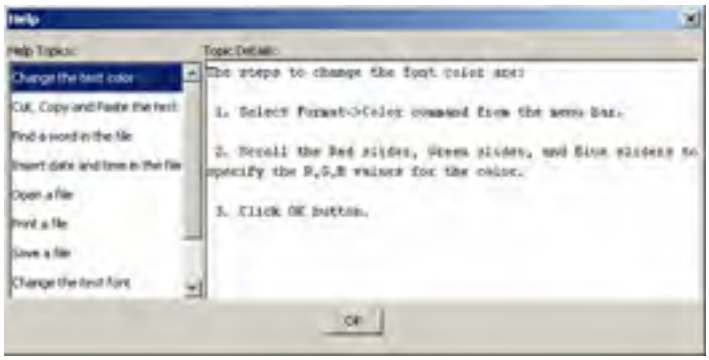


**Figure 6-6:** The Help Dialog Box

When an end user selects any help topic from the list box in the Help dialog box, the Help class invokes the valueChanged() method. This method acts as an event listener and displays the appropriate steps in the text area, based on the list item the end user selects.

When an end user clicks the OK button in the Help dialog box, the actionPerformed() method is invoked. This method sets the visibility of the Help dialog box to false and closes the dialog box.

# Unit Testing

To test the Text Editor:

1. 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;
   ```
2. 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;
   ```
3. Copy the Editor.java, ActionPerform.java, ColorClass.java, FontClass.java, PrintClass.java, and Help.java files to a folder on your computer. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Text Editor, specify the following command at the command prompt:
   ```
   java Editor
   ```
5. Select the File->New command from the menu bar. A new text file is opened in the Text Editor, as shown in Figure 6-7:

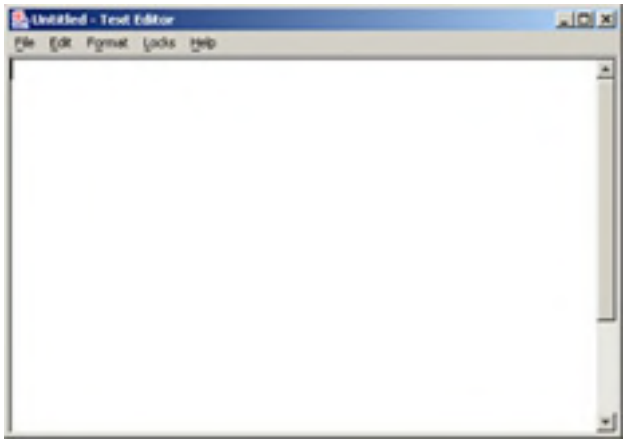


**Figure 6-7:** The Untitled Text Editor Window

6. Write Hello, this is a sample file, in the currently untitled text file.
7. Select the File->Save As command from the menu bar to save the file. A Save As dialog box appears, as shown in Figure 6-8:

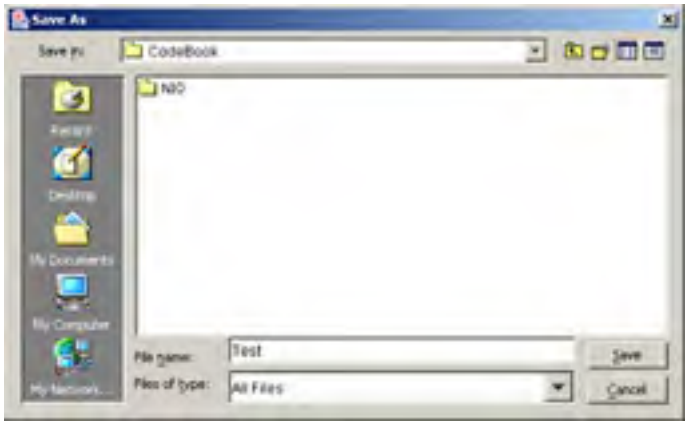


**Figure 6-8:** Save As Dialog Box

8. In the Save As dialog box, specify the file name as Test and location as C:\CodeBook.
9. Select the File->New command from the menu bar to clear the text area.
10. Select the File->Open command from the menu bar to open the file, Test.txt. The Open dialog box appears, as shown in Figure 6-9:


**Figure 6-9:** Open Dialog Box

11. Click the Open button of the Open dialog box that opens the Test.txt file from C:\CodeBook in the Text Editor, as shown in Figure 6-10:

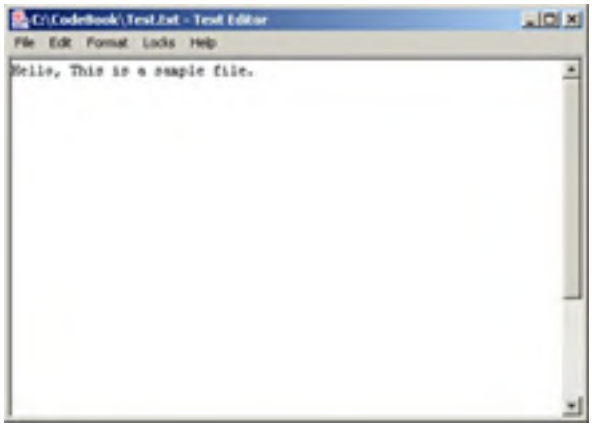


**Figure 6-10:** Displaying the Test.txt File

12. Select the Format->Font command from the menu bar to open the Font dialog box, as shown in Figure 6-4. Specify the font as Verdana, size as 20, and style as Italic. Click the OK button to apply the font changes to the file, as shown in Figure 6-11:


**Figure 6-11:** Displaying the Test.txt File with Selected Font

13. Select the Format->Color command from the menu bar to open the Color dialog box, as shown in Figure 6-10. Specify the RGB values in this dialog box and click OK. The color of the file is modified, as shown in Figure 6-12:

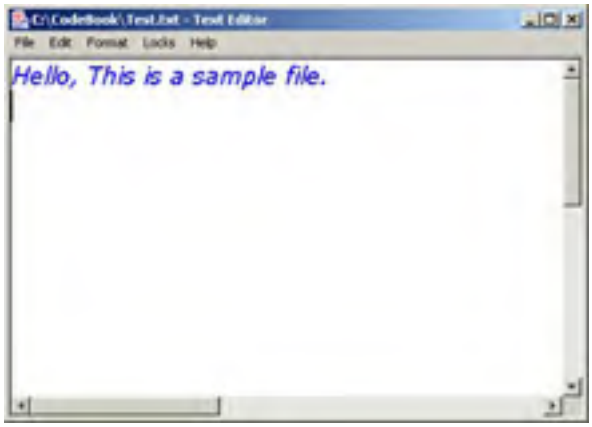


**Figure 6-12:** Displaying the Test.txt File with Selected Color

14. Select the Lock->Exclusive Lock command from the menu bar. This locks the Test.txt file.
15. Select the Start->Programs->Accessories->Notepad command to open a Windows notepad utility.
16. Select the File->Open command from the menu bar of the Notepad utility. Open the Text.txt file from the C:\CodeBook location.
17. Change the contents of the file and select the File->Save command from the menu bar. An error message appears because the file is locked, as shown in Figure 6-13:

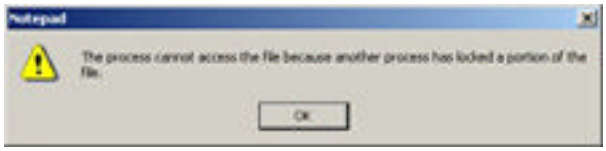


**Figure 6-13:** Displaying Error Message



◄ PREVIOUS NEXT ►

# Chapter 7: Creating a Network Information Application

The New Input/Output (NIO) API provides the java.nio, java.nio.channels, and java.nio.charset packages for buffer management, socket handling, and data transfer. The java.nio package contains the ByteBuffer and CharBuffer classes that you can use to read and store the bytes and characters. To connect the application to a network, you can use the SocketChannel class that is available in the java.nio.channel package. The java.nio.charset package provides two classes, Charset and CharsetDecoder. You can use these classes to set the character sets and decode the bytes to characters.

This chapter explains how to develop a Network Information application using the java.nio, java.nio.channels, and java.nio.charset packages.

## Architecture of the Network Information Application

The Network Information application displays a list of computers connected in a network. Using this application, you can display the echo time or the current system date and time of a specific computer. For example, ABC Inc has corporate offices in three countries, X, Y, and Z. A network administrator in ABC Inc's office in X wants to know the time displayed in the HR Manager's computer, which is located at Y. The Network Information application gives the network administrator the ability to retrieve this information. The Network Information application uses the following files:

- NetCompFrame.java: Creates a user interface for the Network Information application that helps display a list of network computers.
- NetCompConnect.java: Implements the functions of the Network Information application. This file connects the application to the network using socket channels.
- CompInfo.java: Provides information about the network computer.
- CompInfoDialog.java: Creates a dialog box that displays the echo time and current system date and time of a computer.

Figure 7-1 shows the architecture of the Network Information application:

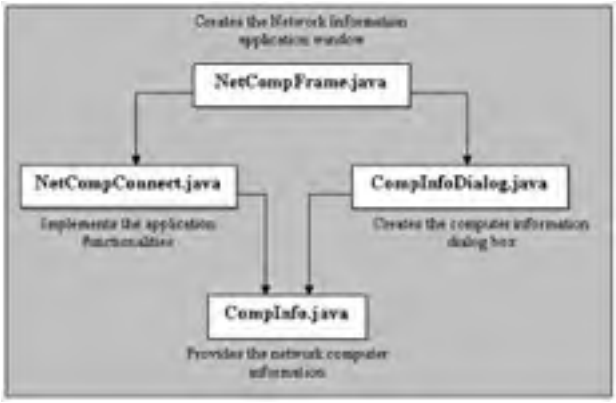


**Figure 7-1:** Architecture of the Network Information Application

In the Network Information application, the NetCompFrame.java file calls the NetCompConnect.java and CompInfoDialog.java files. The NetCompConnect.java file connects the application to the network at the specified port number and calls the CompInfo.java file to retrieve the computer information. The CompInfoDialog.java file displays the computer information by calling the CompInfo.java file.

# Creating the User Interface for the Network Information Application

The NetCompFrame.java file helps create a user interface that allows you to retrieve a list of network computers. You can use this list to retrieve the echo time or the current system date and time of a network computer.

Listing 7-1 shows the contents of the NetCompFrame.java file:

**Listing 7-1: The NetCompFrame.java File**

```
/*Imports javax.swing package classes. */
import javax.swing.*;
/*Imports javax.swing.event package classes. */
import javax.swing.event.*;
/*Imports java.awt package classes. */
import java.awt.*;
import java.awt.event.*;
/*Imports java.awt.event package classes. */
import java.net.*;
/*Imports java.util package classes. */
import java.util.*;
/*
Class NetCompFrame - This class creates the user interface of the Network Information
application. This user interface displays the list of all the computers connected to the network.
Fields:
   cont - Creates the container for the application.
   findComp_lbl - Creates the label of Service/Protocol.
   findComp_cmb - Creates the combo box to select the service or protocol.
   netMask_lbl - Creates the label of the Host IP Address.
   netMask_txt - Creates the text box where the end user can specify the IP address.
   netComp_lbl - Creates the label of the network computers.
   netComp_lst - Creates a list box that displays the computer name.
   netComp_sp - Creates the scrollpane for the list box.
   showComp_btn -Creates the Display List button.
   clear_btn - Creates the Clear button.
   close_btn - Creates the Close button.
   listDataVec - Stores the list of computers in vector form.
   compInfoVec - Stores the computer's information in the vector form.
   selectedItem - Contains the selected item from the list.
Methods:
   actionPerformed() - This method is invoked when end user clicks any of the button
   of the network management application.
   getMachinesList() - This method is invoked to display the list of computers
   connected in the network.
   valueChanged() - This method is invoked end user selects the computer from the computer list.
   itemStateChanged() - This method is invoked when end user selects the
   service/ protocol from the combo box.
   main() - This method creates the main window of the application.
   */
   public class NetCompFrame extends JFrame implements ActionListener, ListSelectionListener, ItemLi
   {
      /* Declares the object of the Container class. */
      Container cont = null;
      /* Declares the objects of the JLabel class. */
      JLabel findComp_lbl;
      JLabel netMask_lbl;
      JLabel netComp_lbl;
      /* Declares the objects of the JComboBox class. */
      JComboBox findComp_cmb;
      /* Declares the object of the JTextField class. */
      JTextField netMask_txt;
      /* Declares the object of the JList class. */
      JList netComp_lst;
      /* Declares the object of the JScrollPane class. */
      JScrollPane netComp_sp;
      /* Declares the objects of the JButton class. */
      JButton showComp_btn;
      JButton clear_btn;
      JButton close_btn;
      /*
      Declares and initializes the objects of the Vector class.
      */
      Vector listDataVec = new Vector();
      Vector compInfoVec = new Vector();
      /* Declares the object of String class. */
      String selectedItem = "";
      /*
      Declares the object of the NetCompConnect class.
      */
      NetCompConnect compConnect = null;
      /*
      Declares the object of the CompInfoDialog class.
      */
      CompInfoDialog dialogBox = null;
      /* Defines the default constructor. */
```

```
   public NetCompFrame(String title)
   {
      super(title);
      try
      {
         /*
         Initializes and sets the look and feel of the application to Windows look and feel.
         */
         UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
      }
      catch(Exception e){}
      /* Sets the size of the application. */
      setSize(350, 400);
      /*
      addWindowListener - It contains a windowClosing() method.
      windowClosing: It is called when the end user clicks the close button of the Window.
      It closes the main window.
      Parameter: we - Represents the object of the WindowEvent class.
      Return Value: NA
      */
      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent we)
         {
            System.exit(0);
         }
      });
   /* Initializes the object of the Container class. */
   cont = getContentPane();
   /* Sets the layout of the container to NULL layout. */
   cont.setLayout(null);
   /* Initializes the objects of the JLabel class. */
   findComp_lbl = new JLabel("Service/Protocol: ");
   netMask_lbl = new JLabel("Host IP Address: ");
   netComp_lbl = new JLabel("Network Computers: ");
   /* Initializes the object of the JTextField class. */
   netMask_txt = new JTextField();
   netMask_txt.setFont(new Font("Verdana", Font.PLAIN, 12));
   /* Initializes the object of the JComboBox class. */
   findComp_cmb = new JComboBox();
   findComp_cmb.setFont(new Font("Verdana", Font.PLAIN, 12));
   /* Adds the items in the combo box. */
   findComp_cmb.addItem("Echo");
   findComp_cmb.addItem("Date Time");
   /* Adds the itemListener event to the combo box. */
   findComp_cmb.addItemListener(this);
   /* Initializes the object of the JList class. */
   netComp_lst = new JList();
   netComp_lst.setFont(new Font("Verdana", Font.PLAIN, 12));
   netComp_lst.addListSelectionListener(this);
   /* Initializes the object of the JScrollPane class. */
   netComp_sp = new JScrollPane(netComp_lst);
   /* Initializes the objects of the JButton class. */
   showComp_btn = new JButton("Display List");
   clear_btn = new JButton("Clear");
   close_btn = new JButton("Close");
   /* Adds actionListener events to the JButton objects. */
   showComp_btn.addActionListener(this);
   clear_btn.addActionListener(this);
   close_btn.addActionListener(this);
   /* Sets the position of the components on the application window. */
   findComp_lbl.setBounds(10, 10, 140, 20);
   findComp_cmb.setBounds(160, 10, 100, 20);
   netMask_lbl.setBounds(10, 40, 110, 20);
   netMask_txt.setBounds(160, 40, 170, 20);
   showComp_btn.setBounds(230, 70, 100, 25);
   netComp_lbl.setBounds(10, 100, 300, 20);
   netComp_sp.setBounds(10, 125, 320, 200);
   clear_btn.setBounds(120, 335, 100, 25);
   close_btn.setBounds(230, 335, 100, 25);
   /* Adds the swing components on the container. */
   cont.add(findComp_lbl);
   cont.add(findComp_cmb);
   cont.add(netMask_lbl);
   cont.add(netMask_txt);
   cont.add(netComp_lbl);
   cont.add(netComp_sp);
   cont.add(showComp_btn);
   cont.add(clear_btn);
   cont.add(close_btn);
   /* Sets the visibility of the frame to true. */
   setVisible(true);
   /* Sets the reliability of the frame to false. */
   setResizable(false);
}
/*
actionPerformed() - This method is called when the end user clicks any of the button from the win
Parameters: ae - Represents an object of the ActionEvent class that contains the details of the e
```

```
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   /* This section is executed when an end user clicks the Display List button. */
   if (ae.getSource() == showComp_btn)
   {
      /* Gets the computer name from the selected item. */
      String cmb_selection = (String)findComp_cmb.getSelectedItem();
      /*
      This section is executed when the end user selects the "Echo" option from the combo box.
      */
      if (cmb_selection.trim().equals("Echo"))
      {
         /*
         Gets the machine IP address from the Host IP Address text field.
         */
         String machineIPAddress = netMask_txt.getText();
         if (machineIPAddress==null || machineIPAddress.trim().equals(""))
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please Enter the IP address of your machine.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         try
         {
            /*
            Creates and initializes the object of the InetAddress class.
            */
            InetAddress myMachineAddress = InetAddress.getByName(machineIPAddress.trim());
         }
         catch (Exception ex)
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please enter a valid IP address.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         /* Calls the getMachinesList() method. */
         getMachinesList(machineIPAddress.trim(), 7);
      }
      /*
      This section is executed when end user selects the "Date Time" option from the combo box.
      */
      else if (cmb_selection.trim().equals("Date Time"))
      {
         /*
         Gets the machine IP address from the Host IP Address text field.
         */
         String machineIPAddress = netMask_txt.getText();
         if (machineIPAddress==null || machineIPAddress.trim().equals(""))
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please Enter the IP address of your machine.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         try
         {
            /*
            Creates and initializes the object of the InetAddress class.
            */
            InetAddress myMachineAddress = InetAddress.getByName(machineIPAddress.trim());
         }
         catch (Exception ex)
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please enter a valid IP address.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         /* Calls the getMachinesList() method. */
         getMachinesList(machineIPAddress.trim(), 13);
      }
   }
   /* This section is executed when an end user clicks the Clear button. */
   else if (ae.getSource() == clear_btn)
   {
      /* Calls the setListData() method of the JList class. */
      netComp_lst.setListData(listDataVec=new Vector());
   }
   /* This section is executed when an end user clicks the Close button. */
   else if (ae.getSource() == close_btn)
```

```
   {
      System.exit(0);
   }
}
/*
getMachinesList() - This method is invoked when the end user clicks the
Display List button of the Network Management Application window.
Parameters:
myAddress -   Contains the IP address of the user machine.
port - Contains the port number of the user machine.
Return Value: NA
*/
public void getMachinesList(String myAddress, int port)
{
   /* Initializes the object of the Vector class. */
   compInfoVec = new Vector();
   /* Calls the setListData() method of the JList class. */
   netComp_lst.setListData(listDataVec=new Vector());
   /* Creates and initializes the object of the StringTokenizer class. */
   StringTokenizer st = new StringTokenizer(myAddress, ".");
   /* Gets the IP address from the tokenize string. */
   String rawIPAddress = (String)st.nextToken() + "." + (String)st.nextToken() + "." +
   (String)st.nextToken() + ".";
   /* Initializes an object of the String class to store the IP address. */
   String testIPAddress = null;
   for (int i=0; i<256; i++)
   {
      /*
      Gets the IP address of the computers that are connected to the network.
      */
      testIPAddress = rawIPAddress + i;
      /* Creates and initializes the object of the NetCompConnect class. */
      compConnect = new NetCompConnect(testIPAddress, this, port);
      /* Calls the start() method of the NetCompConnect class. */
      compConnect.start();
   }
   try
   {
      Thread.sleep(5000);
   }
   catch (InterruptedException iex){}
   /* Evaluates the size of the vector. */
   int size = compInfoVec.size();
   for (int j=0; j<size; j++)
   {
      /* Creates and initializes the object of the CompInfo class. */
      CompInfo tempObj = (CompInfo)compInfoVec.elementAt(j);
      /* Inserts the element at the end of the vector. */
      listDataVec.addElement(tempObj.getAddress());
   }
   /* Calls the setListData() method of the JList class. */
   netComp_lst.setListData(listDataVec);
}
/*
valueChanged() - This method is called when the end user selects the
computer name from the list.
Parameters: e - Represents an object of the ListSelectionEvent class
that contains the details of the event.
Return Value: NA
*/
public void valueChanged(ListSelectionEvent e)
{
   /*
   This section is executed when an end user changes the selection value of the computer from the
   */
   if (e.getSource() == netComp_lst)
   {
      /* Gets the computer name from the selected item. */
      String selectedItem2 = (String)netComp_lst.getSelectedValue();
      if (selectedItem2 == null)
      {
         return;
      }
      if (selectedItem.equals(selectedItem2))
      {
         /* Clears the item selection in the list. */
         netComp_lst.clearSelection();
      }
      /*
      This section is executed when the previously selected and currently
      selected computer names are different.
      */
      if (!selectedItem.equals(selectedItem2))
      {
         selectedItem = selectedItem2;
         /* Evaluates the size of the vector. */
         int size = compInfoVec.size();
         for (int j=0; j<size; j++)
```

```
                {
                   /* Creates and initializes the object of the CompInfo class. */
                   CompInfo tempObj = (CompInfo)compInfoVec.elementAt(j);
                   /* Gets the IP address using the getAddress() method of the CompInfo class. */
                   String compAddress = tempObj.getAddress();
                   if (compAddress.equals(selectedItem))
                   {
                      /*
                      Gets the computer name from the selected item.
                      */
                      String cmb_selection = (String)findComp_cmb.getSelectedItem();
                      /*
                      Gets the computer information of the selected item using the getCompInfo()
                      method of the CompInfo class. */
                      String compInfo = tempObj.getCompInfo();
                      /*
                      This section is executed when the end user selects "Echo" from the combo box.
                      */
                      if (cmb_selection.trim().equals("Echo"))
                      {
                         /* Creates and initializes the object of the CompInfoDialog class. */
                         dialogBox = new CompInfoDialog(this, "Echo Response", true, 7, compInfo);
                         /* Clears the item selection in the list. */
                         netComp_lst.clearSelection();
                      }
                      /*
                      This section is executed when the end user selects "Date Time" from the combo box.
                      */
                      else if (cmb_selection.trim().equals("Date Time"))
                      {
                         /*
                         Creates and initializes the object of the CompInfoDialog class.
                         */
                         dialogBox = new CompInfoDialog(this, "System Date & Time", true, 13, compInfo);
                         /* Clears the item selection in the list. */
                         netComp_lst.clearSelection();
                      }
                      break;
                   }
                }
             }
          }
       }
       /*
       itemStateChanged - This method is called when the end user selects the
       Service/Protocol from the combo box.
       Parameters: ie - a ListSelectionEvent object containing details of the event.
       Return Value: NA
       */
       public void itemStateChanged(ItemEvent ie)
       {
          /* This section is executed when the end user selects any value from the combo box. */
          if (ie.getSource() == findComp_cmb)
          {
             /* Calls the setListData() method of the JList class. */
             netComp_lst.setListData(listDataVec=new Vector());
          }
       }
       /*
       main() - This method creates the main window of the user interface and displays it.
       Parameters:
       args[] - Contains any command line arguments passed.
       Return Value: NA
       */
       public static void main(String[] args)
       {
          /* Creates and initializes the object of the NetCompFrame class. */
          NetCompFrame frame = new NetCompFrame("Network Information Application");
       }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the NetCompFrame class. This class generates the main window of the Network Information application, as shown in Figure 7-2:

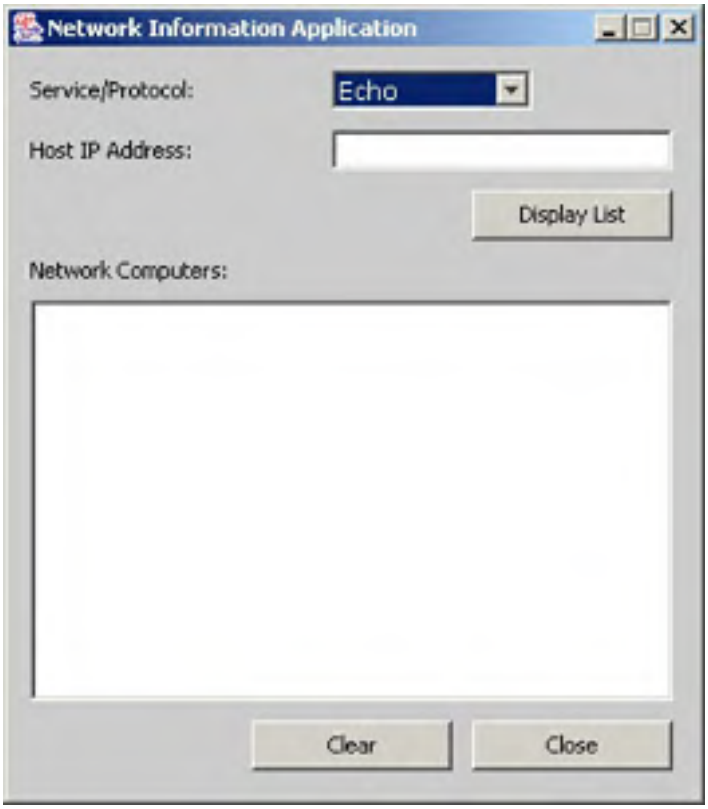


**Figure 7-2:** The Network Information Application User Interface

When the end user specifies the required service or protocol in the combo box and the host IP address in the Host IP Address text field, a list of network computers is displayed in the Network Computers list box.

The methods defined in the above code are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method based on the button the end user clicks. If the end user clicks the Display List button, the actionPerformed() method retrieves the computer name from the selected item and checks the combo box value. If the end user selects Echo in the combo box, the actionPerformed() method retrieves the host IP from the Host IP Address text field and calls the getMachinesList() method with port number 7. If the end user selects the Date/Time value, the actionPerformed() method retrieves the host IP from the Host IP Address text field and calls the getMachinesList() method with port number 13. When an end user clicks the Clear button, the actionPerformed() method calls the setListData() method of the JList class. The setListData() method clears the list box. When an end user clicks the Close button, the actionPerformed() method calls the System.exit(0) method to close the application.
- `getMachinesList()`: Retrieves the IP address of the network computers and initializes the object of the NetCompConnect class. This method then calls the start() method of the NetCompConnect class and initializes the object of the CompInfo class. Next, the getMachinesList() method inserts the element at the end of the vector and calls the setListData() method of the JList class to add the list in the list box.
- `valueChanged()`: Acts as an event listener and activates an appropriate class or method based on the list value the end user selects. When the end user changes the selection of the item in the list, this method checks the selected item with the previously selected item. If both the item names are same, this method calls a break; otherwise it creates and initializes the object of the CompInfo class. This method then gets the IP address using the getAddress() method of the CompInfo class and gets the computer information of the selected item using the getCompInfo() method of the CompInfo class. Next, the valueChanged() method checks the combo box value and creates the object of the CompInfoDialog class based to the value specified in the combo box.
- `itemStateChanged()`: Acts as an event listener and activates an appropriate class or method based on the option the end user changes in the combo box. After the end user modifies the option in the combo box, this method calls the setListData() method of the JList class. The setListData() method clears the list box.


PREVIOUS NEXT

# Implementing the Network Functionality

The NetCompConnect.java file implements the functions of the Network Information application. This file reads the hosts IP address and port number on which the end user wants to create the socket channel. The NetCompConnect.java file then retrieves information, such as echo time and current system date and time of the computer the end user selects. The NetCompConnect.java file passes this information to the CompInfo.java file, which sends it to the CompInfoDialog.java file. The CompInfoDialog.java file displays the computer information in a dialog box.

Listing 7-2 shows the contents of the NetCompConnect.java file:

**Listing 7-2: The NetCompConnect.java File**

```
/* Imports java.net package classes. */
import java.net.*;
/* Imports java.io package classes. */
import java.io.*;
/* Imports java.nio package classes.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;

/*
Class NetCompConnect - This class connects the application to the network using socket channel.
This class also performs the ping and date/time operations.
Fields:
   strCompAddr - Contains the IP address.
   mainApp - Creates the object of the NetCompFrame class.
   port - Contains the port number.
   charset - Contains the charset.
   decoder - Creates the decoder.
   buff - Creates the buffer.
   isa - Contains the object of the InetSocketAddress class.
   sc - Create the socket channel.
Methods:
   connectComp() - This method creates the socket channel at the specified port number.
   run() - This method is invoked when the thread is started.
   */
public class NetCompConnect extends Thread
{
   /* Declares the object of the String class. */
   String strCompAddr = null;
   /* Declares the object of the NetCompFrame class. */
   NetCompFrame mainApp = null;
   /* Declares the object of the Charset class. */
   Charset charset;
   /* Declares the object of the CharsetDecoder class. */
   CharsetDecoder decoder;
   /* Declares the object of the ByteBuffer class. */
   ByteBuffer buff;
   /* Declares the object of the InetSocketAddress class. */
   InetSocketAddress isa;
   /* Declares the object of the SocketChannel class. */
   SocketChannel sc = null;
   int port = 0;
   /* Defines the default constructor. */
   public NetCompConnect(String strCompAddr, NetCompFrame mainApp, int port)
   {
      this.strCompAddr = strCompAddr;
      this.mainApp = mainApp;
      this.port = port;
   }
   /*
      connectComp() - This method creates and initializes the socket at port 7 or 13 to
      retrieve the echo time or date/time of the specific computer.
      Parameter:
      strCompAddr - Stores the computer IP address.
      port - Stores the port number.
      Return Value: NA
   */
   public int connectComp(String strCompAddr, int port)
   {
      String compInfo = "";
      String temp = "";
      String time = "";
      String am_pm = "";
      String date = "";
      String date_time = "";
      /* Gets the current time of the system in milliseconds. */
      long startTime = System.currentTimeMillis();
      try
      {
         /*
         Checks the port number, if port number is 13, this code is executed.
```

```
*/
if (port == 13)
{
   /* Initializes the object of the Charset class. */
   charset = Charset.forName("US-ASCII");
   /* Initializes the object of the CharsetDecoder class. */
   decoder = charset.newDecoder();
   /* Initializes and allocates the size of the buffer. */
   buff = ByteBuffer.allocateDirect(1024);
   /* Initializes the object of the InetSocketAddress class at port 13. */
   isa = new InetSocketAddress(strCompAddr, port);
   try
   {
      /* Opens the socket channel. */
      sc = SocketChannel.open();
      /* Connects the socket channel to the specified IP address. */
      sc.connect(isa);
      /* Clears the buffer. */
      buff.clear();
      /* Reads the data from the buffer. */
      sc.read(buff);
      /* Flips the buffer. */
      buff.flip();
      /*
      Creates and initializes the object of CharBuffer and stores the decoded data from the
      */
      CharBuffer cb = decoder.decode(buff);
      temp = new String(cb.array());
      StringTokenizer st = new StringTokenizer(temp, " ");
      time = (String)st.nextToken();
      am_pm = (String)st.nextToken();
      date = (String)st.nextToken();
      date_time = date + " " + time + " " + am_pm;
      compInfo = compInfo + date_time;
   }
   finally
   {
      if (sc != null)
      /* Close the socket channel. */
      sc.close();
   }
   /*
   Creates the object of CompInfo class and Adds this object to the
   compInfoVec vector of the NetCompFrame class.
   */
   mainApp.compInfoVec.addElement(new CompInfo(strCompAddr, compInfo));
   return 1;
}
/* Checks the port number, if port number is 7, this code is executed. */
else
{
   /* Initializes the object of the InetSocketAddress class at port 7. */
   isa = new InetSocketAddress(strCompAddr, port);
   try
   {
      /* Opens the socket channel. */
      sc = SocketChannel.open();
      /* Connects the socket channel to the specified IP address. */
      sc.connect(isa);
      /* Gets the socket from the socket channel. */
      Socket t = sc.socket();
      /*
      Creates and initializes the object of DataInputStream class to retrieves
      the data from the socket.
      */
      DataInputStream is = new DataInputStream(t.getInputStream());
      /*
      Creates and initializes the object of PrintStream class to put the data to the socket
      */
      PrintStream ps = new PrintStream(t.getOutputStream());
      ps.println("ping");
      String str = is.readLine();
      if (str.equals("ping"))
      {
         /* Gets the current system time. */
         long endTime = System.currentTimeMillis();
         /* Evaluates the echo time. */
         compInfo = (endTime-startTime) + " ms";
         /*
         Creates the object of CompInfo class and Adds this object to the
         compInfoVec vector of the NetCompFrame class.
         */
         mainApp.compInfoVec.addElement(new CompInfo(strCompAddr, compInfo));
         return 1;
      }
      else
      {
         return 0;
```

```
                }
            }
            finally
            {
                if (sc != null)
                /* Close the socket channel. */
                sc.close();
            }
        }
    }
    catch (Exception e)
    {
        return 0;
    }
}
/*
run() - This method is executed when thread is started.
Parameter: NA
Return Value: NA
*/
public void run()
{
    /* Calls the connectComp() method */
    connectComp(this.strCompAddr, this.port);
}
}
```

Download this Listing.

In the above code, the constructor of the NetCompConnect class reads the host IP address, port number, and object of the NetCompFrame class. The methods defined in the above code are:

- `run()`: Calls the connectComp() method to connect the application to the network.
- `connectComp()`: Checks the port number to establish a connection. If the port number is 13, this method initializes the objects of the Charset and CharsetDecoder classes. This method then initializes and allocates the size of the buffer, initializes the object of the InetSocketAddress class at port 13, and opens a socket channel on port 13. Next, the connectComp() method connects the socket channel to the specified host address, reads the bytes into buffer, and decodes the bytes from buffer to character. This method then creates the object of the CompInfo class and adds this object to the compInfoVec vector of the NetCompFrame class. If the port number is 7, the connectComp() method initializes the object of the InetSocketAddress class at port 7 and opens the socket channel on that port. The connectComp() method then connects the socket channel to the specified IP address, gets the socket from the socket channel, and evaluates the echo time in milliseconds. Next, the connectComp() method creates the object of the CompInfo class and adds this object to the compInfoVec vector of the NetCompFrame class.

# Creating the Computer Information File

The CompInfo.java file is used to provide the computer information, such as echo time and current system date and time of a specific computer.

Listing 7-3 shows the contents of the CompInfo.java file:

**Listing 7-3: The CompInfo.java File**

```
/*
Class CompInfo - This class provides the computer information of specific computer.
Fields:
   address - Contains the IP address of the computer.
   info - Contains the computer information's, such as echo time or date/time.
Methods:
   getAddress() - This method returns the IP address of the computer.
   getCompInfo() - This method returns the computer information.
*/
public class CompInfo
{
   /* Declares and initializes the objects of String class. */
   private String address = null;
   private String info = null;
   /* Defines the default constructor. */
   public CompInfo(String address, String info)
   {
      this.address = address;
      this.info = info;
   }
   /*
   getAddress() - This method is used to retrieve the IP address of a specified computer.
   Parameter: NA
   Return Value: String
   */
   public String getAddress()
   {
      return this.address;
   }
   /*
   getCompInfo() - This method is used to retrieve the computer information of a specified computer.
   Parameter: NA
   Return Value: String
   */
   public String getCompInfo()
   {
      return this.info;
   }
}
```

Download this Listing.

In the above code, the constructor of the CompInfo class takes the host IP and computer information as input parameters. The methods defined in the above code are:

- `getAddress()`: Returns the IP address of the selected computer.
- `getCompInfo()`: Returns computer information, such as echo time and current system date and time of the selected network computer.

# Creating the Computer Information Dialog Box

The CompInfoDialog.java file creates a dialog box that displays the information of the network computer.

Listing 7-4 shows the contents of the CompInfoDialog.java file:

**Listing 7-4: The CompInfoDialog.java File**

```
/* Import java packages */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
Class CompInfoDialog - This class creates a computer information dialog box.
Fields:
   cont - Creates the container.
   key_lbl - Creates the label for "Echo Time" or "Date/Time".
   value_lbl - Creates the label to display the result.
Methods:
   actionPerformed() - This method is invoked when end user clicks the OK button.
*/
public class CompInfoDialog extends JDialog implements ActionListener
{
   /* Declares the object of the Container class. */
   Container cont = null;
   /* Declares the object of the JLabel class. */
   JLabel key_lbl;
   JLabel value_lbl;
   /* Declares the object of the JButton class. */
   JButton ok_btn;
   /* Defines the default constructor.   */
   public CompInfoDialog(NetCompFrame parent, String title, boolean modal, int port, String value)
   {
      super(parent, title, modal);
      /* Set the size of the Computer Information dialog box. */
      setSize(200, 140);
      /* Creates the object of the Point class to get the parent frame location. */
      Point p = parent.getLocation();
      /* Set the location of the dialog box. */
      setLocation((int)p.getX()+10, (int)p.getY()+10);
      /*
      addWindowListener - It contains the windowClosing() method.
      windowClosing: It is called when the user clicks the cancel button of the Window.
      It closes the main window.
      Parameter: we- Object of WindowEvent class.
      Return Value: NA
      */
      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent we)
         {
            dispose();
         }
      });
      cont = this.getContentPane();
      /* Sets the layout of the application to null. */
      cont.setLayout(null);
      /* This section is executed, if port number equals to 13. */
      if (port == 13)
      {
         /* Initialize the label. */
         key_lbl = new JLabel("Date and Time", SwingConstants.CENTER);
      }
      /* This section is executed, if port number equals to 7. */
      else
      {
         /* Initialize the label. */
         key_lbl = new JLabel("Echo Time", SwingConstants.CENTER);
      }
      /* Initializes the object of the JLabel class */
      value_lbl = new JLabel(value, SwingConstants.CENTER);
      /* Initializes the object of the JButton class */
      ok_btn = new JButton("Ok");
      /* Adds the actionListener event to the ok button. */
      ok_btn.addActionListener(this);
      /* Sets the position of the swing components on the container. */
      key_lbl.setBounds(0, 10, 200, 20);
      value_lbl.setBounds(0, 40, 200, 20);
      ok_btn.setBounds(70, 70, 60, 20);
      /* Adds the components to the container. */
      cont.add(key_lbl);
      cont.add(value_lbl);
      cont.add(ok_btn);
      show();
```

```
    }
    /*
    actionPerformed() - This method is called when the user selects any menu item from the menu bar.
    Parameters:   ae - an ActionEvent object containing details of the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent ae)
    {
       if (ae.getSource() == ok_btn)
       {
          dispose();
       }
    }
}
```

Download this Listing.

In the above code, the constructor of the CompInfoDialog class reads the object of the NetCompFrame class, title, model, port number, and value. Here, title represents a string that sets the title, model represents the Boolean value, and value represents the string that contains the network information.

When an end user clicks the OK button in the Computer Information dialog box, the actionPerformed() method is invoked. This method calls the dispose() method to close the dialog box.

# Unit Testing

To test the Network Information application:

1. 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;
   ```
2. 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;
   ```
3. Copy the NetCompFrame.java, NetCompConnect.java, CompInfo, and CompInfoDialog.java files to a folder on your computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Network Information application, specify the following command at the command prompt:
   ```
   java NetCompFrame
   ```
   The above command opens the main window of the Network Information application, as shown in Figure 7-2.
5. Select the Echo option from the Service/Protocol combo box and specify the host IP address, such as 192.168.0.36. If you do not specify the IP address, an Error dialog box appears, as shown in Figure 7-3:


**Figure 7-3:** Error Dialog Box

6. If you specify a wrong host IP address, another Error dialog box appears, as shown in Figure 7-4:


**Figure 7-4:** Error Dialog Box with Error Message

7. Click the Display List button of the Network Information application. A list of network computers appears in the Network Computer list box, as shown in Figure 7-5:

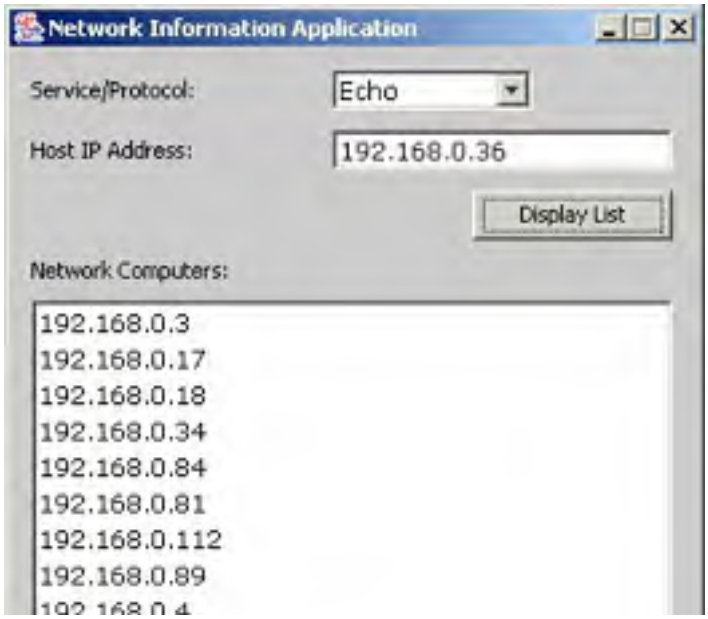


**Figure 7-5:** Network Information Application Window with Ehco Service

8. Select an IP address, such as 192.168.0.89, from the Network Computer list box. A Echo Response dialog box opens, as shown in Figure 7-6:

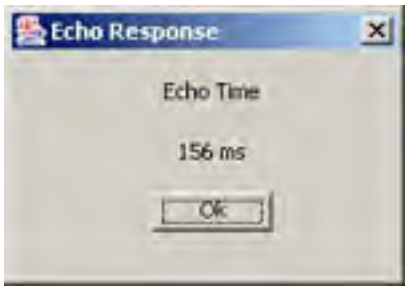


**Figure 7-6:** Echo Response Dialog Box

9. Select the Date/Time option from the Service/Protocol combo box and specify the host IP address, such as 192.168.0.36.
10. Click the Display List button of the Network Information application. A list of network computers appears in the Network Computer list box, as shown in Figure 7-7:

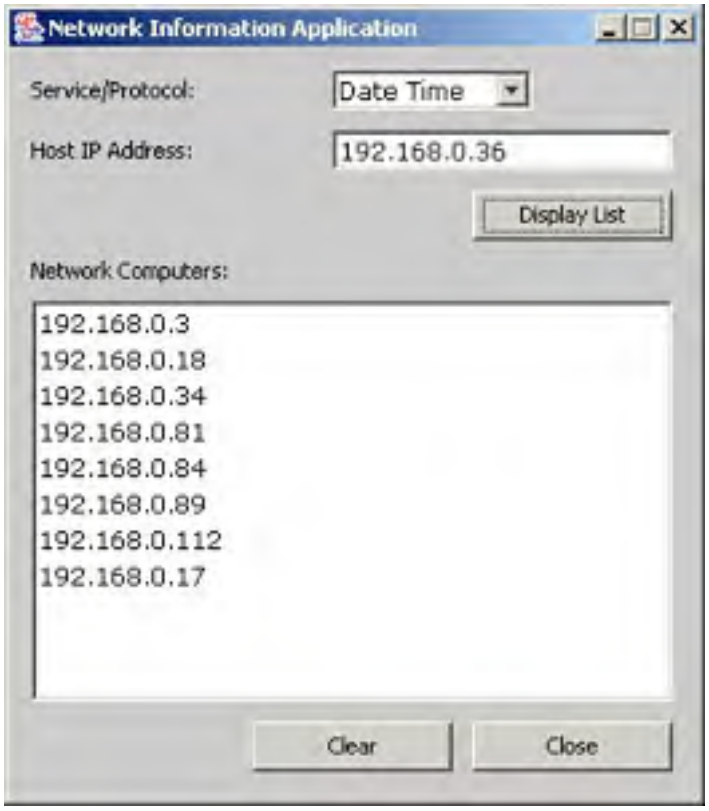


**Figure 7-7:** Network Information Application Window with Date Time Service

11. Select an IP address, such as 192.168.0.89, from the Network Computer list box. A System Date & Time dialog box opens, as shown in Figure 7-8:


**Figure 7-8:** System Date and Time Dialog Box



PREVIOUS NEXT

# Chapter 8: Creating an Encoder/Decoder Application

The New Input/Output (NIO) API provides the java.nio, java.nio.channels, and java.nio.charset packages for buffer management, socket handling, and text encoding and decoding. The java.nio package contains the ByteBuffer and CharBuffer classes that allow you to read and store the bytes and characters. To read a file, you can use the FileChannel class available in the java.nio.channel package. The java.nio.charset package provides the Charset and CharsetEncoder classes that help set the character sets. These classes encode the text using a specific encoding scheme. To decode an encoded text, you can use the CharsetDecoder class the java.nio.charset package.

An encoding scheme is a standard by which you map the coded character set to octets of eight bit sequence. The coded character set is the assignment of numeric values to a set of characters. The encoding scheme defines how a sequence of character encoding is represented as a sequence of bytes. The numeric value of the character set does not match with the encoded bytes. There are several types of encoding schemes, such as UTF-8, UTF-16, ISO-8859-1, UTF-16BE, or UTF-16LE. For example, UTF-8 encoding scheme encodes the coded character code of value less than 0x80 to a sequence of eight bytes.

This chapter explains how to develop an Encoder/Decoder application using the java.nio, java.nio.channels, and java.nio.charset packages.

## Architecture of the Encoder/Decoder Application

The Encoder/Decoder application allows an end user to encode the text using a specific encoding scheme or decode the encoded text into a readable form.

The Encoder/Decoder application uses the following files:

- `EncoderDecoder.java`: Creates a user interface that an end user can use to encode or decode a specified text.
- `EncodingSchemes.java`: Creates a dialog box to select the encoding scheme.

Figure 8-1 shows the architecture of the Encoder/Decoder application:

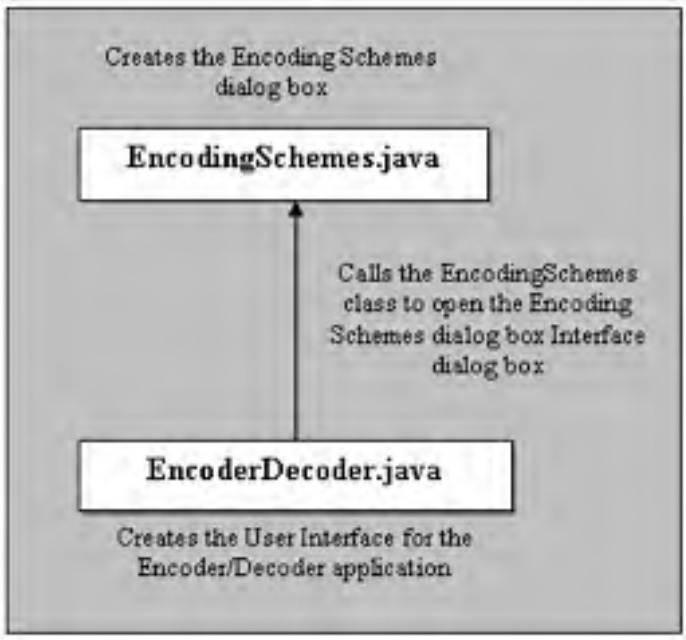


**Figure 8-1:** Architecture of the Encoder/Decoder Application

The EncoderDecoder.java file encodes and decodes a specified text. To select an encoding scheme, the EncoderDecoder.java file calls the EncodingSchemes.java file. The EncodingSchemes.java file returns the encoding scheme name to the EncoderDecoder.java file.

# Creating the User Interface for the Encoder/Decoder Application

The EncoderDecoder.java file helps create a user interface with a set of labels, text box, text areas, and buttons for the Encoder/Decoder application.

Listing 8-1 shows the contents of the EncoderDecoder.java file:

**Listing 8-1: The EncoderDecoder.java File**

```
/* Imports java.nio package classes. */
import java.nio.*;
import java.nio.charset.*;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.*;
import java.nio.channels.FileChannel;
import java.nio.charset.UnsupportedCharsetException;
/* Imports java.io package classes. */
import java.io.*;
import java.io.IOException;
/* Imports javax.swing package classes. */
import javax.swing.*;
import javax.swing.JOptionPane;
/* Imports java.awt package classes. */
import java.awt.*;
/* Imports java.awt.event package classes. */
import java.awt.event.*;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
/*
class EncoderDecoder - This class creates a GUI for the Encoder/Decoder application.
This class also provides the methods to encode the text and decode
the encoded text into a readable format.
Fields:
pan1 - Creates a panel that can contain a text box and two buttons.
titleLabel - Creates an Encoder/Decoder Application label.
selectLabel - Creates a select file label.
encodeLabel - Creates an encoded text label.
decodeLabel - Creates a decoded text label.
selectText - Creates a select file text field.
encodeArea - Creates an Encoded Text text area.
decodeArea - Creates a Decoded Text text area.
encodePane - Creates a scroll pane for the Encoded Text text area.
decodePane - Creates a scroll pane for the Decoded Text text area.
browseButton - Creates a Browse button.
encodeButton - Creates an Encode button.
selectButton - Creates a Select Encoding Scheme button.
resetButton - Creates a Reset button.
decodeButton - Creates a Decode button.
closeButton - Creates a Close button.
file_text - Represents a string that stores the contents of a file.
file - Represents a file.
jfc - Creates a file chooser dialog box.
fin - Creates a file input stream.
fchan - Creates a file channel.
buff_in - Stores the content of the opened files.
buff_out - Stores the content of the encoded file that needs to be further
encrypted in the hexadecimal format.
bbuf - Stores the encoded bytes.
cbuf - Stores the decoded bytes.
fsize - Contains the size of the file.
decoder - Creates a decoder that decodes the encoded text.
encoder - Creates an encoder that encodes the text.
charset - Creates a charset object.
encScheme - Represents an object of the EncodingSchemes class.
Methods:
keyTyped() - This method is invoked when an end user types any text in the Text
to Encode text area.
actionPerformed() - This method is invoked when an end user clicks the any
button of the Encoder/Decoder application.
open() - This method is invoked to open an existing text file that the end
user wants to encode.
encode() - This method is invoked to encode the text.
decode() - This method is invoked to decode the encoded text.
main() - This method creates the main window of the application and displays it.
*/
public class EncoderDecoder extends JDialog implements ActionListener, KeyListener
{
/* Declares the object of the JPanel class. */
JPanel pan1;
/* Declares the objects of the JLabel class. */
JLabel titleLabel;
JLabel textLabel;
```

```
JLabel selectLabel;
JLabel encodeLabel;
JLabel decodeLabel;
/* Declares the object of the JTextField class. */
JTextField selectText;
JTextField schemeText;
/* Declares the objects of the JTextArea class. */
JTextArea encodeArea;
JTextArea decodeArea;
JTextArea textArea;
/* Declares the objects of the JScrollPane class. */
JScrollPane encodePane;
JScrollPane decodePane;
JScrollPane textPane;
/* Declares the objects of the JButton class. */
JButton browseButton;
JButton encodeButton;
JButton selectButton;
JButton resetButton;
JButton decodeButton;
JButton closeButton;
/* Declares the object of the GridBagLayout class. */
GridBagLayout gbl;
/* Declares the object of the GridBagConstraints class. */
GridBagConstraints gbc;
/* Declares the objects of the String class. */
String str;
String file_text;
/* Declares the object of the File class. */
File file;
/* Declares the object of the JFileChooser class. */
JFileChooser jfc;
/* Declares the object of the FileInputStream class. */
FileInputStream fin;
/* Declares the object of the FileChannel class. */
FileChannel fchan;
/* Declares the objects of the ByteBuffer class. */
ByteBuffer buff_in;
ByteBuffer buff_out;
ByteBuffer bbuf;
/* Declares the object of the CharBuffer class. */
CharBuffer cbuf;
/* Declares the object of the CharsetDecoder class. */
CharsetDecoder decoder;
/* Declares the object of the CharsetEncoder class. */
CharsetEncoder encoder;
/* Declares the object of the Charset class. */
Charset charset;
/* Declares the object of the EncodingSchemes class. */
public EncodingSchemes encScheme;
long fsize;
/* Defines the default constructor of the EncoderDecoder class. */
public EncoderDecoder()
{
   /* Sets the title of the Encoder/Decoder application. */
   setTitle("Encoder / Decoder Application");
   /* Sets the size of the Encoder/Decoder application. */
   setSize(497,640);
   /* Sets the visibility of the Encoder/Decoder application to true. */
   setVisible(true);
   /* Sets the re-sizability of the Encoder/Decoder application to false. */
   setResizable(false);
   /* Initializes the object of the GridBagLayout class. */
   gbl = new GridBagLayout();
   /* Sets the Layout */
   getContentPane().setLayout(gbl);
   /* Creates an object of the GridBagConstraints class. */
   gbc = new GridBagConstraints();
   /* Initializes the titleLabel object and adds it to the 0, 0, 3, 1 position with CENTER alignment
   gbc.gridx = 0;
   gbc.gridy = 0;
   gbc.gridwidth = 3;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   titleLabel = new JLabel(" Encoder/Decoder Application");
   titleLabel.setFont(new Font("Verdana", Font.BOLD, 20));
   getContentPane().add(titleLabel, gbc);
   /* Initializes and adds a separator to the 0, 1, 3, 1 position with CENTER alignment. */
   gbc.gridx = 0;
   gbc.gridy = 1;
   gbc.gridwidth = 3;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   getContentPane().add(new JSeparator(), gbc);
   /* Initializes the selectLabel object and adds it to the 0, 2, 1, 1 position with WEST alignment.
   gbc.gridx = 0;
   gbc.gridy = 2;
```

```
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
selectLabel = new JLabel("Select File to Encode:");
getContentPane().add(selectLabel, gbc);
/* Initializes and adds a separator to the 0, 3, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 3;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the selectText text field and adds it to the 0, 4, 1, 1 position with WEST alignme
gbc.gridx = 0;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
selectText = new JTextField(30);
selectText.setFont(new Font("Verdana", Font.PLAIN, 12));
getContentPane().add(selectText, gbc);
/* Initializes the browse object and adds it to the 1, 4, 1, 1 position with CENTER alignment. */
gbc.gridx = 1;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
browseButton = new JButton(" Browse ");
browseButton.addActionListener(this);
getContentPane().add(browseButton, gbc);
/* Initializes the reset object and adds it to the 2, 4, 1, 1 position with EAST alignment. */
gbc.gridx = 2;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.EAST;
resetButton = new JButton(" Reset ");
resetButton.addActionListener(this);
getContentPane().add(resetButton, gbc);
/* Initializes and adds a separator to the 0, 5, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 5;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the selectText and adds it to the 0, 6, 1, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 6;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
schemeText = new JTextField(15);
schemeText.setEditable(false);
schemeText.setText(encScheme.encodingScheme);
schemeText.setFont(new Font("Verdana", Font.PLAIN, 12));
getContentPane().add(schemeText, gbc);
/* Initializes the selectButton object and adds it to the 1, 6, 2, 1 position with WEST alignment
gbc.gridx = 1;
gbc.gridy = 6;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
selectButton = new JButton(" Select Encoding Scheme ");
selectButton.addActionListener(this);
getContentPane().add(selectButton, gbc);
/* Initializes and adds a separator to the 0, 7, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 7;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeLabel object and adds it to the 0, 8, 1, 1 position with WEST alignment.
gbc.gridx = 0;
gbc.gridy = 8;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
textLabel = new JLabel("Text to Encode:");
getContentPane().add(textLabel, gbc);
/* Initializes and adds a separator to the 0, 9, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 9;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
```

```
/* Initializes the textArea and textPane objects. Next, this textArea adds to the textPane
and adds it to the 0, 10, 3, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 10;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
textArea = new JTextArea(7, 44);
textArea.addKeyListener(this);
textArea.setLineWrap(true);
textArea.setWrapStyleWord(true);
textArea.setFont(new Font("Verdana", Font.PLAIN, 12));
textPane = new JScrollPane(textArea);
getContentPane().add(textPane, gbc);
/* Initializes and adds a separator to the 0, 11, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 11;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeButton object and adds to it to the 1, 12, 2, 1 position with WEST align
gbc.gridx = 1;
gbc.gridy = 12;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.EAST;
encodeButton = new JButton(" Encode ");
encodeButton.setEnabled(false);
encodeButton.addActionListener(this);
getContentPane().add(encodeButton, gbc);
/* Initializes and adds a separator to the 0, 13, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 13;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeLabel object and adds it to the 0, 14, 1, 1 position with WEST alignment
gbc.gridx = 0;
gbc.gridy = 14;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
encodeLabel = new JLabel("Encoded Text:");
getContentPane().add(encodeLabel, gbc);
/* Initializes and adds a separator to the 0, 15, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 15;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeArea and encodePane objects. Next, this encodeArea adds to the
encodePane and adds it to the 0, 16, 3, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 16;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
encodeArea = new JTextArea(7, 44);
encodeArea.setEditable(false);
encodeArea.addKeyListener(this);
encodeArea.setLineWrap(true);
encodeArea.setWrapStyleWord(true);
encodeArea.setFont(new Font("Verdana", Font.PLAIN, 12));
encodePane = new JScrollPane(encodeArea);
getContentPane().add(encodePane, gbc);
/* Initializes and adds a separator to the 0, 17, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 17;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the decodeButton object and adds to it to the 0, 18, 2, 1 position with EAST align
gbc.gridx = 1;
gbc.gridy = 18;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.EAST;
decodeButton = new JButton(" Decode ");
decodeButton.setEnabled(false);
decodeButton.addActionListener(this);
getContentPane().add(decodeButton, gbc);
/* Initializes and adds a separator to the 0, 19, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 19;
```

```
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the decodeLabel object and adds it to the 0, 20, 1, 1 position with WEST alignment
gbc.gridx = 0;
gbc.gridy = 20;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
decodeLabel = new JLabel("Decoded Text:");
getContentPane().add(decodeLabel, gbc);
/* Initializes and adds a separator to the 0, 21, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 21;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the decodeArea and decodePane objects. Next, this decodeArea adds to the
decodePane and adds it to the 0, 22, 3, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 22;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
decodeArea = new JTextArea(7, 44);
decodeArea.setEditable(false);
decodeArea.setLineWrap(true);
decodeArea.setWrapStyleWord(true);
decodeArea.setFont(new Font("Verdana",Font.PLAIN,12));
decodePane = new JScrollPane(decodeArea);
getContentPane().add(decodePane, gbc);
/* Initializes and adds a separator to the 0, 23, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 23;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the closeButton object and adds to it to the 0, 24, 3, 1 position with CENTER alig
gbc.gridx = 0;
gbc.gridy = 24;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
closeButton = new JButton(" Close ");
closeButton.addActionListener(this);
getContentPane().add(closeButton, gbc);
/*
addWindowListener - It contains a windowClosing() method.
windowClosing: It is called when the end user clicks the cancel button of the Window.
It closes the main window.
Parameter: we- Represents the object of the WindowEvent class.
Return Value: NA
*/
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
      System.exit(0);
   }
   });
   /* Initializes the object of the EncodingSchemes class. */
   encScheme = new EncodingSchemes(this);
}
/*
keyTyped() - This method is called when the end user enters any text in
the Encoded Text text area.
Parameters: ke - Represent an object of the KeyEvent class that contains
the details of the event.
Return Value: NA
*/
public void keyTyped(KeyEvent ke)
{
   if(ke.getSource()== textArea)
   {
      /* Enables the Encode button. */
      encodeButton.setEnabled(true);
   }
}
/* Defines the abstract methods defined in the KeyListener interface. */
public void keyPressed(KeyEvent ke){}
public void keyReleased(KeyEvent ke){}
/*
actionPerformed() - This method is called when the end user clicks the any button.
Parameters: ev - Represents an object of the ActionEvent class that contains
```

```
the details of the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ev)
{
   /* This section is executed when the end user clicks the Select Encoding Schemes button. */
   if(ev.getSource() == selectButton)
   {
      /* Sets the visibility of the Encoding Schemes frame to true. */
       encScheme.setVisible(true);
   }
   /* This section is executed when the end user clicks the Browse button. */
   else if(ev.getSource() == browseButton)
   {
       selectText.setText("");
       try
       {
         /* Initializes the object of the JFileChooser class. */
          jfc = new JFileChooser();
         /* Sets the file selection mode to FILES_ONLY. */
         jfc.setFileSelectionMode(JFileChooser.FILES_ONLY);
         /* Creates and sets the new file filter for selecting the text files*/
         jfc.setFileFilter(new javax.swing.filechooser.FileFilter()
         {
            /* accept() - This method allows the user to select only text files
            from the file chooser dialog box.
            Parameter: f - Represents a file object.
            Return Value: boolean
            */
            public boolean accept(File f)
            {
               if (f.isDirectory())
               {
                  return true;
               }
               /* Returns the file name */
               String name = f.getName();
               /* Checks whether the file name contains .txt or not */
               if (name.endsWith(".txt") || name.endsWith(".TXT"))
               {
                  return true;
               }
               return false;
            }
            /* getDescription() - This method sets the file description. ]
            Parameter - NA
            Returns Value - String
            */
            public String getDescription()
            {
                return ".txt";
            }
         });
         /* Displays the Open dialog box. */
         jfc.showOpenDialog(this);
         /* Gets the file from the selected location. */
         file = jfc.getSelectedFile();
         if(file==null)
         {
          }
          else
          {
            /* Checks the file object is a File type or Directory type. */
             if(file.isFile())
             {
               /* Gets the absolute path of the file. */
               str = file.getAbsolutePath();
               /* Sets the string of absolute path to the Select File to Encode text field. */
               selectText.setText(str);
               /* Calls the open() method. */
               open();
             }
             else
             {
               /* Displays an Error message dialog box. */
               JOptionPane.showMessageDialog(null, "You have selected an
               invalid file format!", "Error", _
               JOptionPane.WARNING_MESSAGE);
             }
          }
       }
       catch(Exception e)
       {
         System.out.println("Error" + e);
       }
    }
   /* This section is executed when the end user clicks the Encode button. */
    else if(ev.getSource() == encodeButton)
```

```
        {
          if (textArea.getText().equals(""))
          {
             /* Displays an Error message dialog box. */
             JOptionPane.showMessageDialog(this,"You have selected an empty file!",
             "Error", JOptionPane.WARNING_MESSAGE);
          }
          else
          {
             /* Calls the encode() method. */
              encode();
             /* Enables the Decode button. */
             decodeButton.setEnabled(true);
             /* Disables the Encode button. */
             encodeButton.setEnabled(false);
          }
       }
       /* This section is executed when the end user clicks the Decode button. */
       else if(ev.getSource()==decodeButton)
       {
          if (encodeArea.getText().equals(""))
          {
             /* Displays an Error dialog box. */
             JOptionPane.showMessageDialog(this, "The file has no content",
             "Select another File", JOptionPane.WARNING_MESSAGE);
          }
          else
          {
             /* Calls the decode() method. */
             decode();
             /* Enables the Encode button. */
             encodeButton.setEnabled(true);
             /* Disables the Decode button. */
             decodeButton.setEnabled(false);
          }
       }
       /* This section is executed when the end user clicks the Reset button. */
       else if(ev.getSource() == resetButton)
       {
          if(textArea.getText().equals("") && encodeArea.getText().equals("")
          && decodeArea.getText().equals("") &&
          selectText.getText().equals(""))
          {
          }
          else
          {
             /* Clears all the text areas, text field, byte buffer and char buffer. */
             textArea.setText("");
             encodeArea.setText("");
             decodeArea.setText("");
             selectText.setText("");
             buff_in.clear();
             bbuf.clear();
             cbuf.clear();
             /* Disables the Encode button. */
             encodeButton.setEnabled(false);
             /* Disables the Decode button. */
             decodeButton.setEnabled(false);
          }
       }
       /* This section is executed when the end user clicks the Close button. */
       else if(ev.getSource()==closeButton)
       {
          System.exit(0);
       }
    }
    /*
    open() - This method is invoked when the end user clicks the Browse button to open the file.
    Parameter - NA
    Return Value - NA
    */
    public void open()
    {
       try
       {
          /* Initializes the object of the FileInputStream class. */
          fin = new FileInputStream(file);
          /* Gets the file channel from the file input stream. */
          fchan = fin.getChannel();
          /* Gets the size of the file. */
          fsize = fchan.size();
          /* Allocates the size of the Buffer. */
          buff_in = ByteBuffer.allocate((int)fsize);
          /* Reads the buffer from the channel. */
          fchan.read(buff_in);
          /* Rewinds the data in the buffer. */
          buff_in.rewind();
          /* Stores the contents of the buff_in buffer to a string. */
```

```
            file_text = new String(buff_in.array());
            if(file_text.equals(""))
            {
               /* Displays an Error dialog box. */
               JOptionPane.showMessageDialog(null, "The file has no content",
               "Select another File", JOptionPane.WARNING_MESSAGE);
            }
            else
            {
               /* Displays the content stored in the file_text string into Encoded Text text area. */
               textArea.setText(file_text);
               /* Enables the Encode button. */
               encodeButton.setEnabled(true);
            }
            /* Closes the file channel. */
            fchan.close();
            /* Closes the file input stream. */
            fin.close();
         }
         catch(IOException ioe)
         {
            System.err.println("I/O Error on Open");
         }
         catch(UnsupportedCharsetException e)
         {
            System.err.println("The File cannot be encoded");
         }
      }
      /*
      encode() - This method is invoked when the end user clicks the Encode
      button to encode the specified text or a file.
      Parameter - NA
      Return Value - NA
      */
      public void encode()
      {
         try
         {
            /* Initializes the object of the Charset class and sets the encoding scheme. */
            charset = Charset.forName(encScheme.encodingScheme);
            /* Initializes the object of the CharsetEncoder class. */
            encoder = charset.newEncoder();
            /* Gets the text from the Encoded Text text area. */
            String str = textArea.getText();
            /* Encodes the text and stores the encoded text in the byte buffer. */
            bbuf = encoder.encode(CharBuffer.wrap(str));
            buff_out = encoder.encode(CharBuffer.wrap(str));
            /* Creates and initializes the object of the StringBuffer class. */
            StringBuffer sb = new StringBuffer();
            for (int j = 0; buff_out.hasRemaining(); j++)
            {
               /* Converts the bytes into hexadecimal string. */
               int b = buff_out.get();
               int ival = ((int) b) & 0xff;
               char c = (char) ival;
               sb.append(Integer.toHexString (ival));
            }
            String content = sb.toString();
            /* Clears the buff_out byte buffer. */
            buff_out.clear();
            /* Displays the encoded text in the Encoded Text text area. */
            encodeArea.setText(content);
         }
         catch(Exception e)
         {
            System.out.println(e);
         }
      }
      /*
      decode() - This method is invoked when the end user clicks the Decode
      button to decode the encoded text into readable format.
      Parameter - NA
      Return Value - NA
      */
      public void decode()
      {
         try
         {
            /* Initializes the object of the CharsetDecoder class. */
            decoder = charset.newDecoder();
            /* Decodes the encoded text into a readable form and stores it into a
            character buffer. */
            cbuf = decoder.decode(bbuf);
            /* Converts the data stored in cbuf to a string. */
            String s = cbuf.toString();
            /* Displays the decoded text into the Decoded Text text area. */
            decodeArea.setText(s);
            /* Clears the byte and char buffers. */
```

```
            bbuf.clear();
            cbuf.clear();
         }
         catch(Exception e)
         {
            System.out.println(e);
         }
      }
      /*
      main() - This method creates the main window of the user interface and displays it.
      Parameters:
      args[] - Contains any command line arguments passed.
      Return Value: NA
      */
      public static void main(String[] args)
      {
         try
         {
            /* Sets the look and feel of the Encoder/Decoder application to window look and feel. */
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
         }
         catch(Exception e)
         {
            System.out.println("Unknown Look and Feel." + e);
         }
         /* Creates and initializes the object of the EncoderDecoder class. */
         EncoderDecoder ed = new EncoderDecoder();
         /* Displays the main window of the Encoder/Decoder application. */
         ed.show();
      }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the EncoderDecoder class. This class generates the main window of the Encoder/Decoder application, as shown in Figure 8-2:


**Figure 8-2:** The Encoder/Decoder Application User Interface

To open an existing text file, end users can use the Browse button in the user interface. The Text to Encode text area allows end users to specify the text that is to be encoded. The `Encoded Text` text area displays the encoded text whereas the Decoded Text text area displays the decoded text.

The methods defined in the above code are:

- `keyTyped()`: Acts as an event listener and activates an appropriate method based on the text that is to be encoded. When an end user specifies the text in the Text to Encode text area, this method calls the setEnabled() method to enable the Encode button.

- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method based on the button an end user clicks. If the Browse button is clicked, the actionPerformed() method initializes the object of the JFileChooser class and sets the file selection mode to FILES_ONLY. This method then creates and adds a file filter using the JFileChooser object to select only text files. The actionPerformed() method calls the showOpenDialog() method of the JFileChooser class to display the Open dialog box. Next, the actionPerformed() method retrieves the path of the selected file from the file chooser dialog box, displays the path in the Select File to Encode text field, and calls the open() method. If the end user clicks the Reset button, the actionPerformed() method clears the text displayed in the text box and text areas. This method also calls the clear() method of the ByteBuffer class to clear the byte and char buffers. If the end user clicks the Select Encoding Scheme button, the actionPerformed() method calls the setVisible() method of the JFrame class to display the Encoding Schemes window. When the end user clicks the Encode button, the actionPerformed() method calls the encode() method to encode the specified text that is displayed in the Encoded Text text area. If an end user clicks the Decode button, the actionPerformed() method calls the decode() method to decode and display the decoded text in the Decoded Text text area.
- `open()`: Initializes the object of the FileInputStream class and gets the file channel from the file input stream. The open() method gets the size of the file and allocates that size to the byte buffer. The open() method then reads the buffer from the channel and rewinds the buffered data. Next, the open() method stores the contents of the byte buffer to a string and sets it to the Text to Encode text area.
- `encode()`: Initializes the object of the Charset and CharsetEncoder classes, and sets the encoding scheme. The encode() method gets the text from the Text to Encode text area, encodes it, and stores the encoded text in the byte buffer. The open() method then creates and initializes the object of the StringBuffer class, converts the byte to a hexadecimal string, and displays this string in the Encoded Text text area.
- `decode()`: Initializes the object of the CharsetDecoder class. This method decodes the encoded text into readable form and stores it in a character buffer. The decode() method then converts the data stored in the char buffer to a string and displays the string in the Decoded Text text area.

# Creating the Encoding Schemes Dialog Box

The EncodingSchemes.java file allows you to create an Encoding Schemes dialog box. This dialog box provides options to select an encoding scheme to encode the text.

Listing 8-2 shows the contents of the EncodingSchemes.java file:

**Listing 8-2: The EncodingSchemes.java File**

```
/* Imports the java.awt package classes. */
import java.awt.*;
/* Imports the java.nio.charset package classes. */
import java.nio.charset.Charset;
/* Imports the java.util package classes. */
import java.util.SortedMap;
import java.util.Map;
import java.util.Iterator;
import java.util.Collection;
import java.lang.Object;
import java.util.Enumeration;
/* Imports the java.awt.event package classes. */
import java.awt.event.*;
/* Imports the javax.swing package classes. */
import javax.swing.*;
import javax.swing.ButtonModel;
/*
class EncodingSchemes - This class creates a window that displays various encoding schemes.
 Fields:
   title - Creates a title label.
   ok - Creates a OK button.
   cancel - Creates a Cancel button.
   rb - Creates the radio button.
   buttonPanel - Creates a panel for button.
   schemePanel - Creates a panel to display the encoding scheme.
   numberPanel - Creates a panel to display the number of encoding scheme.
   mainPanel - Creates a main panel.
   encodingScheme - Contains the encoding scheme.
   cs- Creates an object of the Charset class.
   c - Creates an object of the Collection class.
   Methods:
   getSchemes() - This method is executed when the EncodingSchemes constructor loads.
   actionPerformed() - This method is invoked when an end user clicks any button of the EncodingSchei
*/
public class EncodingSchemes extends JDialog implements ActionListener
{
   /* Declares the object of the SortedMap class. */
   SortedMap sm;
   /* Declares the object of the JLabel class. */
   JLabel title;
   /* Declares the objects of the JButton class. */
   JButton ok;
   JButton cancel;
   /* Declares the object of the String class. */
   String temp;
   /* Declares the object of the ButtonGroup class. */
   ButtonGroup bg;
   /* Declares the object of the JRadioButton class. */
   JRadioButton rb;
   /* Declares the object of the Charset class. */
   Charset cs;
   /* Declares the object of the Collection class. */
   Collection c;
   /* Declares the objects of the JPanel class. */
   JPanel buttonPanel;
   JPanel schemePanel;
   JPanel numberPanel;
   JPanel mainPanel;
   JScrollPane s_pane;
   /* Declares the object of the Font class. */
   Font f;
   int s;
   public EncoderDecoder ed;
   /* Declares the object of the String class. */
   public static String encodingScheme = "UTF-16";
   /* Defines the default constructor of the EncodingSchemes class. */
   public EncodingSchemes(EncoderDecoder ed)
   {
      this.ed = ed;
      /* Sets the title of the Encoding Schemes window. */
      setTitle("Encoding Schemes");
      /* Sets the size of the Encoding Schemes window. */
      setSize(280, 500);
      /* Sets the re-sizability of the Encoding Schemes window to false. */
```

```
    setResizable(false);
    /* Initializes the object of the ButtonGroup class. */
    bg=new ButtonGroup();
    /* Initializes the object of the Font class. */
    f=new Font("Verdana", Font.BOLD, 12);
    /* Initializes the objects of the JPanel class. */
    buttonPanel=new JPanel();
    schemePanel=new JPanel();
    numberPanel=new JPanel();
    mainPanel=new JPanel();
    /* Initializes the object of the JLabel class. */
    title = new JLabel("Select the Encoding Scheme");
    title.setFont(f);
    /* Initializes and sets the layout of the button panel to grid layout. */
    buttonPanel.setLayout(new GridLayout(1, 2));
    /* Initializes and sets the layout of the number panel to border layout. */
    numberPanel.setLayout(new BorderLayout());
    /* Initializes and sets the layout of the main panel to border layout. */
    mainPanel.setLayout(new BorderLayout());
    /* Initializes the objects of the JButton class. */
    ok=new JButton("OK");
    cancel=new JButton("Cancel");
    /* Adds the action listener events to the Ok and Cancel buttons. */
    ok.addActionListener(this);
    cancel.addActionListener(this);
    /* Calls the getSchemes() method. */
    getSchemes();
    /* Adds various swing components on the panels. */
    numberPanel.add(title, BorderLayout.NORTH);
    buttonPanel.add(ok);
    buttonPanel.add(cancel);
    mainPanel.add(numberPanel, BorderLayout.NORTH);
    mainPanel.add(s_pane, BorderLayout.CENTER);
    mainPanel.add(buttonPanel, BorderLayout.SOUTH);
    /* Adds the main panel to frame. */
    getContentPane().add(mainPanel);
    /*
    addWindowListener - It contains a windowClosing() method.
    windowClosing(): This method is called when the end user clicks the close button of the Window
    Parameter: we- Represents an object of the WindowEvent class.
    Return Value: NA
    */
    addWindowListener(new WindowAdapter()
    {
       public void windowClosing(WindowEvent we)
       {
          dispose();
       }
    });
}
/*
getSchemes() - This method returns the selected encoding scheme.
Parameter - NA
Return Value - NA
*/
public void getSchemes()
{
    try
    {
       /* Retrieves the available character schemes. */
       sm=cs.availableCharsets();
       /* Gets the size of sorted map object. */
       s=sm.size();
       /* Sets the layout of the schemePanel panel to the GridLayout. */
       schemePanel.setLayout(new GridLayout(s, 1));
       /* Retrieves the value from the sorted model object. */
       c=sm.values();
       /* Creates and initializes the object of the Iterator class. */
       Iterator i=c.iterator();
       while(i.hasNext())
       {
          temp = i.next().toString();
          /* Initializes the object of the JRadioButton class. */
          rb = new JRadioButton(temp);
          if(temp.equals("UTF-16"))
          {
             rb.setSelected(true);
          }
          /* Adds the radio button to the button group. */
          bg.add(rb);
          /* Adds the radio button group to the schemePanel panel. */
          schemePanel.add(rb);
          s_pane = new JScrollPane(schemePanel);
          }
       }catch(Exception e)
       {
          System.out.println(e);
       }
```

```
        }
        /*
        actionPerformed() - This method is called when the end user clicks the any button.
        Parameters: ev - Represents an objects of the ActionEvent class that contains the details of t
        Return Value: NA
        */
        public void actionPerformed(ActionEvent ev)
        {
           /* This section is executed when the end user clicks the OK button. */
           if(ev.getSource()==ok)
           {
              /* Creates and initializes the object of the Enumeration class to
              get the elements from the button group. */
              Enumeration enum=bg.getElements();
              while(enum.hasMoreElements())
              {
                 JRadioButton button = (JRadioButton)enum.nextElement();
                 /* Checks the status of a radio button. */
                 if (button.isSelected())
                 {
                    /* Retrieves the text from the radio button. */
                    encodingScheme = button.getText();
                    ed.schemeText.setText(encodingScheme);
                 }
              }
              /* Disposes the window. */
              dispose();
           }
           /* This section is executed when the end user clicks the Cancel button. */
           else if(ev.getSource()==cancel)
           {
              /* Disposes the window. */
              dispose();
           }
     }
}
```

Download this Listing.

In the above code, the constructor of this class creates the Encoding Schemes dialog box for the Encoder/Decoder application, as shown in Figure 8-3:

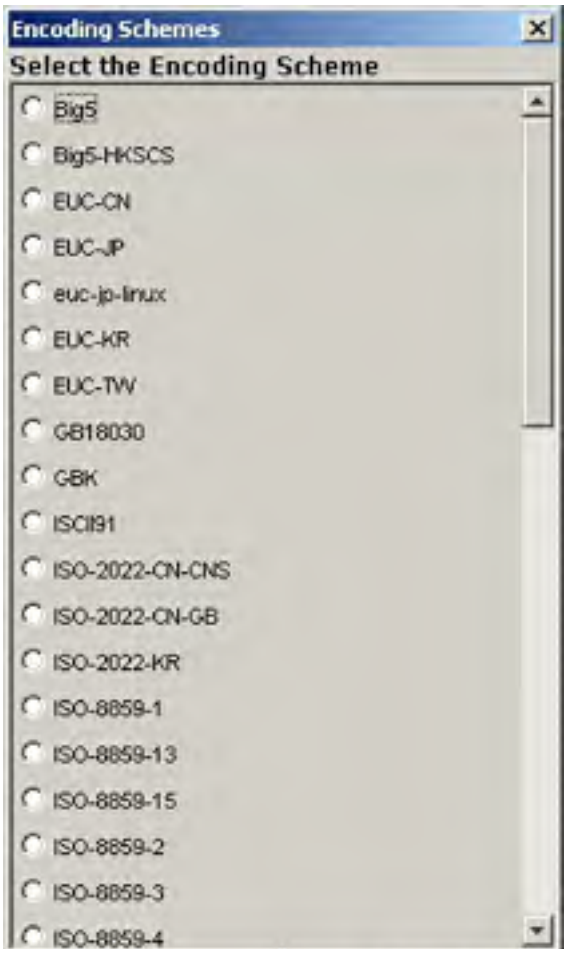


**Figure 8-3:** The Encoding Schemes Dialog Box

The methods defined in the above code are:

- `getSchemes()`: Retrieves the available character schemes and stores it to the sorted map. The getSchemes() method gets the size of the sorted map in the integer variable, s, and sets the layout of the schemePanel panel to sX1 grid layout. This method then retrieves the value from the sorted model object and initializes the object of the JRadioButton class. Finally, this method adds the radio button to the button group and the radio button group to the schemePanel panel.

- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method based on the button the end user clicks. If the OK button is clicked, the actionPerformed() method checks the status of the radio button. This method then gets the text associated with this radio button and stores it to the static variable, encodingScheme. When an end user clicks the Cancel button, the actionPerformed() method calls the dispose() method to close the Encoding Schemes window.



◄ PREVIOUS NEXT ►

## Unit Testing

To test the Encoder/Decoder application:

1. 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;
   ```
2. 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;
   ```
3. Copy the EncoderDecoder.java and EncodingSchemes.java files to a folder on your computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Encoder/Decoder application, specify the following command at the command prompt:
   ```
   java EncoderDecoder
   ```
   The above code opens the main window of the Encoder/Decoder application, as shown in Figure 8-2.
5. Click the Browse button and select the test.txt file from the C drive of the computer.

   Figure 8-4 shows the contents of the test.txt file:


**Figure 8-4:** Displaying test.txt File in the Encoder/Decoder Application

6. Click the Select Encoding Scheme button to open the Encoding Schemes dialog box and select the UTF-16BE encoding scheme.
7. Click the OK button of the Encoding Schemes dialog box.
8. Click the Encode button to encode the test.txt file using the UTF-16BE encoding scheme, as shown in Figure 8-5:


**Figure 8-5:** Displaying Encoded File in the Encoder/Decoder Application

9. Click the Decode button to decode the encoded file, as shown in Figure 8-6:


**Figure 8-6:** Displaying Decoded File in the Encoder/Decoder Application

10. Click the Reset button to reset the Encoder/Decoder application.
11. Specify the text you want to encode in the Text to Encode text area, as shown in Figure 8-7:


**Figure 8-7:** Displaying Text to Encode in the Encoder/Decoder Application

12. Repeat steps 6 through 9 to encode and decode the text specified in the Text to Encode text area. The results are displayed in Figure 8-8:


**Figure 8-8:** Displaying Encoded and Decoded Text

# Index

## A-J

# Index

## N-S

# Index

## T

# List of Figures

## Chapter 6: Creating a Text Editor Application



## Chapter 7: Creating a Network Information Application



## Chapter 8: Creating an Encoder/Decoder Application

# List of Examples

## Chapter 2: Creating a Chat Application



## Chapter 3: Creating a File Download Application



## Chapter 4: Creating a File Search Application



## Chapter 5: Creating a Printer Management Application



## Chapter 6: Creating a Text Editor Application



## Chapter 7: Creating a Network Information Application

# Chapter 8: Creating an Encoder/Decoder Application





PREVIOUS

Contents

Related Links:

InstantCode Series

**Collections:**

BusinessPro

ITPro


**Java InstantCode: Developing Applications Using Java NIO**

SkillSoft Press © 2004

Learn about the New I/O (NIO) API introduced in J2SDK v 1.4 that p
improved performance in the areas of polling, buffer management, c
more.

# 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

This book describes the New I/O (NIO) API introduced in J2SDK v 1.4 that provides new features and improved performance in the areas of polling, buffer management, scalable network and advanced file system I/O, character converters, and regular-expression matching. It also explains how the NIO API supplements the existing I/O facilities. In addition, it describes how to implement these new features to improve the efficiency of Java applications.

## About the Author

Vishal Jayaswal holds a Bachelor's degree in IT Engineering. He is proficient in technologies such as C++, Java, HTML, DHTML, J2EE, EJB, JNDI, JMS, and RMI and has been involved in technical writing for NIIT in the same areas. In addition, he has written books on Data Mining and Java NIO. He has a sound knowledge of Adobe Photoshop and DreamWeaver.

## Credits

I would like to thank Gaurav Bhatla, Reena Roy, and S. Sripriya, for helping me complete the book on time. I also thank the editors and the quality assurance team for their timely help.

# Copyright

Java InstantCode: Developing Applications Using Java NIO


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

information@skillsoft.com

# Chapter 1: Introduction to New Input/Output API

Java2 Platform, Standard Edition (J2SE) provides a New Input/Output (NIO) API that provides features for polling, buffer management, scalable network and advanced I/O file system, character conversion, and regular-expression matching. Development of NIO API was proposed in Java Specification Request (JSR) 51. JSR contains description of the proposal to develop new specifications for the for the Java platform. Using NIO API, you can create applications that can search for regular expressions in a file, read and write data in a file, transfer data over a network, and encode and decode characters. In addition, NIO API provides support for scalable I/O operations for sockets and files, such as file locking and memory mapping.

NIO API consists of the java.nio, java.nio.channels, java.nio.channels.spi, java.nio.charset, and java.nio.charset.spi packages.

This chapter explains the packages provided by NIO API, along with the classes included in each package. This chapter also explains the commonly used methods provided by each class.

## The java.nio Package

The java.nio package contains various classes and methods for buffer management. You can use the java.nio classes to store the content of a file in a buffer in the form of bytes and characters. Buffer is a container to store fixed amount of specific primitive type data, such as byte, character, integer, and float.

> **Note** To learn more about java.nio package, see: http://java.sun.com/j2se/1.4.2/docs/api/java/nio/package-summary.html.

The various attributes of a buffer are:

- `capacity`: Represents the number of elements stored in a buffer.
- `limit`: Represents the index of the first element stored in the buffer. You cannot perform read and write operation on the limit. The limit of a buffer is always positive and greater than the capacity.
- `position`: Represents the valid index position in the buffer for performing read and write operation. The position is always positive and greater than the limit.

Figure 1-1 shows the class diagram for the java.nio package:

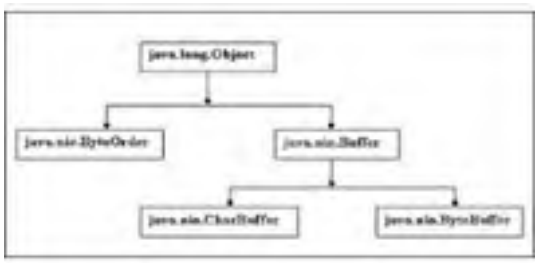

**Figure 1-1:** The java.nio Package

### Buffer

The Buffer class is an abstract class that provides common methods to all its subclasses, such as ByteBuffer, ByteOrder, and CharBuffer. The various methods declared in the Buffer class are:

- `capacity()`: Returns an integer that specifies the storage capacity of the buffer.
- `clear()`: Clears the buffer. This method makes the buffer ready for a new sequence of channel read or put operation. The clear() method also returns a reference to buffer objects.
- `flip()`: Sets the buffer to empty state from a fill state and makes a buffer ready for channel-write or get operation.
- `hasRemaining()`: Returns a Boolean value to indicate whether or not any element exists between the current position and the limit. The method returns true if any element exists in the buffer.
- `isReadOnly()`: Returns a Boolean value that indicates whether or not the buffer is read-only. The method returns true if the buffer is read-only.
- `limit()`: Returns an integer value that indicates the buffer limit. This method throws the IllegalArgumentException exception that occurs when an illegal argument is passed within the limit() method.
- `mark()`: Sets the mark of a buffer at its position. A mark is the index to which the position of a buffer is reset.
- `position()`: Returns an integer that indicates the current position of the buffer.
- `remaining()`: Returns an integer that indicates the number of elements between the current position and limit of the buffer.
- `reset()`: Resets the current position of a buffer to the previously-marked position. This method throws the InvalidMarkException exception that occurs when the mark is not defined in the buffer.
- `rewind()`: Reverses the order of the data in the buffer. The position is set to zero and discards the mark of the

buffer.

### ByteBuffer

The ByteBuffer class represents a byte buffer. This is a subclass of the Buffer class. The most commonly used methods in the ByteBuffer class are:

- `allocate()`: Returns the ByteBuffer class object. The allocate() method allocates a new byte buffer. You need to specify the size of the new byte buffer.
- `array()`: Returns a byte array. The returned array backs up the buffer. If you change the content of the buffer, the content in the returned array is also changed. This method throws two exceptions, ReadOnlyBufferException and UnsupportedOperationException. The ReadOnlyBufferException exception occurs when the put() or compact() method is invoked on a read-only buffer. The UnsupportedOperationException exception occurs when the requested operation is not supported by the Java Virtual Machine (JVM).
- `arrayOffset()`: Returns an integer value that specifies the offset within the backup array of the buffer. This method also throws two exceptions, ReadOnlyBufferException and UnsupportedOperationException.
- `asCharBuffer()`: Returns an object of the CharBuffer class. The asCharBuffer() method creates a view of the byte buffer as a char buffer.
- `asDoubleBuffer()`: Returns an object of the DoubleBuffer class. This method creates a view of the byte buffer as a double buffer.
- `asFloatBuffer()`: Returns an object of the FloatBuffer class. This method creates a view of the byte buffer as a float buffer.
- `asIntBuffer()`: Returns an object of the IntBuffer class. The asIntBuffer() method creates a view of the byte buffer as int buffer.
- `asLongBuffer()`: Returns an object of the LongBuffer class. This method creates a view of the byte buffer as a long buffer.
- `asReadOnlyBuffer()`: Returns an object of the ByteBuffer class. This method creates a new read-only byte buffer that shares the content of the byte buffer with another byte buffer.
- `asShortBuffer()`: Returns an object of the ShortBuffer class. This method creates a view of the byte buffer as a short buffer.
- `compact()`: Returns a ByteBuffer object. The compact() method compacts the buffer by copying the bytes between the current position and limits to the beginning of the buffer. This method throws a ReadOnlyBufferException, if the buffer is read-only.
- `compareTo()`: Compares the object of the ByteBuffer class to another object of another buffer and returns an integer value: positive, negative, or zero. A positive integer indicates that the buffer is greater than the specified buffer. A negative integer indicates that the buffer is less than the specified buffer whereas zero indicates that the buffer and the specified buffer are equal. The compareTo() method throws a ClassCastException if the object passed is not a byte buffer.
- `duplicate()`: Returns an object ByteBuffer class. The duplicate() method creates a new byte buffer and shares the content of the buffer with the newly created byte buffer.
- `getChar()`: Returns a char type value at the current position of the buffer. The getChar() method reads the next two bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by two. The getChar() method throws the BufferUnderflowException exception if there are less than two bytes in the buffer.
- `getDouble()`: Returns a double value at the current position of the buffer. The getDouble() method reads the next eight bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by eight. The getDouble() method throws the BufferUnderflowException exception if there are less than eight bytes in the buffer.
- `getFloat()`: Returns a float type value at the current position of the buffer. The getFloat() method reads the next four bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by four. The getFloat() method throws the BufferUnderflowException exception if there are less than four bytes in the buffer.
- `getInt()`: Returns an int type value at the current position of the buffer. The getInt() method reads the next four bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by four. The getInt() method throws the BufferUnderflowException exception if there are less than four bytes in the buffer.
- `getLong()`: Returns a long type value at the current position of the buffer. The getLong() method reads the next eight bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by eight. The getLong() method throws the BufferUnderflowException exception if there are less than 8 bytes in the buffer.
- `getShort()`: Returns a short type value at the current position of the buffer. The getShort() method reads the next two bytes from the current position of the buffer, converts the bytes into a char value according to the current byte order, and then increments the position by two. The getShort() method throws the BufferUnderflowException exception if there are less than two bytes in the buffer.
- `putChar()`: Returns an object of the ByteBuffer class. The putChar() method writes two bytes at the current

position into the buffer and then increments the position by two. The putChar() method throws the BufferOverflowException exception if there are less than two bytes left in the byte buffer. In addition, the putChar() method throws the ReadOnlyBufferException exception if the buffer is read-only.

- `putDouble()`: Returns an object of the ByteBuffer class. The putDouble() method writes eight bytes that contain the specified char value into the buffer at the current position, and then increments the position by eight. The putDouble() method throws the BufferOverflowException exception if there are less than 8 bytes left in the byte buffer. In addition, the putDouble() method throws the ReadOnlyBufferException exception if the buffer is read-only.
- `putFloat()`: Returns an object of the ByteBuffer class. The putFloat() method writes four bytes that contain the specified char value into the buffer at the current position, and then increments the position by four. The putFloat() method throws the BufferOverflowException exception if there are less than four bytes left in the byte buffer. In addition, the putFloat() method throws a ReadOnlyBufferException, if the buffer is read-only.
- `putInt()`: Returns an object of the ByteBuffer class. The putInt() method writes four bytes that contain the specified char value into the buffer at the current position, and then increments the position by four. The putInt() method throws the BufferOverflowException exception if there are less than four bytes left in the byte buffer. In addition, the putInt() method throws a ReadOnlyBufferException, if the buffer is read-only.

### ByteOrder

The ByteOrder class defines the constants that determine the type of byte order to use when storing or retrieving byte values from a buffer. This class acts as a type-safe enumeration classes. Enumerations help create named values. The ByteOrder class defines two static values that denote byte order:

- `BIG_ENDIAN`: Denotes the BIG_ENDIAN byte order. The bytes of a multibyte value are ordered from the most significant to least significant.
- `LITTLE_ENDIAN`: Denotes the LITTLE_ENDIAN byte order. The bytes of a multibyte value are ordered from the least significant to most significant.

The most commonly used method in the ByteOrder class is the nativeOrder() method. This method returns the native byte order of the hardware on which JVM is running. Using the nativeOrder() method, the performance-sensitive Java code can allocate direct buffers with the same byte order as the hardware. The nativeOrder() method is a static method.

### CharBuffer

The CharBuffer class represents a character buffer. The CharBuffer class provides methods to copy the String type variables into CharBuffer. All these methods return the CharBuffer object. The most commonly used methods in the CharBuffer class are:

- `allocate()`: Creates a buffer object and allocates space to hold data elements. You also need to specify the size of the buffer to hold data elements when you invokes the allocate() method. This method throws two exceptions, ReadOnlyBufferException and UnsupportedOperationException.
- `wrap()`: Wraps a character array into a buffer.
- `slice()`: Creates a new character buffer. The content of the new buffer starts from the current position of the buffer.
- `duplicate()`: Creates a new buffer. The new buffer shares the content of the buffer.
- `asReadOnlyBuffer()`: Creates a new read-only character buffer. The new character buffer shares the content of the buffer.
- `get()`: Reads the specified character from the current position of the buffer, and then increments the position by one. The method throws the BufferUnderFlowException exception if the current position of the buffer is smaller than the limit of the buffer.
- `put()`: Writes the specified character into the current position of the buffer, and then increments the position by one. This method throws two exceptions, BufferOverflowException and ReadOnlyBufferException.
- `hasArray()`: Returns a Boolean value. The hasArray() method indicates whether or not the given buffer is copied by the character array.

# The java.nio.channels Package

The java.nio.channels package defines channels. The channels represent connections to entities that can perform I/O operations. The various entities are files, sockets server socket, and selectors. The entities use the channels to perform I/O operations, such as read and write. In addition, this package helps you create a non-blocking server. A non-blocking server handles multiple client requests without blocking them. A non-blocking server is created using the selector class of NIO. A selector is a selectable channel, which is a special type of channel that can be put in a non-blocking mode. The selectors process all the client requests and send the requests to the server.

**Note** To learn more about java.nio.channels package, see:
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/package-summary.html

Figure 1-2 shows the class diagram for the java.nio.channels package:

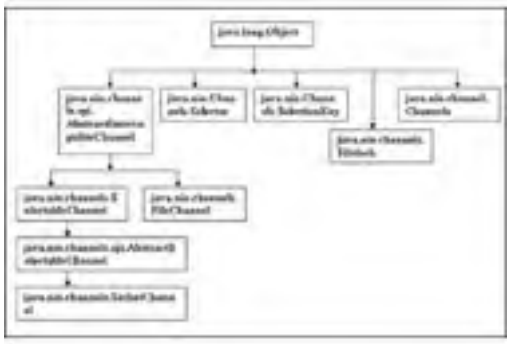

**Figure 1-2:** The java.nio.channels Package

### Channel

The Channel class contains utility methods to work with channels and streams. The most commonly used methods in the Channel class are:

- `newChannel()`: Returns a new readable byte channel value. The newChannel() method constructs a channel that can read bytes from a given stream.
- `newInputStream()`: Returns the InputStream object. The newInputStream() method constructs a stream that can read bytes from a given channel.
- `newReader()`: Returns the Reader object. The newReader() method constructs a reader that decodes the bytes. The reader uses a decoder to decode bytes from a given channel.
- `newWriter()`: Returns the Writer object. The newWriter() method constructs a writer that encodes characters. The writer uses an encoder to encode the characters and writes the resulting bytes to a given channel.

### FileChannel

The FileChannel class provides a channel to read, write, map, and manipulate a file. The FileChannel class defines methods that can read and write bytes from a specific location in a file. The most commonly used methods in the FileChannel class are:

- `force()`: Updates the files that use the channel. The force() method also writes a back up file to the storage device. The force() method throws the ClosedChannelException exception if the channel is closed. In addition, the force() method throws the IOException exception if some other types of input/output errors occur.
- `lock()`: Returns an object of the FileLock class. The lock() method acquires a lock on a specific portion of a file or a complete file that uses the channel.
- `map()`: Returns an object of the MappedByteBuffer class. The map() method maps an area of a file of the channel into the memory. The mapping can be read-only, read/write, or private. The method throws the NonReadableChannelException exception if the source channel is not open to read but the mode is READ_ONLY. The method throws the NonWritableChannelException exception if the channel is not open to write and the mode is READ_WRITE or PRIVATE.
- `read()`: Returns an integer value that indicates the number of bytes read. The read() method reads bytes from the channel into a given buffer. In addition, the read() method throws the NonReadableChannelException exception if the channel is not open to read. The method may throw the ClosedChannelException exception if the channel is closed.
- `size()`: Returns a long value that indicates the size of the file in the channel. The size() method throws the ClosedChannelException exception if the channel is closed. In addition, the size() method throws the IOException exception if some input/output error occurs.
- `transferFrom()`: Returns a long type value. The transferFrom() method transfers bytes into a file of the channel. In addition, the transferFrom() method throws a NonReadableChannelException exception if the source channel is not open to read. The method may throw the NonWritableChannelException exception if the channel is not open for performing write operation.

- `transferTo()`: Returns a long type value. The transferTo() method transfer bytes from a file of the channel to a given writable byte channel. In addition, the transferTo() method throws the NonReadableChannelException exception if the source channel is not open to read. The method may throw the NonWritableChannelException exception if the channel is not open for performing write operation.
- `write()`: Returns an int value that indicates the number of bytes written to the specified file. The bytes are written to the file specified The write() method writes bytes from the channel into a given buffer. In addition, the write() method throws the NonWriteableChannelException exception if the channel is not open to write. The method can also throw the ClosedChannelException exception if the channel is closed.

### FileLock

The FileLock class contains methods that perform file lock operations on a file. This class works as a token that represents a lock on a portion of a file. You can acquire a lock on a file in exclusive or shared mode. The most commonly used methods in the FileLock class are:

- `channel()`: Returns an object of the FileChannel class.
- `isShared()`: Returns a Boolean value that indicates whether or not the type of lock is shared.
- `isValid()`: Returns a Boolean value that indicates whether or not the lock is valid.
- `position()`: Returns a long value that indicates the first byte of the locked region.
- `release()`: Releases a lock from the file.
- `size()`: Returns a long value that indicates the total number of bytes in the locked region.

### Selector

The Selector class represents a multiplexor of the SelectableChannel class object. Moreover, the Selector class manages the information about the registered channels and their respective states. When a channel registers with the selector, the selector is responsible to update the state of a channel. The Selector class uses the service provider classes of the java.nio.channels.spi package to create a new selector. The most commonly used methods of the Selector class are:

- `isOpen()`: Returns a Boolean value that indicates whether or not the selector is open. The isOpen() method returns true if the selector is open.
- `keys()`: Returns an object of the Set interface that indicates the key set of the selector. The Set interface represents a collection that contains no duplicate elements. This method throws the ClosedSelectorException exception. This exception occurs when the selector on which you perform I/O operation is closed.
- `open()`: Returns an object of the Selector class. The open() method opens a selector. In addition, the open() method throws the IOException exception if an input/output error occurs.
- `provider()`: Returns an object of the SelectorProvider class. The SelectorProvider class is a provider that creates the channel.

### SocketChannel

The SocketChannel class represents a selectable channel. The selectable channel is a type of channel that connects a selector to a stream-oriented socket using TCP/IP. The SocketChannel class contains methods to work with the socket channel. The most commonly used methods in the SocketChannel class are:

- `connect()`: Returns a Boolean value that indicates whether or not a connection is established. In addition, the connect() method returns true if a connection exists, otherwise the method returns false. The connect() method throws the ClosedChannelException exception if the channel is closed. In addition, the connect() method throws the AlreadyConnectedException exception if the channel is already connected.
- `finishConnect()`: Returns a Boolean value that indicates whether or not the process to connect a socket channel is complete. In addition, the finishConnect() method returns true if the socket of the channel is connected. The method throws a ClosedChannelException exception if the channel is closed. In addition, the finishConnect() method throws an IOException exception if any input/output error occurs.
- `isConnected()`: Returns a Boolean value that indicates whether or not the network socket of a channel is connected to the server. In addition, the isConnected() method returns true if the network socket of a channel is connected, otherwise the method returns false.
- `open()`: Returns an object of the SocketChannel class. In addition, the connect() method opens a socket channel. This method also throws an IOException exception, if any input/output error occurs.
- `read()`: Returns an int value that indicates the number of bytes read. The `read()` method reads a series of bytes from the channel in a buffer. This method also throws an IOException exception.
- `socket()`: Returns an object of the Socket class. The socket() method retrieves a socket from the associated channel.
- `write()`: Returns an int value that indicates the number of bytes written. The `write()` method writes a series of bytes to the channel from a given buffer. In addition, the write() method throws the NotYetConnectedException exception if the channel is not connected. The write() method throws the IOException exception if any input/output error occurs.

### SelectionKey

The SelectionKey class creates and manages a selection key. A selection key is a token that represents the registration of the SelectableChannel class with the Selector class. This token is created each time the objects of a channel get registered with a selector. The most commonly used methods in the SelectionKey class are:

- `attach()`: Returns an object of the Object class. The `attach()` method attaches an object to the selection key.
- `attachment()`: Returns an object of the Object class. The attachment() method retrieves the object that is currently attached to the selection key.
- `channel()`: Returns an object of the SelectChannel() class that indicates a selector for the selection key.
- `isAcceptable()`: Returns a Boolean value that indicates whether or not the selection key is ready to accept a new socket connection. The isAcceptable() method returns true if the readyOps() method and OP_ACCEPT field are greater than zero.
- `isReadable()`: Returns a Boolean value that indicates whether or not the selection key is ready to read. The isReadable() method returns true if readyOps() method and OP_READ field are greater than zero.
- `isValid()`: Returns a Boolean value that indicates whether or not the selection key is valid. The isValid() method returns true if the selection key is valid.
- `isWritable()`: Returns a Boolean value that indicates whether or not the selection key is ready to write. The isWritable() method returns true if readyOps() method and OP_WRITE field are greater than zero.

The SelectionKey class also defines four static variables:

- `OP_ACCEPT`: Represents the operation-set bit for socket-accept operations.
- `OP_CONNECT`: Represents the operation-set bit for socket-connect operations.
- `OP_READ`: Represents the operation-set bit for read operations.
- `OP_WRITE`: Represents the operation-set bit for write operations.



◄ PREVIOUS NEXT ►

# The java.nio.channels.spi Package

The java.nio.channels.spi package contains various service-provider classes and methods to create a new selector provider class.

**Note** To learn more about java.nio.channels.spi package, see:
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/spi/package-summary.html

The commonly used classes in the java.nio.channels.spi package are:

- `SelectProvider`
- `AbstractSelectot`

### SelectorProvider

The SelectorProvider class defines various methods to create datagram channel, socket channel, and selectable channels. The most commonly used methods in the SelectorProvider class are:

- `openDatagramChannel()`: Returns an object of the DatagramChannel class. The openDatagramChannel() method opens a datagram channel. The datagram channel is a type of channel that connects the selector to a stream-oriented socket using User Datagram Protocol (UDP).
- `openSelectors()`: Returns an object of the AbstractSelector class. The openSelectors() method opens a selector.
- `openSocketChannel()`: Returns an object of the SocketChannel class. The openSocketChannel() method opens a socket channel.
- `provider()`: Returns an object of the SelectorProvider class that contains a default selector provider. The provider() method is a static method.

### AbstractSelector

The AbstractSelector class works as a base implementation class for selectors. The AbstractSelector class defines methods that work with selectors. The most commonly used methods in the AbstractSelector class are:

- `begin()`: Identifies the beginning of the input/output operation.
- `end()`:Identifies the end of the input/output operation.
- `isOpen()`: Returns a Boolean value that indicates whether the selector is open or close. The isOpen() method returns true if the selector is open.
- `provider()`: Returns an object of the SelectorProvider class that creates the channel.
- `register()`: Returns an object of the SelectionKey class. The register() method registers the given channel with the selector.

# The java.nio.charset Package

The java.nio.charset package contains various classes and methods for character set conversion, regular expression matching, and for encoding and decoding data. This package provides the charset, encoder, and decoder classes to convert data between bytes and coded characters. The character data is encoded for transmission over a network or for storage in a file.

A character set is a set of characters, such as alphabets from A to Z, a to z, and special characters. The coded character set is an assignment of numeric value to each character in the character set using the standard encoding scheme. The encoding scheme is a process of mapping the coded character set to a sequence of Octets. A collection of 8 bytes is known as Octet.

The charset are the combination of coded character and encoding schemes. The various standard charsets are US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, and UTF-16.

> **Note** To learn more about java.nio.charset package, see: http://java.sun.com/j2se/1.4.2/docs/api/java/nio/charset/package-summary.html

Figure 1-3 shows the class diagram for the java.nio.charset package:

**Figure 1-3:** The java.nio.charset Package

### Charset

The Charset class defines methods to retrieve various names associated with a character set. In addition, the Charset class defines methods to create encoders and decoders. The most commonly used methods of the Charset class are:

- `aliases()`: Returns an object of the Set interface that indicates the aliases of a charset.
- `availableCharsets()`: Returns an object of the SortedMap interface that contains one entry for each charset supported by the current JVM. The availableCharsets() method develops a sorted map from canonical charset names to charset objects. The map contains one entry for each charset, which the current JVM supports.
- `decode()`: Returns an object of the CharBuffer class. The decode() method decodes bytes in the charset into Unicode characters.
- `displayName()`: Returns a string that represents a readable name.
- `encode()`: Returns an object of the ByteBuffer class. The encode() method encodes Unicode characters into bytes in the charset.
- `forName()`: Returns an object of the Charset class for the standard charset. In addition, the forName() method throws the IllegalCharsetNameException exception if the given charset name is illegal and the UnsupportedCharsetException exception if there is no support for the named charset.
- `hashCode()`: Returns an int value that indicates the calculated hashcode for the charset. The hashcode is an object that contains the hash id and hash state of the hash table.
- `newDecoder()`: Returns an object of the CharsetDecoder class. The newDecoder() method constructs a new decoder for the charset.
- `newEncoder()`: Returns an object of the CharsetEncoder class. The newEncoder() method constructs a new encoder for the charset.

### CharsetDecoder

The CharsetDecoder class defines methods that transform a sequence of bytes into a sequence of 16-bit Unicode characters. The source buffer is a byte buffer and the resultant buffer is a character buffer. The most commonly used methods in the CharsetDecoder class are:

- `charset()`: Returns an object of the Charset class that creates the decoder.
- `decode()`: Returns an object of the CoderResult class. The decode() method decodes the bytes from the source buffer and writes the resultant character in the target buffer. In addition, the decode() method throws the IllegalStateException exception if a decode operation is already in progress.
- `decodeLoop()`: Returns an object of the CoderResult class. The decodeLoop() method decodes one or more bytes into one or more characters.
- `flush()`: Returns an object of the CoderResult class. The flush() method clears the decoder.
- `isAutoDetecting()`: Returns a Boolean value. The isAutoDetecting() method indicates whether the decoder implements an auto-detecting charset.
- `isCharsetDetected()`: Returns a Boolean value that indicates whether or not the decoder has detected a

charset.

- `replacement()`: Returns a string value that indicates the replacement value of the decoder.
- `reset()`: Returns an object of the CharsetDecoder class. The reset() method resets the decoder.

### CharsetEncoder

The CharsetEncoder class defines methods that transform a sequence of 16-bit Unicode characters into a sequence of bytes. The source buffer is a character buffer, and the resultant buffer is a byte buffer. The most commonly used methods in the CharsetDecoder class are:

- `canEncode()`: Returns a Boolean value that indicates whether the encoder can encode the given character.
- `charset()`: Returns an object of the Charset class that creates the encoder.
- `encode()`: Returns an object of the CoderResult class. The encode() method encodes the characters from the source buffer and writes the resultant bytes in the target buffer. In addition, the encode() method throws the IllegalStateException exception if an encode operation is already in progress.
- `encodeLoop()`: Returns an object of the CoderResult class. The encodeLoop() method encodes one or more bytes into one or more characters.
- `flush()`: Returns an object of the CoderResult class. The flush() method flushes the encoder.
- `replacement()`: Returns a string value that indicates the replacement value of the encoder.
- `reset()`: Returns an object of CharsetEncoder class. The reset() method resets the encoder.

### CoderResult

The CoderResult class defines the result state of a coder. A coder consumes bytes or characters from an input buffer, translates them, and writes the resulting characters or bytes to an output buffer. The translation process can be stopped due to various reasons, such as underflow, overflow, malformed-input error, and unmappable-character error. A coder can be an encoder or decoder. The most commonly used methods in the CoderResult class are:

- `isError()`: Returns a string value that describes a coder result.
- `isMalformed()`: Returns a Boolean value that indicates whether the object describes a malformed-input error. The malformed-input error occurs when an input byte sequence is not legal for a specified charset. The isMalformed() method returns true if an object indicates malformed-input error else returns false.
- `isOverFlow()`: Returns a Boolean value that indicates whether the object describes overflow condition. An overflow condition occurs when the buffer that stores the decoded data reaches the buffer limit. The isOverFlow() method returns true if the object indicates overflow, otherwise the method returns false.
- `isUnderFlow()`: Returns a Boolean value that indicates whether the object describes underflow condition. An underflow condition occurs when the buffer that stores the decoded data is empty. The isUnderFlow() method returns true if the object indicates underflow, otherwise the method returns false.
- `isUnmappable()`: Returns a Boolean value that indicates whether the object describes an unmappable-character error. The unmappable-character error occurs when an input character or byte sequence is valid but cannot map to an output byte or character sequence. The isUnmappable() method returns true if the object indicates unmappable-character error, otherwise the method returns false.



PREVIOUS NEXT

# The java.nio.charset.spi Package

The java.nio.charset.spi package contains a service-provider class to create a new charset class. This package contains only one class, CharSetProvider.

**Note** To learn more about java.nio.charset.spi package, see:
http://java.sun.com/j2se/1.4.2/docs/api/java/nio/charset/spi/package-summary.html

### CharSetProvider

The CharSetProvider class is a subclass of the Charset class. This class takes no arguments in the default constructor. You can look up the Charset provider with the help of the loaded context class of the current thread. The methods used in the CharSetProvider class are:

- `charsetForName()`: Returns an object of the Charset class. The charsetForName() method takes charsetName of String type as an input parameter.
- `charsets()`: Returns a collection of objects of the Charset class supported by the provider.

# Chapter 2: Creating a Chat Application

The New Input/Output (NIO) Application Programming Interface (API) supports the java.nio and java.nio.channels packages for handling advance I/O file system and sockets, multiplexing, and non-blocking data transfer between the client and the server. The java.nio package contains ByteBuffer classes that allow you to store data in bytes. The java.nio.channels package provides various classes, such as FileChannel, SelectionKey, Selector, ServerSocketChannel, and SocketChannel. You can use these classes to initialize a socket, connect the socket to the server, and transfer data using this socket.

This chapter explains how to develop a Chat application using the java.nio and java.nio.channels packages.

## Architecture of the Chat Application

Using the Chat application, an end user can send private and broadcast messages. A private message is sent to an individual end user, while a broadcast message is sent to all the end users who are connected to the chat server.

The Chat application contains two folders, Server and Client.

The Server folder contains the following files:

- `ChatServer.java`: Initializes all _ the chat server classes. This is the main class of the chat server.
- `UAServer_Socket.java`: Creates, _ registers, and maps a socket to broadcast a message to all the end users connected in a chat session.
- PRServer_Socket.java: Creates, registers, and maps a socket to send private messages to a specified end user.
- `Msgbroadcast.java`: Broadcasts messages to all the end users connected in a chat session.
- `SocketCallback.java`: Uses sockets to communicate with the chat client.
- `AppendUserList.java`: Stores the names of all the end users connected in a chat session.

The Client folder contains the following files:

- `ChatLogin.java`: Creates the Chat Login window for the chat application.
- `ChatClient.java`: Creates the main window of the chat application. This window contains a text pane and a list box that display the messages and the user list.
- `CClient.java`: Implements the methods that are declared or called in the chat application client. This file connects the chat client to the chat server.
- `Messager.java`: Defines an abstract _ method to read and write the message to the text pane of the chat application client.

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

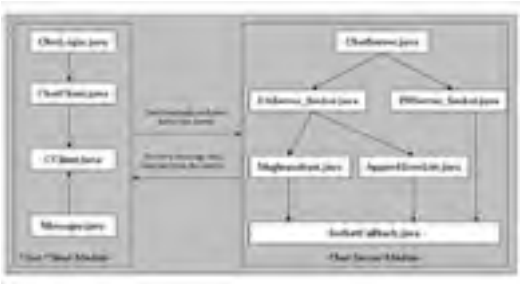

**Figure 2-1:** Architecture of the Chat Application

In the Chat application, there are two separate modules, chat server and chat client. In the chat server, the ChatServer.java file calls the UAServer_Socket.java file and PRServer_Socket.java file to create sockets to send the private and broadcast messages. The UAServer_Socket.java file calls the Msgbroadcast.java and AppendUserList.java files. Next, the Msgbroadcast.java file calls the SocketCallback.java file to broadcast the message to all the chat users. The AppendUserList.java file calls the SocketCallback.java file to add a new user to the user list. The PRServer_Socket.java file again calls the SocketCallback.java file to send the private message to a specific chat user.

In the chat client, the ChatLogin.java file calls the ChatClient.java file to open the chat client window. Next, ChatClient.java calls the CClient.java and Messager.java files. The CClient.java file establishes a connection between the client and the server to send and receive messages. The Messager.java file provides an abstract method, message(), for chat users.

# Creating the Chat Server

To create the chat server, you need to create the following files:

- ChatServer.java file
- UAServer_Socket.java file
- PRServer_Socket.java file
- Msgbroadcast.java file
- AppendUserList.java file
- SocketCallback.java file

### Creating the ChatServer.java File

The ChatServer.java file helps create a Command Line Interface (CLI)-based chat server. The chat server creates sockets to send and receive private and broadcast messages. To allow multiple users to connect to the chat server at the same time, you need to configure the chat server as non blocking.

Listing 2-1 shows the contents of the ChatServer.java file:

**Listing 2-1: The ChatServer.java File**

```
/*Imports java.net package class.*/
import java.net.*;
/Imports java.nio package class.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
/*Imports java.util package class.*/
import java.util.*;
/*Imports java.io package class.*/
import java.io.*;
/*
class ChatServer - Initializes all the chat server classes. This is the main class of the chat _
server.
   Field:
   SendAllPort: Stores the default port number.
   SendPrivatePort: Stores the default port number.
   UserList: Stores the list of users.
   Method:
   start_server(): Creates the objects of socket classes and starts the thread to run _
   the server.
   main(): Creates the object of the ChatServer class and call the Start_Server() _
   method.
*/
public class ChatServer
{
   /*Declares the port numbers.*/
   private int SendAllPort=9999;
   private int SendPrivatePort=8888;
   /*Creates and initializes the object of the ArrayList class.*/
   ArrayList UserList=new ArrayList();
   /*Defines the default constructor.*/
   public ChatServer(){}
   /*
   start_server() - This method is called to start the chat server.
   Parameters:   NA
   Return Value: NA
   */
   public void start_server()
   {
      /*Creates the object of the UAServer_Socket class.*/
      UAServer_Socket ua=new UAServer_Socket(SendAllPort,UserList);
      /*Creates the object of the PRServer_Socket class.*/
      PRServer_Socket p=new PRServer_Socket(SendPrivatePort,UserList);
      /*Initializes and starts a new thread to create the socket for broadcasting.*/
      new Thread(ua).start();
      /*Initializes and starts a new thread to create the socket for private messaging.*/
      new Thread(p).start();
      System.out.println("Server is started. Now waiting for client requests...");
   }
   /*
      main() - This method creates the main window of the user interface and displays it.
      Parameters:
      args[] - Contains any command line arguments passed.
      Return Value: NA
   */
   public static void main(String[] args)
   {
      /*Creates and initializes the objects of the ChatServer class.*/
```

```
      ChatServer cs=new ChatServer();
      /* Calls the Start_Server() method. */
      cs.start_server();
   }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the ChatServer class and calls the start_server() method. The start_server() method creates the object of the UAServer_Socket and PRServer_Socket classes. The start_server() method then initializes and starts a new thread to create the socket to send and receive private and broadcast messages.

### Creating the UAServer_Socket.java File

The UAServer_Socket.java file creates the socket connection to broadcast the messages. The UAServer_Socket class initializes and registers the broadcast socket to the chat server.

Listing 2-2 shows the contents of the UAServer_Socket.java file:

**Listing 2-2: The UAServer_Socket.java File**

```
/*Imports the java.net package class.*/
import java.net.*;
/*Imports the java.net package class.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
/*Imports the java.util package class.*/
import java.util.*;
/*Imports the java.io package class.*/
import java.io.*;
/*
Class UAServer_Socket - Creates the socket, registers and maps the socket to broadcast the _
message to all the end users connected in a chat session.
   Fields:
      Port - Contains the port number.
      srvr_sckt_chnel - Creates the server socket channel.
      sckt_manager - Manages the various sockets.
      selkey - Selects the appropriate socket.
      tdata - Stores the array of bytes.
      broadcastMessage - Stores the messages.
   Method:
      init_socket() - Initializes the private messaging socket.
      register_server() - Registers the server.
      accept_connection() - Accepts the connections from the server.
      read_Message() - Reads the message from the server socket.
      closeRemoteChannel() - Closes all the remote channels.
      write_Message() - Writes the messages to the server socket.
      got_connection() - Retrieves the connection.
*/
public class UAServer_Socket implements Runnable
{
   /*Declares the port for broadcasting.*/
   int port=9999;
   /*Declares the object of the ServerSocketChannel class.*/
   ServerSocketChannel srvr_sckt_chnel=null;
   /*Declares the object of the Selector class.*/
   Selector sckt_manager=null;
   /*Declares the object of the Iterator class.*/
   Iterator searcher=null;
   /*Declares the object of the SelectionKey class.*/
   SelectionKey selkey=null;
   /*Declares the objects of the ArrayList class.*/
   ArrayList broadcastMessage=new ArrayList();
   /*Declares the object of the ByteBuffer class and allocate the size to the byte buffer.*/
   ByteBuffer buffer=ByteBuffer.allocateDirect(1024);
   int buflen=0;
   byte[] tdata=null;
   int conId=0;
   int kichek =0;
   /*
   Defines the constructor that provides the port number as input.
   */
   public UAServer_Socket(int port)
   {
      this.port=port;
   }
   /* Defines the constructor that provides the port number and user list as input. */
   public UAServer_Socket(int port, ArrayList conlist)
   {
      this.port=port;
      broadcastMessage=conlist;
   }
   /* Defines the constructor that provides the user list as input. */
   public UAServer_Socket(ArrayList conlist)
```

```
{
   broadcastMessage=conlist;
}
/*
init_socket() - This method initializes the chat server.
   Parameter: NA
   Return Value: NA
*/
public void init_socket() throws Exception
{
   /*Opens the selector*/
   sckt_manager=SelectorProvider.provider().openSelector();
   /*Opens the server socket channel.*/
   srvr_sckt_chnel=ServerSocketChannel.open();
   /*Configures the server as NON Blocking.*/
   srvr_sckt_chnel.configureBlocking(false);
   /*Creates and initialize the object of the InetSocketAddress class.*/
   InetSocketAddress fulnetaddr=new InetSocketAddress(port);
   /*Binds the server socket.*/
   srvr_sckt_chnel.socket().bind(fulnetaddr);
   /*Calls the register()method.*/
   register_server(srvr_sckt_chnel,SelectionKey.OP_ACCEPT);
}
/*
   register_server() - This method registers the chat server.
   Parameter:
       Ssc: Represents the object of the ServerSocketChannel class.
       selectionkey_ops: Integer type that determines the selection option.
    Return Value: NA
*/
public void register_server(ServerSocketChannel ssc,int selectionkey_ops)throws Exception
{
   ssc.register(sckt_manager,selectionkey_ops);
}
/*
run() - This method calls the methods to initialize the socket and accept the connection of _
the sockets.
   Parameters: NA
   Return Value: NA
*/
public void run()
{
   try
   {
      /*Calls the initialization method.*/
      init_socket();
      /*Calls the method to accept the connection.*/
      accept_connection();
   }
   catch(Exception ej)
   {
      ej.printStackTrace();
   }
}
/*
accept_connection() - This method accepts the connection from the server.
Parameter: NA
Return Value: NA
*/
public void accept_connection()throws Exception
{
   while (true)
      {
         /*Calls the select() method of the Selector class.*/
         kichek=sckt_manager.select();
         if(kichek>=0)
         {
            /*
            Initializes the object of the Iterator class by calling the selectedKeys() methodof t
            */
            searcher=sckt_manager.selectedKeys().iterator();
            while(searcher.hasNext())
            {
               /*Gets the next element.*/
               selkey=(SelectionKey)searcher.next();
               /*Removes the item from the searcher.*/
               searcher.remove();
               /*When the message provider gets new connection from user, this section is _
               executed.*/
               if(selkey.isAcceptable())
               {
                  /*Creates and initializes the object of the ServerSocketChannel class to _
                  get the channel withthe help of the channel() method.*/
                  ServerSocketChannel server=(ServerSocketChannel)selkey.channel();
                  /* Calls the got_connection() method.*/
                  got_connection(server);
                  System.out.println("connection establish");
            }
```

```
/* When the selector provider gets the stream for reading by the server from end user, this _
section is executed.*/
else if(selkey.isReadable())
            {
               /*Calls the read_Message() method.*/
               read_Message((SocketCallback)selkey.attachment());
               }

/*When the selector Provider gets the stream for writing to user, this section is executed.*/
else if(selkey.isWritable())
               {
                  /*Creates the object of the SocketCallback class and call the attachment() _
                  method of the SelectionKey class.*/
                  SocketCallback  Vsc=(SocketCallback)selkey.attachment();
                  String m_essage="";
                  if (Vsc.getString().length()>0)
                  m_essage =Vsc.getString();
                  if(Vsc.get_broad_msg().length()>0)
                  m_essage +=Vsc.get_broad_msg();
                  /* Calls the write_Message(). */
                  write_Message(Vsc,m_essage);
                  }
                  }
               }
            else
            break;
                     }
               }
   /*
   read_Message() - This method reads the messages from the server socket.
      Parameter:
       sc: Represents the object of the SocketCallback class.
       Return Value:
       String: Returns a string value.
   */
   public String read_Message(SocketCallback sc)
   {
      try
      {
         if (sc.isValidUser())
         {
               /* Clears the buffer. */
            buffer.clear();
/*Reads the bytes from the channel. */
int nbyte=sc.getChannel().read(buffer);
            if(nbyte==-1)
            {
               /*Closes the channel. */
               closeRemoteChannel(sc);
               sc.getChannel().close();
               return "";
            }
            if (nbyte>0)
            {
               /*Flips the buffer.*/
               buffer.flip();
               /*Initializes the object of the Byte class.*/
               tdata=new byte[nbyte];
               /*Retrieves the data from the buffer.*/
               buffer.get(tdata,0,nbyte);
               /*Initializes the object of the Inner_checker class.*/
               String s=new String(tdata);
               if(tdata[0]==31)
               {
                  /*Initializes the object of the AppendUserList class.*/
                  new AppendUserList(sc,broadcastMessage);
                  return "";
               }
               else
               {
                  if (s.length()>0)
                  {
                  sc.addString(s);
                  /*Initializes the object of the Msgbroadcast class.*/
                  new Msgbroadcast(broadcastMessage,s,sc.getUserId());
                  return s;
                  }
               }
            }
         }
         else
         {
            /*Clears the buffer.*/
            buffer.clear();

            /*Reads the bytes from the channel.*/
            int
            nbyte=sc.getChannel().read(buffer);
```

```
            if(nbyte==-1)
            {
               /*Closes the channel.*/
               closeRemoteChannel(sc);
               sc.getChannel().close();
               return "";
            }
            if (nbyte>0)
              {
               /*Flips the buffer.*/
               buffer.flip();
               /*Initializes the object of the Byte class.*/
                           tdata=new byte[nbyte];
               /*Retrieves the data from the buffer.*/
               buffer.get(tdata,0,nbyte);
               /*Initializes the object of the Inner_checker class.*/
               new UAinnerChecker(sc,tdata,nbyte);
                           }
                     }
            }
      catch(IOException ex)
      {
         try
         {
            /*Calls the closeRemoteChannel() method.*/
                        closeRemoteChannel(sc);
                        /* Closes the channel. */
                        sc.getChannel().close();
                     }
                     catch(IOException ec)
                     {
                        ec.printStackTrace();
                     }
                  }
                  /*Returns a null string.*/
      return "";
      }
      /*
         closeRemoteChannel() - This method close the remote channel.
            Parameter:
            sCb: Represents the object of the SocketCallback class.
            Return Value: NA
   */
      public void closeRemoteChannel(SocketCallback sCb)
   {
      int i = broadcastMessage.indexOf(sCb);
      if (i >= 0)
      {
         /*Removes the broadcast message from the list.*/
         broadcastMessage.remove(i);
      }
   }
   /*
      write_Message() - This method writes the messages to the server socket.
         Parameter:
         sc: Represents the object of the SocketCallback class.
         msg: Stores the message.
         Return Value: NA
   */
   public void write_Message(SocketCallback sc,String msg)throws Exception
   {
      if (msg.length()>0)
      {
         /*Calls the doBroadcast() method.*/
         sc.doBroadcast();
         /*Creates the object of the ByteBuffer class. Next, wraps the bytes that are retrieved
         from the message.*/
         ByteBuffer b=ByteBuffer.wrap(msg.getBytes());
         /*Writes the byte buffer to the server socket.*/
         sc.getChannel().write(b);
         }
      }
   /*
   got_connection() - This method gets the connection from the server.
      Parameter:
      server: Represents the object of the ServerSocketChannel class.
      Return Value: NA
   */
   private void got_connection(ServerSocketChannel servr)throws Exception
   {
      /*Initializes the object of the SocketChannel class to accept the connection.*/
      SocketChannel sc=servr.accept();
      /*Sets the server to Non Blocking server.*/
      sc.configureBlocking(false);
      /*Registers the server socket with the selector.*/
      SelectionKey skey=sc.register(sckt_manager,SelectionKey.OP_READ|SelectionKey.OP_WRITE);
      /*Creates the object of the SocketCallback class.*/
      SocketCallback callbk=new SocketCallback(sc);
```

```
   skey.attach(callbk);
   }
/*
   finalize() - This method is invoked when message is read or sent.
   Parameter: NA
   Return Value: NA
*/
public void finalize() throws IOException
{
   /*Closes the server socket channel.*/
            srvr_sckt_chnel.close();
            /* Closes the selector. */
            sckt_manager.close();
}

/*
Inner Class: UAinnerChecker - This class checks the user validity.
   Methods:
   run(): Runs the started threads.
   Chk_Validity(): Checks the user validity.
   Chk_Valid_User(): Checks user validation.
*/
class UAinnerChecker extends Thread
{
   /* eclares the objects of the SocketCallback class.*/
   SocketCallback u=null;
   byte[] b=null;
   int len=0;
   /*Defines the constructor that takes the SocketCallback class object and integer type
   value as parameters.*/
   UAinnerChecker(SocketCallback call,byte[] b,int len)
   {
      this.u=call;
      this.b=b;
      this.len=len;
      start();
   }
   /*
      run() - This method is called when the thread is started to call the thread methods.
         Parameters:   NA
         Return Value: NA
   */
   public void run()
   {
      /*Calls the Chk_Validity() method.*/
      Chk_Validity();
   }
   /*
   Chk_Validity() - This method check the user validity.
      Parameter: NA
      Return Value: NA
   */
   private void Chk_Validity()
   {
      /*Declares the objects of the String class.*/
      String u_id="";
      String pwd="";
      String info="";
      /* Declares the FLAGS. */
      boolean id_flag=true;
      boolean pwd_flag=true;
      try
      {
         for (int x=0;x<len;x++ )
         {
            if(id_flag)
            {
               /*Checks the user name separator "28" in the string.*/
               if (b[x]==28)
               {
               id_flag=false;
               }
               else
               u_id+=(char)b[x];
            }
            else if(pwd_flag)
            {
               /*Checks the password separator "29" in the string.*/
               if (b[x]==29)
               {
               pwd_flag=false;
               }
               else
               pwd+=(char)b[x];
           }
           else
           info+=(char)b[x];
         }
```

```
/*Checks the chat user is valid or not.*/
if(Chk_Valid_User(u_id, pwd))
{
   /*Calls the setUserId() method of the SocketCallback class.*/
   u.setUserId(u_id);
   /*Calls the setPwd() method of the SocketCallback class.*/
   u.setPwd(pwd);
   /*Calls the setUinfo() method of the SocketCallback class.*/
   u.setUinfo(info);
   /*Calls the ValidUser() method of the SocketCallback class.*/
   u.ValidUser(true);
   /*Increments the counter by one. */
   conId++;
   /*Calls the setConid() method of the SocketCallback class.*/
   u.setConid(conId);
   broadcastMessage.add(u);
}
else
{
   /*Closes the channel.*/
   u.getChannel().close();
}
}
catch(Exception e3)
         {
            e3.printStackTrace();
         }
      }
      /*
         Chk_Valid_User() - This method checks the user is valid or not.
            Parameter:
            Uid: Stores the user name.
            pass: Stores the password.
            Return Value: NA
      */
public boolean Chk_Valid_User(String Uid,String pass)
      {
           return true;
        }
   }
}
```

Download this Listing.

In the above code, the UAServer_Socket class creates a socket on port 9999 that allows end users to send and receive broadcast messages and an updated user list. The methods defined in this code are:

- `run()`: Calls the init_socket() method to initialize the socket and calls the accept_connection() method to accept the connection from the client.
- `init_socket()`: Opens the selector using the openSelector() method of the Selector class and the server socket channel using the open() method of the ServerSocketChannel class. This method then configures the server as non blocking using the configureBlocking() method of the ServerSocketChannel class. The init_socket() method creates the object of the InetSocketAddress class at port 9999 and binds the socket to the channel at that server IP. Finally, this method calls the register() method of the ServerSocketChannel to register the server.
- `accept_connection()`: Calls the select() method of the Selector class when the connection is established. This method then initializes the object of the Iterator class by calling the selectedKeys() method of the Selector class. If the message provider gets a new connection from the end user, the accept_connection() method creates and initializes the object of the ServerSocketChannel class to retrieve the channel using the getChannel() method. Next, the accept_connection() method calls the got_connection() method to get the connection. If the selector provider gets a stream to read the message from the chat user, the accept_connection() method calls the read_Message() method to read the message. If the selector provider gets the stream to write a message to the chat user, the accept_connection() method creates the object of the SocketCallback class and calls the attachment() method of the SelectionKey class. Finally, the accept_connection() method calls the write_Message() to write the message.
- `got_connection()`: Initializes the object of the SocketChannel class to accept the connection and sets the server to non blocking using the configureBlocking() method. The got_connection() method then registers the server socket with the selector, creates the object of the SocketCallback class, and calls the attach() method of the SelectionKey class.
- `read_Message()`: Checks end user validity using the isValidUser() method of the SocketCallback class. If the end user is valid, the read_Message() method clears the buffer and reads the bytes from the channel to the buffer. The read_Message() method then retrieves the data from the buffer and separates the user name and broadcast message from the incoming data. If the data received from the server is an instance of the user name, the read_Message() method initializes the object of the AppendUserList class to add the end user to the user list. If the data received from the server is an instance of a message, the read_Message() method initializes the object of the Msgbroadcast class. If the end user is not valid, the read_Message() method clears the buffer, calls the closeRemoteChannel() method, and initializes the object of the UAinnerChecker class.
- `write_Message()`: Calls the doBroadcast() method of the SocketCallback class. The write_Message() method then creates the object of the ByteBuffer class and wraps the message bytes. Finally, the write_Message() method writes the byte buffer to the server socket.
- closeRemoteChannel(): Removes all the messages stored in the broadcastMessage array list.

- `finalize()`: Calls the close() method to close the server socket channel and selector.

The UAServer_Socket class consists of an inner class, UainnerChecker, which allows you to validate the user name and password. The methods defined in this inner class are:

- `run()`: Calls the Chk_Validity() method. This method is invoked when the constructor of the UAinnerChecker class is executed.
- `Chk_Validity()`: Sets the id_flag and pwd_flag for the user id and password and checks the end user validity using the Chk_Valid_User() method. After validation, the Chk_Validity() method calls the setConid() method of the SocketCallback class.
- `Chk_Valid_User()`: Returns true if the user name and password you specify are valid.

### Creating the PRServer_Socket.java File

The PRServer_Socket.java file creates the socket connection to send and receive private messages. The PRServer_Socket class initializes and registers the socket to the chat server.

Listing 2-3 shows the contents of the PRServer_Socket.java file:

**Listing 2-3: The PRServer_Socket.java File**

```
/*Imports the java.net package class.*/
import java.net.*;
/*Imports the java.net package class.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
/*Imports the java.util package class.*/
import java.util.*;
/*Imports the java.io package class.*/
import java.io.*;
/*
Class PRServer_Socket - Creates the socket, registers and maps the socket to send private _
messages to the specified end user.
      Fields:
      port: Contains the port number.
      srvr_sckt_chnel: Creates the server socket channel.
      sckt_manager: Manages the various sockets.
      selkey: Selects the appropriate socket.
      tdata: Stores the array of the bytes.
      userName: Stores the name of the private message user.
      privateMessage - Stores the messages.
      Methods:
      init_socket(): Initializes the private messaging socket.
      register_server(): Registers the server.
      run(): Runs the started threads.
            accept_connection() - Accepts the connections from the server.
            read_Message() - Reads the message from the server socket.
            closeRemoteChannel() - Closes all the remote channels.
            write_Message() - Writes the messages to the server socket.
            got_connection() - Retrieves the connection.
            finalize() - Finalizes the server connection.
*/
public class PRServer_Socket implements Runnable
{
   /*Declares the port for private messaging.*/
   int port=8888;
   /*Declares the object of the ServerSocketChannel class.*/
   ServerSocketChannel srvr_sckt_chnel=null;
   /*Declares the object of the Selector class.*/
   Selector sckt_manager=null;
   /*Declares the object of the Iterator class.*/
   Iterator searcher=null;
   /*Declares the object of the SelectionKey class.*/
   SelectionKey selkey=null;
   /*Declares the object of the ByteBuffer class and allocate the size to byte buffer.*/
   ByteBuffer buffer=ByteBuffer.allocateDirect(1024);
   /*Declares the objects of the ArrayList class.*/
   ArrayList userName=new ArrayList();
   ArrayList privateMessage = new ArrayList();
   int conId=0;
   int buflen=0;
   byte[] tdata=null;
   int kichek =0;
   /* Defines the constructor that provides the port number as input. */
   public PRServer_Socket(int port)
   {
      this.port=port;
   }
   /*Defines the constructor that provides the port number and user list as input.*/
   public PRServer_Socket(int port, ArrayList conlist)
   {
      this.port=port;
      userName=conlist;
   }
```

```
   /*Defines the constructor that provides the user list as input.*/
   public PRServer_Socket(ArrayList conlist)
   {
      userName=conlist;
   }
   /*
   init_socket() - This method initializes the chat server.
   Parameter: NA
   Return Value: NA
   */
   public void init_socket() throws Exception
   {
      /*Opens the selector.*/
      sckt_manager=SelectorProvider.provider().openSelector();
      /*Opens the server socket channel.*/
      srvr_sckt_chnel=ServerSocketChannel.open();
      /* Configures the server as NON Blocking.*/
      srvr_sckt_chnel.configureBlocking(false);
      /*Creates and initializes the object of the InetSocketAddress class.*/
      InetSocketAddress fulnetaddr=new InetSocketAddress(port);
      /*Binds the server socket.*/
      srvr_sckt_chnel.socket().bind(fulnetaddr);
      /*Calls the register()method.*/
      register_server(srvr_sckt_chnel,SelectionKey.OP_ACCEPT);
   }
   /*
      register_server() - This method register the chat server.
         Parameter:
         ssc: Represents the object of the ServerSocketChannel class.
         selectionkey_ops: Integer type that determines the selection option.
         Return Value: NA
   */
public void register_server(ServerSocketChannel ssc, int selectionkey_ops)throws Exception
   {
      /*Calls the register() method of the ServerSocketChannel class*/
      ssc.register(sckt_manager,selectionkey_ops);
   }
   /*
      run() - This method is called when the thread is started.
         Parameters:   NA
         Return Value: NA
   */
   public void run()
   {
      try
      {
         /*Calls the initialization method.*/
         init_socket();
         /*Calls the method to accept the connection.*/
         accept_connection();
      }
      catch(Exception ej)
      {
         ej.printStackTrace();
      }
   }
   /*
      accept_connection() - This method accepts the connection from the server.
         Parameter: NA
         Return Value: NA
   */
   public void accept_connection()throws Exception
   {
      while (true)
        {
            /*Calls the select() method of the Selector class.*/
         kichek = sckt_manager.select();
         if(kichek >= 0)
         {
            /*Initializes the object of the Iterator class by calling the selectedKeys() method
            of the Selector class.*/
            searcher = sckt_manager.selectedKeys().iterator();
            while(searcher.hasNext())
            {
               /*Gets the next element.*/
               selkey = (SelectionKey)searcher.next();
               /*Removes the item from the searcher.*/
               searcher.remove();
               /*When the message provider gets new connection from the end user, this section
               is executed.*/
               if(selkey.isAcceptable())
               {
                  /*Creates and initializes the object of the ServerSocketChannel class to get _
                  the channel using the channel() method.*/
                  ServerSocketChannel server=(ServerSocketChannel)selkey.channel();
                  /*Calls the got_connection() method.*/
                  got_connection(server);
                  System.out.println("connection establish");
```

```
                }
/*When the selector provider gets the stream for reading by the server from the end user, this _
section is executed.*/
else if(selkey.isReadable())
                {
                   /*Calls the read_Message() method*/
                   read_Message((SocketCallback)selkey.attachment());
                }
/* When selector Provider gets stream for writing to the end user, then this section is executed. */
else if(selkey.isWritable())
                {
                   /* Creates the object of the SocketCallback class and calls the attachment() _
                   method of the SelectionKey class. */
                   SocketCallback  vsc =(SocketCallback)selkey.attachment();
                   String m_essage="";
                   if (vsc.getString().length()>0)
                   m_essage =vsc.getString();
                   if(vsc.get_broad_msg().length()>0)
                   m_essage +=vsc.get_broad_msg();
                   /* Calls the write_Message(). */
                   write_Message(vsc,m_essage);
                }
             }
          }
          else
          break;
       }
    }
    /*
read_Message() - This method reads the messages from the server socket.
       Parameter:
sc: Represents the object of the SocketCallback class
       Return Value:
       String: Returns a string value.
    */
    public String read_Message(SocketCallback sc)
    {
       try
       {
          /*Clears the buffer.*/
          buffer.clear();
/*Reads the bytes from the channel.*/
int nbyte=sc.getChannel().read(buffer);
          if(nbyte==-1)
          {
             /*Closes the channel.*/
             sc.getChannel().close();
             return "";
          }
          if (nbyte>0)
          {
             /* Flips the buffer. */
             buffer.flip();
             /*Initializes the object of the Byte class.*/
             tdata=new byte[nbyte];
             /*Retrieves the data from the buffer. */
             buffer.get(tdata,0,nbyte);
             /*Initializes the object of the Inner_checker class.*/
             new Inner_checker(sc,tdata,tdata.length);
          }
        }
       catch(IOException ex)
       {
          try
          {
             /*Calls the closeRemoteChannel() method.*/
             closeRemoteChannel(sc);
             /*Closes the channel.*/
             sc.getChannel().close();
          }
          catch(IOException ec)
          {
             ec.printStackTrace();
          }
       }
       /*Returns a null string.*/
       return "";
    }
    /*
       closeRemoteChannel() - This method close the remote channel
          Parameter:
       sCb: Represents the object of the SocketCallback class.
          Return Value: NA
    */
       public void closeRemoteChannel(SocketCallback sCb)
          {
             int i = userName.indexOf(sCb);
             if (i >= 0)
```

```
            {
            /*Removes the user name from the list.*/
                     userName.remove(i);
                  }
                  int j = privateMessage.indexOf(sCb);
                  if (j >= 0)
                  {
            /*Removes the private message from the user list.*/
                     privateMessage.remove(j);
      }
   }
   /*
write_Message() - This method write the messages to the server socket
      Parameter:
sc: Represents the object of the SocketCallback class.
      msg: Stores the message.
      Return Value: NA
   */
public void write_Message(SocketCallback sc,String msg)throws Exception
   {
      if (msg.length()>0)
      {
         /*Calls the doBroadcast() method.*/
         sc.doBroadcast();
         /*Creates the object of the ByteBuffer class. Next, wraps the bytes that are retrieved
         from the message. */
         ByteBuffer b=ByteBuffer.wrap(msg.getBytes());
         /*Writes the byte buffer to the server socket.*/
         sc.getChannel().write(b);
      }
   }
   /*
got_connection() - This method gets the connection from the server.
      Parameter:
server: Represents the object of the ServerSocketChannel class.
      Return Value: NA
   */
private void got_connection(ServerSocketChannel servr)throws Exception
   {
      /*Initializes the object of the SocketChannel class to accept the connection.*/
      SocketChannel sc=servr.accept();
      /*Sets the server to Non Blocking server.*/
      sc.configureBlocking(false);
      /Registers the server socket with the selector.*/
      SelectionKey skey=sc.register(sckt_manager,SelectionKey.OP_READ|SelectionKey.OP_WRITE);
      /*Creates the object of the SocketCallback class.*/
      SocketCallback callbk=new SocketCallback(sc);
      privateMessage.add(callbk);
      skey.attach(callbk);
   }
   /*
finalize() - This method is invoked when message is read or sent.
      Parameter: NA
      Return Value: NA
   */
   public void finalize() throws IOException
   {
      /*Closes the server socket channel.*/
      srvr_sckt_chnel.close();
      /*Closes the selector.*/
      sckt_manager.close();
   }
   /*
      Inner Class: Inner_checker - This class checks the end user validity.
            Methods:
            run(): Runs the started threads.
            readName(): Reads the user name.
      storeMessage_on_Database(): Stores the messages.
   */
   class Inner_checker extends Thread
   {
      /*Declares the objects of the SocketCallback class.*/
      SocketCallback to_m=null;
      SocketCallback from=null;
      byte ib[]=null;
      int blen=0;
      /*Defines the constructor that takes the SocketCallback class object and integer type _
      value as parameters.*/
      public Inner_checker(SocketCallback isc,byte[] msgbyte,int mlen)
            {
               from=isc;
               ib=msgbyte;
               blen=mlen;
               /*Calls the start() method.*/
               start();
            }
      /*
         run() - This method is called when the thread is started.
```

```
            Parameters:   NA
            Return Value: NA
      */
      public void run()
      {
         boolean V_U_chk=false;
         /*Initializes the object of the Iterator class that gets the user name.*/
         Iterator chkval=userName.iterator();
         /*Gets the user id.*/
         String fromName = "" + from.getUserId();
         String[] s=null;
         if(fromName.length()<1)
         {
            /*Calls the readName() method.*/
            s=readName(ib,blen,1);
            return;
         }
         else
         {
            /*Calls the readName() method.*/
            s=readName(ib,blen,0);
            /*Gets the user id.*/
            fromName=from.getUserId();
         }
           while(chkval.hasNext())
         {
            to_m=(SocketCallback)chkval.next();
            if(fromName.equals(to_m.getUserId()))
            {
                  V_U_chk=true;
                  break;
            }
         }
            if(!V_U_chk)
         {
            try
            {
            from.getChannel().close();
            int iobj = privateMessage.indexOf(from);
            if (iobj >= 0)
               {
                  /*Removes private message from the privateMessage array list.*/
                  privateMessage.remove(iobj);
               }
            return;
            }
            catch(Exception ec)
            {
               ec.printStackTrace();
            }
         }
/*Initiates the object of the Iterator class to iterate the private message.*/
Iterator n=privateMessage.iterator();
         SocketCallback msgTo=null;
         while(n.hasNext())
         {
            msgTo=(SocketCallback)n.next();
            if(msgTo.getUserId().equals(s[0]))
            {
               /*Calls the append_broad_msg() message of the SocketCallback class.*/
               msgTo.append_broad_msg(from.getUserId()+ ":"+s[1]);
               /*Calls the storeMessage_on_Database() method.*/
               storeMessage_on_Database(s[1],s[0],fromName);
               break;
            }
         }
      }
      /*
         readName() - This method reads the name of the user.
            Parameter:
            b: Represents a byte array.
            len: Contains the length.
            chk: Represents the chk integer.
            Return Value: NA
      */
private String[] readName(byte[] b,int len,int chk)
      {
         String tmp="";
         int sindex=0;
         byte t[]=null;
         String tmpStr[]=new String[2];
         String usrNAME="";
         for(int x=0;x<len;x++)
         {
            /*Checks for read name separator.*/
            if(b[x]==3)
            {
               sindex=x;
```

```
                break;
             }
             else
             tmp+=(char)b[x];
          }
                if(chk==1)
          {
             for(int x=sindex+1;x<len;x++)
             {
                if(b[x]==3)
                {
                   sindex=x;
                   break;
                }
                else
/*Reads from the end user.*/
                usrNAME+=(char)b[x];
             }
             /*Sets user id.*/
             from.setUserId(usrNAME);
          }
             int tmpval=len-sindex;
             t=new byte[tmpval];
          /*Copies the array.*/
          System.arraycopy(b,sindex+1,t,0,tmpval-1);
          tmpStr[0]=tmp;
          /*Creates and initializes the object of the String class.*/
          tmpStr[1]=new String(t);
          /*Returns the user name array string.*/
          return tmpStr;
       };
       /*
          storeMessage_on_Database() - This method stores the messages on database.
             Parameter:
             u_msg: Contains the message.
             to_clientID: Contains the sender user name.
             from_id: Contains the receiver user name.
             Return Value: NA
       */
public void storeMessage_on_Database(String u_msg,String to_clientID,String from_id)
       {
          /*Inserts code here to implement this method.*/
       };

    };
}
```

Download this Listing.

In the above code, the PRServer_Socket class creates a socket on port 8888 to send and receive private messages. The methods defined in this code are:

- `run()`: Calls the init_socket() method to initialize the socket and calls the accept_connection() method to accept the connection.
- `init_socket()`: Opens the selector using the openSelector() method of the Selector class and opens the server socket channel using the open() method of the ServerSocketChannel class. The init_socket() method then configures the server as non blocking using the configureBlocking() method of the ServerSocketChannel class, and creates the object of the InetSocketAddress class at port 9999. Finally, this method binds the socket to the channel at that server IP and calls the register_server() method.
- `register_server()`: Calls the register() method of the ServerSocketChannel to register the server.
- `accept_connection()`: Calls the select() method of the Selector class and checks the value of a variable, kichek. This method then initializes the object of the Iterator class by calling the selectedKeys() method of the Selector class. If the searcher variable has more elements, the accept_connection() method gets the next element and removes the item from the searcher. If the message provider gets a new connection from the end user, this method creates and initializes the object of the ServerSocketChannel class to get the channel using the channel() method. Next, the accept_connection() method calls the got_connection() method to get the connection. If the selector provider gets the stream to read the message from the end user, the accept_connection() method calls the read_Message() method to read the message. If the selector Provider gets the stream to write the message that is to be sent by the chat server to the end user, the accept_connection() method creates the object of the SocketCallback class and calls the attachment() method of the SelectionKey class. This method then calls the write_Message() method to write the message.
- `read_Message()`: Reads the bytes from the channel to the buffer and flips the buffer. This method then retrieves the data from the buffer and initializes the object of the Inner_checker class.
- `closeRemoteChannel()`: Removes all the messages stored in the privateMessage array list and the user names stored in the userName array list.
- `write_Message()`: Calls the doBroadcast() method of the SocketCallback class and creates the object of the ByteBuffer class. The write_Message() method then wraps the message bytes and writes the byte buffer to the server socket.

- `got_connection()`: Initializes the object of the SocketChannel class to accept the connection. This method then sets the server to a non blocking server using the configureBlocking() method and registers the server socket with the selector. The got_connection() method creates the object of the SocketCallback class and calls the attach() method of the SelectionKey class.
- `finalize()`: Calls the close() method to close the server socket channel and selector.

The above code uses an inner class, named Inner_checker, to send private messages. The methods defined in this inner class are:

- `run()`: Sends and stores private messages in the database. This method calls the readName() method to read the name of the sender. If the names of the sender and receiver are identical, the run() method calls the break method. Otherwise, the run() method initializes the object of the Iterator class to iterate the private message and calls the append_broad_msg() message of the SocketCallback class to send the private message. The run() method calls the storeMessage_on_Database() method to store the messages in a database.
- `readName()`: Reads the data string from the server and separates the user name from this string using a separator, 3. This method reads the name and calls the setUserId() method of the SocketCallback class.

**Note** The storeMessage_on_Database() method stores the messages in the database, but this method is not implemented in the above code. You can implement this method for future enhancements.

### Creating the Msgbroadcast.java File

The Msgbroadcast.java file sends the message to all the end users connected to the chat server.

Listing 2-4 shows the contents of the Msgbroadcast.java file:

**Listing 2-4: The Msgbroadcast.java File**

```
/*Imports the java.util package class.*/
import java.util.*;
/*
   Class Msgbroadcast - Broadcasts the messages to all the end users connected in a chat session.
   storer - Stores the user list.
   sc - Stores the object of the SocketCallback class.
   next - Contains the Iterator object.
   msg - Contains the message.
   userId - contains the name of the end user.
*/
public class Msgbroadcast extends Thread
{
   /*Declares the object of the ArrayList class.*/
   ArrayList storer = null;
   /*Declares the object of the SocketCallback class.*/
   SocketCallback sc = null;
   /*Declares the object of the Iterator class.*/
   Iterator next = null;
   /* Declares the object of the String class.*/
   String msg = "";
   String userId = null;
   /*Defines the default constructor of the Msgbroadcast class.*/
   Msgbroadcast(ArrayList store,String msg,String userId)
   {
      storer = store;
      this.msg = msg;
      this.userId = userId;
      /*Calls the start() method of the Thread class. */
      start();
   }
   /*
      run() - This method is called when the thread is started.
         Parameters:   NA
         Return Value: NA
   */
   public void run()
   {
      /* Initializes the object of the Iterator class to get the user id from the user list. */
      next=storer.iterator();
      while(next.hasNext())
      {
         /* Gets the user id from the socket. */
         sc=(SocketCallback)next.next();
         /* Broadcasts the messages to all end users. */
         sc.append_broad_msg(userId + ":" + msg);
      }
   }
}
```

Download this Listing.

In the above code:

- The constructor of the Msgbroadcast class calls the start() method of the Thread class to invoke the run() method.

- The run() method initializes the object of the Iterator class to get the user name from the user list. The run() method also retrieves the user name from the server socket and calls the append_broad_msg() method of the SocketCallback class.

### Creating the AppendUserList.java File

The AppendUserList.java file retrieves the user list from the server, adds a new end user to it, and returns the updated user list to the server.

Listing 2-5 shows the contents of the AppendUserList.java file:

**Listing 2-5: The AppendUserList.java File**

```
/* Imports the java.util package class. */
import java.util.*;
/*
   Class AppendUserList - This class stores the name of all the end users connected in the chat sess
   Field:
   sc: Contains the object of the SocketCallback class.
   uL: Stores the user list in an array list.
   Method:
   run() - This method is called by the start() method of Thread class. This method adds the
   user id.
*/
public class AppendUserList extends Thread
{
   /* Declares the objects of the SocketCallback class. */
   SocketCallback csc=null;
   SocketCallback sc=null;
   /* Declares the object of the ArrayList class. */
   ArrayList uL=null;
   /* Defines default constructor of the AppendUserList class. */
   public AppendUserList(SocketCallback csc,ArrayList UL)
   {
      this.csc=csc;
      this.uL=UL;
      /* Calls the start() method of the Thread class. */
      start();
   }
   /*
run() - This method is called when the thread is started.
      Parameters:   NA
      Return Value: NA
   */
   public void run()
{
   /*Creates and initializes the object of the Iterator class to get the user id from the user list.
   Iterator next=uL.iterator();
   /*Creates a string that contains the user id separator "31".*/
   String s=""+(char)31;
   while(next.hasNext())
   {
      /* Gets the user id from the socket. */
      sc=(SocketCallback)next.next();
      s=s+sc.getUserId()+(char)31;
   }
   /*Adds a user id to the user list.*/
   this.csc.append(s);
   }
}
```

Download this Listing.

In the above code:

- The constructor of the AppendUserList class calls the start() method of the Thread class that invokes the run() method.
- The run() method creates and initializes the object Iterator class to get the user name from the user list.
- The run() method also creates a string that contains the user id with separator, 31.
- The run() method retrieves the user id from the socket and adds the user name to the user list.

### Creating the SocketCallback.java File

The SocketCallback.java file contains the methods to provide connectivity between the client and the server. The SocketCallback class establishes a connection between the chat server and the chat client and gets the file channel to send messages to all the users connected to the chat server.

Listing 2-6 shows the contents of the SocketCallback.java file:

**Listing 2-6: The SocketCallback.java File**

```
/* Imports the java.nio package class. */
import java.nio.*;
import java.nio.channels.*;
/*
Class PRServer_Socket - Uses the sockets to communicate with the chat client.
   Fields:
   soket: Contains the object of the SocketChannel class.
   UserId: Contains the user id.
   Pwd: Contains the password.
   Uinfo: Contains the user information
   broadcast_msg: Contains the broadcasting message.
   Methods:
   isValidUser(): Checks the end user is valid or not.
   ValidUser(): Returns the valid user.
   getChannel(): Returns the socket channel.
   setUserId(): Sets the user id.
   getUserId(): Retrieves the user id.
   setPwd(): Sets the password.
   getPwd(): Retrieves the password.
   setUinfo(): Sets the user information.
   getUinfo(): Retrieves the user information.
   append(): Appends the message.
   append_broad_msg(): Appends the broadcasting messages.
   get_broad_msg(): Retrieves the broadcast messages.
   addString(): Adds message string.
   getString(): Retrieves the messages string.
   getConid(): Returns the connection id.
   doBroadcast(): Broadcasts the message to all user.
   isBroadcast(): Checks message is broadcasted or not.
   setConid(): Sets the connection id.
*/
public class SocketCallback
{
   /*Declares the object of the SocketChannel class.*/
   private SocketChannel soket=null;
   /*Declares the objects of the String class.*/
   private String str="";
   private String UserId="";
   private String Pwd="";
   private String Uinfo="";
   private int cid=0;
   private boolean isvalid_user=false;
   private boolean userMark=false;
   private boolean msgbroadcast=false;
   String broadcast_msg="";
   /*Defines the default constructor.*/
   public SocketCallback(SocketChannel sc)
   {
      soket=sc;
   }
   /* The isValidUser() implementation. */
   public boolean isValidUser()
   {
      return isvalid_user;
   }
   /*The ValidUser() implementation.*/
   public void ValidUser(boolean b)
   {
      isvalid_user=b;
   }
   /*The getChannel() implementation.*/
   public SocketChannel getChannel()
   {
      return soket;
   }
   /*The setUserId() implementation.*/
   public void setUserId(String u)
   {
      UserId=u;
   }
   /*The getUserId() implementation.*/
   public String getUserId()
   {
      return UserId;
   }
   /*The setPwd() implementation.*/
   public void setPwd(String u)
   {
      Pwd=u;
   }
   /* The getPwd() implementation. */
   public String getPwd()
   {
      return Pwd;
   }
   /* The setUinfo() implementation. */
```

```
    public void setUinfo(String u)
    {
       Uinfo=u;
    }
    /* The getUinfo() implementation. */
    public String getUinfo()
    {
       return Uinfo;
    }
    /* The append() implementation. */
    public void append(String s)
    {
       broadcast_msg=s+broadcast_msg;
    }
    /*The append_broad_msg() implementation.*/
    public void append_broad_msg(String msg)
    {
       if (msg.length()>0)
    {
       broadcast_msg+=msg;
    }
    }
    /*The get_broad_msg() implementation.*/
    public String get_broad_msg()
    {
       return broadcast_msg;
    }
    /*The addString() implementation.*/
    public void addString(String s)
    {
       str=s;
    }
    /*The getString() implementation.*/
    public String getString()
    {
       String str1=new String(str);
       str="";
       return str1;
    }
    /*The getConid() implementation.*/
    public int getConid()
    {
       return cid;
    }
    /*The doBroadcast() implementation.*/
    public void doBroadcast(boolean chk)
    {
       msgbroadcast=chk;
    }
    /*The doBroadcast() implementation.*/
    public void doBroadcast()
    {
       broadcast_msg="";
    }
    /* The isBroadcast() implementation. */
    public boolean isBroadcast()
    {
       return msgbroadcast;
    }
    /* The setConid() implementation. */
    public void setConid(int id)
    {
       cid=id;
    }
}
```

Download this Listing.

In the above code, the SocketCallback class implements all the methods that are invoked in the server classes. The methods defined in this code are:

- `isValidUser()`: Checks whether or not the end user is valid. This method returns true if the end user is valid.
- `getChannel()`: Returns the socket channel.
- `setUserId()`: Sets the user name.
- `getUserId()`: Retrieves the user name.
- `setPwd()`: Sets the password.
- `getPwd()`: Retrieves the password.
- `setUinfo()`: Sets the user information.
- `getUinfo()`: Retrieves the user information.

- `append()`: Appends the message.
- `append_broad_msg()`: Appends the broadcasting messages.
- `get_broad_msg()`: Retrieves the broadcast messages.
- `addString()`: Adds message string.
- `getString()`: Retrieves the message string.
- `getConid()`: Returns the connection id.
- `doBroadcast()`: Broadcasts the message to all the end users.
- `isBroadcast()`: Checks whether or not the message is broadcasted.
- `setConid()`: Sets the connection id.



◄ PREVIOUS NEXT ►

# Creating the Chat Client

To create the chat client, you need to create the following files:

- ChatLogin.java file
- ChatClient.java file
- CClient.java file
- Messager.java file

### Creating ChatLogin.java File

The ChatLogin.java file helps create a login window. In the login window, an end user enters the server IP address, user name, and password to connect to the chat server.

Listing 2-7 shows the contents of the ChatLogin.java file:

**Listing 2-7: The ChatLogin.java File**

```
/* Imports the javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
import javax.swing.JPasswordField;
import javax.swing.BorderFactory;
/*Imports the java.awt package classes.*/
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
import java.awt.Font;
/*Imports the javax.swing.event package classes.*/
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JOptionPane;
import javax.swing.UIManager;
import javax.swing.JSeparator;
/*
class ChatLogin - Creates the Chat Login window for the chat application.
      Method:
      - actionPerformed()
      - main()
*/
public class ChatLogin extends JDialog implements ActionListener
{
   JPanel panel;
   JPanel pane;
   /* Declares the objects of the JLabel class. */
   JLabel titleLabel;
   JLabel nameLabel;
   JLabel pwdLabel;
   JLabel ipLabel;
   /* Declares the objects of JTextField class */
   JTextField nameText;
   JPasswordField pwdText;
   JTextField ipText;
   /*Declares the objects of the JButton class.*/
   JButton connect;
   JButton cancel;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   int value;
   /* Defines the default constructor.*/
   public ChatLogin()
   {
      /*Sets the title of the Font dialog box.*/
      setTitle("Chat Login Window");
      /*Sets the size of Font dialog box.*/
      setSize(280, 170);
      /*Sets resizable button to FALSE.*/
      setResizable(false);
      /*Initializes the object of the GridBagLayout class*/
      gbl = new GridBagLayout();
      /* Sets the Layout. */
      getContentPane().setLayout(gbl);
      /*Creates an object of the GridBagConstraints class.*/
```

```
        gbc = new GridBagConstraints();
        /*
           Initializes the title label object and adds it to the 1, 1, 2, 1 position with WEST
           alignment.
        */
           gbc.gridx = 1;
           gbc.gridy = 1;
           gbc.gridwidth = 2;
           gbc.gridheight = 1;
           gbc.anchor = GridBagConstraints.CENTER;
           pane = new JPanel();
        pane.setLayout(new FlowLayout());
        titleLabel = new JLabel("Chat Login Window");
        titleLabel.setFont(new Font("Verdana",Font.BOLD,20));
        pane.add(titleLabel);
        getContentPane().add(pane, gbc);
        gbc.gridx = 1;
        gbc.gridy = 2;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        getContentPane().add(new JSeparator(), gbc);
        /*
           Initializes the IP address label object and adds it to the 1, 3, 1, 1 position with
           EAST alignment
        */
        gbc.gridx = 1;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.EAST;
        ipLabel = new JLabel("Server IP: ");
        getContentPane().add(ipLabel, gbc);
        /*
Initializes the IP address text field object and adds it to the 2, 3, 1, 1 position with WEST alignm
        */
        gbc.gridx = 2;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        ipText = new JTextField("192.168.0.36", 15);
        ipText.setFont(new Font("Verdana",Font.PLAIN,12));
        getContentPane().add(ipText, gbc);
        gbc.gridx = 1;
        gbc.gridy = 4;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        getContentPane().add(new JSeparator(), gbc);
        /*
Initializes the user name label object and adds it to the 1, 5, 1, 1 position with EAST alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 5;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.EAST;
        nameLabel = new JLabel("User Name: ");
        getContentPane().add(nameLabel, gbc);
        /*
           Initializes the user name text field object and adds it to the 2, 5, 1, 1 position with
           WEST alignment.
        */
        gbc.gridx = 2;
        gbc.gridy = 5;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        nameText = new JTextField(15);
        nameText.setFont(new Font("Verdana",Font.PLAIN,12));
        getContentPane().add(nameText, gbc);
        gbc.gridx = 1;
        gbc.gridy = 6;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        getContentPane().add(new JSeparator(), gbc);
        /*
           Initializes the password label object and adds it to the 1, 7, 1, 1 position with EAST
           alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 7;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.EAST;
        pwdLabel = new JLabel("Password: ");
```

```
      getContentPane().add(pwdLabel, gbc);
      /*
         Initializes the password text field object and adds it to the 2, 7, 1, 1 position with
         WEST alignment.
      */
      gbc.gridx = 2;
      gbc.gridy = 7;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      pwdText = new JPasswordField(15);
      pwdText.setFont(new Font("Verdana",Font.PLAIN,12));
      pwdText.addActionListener(this);
      getContentPane().add(pwdText, gbc);
      gbc.gridx = 1;
      gbc.gridy = 8;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      getContentPane().add(new JSeparator(), gbc);
      /*
         Initializes the OK and Cancel button. Adds the button to the 2, 9, 1, 1 position with
         WEST alignment.
      */
      gbc.gridx = 2;
      gbc.gridy = 9;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      panel = new JPanel();
      panel.setLayout(new FlowLayout(FlowLayout.LEFT));
      connect = new JButton("Connect");
      connect.addActionListener(this);
      cancel = new JButton("Cancel");
      cancel.addActionListener(this);
      panel.add(connect);
      panel.add(cancel);
      getContentPane().add(panel, gbc);

      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent we)
         {
            System.exit(0);
         }
      });
   }
   /*
actionPerformed() - This method is called when the user clicks the any button.
Parameters: ae: Represents an object of the ActionEvent class that contains the details of the event
   Return Value: NA
   */
   public void actionPerformed(ActionEvent ae)
   {
      if(ae.getSource() == connect)
      {
/*Creates an object of the ChatClient class.*/
         ChatClient tcc = new ChatClient();
/*Calls the con() method to connect to the server.*/
tcc.con(nameText.getText(), pwdText.getText(), ipText.getText());
         this.setVisible(false);
      }
      else if(ae.getSource() == cancel)
      {
         System.exit(0);
      }
      else if(ae.getSource() == pwdText)
      {
         /*Creates an object of the ChatClient class.*/
         ChatClient tcc = new ChatClient();
         /*Calls the con() method to connect to the server.*/
         tcc.con(nameText.getText(), pwdText.getText(), ipText.getText());
         this.setVisible(false);
         }
         }
         /*Main method.*/
         public static void main(String args[])
         {
            try
            {
               /*Sets the window look and feel to the application.*/
               UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
            }
            catch(Exception e){}
            ChatLogin cl = new ChatLogin();
            cl.show();
      }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the ChatLogin class that allows you to open the Chat Login window, as shown in Figure 2-2:


**Figure 2-2:** The Chat Login Window

After an end user specifies the server IP address, user name, and password and clicks any button on the Chat Login window, the chat application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate class and method, based on the button the end user clicks. If the end user clicks the Connect button or presses the Enter key, the actionPerformed() method creates an object of the ChatClient class and calls the con() method of the ChatClient class.

### Creating ChatClient.java File

The ChatClient.java file helps create a user interface for the chat client. End users can use this interface to display the user list of the connected users. The chat application user interface contains a text pane that displays the messages sent by other chat users.

Listing 2-8 shows the contents of the ChatClient.java file:

**Listing 2-8: The ChatClient.java File**

```
/*Imports the java.awt package classes.*/
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.List;
import java.awt.Point;
/*Imports the java.awt.event package classes.*/
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
/*Imports java.io package classes.*/
import java.io.PrintStream;
/*Imports java.util package classes.*/
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util.Vector;
/*Imports the javax.swing package classes.*/
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.JOptionPane;
import javax.swing.UIManager;
/*Imports the java.swing.text package classes.*/
import javax.swing.text.AttributeSet;
import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;
/*
Class ChatClient - Creates the main window of the chat application. This window contains a text
pane to display the messages and list box to display the user list.
   Methods:
      - prepareAllStyles()
      - run()
      - itemStateChanged()
```

```
      - actionPerformed()
      - con()
      - AddUser()
   Class
      - ListofUser
      - PrivateChat
      - BroadcastChat
*/
public class ChatClient extends JFrame implements ActionListener, Runnable, ItemListener
{
   /*Declares the AWT and Swing components.*/
   JPanel paneMain;
   JPanel paneLeft;
   JPanel paneRight;
   JPanel paneBottom;
   JLabel welcomeLabel;
   JLabel listLabel;
   JLabel pMsgLabel;
   JLabel bMsgLabel;
   JTextField msgText;
   JTextArea pArea;
   JTextField empty;
   List clientList;
   JScrollPane pAreaScroll;
   JScrollPane listScroll;
   JButton sendButton;
   JButton privateButton;
   JButton logoutButton;
   JTextField listText;
   SimpleAttributeSet aset;
   AttributeSet current;
   Hashtable ht = null;
   DefaultStyledDocument doc = null;
   JTextPane tp = null;
   /*Declares the object of the Thread class.*/
   Thread thread;
   Thread t;
   Thread th;
   /*Declares the static string to set the text pane attributes.*/
   final String NORMALBLUE = "NormalBlue";
   final String BOLDBLUE = "BoldBlue";
   final String NORMALBLACK = "NormalBlack";
   final String BOLDGREEN = "BoldGreen";
   final String NORMALRED = "NormalRed";
   final String ITALICRED = "ItalicRed";
   /*Declares the objects of the String class.*/
   String nameStr1 = null;
   String msgStr1 = null;
   String nameStr2 = null;
   String msgStr2 = null;
   String str3 = null;
   String str4 = null;
   String str5 = null;
   String str6 = null;
   String msg;
   String user;
   int value;
   int count=0;
   int i;
   /*Creates the object of the PrivateChat class.*/
   PrivateChat Pchat = new PrivateChat();
   /*Creates the object of the BroadcastChat class.*/
   BroadcastChat Achat = new BroadcastChat();
   /*Creates the object of the ListofUser class.*/
   ListofUser UL=new ListofUser();
   /*Creates the object of the CClient class.*/
   CClient CC = null;
   /*Defines the default constructor.*/
   public ChatClient()
   {
      setSize(700, 400);
      setTitle("Chat Client");
      setResizable(false);
      paneMain = new JPanel();
      paneMain.setLayout(new BorderLayout());
      getContentPane().add(paneMain);
      paneLeft = new JPanel();
      paneLeft.setLayout(new BorderLayout());
      doc = new DefaultStyledDocument();
      ht = new Hashtable();
      tp = new JTextPane(doc)
      {
         public void paintComponent(Graphics g)
         {
         super.paintComponent(g);
         }
         public void paint(Graphics g)
         {
```

```
                super.paint(g);
            }
        };
        tp.setFont(new Font("Verdana", Font.PLAIN, 12));
        tp.setEditable(false);
        pAreaScroll = new JScrollPane(tp);
        paneLeft.add(pAreaScroll, BorderLayout.CENTER);
        paneMain.add(paneLeft, BorderLayout.CENTER);
        paneRight = new JPanel();
        paneRight.setLayout(new BorderLayout());
        listLabel = new JLabel("User List");
        listText = new JTextField(10);
        listText.setVisible(false);
        clientList = new List(3);
        clientList.addActionListener(this);
        clientList.addItemListener(this);
        clientList.select(0);
        listScroll = new JScrollPane(clientList);
        paneRight.add(listText, BorderLayout.SOUTH);
        paneRight.add(listLabel, BorderLayout.NORTH);
        paneRight.add(listScroll, BorderLayout.CENTER);
        paneMain.add(paneRight, BorderLayout.EAST);
        paneBottom = new JPanel();
        paneBottom.setLayout(new FlowLayout());
        msgText = new JTextField(33);
        msgText.requestFocus();
        msgText.addActionListener(this);
        msgText.setFont(new Font("Verdana",Font.PLAIN,14));
        sendButton = new JButton("Send");
        privateButton = new JButton("Private Message");
        logoutButton = new JButton("Logout");
        empty = new JTextField(10);
        paneBottom.add(msgText);
        paneBottom.add(sendButton);
        paneBottom.add(privateButton);
        paneBottom.add(logoutButton);
        paneBottom.add(empty);
        paneMain.add(paneBottom, BorderLayout.SOUTH);
        privateButton.addActionListener(this);
        sendButton.addActionListener(this);
        logoutButton.addActionListener(this);
        prepareAllStyles();
        t = new Thread(this);
        /*
            addWindowListener - It contains a windowClosing() method.
            windowClosing: It is called when the user clicks the cancel button of the Window. It _
            closes the main window.
                Parameter: we- Object of WindowEvent class.
                Return Value: NA
        */
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                value = JOptionPane.showConfirmDialog(null, "Are you sure you want to close the chat
                application?", "Close", JOptionPane.YES_NO_OPTION);
                if(value == 0)
                {
                    try
                    {
                        /*Calls the sendBroadMsg() method to inform all the users.*/
                        CC.sendBroadMsg(user + "* has left the chat session." + "+REMOVE");
                        System.exit(0);
                    }
                    catch(Exception e){}
                }
                else if(value == 1)
                {
                }
            }
        });
    }
/*Defines the prepareAllStyles() method that set the text attribute values.*/
public void prepareAllStyles()
    {
        aset = new SimpleAttributeSet();
        StyleConstants.setForeground(aset,Color.blue);
        StyleConstants.setFontSize(aset,12);
        StyleConstants.setFontFamily(aset,"Verdana");
        ht.put(NORMALBLUE,aset);
        aset = new SimpleAttributeSet();
        StyleConstants.setForeground(aset,Color.blue);
        StyleConstants.setFontSize(aset,12);
        StyleConstants.setFontFamily(aset,"Verdana");
        StyleConstants.setBold(aset, true);
        ht.put(BOLDBLUE,aset);
        aset = new SimpleAttributeSet();
        StyleConstants.setForeground(aset,Color.black);
```

```
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      ht.put(NORMALBLACK,aset);
      aset = new SimpleAttributeSet();
      StyleConstants.setForeground(aset, new Color(0, 128, 0));
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      StyleConstants.setBold(aset, true);
      ht.put(BOLDGREEN,aset);
      aset = new SimpleAttributeSet();
      StyleConstants.setForeground(aset,Color.red);
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      ht.put(NORMALRED,aset);
      aset = new SimpleAttributeSet();
      StyleConstants.setForeground(aset,Color.red);
      StyleConstants.setFontSize(aset,12);
      StyleConstants.setFontFamily(aset,"Verdana");
      StyleConstants.setItalic(aset, true);
      ht.put(ITALICRED,aset);
   }
   /*
itemStateChanged() - This method is called when the end user selects the user name from the user lis
   Parameters: ae: Represent an object of the ActionEvent class that contains the details of the eve
      Return Value: NA
   */
   public void itemStateChanged(ItemEvent ie)
   {
      if(ie.getSource() == clientList)
      {
listText.setText(clientList.getSelectedItem());
         i = clientList.getSelectedIndex();
      }
   }
   /*
      actionPerformed() - This method is called when the end user clicks any button.
         Parameters: ae: Represent an object of the ActionEvent class that contains the details
      of the event.
      Return Value: NA
   */
   public void actionPerformed(ActionEvent ae)
   {
      /*
         This section is executed when the end user clicks the Private Message button of the applica
      */
      if(ae.getSource() == privateButton)
      {
         try
         {
if(listText.getText().equals("*" + user))
            {
               JOptionPane.showMessageDialog(null, "You cannot send private message to _
               yourself!", "Error",
               JOptionPane.ERROR_MESSAGE);
            }
            else
            {
               current = (AttributeSet)ht.get(BOLDBLUE);
               doc.insertString(doc.getLength(), "To " + listText.getText() + ": " ,current);
               current = (AttributeSet)ht.get(NORMALBLACK);
               doc.insertString(doc.getLength(), msgText.getText() +  "\n" ,current);
               Point pt1=tp.getLocation();
               Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
               pAreaScroll.getViewport().setViewPosition(pt2);
            }
         }
catch(Exception e) {}
if(msgText.getText().trim().equals(""))
         {
         }
         else
         {
            /*Calls the sendPrivateMessage() method to send the private message.*/
            CC.sendPrivateMessage(msgText.getText().trim(),listText.getText());
            msgText.setText("");
         }
      }
      /*
This section is executed, when the end user clicks the Send button of the application.
      */
      else if(ae.getSource() == sendButton)
      {
         if(msgText.getText().trim().equals(""))
         {
         }
         else
         {
            /*Calls the sendBroadMsg() method to send the broadcast message.*/
```

```
            CC.sendBroadMsg(msgText.getText().trim() + "+MSG");
            msgText.setText("");
         }
      }
      /*
This section is executed, when the end user clicks the Logout button of the application.
      */
      else if(ae.getSource() == logoutButton)
      {
         value = JOptionPane.showConfirmDialog(null, "Are you sure you want to logout?", "Logout",
         JOptionPane.YES_NO_OPTION);
         if(value == 0)
         {
         try
         {
         /*Calls the sendBroadMsg() method to inform the end user has left the chat session.*/
         CC.sendBroadMsg(user + "* has left the chat session." + "+REMOVE");
         System.exit(0);
            }
            catch(Exception e){}
         }
         else if(value == 1)
         {
         }
      }
      /*
This section is executed, when the end user enter the sending text and press the enter button
from the keyboard.
      */
      else if(ae.getSource() == msgText)
      {
/*Calls the sendBroadMsg() method to send the broadcast message.*/
CC.sendBroadMsg(msgText.getText().trim() + "+MSG");
         msgText.setText("");
      }
   }
/*Implementation of con() method. This method establishes the connection between client and server.*
public void con(String userName, String pwd, String ipAdd)
   {
      user = userName;
      CC = new CClient();
      /*Calls the addMessager() method of the CClient class.*/
      CC.addMessager(UL , Achat, Pchat);
      try
      {
         /*Calls the connect() method of the CClient class.*/
         CC.connect(userName, pwd, ipAdd);
      }
      catch(Exception e)
      {
         JOptionPane.showMessageDialog(null, "Unable to connect the server!", "Error",
         JOptionPane.ERROR_MESSAGE);
         ChatLogin cl = new ChatLogin();
         cl.show();
         this.setVisible(false);
         return;
      }
setTitle("Chat Application - " + userName);
/*Starts the thread to send to inform all the users.*/
      try
      {
         count = 1;
         t = new Thread(this);
         t.setPriority(Thread.MAX_PRIORITY);
         t.sleep(100);
         t.start();
      }
      catch(Exception e){};
/*Starts the thread to display the user list.*/
      try
      {
         count = 2;
         thread = new Thread(this);
         thread.sleep(100);
         thread.start();
      }
      catch(Exception e){}
      this.show();
      tp.repaint();
   }
/*Creates the class to display the user list.*/
   class ListofUser implements Messager
   {
      public void message(String m)
      {
         clientList.clear();
         AddUser(m);
         clientList.select(i);
```

```
      }
   };
/*Creates the class to send the private message.*/
class PrivateChat implements Messager
   {
      public void message(String m)
      {
         try
         {
            /*Inserts text in the text pane with the specified attributes.*/
            m=m.substring(0,m.length()-1);
            StringTokenizer st1 = new StringTokenizer(m, ":");
            nameStr1 = st1.nextToken();
            msgStr1 = st1.nextToken();
            current = (AttributeSet)ht.get(BOLDBLUE);
            doc.insertString(doc.getLength(), "From " + nameStr1 + ": ", current);
            current = (AttributeSet)ht.get(NORMALBLACK);
            doc.insertString(doc.getLength(), msgStr1 + "\n" , current);
            Point pt1=tp.getLocation();
            Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
            pAreaScroll.getViewport().setViewPosition(pt2);
         }
         catch(Exception e){}
         tp.repaint();
      }
   };
/*Creates the class to send the broadcast message.*/
   class BroadcastChat implements Messager
   {
      public void message(String m)
      {
         try
         {
            /*Inserts text in the text pane with the specified attributes.*/
            StringTokenizer st2 = new StringTokenizer(m, ":");
            nameStr2 = st2.nextToken();
            msgStr2 = st2.nextToken();
            StringTokenizer st3 = new StringTokenizer(msgStr2, "+");
            str3 = st3.nextToken();
            str4 = st3.nextToken();
            if(str4.equals("ADD"))
            {
               StringTokenizer st4 = new StringTokenizer(str3, "*");
               str5 = st4.nextToken();
               str6 = st4.nextToken();
               if(str5.equals(user))
               {
                  try
               {
                  msg = "Welcome to chat session!";
                  current = (AttributeSet)ht.get(BOLDBLUE);
                  tp.setCharacterAttributes(current,true);
                  doc.insertString(doc.getLength(),  msg + "\n",current);
               }
               catch(Exception e){}
               }
               else
               {
                  try
                  {
                     current = (AttributeSet)ht.get(ITALICRED);
                     doc.insertString(doc.getLength(), str5 + str6 + "\n",current);
                     Point pt1=tp.getLocation();
                     Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                     pAreaScroll.getViewport().setViewPosition(pt2);
                  }
               catch(Exception e){}
               }
            }
else if(str4.equals("REMOVE"))
            {
               StringTokenizer st4 = new StringTokenizer(str3, "*");
               str5 = st4.nextToken();
               str6 = st4.nextToken();
               if(str5.equals(user))
               {
                  try
                  {
                     msg = "You have successfully logged out from this chat session.";
                     current = (AttributeSet)ht.get(ITALICRED);
                     doc.insertString(doc.getLength(), msg + "\n",current);
                     Point pt1=tp.getLocation();
                     Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                     pAreaScroll.getViewport().setViewPosition(pt2);
                  }
               catch(Exception e){}
               }
```

```
                else
                {
                   try
                   {
                      current = (AttributeSet)ht.get(ITALICRED);
                      doc.insertString(doc.getLength(), str5 + str6 + "\n",current);
                      Point pt1=tp.getLocation();
                      Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                      pAreaScroll.getViewport().setViewPosition(pt2);
                   }
                catch(Exception e){}
                }
             }
             else if (str4.equals("MSG"))
             {
                try
                {
                   current = (AttributeSet)ht.get(BOLDGREEN);
                   doc.insertString(doc.getLength(), nameStr2 + ": " , current);
                   current = (AttributeSet)ht.get(NORMALBLUE);
                   doc.insertString(doc.getLength(), str3 + "\n",current);
                   Point pt1=tp.getLocation();
                   Point pt2=new Point((int)(0),(int)(tp.getBounds().getHeight()));
                   pAreaScroll.getViewport().setViewPosition(pt2);
                }
                catch(Exception e){}
             }
          }
          catch (Exception ex) {}
          tp.repaint();
       }
    };
/*Creates the class to add a new end user into the user list.*/
private void AddUser(String s)
    {
       String tmp="";
       byte b[]=s.getBytes();
       if(s.length()>1)
        {
          for(int x=1;x<b.length;x++)
          {
             if(b[x]==31)
             {
                if(tmp.equals(user))
                {
                   clientList.add("*"+tmp);
                }
                else
                {
                   clientList.add(tmp);
                }
                tmp="";
             }
             else
             tmp+=(char)b[x];
          }
       }
    };
/*This method is invoked when thread is started.*/
public void run()
    {
       if(count == 1)
       {
          /*Calls the sendBroadMsg() to inform all the end users.*/
          CC.sendBroadMsg(user + "* has joined the chat session." + "+ADD");
       }
       else if(count == 2)
       {
          while(true)
          {
             try
             {
                /*Calls the sendListMsg() method to display the list of users.*/
                CC.sendListMsg("");
                Thread.sleep(5000);
             }
catch(Exception ee){ee.printStackTrace();}
          }
       }
    }
}
```

Download this Listing.

In the above code, the constructor of the ChatClient class creates the user interface for the chat client window, as shown in Figure 2-3:

**Figure 2-3:** The Chat Client Window

The text pane displays the messages the end users send or receive. The user interface allows an end user to send private and broadcast messages to other end users connected with the chat server. To send a public or broadcast message to all the users, the end user needs to type the message in the text box in the bottom pane and click the Send button. To send a private message to a particular chat client, the end user needs to select the client's user name from the user list, type the message in the text box, and click the Private Message button.

The inner classes defined in the above code are:

- `ListofUser`: Implements the Messager interface and defines the abstract method, message(). The message() method calls the clear() method to clear the user list and the AddUser() method to add the end user.
- `PrivateChat`: Implements the Messager interface and defines the abstract method, message(). The message() method tokenizes the message to retrieve the user name and the message. The message() method then sets the attributes of the document and calls the insertString() method to insert the message in the text pane.
- `BroadcastChat`: Implements the Messager interface and defines the abstract method, message(). The message() method tokenizes the message to retrieve the user name and the message. Next, the message() method again tokenizes the message to check the message type, such as new end user connected message, user left message, or simple broadcast message. The message() method then sets the attributes of the document and calls the insertString() method to insert the message in the text pane.

The methods defined in the above code are:

- `con()`: Accepts the user name, password, and IP address and calls the addMessager() method of the CClient class. The con() method also calls the connect() method of the CClient class to connect the end user to the chat server.
- `run()`: Checks the value of the variable count. If the count value is 1, this method calls the sendBroadMsg() method of the CClient class to send a message to all the users that a new end user has joined the chat session. If the count value is 2, the run() method calls the sendListMsg() method of the CClient class to display the user list.
- `prepareAllStyles()`: Sets the text attribute values. This method initializes the object of the SimpleAttributeSet class and calls the setForeground(), setFontSize(), setFontFamily(), setBold(), and setItalic() methods of the StyleConstants class to set the document styles.
- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method, based on the button the end user clicks on the Chat Client window. If the end user clicks the Send button of the Chat application, the actionPerformed() method calls the sendBroadMsg() method to send the message to all the users connected to the chat server. If the end user clicks the Private Message button of the Chat application, the actionPerformed() method calls the sendPrivateMessage() method to send the message to a specified chat user. If an end user clicks the Logout button of the Chat application, the actionPerformed() method calls the sendBroadMsg() method to send a message to all the users that this end user has left the chat session. The actionPerformed() method then calls the System.exit(0) method to close the Chat application window.
- `itemStateChanged()`: Acts as an event listener and activates an appropriate method, based on the item the end user selects from the user list box.
- `AddUser()`: Adds the user to the user list. This method separates the user name from the message sent by the chat server to chat client.

### Creating CClient.java File

The CClient.java file implements the core functionality of the chat client module. This file reads the server IP address, user name, and password from the Chat Login window and connects the end user to the chat server. An end user can use this file to send and receive messages from the chat server.

Listing 2-9 shows the contents of the CClient.java file:

**Listing 2-9: The CClient.java File**

```
/* Imports the java packages. */
import java.nio.*;
import java.nio.channels.*;
import java.nio.channels.spi.*;
import java.util.*;
import java.io.*;
import java.net.*;
```

```
/*
Class CClient - Implements the methods declared or called in the chat application client. This _
file connects the chat client to chat server.
   Method:
      - addMessager()
      - getBroadCastSender()
      - getPrivateSender()
      - connect()
      - runClient()
      - run()
      - sendBroadMsg()
      - sendListMsg()
      - sendPrivateMessage()

*/
public class CClient extends Thread
{
   /* Declares the objects and initializes the variables. */
   private String userId="";
   private String pwd="";
   ByteBuffer buffer=ByteBuffer.allocateDirect(1024);
   SocketChannel Psc=null;
   SocketChannel Asc=null;
   private InetSocketAddress AAddr=null;
   private InetSocketAddress PAddr=null;
   private int APort=9999;
   private int PPort=8888;
   boolean connectStatus=false;
   private ArrayList Lmsg=null;
   private boolean Validity_Checker=false;
   public Messager usrlist=null;
   public Messager bcmsg=null;
   public Messager prmsg=null;
   boolean chkfirst=true;
   /* Defines the default constructor. */
   public CClient()
   {
   }
   /* Implements the addMessager() method. */
   public void addMessager(Messager usrlist, Messager bcmsg, Messager prmsg)
   {
      this.usrlist=usrlist;
      this.bcmsg=bcmsg;
      this.prmsg=prmsg;
   }
   /*Implements the connect() method. This method connects the specified user to the server.*/
   public void connect(String UserId,String Password,String HostAddress) throws
   NotYetConnectedException
   {
      this.userId=UserId;
      this.pwd=Password;
      try
      {
      Asc=SocketChannel.open(new InetSocketAddress(HostAddress,APort));
      Psc=SocketChannel.open(new InetSocketAddress(HostAddress,PPort));
      while(!Asc.finishConnect())
      {
      }
      while(!Psc.finishConnect())
      {
      }
      runClient();
      sendBroadMsg("");
      sendPrivateMessage(" "," ");
      }
      catch(Exception e1)
      {
         throw new NotYetConnectedException();
      }
   }
/* Implements the runClient() method. This method runs the client. */
   public void runClient() throws Exception
   {
      this.start();
      new PrivateReceiver(Psc);
   }
/*Implements the run() method. This method is invoked when thread is started.*/
public void run()
   {
      byte[] Tdata=null;
      while(true)
      {
         try
         {
            buffer.clear();
            int nbyte=Asc.read(buffer);
            if(nbyte==-1)
            {
```

```
                bcmsg.message("Connection Closed");
                Asc.close();
                return;
             }
             if (nbyte>0)
             {
/*Reads the messages from the chat server.*/
buffer.flip();
Tdata=new byte[nbyte];
buffer.get(Tdata,0,nbyte);
if(Tdata[0]!=31)
             {
                String s=new String(Tdata);
                bcmsg.message(s);
             }
             else
             {
                String s=new String(Tdata);
                usrlist.message(s);
             }
          }
          catch(IOException ex)
          {
             try
             {
                Asc.close();
             }
             catch(IOException ec)
             {
                ec.printStackTrace();
             }
          }
       }
    }
/*Implements the sendBroadMsg() method. This method broadcast the message to all the end users.*/
    public void sendBroadMsg(String msg)
    {
       try
       {
          if(!connectStatus)
          {
             String tmps=userId+(char)28+pwd+(char)29+"";
             ByteBuffer b=ByteBuffer.wrap(tmps.getBytes());
             Asc.write(b);
             connectStatus=true;
          }
          else
          {
             ByteBuffer b=ByteBuffer.wrap(msg.getBytes());
             Asc.write(b);
          }
       }
       catch(Exception sb)
       {
          sb.printStackTrace();
       }
    }
/*Implements the sendListMsg() method. This method sends the user list to the end user.*/
public void sendListMsg(String msg)
    {
       try
       {
          String tmps=""+(char)31+"";
          ByteBuffer b=ByteBuffer.wrap(tmps.getBytes());
          Asc.write(b);
       }
       catch(Exception sl)
       {
          sl.printStackTrace();
       }
    }
/*Implements the sendPrivateMessage() method.*/
public void sendPrivateMessage(String Msg,String to)
    {
       try
       {
          String Ts=null;
          if(chkfirst)
          {
             Ts=to+(char)3+userId+(char)3+Msg;
             chkfirst=false;
          }
          else
             Ts=to+(char)3+Msg;
             ByteBuffer b=ByteBuffer.wrap(Ts.getBytes());
             Psc.write(b);
```

```
        }
        catch(Exception se)
        {
           se.printStackTrace();
        }
     }
     /*Creates the PrivateReceiver class.*/
     class PrivateReceiver extends Thread
     {
        SocketChannel P_channel=null;
        ByteBuffer buff=ByteBuffer.allocateDirect(1024);
        byte[] Tdata=null;
        /*Defines the default constructor.*/
        PrivateReceiver(SocketChannel chan )
        {
            this.P_channel=chan;
           start();
        }
        /*Implements the run() method.*/
        public void run()
        {
           while(true)
           {
              try
              {
                 buff.clear();
                 int nbyte=P_channel.read(buff);
                 if(nbyte==-1)
                 {
                    prmsg.message("Connection Closed");
                    P_channel.close();
                    return;
                 }
                 if (nbyte>0)
                 {
                    /* Reads the data. */
                    buff.flip();
                    Tdata=new byte[nbyte];
                    buff.get(Tdata,0,nbyte);
                    String s=new String(Tdata);
                    prmsg.message(s);
                 }
              }
              catch(IOException ex)
              {
                 try
                 {
                 P_channel.close();
                 }
                 catch(IOException ec)
                 {
                    ec.printStackTrace();
                 }
              }
           }
         }
     };
}
```

Download this Listing.

In the above code, the CClient class defines the default contractor. The methods defined in this code are:

- `addMessager()`: Retrieves the user list, broadcast message, and private message.
- `connect()`: Opens two sockets to send private and broadcast messages. This method then calls the runClient(), sendBroadMsg(), and sendPrivateMsg() methods.
- `runClient()`: Starts a thread, initializes the object of the PrivateReceiver class, and invokes the run() method.
- `run()`: Reads the bytes from the socket to the buffer and separates the user name and broadcast message from the received message. The run() method then calls the message() method to send the broadcast message and the user list.
- `sendBroadMsg()`: Checks the connection status, wraps the message to the byte buffer, and writes the buffer to the server socket.
- `sendListMsg()`: Wraps the message to the byte buffer and writes the buffer to the server socket.
- `sendPrivateMessage()`: Checks the destination user id, wraps the message to the byte buffer, and writes the buffer to the server socket.

The CClient class contains a sub class, PrivateReceiver, which receives the message from the server. This class defines the default constructor that calls the start() method of the Thread class to invoke the run() method. The run() method clears the buffer and reads the bytes from the channel to the buffer. This method then flips the buffer and calls the message() method to send a private message.

### Creating Messager.java File

The Messager.java file creates an interface that declares the message() method. This method helps display the message in the text pane of the chat client window.

Listing 2-10 shows the contents of the Messager.java file:

**Listing 2-10: The Messager.java File**

```
/*Defines an abstract method to read and write the message to the text pane of the chat _
application client.*/
public interface Messager
{
   /*Declares the message() method.*/
   public void message(String msg);
}
```

Download this Listing.

In the above code, the Messager interface declares an abstract message() method. This method is defined later, where this interface is implemented to send or receive message from the chat server..

## Unit Testing

To test the Chat application:

1. 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;
   ```
2. 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;
   ```
3. Copy the ChatServer.java, UAServer_Socket.java, PRServer_Socket.java, AppendUserList.java, Msgbroadcast.java, and SocketCallback.java files to a folder on the server computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. Copy the ChatLogin.java, ChatClient.java, CClient.java, and Messager.java files to a folder on the client computer. Compile the files using the following javac command:
   ```
   javac *.java
   ```
5. Run the Chat application server using the following command at the command prompt:
   ```
   java ChatServer
   ```
6. Run the Chat application login window using the following command at the command prompt:
   ```
   java ChatLogin
   ```
   The Chat Login window appears, as shown in Figure 2-2.
7. Specify the server IP address, user name, and password. Click the Connect button. The Chat Application - user1 window appears, as shown in Figure 2-4:

**Figure 2-4:** Welcome Message in the Chat Application - user1 Window

An Error dialog box appears, if the process to establish a connection between the client and the server fails, as shown in Figure 2-5:

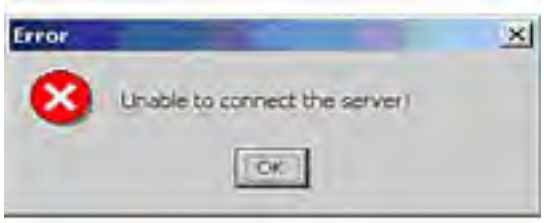


**Figure 2-5:** The Error Dialog Box

8. Run the Chat application login window using the following command at the command prompt from another client computer:
   ```
   java ChatLogin
   ```
   The Chat Login window appears, as shown in Figure 2-2.
9. Specify the server IP address, user name, and password. Click the Connect button. The Chat Application - user2 window appears, as shown in Figure 2-6:

**Figure 2-6:** The Chat Application - user2 Window with List of Users

When user2 joins the chat session, a message, user2 has joined the chat session appears in the text pane of the user1 chat window, as shown in Figure 2-7:

**Figure 2-7:** The Chat Application - user1 Window with user2 in Chat Session

10. Type Hi all in the text box of the Chat application window of user1 and click the Send button. The message is broadcasted to all the users, as shown in Figure 2-8:

**Figure 2-8:** Receiving Message in the Chat Application - user2 Window

Figure 2-9 shows the broadcast message that appears in the user2 client window:

**Figure 2-9:** Receiving Message in the Chat Application - user2 Window

11. Type Hi user2! in the text box of the user1 Chat application window, select user2 from the User List, and click the Private Message button. The message is sent to the user2 only, as shown in Figure 2-10:

**Figure 2-10:** Receiving Message in the Chat Application - user1 Window

Figure 2-11 shows the private message that appears in the user2 client window:

**Figure 2-11:** Private Message in the Chat Application - user2 Window

12. Click the Logout button of the Chat Application - user2 window. A Confirm dialog box appears, as shown in Figure 2-12:

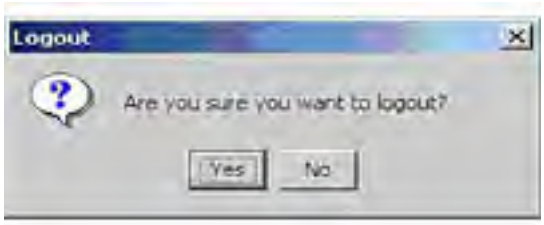


**Figure 2-12:** The Confirm Dialog Box

13. Click Yes on the Confirm dialog box. A message, user2 has left the chat session appears in the text pane of all the chat users, as shown in Figure 2-13:

**Figure 2-13:** The Chat Application - user1 Window

# Chapter 3: Creating a File Download Application

The New Input/Output (NIO) API supports the java.nio and java.nio.channels packages for buffer management and advance I/O file system. The NIO API allows you to use the java.rmi package for file transfer across the Java Virtual Machine (JVM). The java.nio package contains the ByteBuffer classes, which you can use to store the content of a file. The java.nio.channels package provides the FileChannel class, which you can use to read and write the file. The java.rmi package provides remote interface to call the methods from the remote client.

This chapter explains how to develop a File Download application. The application uses the above-mentioned NIO and Remote Method Invocation (RMI) packages to download files from the remote machines.

## Architecture of the File Download Application

The File Download application allows an end user to view a list of files stored at a specific location on the file server. The end user can select a file from the list and download it at the specified location.

The File Download application uses the following files:

- `FileRemote.java`: Creates a remote interface that declares the remote method for the application.
- `FileRemoteImpl.java`: Creates an implementation file that defines the remote methods declared in the remote interface.
- `FileInfo.java`: Creates a class that contains details, such as name, path, and size, of a particular file.
- `FileServer.java`: Creates a file server that binds the remote objects to the RMI registry. As a result, a client can invoke the object from the remote location.
- `FileClient.java`: Creates a user interface for the File Download application. The user interface helps an end user display a list of files stored in the file server in a tabular form. This interface also allows the end user to select and download a particular file.
- `ProgressTest.java`: Creates a Download Status dialog box that contains a progress bar. The progress bar indicates the percentage of file that has been downloaded.

Figure 3-1 shows the architecture of the File Download application:

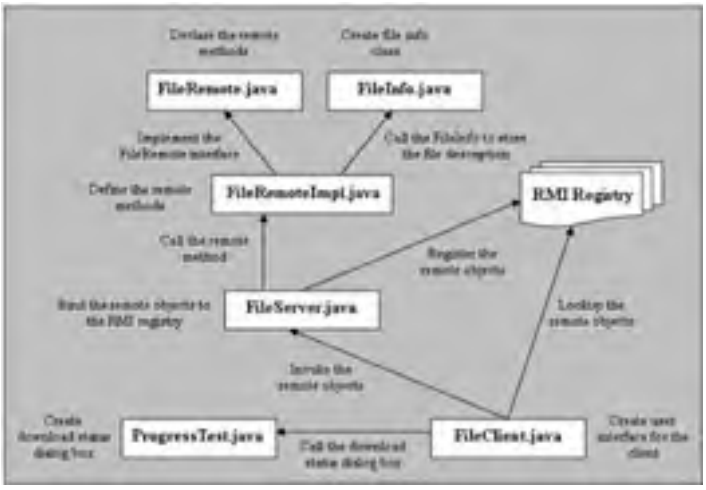


**Figure 3-1:** Architecture of the File Download Application

In the File Download application, the FileClient.java file calls the FileServer.java file to invoke the remote methods to display a file list and download a specified file. The FileServer.java file calls the FileRemoteImpl.java file, which finds and returns a list of files available at the specified location to the FileClient.java file. To download a particular file, the FileRemoteImpl.java file reads and transfers that file to the client computer. The FileRemoteImpl.java file implements the FileRemote.java file, which creates a remote interface to declare the remote method. The FileRemoteImpl.java file calls the FileInfo.java file to store the file description. The FileClient.java file calls the ProgressTest.java file to open the Download Status dialog box.

# Creating the Remote Interface

To create the client-server application using RMI, you need to create a remote interface that declares the business logic of the application. The FileRemote.java file is a remote interface for the File Download application that defines the prototypes of the methods implemented by the FileRemoteImpl.java file.

Listing 3-1 shows the contents of the FileRemote.java file:

**Listing 3-1: The FileRemote.java File**

```
/* Imports java.rmi package classes. */
import java.rmi.Remote;
import java.rmi.RemoteException;
/* Imports java.util package classes. */
import java.util.Vector;
/*
Interface FileRemote - This interface declares the remote methods.
Methods:
downloadFile() - Downloads the file from a remote location.
displayList() - Displays the list of files stored in the server to the client.
*/
public interface FileRemote extends Remote
{
   /* Declares downloadFile() method. */
   public byte[] downloadFile(String filename) throws RemoteException;
   /* Declares displayList() method. */
   public Vector displayList() throws RemoteException;
}
```

Download this Listing.

In the above code:

- The FileRemote interface defines the methods to perform various operations of the File Download application. These methods include displayList() and downloadFile().
- The displayList() method displays the file list to the client machine.
- The downloadFile() method downloads the selected file from the remote machine.

# Creating File Descriptor

The FileInfo.java file creates a file descriptor class that stores the description of a particular file.

Listing 3-2 shows the content of the FileInfo.java file:

**Listing 3-2: The FileInfo.java File**

```
/* Imports the java.io package class. */
import java.io.Serializable;
/*
class FileInfo - This class stores the file description, such as file index, file name, and file siz
Fields:
fileIndex - Contains the file name.
fileName - Contains the file path.
fileSize - Contains the file size.
*/
public class FileInfo implements Serializable
{
   /* Declares the objects of String class. */
   String fileIndex;
   String fileName;
   String fileSize;
   /* Defines default constructors. */
   public FileInfo(String fileIndex, String fileName, String fileSize)
   {
      this.fileIndex = fileIndex;
      this.fileName = fileName;
      this.fileSize = fileSize;
   }
}
```

Download this Listing.

In the above code, the FileInfo() constructor creates the file descriptor class. The fileName, filePath, and fileSize strings contain the file name, file location, and file size, respectively.

# Creating the Implementation File

The implementation file implements all the business methods that are declared in the remote interface. FileRemoteImpl.java is an implementation file that defines all the methods that are declared in the remote interface.

Listing 3-3 shows the content of the FileRemoteImpl.java file:

**Listing 3-3: The FileRemoteImpl.java**

```
/* Imports java.io package classes.*/
import java.io.*;
import java.io.File;
/* Imports java.rmi package classes. */
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
/* Imports java.nio package classes. */
import java.nio.*;
import java.nio.charset.*;
import java.nio.channels.*;
/* Imports java.util package classe. */
import java.util.Vector;
/*
Class FileRemoteImpl - This is the implementation class that implements all
the methods declared in the remote interface.
Fields:
name - Stores the name.
fi - Represents the object of FileInfo class.
Methods:
downloadFile() - This method is called when the end user clicks the window close button.
displayList() - This method is invoked when an end user selects any command from the menu bar.
*/
public class FileRemoteImpl extends UnicastRemoteObject implements FileRemote
{
   private String name;
   public FileInfo fi;
   /* Defines default constructor. */
   public FileRemoteImpl(String str) throws RemoteException
   {
      super();
      name = str;
      }
      /*
      downloadFile() - Defines the downloadFile() method.
      Parameter: filename - Represents the name of the file.
      Return Value: bufferFile[] - Represents the byte array that contains the file data.
      */
         public byte[] downloadFile(String filename)
      {
         /* Declares object of FileInputStream class. */
         FileInputStream fin;
         /* Declares object of FileChannel class. */
         FileChannel fchan;
         /* Declares object of ByteBuffer class. */
         ByteBuffer buff;
         long fsize;
         String str;
         byte bufferFile[] = null;
         try
         {
            /* Creates instance of FileInputStream class. */
            fin = new FileInputStream(filename);
            /* Gets the channel from file input stream. */
            fchan = fin.getChannel();
            /* Retrieves the size of the file channel. */
            fsize = fchan.size();
            /* Allocates the size of byte buffer. */
            buff = ByteBuffer.allocate((int)fsize);
            /* Reads the file from the channel to buffer. */
            fchan.read(buff);
            /* Rewinds the buffer. */
            buff.rewind();
            /* Creates a byte buffer of size equals to the file size. */
            bufferFile = new byte[(int)fsize];
            for(int i=0; i<(int)fsize; i++)
            {
               /* Stores the data into a byte array. */
               bufferFile[i] = buff.get();
            }
         }
         catch(Exception e)
         {
            System.out.println("Error: " + e);
          }
         /* Returns the byte array. */
```

```
            return(bufferFile);
         }
      /*
      displayList() - Defines the displayList() method.
      Parameter: NA
      Return Value: v - Represents a vector that contains the list of files and their details.
      */
      public Vector displayList() throws RemoteException
      {
         /*
         Creates and initializes the object of the Vector class.
         */
         Vector v = new Vector();
         /* Creates objects of the String class. */
         String SNo = "";
         String fileName = "";
         String size = "";
         /* Creates an object of the String array. */
         String files[] = null;
         int count =1;
         try
         {
            /* Creates and initialize the object of the File class. */
            File folder = new File("C:/CodeBook/NIO/FileTransfer/Server");
            /* Stores the list of files in the string array. */
            files = folder.list();
            for (int i=0; i<files.length; i++)
            {
               /* Creates and initialize the object of the File class. */
               File file = new File(folder.getAbsolutePath() + File.separator + files[i]);
               /*
               Checks whether the file object is a File type or a Directory type.
               */
               if (file.isFile())
               {
                  /* Stores the serial number. */
                  SNo = String.valueOf((count));
                  /* Stores the file name. */
                  fileName = file.getName();
                  /* Stores the file size. */
                  size = String.valueOf(file.length());
                  /* Initializes the object of the FileInfo class. */
                  fi = new FileInfo(SNo, fileName, size);
                  /*
                  Adds the object of the FileInfo class at the end of the vector.
                  */
                  v.addElement(fi);
                  /* Increments the counter by 1. */
                  count++;
               }
            }
         }
         catch(Exception e)
         {
            e.printStackTrace();
            System.out.println("Error! " + e);
         }
/* Returns the vector. */
         return v;
      }
}
```

Download this Listing.

In the above code, the FileRemoteImpl class creates an instance of the FileInfo class to store the file description. The FileRemoteImpl class defines the following remote methods:

- `downloadFile()`: Uses the filename as a parameter and returns the byte array of that file. This method then creates the instance of the FileInputStream, FileChannel, and ByteBuffer classes. Next, the downloadFile() method initializes the instance of the FileInputStream class and calls the getChannel() method of the FileChannel class. The downloadFile() method then initializes the object of the ByteBuffer class and allocates the size of the byte buffer. Next, the downloadFile() method reads the bytes from the file channel to the byte buffer and rewinds the bytes in the byte buffer. Finally, the downloadFile() method initializes a byte array called bufferFile, puts the values from the byte buffer into bufferFile, and returns the bufferFile byte array.
- `displayList()`: Returns the list of files in the form of a vector. This method creates and initializes an object of the Vector class. The displayList() method then initializes the SNo, fileName, and size string variables and creates a file object using the specified directory location. Next, this method calls the list() method of the File class to retrieve a list of files and subdirectories available within the specified directory location. The displayList() method then stores this list in the files[] String array. The displayList() method checks whether the file object is of File type or Directory type. If the file object is of File type, the displayList() method retrieves the name, path, and size from that file. The displayList() method then initializes an object of the FileInfo class using the SNo, fileName, and size variables. Next, this method adds the file descriptor object to a vector and increments the counter by one. Finally, the displayList() method returns the file list vector to the client machine.

# Creating the File Server

The FileServer.java file contains the main function of the file server. The file server binds the remote object to the RMI registry. So, any authorized client can access the remote objects.

Listing 3-4 shows the content of the FileServer.java file:

**Listing 3-4: The FileServer.java**

```
/* Imports java.io package classes. */
import java.io.*;
/* Imports java.rmi package classes. */
import java.rmi.*;
/*
class FileServer - Creates the RMI server that registers all the remote objects.
Method:
main() - Starts the RMI server and registers the remote objects. */
public class FileServer
{
   public static void main(String args[])
   {
      /* Sets the RMI security manager. */
      System.setSecurityManager(new RMISecurityManager());
      try
      {
         /* Creates the remote object. */
         FileRemote f = new FileRemoteImpl("FServer");
         /* Binds the remote object to the RMI registry. */
         Naming.rebind("FServer", f);
         System.out.println("Object is registered.");
         System.out.println("Now server is waiting for client request");
        }
      catch(Exception e)
      {
        e.printStackTrace();
        System.out.println("FileServer: " + e);
      }
   }
}
```

Download this Listing.

In the above code:

- The main() method creates an instance of the RMISecurityManager class. This method sets the security manager to the File Download application using the setSecurityManager() method.
- The main() method also creates an object of the FileRemoteImpl class and binds the remote object to the RMI registry using the rebind() method.

# Creating the User Interface for the File Download Application

The FileClient.java file helps create a user interface that contains a set of labels and buttons. This interface also has an empty tabular space where the file lists are displayed. End users can select a file from the list and download it.

Listing 3-5 shows the contents of the FileClient.java file:

**Listing 3-5: The FileClient.java File**

```
/* Imports java.io package class. */
import java.io.*;
import java.io.File;
/* Imports java.rmi package class. */
import java.rmi.*;
/* Imports java.nio package class. */
import java.nio.*;
import java.nio.charset.*;
import java.nio.channels.*;
/* Imports java.util package class. */
import java.util.Vector;
import java.util.StringTokenizer;
/* Imports java.awt package classes. */
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.Color;
import java.awt.*;
/* Imports java.awt.event package classes. */
import java.awt.event.*;
/* Imports javax.swing package classes. */
import javax.swing.*;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.LookAndFeel;
import javax.swing.UIManager;
/* Imports javax.swing.table package classes. */
import javax.swing.table.*;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.DefaultTableColumnModel;
/*
Class FileClient - Initializes the interface and loads all components like table,
label and button on the main window. This class is the main class of the application.
   Fields:
title - Contains the content of the title label.
panel - Contains the swing components.
scrollpane - Contains the table component.
table - Contains the list of files.
model - Represents the default table model.
listButton - Creates the Get File List button.
downloadButton - Creates the Download File button.
fileName - Stores the file name.
filePath - Stores the file path.
thread - Creates a progress bar thread.
th - Creates a downloading thread.
fName - Stores the file name.
fExt - Stores the file extension.
fout - Contains the file output stream.
fchan - Contains the file channel.
Methods:
addWindowListener() - This method is called when an end user clicks the window close button to close
actionPerformed() - This method is invoked when an end user selects any command from the menu bar.
display() - This method is called when an end user clicks the Get File List button to display the fi
download() - This method is called when an end user clicks the Download File button to download the
main() - This method creates the main window of the application and displays it.
*/
public class FileClient extends JFrame implements ActionListener , Runnable
{
/* Declares the object of the JLabel class. */
   JLabel title;
   /* Declares the object of the JPanel class. */
   JPanel panel;
   JPanel pan1;
   JPanel pan2;
```

```
JPanel pan3;
/* Declares the object of the JScrollPane class. */
JScrollPane scrollpane;
/* Declares the object of the JTable class. */
JTable table;
/* Create and initialize the object of the Vector class. */
Vector vRow = new Vector();
Vector vCol = new Vector();
/* Declares the object of the DefaultTableModel class. */
DefaultTableModel model;
/* Declare the objects of the JButton class. */
JButton listButton;
JButton downloadButton;
/* Declares the object of the GridBagLayout class. */
GridBagLayout gbl;
/* Declares the object of the GridBagConstraints class. */
 GridBagConstraints gbc;
static String str1, str2;
/* Declares the objects of the String class. */
String fileName;
String filePath;
Thread thread;
Thread th;
String fName;
String fExt;
String fname = null;
String fpath = null;
String path = null;
static String ipAdd;
/* Declares the object of the ProgressTest class. */
public ProgressTest pt;
/* Declares the object of the FileOutputStream class. */
FileOutputStream fout;
/* Declares the object of the FileChannel class. */
FileChannel fchan;
int count = 0;
int filesize, fs;
int value;
/* Declares the default constructor of the FileClient class. */
public FileClient()
{
   try
   {
      /* Sets windows look and feel to the application. */
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
   }
   catch(Exception e){}
   /* Sets the size of the window. */
   setSize(470, 515);
   /* Sets the reliability of the progress dialog box to false. */
   setResizable(false);
   /* Sets the title of the progress dialog box. */
   setTitle("File Download Application");
   /* Initializes the object of the GridBagLayout class. */
   gbl = new GridBagLayout();
   /* Sets the Layout */
   getContentPane().setLayout(gbl);
   /* Creates an object of the GridBagConstraints class. */
   gbc = new GridBagConstraints();
   /*
   Initializes the title label object and adds it to the 1,1,1,1 position with CENTER alignment.
   */
   gbc.gridx = 1;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   pan1 = new JPanel();
   pan1.setLayout(new BorderLayout());
   title = new JLabel(" File Download Application ");
   title.setFont(new Font("Verdana",Font.BOLD,20));
   pan1.add(title, BorderLayout.CENTER);
   getContentPane().add(pan1, gbc);
   /*
   Initializes listButton and downloadButton, adds these buttons to the panel. Next,
   adds it to the 1,2,1,1 position with CENTER alignment.
   */
   gbc.gridx = 1;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   pan2 = new JPanel();
   pan2.setLayout(new FlowLayout());
   listButton = new JButton("Get File List");
   listButton.addActionListener(this);
   pan2.add(listButton);
   downloadButton = new JButton("Download the File");
```

```
    downloadButton.setEnabled(false);
    downloadButton.addActionListener(this);
    pan2.add(downloadButton);
    getContentPane().add(pan2, gbc);
    /*
    Initializes the table label object and adds it to the 1,3,1,1 position with CENTER alignment.
    */
    gbc.gridx = 1;
    gbc.gridy = 3;
    gbc.gridwidth = 1;
    gbc.gridheight = 1;
    gbc.anchor = GridBagConstraints.CENTER;
    pan3 = new JPanel();
    pan3.setLayout(new BorderLayout());
    /*
    Adds the elements at the end of the column vector.
    */
    vCol.addElement("S. No.");
    vCol.addElement("File Name");
    vCol.addElement("Size (Bytes)");
    /* Initializes the object of the DefaultTableModel class to create a table model. */
    model = new DefaultTableModel(vRow, vCol);
    /* Initializes the object of JTable */
    table = new JTable(model);
    /* Sets single selection on the table */
    table.setSelectionMode(0);
    /*
    Creates instance of TableColumn class and set the width of the 0th column.
    */
    int vColIndex = 0;
    TableColumn col = table.getColumnModel().getColumn(vColIndex);
    int width = 50;
    col.setPreferredWidth(width);
    /*
    Creates instance of TableColumn class and set the width of the 1st column.
    */
    vColIndex = 1;
    col = table.getColumnModel().getColumn(vColIndex);
    width = 300;
    col.setPreferredWidth(width);
    /*
    Creates instance of TableColumn class and sets the width of the 2nd column.
    */
    vColIndex = 2;
    col = table.getColumnModel().getColumn(vColIndex);
    width = 100;
    col.setPreferredWidth(width);
    scrollpane = new JScrollPane(table);
    pan3.add(scrollpane, BorderLayout.CENTER);
    getContentPane().add(pan3, gbc);
    /*
    addWindowListener - Contains a windowClosing() method.
    windowClosing: Closes the main window It is called when the end user
    clicks the cancel button of the Window.
    Parameter: we- Object of the WindowEvent class.
    Return Value: NA
    */
    addWindowListener(new WindowAdapter()
    {
    public void windowClosing(WindowEvent we)
       {
          System.exit(0);
       }
    });
}
/*
actionPerformed() - This method is called when the end user selects any menu item from the menu b
Parameters:   ae - An ActionEvent object containing the details of the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
    /*
    This is executed when end user clicks the Get File List button.
    */
    if(ae.getSource() == listButton)
       {
          /* Calls the display method. */
          display();
          /* Sets the caption of the listButton to "Refresh". */
          listButton.setText("Refresh");
          /* Enables the download button. */
          downloadButton.setEnabled(true);
    }
    /*
    This is executed when end user clicks the Download File button.
    */
    else if(ae.getSource() == downloadButton)
```

```
        {
           int n = table.getSelectedRow();
           if ( n != -1)
           {
              /* Retrieves the value of specified column. */
              Object obj1 = table.getValueAt(n, 1);
              Object obj2 = table.getValueAt(n, 2);
              fname = (obj1).toString();
              fs = Integer.parseInt(obj2.toString());
              filesize = fs;
              /* Opens a Confirm dialog box. */
              value = JOptionPane.showConfirmDialog(null, "Are you sure you want to download the file?
              "Confirm", JOptionPane.YES_NO_OPTION);
              if(value == 0)
              {
                 count = 1;
                 /* Initializes a new thread. */
                 th = new Thread(this);
                 /* Starts the thread. */
                 th.start();
              }
              else if(value == 1)
              {
              }
           }
           else
           {
              /* Opens the error messages. */
              JOptionPane.showMessageDialog(null, "You must select a file before downloading!",
              "Message", JOptionPane.ERROR_MESSAGE);
           }
        }
     }
     /*
     display() - This method is called when the end user clicks the List the
     File button of the File Download Application window.
     Parameters:   NA
     Return Value: NA
     */
     public void display()
      {
        /* Creates and initialize the object of the vector class. */
        Vector vec = new Vector();
        /* Sets the number of rows in a table model to ZERO. */
        model.setRowCount(0);
        /* Creates the object of the FileInfo class. */
        FileInfo obj;
        try
        {
           String str = "rmi://"+ipAdd+"/FServer";
           /*
           Creates an instance of the FileRemote interface that looks up the remote
           object from the specified location.
           */
           FileRemote f = (FileRemote)Naming.lookup(str);
           /*
           Calls the displayList() method that returns the list of files stored in the File server.
           */
           vec = f.displayList();
           if((vec!=null) && (vec.size()>0))
           {
           for(int i=0; i<vec.size(); i++)
              {
                 /* Initializes the vRow object of the Vector class. */
                 vRow = new Vector();
                 /* Retrieves the value from the ith location. */
                 obj = (FileInfo)vec.elementAt(i);
                 /* Adds the fileIndex at the end of the vector. */
                 vRow.addElement(obj.fileIndex);
                 /* Adds the fileName at the end of the vector. */
                 vRow.addElement(obj.fileName);
                 /* Adds the fileSize at the end of the vector. */
                 vRow.addElement(obj.fileSize);
                 /* Adds the row in the table model. */
                 model.addRow(vRow);
              }
           }
        }
        catch(Exception e)
        {
           System.out.println("Vector Error:" + e);
        }
     }
  /*
  download() - This method is called when the end user clicks the Download
  File button of the File Download Application window.
  Parameters:
  fileName - Contains the name of the file.
```

```
filePath - Contains the path of the file.
Return Value: NA
*/
public void download(String fileName, String filePath)
    {
      /* Declares the object of the ByteBuffer class. */
      ByteBuffer buff;
      long fsize;
      try
      {
         String str = "rmi://"+ipAdd+"/FServer";
         /*
         Creates an instance of the FileRemote interface that looks up the remote
         object from the specified location.
         */
         FileRemote f = (FileRemote)Naming.lookup(str);
         /*
         Creates and initialize the object of the StringTokenizer class.
         */
         StringTokenizer st = new StringTokenizer(fileName, ".");
         /*
         Retrieves the value of the string before the delimiter "."
         */
         fName = st.nextToken();
         /*
         Retrieves the value of the string after the delimiter ".".
         */
         fExt = st.nextToken();
         path = filePath + "." + fExt;
         count = 2;
         /* Initializes the object of the Thread class. */
         thread = new Thread(this);
         /* Starts the thread. */
         thread.start();
         /*
         Calls the downloadFile() method that returns the content of the file.
         */
         byte[] data = f.downloadFile(fileName);
         try
         {
            /* Initializes the object of the FileOutputStream class. */
            fout = new FileOutputStream(path);
            /* Gets the channel from the output stream. */
            fchan = fout.getChannel();
            /* Allocates the size of the buffer. */
            buff = ByteBuffer.allocateDirect((int)(data.length));
            for(int i=0; i<data.length; i++)
            /* Reads the data from the byte array to the buffer. */
            buff.put(data[i]);
            /* Rewinds the buffer. */
            buff.rewind();
            /* Writes the buffer to the channel. */
            fchan.write(buff);
            /* Closes the channel. */
            fchan.close();
            /* Closes the output stream. */
            fout.close();
            }
         catch(Exception e)
         {
         }
      }
      catch(Exception e)
      {
         e.printStackTrace();
System.out.println("Error: " + e);
      }
   }
/*
run() - This method is called when the thread is started.
Parameters:   NA
Return Value: NA
*/
   public void run()
   {
      if(count == 1)
      {
         try
         {
            /* Creates and initialize the object of the JFileChooser class. */
            JFileChooser jfc = new JFileChooser();
            /* Sets the selection mode of the file chooser. */
            jfc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
            /* Displays the Save dialog box. */
            value = jfc.showSaveDialog(this);
            if(value == JFileChooser.APPROVE_OPTION)
            {
               try
```

```
                {
                   /* Selects the file. */
                   File file = jfc.getSelectedFile();
                   /* Retrieves the file path. */
                   fpath = file.getAbsolutePath();
                   /* Calls the downlaod() method. */
                   download(fname, fpath);
                }
                catch(Exception e)
                {
                   e.printStackTrace();
                   System.out.println("Error" + e);
                }
             }
          }
          catch(Exception e)
          {
             e.printStackTrace();
             System.out.println("Error" + e);
          }
       }
       else if(count == 2)
       {
          /* Initializes the object of ProgressTest class. */
          pt = new ProgressTest(filesize, path, th);
          /* Sets the title. */
          pt.setTitle("Download Complete.");
          /* Sets the label text. */
          pt.label.setText("Download Complete.");
          /* Sets the button caption. */
          pt.ok.setText(" OK ");
          /* Hides the progress bar. */
          pt.bar.setVisible(false);
       }
    }
    /*
    main() - This method creates the main window of the user interface and displays it.
    Parameters:
    args[] - Contains any command line arguments passed.
    Return Value: NA
    */
    public static void main(String args[])
    {
       ipAdd = args[0];
       /*
       Creates and initializes the object of the FileClient class.
       */
       FileClient fc = new FileClient();
       fc.show();
       }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the FileClient class. This class generates the main window of the File Download application, as shown in Figure 3-2:

**Figure 3-2:** The File Download Application User Interface

The application interface contains a tabular space where the list of files is displayed. The various methods defined in the above listing are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the button the end user clicks. If the end user clicks the Get File List button, the actionPerformed() method calls a display() method to display the list of files stored in the server. The actionPerformed() method also changes the button caption to Get File List to Refresh and enables the Download the File button. If an end user clicks the Download the File button, the actionPerformed() method retrieves the file name and size from the selected row of the table and opens a Confirm dialog box. After the end user clicks OK in the Confirm dialog box, the actionPerformed() method sets the value of the count variable to 1 and starts the newly initialized thread.
- `display()`: Creates and initializes the object of the vector class. This method then sets the row count of the table model to 0 and creates an instance of the FileInfo object to retrieve the description of a file. The display() method looks up the remote object from the specified location and calls the displayList() method that returns the list of files stored in the file server. Next, the display() method retrieves the value from the file list vector and adds the element to the table row, vRow vector. Finally, the display() method adds the row vector to the table model.
- `download()`: Takes the file name and file path as input parameters to download the specified file and declares the object of the ByteBuffer class. Next, the download() method looks up the remote object from the specified location and creates and initializes the object of the StringTokenizer class to get the file name extension. This method then sets the counter value to 2 and starts another new thread. The download() method also calls the downloadFile() method that returns the file in the form of a byte array. The download() method then initializes the object of the FileOutputStream class to write a file at a specified location and calls the getChannel() method of the FileChannel class. Next, this method initializes the object of the ByteBuffer class, allocates the size of the byte buffer, reads the bytes from the byte array to the buffer, and rewinds the byte buffer. Finally, the download() method writes the byte buffer to the file channel and close the file channel and output stream.
- `run()`: Runs the threads that are started in the application and checks the count value. If the value is 1, the run() method creates and initializes the object of the JFileChooser class to open the Save dialog box. This method then initializes the File object and gets the file using the getSelectedFile() method and the path using the getAbsolutePath() method of the File class. Next, the run() method calls the download() method. If the count value is 2, the run() method initializes the object of the ProgressTest class and opens the Download Status dialog box. After downloading, the run() method changes the title, label caption, and button caption to indicate that the file has been downloaded.



◄ PREVIOUS NEXT ►

# Creating a Download Status Dialog Box

The ProgressTest.java file helps create a Download Status dialog box for the File Dialog application. This dialog box displays a progress bar and a label to show the download status.

Listing 3-6 shows the contents of the ProgressTest.java file:

**Listing 3-6: The ProgressTest.java File**

```
/* Imports java.util package class. */
import java.util.*;
/* Imports java.awt package classes. */
import java.awt.*;
/* Imports java.awt.event package class. */
import java.awt.event.*;
/* Imports javax.swing package classes. */
import javax.swing.*;
/* Imports javax.swing.event package class. */
import javax.swing.event.*;
/* Imports java.io package class. */
import java.io.File;
/*
class ProgressTest - Creates a Download Status dialog box that indicates how much data is downloaded
Fields:
label - Contains the content of title label.
ok - Creates an OK button.
size - Contains the size of the file.
path - Contains the path where the end user wants to save the file.
th - Contains the instance of the downloading thread.
Method:
performTask() - This method is called when the ProgressBar is painted.
actionPerformed() - This method is invoked when the end user clicks the OK or Cancel button.
*/
public class ProgressTest extends JDialog implements ActionListener
{
   /* Declares the objects of the JLabel class. */
   JLabel label;
   JLabel label_1;
   JLabel label_2;
   /* Declares the object of the JButton class. */
   JButton ok;
   /* Declares the object of the JProgressbar class. */
   JProgressBar bar;
   /* Declares the object of the GridBagLayout class. */
   GridBagLayout gbl;
   /* Declare the object of the GridBagConstraints class. */
   GridBagConstraints gbc;
   /* Declares the objects of the string class. */
   String str;
   String path;
   /* Declares and initialize the size as integer. */
   int size = 100;
   /* Declares the object of the Thread class. */
   Thread th;
   int n = 0;
   /*
   ProgressTest() - This is the default constructor of the ProgressTest class.
   Parameter:
   size - Represents the file size.
   path - Represents the file path where the end user saves the file.
   th - Represents the instance of the Thread class.
   */
   public ProgressTest(int size, String path, Thread th)
   {
      this.size = size;
      this.path = path;
      this.th = th;
      /* Sets the size of the Download Status dialog box. */
      setSize(270, 150);
      /* Sets the visibility of the Download Status dialog box. */
      setVisible(true);
      /*
      Sets the reliability of the Download Status dialog box to false.
      */
      setResizable(false);
      /* Sets the title of the Download Status dialog box. */
      setTitle("Download Status");
      /* Initializes the object of the G   ridBagLayout class. */
      gbl = new GridBagLayout();
      /* Sets the Layout. */
      getContentPane().setLayout(gbl);
      /* Creates an object of the GridBagConstraints class. */
      gbc = new GridBagConstraints();
      /*
```

```
Initializes the label object and add it to the 1,1,1,1 position with WEST alignment.
*/
gbc.gridx = 1;
gbc.gridy = 1;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
label = new JLabel(" File is downloading");
getContentPane().add(label, gbc);
/*
Initializes a blank label object and add it to the 1,2,1,1 position with WEST alignment.
*/
gbc.gridx = 1;
gbc.gridy = 2;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
label_1 = new JLabel(" ");
getContentPane().add(label_1, gbc);
/*
Initializes an object of the JprogressBar class and adds it to the 1,3,1,1 position with CENTE
*/
gbc.gridx = 1;
gbc.gridy = 3;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
bar = new JProgressBar();
/* Sets the size of the progress bar. */
bar.setPreferredSize(new Dimension(225, 20));
bar.setMinimum( 0 );
bar.setMaximum( size );
bar.setValue( 0 );
bar.setBorderPainted(true);
bar.setStringPainted(true);
getContentPane().add(bar, gbc);
/*
Initializes a blank label object and adds it to the 1,4,1,1 position with WEST alignment.
*/
gbc.gridx = 1;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
label_2 = new JLabel(" ");
getContentPane().add(label_2, gbc);
/*
Initializes an object of the Button class and adds it to the 1,5,1,1 position with CENTER alig
*/
gbc.gridx = 1;
gbc.gridy = 5;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
ok = new JButton(" Cancel ");
ok.addActionListener(this);
getContentPane().add(ok, gbc);
n = 100;
/*
Starts the progress bar until the file is downloaded.
*/
for( int iCtr = 1; iCtr < size+1; iCtr +=n )
{
   /*
   Calls the performTask() method to insert a time delay.
   */
   performTask( iCtr );
   /* Updates the progress indicator and label. */
   label.setText( "Downloading "+ iCtr + " bytes of " + size + " bytes.");
   /* Creates an object of the Rectangle class that gets the label bound. */
   Rectangle labelRect = label.getBounds();
   labelRect.x = 0;
   labelRect.y = 0;
   /* Paints the label. */
   label.paintImmediately( labelRect );
   /* Sets the value to the progress bar. */
   bar.setValue( iCtr );
   /*
   Creates an object of the Rectangle class that gets the progress bar bound.
   */
   Rectangle progressRect = bar.getBounds();
   progressRect.x = 0;
   progressRect.y = 0;
   /* Paints the progress bar. */
   bar.paintImmediately( progressRect );
}
```

```
    }
    /*
    performTask() - This method provides a time delay for the progress bar creating loop.
    Paramter: ictr
    Return Value: NA
    */
    public void performTask( int iCtr )
    {
       Random random = new Random( iCtr );
       for( int i = 0; i < random.nextFloat() * 1000; i++ )
       {
          /* Runs the loop */
       }
    }
    /*
    actionPerformed() - This method is invoked when the end user clicks the button.
    Parameter: ae - an ActionEvent object containing the details of the event.
    Returns Value: NA
    */
    public void actionPerformed(ActionEvent ae)
    {
       String arg = (String)ae.getActionCommand();
       if(arg.equals(" Cancel "))
       {
          /* Stops the downloading thread. */
          th.stop();
          try
          {
             /* Creates instance of the File class. */
             File f = new File(path);
             /* Performs deletion until the file is deleted. */
             while(f.delete())
             {
                /* Delete the file. */
                f.delete();
             }
             /* Hides the Download Status dialog box. */
             this.setVisible(false);
          }
          catch(Exception e)
          {
             System.out.println("Error in I/O");
          }
       }
       else if(arg.equals(" OK "))
       {
          /* Hides the Download Status dialog box. */
          this.setVisible(false);
       }
    }
}
```

Download this Listing.

In the above code, the ProgressTest() constructor creates the Download Status dialog box, as shown in Figure 3-3:


**Figure 3-3:** The Download Status Dialog Box

If an end user clicks the Cancel button on the Download Status dialog box, the actionPerformed() method is invoked. This method stops the downloading thread. The actionPerformed() method then creates an instance of the File class and deletes the file that is being downloaded.

## Unit Testing

To test the File Download application:

1. 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;
   ```
2. 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;
   ```
3. Copy the FileRemote.java, FileRemoteImpl.java, FileInfo.java, FileServer.java, FileClient.java, and ProgressTest.java files to a folder on your computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. Generate the stub and skeleton files for the File Download application using the following command:
   ```
   rmic FileRemoteImpl
   ```
5. Start the RMI registry using the following command:
   ```
   start rmiregistry
   ```
6. Copy the FileRemote.class, FileRemoteImpl.class, FileInfo.class, FileServer.class, FileRemoteImpl_Stub.class, and FileRemoteImpl_Skel.class files to the folder, named Server, where your server machine is hosted. On the command prompt, use the cd command to move to the folder where you have copied the class files.
7. Run the server of the File Download application using the following command at the command prompt:
   ```
   java FileServer
   ```
8. Copy the FileRemoteImpl.class, FileInfo.class, FileRemoteImpl_Stub.class, FileClient.class, and ProgressTest.class files to the folder, named Client, where your client machine is hosted. On the command prompt, use the cd command to move to the folder where you have copied the class files.
9. Create a java.policy file to authenticate the client to access the remote object using the following command:
   ```
   Policytool
   ```
10. Run the client of the File Download application. You need to specify the IP address of the server on the command line. To execute the client, specify the following command at the command prompt:
    ```
    java FileClient 192.168.0.36
    ```
11. Click the Get File List button on the File Download Application window to display the list of files that are available for download. The File Download Application window with the file list appears, as shown in Figure 3-4:


**Figure 3-4:** Displaying File Download Application

12. Click the Download the File button without selecting any file from the table. An error message appears, as shown in Figure 3-5:


**Figure 3-5:** Error Dialog Box

13. Select a file that you want to download.
14. Click the Download the File button to start the file download process. A Confirm dialog box appears, as shown in Figure 3-6:


**Figure 3-6:** Confirm Dialog Box

15. Click OK in the Confirm dialog box. A Save dialog box opens, as shown in Figure 3-7:

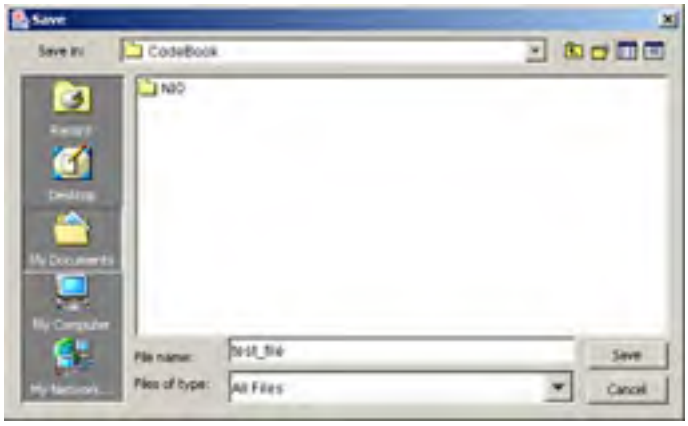


**Figure 3-7:** Save Dialog Box

16. Click the Save button of the Save dialog box. A Download Status dialog box opens, as shown in Figure 3-8:


**Figure 3-8:** Download Status Dialog Box

To stop the download, the end user can click Cancel. After download, a Download Complete dialog box opens, as shown in Figure 3-9:

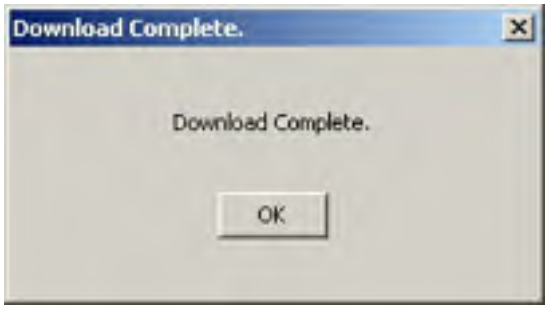


**Figure 3-9:** Download Complete Dialog Box



PREVIOUS NEXT

# Chapter 4: Creating a File Search Application

The New Input/Output (NIO) API supports the java.nio, java.nio.channels, and java.util.regex packages for buffer management, character conversion, and regular expression matching. The java.nio and java.nio.channels packages contain the File, FileChannel, and CharBuffer classes to read and buffer the contents of a file. The java.util.regex package provides the Pattern and Matcher classes to match regular expressions.

This chapter explains how to develop a File Search application, which uses the above-mentioned NIO packages to search for specific files.

## Architecture of the File Search Application

The File Search application enables an end user to search for files that are available at a specific location and contain a specific text pattern. The application also searches for all the instances of the specified text pattern in the files and displays the start position of the pattern to end users.

The File Search application uses the following files:

- `Search.java`: Creates a user interface that helps an end user specify the criteria to search for specific files and view the retrieved result
- `FileList.java`: Searches for files that are available at the specified location and contain the specified text pattern
- `Help.java`: Creates a window that lists the steps to use the File Search application

Figure 4-1 shows the architecture of the File Search application:

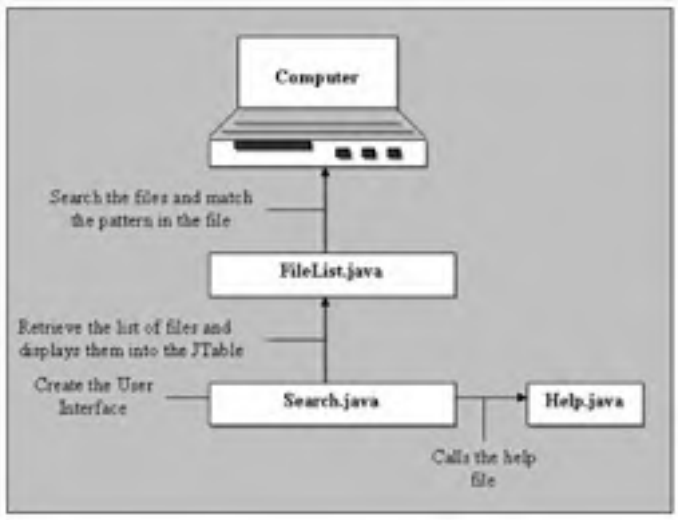


**Figure 4-1:** Architecture of the File Search Application

In the File Search application, the Search.java file calls the FileList.java file to search for files that are present at the specified location and contain the specified text pattern. The FileList.java file searches for these files and passes the search results to the Search.java file. The Search.java file then displays these results to the end user in a tabular format. Before starting the file search process, if an end user opts to view the application help file, the Search.java file calls the Help.java file to display the steps to use the File Search application.

# Creating the User Interface for the File Search Application

The Search.java file helps you create a user interface with a set of text boxes, labels, and buttons for the File Search application. End users can use this interface to specify a location and text pattern to search for specific files. The right pane of the File Search application interface contains an empty space where the file search results are displayed to the end user.

Listing 4-1 shows the contents of the Search.java file:

**Listing 4-1: The Search.java File**

```
/* Imports the required I/O classes. */
import java.io.File;
/* Imports the required Util classes. */
import java.util.Vector;
/* Imports the required AWT classes */
import java.awt.event.*;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.Color;
import java.awt.*;
/* Imports the required Swing classes */
import javax.swing.*;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.LookAndFeel;
import javax.swing.UIManager;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
/*
class Search - This is the main class of the application. This class initializes the
interface and loads all components, such as table, before displaying the result.
Methods:
listFiles() - This method searches for files that meet the specified search criteria.
Usage() - This method displays an error message.
Main() - This method creates the main window of the application and displays all the components and
*/
public class Search extends JFrame implements ActionListener, Runnable
{
   /* Declare objects of the JLabel class. */
   JLabel labelText;
   JLabel labelLook;
   JLabel labelSearch;
   JLabel labelResult;
   /* Declare objects of the JTextField class. */
   JTextField textText;
   JTextField textLook;
   /* Declare objects of the JButton class. */
   JButton search;
   JButton cancel;
   JButton browse;
   JButton help;
   /* Declare objects of the JPanel class. */
   JPanel panel;
   JPanel paneLeft;
   JPanel paneRight;
   JPanel pane1;
   JPanel pane2;
   JPanel pane3;
   JPanel pane4;
   JPanel pane5;
   JPanel p1;
   JPanel p2;
   JPanel p3;
   /* Declare object of the JScrollPane class. */
   JScrollPane scrollpane;
   /* Declare String objects. */
   static String strText;
   static String strLook;
   /* Declare an object of the JTable class. */
   JTable table;
   /* Declare and initialize a counter.*/
   int count = 0;
   int FLAG = 0;
   /*
```

```
Declare and initialize an object of the Object class.
*/
Vector vRow = new Vector();
Vector vCol = new Vector();
public FileList fl = new FileList(this);
public Help h;
/* Declare arrays of the String class.*/
DefaultTableModel model;
Thread thread;
int countFile;
/* Implement a constructor of the Search class. */
public Search()
{
   try
   {
      /*
      Initialize and set the look and feel of the application to Windows look and feel.
      */
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
   }
   catch(Exception e)
   {
      /*
      If an error occurs while loading the Windows look and feel, an error is
      displayed and the application is closed.
      */
      usage();
   }
   /* Set the size of the application frame. */
   setSize(800, 600);
   /* Set the Title of the application frame. */
   setTitle("File Search Utility");
   /*
   Initialize a panel, containing two subpanels, for the application frame.
   */
   panel = new JPanel();
   /* Set the layout of the frame as Grid layout. */
   panel.setLayout(new GridLayout(0, 2));
   /* Add the panel to the frame. */
   getContentPane().add(panel);
   /*
   Initialize the other two subpanels.
   Add the first subpanel (paneLeft) to the left side of the main panel.
   Add the second subpanel (paneRight) to the right side of the main panel.
   */
   paneLeft = new JPanel();
   paneRight = new JPanel();
   panel.add(paneLeft);
   panel.add(paneRight);
   /*
   Set the layout of the left subpanel as Border layout.
   Initialize a new Grid panel.
   Set the layout of the new panel as Grid layout.
   Add this panel to the left subpanel.
   */
   paneLeft.setLayout(new BorderLayout());
   pane5 = new JPanel();
   pane5.setLayout(new FlowLayout(FlowLayout.RIGHT));
   help = new JButton("Help");
   help.addActionListener(this);
   pane5.add(help);
   paneLeft.add(pane5, BorderLayout.SOUTH);
   /*
   Set the layout of the left subpanel as Border layout.
   Initialize a new Grid panel.
   Set the layout of the new panel as Grid Layout.
   Add this panel to the left subpanel.
   */
   pane3 = new JPanel();
   pane3.setLayout(new GridLayout(6,0));
   paneLeft.add(pane3, BorderLayout.NORTH);
   /*
   Initialize a new panel.
   Set the layout of this panel as Flow layout.
   Initialize a new label and add this label to the panel.
   Add this panel to the first grid of the new Grid panel.
   */
   p1 = new JPanel();
   p1.setLayout(new FlowLayout(FlowLayout.LEFT));
   labelLook = new JLabel("Search for file(s) in:");
   p1.add(labelLook);
   labelLook.setFont(new Font("Verdana",Font.BOLD,12));
   pane3.add(p1);
   /*
   Initialize a new panel.
   Set the layout of this panel as Flow layout.
   Initialize a text field and add it to the panel.
   Initialize a button and add it to the panel.
```

```
Add this panel to the second grid of the new Grid panel.
*/
panel = new JPanel();
panel.setLayout(new FlowLayout(FlowLayout.LEFT));
textLook = new JTextField(25);
browse = new JButton("Browse");
browse.addActionListener(this);
panel.add(textLook);
panel.add(browse);
textLook.setFont(new Font("Verdana",Font.PLAIN,12));
pane3.add(panel);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize a new label and add this label to the panel.
Add this panel to the third grid of the new Grid panel.
*/
p2 = new JPanel();
p2.setLayout(new FlowLayout(FlowLayout.LEFT));
labelText = new JLabel("Containing text:");
p2.add(labelText);
labelText.setFont(new Font("Verdana",Font.BOLD,12));
pane3.add(p2);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize a text field and add it to the panel.
Add this panel to the fourth grid of the new Grid panel.
*/
p3 = new JPanel();
p3.setLayout(new FlowLayout(FlowLayout.LEFT));
textText = new JTextField(33);
p3.add(textText);
textText.setFont(new Font("Verdana",Font.PLAIN,12));
pane3.add(p3);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize buttons and add these buttons to the panel.
Add this panel to the fifth grid of the new Grid panel.
*/
pane2 = new JPanel();
pane2.setLayout(new FlowLayout(FlowLayout.LEFT));
search = new JButton("Search");
cancel = new JButton("Cancel");
pane2.add(search);
search.addActionListener(this);
pane2.add(cancel);
cancel.addActionListener(this);
pane3.add(pane2);
/*
Initialize a new panel.
Set the layout of this panel as Flow layout.
Initialize buttons and add these buttons to the panel.
Add this panel to the fifth grid of the new Grid panel.
*/
pane4 = new JPanel();
pane4.setLayout(new FlowLayout(FlowLayout.LEFT));
labelSearch = new JLabel("Search Result(s): ");
labelSearch.setFont(new Font("Verdana",Font.BOLD,12));
labelResult = new JLabel("0 files.");
labelResult.setFont(new Font("Verdana",Font.BOLD,12));
pane4.add(labelSearch);
pane4.add(labelResult);
pane3.add(pane4);
/*
Set the layout of the right subpanel as Border layout.
Initialize an object of the table.
Initialize an object of the scroll panel.
Add the table to the scroll panel.
Add this scroll panel to the right subpanel.
*/
paneRight.setLayout(new BorderLayout());
model = new DefaultTableModel();
vCol.addElement("File Name");
vCol.addElement("Start Position");
vCol.addElement("Path");
vCol.addElement("Size");
model = new DefaultTableModel(vRow, vCol);
table = new JTable(model);
scrollpane = new JScrollPane(table);
paneRight.add(scrollpane, BorderLayout.CENTER);
/* Pack the components of the frame. */
doLayout();
pack();
/*
addWindowListener: It contains the windowClosing() method.
windowClosing: It is called when the end user clicks the Cancel button on the window.
```

```
   It closes the main window.
   we parameter: An object of the WindowEvent class.
   Return Value: NA
   */
   addWindowListener(new WindowAdapter()
   {
      public void windowClosing(WindowEvent we)
      {
         System.exit(0);
      }
   });
}
/*
actionPerformed(): This method is called when the end user clicks the Browse,
Search, or Cancel button.
ae parameter: An ActionEvent object that contains information about the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   /*
   This is executed when an end user clicks the Search button.
   */
   if(ae.getSource() == search)
   {
      countFile = 0;
      count = 0;
      FLAG = 0;
      model.setNumRows(0);
      strText = textText.getText();
      strLook = textLook.getText();
      /*
      If both the text field have the required text, the actionPerformed() method creates
      a new instance of the Thread class, which further calls the listFiles() method.
      */
      if(strText.equals("") == false && strLook.equals("") == false)
      {
         try
         {
            thread = new Thread(this);
            thread.start();
         }
         catch(Exception e)
         {
      System.out.println("Error!" + e);
         }
      }
      /*
      Otherwise, display an Error message box.
      The Error message box is displayed by calling the showMessageDialog()
      method of the JOptionPane class.
      showMessageDialog(): It is called by the system automatically, when required.
      Parameter:
      Parent Name: Name of the parent frame.
      Display Message: Text to be displayed on the message box.
      Title: Sets the title of the message box.
      Message Box Option: Type of message box.
      Return Value: NA
      */
      else
      {
         JOptionPane.showMessageDialog(this, "You must enter the searching text.",
         "Alert Message", JOptionPane.WARNING_MESSAGE);
      }
   }
   /*
   This is executed when the end user clicks the Search button.
   */
   else if (ae.getSource() == browse)
   {

      try
      {
         /* Initialize an object of the JFileChooser class. */
         JFileChooser jfc = new JFileChooser();
         /* Set the mode of the FileChooser box. */
         jfc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
         jfc.showOpenDialog(this);
         /*
         Get the selected file using the getSelectedFile() method.
         */
         File file = jfc.getSelectedFile();
         /*
         Retrieve the value of the absolute path of the file. Store the value into a string.
         */
         String str = file.getAbsolutePath();
         textLook.setText(str);
      }
```

```
            catch(Exception e)
            {
               +System.out.println("Error" + e);
            }
         }
         /*
         This is executed when an end user clicks the Search button.
         */
         else if (ae.getSource() == cancel)
         {
            /*
            This hides and closes all the components of the application.
            */
            search.setEnabled(true);
            FLAG = 1;
         }
         /*
         This is executed when an end user clicks the Help button.
         */
         else if (ae.getSource() == help)
         {
            /* It shows the help utility. */
            h = new Help();
            h.show();
         }
      }
      public void run()
      {
         fl.listFiles(strLook);
      }
      /*
      This is the main method that creates an instance of the Search class and shows the main frame.
      */
      public static void main(String[] args)
      {
              Search s = new Search();
         s.show();
      }
      /*
      A utility method to display the invocation syntax and exit.
      */
      public static void usage()
      {
           System.err.println("Error");
           System.exit(1);
       }
}
```

Download this Listing.

In the above listing, the main() method creates an instance of the Search class. This class generates the main window of the File Search application, as shown in Figure 4-2:

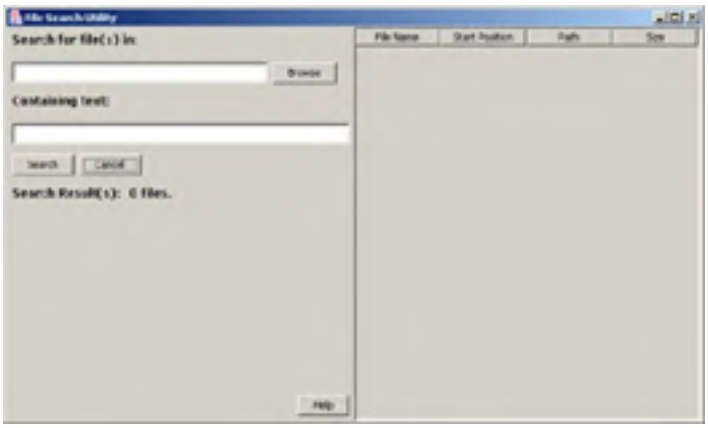

**Figure 4-2:** The File Search Application User Interface

The text boxes in the above figure enable an end user to specify the location and the text pattern to search for specific files. The empty space in the right pane of the window displays the results of the file search.

When an end user clicks any button on the File Search Utility window, the File Search application invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate class, based on the button that the end user clicks.

When an end user clicks the Browse button adjacent to the Search for file(s) in text box, the actionPerformed() method creates an instance of the JFileChooser class to open the File dialog box. This dialog box enables the end user to browse to a specific location, where the File Search application should search for the desired files.

After specifying the search criteria, when an end user clicks the Search button, the actionPerformed() method creates an instance of the Thread class and calls the start() method of this class. The start() method further calls the run() method of the Thread class, which calls the listFiles() method to search for files that meet the specified criteria.

The Search.java file declares an object of the JTable class. This object creates a tabular format to display the search results, returned by the listFiles() method, to the end user.

The Cancel button on the File Search Utility window helps an end user stop the file search process before it is complete. When an end user clicks the Cancel button, the actionPerformed() method sets the value of the FLAG variable, declared in the Search class, to 1 and stops the running instance of the Thread class.

If an end user clicks the Help button, the actionPerformed() method creates an instance of the Help class and calls the show() method of this class. The show() method opens the Help window, which displays the sequence of steps to search for files using the File Search application.



PREVIOUS NEXT

# Implementing the Search Functionality

The FileList.java file implements the core functionality of the File Search application. This file reads the location and text pattern that the end user specifies on the File Search Utility window and retrieves information, such as the number of files meeting the search criteria, file names, start position of the specified text pattern within these files, and file sizes. Finally, the FileList.java file passes this information to the Search.java file, which displays this search information to the end user in a tabular format.

Listing 4-2 shows the contents of the FileList.java file:

**Listing 4-2: The FileList.java File**

```
/* Imports the required I/O classes. */
import java.io.FileInputStream;
import java.io.File;
import java.io.IOException;
/* Imports the required NIO classes. */
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.charset.Charset;
import java.nio.charset.CharsetDecoder;
import java.nio.channels.FileChannel;
import java.nio.charset.UnsupportedCharsetException;
/* Imports the required Util classes. */
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.regex.PatternSyntaxException;
import java.util.Vector;
/* Imports the required Swing classes */
import javax.swing.JOptionPane;
/*
class FileList - This class is the subclass of the application. It displays the list of files in a t
*/
public class FileList
{
   /* Initialize the object of Search class. */
   Search se;
   int option;
   int count = 1;
   /*
   Declare the default constructor of the FileList class.
   */
   public FileList(Search se)
   {
      this.se = se;
   }
   /*
   listFiles() - This method is called when the end user clicks the Search button of the application
   Parameter:
   dirname: It is a string type that contains the name of the directory.
   Return Value - NA
   */
   public void listFiles(String dirname)
   {
      String encodingName = "UTF-8";
      /* Set the flag value to CASE INSENSITIVE. */
      int flags = Pattern.CASE_INSENSITIVE;
      try
      {
         /*
         Initialize the instance of the Charset class and register the class with the
         encoding name.
         */
         Charset charset = Charset.forName(encodingName);
         /*
         Initialize the pattern object and compile it with the search pattern specified
         by the end user.
         */
         Pattern pattern = Pattern.compile(se.strText, flags);
         try
         {
            /*
            Create a file object that contains the names of all the files and directories.
            */
            File folder = new File(dirname);
            /*
            Store the list of files and directories in a String array.
            */
            String files[] = folder.list();
            for (int i=0; i<files.length-1; i++)
            {
               File file = new File(folder.getAbsolutePath() + File.separator + files[i]);
               /*
               Check if the file object is a file or a directory.
               */
```

```
            if (se.FLAG == 0)
            {
               if (file.isFile())
               {
                  /* This will store the complete text of the file. */
                  CharBuffer chars;
                  try
                  {
                     /*
                     Handle the errors of each file locally and open a file channel to the named
                     */
                     FileInputStream stream = new FileInputStream(file);
                     FileChannel f = stream.getChannel();
                     /*
                     Calls the map() method of the FileChannel class, which returns an
                     object of the ByteBuffer class. The map() method uses the
                     MapMode.READ_ONLY attribute of the FileChannel class to open the files.
                     */
                     ByteBuffer bytes = f.map(FileChannel.MapMode.READ_ONLY, 0, f.size( ));
                     /*
                     We can close the file once it is mapped to the memory.
                     Closing the stream closes the file channel also.
                     */
                     stream.close( );
                     /*
                     Decode the entire ByteBuffer into one big CharBuffer.
                     */
                     chars = charset.decode(bytes);
                  }
                  catch(IOException e)
                  {
                     /*
                     File not found. Print an error message. Move to the next file.
                     */
                     System.err.println(e);
                     continue;
                  }
                  /*
                  A Matcher holds the state of a given pattern and text. Start matching the text.
                  */
                  Matcher matcher = pattern.matcher(chars);
                  while(matcher.find())
                  {
                     try
                     {
                        se.vRow = new Vector();
                        /*
                        Add elements to the row vector.
                        */
                        se.vRow.addElement(file.getName());
                        se.vRow.addElement(Integer.toString(matcher.start()));
                        se.vRow.addElement(file.getParent());
                        se.vRow.addElement(Long.toString(file.length()) + " bytes");
                        /*
                        Add a row to the table model.
                        */
                        se.model.addRow(se.vRow);
                        se.countFile = 1;
                        se.labelResult.setText(Integer.toString(se.model.getRowCount()) + "file(s
               }
               catch(Exception e)
               {
                  System.out.println("Error!" + e);
               }
            }
            }
            else
            /*
            If the file object is a directory, it calls the listFiles() method again.
            */
            listFiles(file.getAbsolutePath());
         }
         else if (se.FLAG == 1)
         {
            break;
         }
      }
   }
   catch (Exception ex)
   {
      /*  Print Error message. Return. */
      ex.printStackTrace();
      return;
      }
   }
   catch(UnsupportedCharsetException e)
   {
      /*
```

```
            Bad encoding name. Print exception name.
            */
            System.err.println("Unknown encoding: " + encodingName);
         }
         catch(PatternSyntaxException e)
         {
            /*
            Bad pattern-matching. Print exception name.
            */
            System.err.println("Syntax error in search pattern: " + e.getMessage());
         }
         if(se.countFile == 0)
         {
            JOptionPane.showMessageDialog(null, "No file found!", "Alert Message",
            JOptionPane.WARNING_MESSAGE);
            se.countFile = 1;
         }
      }
}
```

Download this Listing.

The above listing creates an instance of the FileList class, which accepts the Search class as an input parameter. When an end user clicks the Search button on the File Search Utility window, the listFiles() method is invoked. The dirname parameter of this method stores the file location specified by the end user. The listFiles() method creates a new instance of the Charset class and registers it with Unicode Transformation Format-8 (UTF-8). Next, the method creates an instance of the Pattern class, initializes the pattern object, and stores the text pattern specified by the end user in this pattern object. The listFiles() method creates a file object using the directory location specified by the end user. Next, the listFiles() method calls the list() method of the File class to retrieve a list of files and subdirectories available within the specified directory location and stores this list in the files[ ] String array.

The listFiles() method now iterates through the files[ ] String array to check for the type of files. If the file is of the type File class, the listFiles() method creates a new instance of the FileInputStream class and calls the getChannel() method of this class. The getChannel() method returns an object of the FileChannel class. Next, the listFiles() method calls the map() method of the FileChannel class, which returns an object of the ByteBuffer class. The listFiles() method then calls the close() method of the FileInputStream class to close the file after it is mapped to the memory.

After mapping the files available at the specified location and containing the specified text pattern to the memory, the listFiles() method calls the matcher() method of the Pattern class. The Pattern class contains the text pattern specified by the end user. The matcher() method returns an object of the Matcher class. The listFiles() method calls the find() method of the Matcher class to obtain information about the files that contain the specified text pattern. The search result, which contains the total number of files along with file information, such as file name and file size, is then returned to the Search class to be displayed to the end user.

# Creating the Help File

The Help.java file enables you to create the Help window of the File Search application. This window displays the steps to use the File Search application to the end user.

Listing 4-3 shows the contents of the Help.java file:

**Listing 4-3: The Help.java File**

```
/*
Import the packages to use their classes in this class.
*/
import java.awt.event.*;
import java.awt.FlowLayout;
import java.awt.Font;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.JDialog;
/* A class for the Help file. */
public class Help extends JDialog implements ActionListener
{
   JLabel label1;
   JTextArea area;
   JScrollPane sp;
   JPanel pane;
   String str;
   String text;
   JButton ok;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   /*
   Define the default constructor of the Help class.
   */
   public Help()
   {
      /*
      Set the size. Initialize the panel and set the layout. Add this panel to the frame.
      */
      setTitle("Help");
      setSize(460, 260);
      setVisible(true);
      setResizable(false);
      gbl = new GridBagLayout();
      getContentPane().setLayout(gbl);
      gbc = new GridBagConstraints();
      /*
      Initialize the Label. Add the label to the panel.
      */
      gbc.gridx = 1;
      gbc.gridy = 1;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.CENTER;
      label1 = new JLabel("File Search Utility Help");
      Font f = new Font("Veradan",Font.BOLD,14);
      /*
      Set the Font style and size of the label caption.
      */
      label1.setFont(f);
      getContentPane().add(label1, gbc);
      /*
      Create a text area and set the text.
      Set the font as Verdana in the text area.
      Set editable false in the text area.
      Add this text area to the scroll pane.
      */
      gbc.gridx = 1;
      gbc.gridy = 2;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.CENTER;
      area = new JTextArea(10, 40);
      text = "The steps to search for a file are:\n\n"
      + "1. Click the Browse button to locate a directory.\n"
      + "2. Enter the text that you want to search in the Containing text text box.\n"
      + "3. Click the Search button to start the search process.\n"
      + "4. Click the Cancel button to stop the search process before it is complete.";
      Font f1 = new Font("Veradan",Font.PLAIN,12);
```

```
        area.setFont(f1);
        area.setText(text);
        area.setLineWrap(true);
        area.setWrapStyleWord(true);
        area.setEditable(false);
        sp = new JScrollPane(area);
        getContentPane().add(sp, gbc);
        /*
        Initialize the OK button. Add action listener.
        */
        gbc.gridx = 1;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        ok = new JButton(" OK ");
        ok.addActionListener(this);
        getContentPane().add(ok, gbc);

    }
    /*
    actionPerformed() This method is called when the end user clicks the OK button.
    Parameters:
    ae - An ActionEvent object containing information about the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent ae)
    {
        /*
        This is executed when the end user clicks the Search button.
        */
        if(ae.getSource() == ok)
        {
            this.setVisible(false);
        }
    }
}
```

Download this Listing.

In the above listing, the Help() constructor creates the Help window, as shown in Figure 4-3:


**Figure 4-3:** The Help Window

When an end user clicks the OK button on the Help window, the actionPerformed() method is invoked. This method sets the visibility of the Help window to False and closes the window.



PREVIOUS NEXT

## Unit Testing

To test the File Search application:

1. 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;
   ```
2. 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;
   ```
3. Copy the Search.java, FileList.java, and Help.java files to a folder on your computer. Next, compile the files using the javac command as follows:
   ```
   javac *.java
   ```
4. To run the File Search application, specify the following command at the command prompt:
   ```
   java Search
   ```
5. Click the Browse button on the File Search Utility window to specify the location where you want to search for files.
6. Specify a text string in the Containing text box to limit your search to those files that contain this text pattern.
7. Click the Search button to start the file search process. The File Search Utility window with the search results appears, as shown in Figure 4-4:

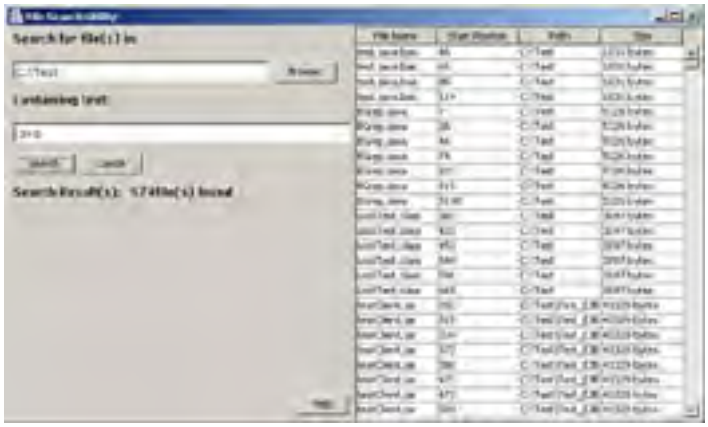

**Figure 4-4:** The File Search Utility Window with File Search Results

The right pane of the window shows the name and size of 974 files that are located within the C:\Test folder and contain the text string, java. The window also displays the start positions of the text pattern, java, in these 974 files.

# Chapter 5: Creating a Printer Management Application

The Java New Input/Output (NIO) API provides the java.nio, java.nio.channels, and java.awt.print packages, which you can use to perform various printer management functions. The java.nio package provides two classes, ByteBuffer and CharBuffer. You can use these classes to buffer a file. To read and write text and image files, you can use two classes provided by java.nio.channels, File and FileChannels. The java.awt.print package provides the PrinterJob and PageFormat classes to specify the properties of a file and print it. You can also use this package to manage the printers connected on the network.

This chapter describes how to develop a Printer Management application using the java.nio, java.nio.channels, and java.awt.print packages. It also explains how to read, write, buffer, format, and print text and image files. The application manages the printers that are connected to the computers that have Windows operating system.

## Architecture of the Printer Management Application

The Printer Management application provides an interface that allows end users to create a book, add a document to the book, specify the document page setup, and simultaneously print the required number of copies of that book. The book interface also allows end users to customize the print settings and use the application to manage the network printers. The application provides a Print dialog box that displays the status of the printer.

The Printer Management application uses the following files:

- `PrintFile.java`: Creates a user interface for the Printer Management application. This class also allows the end user to browse and open a file.
- `CreateBookInterface.java`: Creates a book interface that helps the end user to add documents that are to be printed in a book.
- `PrintComp.java`: Creates the Print dialog box to print the currently open document.

Figure 5-1 shows the architecture of the Printer Management application:

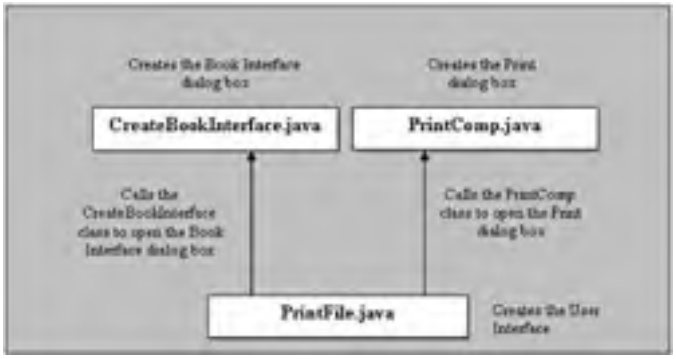


**Figure 5-1:** Architecture of the Printer Management Application

The PrintFile.java file is the main class of the Printer Management application. The File menu of the PrintFile.java file allows end users to open a document, specify its page setup, and print it. If they select the File-> Print->Print Default or File->Page Setup & Print menu item, the PrintFile.java file calls the PrintComp.java file, which prints the currently open text or image file.

If end users select the File-> Print Custom menu item, the PrintFile.java file calls the CreateBookInterface.java file, which creates a book that can be printed. End users can specify the page format and the number of copies of each page they want to print. All the pages added to the book can be printed using one print command.

# Creating the User Interface for the Printer Management Application

You can use the PrintFile.java file to create a user interface that has a menu and a tabbed pane. You can click the Print Text tab from the tabbed pane to open and print a text document. To open and print an image, click the Print Image tab from the tabbed pane. The user interface provides a set of menu items to open a document, specify the page setup, and print that document.

Listing 5-1 shows the contents of the PrintFile.java file:

**Listing 5-1: The PrintFile.java File**

```
/* Imports required swing classes. */
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTabbedPane;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JLabel;
import javax.swing.JFrame;
import javax.swing.JMenuItem;
import javax.swing.UIManager;
/* Imports required io classes. */
import java.io.File;
import java.io.FileInputStream;
/* Imports required nio classes. */
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
/* Imports required awt classes. */
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowListener;
import java.awt.event.WindowAdapter;
/*
class PrintFile - This class is the main class of the application. This class creates the user inter
Methods:
actionPerformed() - This method is invoked when an end user clicks any button.
main() - This method creates the main window of the application and displays it.
*/
class PrintFile extends JFrame implements ActionListener, Runnable
{
/* Declares object of JTabbedPane class. */
 JTabbedPane tabbedpane;
/* Declares object of Container class. */
 Container container;
/* Declares object of JMenuBar class. */
 JMenuBar menubar;
/* Declares object of JMenu class. */
 JMenu file_menu;
 JMenu print_menuitem;
/* Declares object of JMenuItem class. */
JMenuItem setup_menuitem, exit_menuitem, open_menuitem,prin_default,prin_custom;
/* Declares object of JLabel class. */
 JLabel label;
/* Declares objects of JScrollPane class. */
 JScrollPane image_scrollpane;
 JScrollPane text_scrollpane;
/* Declares object of JTextArea class. */
 JTextArea textarea;
/* Declares objects of JFileChooser class. */
 JFileChooser textfilechooser, imgfilechooser;
/* Declares object of CreateBookInterface class. */
 CreateBookInterface bIn;
 Thread thread1, thread2;
 int count = 0;
/* Implements the constructor of PrintFile class. */
 public PrintFile(String title)
 {
 super(title);
/*
Sets the look and feel of the application to windows.
*/
 try
 {
UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
 }
```

```
 catch(Exception e)
 {
System.out.println("Unknown Look and Feel." + e);
 }
/*Sets the size of user interface. */
 setSize(600,400);
/* Sets the resizable property of Jframe to false. */
 setResizable(false);
 bIn = new CreateBookInterface();
/* Adds window listener to the JFrame. */
 addWindowListener(new WindowAdapter()
 {
public void windowClosing(WindowEvent we)
 {
 System.exit(0);
 }
 });
 container = getContentPane();
/*Sets the layout of the container to BorderLayout. */
 container.setLayout(new BorderLayout());
/* Creates objects of JFileChooser class. */
 textfilechooser = new JFileChooser();
 imgfilechooser = new JFileChooser();
/*
Creates and sets the new file filter for selecting the text files.
*/
textfilechooser.setFileFilter(new javax.swing.filechooser.FileFilter()
 {
 public boolean accept(File f)
 {
 if(f.isDirectory())
 {
 return true;
 }
 String name=f.getName();
if ((name.endsWith(".jpg")) ||(name.endsWith(".gif")) || (name.endsWith(".GIF")) || (name.endsWith("
 {
 return false;
 }
 return true;
 }
 public String getDescription(){
 return "Text Files";
 }
 });
/*
Creates and set the new file filter for selecting the image files.
*/
imgfilechooser.setFileFilter(new javax.swing.filechooser.FileFilter ()
 {
/*
This method lets the end user select only JPG and GIF files from the file chooser.
*/
 public boolean accept(File f)
 {
 if (f.isDirectory())
 {
 return true;
 }
 String name = f.getName();
if (name.endsWith(".jpg") || name.endsWith(".gif") || name.endsWith(".GIF") || name.endsWith(".JPG")
 {
 return true;
 }
 return false;
 }
 public String getDescription() {
 return ".jpg, .gif";
 }
 });
/*
Initializes the menu bar. Sets menu bar to the frame.
*/
 menubar = new JMenuBar();
 setJMenuBar(menubar);
/* Initializes the menu. Adds menus to the menu bar. */
 file_menu = new JMenu("File");
 print_menuitem = new JMenu("Print");
 menubar.add(file_menu);
/*
Initializes the menu items and adds the menu item to the particular menu.
*/
 open_menuitem = new JMenuItem("Open");
setup_menuitem = new JMenuItem("Page Setup & Print");
```

```
 exit_menuitem = new JMenuItem("Exit");
prin_default=new JMenuItem("Print Default");
prin_custom=new JMenuItem("Print Custom");
 file_menu.add(open_menuitem);
 file_menu.add(setup_menuitem);
 print_menuitem.add(prin_default);
 print_menuitem.add(prin_custom);
 file_menu.add(print_menuitem);
 file_menu.add(exit_menuitem);
/*Adds the action listener with each menu item.*/
 open_menuitem.addActionListener(this);
 setup_menuitem.addActionListener(this);
 prin_default.addActionListener(this);
 prin_custom.addActionListener(this);
 exit_menuitem.addActionListener(this);
/* Creates an object of JTabbedPane class.*/
 tabbedpane = new JTabbedPane();
/* Creates a new JScrollPane for the text file.*/
text_scrollpane = new JScrollPane(textarea = new JTextArea());
/* Sets the text wrap style of the JTextArea class.*/
 textarea.setLineWrap(true);
 textarea.setWrapStyleWord(true);
/* Adds the JScrollPane to JTabbedPane.*/
tabbedpane.add("Print Text", text_scrollpane );
/* Creates an object of JScrollPane class for the image file.*/
image_scrollpane = new JScrollPane(label =new JLabel());
/* Adds the JScrollPane to JTabbedPane*/
tabbedpane.add("Print Image", image_scrollpane);
/* Adds the JTabbedPane to container.*/
container.add(tabbedpane, BorderLayout.CENTER);
 setVisible(true);
 }
/*
actionPerformed() - This method is called when the end user clicks any button.
Parameters: ae - an ActionEvent object containing the details of the event.
Return Value: NA
*/
 public void actionPerformed(ActionEvent ae)
 {
/*
This section is executed when the end user selects the File-> Page Setup & Print menu item.
*/
 if (ae.getSource()==setup_menuitem)
 {
 count =1;
 thread1 = new Thread(this);
 thread1.start();
 }
/*
This section is executed when the end user selects the File -> Print Default menu item.
*/
 if (ae.getSource()==prin_default)
 {
 count =2;
 thread2 = new Thread(this);
 thread2.start();
 }
/*
This section is executed when the end user selects the File -> Print Custom menu item.
*/
 if(ae.getSource()==prin_custom)
 {
 if (tabbedpane.getSelectedIndex()==0)
 bIn.setComponent(textarea);
 else
 bIn.setComponent(label);
 bIn.setVisible(true);
 }
/*
This section is executed when the end user selects the File->Exit menu item.
*/
 if (ae.getSource()==exit_menuitem)
 {
/* Terminates the application. */
 System.exit(0);
 }
/*
This section is executed when the end user selects the File -> Open menu item.
*/
 if (ae.getSource()==open_menuitem)
 {
/* Checks whether the text's tab is selected in the tabbed pane. */
 if (tabbedpane.getSelectedIndex()==0)
 {
int returnVal = textfilechooser.showOpenDialog(this);
if(returnVal == JFileChooser.APPROVE_OPTION)
```

```
 {
 try
 {
/*
Reads the contents of opened file and sets the contents of the opened file in the text area.
*/
FileInputStream inputStream = new FileInputStream(textfilechooser.getSelectedFile().getAbsolutePath(
FileChannel fileIn=inputStream.getChannel();
long fsize=fileIn.size();
ByteBuffer buffin=ByteBuffer.allocate((int)fsize);
fileIn.read(buffin);
buffin.rewind();
String readLine=new String(buffin.array());
textarea.setText(readLine);
fileIn.close();
inputStream.close();
 }
 catch(Exception ex)
 {}
bIn=new CreateBookInterface();
 }
 }
/* Checks whether the image's tab is selected in the tabbed pane. */
else if (tabbedpane.getSelectedIndex()==1)
 {
int returnVal = imgfilechooser.showOpenDialog(this);
if(returnVal == JFileChooser.APPROVE_OPTION)
 {
 try
 {
label.setIcon(new ImageIcon(imgfilechooser.getSelectedFile().getAbsolutePath()));
 }
 catch(Exception ex)
 {}
 }
 }
 }
 }
public void run()
 {
 if(count == 1)
 {
 if (tabbedpane.getSelectedIndex()==0)
new PrintComp(textarea).pageSetupAndPrint();
 else
new PrintComp(label).pageSetupAndPrint();
 }
 else if(count == 2)
 {
if (tabbedpane.getSelectedIndex()==0)
 PrintComp.printComponent(textarea);
 else
 PrintComp.printComponent(label);
 this.setVisible(true);
 }
 }
/*
main() - This method creates the main window of the user interface and displays it.
Parameters:
args[] - Contains any command line arguments passed.
Return Value: NA
*/
 public static void main(String[] args)
 {
PrintFile frame = new PrintFile("Printer Management Application");
 }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the PrintFile.java class. This class generates the main window of the Printer Management application, as shown in Figure 5-2:

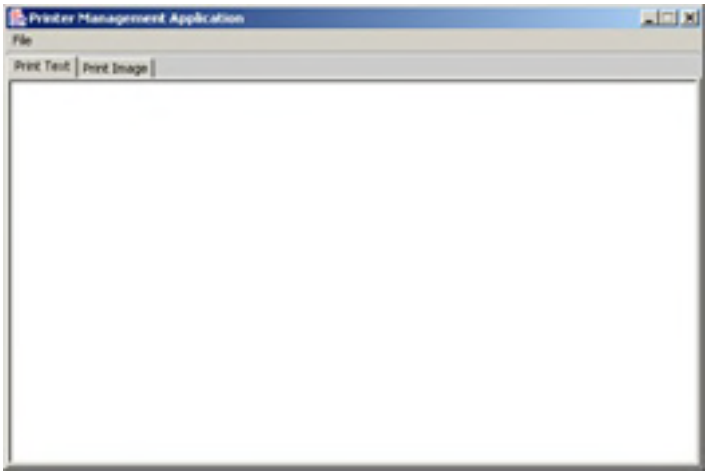


**Figure 5-2:** The Printer Management Application User Interface

When the end user selects the File menu, the options associated with it appear, as shown in Figure 5-3:

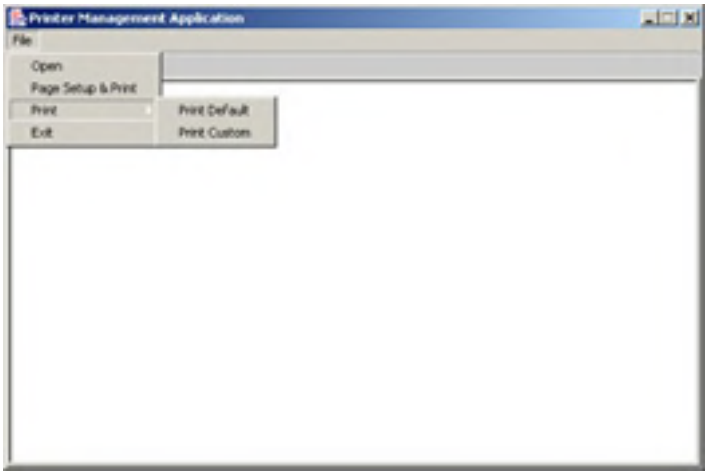


**Figure 5-3:** The File Menu of the Printer Management Application

To open a file, end users need to select the File-> Open menu option. The Open dialog box appears. When an end user selects the file and clicks the Open button of Open dialog box, the content of the selected file is displayed in the text area.

If end users select the File -> Page Setup & Print menu option, the actionPerformed() method sets the count value to 1 and calls the start() method of the Thread class to start the thread, thread1.

If the File->Print->Print Default menu option is selected, the actionPerformed() method sets the count value to 2 and calls the start() method of the Thread class to start the thread, thread2.

When the end user selects the File->Print->Print Custom menu option, the actionPerformed() method calls the setComponent() method of the CreateBookInterface class and opens a Book Interface dialog box.

The application terminates if the end user selects the File->Exit menu option.

The methods defined in the above listing are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the menu option the end user selects. This method is invoked when an end user selects any option from the File menu.
- `run()`: Checks the count value. If the count value is 1, this method creates an instance of the PrintComp class and calls the pageSetupAndPrint() method of the PrintComp class. If the count value is 2, the run() method creates an instance of PrintComp class.


◄ PREVIOUS NEXT ►

# Implementing the Print Functionality

The PrintComp.java file implements the Printable interface of Java. This file allows an end user to print a document through the network printer.

Listing 5-2 shows the contents of the PrintComp.java file:

**Listing 5-2: The PrintComp.java File**

```
/* Imports required awt classes. */
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;
import java.awt.print.PageFormat;
/* Imports required swing classes. */
import javax.swing.JComponent;
/*
class PrintComp - This class is the subclass of the application. This class implements the Printable
Methods:
printComponent() - Invokes the print() method of PrintComp class and prints a component.
pageSetupAndPrint() - Allows the end user to specify the page setup and print the page.
print() - Prints the page.
print()-Prints a graphic at the specified page index in the specified page format.
*/
class PrintComp implements Printable
{
/* Declares object of the JComponent class.*/
 private JComponent component;
/* Declares object of the PrinterJob class.*/
PrinterJob printJob = PrinterJob.getPrinterJob();
/* Declares object of the PageFormat class*/
 PageFormat pageFormat= printJob.defaultPage();
/*
printComponent - This method is called when the end user clicks the Print Default menu item in the u
Parameters:
c - An object of the JComponent class.
Return Value: NA
*/
 public static void printComponent(JComponent c)
 {
 new PrintComp(c).print();
 }
/* Implements the constructor of the PrintComp class. */
 public PrintComp(JComponent component)
 {
 this.component= component;
 }
/*
pageSetupAndPrint - This method is called when the end user clicks the Page Setup and Print button.
Parameters: NA
Return Value: NA
*/
 public void pageSetupAndPrint()
 {
/* Initializes the object of the PageFormat class. */
 pageFormat = printJob.pageDialog(pageFormat);
/* Sets the object of the PrinterJob class to printable. */
 printJob.setPrintable(this, pageFormat);
 if (printJob.printDialog())
 {
 try
 {
/* Prints the document. */
 printJob.print();
 }
 catch(PrinterException pe)
 {
System.out.println("Error in printing !!! " + pe);
 }
 }
 }
/*
print - This method is called when the end user clicks the Print button.
Parameters: NA
Return Value: NA
*/
 public void print()
 {
/* Sets the object of the PrinterJob class to printable. */
```

```
 printJob.setPrintable(this, pageFormat);
/* Checks the return value of PrintDialog. */
 if (printJob.printDialog())
 {
 try
 {
/* Prints the document. */
 printJob.print();
 }
 catch(PrinterException pe)
 {
System.out.println("Error in printing !!! " + pe);
 }
 }
 }
/*
print() - This method defines a Printable interface.
Parameters:
g - Represents the object of the Graphics class.
pf - Represents the object of the PageFormat class.
index - Represents an index of the page.
Return Value: int PAGE_EXIST
*/
public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
 {
/*
Creates an object of the Graphics2D class and convert simple graphics to 2D graphics.
*/
 Graphics2D g2 = (Graphics2D)g;
/*Gets size of document. */
 Dimension d = component.getSize();
/* Gets the width of document in pixels. */
 double componentWidth = d.width;
/* Gets the height of document in pixels. */
 double componentHeight = d.height;
/* Gets the height of printer page. */
double pageHeight = pf.getImageableHeight();
/* Get the width of printer page*/
double pageWidth = pf.getImageableWidth();
/*
Sets the value of scale for the document to be printed.
*/
 double scale = pageWidth/componentWidth;
int pages = (int)Math.ceil(scale * componentHeight / pageHeight);
/* Does not print empty pages. */
 if(pageIndex >= pages)
 {
 return Printable.NO_SUCH_PAGE;
 }
/*
Shifts the graphic to line up with the beginning of print-imageable region.
*/
g2.translate(pf.getImageableX(), pf.getImageableY());
/*
Shifts the graphic to line up with the beginning of the next page to print.
*/
 g2.translate(0f, -pageIndex*pageHeight);
/* Scales the page so that the width fits. */
 g2.scale(scale, scale);
/* Repaints the page. */
 component.paint(g2);
 return Printable.PAGE_EXISTS;
 }
}
```

Download this Listing.

In the above code, the constructor of the PrintComp class uses the object of the JComponent class to load the text or image file. The methods defined in the listing are:

- `printComponent()`: Calls the print() method to print the document.
- `pageSetupAndPrint()`: Initializes the object of the PageFormat class and sets the object of the PrinterJob class to printable. This method then calls the print() method of the PrinterJob class.
- `print()`: Calls the print() method of the PrinterJob class.
- `print(Graphics g, PageFormat pf, int pageIndex)`: Creates an object of the Graphics2D class and converts simple graphics to 2D ones. This method then gets the size of the document and sets its scale value. Next, the method calls the translate() and scale() methods of the Graphics2D class. Finally, the print() method calls the paint() method of the Component class to paint the document with new dimensions and returns Printable.PAGE_EXISTS.

# Creating the Book Interface

The CreateBookInterface.java file helps you to create a book interface for the document that is to be printed.

Listing 5-3 shows the contents of the CreateBookInterface.java file:

**Listing 5-3: The CreateBookInterface.java File**

```
/* Imports java.awt package classes. */
import java.awt.Graphics;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowListener;
import java.awt.event.WindowAdapter;
/* Imports awt.print package classes. */
import java.awt.print.Book;
import java.awt.print.PrinterJob;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;
/* Imports javax.swing package classes. */
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JCheckBox;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.DefaultListModel;
import javax.swing.ButtonGroup;
/* Imports java.util package classes. */
import java.util.Enumeration;
import java.util.Vector;
/*
class
CreateBookInterface - This class is the subclass of the application. This class creates a book for t

Methods:
actionPerformed() - This method is invoked when an end user clicks any button.
setComponent() - This method sets the component to the currently opened document.
createBook() - This method is called when the end user clicks the Print Custom menu option.
addPagesInBook()- This method is called when the end user clicks the Add Pages In Book button.
printBook()- This method is called when the end user clicks the Print button.
*/
public class CreateBookInterface extends JFrame implements ActionListener
{
/* Declares object of JComponent class. */
 JComponent component;
/* Declares object of Book class. */
 Book book = null;
/* Declares object of DefaultListModel class. */
 DefaultListModel dlm;
/* Declares object of PrinterJob class. */
 PrinterJob printJob = null;
/* Declares object of ButtonGroup class.*/
 ButtonGroup bg;
/* Declares object of JCheckBox class.*/
 JCheckBox lscape = null;
 JCheckBox potrait=null;
/* Declares object of JTextField class. */
 JTextField pageNo = null;
/* Declares object of JList class. */
 JList bookItemList = null;
/* Declares object of JButton class. */
 JButton add, print, reset;
 String format;
 int pageNum;
/* Declares object of BookPrintable class. */
 BookPrintable bookPrintable;
/*
Implements the constructor of the CreateBookInterface class.
*/
 CreateBookInterface()
 {
/*Sets the title of the CreateBookInterface class. */
```

```
 setTitle("Book Interface");
/* Sets the resizable property to false. */
 setResizable(false);
/*
Adds the window closing event to the CreateBookInterface class.
*/
 addWindowListener(new WindowAdapter()
 {
public void windowClosing(WindowEvent we)
 {
 setVisible(false);
 }
 });
/* Sets the size of the CreateBookInterface class. */
 setSize(400,250);
 setLocation(50,50);
/* Initializes the object of the BookPrintable class. */
 bookPrintable = new BookPrintable();
/* Creates a new ButtonGroup object. */
 bg = new ButtonGroup();
/* Creates a new object of the DefaultListModel class. */
 dlm = new DefaultListModel();
/* Creates a new object of the JPanel class. */
 JPanel selectPanel = new JPanel();
/* Sets the layout of the Jpanel class. */
selectPanel.setLayout(new BorderLayout());
/* Creates a new object of the JPanel class. */
 JPanel formatPanel = new JPanel();
/* Sets the layout of the Jpanel class. */
formatPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
/* Adds a new JLabel to the panel. */
formatPanel.add(new JLabel("Page Format : "));
/* Creates new JCheckBox objects. */
 lscape=new JCheckBox("Landscape",true);
 potrait=new JCheckBox("Potrait",false);
/* Adds the JCheckBox objects to ButtonGroup. */
 bg.add(lscape);
 bg.add(potrait);
/*
Adds the objects of JCheckBox to the JPanel, formatPanel.
*/
 formatPanel.add(lscape);
 formatPanel.add(potrait);
/* Adds formatPanel to the panel. */
 selectPanel.add("North", formatPanel);
/* Creates a new object of the JPanel class. */
 JPanel pageNoPanel = new JPanel();
/* Sets the layout of the JPanel class. */
pageNoPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
/* Adds a new JLabel to the JPanel,pageNoPanel. */
pageNoPanel.add(new JLabel("Number of Pages : "));
/* Creates a new object of the JTextField class. */
 pageNo = new JTextField(5);
/* Adds JTextBox to the JPanel,pageNoPanel. */
 pageNoPanel.add(pageNo);
/*
Adds the pageNoPanel to the selectPanel panel.
*/
 selectPanel.add("Center", pageNoPanel);
 add = new JButton("Add in the Book");
/* Adds the action listener to the add button. */
 add.addActionListener(this);
/* Adds the add button to the selectPanel panel. */
 selectPanel.add("South", add);
getContentPane().add("North", selectPanel);
/* Creates an object of the JList class. */
 bookItemList = new JList(dlm);
/* Sets the background color of the JList object to gray. */
 bookItemList.setBackground(Color.gray);
getContentPane().add("Center", new JScrollPane(bookItemList));
/* Creates a new JPanel, ctrlPanel. */
 JPanel ctrlPanel = new JPanel();
/*
Sets the layout of the ctrlPanel JPanel to GridLayout.
*/
 ctrlPanel.setLayout(new GridLayout(1,2));
 print = new JButton("Print");
 print.setEnabled(false);
/* Adds action listener to the print JButton. */
 print.addActionListener(this);
 ctrlPanel.add(print);
 reset = new JButton("Reset");
 reset.setEnabled(false);
/* Adds action listener to the reset JButton. */
 reset.addActionListener(this);
 ctrlPanel.add(reset);
 getContentPane().add("South", ctrlPanel);
/* Invokes the createBook() method*/
```

```
 createBook();
 setVisible(false);
 }
/*
actionPerformed() - This method is called when the end user clicks any button.
Parameters: ae - An ActionEvent object containing details of the event.
Return Value: NA
*/
 public void actionPerformed(ActionEvent ae)
 {
/*
This section is executed when the end user clicks the Add in the Book button.
*/
if(ae.getActionCommand().equals("Add in the Book"))
 {
/*
Gets all the elements added to the ButtonGroup, bg in an object of the Enumeration class.
*/
 Enumeration enum=bg.getElements();
/*
The while loop runs until there is any element in the enumeration.
*/
 while(enum.hasMoreElements())
 {
/*
Retrieves the next element of the enumeration in an object of the JCheckBox class.
*/
JCheckBox button=(JCheckBox)enum.nextElement();
/* Checks if the check box is selected.*/
 if (button.isSelected())
 {
/* Gets the text of the check box in a string.*/
format=button.getText();
 }
 }
 try
 {
pageNum = Integer.parseInt(pageNo.getText().trim());
 }
 catch (Exception ex)
 {
System.out.println("Error : Invalid entry of page number.");
 return;
 }
 print.setEnabled(true);
/* Adds the currently open document to the book.*/
 addPagesInBook(format, pageNum);
 }
/*
This section is executed when the end user clicks the Print button.
*/
else if(ae.getActionCommand().equals("Print"))
 {
 reset.setEnabled(true);
 print.setEnabled(false);
 setComponent(null);
/* Invokes the printBook() method.*/
 printBook();
 }
/*
This section is executed when the end user clicks the Reset button.
*/
else if(ae.getActionCommand().equals("Reset"))
 {
 print.setEnabled(true);
 reset.setEnabled(false);
/* Clears the JList.*/
 dlm.clear();
/* Clears the book. */
 book = null;
/* Invokes the createBook() method*/
 createBook();
 }
 }
/*
setComponent() - This method sets the value of the object of the JComponent class.
Parameters: component - An object of the JComponent class.
Return Value: NA
*/
 void setComponent(JComponent component)
 {
 this.component= component;
 }
/*
createBook() - This method creates a book to add the document to be printed.
Parameters: NA
```

```
Return Value: NA
*/
 void createBook()
 {
 book = new Book();
 printJob = PrinterJob.getPrinterJob();
 }
/*
createBook() - This method is called when the end user clicks the Add in the Book button.
Parameters:
format- A string that represents the format in which the document to be printed is added to the book
pageNumber- An integer representing the number of pages to be added to the book.
Return Value: NA
*/
void addPagesInBook(String format, int pageNumber)
 {
/* Creates an object of PageFormat class. */
PageFormat pageFormat = printJob.defaultPage();
/* Checks if the Potrait check box is checked. */
 if(format.equals("Potrait"))
 {
pageFormat.setOrientation (PageFormat.PORTRAIT);
 }
/* Checks if the Landscape check box is checked. */
 else if(format.equals("Landscape"))
 {
pageFormat.setOrientation (PageFormat.LANDSCAPE);
 }
/* Appends the document to be printed to the book. */
book.append(bookPrintable,pageFormat, pageNumber);
String temporarySt=new String("Book Item Added : Page Format : "+format+" : "+" Number of Pages : "+
/*
Adds the details of the document to be printed to a JList.
*/
 dlm.addElement(temporarySt);
 }
/*
printBook() - This method is called when the end user clicks the Print button.
Parameters: NA
Return Value: NA
*/
 void printBook()
 {
 printJob.setPageable(book);
 try
 {
/* Calls the print() method to print a book. */
 printJob.print();
 }
 catch(PrinterException pe)
 {
System.out.println("Error in printing !!! " + pe);
 }
 }
/*
class BookPrintable - This class is the subclass of the application. This class implements the Print
Methods:
print()-This method is called when the end user prints a document.
*/
 class BookPrintable implements Printable
 {
/*
print() - This method defines the Printable interface.
Parameters:
g - Represents the object of the Graphics class.
pf - Represents the object of the PageFormat class.
index - Represents an index of a page.
Return Value: int PAGE_EXIST
*/
public int print(Graphics g, PageFormat pf, int pageIndex)
 {
/* Implements print code here. */
 return Printable.PAGE_EXISTS;
 }
 }
}
```

Download this Listing.

In the above code, the CreateBookInterface class allows you to create a user interface to create and print a book, as shown in Figure 5-4:

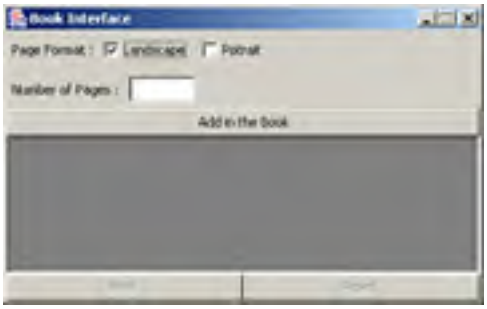


**Figure 5-4:** The Book Interface Window

The methods defined in the above code are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate method, based on the button the end user clicks in the Book Interface dialog box. If the end user clicks the Add in the Book button, the actionPerformed() method calls the addPagesInBook() method. If the Print button is clicked, the actionPerformed() method calls the printBook() method. When the end user clicks the Reset button, the actionPerformed() method calls the clear() method of the JComponent class and then calls the createBook() method of Book class.
- `createBook()`: Initializes the object of the Book class and gets the object of the PrinterJob class using the getPrinterJob() method.
- `addPagesInBook()`: Uses format and paneNumber as input parameters. The addPagesInBook() method creates an object of the PageFormat class. This method then checks the page format style and sets the appropriate page orientation. Next, the addPagesInBook() method appends to the book the document that is to be printed. Finally, the addPageInBook() method adds the elements to the list box.
- `printBook()`: Calls the setPageable() method of the PrinterJob class to set the page. The printBook() method calls the print() method to print the file.

The CreateBookInterface class contains a sub class called BookPrintable, which implements the Printable interface of the book. The BookPrintable class calls the print() method to print the file. You can define the print() method to print a customized book.



◄ PREVIOUS NEXT ►

# Unit Testing

To test the Printer Management application:

1. 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;
   ```
2. 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;
   ```
3. Copy the PrintFile.java, PrintComp.java, and CreateBookInterface.java files to a folder on your computer. On the command prompt, use the cd command to move to that folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Printer Management application, specify the following command at the command prompt:
   ```
   java PrintFile
   ```
5. Click the Print Text tab in the user interface of the Printer Management application.
6. Select the File->Open command from the File menu of the Printer Management Application. Browse and open the text file that is to be printed using the Open dialog box. The selected file appears in JTabbedPane, as shown in Figure 5-5:

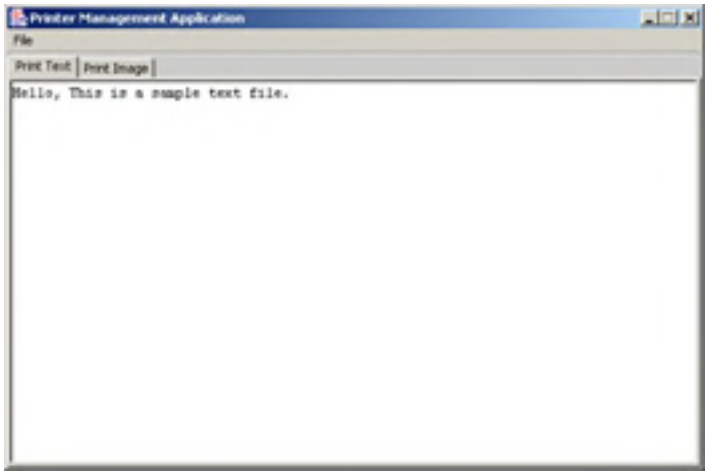


**Figure 5-5:** Printer Management Application Window

7. Select the File->Page Setup & Print command. The Page Setup dialog box opens, as shown in Figure 5-6:


**Figure 5-6:** Page Setup Dialog Box

8. Click the Printer button on the Page Setup dialog box. A new Page Setup dialog box appears, which displays the name, status, type and location of the current printer, as shown in Figure 5-7:

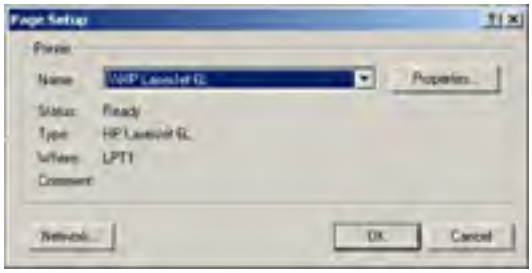


**Figure 5-7:** Displaying the Printer Status

9. Click the Network button to displays all the printers available on the network, as shown in Figure 5-8:

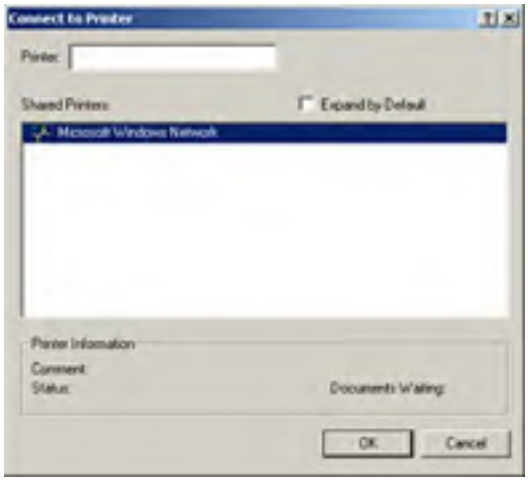


**Figure 5-8:** Connect to Printer Dialog Box

10. Click the OK button on the Page Setup dialog box. A Print dialog box opens, as shown in Figure 5-9:

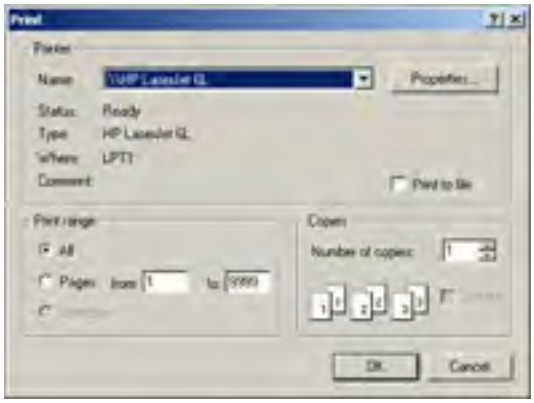


**Figure 5-9:** Print Dialog Box

11. Click OK to print the current file.
12. Click the Print Image tab. The image view of the Printer Management application appears, as shown in Figure 5-10:

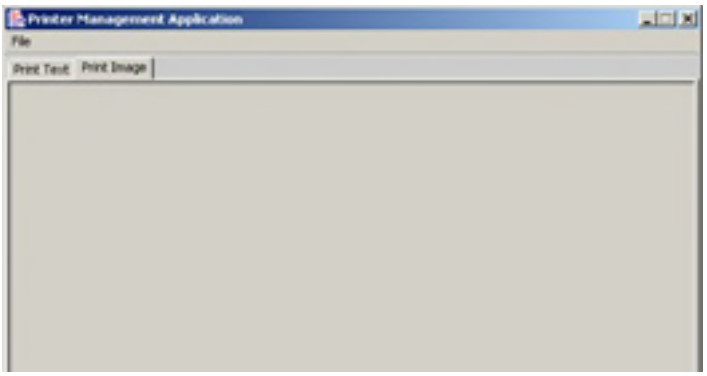

**Figure 5-10:** Printer Management Application with Print Image Tabbed Pane

13. Select File->Open to open an image in the Print Image tabbed pane, as shown in Figure 5-11:

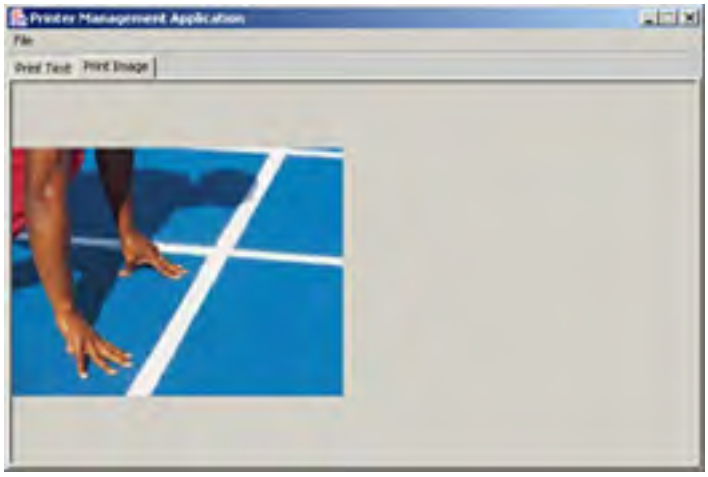


**Figure 5-11:** Displaying an Image File in the Print Image Tabbed Pane

14. Select File->Print->Print Default to print the current page. Select File->Print->Print Custom to print a customized book, as shown in Figure 5-4. When the end user selects the page format using the Page Format check box and specifies the number of pages in Number of Pages test box of the Book Interface dialog box. Now, clicks the Add in the Book button in the Book Interface dialog box. The results are displayed in the list box of the Book Interface, as shown in Figure 5-12:

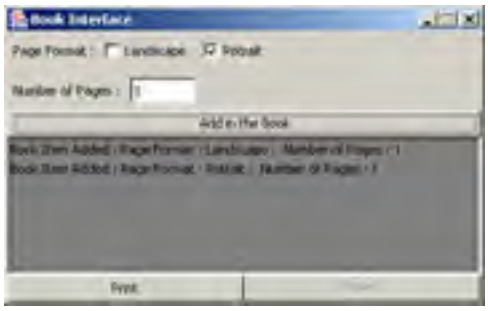


**Figure 5-12:** Displaying a Book Interface dialog

15. Click the Print button to print the book items.

# Chapter 6: Creating a Text Editor Application

The New Input/Output (NIO) API provides the java.nio and java.nio.channels packages for buffer management, advance I/O file system, and file locking mechanism. The java.nio package contains the ByteBuffer classes that allow you to store the contents of a text file. To read, write, and lock the text files, you can use the FileLock and FileChannel classes available in the java.nio.channels package.

This chapter explains how to develop a Text Editor application using the java.nio and java.nio.channels packages and create, edit, print, and save text files.

## Architecture of the Text Editor Application

The Text Editor application allows you to read, write, print, and lock a specific text file. You can also change the color and font of the text file. The Text Editor application provides various editing and formatting features, such as cut, copy, paste, undo, redo, select all, find, and word wrap.

The Text Editor application uses the following files:

- `Editor.java`: Creates a user interface that contains a menu bar and a text area. There are five menus on the menu bar: File, Edit, Format, Locks, and Help. Each menu contains several menu items. You can use these items to perform various tasks, such as opening, saving, or printing a file.
- `ActionPerform.java`: Implements the commands to create, open, save, lock, or copy a file. It also implements commands to find a word in a file. In addition, this class also implements the commands to cut, copy, paste, undo, redo, set font, set text color, and insert date in the file.
- `FontClass.java`: Creates a font dialog box that provides options to change the type, size, and style of the font in the text file.
- `ColorClass.java`: Creates a color dialog box with sliders. You can use this dialog box to specify the Red Green Blue (RGB) value and change the color of the file text.
- `PrintClass`: Opens a print dialog box that allows you to print a document. You can also use this dialog box to change the properties of the document, such as page layout and paper quality.
- `Help.java`: Creates a help dialog box that lists the steps to use the Text Editor.

Figure 6-1 shows the architecture of the Text Editor application:

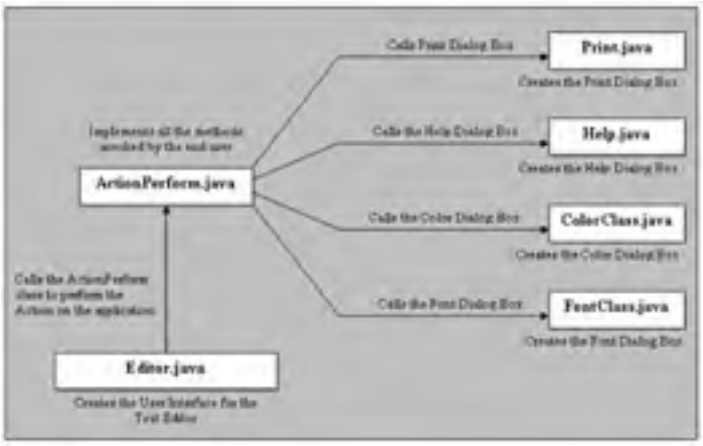


**Figure 6-1:** Architecture of the Text Editor Application

The Editor.java file calls ActionPerform.java to perform the file edit, format, and lock operations. After the ActionPerform.java file implements these operations, it returns the results to the Editor.java file. The Editor.java file then displays the results to the end user. The ActionPerform.java file calls the PrintClass.java file to print a file. To change the font type, size, and style of the text file, the ActionPerform.java file calls the FontClass.java file. The ActionPerform.java file calls the ColorClass.java file to change the color of the file text.

If an end user selects the help file option, the Editor.java file calls the ActionPerform.java file, which, in turn, calls the Help.java file. The Help.java file lists the steps required to use the Text Editor.

# Creating the User Interface for the Text Editor Application

The Editor.java file helps create a user interface with a menu bar and a text area for the Text Editor application. Using this interface, the end user can create, open, save, lock, edit, or format a new file or the existing file. The user interface also contains a text area where the contents of the new file or existing file are displayed to the end user.

Listing 6-1 shows the contents of the Editor.java file:

**Listing 6-1: The Editor.java File**

```
/* Imports the java.net package class. */
import java.net.URL;
/* Imports javax.swing.undo package classes. */
import javax.swing.undo.UndoManager;
import javax.swing.undo.CannotUndoException;
import javax.swing.undo.CannotRedoException;
/* Imports javax.swing package classes. */
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.UIManager;
import javax.swing.Action;
import javax.swing.KeyStroke;
import javax.swing.AbstractAction;
/* Imports java.awt package class. */
import java.awt.BorderLayout;
/* Imports java.awt.event package classes. */
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
/* Imports javax.swing.event package classes. */
import javax.swing.event.UndoableEditListener;
import javax.swing.event.UndoableEditEvent;
/*
Class Editor - This class is the main class of the application. This class initializes
the interface and loads all components like text area and menu bar on the main window.
Fields:
   menu - Creates the menu bar.
   file - Creates the File menu.
   edit - Creates the Edit menu.
   format - Creates the Format menu.
   lock - Creates the Lock menu.
   help - Creates the Help menu.
   newfile - Creates the New menu item.
   open - Creates the Open menu item.
   print - Creates the Print menu item.
   save - Creates the Save menu item.
   saveas - Creates the Save As menu item.
   exit - Creates the Open Exit item.
   cut - Creates the Cut menu item.
   copy - Creates the Copy menu item.
   paste - Creates the Paste menu item.
   find - Creates the Find menu item.
   selectAll - Creates the Select All menu tem.
   datetime - Creates the Date/Time menu item.
   font - Creates the Open Font item.
   color - Creates the Open Color item.
   about - Creates the Open Help item.
   area - Creates an object the JTextArea class that contains the file contents.
   panel - Creates the panel menu item that contains all the AWT and swing components.
   scrollpane - Creates an object of the JScrollPane class that is added to the text area.
   str - Creates the String object.
   wordWrap - Creates the Word Wrap menu item.
   exclusiveLock - Creates the Exclusive menu item.
   shareLock - Creates the Share menu item.
Methods:
   addWindowListener() - This method is called when end user clicks the window close button.
   actionPerformed() - This method is invoked when an end user selects any command from the menu bar
   undoableEditHappened() - This method is called when an end user performs undo and redo operation.
   main() - This method creates the main window of the application and displays it.
*/
public class Editor extends JFrame implements ActionListener, UndoableEditListener, Runnable
{
   /* Creates an object of the ActionPerform class. */
   public ActionPerform action = new ActionPerform(this);
```

```
/* Creates an object of the UndoManager class. */
UndoManager undoManager = new UndoManager();
/* Creates an instance of the UndoAction class. */
UndoAction undoAction = new UndoAction();
/* Creates an instance of the RedoAction class. */
RedoAction redoAction = new RedoAction();
/* Declares an object of the JMenuBar class. */
JMenuBar menu;
/* Declares objects of the JMenu class. */
JMenu file;
JMenu edit;
JMenu format;
JMenu lock;
JMenu help;
/* Declares objects of the JMenuItem class. */
JMenuItem newfile;
JMenuItem open;
JMenuItem print;
JMenuItem save;
JMenuItem saveas;
JMenuItem exit;
JMenuItem cut;
JMenuItem copy;
JMenuItem paste;
JMenuItem find;
JMenuItem selectAll;
JMenuItem datetime;
JMenuItem font;
JMenuItem color;
JMenuItem about;
/* Declares an object of the JTextArea class. */
JTextArea area;
/* Declares an object of the JPanel class. */
JPanel panel;
/* Declares an object of the JScrollPane class. */
JScrollPane scrollpane;
/* Declares an object of the String class. */
String str;
/* Declares objects of the JCheckBoxMenuItem class. */
JCheckBoxMenuItem wordWrap;
JCheckBoxMenuItem exclusiveLock;
JCheckBoxMenuItem shareLock;
/* Declares objects of the Thread class. */
Thread t1;
Thread t2;
Thread t3;
int act;
/* Defines the constructor of the Editor class. */
public Editor()
{
   /*
   Initializes the menu bar and sets the menu bar to the frame.
   */
   menu = new JMenuBar();
   setJMenuBar(menu);
   setTitle("Text Editor");
   /*
   Initializes the menu and adds the menus to the menu bar.
   */
   file = new JMenu("File");
   menu.add(file);
   edit = new JMenu("Edit");
   menu.add(edit);
   format = new JMenu("Format");
   menu.add(format);
   lock = new JMenu("Locks");
   menu.add(lock);
   help = new JMenu("Help");
   menu.add(help);
   /* Sets the mnemonic to the menu. */
   file.setMnemonic('f');
   edit.setMnemonic('e');
   lock.setMnemonic('l');
   format.setMnemonic('o');
   help.setMnemonic('h');
   /*
   Initializes the menu items.
   Adds the menu item to the particular menu.
   Adds the action listener with each menu item.
   Sets accelerator to each menu item.
   */
   /* File->New */
   newfile = new JMenuItem("New");
   newfile.addActionListener(this);
   newfile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, ActionEvent.CTRL_MASK));
   file.add(newfile);
   /* File->Open */
   open = new JMenuItem("Open...");
```

```
open.addActionListener(this);
open.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, ActionEvent.CTRL_MASK));
file.add(open);
/* File->Save */
save = new JMenuItem("Save");
save.addActionListener(this);
save.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, ActionEvent.CTRL_MASK));
file.add(save);
/* File->Save As */
saveas = new JMenuItem("Save As...");
saveas.addActionListener(this);
file.add(saveas);
/* File->Print */
print = new JMenuItem("Print");
print.addActionListener(this);
print.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P, ActionEvent.CTRL_MASK));
file.add(print);
/* File->Exit */
exit = new JMenuItem("Exit");
exit.addActionListener(this);
exit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F4, ActionEvent.CTRL_MASK));
file.add(exit);
/* Inserts separators at the 4th and 6th positions. */
file.insertSeparator(4);
file.insertSeparator(6);
/* Edit->Undo */
edit.add(undoAction);
/* Edit->Redo */
edit.add(redoAction);
/* Edit->Cut */
cut = new JMenuItem("Cut");
edit.add(cut);
cut.addActionListener(this);
cut.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK));
/* Edit->Copy */
copy = new JMenuItem("Copy");
edit.add(copy);
copy.addActionListener(this);
copy.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C, ActionEvent.CTRL_MASK));
/* Edit->Paste */
paste = new JMenuItem("Paste");
edit.add(paste);
paste.addActionListener(this);
paste.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_V, ActionEvent.CTRL_MASK));
/* Edit->Find */
find = new JMenuItem("Find");
edit.add(find);
find.addActionListener(this);
find.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F, ActionEvent.CTRL_MASK));
/* Edit->Select All */
selectAll = new JMenuItem("Select All");
edit.add(selectAll);
selectAll.addActionListener(this);
selectAll.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A, ActionEvent.CTRL_MASK));
/* Edit->Date/Time */
datetime = new JMenuItem("Date/Time");
edit.add(datetime);
datetime.addActionListener(this);
datetime.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F5, ActionEvent.CTRL_MASK));
/* Inserts separators at the 2nd, 6th, and 9th positions. */
edit.insertSeparator(2);
edit.insertSeparator(6);
edit.insertSeparator(9);
/* Format->Font */
font = new JMenuItem("Font");
format.add(font);
font.addActionListener(this);
/* Format->Color */
color = new JMenuItem("Color");
format.add(color);
color.addActionListener(this);
/* Format->Word Wrap */
wordWrap = new JCheckBoxMenuItem("Word Wrap");
format.add(wordWrap);
wordWrap.addActionListener(this);
/* Lock->Exclusive Lock */
exclusiveLock = new JCheckBoxMenuItem("Exclusive Lock");
exclusiveLock.setEnabled(false);
lock.add(exclusiveLock);
exclusiveLock.addActionListener(this);
/* Lock->Share Lock */
shareLock = new JCheckBoxMenuItem("Share Lock");
shareLock.setEnabled(false);
lock.add(shareLock);
shareLock.addActionListener(this);
/* Help->Help Topics */
about = new JMenuItem("Help Topics");
help.add(about);
```

```
about.addActionListener(this);
/* Initializes the panel. */
panel = new JPanel();
/* Sets the panel layout as BorderLayout. */
panel.setLayout(new BorderLayout());
/* Adds the panel to the frame. */
getContentPane().add(panel, BorderLayout.CENTER);
/* Initializes the object of text area */
area = new JTextArea(25, 65);
/*
Adds UndoableEditListener to the area for undo and redo operations.
*/
area.getDocument().addUndoableEditListener(this);
/* Sets line wrap to false. */
area.setLineWrap(false);
/* Sets word wrap style to false. */
area.setWrapStyleWord(false);
/*
Initializes the object of the Scrollpane class and adds the text area to the scroll pane.
*/
scrollpane = new JScrollPane(area);
/* Adds the scroll pane to the panel. */
panel.add(scrollpane);
/* Sets default close operation to false. */
setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
/*
addWindowListener - It contains a windowClosing() method.
windowClosing: It is called when the user clicks the cancel button of the Window.
It closes the main window.
Parameter: we- Represents an object of the WindowEvent class.
Return Value: NA
*/
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
       System.exit(0);
    }
 });
}
/*
undoableEditHappened() - This method is called when the user wants to perform undo/redo.
Parameters: e - Represents an object of UndoableEditEvent class that contains
the details of the undo/redo events.
Return Value: NA
*/
public void undoableEditHappened(UndoableEditEvent e)
{
   /*
   Inserts an Edit at indexOfNextAdd and removes any old edits that were at indexOfNextAdd or
   */
    undoManager.addEdit(e.getEdit());
   /* Calls the update() method to perform undo operations. */
    undoAction.update();
   /* Calls the update() method to perform redo operations. */
    redoAction.update();
}
/*
actionPerformed() - This method is called when the end user selects any menu item from the men
Parameters: ae - Represents an object of the ActionEvent class that contains the details of th
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   /*
   This section is executed when the end user selects the File->New command to the menu bar.
   */
    if(ae.getSource() == newfile)
    {
      /* Calls the newFile() method. */
       exclusiveLock.setEnabled(false);
       shareLock.setEnabled(false);
       action.newFile();
    }
   /*
   This section is executed when the end user selects the File->Open command to the menu bar.
   */
    else if(ae.getSource() == open)
    {
      /* Calls the openFile() method. */
       action.openFile();
    }

   /*
   This section is executed when the end user selects the File->Save command to the menu bar.
   */
    else if(ae.getSource() == save)
    {
```

```
   /* Calls the saveFile() method. */
    action.saveFile();
 }
/*
This section is executed when the end user selects the File->Save As command to the menu ba
*/
 else if(ae.getSource() == saveas)
 {
   /* Calls the saveAsFile() method. */
    action.saveAsFile();
 }
/*
This section is executed when the end user selects the File->Print command to the menu bar.
*/
 else if(ae.getSource() == print)
 {
   /* Calls the printFile() method. */
    act = 1;
    t1 = new Thread(this);
    t1.start();
 }
/*
This section is executed when the end user selects the Edit->Exit command to the menu bar.
*/
 else if(ae.getSource() == exit)
 {
   /* Calls the exitFile() method. */
    action.exitFile();
 }
/*
This section is executed when the end user selects the Edit->Cut command to the menu bar.
*/
 else if(ae.getSource() == cut)
 {
   /* Calls the cutFile() method. */
    action.cutFile();
 }
/*
This section is executed when the end user selects the Edit->Copy command to the menu bar.
*/
 else if(ae.getSource() == copy)
 {
   /* Calls the copyFile() method. */
    action.copyFile();
 }
/*
This section is executed when the end user selects the Edit->Paste command to the menu bar.
*/
 else if(ae.getSource() == paste)
 {
      /* Calls the pasteFile() method. */
       action.pasteFile();
 }
/*
This section is executed when the end user selects the Edit->Date/Time command to the menu
*/
 else if(ae.getSource() == datetime)
 {
   /* Calls the dateTime() method. */
    action.dateTime();
 }
/*
This section is executed when the end user selects the Edit->Find command to the menu bar.
*/
 else if(ae.getSource() == find)
 {
   /* Calls the findFile() method. */
    action.findFile();
 }
/*
This section is executed when the end user selects the Edit->Select All command to the menu
*/
 else if(ae.getSource() == selectAll)
 {
   /* Calls the selectAllFile() method. */
    action.selectAllFile();
 }
/*
This section is executed when the end user selects the Format->Font command to the menu bar
*/
 else if(ae.getSource() == font)
 {
   /* Calls the fontFile() method. */
    action.fontFile();
 }
/*
This section is executed when the end user selects the Format->Color command to the menu ba
```

```
      */
       else if(ae.getSource() == color)
       {
         /* Calls the colorFile() method. */
          action.colorFile();
       }
      /*
      This section is executed when the end user selects the Format->Word Wrap command to the men
      */
       else if(ae.getSource() == wordWrap)
       {
         /* Calls the warpFile() method. */
          action.wrapFile();
       }
      /*
      This section is executed when the end user selects the Lock->Exclusive command to the menu
      */
       else if(ae.getSource() == exclusiveLock)
       {
          act = 2;
          t2 = new Thread(this);
          t2.start();
       }
      /*
      This section is executed when the end user selects the Lock->Share command to the menu bar.
      */
       else if(ae.getSource() == shareLock)
       {
          act = 3;
          t3 = new Thread(this);
          t3.start();
       }
      /*
      This section is executed when the end user selects the Help->Help Topics command to the men
      */
       else if(ae.getSource() == about)
       {
         /* Calls the aboutFile() method. */
          action.aboutFile();
       }
    }
    public void run()
    {
       if(act==1)
       {
          action.printFile();
    }
    else if(act==2)
    {
      /* Calls the exLockFile() method. */
       action.exLockFile();
    }
    else if(act==3)
    {
      /* Calls the shLockFile() method. */
       action.shLockFile();
    }
 }
/*
main() - This method creates the main window of the user interface and displays it.
Parameters:
args[] - Contains any command line arguments passed.
Return Value: NA
*/
 public static void main(String args[])
 {
    try
    {
      /*
      Initializes and sets the look and feel of the application to Windows look and feel.
      */
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    }
    catch(Exception e)
    {
      System.out.println("Unknown Look and Feel." + e);
    }
   /* Creates an object of the Editor class. */
    Editor ed = new Editor();
   /* Sets the size of the main window. */
    ed.setSize(725, 460);
   /* Sets the visibility of the window to TRUE. */
    ed.setVisible(true);
   /* Displays the main window. */
    ed.show();
 }
/*
Class UndoAction - Extends the AbstractAction class to perform undo operations.
```

```
    Method:
    UndoAction() - Implements the default constructor of the UndoClass.
    actionPerformed() - Performs the action event.
    update() - Updates the change made by the end user. */
     class UndoAction extends AbstractAction
     {
       /* Defines the default constructor. */
        public UndoAction()
     {
        super("Undo");
        setEnabled(false);
     }
    /*
    actionPerformed() - This method is called when the end user select the Edit->Undo command from th
    Parameters: e - Represent an object of the ActionEvent class that contains details of the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent e)
     {
        try
        {
           undoManager.undo();
        }
        catch (Exception ex)
        {
          System.out.println("Error: " + ex);
        }
        update();
        redoAction.update();
     }
    /*
    update() - This method performs undo operations.
    Parameters: NA
    Return Value: NA
    */
     protected void update()
     {
        if(undoManager.canUndo())
        {
             setEnabled(true);
             putValue("Undo", undoManager.getUndoPresentationName());
        }
        else
        {
          setEnabled(false);
          putValue(Action.NAME, "Undo");
        }
     }
 }
/*
Class UndoAction - Extends the AbstractAction class to perform redo operations.
Method:
RedoAction() - Implements the default constructor of UndoClass.
actionPerformed() - Performs the action event.
update() - Updates the changes made by the end user.
*/
 class RedoAction extends AbstractAction
 {
   /* Defines the default constructor. */
    public RedoAction()
    {
       super("Redo");
       setEnabled(false);
    }
      /*
      actionPerformed() - This method is called when the end user selects the
      Edit->Redo command from the menu bar.
      Parameters: e - Represents an object of the ActionEvent class that contains
      the details of the event.
      Return Value: NA
      */
      public void actionPerformed(ActionEvent e)
      {
         try
         {
             undoManager.redo();
         }
          catch (Exception ex)
          {
            System.out.println("Error: " + ex);
          }
          update();
          undoAction.update();
       }
      /*
      update() - This method performs the redo operations.
      Parameters: NA
```

```
        Return Value: NA
        */
        protected void update()
        {
            if(undoManager.canRedo())
            {
                 setEnabled(true);
                 putValue("Redo", undoManager.getRedoPresentationName());
            }
            else
            {
               setEnabled(false);
              putValue(Action.NAME, "Redo");
            }
          }
      }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the Editor class. This class generates the main window of the Text Editor, as shown in Figure 6-2:

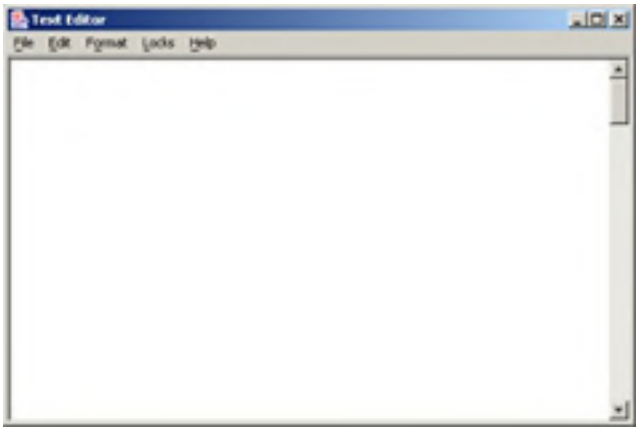


**Figure 6-2:** The Text Editor User Interface

When an end user clicks a menu item from the menu bar, the Text Editor invokes the actionPerformed() method. This method acts as an event listener and activates an appropriate class or method, based on the selected menu item. The menu options in the Text Editor application are:

- File Menu: Contains the following menu items:
  - New: Calls a newFile() method to create a new text file.
  - Open: Calls an openFile() method to open an existing text file.
  - Save: Calls a saveFile() method to save an existing text file.
  - Save As: Calls a saveAsFile() method to save a newly created text file.
  - Print: Calls the printFile() method to display the Print dialog box, based on the value assigned to the act variable.
  - Exit: Calls an exitFile() method to close the Text Editor application.
- Edit Menu: Contains the following menu items:
  - Undo: Calls an undo() method to undo the last operation performed on the file.
  - Redo: Calls a redo() method to redo the last undo operation performed on the file.
  - Cut: Calls a cutFile() method to cut a selected text from the text file.
  - Copy: Calls a copyFile() method to copy a selected text from the text file.
  - Paste: Calls a pasteFile() method to paste the cut or copied text at a specific position.
  - Find: Calls a findFile() method to find a specific word in the file.
  - Select All: Calls a selectAllFile() method to select all the contents of the file.
  - Date/Time: Calls a dateTime() method to insert the current date and time in the text file.

- Format Menu: Contains the following menu items:
  - Font: Calls a fontFile() method to display the Font dialog box.
  - Color: Calls a colorFile() method to display the Color dialog box.
  - Word Wrap: Calls a wrapFile() method to wrap the text style in the text area.
- Locks Menu: Contains the following menu items:
  - Exclusive Lock: Calls the exLockFile() method based on the value assigned to the act variable. The exLockFile() method locks the file with an exclusive lock.
  - Share Lock: Calls the shLockFile() method based on the value assigned to the act variable. The shLockFile() method locks the file with the share lock.
- Help Menu: Contains one menu item, Help Topics. This menu item calls an aboutFile() method to display the Help dialog box.

The Editor.java file creates objects of the UndoManager, UndoAction, and RedoAction classes, which perform the undo and redo operations on the file. The UndoAction and RedoAction classes contain protected update() methods that enable or disable the Undo and Redo menu items of the Edit menu, respectively.



◄ PREVIOUS NEXT ►

# Implementing the Text Editor Application

The ActionPerform.java file implements the core functionality of the Text Editor application. This file defines the methods, such as newFile(), openFile(), saveFile(), printFile(), cutFile(), exLockFile(), or shLockFile() that are invoked when an end user clicks a specific menu item from the menu bar.

Listing 6-2 shows the contents of the ActionPerform.java file:

**Listing 6-2: The ActionPerform.java File**

```
/* Imports the java.util package classes. */
import java.util.Date;
import java.util.StringTokenizer;
/* Imports the java.io package classes. */
import java.io.File;
import java.io.RandomAccessFile;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
/* Imports the javax.swing package classes. */
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import java.awt.Font;
/* Imports the java.awt.event package classes. */
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
/* Imports the java.nio package classes. */
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.channels.FileLock;
/*
The class ActionPerform - This class implements the core functionality of the Editor.
This class provides the implementation of methods that are called in the Editor class.
Fields:
   value - Contains the return value of the showDialog() method.
   option - Contains 0 or 1 value.
   content - Contains the content of a file.
   name - Contains the file name.
   word - Contains the word that to be searched.
   str - Represents a string variable.
Methods:
   newFile() - This method opens a new file.
   openFile() - This method calls the method to open an existing file.
   saveFile() - This method calls the method to save the existing file.
   saveAsFile() - This method calls the method to save the new file.
   printFile() - This method enables you to print the file.
   exitFile() - This method closes the application.
   cutFile() - This method cuts the selected text.
   copyFile() - This method copies the selected text.
   pasteFile() - This method pastes the cut or copied text.
   dateTime() - This method inserts date and time in the file.
   exLockFile() - This method locks the file with exclusive lock.
   shLockFile() - This method locks the file with share lock.
   findFile() - This method finds a specified word in the file.
   wrapFile() - This method wraps the text line in the file.
   selectAllFile() - This method selects all the contents of the file.
   aboutFile() -This method opens the help file.
   colorFile() - This method opens a Color dialog box.
   fontFile() - This method opens a Font dialog box.
   open() - This method opens a new file.
   save() - This method saves the existing file.
   saveAs() - This method saves the newly created file.
*/
public class ActionPerform
{
   int value;
   int option;
   String content = null;
   String name = null;
   String word;
   String str;
   /* Creates an instance of the JFileChooser() class. */
   JFileChooser jfc = new JFileChooser(".");
   /* Creates an instance of the Editor class. */
   public Editor ed;
   /* Creates an object of the FontClass() method. */
   public FontClass font = new FontClass();
   /* Creates an object of the ColorClass() method. */
   public ColorClass cc = new ColorClass();
   /* Creates an object of the Help() method. */
   public Help h = new Help();
   /* Creates an object of the PrintClass() method. */
   public PrintClass pc;
```

```
/*
Defines default constructor of the ActionPerform class.
*/
public ActionPerform(Editor ed)
{
   this.ed = ed;
}
/* Implementation of the newFile() method. */
 public void newFile()
{
   if(!ed.area.getText().equals("") && !ed.area.getText().equals(content))
   {
      if(name == null)
      {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the saveAs() method. */
             saveAs();
            /* Sets the text area to be NULL. */
             ed.area.setText("");
          }

          if(option == 1)
          {
             ed.area.setText("");
          }
       }
       else
       {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the save() method. */
            save();
            ed.area.setText("");
          }
          if(option == 1)
          {
             ed.area.setText("");
          }
       }
    }
    else
    {
       ed.area.setText("");
    }
   /* Sets the title of the main window. */
    ed.setTitle("Untitled - Text Editor");
   }
   /* Implementation of openFile() method. */
   public void openFile()
   {
   if(!ed.area.getText().equals("") && !ed.area.getText().equals(content))
   {
      if(name == null)
      {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the saveAs() method. */
            saveAs();
            open();
         }
         if(option == 1)
         {
            open();
         }
       }
       else
       {
         /* Shows a Confirm dialog box. */
         option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
         if(option == 0)
         {
            /* Calls the save() method. */
            save();
            /* Calls the open() method. */
            open();
          }
          if(option == 1)
         {
            /* Calls the save() method. */
            open();
          }
```

```
        }
      }
      else
      {
        /* Calls the open() method. */
         open();
      }
   }
  /* Implementation of the saveFile() method. */
   public void saveFile()
   {
      if(name == null)
      {
        /* Calls the saveAs() method. */
         saveAs();
      }
      else
      {
        /* Calls the save() method. */
         save();
      }
   }
  /* Implementation of the saveAsFile() method. */
   public void saveAsFile()
   {
     /* Calls the saveAs() method. */
      saveAs();
   }
  /* Implementation of printFile() method. */
   public void printFile()
   {
        /*
        Creates an instance of the PrintClass that inputs the ed.area component and ed
        object of the Editor class as parameters.
        */
         pc = new PrintClass(ed.area, ed);
        /* Calls the print() method. */
         pc.print();
   }
  /* Implementation of the exitFile() method. */
   public void exitFile()
   {
     if(!ed.area.getText().equals("") && !ed.area.getText().equals(content))
      {
         if(name == null)
        {
           /* Shows a Confirm dialog box. */
           option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
           if(option == 0)
           {
              /* Calls the saveAs() method. */
              saveAs();
              /* Closes the main application. */
              System.exit(0);
           }
           if(option == 1)
           {
              System.exit(0);
           }
        }
        else
        {
           /* Shows a Confirm dialog box. */
           option = JOptionPane.showConfirmDialog(null,"Do you want to save the file?");
           if(option == 0)
           {
              /* Calls the save() method. */
              save();
              System.exit(0);
           }
           if(option == 1)
           {
              System.exit(0);
           }
        }
     }
     else
     {
         System.exit(0);
     }
  }
  /* Implementation of the cutFile() method. */
   public void cutFile()
   {
     /* Cuts the selected area from the text area. */
      ed.area.cut();
   }
     /* Implementation of the copyFile() method. */
```

```
   public void copyFile()
   {
      /* Copies the selected area from the text area. */
      ed.area.copy();
   }
   /* Implementation of the pasteFile() method. */
   public void pasteFile()
   {
      /* Pastes the cut/copied area to the text area. */
       ed.area.paste();
   }
   /* Implementation of the selectAllFile() method. */
   public void selectAllFile()
   {
      /* Selects all the content of the text area. */
       ed.area.selectAll();
    }
   /* Implementation of the findFile() method. */
    public void findFile()
    {
       try
      {
         /* Shows a word input dialog box. */
         word = JOptionPane.showInputDialog("Type the word to find");
         while(ed.area.getText().indexOf(word) == -1)
         {
         /* Shows a message dialog box. */
         JOptionPane.showMessageDialog(null,"Word not found!","No
         match",JOptionPane.WARNING_MESSAGE);
         word = JOptionPane.showInputDialog(" Type the word to find");
    }
    /* Selects the word in the text area. */
    ed.area.select(ed.area.getText().indexOf(word),
    ed.area.getText().indexOf(word) + word.length());
    }
    catch(Exception ex)
    {
      /* Showss an error message dialog box. */
      JOptionPane.showMessageDialog
      (null,"Search canceled","Abourted",JOptionPane.WARNING_MESSAGE);
    }
 }
/* Implementation of the dateTime() method. */
 public void dateTime()
 {
   /* Creates an object of the Date class. */
    Date d = new Date();
   /* Converts time to string and display it. */
    String str = d.toLocaleString();
   /* Appends the date and time in the text area. */
    ed.area.append(str);
 }
/* Implementation of the fontFile() method. */
 public void fontFile()
 {
   /* Shows the Font dialog box. */
   font.setVisible(true);
   font.pack();
   /* Calls the getOk() method of FontClass. */
   font.getOk().addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         /*
         Sets the font name, size, and style of the file text.
         */
         ed.area.setFont(font.font());
         font.label.setFont(new Font("Arial",Font.PLAIN,15));
         font.setVisible(false);
      }
   });
   /* Calls the getCancel() method of FontClass. */
   font.getCancel().addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent ae)
      {
         /* Hides the Font dialog box. */
         font.label.setFont(new Font("Arial",Font.PLAIN,15));
         font.setVisible(false);
      }
   });
}
/* Implementation of the colorFile() method. */
public void colorFile()
{
   /* Shows the Color dialog box. */
   cc.setVisible(true);
```

```
      cc.pack();
      /* Calls the getOk() method of ColorClass. */
      cc.getOk().addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent ae)
         {
            /* Sets the color of the file text. */
            ed.area.setForeground(cc.color());
            cc.setVisible(false);
         }
      });
      /* Calls the getCancel() method of ColorClass. */
      cc.getCancel().addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent ae)
         {
            /* Hides the Color dialog box. */
            cc.setVisible(false);
         }
      });
   }
   /* Implementation of the wrapFile() method. */
   public void wrapFile()
   {
      if(ed.wordWrap.getState() == true)
      {
         /* Sets the line and word wrap style to TRUE. */
         ed.area.setLineWrap(true);
         ed.area.setWrapStyleWord(true);
      }
      else
      {
         /* Sets the line and word wrap style to FALSE. */
         ed.area.setLineWrap(false);
         ed.area.setWrapStyleWord(false);
      }
   }
   /* Implementation of the aboutFile() method. */
   public void aboutFile()
   {
      /* Shows the Help dialog box. */
      h.setVisible(true);
      /* Calls the getOk() method of Help. */
      h.getOk().addActionListener(new ActionListener()
      {
      public void actionPerformed(ActionEvent ae)
      {
         /* Hides the Help dialog box. */
         h.setVisible(false);
      }
   });
   }
   /* Implementation of exLockFile() method. */
   public void exLockFile()
   {
      try
      {
         /* Creates an object of the File class. */
         File file = new File(str);
         /* Creates an object of the RandomAccessFile class. */
         RandomAccessFile raf = new RandomAccessFile(file, "rw");
         /* Creates an object of the FileChannel class. */
         FileChannel channel = raf.getChannel();
         if(ed.exclusiveLock.getState() == true)
         {
            /*
            Uses the file channel to create an exclusive lock on the file.
            The lock() method blocks until it can retrieve the lock.
            */
            FileLock lock = channel.lock(0, Long.MAX_VALUE, false);
         }
         else if(ed.exclusiveLock.getState() == false)
         {
            /* Closes the channel. */
            channel.close();
         }
      }
      catch (Exception e)
      {
         System.out.println("Error: " + e);
      }
   }
   /* Implementation of the shLockFile() method. */
   public void shLockFile()
   {
      try
      {
         /* Creates an object of the File class. */
```

```
            File file = new File(str);
            /* Create an object of the RandomAccessFile class. */
            RandomAccessFile raf = new RandomAccessFile(file, "rw");
            /* Creates an object of the FileChannel class. */
            FileChannel channel = raf.getChannel();
            if(ed.shareLock.getState() == true)
            {
               /*
               Uses the file channel to create a share lock on the file.
               The lock() method blocks until it can retrieve the lock.
               */
               FileLock lock = channel.lock(0, Long.MAX_VALUE, true);
            }
            else if(ed.shareLock.getState() == false)
            {
               /* Closes the channel. */
               channel.close();
            }
         }
         catch (Exception e)
         {
            System.out.println("Error:" + e);
         }
      }
      /* Implementation of open() method. */
      public void open()
      {
         value = jfc.showOpenDialog(ed);
         if(value == JFileChooser.APPROVE_OPTION)
         {
            ed.area.setText(null);
            ed.exclusiveLock.setEnabled(true);
            ed.shareLock.setEnabled(true);
            try
            {
               /* Gets the file name. */
               name = jfc.getSelectedFile().getPath();
               /* Creates an object of FileInputStream class. */
               FileInputStream fin = new FileInputStream(jfc.getSelectedFile());
               /*
               Creates a channel and get the channel from the FileInputStream
               */
               FileChannel fchan = fin.getChannel();
               /* Stores the size of file in long variable */
               long fsize = fchan.size();
               /*
               Creates an object of the Bytebuffer class and allocate the size of this byte buffer
               */
               ByteBuffer buff = ByteBuffer.allocate((int)fsize);
               /* Reads the bytes from channel to byte buffer*/
               fchan.read(buff);
               /* Rewinds the byte buffer */
               buff.rewind();
               /*
               Returns the byte buffer into an array and converts this array into string
               */
               String str = new String(buff.array());
               /* Appends this string to the text area */
               ed.area.append(str);
               content = ed.area.getText();
               /* Closes the channel */
               fchan.close();
               /* Closes the input stream */
               fin.close();
            }
            catch(IOException ioe)
            {
               System.err.println("I/O Error on Open");
            }
            /* Sets the title of the main window */
            ed.setTitle(jfc.getSelectedFile().getAbsolutePath()+ " - Text Editor");
            str = jfc.getSelectedFile().getAbsolutePath();
         }
      }
      /* Implementation of save() method. */
      public void save()
      {
         if(value == JFileChooser.APPROVE_OPTION)
         {
            try
            {
               /* Creates an object of FileOutputStream class */
               FileOutputStream fout = new FileOutputStream(jfc.getSelectedFile());
               /* Creates an object of PrintWriter class */
               PrintWriter pw = new PrintWriter(fout);
               content = ed.area.getText();
               /* Creates an object of StringTokenizer class */
               StringTokenizer st=new StringTokenizer(content,System.getProperty("line.separator"));
```

```
            while(st.hasMoreTokens())
            {
               pw.println(st.nextToken());
            }
            /* Closes print writer */
            pw.close();
            /* Closes output stream */
            fout.close();
         }
         catch(IOException ioe)
         {
            System.err.println("I/O Error on Save");
         }
         /* Sets the title of the window */
         ed.setTitle(jfc.getSelectedFile().getName()+ " - Text Editor");
         str = jfc.getSelectedFile().getAbsolutePath();
      }
   }
   /* Implementation of saveAs() method. */
   public void saveAs()
   {
      jfc.setDialogTitle("Save As");
      value = jfc.showSaveDialog(ed);
      if(value == JFileChooser.APPROVE_OPTION)
      {
         try
         {
            /* Creates an object of FileOutputStream class */
            FileOutputStream fout = new FileOutputStream(jfc.getSelectedFile() + ".txt");
            /* Creates an object of PrintWriter class */
            PrintWriter pw = new PrintWriter(fout);
            content = ed.area.getText();
            name = jfc.getSelectedFile().getPath();
            /* Creates an object of StringTokenizer class */
            StringTokenizer st=new StringTokenizer(content,System.getProperty("line.separator"));
            while(st.hasMoreTokens())
            {
               pw.println(st.nextToken());
            }
            /* Closes print writer */
            pw.close();
            /* Closes output stream */
            fout.close();
         }
         catch(IOException ioe)
         {
            System.err.println ("I/O Error on Save");
         }
         /* Sets the title of the window */
         ed.setTitle(jfc.getSelectedFile().getAbsolutePath() + " - Text Editor");
         str = jfc.getSelectedFile().getAbsolutePath();
      }
   }
}
```

Download this Listing.

In the above code, the ActionPerform class creates instances of the FontClass, PrintClass, Help, PrintClass, Editor, and JFileChooser classes. The ActionPerform() constructor accepts the Editor class as an input parameter. The methods defined in the above listing are:

- `newFile()`: Checks whether or not the text area contains text. If the text area contains text, the newFile() method checks the file name using the name variable. If the name returns a null file name, the newFile() method calls the saveAs() method to save the file as a new one; otherwise the newFile() method calls the save() method to save the existing file.
- `openFile()`: Checks whether or not the text area contains text. If the text area contains text, the openFile() method checks the file name using the name variable. If the name returns a null file name, the openFile() method calls the saveAs() and open() methods to open a file; otherwise the openFile() method calls the save() and open() methods to open a file.
- `saveFile()`: Checks the file name using the name variable. If the name returns a null file name, the saveFile() method calls the saveAs() method to save the file as a new one; otherwise the saveFile() method calls the save() method to save the existing file.
- `saveAsFile()`: Calls the saveAs() method to save the file.
- `printFile()`: Creates an instance of the PrintClass class. The printFile() method calls the print() method to print the file.
- `exitFile()`: Checks whether or not the text area contains text. If the text area contains text, the exitFile() method checks the file name using the name variable. If the name returns a null file name, the exitFile() method calls the saveAs() and System.exit(0) method to close the file; otherwise the exitFile() method calls the save() and System.exit(0) methods to close the file.
- `cutFile()`: Calls the cut() method to cut the selected area of the text area.

- `copyFile()`: Calls the copy() method to copy the selected area of the text area.
- `pasteFile()`: Calls the paste() method to paste the copied or cut text at the specific position in the text area.
- `selectAllFile()`: Calls the selectAllFile() method to select all the contents of the text area.
- `dateTime()`: Creates an instance of the Date class and inserts the current date and time at the curser position in the text area.
- `findFile()`: Opens an Input dialog box to input the word that is to be searched in the file. If the findFile() method finds the specified word in the file, it selects the word in the text area; otherwise it opens a Message dialog box.
- `fontFile()`: Opens the Font dialog box and calls the getOk() method to set the font of the text that is displayed in the text area. The fontFile() method calls the getCancel() method to close the Font dialog box.
- `colorFile()`: Opens the Color dialog box and calls the getOk() method to set the color of the text that is displayed in the text area. The fontFile() method calls the getCancel() method to close the Color dialog box.
- `wordWrap()`: Checks the status of the Word Wrap check box. If the Word Wrap check box returns true, the wordWrap() method sets the line and word wrap styles to true; otherwise it sets the line and word wrap styles to false.
- `exLockFile()`: Creates an instance of the File class. The exLockFile() method creates an instance of the RandomAccessFile class and calls the getChannel() method of the FileChannel class. The exLockFile() method then checks the status of the Exclusive Lock check box. If the Exclusive Lock check box returns true, the exLockFile() method locks the currently opened file with an exclusive lock.
- `shLockFile()`: Creates an instance of the File class. The exLockFile() creates an instance of RandomAccessFile class and calls the getChannel() method of the FileChannel class. The shLockFile() method then checks the status of the Share Lock check box. If the Share Lock check box returns true, the shLockFile() method locks the currently opened file with a share lock.
- `aboutFile()`: Opens a Help dialog box and calls the getOk) method to close the Help dialog box.
- `open()`: Displays an Open dialog box that is to open a file from a specific location. The open() method creates an instance of the FileInputStream class and calls the getChannel() method of the FileChannel class. The open() method then stores the size of the file in a variable of long type, creates an object of the ByteBuffer class, and allocates the size to the byte buffer. Next, the open() method reads the bytes from the channel to the byte buffer and rewinds the byte buffer. Finally, the open() method returns the byte buffer into an array, converts this array into a string, and appends this string to the text area.
- `save()`: Creates an instance of the FileOutputStream class. The save() method creates an instance of the PrintWriter class and an object of the StringTokenizer class. This method then writes the file at a specified location using the PrintWriter and StringTokenizer class instances.
- `saveAs()`: Displays a Save As dialog box that is to save a new file at a specific location using save() method.



PREVIOUS NEXT

# Creating the Print Dialog Box

You can use the PrintClass.java file to open the default Print dialog box for the Text Editor application. In this dialog box, you can specify the print range, number of copies, and page property before you print the file.

Listing 6-3 shows the contents of the PrintClass.java file:

**Listing 6-3: The PrintClass.java File**

```
/* Imports java.awt.print package class. */
import java.awt.print.PrinterJob;
import java.awt.print.PrinterException;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
/* Imports java.awt package classes. */
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Graphics2D;
/* Imports javax.swing package class. */
import javax.swing.RepaintManager;
/*
class PrintClass - This class implements Printable interface and enable the end user to print the do
Fields:
   compPrint - Contains the document that to be printed.
Methods:
   print() - This method prints the current document opened in the text editor window
*/
public class PrintClass implements Printable
{
   /* Declares the object of Component class. */
   private Component compPrint;
   public Editor editor;
   /* Defines the default constructor. */
   public PrintClass(Component compPrint, Editor editor)
   {
      this.compPrint = compPrint;
      this.editor = editor;
   }
   /*
   print() - This method prints the document
   Parameters: lse - NA
   Return Value: NA
   */
   public void print()
   {
      /* Creates an object of the PrinterJob class */
      PrinterJob printer = PrinterJob.getPrinterJob();
      /* Sets the object of PrinterJob class to printable */
      printer.setPrintable(this);
      if(printer.printDialog())
      {
         try
         {
            /* Prints the document */
            printer.print();
         }
         catch(PrinterException pe)
         {
            System.out.println("Error: " + pe);
         }
      }
      editor.show();
   }
   /*
   print() - This method opens a Print dialog box
   Parameters:
   g - Represents the object of Graphics class
   format - Represents the object of PageFormat class
   index - Represents an index of page
   Return Value: int PAGE_EXIST
   */
   public int print(Graphics g, PageFormat format, int index)
   {
      if(index > 0)
      {
         return(NO_SUCH_PAGE);
      }
      else
      {
         /*
         Creates an object of the Graphics2D class and converts simple graphics to 2D graphics
         */
         Graphics2D g2d = (Graphics2D)g;
         /*
```

```
            Translates the origin of the Graphics2D context to the point (x, y)
            in the current coordinate system
            */
            g2d.translate(format.getImageableX(), format.getImageableY());
            /* Creates the object of the RepaintManager class */
            RepaintManager manager1 = RepaintManager.currentManager(compPrint);
            /* Sets the double buffer to FALSE */
            manager1.setDoubleBufferingEnabled(false);
            /* Paints the component */
            compPrint.paint(g2d);
            /* Creates the object of RepaintManager class */
            RepaintManager manager2 = RepaintManager.currentManager(compPrint);
            /* Sets the double buffer to TRUE */
            manager2.setDoubleBufferingEnabled(true);
            /* Returns the PAGE_EXIST value */
            return(PAGE_EXISTS);
        }
    }
}
```

Download this Listing.

In the above code, the PrintClass() constructor of the PrintClass takes two arguments, compPrint and editor. The compPrint argument is an object of the Component class that contains the printing content. The editor argument is an object of the Editor class.

The print() method creates an object of the PrinterJob class. This method then sets the printable property on the object of the PrinterClass to open the Print dialog box, as shown in Figure 6-3:

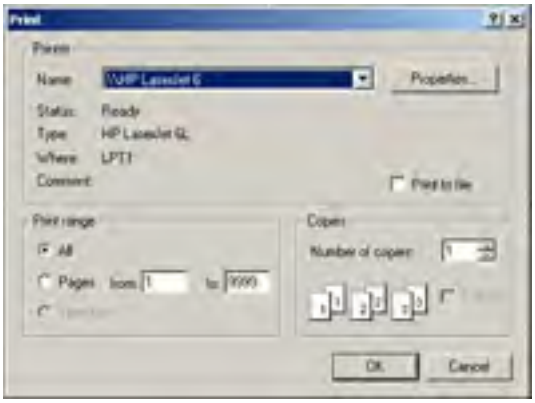


**Figure 6-3:** The Print Dialog Box

The Printer panel in the Print dialog box displays the status of the printer selected by end users in the Name combo box. The Properties button in this panel helps set the layout and quality of the page. In the Print range panel, end users can select a radio button to specify the print range. The Copies panel provides a list box that allows end users to specify the number of copies of a document they want to print.

When end users click the OK button in the Print dialog box, the print() method of PrinterJob class is invoked. This method prints the file.

If end users click the Cancel button of the Print Dialog box, the editor.show() method is invoked. This method closes the Print dialog box and sets the focus to the main window of the Text Editor application.

If end users click the Properties button of the Help dialog box, the print(Graphics g, PageFormat format, int index) method is invoked. This method uses three objects as parameters: g object of the Graphics class, format object of the PageFormat class, and index object of the Integer class. The print() method creates the object of the Graphics2D class. The object of the Graphics2D class translates the origin of the Graphics2D context to the point x, y in the current coordinate system. The print() method then creates the object of the RepaintManager class and sets the double buffer property to false. Next, the print() method paints the component using the paint() method. The print() method again creates the object of the RepaintManager class and sets the double buffer property to TRUE. Finally, the print() method returns the PAGE_EXISTS variable.

# Creating the Font Dialog Box

The FontClass.java file allows you to create the Font dialog box for the Text Editor application. End users can use this dialog box to specify the font type, size, and style to change the font of the text file.

Listing 6-4 shows the contents of the FontClass.java file:

**Listing 6-4: The FontClass.java File**

```
/* Imports javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
/* Imports java.awt package classes. */
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.FlowLayout;
import java.awt.Font;
/* Imports javax.swing.event package classes. */
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
/*
class FontClass - This class creates a Font dialog box that enables the
end user to change the font, size, and type of the text.
Fields:
   fontLabel - Contains the content of Font label.
   sizeLabel - Contains the content of Size label.
   typeLabel - Contains the content of Type label.
   previewLabel - Contains the content of preview.
   label - Contains the preview contents.
   fontText - Contains the selected font name.
   typeText - Contains the selected font type name.
   sizeText - Contains the selected font size name.
   fontScroll - Contains the Font list.
   typeScroll - Contains the Type list.
   sizeScroll - Contains the Size list.
   fontList - Contains all the available font.
   typeList - Contains all the available types of font style.
   sizeList - Contains all the available font size.
   ok - Creates an OK button.
   cancel - Creates a cancel button.
Methods:
   getOK() - This method returns the OK button object
   getCancel() - This method returns the Cancel button object
   valueChanged() - This method is invoked when an end user select the item from the List box.
   font() - This method returns the font
*/
public class FontClass extends JDialog implements ListSelectionListener
{
   /* Declares the objects of the JPanel class. */
   JPanel pan1;
   JPanel pan2;
   JPanel pan3;
   /* Declares the objects of the JLabel class. */
   JLabel fontLabel;
   JLabel sizeLabel;
   JLabel typeLabel;
   JLabel previewLabel;
   /* Declares the objects of the JTextField class. */
   JTextField label;
   JTextField fontText;
   JTextField sizeText;
   JTextField typeText;
   /* Declares the objects of the JScrollPane class. */
   JScrollPane fontScroll;
   JScrollPane typeScroll;
   JScrollPane sizeScroll;
   /* Declares the objects of the JList class. */
   JList fontList;
   JList sizeList;
   JList typeList;
   /* Declares the objects of the JButton class. */
   JButton ok;
   JButton cancel;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   /* Defines the default constructor. */
   public FontClass()
```

```
{
   /* Sets the title of the Font dialog. */
   setTitle("Font Dialog");
   /* Sets the size of Font dialog. */
   setSize(300, 400);
   /* Sets resizable button to false. */
   setResizable(false);
   /* Initializes the object of the GridBagLayout class. */
   gbl = new GridBagLayout();
   /* Sets the Layout. */
   getContentPane().setLayout(gbl);
   /* Creates an object of GridBagConstraints class. */
   gbc = new GridBagConstraints();
   /*
   Initializes the Font label object and add it to the 1, 1, 1, 1 positions with WEST alignment.
   */
   gbc.gridx = 1;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontLabel = new JLabel("Fonts: ");
   getContentPane().add(fontLabel, gbc);
   /*
   Initializes the Size label object and adds it to the 2, 1, 1, 1 positions with WEST alignment.
   */
   gbc.gridx = 2;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeLabel = new JLabel("Sizes: ");
   getContentPane().add(sizeLabel, gbc);
   /*
   Initializes the Types label object and adds it to the 3, 1, 1, 1 positions with WEST alignment
   */
   gbc.gridx = 3;
   gbc.gridy = 1;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeLabel = new JLabel("Types: ");
   getContentPane().add(typeLabel, gbc);
   /*
   Initializes the Font text field object and adds it to the 1, 2, 1, 1 positions with WEST align
   */
   gbc.gridx = 1;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   fontText = new JTextField("Arial", 12);
   getContentPane().add(fontText, gbc);
   /*
   Initializes the Size text field object and adds it to the 2, 2, 1, 1 positions with WEST align
   */
   gbc.gridx = 2;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   sizeText = new JTextField("8", 4);
   getContentPane().add(sizeText, gbc);
   /*
   Initializes the Types text field object and adds it to the 3, 2, 1, 1 positions with WEST alig
   */
   gbc.gridx = 3;
   gbc.gridy = 2;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   typeText = new JTextField("Regular", 6);
   getContentPane().add(typeText, gbc);
   /*
   Initializes the Font list object and add it to the Font scroll pane object.
   Adds this scroll pane object to 1, 3, 1, 1 positions with WEST alignment
   */
   gbc.gridx = 1;
   gbc.gridy = 3;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.WEST;
   String[] fonts = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames
   fontList = new JList(fonts);
   fontList.setFixedCellWidth(110);
   fontList.addListSelectionListener(this);
   fontList.setSelectedIndex(0);
   fontScroll = new JScrollPane(fontList);
```

```
        getContentPane().add(fontScroll, gbc);
        /*
        Initializes the Size list object and add it to the Size scroll pane object.
        Adds this scroll pane object to 2, 3, 1, 1 positions with WEST alignment
        */
        gbc.gridx = 2;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        String[] sizes = {"8","10","12","14","16","18","20","24","28","32","48","72"};
        sizeList = new JList(sizes);
        sizeList.setSelectedIndex(0);
        sizeList.setFixedCellWidth(20);
        sizeList.addListSelectionListener(this);
        sizeScroll = new JScrollPane(sizeList);
        getContentPane().add(sizeScroll, gbc);
        /*
        Initializes the Types list object and adds it to the Types scroll pane object.
        Next, adds this scroll pane object to 3, 3, 1, 1 positions with WEST alignment
        */
        gbc.gridx = 3;
        gbc.gridy = 3;
        gbc.gridwidth = 1;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.WEST;
        String[] types = {"Regular", "Bold", "Italic", "Bold Italic"};
        typeList = new JList(types);
        typeList.setFixedCellWidth(60);
        typeList.addListSelectionListener(this);
        typeList.setSelectedIndex(0);
        typeScroll = new JScrollPane(typeList);
        getContentPane().add(typeScroll, gbc);
        /*
        Initializes the preview label and adds it to 1,4,3,1 positions with CENTER alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 4;
        gbc.gridwidth = 3;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        pan1 = new JPanel();
        pan1.setLayout(new FlowLayout());
        previewLabel = new JLabel("Preview:");
        pan1.add(previewLabel);
        getContentPane().add(pan1, gbc);
        /*
        Initializes the preview text field and adds it to 1,5,3,1 positions with CENTER alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 5;
        gbc.gridwidth = 3;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        pan2 = new JPanel();
        pan2.setLayout(new FlowLayout());
        label = new JTextField("AaBaCcDdeEfFgGhHjJ", 15);
        label.setEditable(false);
        label.setBorder(BorderFactory.createEtchedBorder());
        label.setFont(new Font("Arial",Font.PLAIN,20));
        pan2.add(label);
        getContentPane().add(pan2, gbc);
        /*
        Initializes the OK and Cancel button and adds these two buttons to the panel.
        Sets layout of the panel to FlowLayout. Now add this panel to the 1, 6, 4, 1
        positions with CENTER alignment.
        */
        gbc.gridx = 1;
        gbc.gridy = 6;
        gbc.gridwidth = 3;
        gbc.gridheight = 1;
        gbc.anchor = GridBagConstraints.CENTER;
        pan3 = new JPanel();
        pan3.setLayout(new FlowLayout());
        ok = new JButton("OK");
        cancel = new JButton("Cancel");
        pan3.add(ok);
        pan3.add(cancel);
        getContentPane().add(pan3, gbc);
    }
    /*
    valueChanged() - This method is called when the user selects any menu item from the menu bar.
    Parameters: lse - a ListSelectionEvent object containing details of the event.
    Return Value: NA
    */
    public void valueChanged(ListSelectionEvent lse)
```

```
    {
       try
       {
          /*
          This section is executed, when end user selects the item from Font list.
          */
          if(lse.getSource() == fontList)
          {
             Font f1 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
             Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
             fontText.setText(String.valueOf(fontList.getSelectedValue()));
             label.setFont(f1);
          }
          /*
          This section is executed, when end user selects the item from Size list.
          */
          else if(lse.getSource() == sizeList)
          {
             Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
             Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
             sizeText.setText(String.valueOf(sizeList.getSelectedValue()));
             label.setFont(f2);
          }
          /*
          This section is executed, when end user selects the item from Type list.
          */
          else if(lse.getSource() == typeList)
          {
             Font f2 = new Font(String.valueOf(fontList.getSelectedValue()),typeList.getSelectedIndex
             Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
             typeText.setText(String.valueOf(typeList.getSelectedValue()));
             label.setFont(f2);
          }
       }
       catch(Exception nfe){}
    }
    /*
    font() - This method is set the font of the file text.
    Parameters: NA
    Return Value: font
    */
    public Font font()
    {
       /* Creates an object of the Font class. */
       Font font = new Font(String.valueOf(fontList.getSelectedValue()), typeList.getSelectedIndex(),
       Integer.parseInt(String.valueOf(sizeList.getSelectedValue())));
       /* Returns the font object */
       return font;
    }
    /*
    getOk() method - This method is invoked when end user click the OK button of the Font dialog box
    parameter - NA
    return value - ok
    */
    public JButton getOk()
    {
       return ok;
    }
    /*
    getCancel() method - This method is invoked when end user click the Cancel button of the Font dia
    parameter - NA
    return value - cancel
    */
    public JButton getCancel()
    {
       return cancel;
    }
}
```

Download this Listing.

In the above code, the FontClass() constructor creates the Font dialog box, as shown in Figure 6-4:


**Figure 6-4:** The Font Dialog Box

When an end user selects any item from the list box of the Font dialog box, the Font class invokes the valueChanged() method. This method acts as an event listener and activates the appropriate method to set the font of the preview label.

The FontClass defines two methods, getOK() and getCancel(), to perform the OK and Cancel operations. These methods return the object of the Button class.

When an end user clicks the OK button in the Font dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Font dialog box to false and changes the font of the file text, based on the selected font type, size, and style.

If an end user clicks the Cancel button in the Font dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Font dialog box to false and closes the Font dialog box.



◄ PREVIOUS NEXT ►

# Creating the Color Dialog Box

You can create the Color dialog box for the Text Editor application using the ColorClass.java file. This dialog box displays a set of labels, sliders, and buttons using which you can specify the RGB values to change the color of the text file.

Listing 6-5 shows the contents of the ColorClass.java file:

**Listing 6-5: The ColorClass.java File**

```
/* Imports javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JSlider;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
/* Imports java.awt package classes. */
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.Color;
/* Imports javax.swing.event package classes. */
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
/*
class ColorClass - This class creates a Color dialog box that enables the end user to
change the color of the file text.
Fields:
   panel - Contains all the components of the Color dialog.
   redLabel - Contains the content of Red label.
   greenLabel - Contains the content of Green label.
   blueLabel - Contains the content of Blue label.
   previewLabel - Contains the content of Preview label.
   redSlider - Enables the end user to select the red value.
   greenSlider - Enables the end user to select the green value.
   blueSlider - Enables the end user to select the blue value.
   labelText - Contains the content of preview text.
   ok - Creates an OK button.
   cancel - Creates a cancel button.
   r - Stores the Red value.
   g - Stores the Green value.
   b - Stores the Blue value.
Methods:
   getOK() - This method returns the OK button object.
   getCancel() - This method returns the Cancel button object.
   stateChanged() - This method is invoked when an end user slide slider.
   color() - This method returns the color.
   */
public class ColorClass extends JDialog implements ChangeListener
{
   /* Declares the object of the JPanel class. */
   JPanel panel;
   /* Declares the objects of the JLabel class. */
   JLabel redLabel;
   JLabel greenLabel;
   JLabel blueLabel;
   JLabel previewLabel;
   /* Declares the objects of the JSlider class */
   JSlider redSlider;
   JSlider greenSlider;
   JSlider blueSlider;
   /* Declares the object of the JTextField class */
   JTextField labelText;
   /* Declares the objects of the JButton class */
   JButton ok;
   JButton cancel;
   /* Declares the integer for storing the RGB values */
   int r = 0;
   int g = 0;
   int b = 0;
   public Editor ed;
   /* Defines the default constructor of the ColorClass class. */
   public ColorClass()
   {
      /* Sets the title of the Font dialog. */
      setTitle("Color Dialog");
      /* Sets resizable button to false. */
      setResizable(false);
      /* Initializes the object of the JPanel class. */
      panel = new JPanel();
      /* Sets the Layout as GridLayout.*/
      panel.setLayout(new GridLayout(5,2,1,1));
      /* Adds the panel to Color dialog frame */
      getContentPane().add(panel);
```

```
        /* Initializes and adds Red label to the panel. */
        redLabel = new JLabel("Red: ");
        panel.add(redLabel);
        /* Initializes and adds Red slider to the panel. */
        redSlider = new JSlider(0, 255, 1);
        panel.add(redSlider);
        /* Sets Border to the Red slider. */
        redSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Adds state change listener to Red slider. */
        redSlider.addChangeListener(this);
        /* Initializes and adds Green label to the panel. */
        greenLabel = new JLabel("Green: ");
        panel.add(greenLabel);
        /* Initializes and adds Green slider to the panel. */
        greenSlider = new JSlider(0, 255, 1);
        panel.add(greenSlider);
        /* Sets Border to the Green slider. */
        greenSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Adds state change listener to Green slider. */
        greenSlider.addChangeListener(this);
        /* Initializes and adds Blue label to the panel. */
        blueLabel = new JLabel("Blue: ");
        panel.add(blueLabel);
        /* Initializes and adds Blue slider to the panel. */
        blueSlider = new JSlider(0, 255, 1);
        panel.add(blueSlider);
        /* Sets Border to the Blue slider. */
        blueSlider.setBorder(BorderFactory.createEtchedBorder());
        /* Adds state change listener to Blue slider. */
        blueSlider.addChangeListener(this);
        /* Initializes and add Preview label to the panel. */
        previewLabel = new JLabel("Preview: ");
        panel.add(previewLabel);
        /*
        Initializes and adds Preview text field to the panel.
        */
        labelText = new JTextField(10);
        panel.add(labelText);
        /* Initializes and adds OK button to the panel. */
        ok = new JButton("OK");
        panel.add(ok);
        /* Initializes and adds Cancel button to the panel. */
        cancel = new JButton("Cancel");
        panel.add(cancel);
    }
    /*
    stateChanged() - This method is called when the user slides any slider of the Color dialog box.
    Parameters: ce - Represents an object of the ChangeEvent class that contains the details of the e
    Return Value: NA
    */
    public void stateChanged(ChangeEvent ce)
    {
        /*
        This section is executed when end user slides the Red slider.
        */
        if(ce.getSource() == redSlider)
        {
            r = redSlider.getValue();
        }
        /*
        This section is executed, when end user slides the Green slider.
        */
        else if(ce.getSource() == greenSlider)
        {
            g = greenSlider.getValue();
        }
        /*
        This section is executed, when end user slides the Blue slider.
        */
        else if(ce.getSource() == blueSlider)
        {
            b = blueSlider.getValue();
        }
        /* Creates the object of the Color class. */
        Color c = new Color(r, g, b);
        /*
        Sets the background color of the preview text field.
        */
        labelText.setBackground(c);
    }
    /*
    color() - This method is set color of the file text.
    Parameters: NA
    Return Value: color
    */
    public Color color()
    {
        Color color = new Color(redSlider.getValue(), greenSlider.getValue(), blueSlider.getValue());
```

```
        return color;
    }
    /*
    getOk() method - This method is invoked when end user click the OK button of the Color dialog box
    parameter - NA
    return value - ok
    */
    public JButton getOk()
    {
        return ok;
    }
    /*
    getCancel() method - This method is invoked when end user click the Cancel
    button of the Color dialog box.
    parameter - NA
    return value - cancel
    */
    public JButton getCancel()
    {
        return cancel;
    }
}
```

Download this Listing.

In the above code, the ColorClass() constructor creates the Color dialog box, as shown in Figure 6-5:


**Figure 6-5:** The Color Dialog Box

When the end user slides any slider in the Color dialog box, the ColorClass class invokes the stateChanged() method. This method acts as an event listener and activates the appropriate method to set the color of the preview label.

The ColorClass class defines two methods, getOk() and getCancel(), to perform the OK and Cancel operations. These methods return the object of the Button class.

When the end user clicks the OK button on the Color dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Color dialog box to false and changes the color of the text file based on the selected RGB values.

If the end user clicks the Cancel button on the Color dialog box, the actionPerformed() method is invoked in the ActionPerform class. This method sets the visibility of the Color dialog box to false and closes the Color dialog box.



◄ PREVIOUS NEXT ►

# Creating the Help File

You can to create the Help dialog box of the Text Editor application using the Help.java file. This dialog box displays a list box and a text area. When you select a help topic name from the list box, information covered under that topic is displayed in the text area.

Listing 6-6 shows the contents of the Help.java file:

**Listing 6-6: The Help.java File**

```
/* Imports javax.swing package classes. */
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JList;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JDialog;
/* Imports java.awt package classes. */
import java.awt.FlowLayout;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
/* Imports javax.swing.event package classes. */
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
/*
class Help - This class creates an Help dialog that enables the end user to read the user manual of
Fields:
   panel - Contains the document that to be printed.
   title - Contains the content of Title label.
   label1 - Contains the content of Help Topic label.
   label2 - Contains the content of Topic Details label.
   area - Displays the detail of sleeted topic.
   ok - Creates an OK button.
Methods:
valueChanged() - This method is invoked when an end user select the item from the List box.
*/
public class Help extends JDialog implements ListSelectionListener
{
   /* Declares the object of the JPanel class. */
   JPanel panel;
   /* Declares the objects of the JLabel class. */
   JLabel title;
   JLabel label1;
   JLabel label2;
   /* Declares the objects of the JScrollPane class. */
   JScrollPane listScroll;
   JScrollPane areaScroll;
   /* Declares the object of the JList class. */
   JList list;
   /* Declares the object of the JTextArea class. */
   JTextArea area;
   /* Declares the object of the JButton class. */
   JButton ok;
   GridBagLayout gbl;
   GridBagConstraints gbc;
   String str = null;
   /* Defines the default constructor. */
   public Help()
   {
      /* Sets the title of the Help dialog. */
      setTitle("Help");
      /* Sets the size of the Help dialog. */
      setSize(600, 300);
      /* Sets resizable button to false. */
      setResizable(false);
      /* Initializes the object of the GridBagLayout class. */
      gbl = new GridBagLayout();
      /* Sets the Layout. */
      getContentPane().setLayout(gbl);
      /* Creates an object of GridBagConstraints class. */
      gbc = new GridBagConstraints();
      /*
      Initializes the Help Topic label object and adds it to the 1, 1, 1, 1 positions with WEST alig
      */
      gbc.gridx = 1;
      gbc.gridy = 1;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      label1 = new JLabel("Help Topics:");
      getContentPane().add(label1, gbc);
      /*
      Initializes the Topic Details label object and adds it to the 2, 1, 1, 1 positions with WEST a
```

```
      */
      gbc.gridx = 2;
      gbc.gridy = 1;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      label2 = new JLabel("Topic Details:");
      getContentPane().add(label2, gbc);
      /*
      Initializes the Help Topic List object and adds it to the Help Topic scroll pane object.
      Next, adds this scroll pane object to 1, 2, 1, 1 positions with WEST alignment.
      */
      gbc.gridx = 1;
      gbc.gridy = 2;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      String[] listString = {"Change the text color",
      "Cut, Copy and Paste the text",
      "Find a word in the file",
      "Insert date and time in the file",
      "Open a file",
      "Print a file",
      "Save a file",
      "Change the text font",
      "Using file locks",
      "Wrap the file text"};
      list = new JList(listString);
      /* Sets the height of the list box. */
      list.setFixedCellHeight(25);
      list.addListSelectionListener(this);
      listScroll = new JScrollPane(list);
      getContentPane().add(listScroll, gbc);
      /*
      Initializes the text area object and adds it to the text area scroll pane object.
      Next, adds this scroll pane objects to 2, 2, 1, 1 positions with WEST alignment.
      */
      gbc.gridx = 2;
      gbc.gridy = 2;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.WEST;
      area = new JTextArea(12, 60);
      /*
      Sets the editable property of the text area to false.
      */
      area.setEditable(false);
      /* Sets the Line and word wrap style to TRUE */
      area.setLineWrap(true);
      area.setWrapStyleWord(true);
      areaScroll = new JScrollPane(area);
      getContentPane().add(areaScroll, gbc);
      /*
      Initializes the OK button and adds this OK buttons to the panel. Next, sets the layout
      of the panel to FlowLayout. Now, adds this panel to the 1, 3, 2, 1 positions with CENTER align
      */
      gbc.gridx = 1;
      gbc.gridy = 3;
      gbc.gridwidth = 2;
      gbc.gridheight = 1;
      gbc.anchor = GridBagConstraints.CENTER;
      panel = new JPanel();
      panel.setLayout(new FlowLayout());
      ok = new JButton("OK");
      panel.add(ok);
      getContentPane().add(panel, gbc);
   }
   /*
   valueChanged() - This method is called when the user selects any help topic from the list
   Parameters: lse - Represents an object of the ListSelectionEvent class that contains the details
   Return Value: NA
   */
   public void valueChanged(ListSelectionEvent lse)
   {
      try
      {
         /*
         This section is executed, when end user selects the item from Help Topic list.
         */
         if(lse.getSource() == list)
         {
            /*
            When end user selects "Change the text color" from the List.
            */
            if(list.getSelectedIndex() == 0)
            {
               str = "The steps to change the font color are:\n\n" + " 1.
               Select Format->Color command from the menu bar.\n\n" + " 2.
```

```
   Scroll the Red slider, Green slider, and Blue sliders to specify the
   R, G, B values for the color.\n\n" + " 3. Click OK button.";
}
/*
When end user selects "Cut, Copy, and Paste the text" from the List.
*/
else if(list.getSelectedIndex() == 1)
{
   str = "The steps to cut, copy, and paste the text are:\n\n" + " 1.
   Select the Edit->Cut command from the menu bar to cut the selected text.
   \n\n" + " 2. Select the Edit->Copy command from the menu bar to copy
   the selected text.\n\n" + " 3. Select the Edit-Paste command from the menu
   bar to paste the cut or copied text to the specified location.";
}
/*
When end user selects "Find a word in the file" from the List.
*/
else if(list.getSelectedIndex() == 2)
{
   str = "The steps to find a specific character or word are:\n\n" + " 1.
   Select the Edit->Find command from the menu bar.\n\n" + " 2.
   Specify the character or word in the Type the Word to find text box.\n\n" +
   " 3. Click the OK button.\n\n" + "
   4. Select the Edit->Find Next command from the menu bar to find the next word.";
}
/*
When end user selects "Insert date and time in the file" from the List.
*/
else if(list.getSelectedIndex() == 3)
{
   str = "The step to insert date and time in the document is:\n\n" + " 1.
   Select Edit->Date/Time command from the menu bar to insert the
   current date and time in the file.";
}
/*
When end user selects "Open a file" from the List.
*/
else if(list.getSelectedIndex() == 4)
{
   str = "The steps to open a file are:\n\n" + " 1. Select File->
   Open command from the menu bar.\n\n" + " 2. Browse the File that
   you want to open.\n\n" + " 3. Select the appropriate file in
   File name text box.\n\n" + " 4. Click OK button.";
}
/*
When end user selects "Print a file" from the List.
*/
else if(list.getSelectedIndex() == 5)
{
   str = "The steps to print the document are:\n\n" + " 1. Select File->
   Print command from the menu bar.\n\n" + " 2. Specify the print range.\n\n" + "
   3. Specify the copies in Number of copies text box.\n\n" + " 4. Click OK button.";
}
/*
When end user selects "Save a file" from the List.
*/
else if(list.getSelectedIndex() == 6)
{
   str = "The steps to save a new document are:\n\n" + " 1. Select File->
   Save As command from the menu bar.\n\n" + " 2. Browse the location where
   you want to save the file.\n\n" + " 3. Specify the name of the file in
   File name text box.\n\n" + " 4. Click OK button.\n\n" + "
   5. Select File->Save command from the menu bar to save the existing file.";
}
/*
When end user selects "Change the text font" from the List.
*/
else if(list.getSelectedIndex() == 7)
{
   str = "The steps to change the font are:\n\n" + " 1. Select the Format->
   Font command from the menu bar.\n\n" + " 2. Select the font from the
   Fonts list box.\n\n" + " 3. Select the font size from the Size list box.\n\n" + "
   4. Select the font style from the Style list box.";
}
/*
When end user selects "Using file locks" from the List.
*/
else if(list.getSelectedIndex() == 8)
{
   str = "The steps to lock the files are:\n\n" + " 1. Select Lock->
   Exclusive command from the menu bar to lock the file with exclusive lock.\n\n" + "
   2. Select Lock->Share command from the menu bar to lock the file with share lock.";
}
/*
When end user selects "Wrap the file text" from the List.
*/
else if(list.getSelectedIndex() == 9)
{
```

```
                str = "The step to wrap a text in the window is:\n\n" + "
                1. Select Format->Word Wrap command from the menu bar.";
             }
             /* Sets the text to the text area. */
             area.setText(str);
             }
          }
          catch(Exception nfe)
          {
       }
    }
    /*
    getOk() method - This method is invoked when end user click the OK button of the Font dialog box.
    parameter - NA
    return value - ok
    */
    public JButton getOk()
    {
       return ok;
    }
}
```

Download this Listing.

In the above code, the Help() constructor creates the Help dialog box, as shown in Figure 6-6:

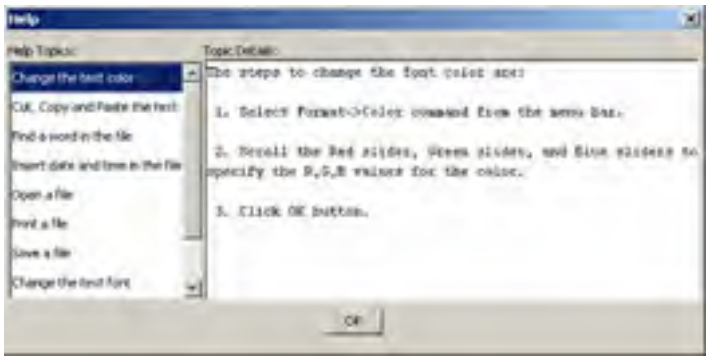


**Figure 6-6:** The Help Dialog Box

When an end user selects any help topic from the list box in the Help dialog box, the Help class invokes the valueChanged() method. This method acts as an event listener and displays the appropriate steps in the text area, based on the list item the end user selects.

When an end user clicks the OK button in the Help dialog box, the actionPerformed() method is invoked. This method sets the visibility of the Help dialog box to false and closes the dialog box.

# Unit Testing

To test the Text Editor:

1. 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;
   ```
2. 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;
   ```
3. Copy the Editor.java, ActionPerform.java, ColorClass.java, FontClass.java, PrintClass.java, and Help.java files to a folder on your computer. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Text Editor, specify the following command at the command prompt:
   ```
   java Editor
   ```
5. Select the File->New command from the menu bar. A new text file is opened in the Text Editor, as shown in Figure 6-7:

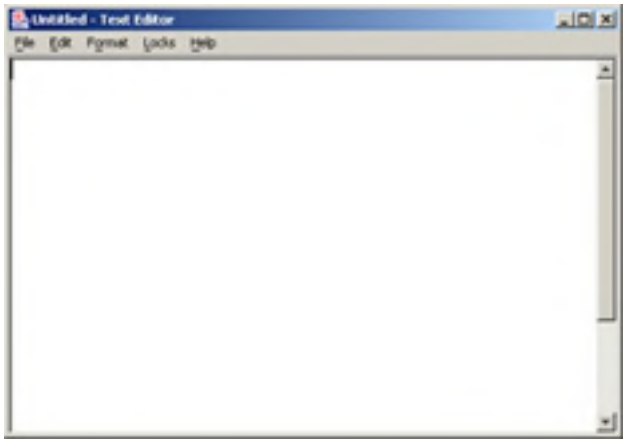


**Figure 6-7:** The Untitled Text Editor Window

6. Write Hello, this is a sample file, in the currently untitled text file.
7. Select the File->Save As command from the menu bar to save the file. A Save As dialog box appears, as shown in Figure 6-8:

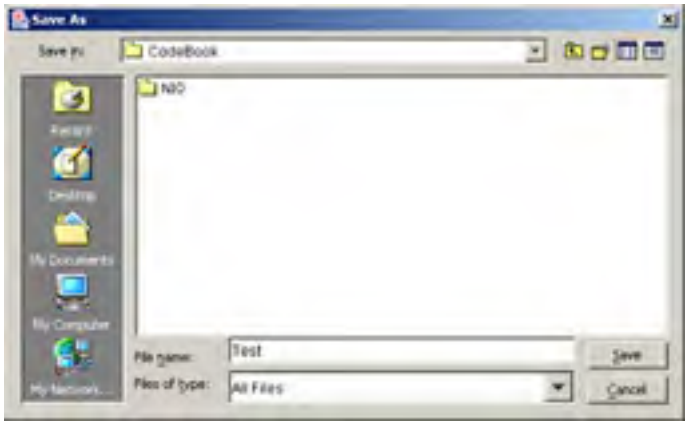


**Figure 6-8:** Save As Dialog Box

8. In the Save As dialog box, specify the file name as Test and location as C:\CodeBook.
9. Select the File->New command from the menu bar to clear the text area.
10. Select the File->Open command from the menu bar to open the file, Test.txt. The Open dialog box appears, as shown in Figure 6-9:


**Figure 6-9:** Open Dialog Box

11. Click the Open button of the Open dialog box that opens the Test.txt file from C:\CodeBook in the Text Editor, as shown in Figure 6-10:

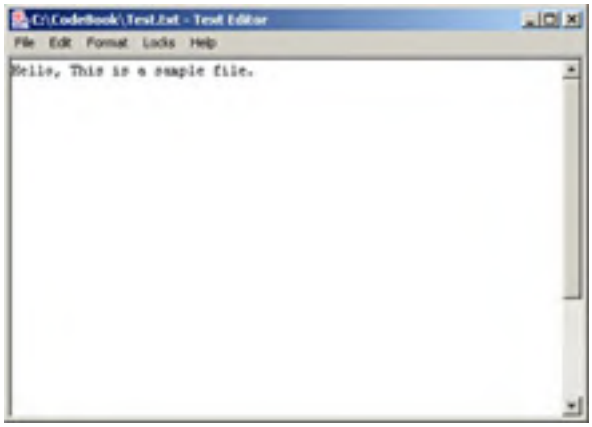


**Figure 6-10:** Displaying the Test.txt File

12. Select the Format->Font command from the menu bar to open the Font dialog box, as shown in Figure 6-4. Specify the font as Verdana, size as 20, and style as Italic. Click the OK button to apply the font changes to the file, as shown in Figure 6-11:


**Figure 6-11:** Displaying the Test.txt File with Selected Font

13. Select the Format->Color command from the menu bar to open the Color dialog box, as shown in Figure 6-10. Specify the RGB values in this dialog box and click OK. The color of the file is modified, as shown in Figure 6-12:

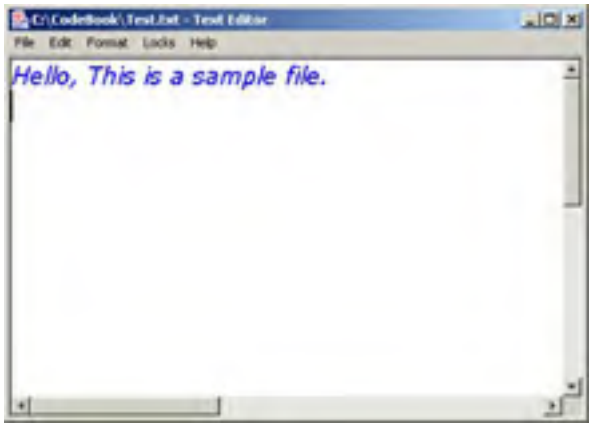


**Figure 6-12:** Displaying the Test.txt File with Selected Color

14. Select the Lock->Exclusive Lock command from the menu bar. This locks the Test.txt file.
15. Select the Start->Programs->Accessories->Notepad command to open a Windows notepad utility.
16. Select the File->Open command from the menu bar of the Notepad utility. Open the Text.txt file from the C:\CodeBook location.
17. Change the contents of the file and select the File->Save command from the menu bar. An error message appears because the file is locked, as shown in Figure 6-13:

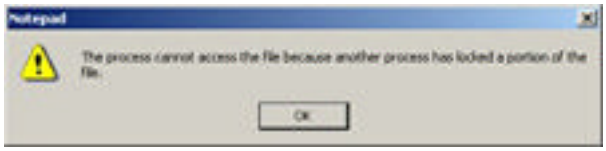


**Figure 6-13:** Displaying Error Message



◄ PREVIOUS NEXT ►

# Chapter 7: Creating a Network Information Application

The New Input/Output (NIO) API provides the java.nio, java.nio.channels, and java.nio.charset packages for buffer management, socket handling, and data transfer. The java.nio package contains the ByteBuffer and CharBuffer classes that you can use to read and store the bytes and characters. To connect the application to a network, you can use the SocketChannel class that is available in the java.nio.channel package. The java.nio.charset package provides two classes, Charset and CharsetDecoder. You can use these classes to set the character sets and decode the bytes to characters.

This chapter explains how to develop a Network Information application using the java.nio, java.nio.channels, and java.nio.charset packages.

## Architecture of the Network Information Application

The Network Information application displays a list of computers connected in a network. Using this application, you can display the echo time or the current system date and time of a specific computer. For example, ABC Inc has corporate offices in three countries, X, Y, and Z. A network administrator in ABC Inc's office in X wants to know the time displayed in the HR Manager's computer, which is located at Y. The Network Information application gives the network administrator the ability to retrieve this information. The Network Information application uses the following files:

- NetCompFrame.java: Creates a user interface for the Network Information application that helps display a list of network computers.
- NetCompConnect.java: Implements the functions of the Network Information application. This file connects the application to the network using socket channels.
- CompInfo.java: Provides information about the network computer.
- CompInfoDialog.java: Creates a dialog box that displays the echo time and current system date and time of a computer.

Figure 7-1 shows the architecture of the Network Information application:

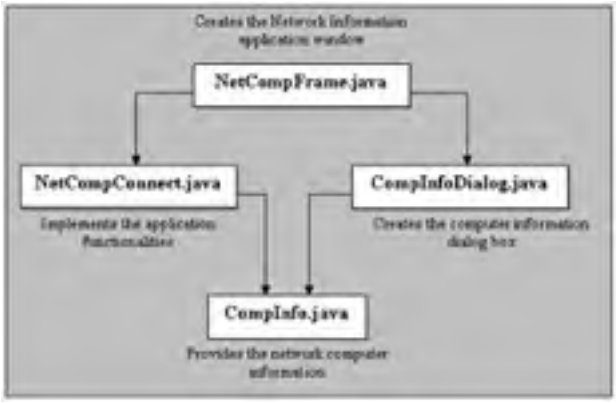


**Figure 7-1:** Architecture of the Network Information Application

In the Network Information application, the NetCompFrame.java file calls the NetCompConnect.java and CompInfoDialog.java files. The NetCompConnect.java file connects the application to the network at the specified port number and calls the CompInfo.java file to retrieve the computer information. The CompInfoDialog.java file displays the computer information by calling the CompInfo.java file.

# Creating the User Interface for the Network Information Application

The NetCompFrame.java file helps create a user interface that allows you to retrieve a list of network computers. You can use this list to retrieve the echo time or the current system date and time of a network computer.

Listing 7-1 shows the contents of the NetCompFrame.java file:

**Listing 7-1: The NetCompFrame.java File**

```
/*Imports javax.swing package classes. */
import javax.swing.*;
/*Imports javax.swing.event package classes. */
import javax.swing.event.*;
/*Imports java.awt package classes. */
import java.awt.*;
import java.awt.event.*;
/*Imports java.awt.event package classes. */
import java.net.*;
/*Imports java.util package classes. */
import java.util.*;
/*
Class NetCompFrame - This class creates the user interface of the Network Information
application. This user interface displays the list of all the computers connected to the network.
Fields:
   cont - Creates the container for the application.
   findComp_lbl - Creates the label of Service/Protocol.
   findComp_cmb - Creates the combo box to select the service or protocol.
   netMask_lbl - Creates the label of the Host IP Address.
   netMask_txt - Creates the text box where the end user can specify the IP address.
   netComp_lbl - Creates the label of the network computers.
   netComp_lst - Creates a list box that displays the computer name.
   netComp_sp - Creates the scrollpane for the list box.
   showComp_btn -Creates the Display List button.
   clear_btn - Creates the Clear button.
   close_btn - Creates the Close button.
   listDataVec - Stores the list of computers in vector form.
   compInfoVec - Stores the computer's information in the vector form.
   selectedItem - Contains the selected item from the list.
Methods:
   actionPerformed() - This method is invoked when end user clicks any of the button
   of the network management application.
   getMachinesList() - This method is invoked to display the list of computers
   connected in the network.
   valueChanged() - This method is invoked end user selects the computer from the computer list.
   itemStateChanged() - This method is invoked when end user selects the
   service/ protocol from the combo box.
   main() - This method creates the main window of the application.
   */
   public class NetCompFrame extends JFrame implements ActionListener, ListSelectionListener, ItemLi
   {
      /* Declares the object of the Container class. */
      Container cont = null;
      /* Declares the objects of the JLabel class. */
      JLabel findComp_lbl;
      JLabel netMask_lbl;
      JLabel netComp_lbl;
      /* Declares the objects of the JComboBox class. */
      JComboBox findComp_cmb;
      /* Declares the object of the JTextField class. */
      JTextField netMask_txt;
      /* Declares the object of the JList class. */
      JList netComp_lst;
      /* Declares the object of the JScrollPane class. */
      JScrollPane netComp_sp;
      /* Declares the objects of the JButton class. */
      JButton showComp_btn;
      JButton clear_btn;
      JButton close_btn;
      /*
      Declares and initializes the objects of the Vector class.
      */
      Vector listDataVec = new Vector();
      Vector compInfoVec = new Vector();
      /* Declares the object of String class. */
      String selectedItem = "";
      /*
      Declares the object of the NetCompConnect class.
      */
      NetCompConnect compConnect = null;
      /*
      Declares the object of the CompInfoDialog class.
      */
      CompInfoDialog dialogBox = null;
      /* Defines the default constructor. */
```

```
   public NetCompFrame(String title)
   {
      super(title);
      try
      {
         /*
         Initializes and sets the look and feel of the application to Windows look and feel.
         */
         UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
      }
      catch(Exception e){}
      /* Sets the size of the application. */
      setSize(350, 400);
      /*
      addWindowListener - It contains a windowClosing() method.
      windowClosing: It is called when the end user clicks the close button of the Window.
      It closes the main window.
      Parameter: we - Represents the object of the WindowEvent class.
      Return Value: NA
      */
      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent we)
         {
            System.exit(0);
         }
      });
   /* Initializes the object of the Container class. */
   cont = getContentPane();
   /* Sets the layout of the container to NULL layout. */
   cont.setLayout(null);
   /* Initializes the objects of the JLabel class. */
   findComp_lbl = new JLabel("Service/Protocol: ");
   netMask_lbl = new JLabel("Host IP Address: ");
   netComp_lbl = new JLabel("Network Computers: ");
   /* Initializes the object of the JTextField class. */
   netMask_txt = new JTextField();
   netMask_txt.setFont(new Font("Verdana", Font.PLAIN, 12));
   /* Initializes the object of the JComboBox class. */
   findComp_cmb = new JComboBox();
   findComp_cmb.setFont(new Font("Verdana", Font.PLAIN, 12));
   /* Adds the items in the combo box. */
   findComp_cmb.addItem("Echo");
   findComp_cmb.addItem("Date Time");
   /* Adds the itemListener event to the combo box. */
   findComp_cmb.addItemListener(this);
   /* Initializes the object of the JList class. */
   netComp_lst = new JList();
   netComp_lst.setFont(new Font("Verdana", Font.PLAIN, 12));
   netComp_lst.addListSelectionListener(this);
   /* Initializes the object of the JScrollPane class. */
   netComp_sp = new JScrollPane(netComp_lst);
   /* Initializes the objects of the JButton class. */
   showComp_btn = new JButton("Display List");
   clear_btn = new JButton("Clear");
   close_btn = new JButton("Close");
   /* Adds actionListener events to the JButton objects. */
   showComp_btn.addActionListener(this);
   clear_btn.addActionListener(this);
   close_btn.addActionListener(this);
   /* Sets the position of the components on the application window. */
   findComp_lbl.setBounds(10, 10, 140, 20);
   findComp_cmb.setBounds(160, 10, 100, 20);
   netMask_lbl.setBounds(10, 40, 110, 20);
   netMask_txt.setBounds(160, 40, 170, 20);
   showComp_btn.setBounds(230, 70, 100, 25);
   netComp_lbl.setBounds(10, 100, 300, 20);
   netComp_sp.setBounds(10, 125, 320, 200);
   clear_btn.setBounds(120, 335, 100, 25);
   close_btn.setBounds(230, 335, 100, 25);
   /* Adds the swing components on the container. */
   cont.add(findComp_lbl);
   cont.add(findComp_cmb);
   cont.add(netMask_lbl);
   cont.add(netMask_txt);
   cont.add(netComp_lbl);
   cont.add(netComp_sp);
   cont.add(showComp_btn);
   cont.add(clear_btn);
   cont.add(close_btn);
   /* Sets the visibility of the frame to true. */
   setVisible(true);
   /* Sets the reliability of the frame to false. */
   setResizable(false);
}
/*
actionPerformed() - This method is called when the end user clicks any of the button from the win
Parameters: ae - Represents an object of the ActionEvent class that contains the details of the e
```

```
Return Value: NA
*/
public void actionPerformed(ActionEvent ae)
{
   /* This section is executed when an end user clicks the Display List button. */
   if (ae.getSource() == showComp_btn)
   {
      /* Gets the computer name from the selected item. */
      String cmb_selection = (String)findComp_cmb.getSelectedItem();
      /*
      This section is executed when the end user selects the "Echo" option from the combo box.
      */
      if (cmb_selection.trim().equals("Echo"))
      {
         /*
         Gets the machine IP address from the Host IP Address text field.
         */
         String machineIPAddress = netMask_txt.getText();
         if (machineIPAddress==null || machineIPAddress.trim().equals(""))
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please Enter the IP address of your machine.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         try
         {
            /*
            Creates and initializes the object of the InetAddress class.
            */
            InetAddress myMachineAddress = InetAddress.getByName(machineIPAddress.trim());
         }
         catch (Exception ex)
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please enter a valid IP address.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         /* Calls the getMachinesList() method. */
         getMachinesList(machineIPAddress.trim(), 7);
      }
      /*
      This section is executed when end user selects the "Date Time" option from the combo box.
      */
      else if (cmb_selection.trim().equals("Date Time"))
      {
         /*
         Gets the machine IP address from the Host IP Address text field.
         */
         String machineIPAddress = netMask_txt.getText();
         if (machineIPAddress==null || machineIPAddress.trim().equals(""))
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please Enter the IP address of your machine.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         try
         {
            /*
            Creates and initializes the object of the InetAddress class.
            */
            InetAddress myMachineAddress = InetAddress.getByName(machineIPAddress.trim());
         }
         catch (Exception ex)
         {
            /* Displays an Error message dialog box. */
            JOptionPane.showMessageDialog(this, "Please enter a valid IP address.");
            /* Clears the text of the Host IP Address text field. */
            netMask_txt.setText("");
            return;
         }
         /* Calls the getMachinesList() method. */
         getMachinesList(machineIPAddress.trim(), 13);
      }
   }
   /* This section is executed when an end user clicks the Clear button. */
   else if (ae.getSource() == clear_btn)
   {
      /* Calls the setListData() method of the JList class. */
      netComp_lst.setListData(listDataVec=new Vector());
   }
   /* This section is executed when an end user clicks the Close button. */
   else if (ae.getSource() == close_btn)
```

```
   {
      System.exit(0);
   }
}
/*
getMachinesList() - This method is invoked when the end user clicks the
Display List button of the Network Management Application window.
Parameters:
myAddress -   Contains the IP address of the user machine.
port - Contains the port number of the user machine.
Return Value: NA
*/
public void getMachinesList(String myAddress, int port)
{
   /* Initializes the object of the Vector class. */
   compInfoVec = new Vector();
   /* Calls the setListData() method of the JList class. */
   netComp_lst.setListData(listDataVec=new Vector());
   /* Creates and initializes the object of the StringTokenizer class. */
   StringTokenizer st = new StringTokenizer(myAddress, ".");
   /* Gets the IP address from the tokenize string. */
   String rawIPAddress = (String)st.nextToken() + "." + (String)st.nextToken() + "." +
   (String)st.nextToken() + ".";
   /* Initializes an object of the String class to store the IP address. */
   String testIPAddress = null;
   for (int i=0; i<256; i++)
   {
      /*
      Gets the IP address of the computers that are connected to the network.
      */
      testIPAddress = rawIPAddress + i;
      /* Creates and initializes the object of the NetCompConnect class. */
      compConnect = new NetCompConnect(testIPAddress, this, port);
      /* Calls the start() method of the NetCompConnect class. */
      compConnect.start();
   }
   try
   {
      Thread.sleep(5000);
   }
   catch (InterruptedException iex){}
   /* Evaluates the size of the vector. */
   int size = compInfoVec.size();
   for (int j=0; j<size; j++)
   {
      /* Creates and initializes the object of the CompInfo class. */
      CompInfo tempObj = (CompInfo)compInfoVec.elementAt(j);
      /* Inserts the element at the end of the vector. */
      listDataVec.addElement(tempObj.getAddress());
   }
   /* Calls the setListData() method of the JList class. */
   netComp_lst.setListData(listDataVec);
}
/*
valueChanged() - This method is called when the end user selects the
computer name from the list.
Parameters: e - Represents an object of the ListSelectionEvent class
that contains the details of the event.
Return Value: NA
*/
public void valueChanged(ListSelectionEvent e)
{
   /*
   This section is executed when an end user changes the selection value of the computer from the
   */
   if (e.getSource() == netComp_lst)
   {
      /* Gets the computer name from the selected item. */
      String selectedItem2 = (String)netComp_lst.getSelectedValue();
      if (selectedItem2 == null)
      {
         return;
      }
      if (selectedItem.equals(selectedItem2))
      {
         /* Clears the item selection in the list. */
         netComp_lst.clearSelection();
      }
      /*
      This section is executed when the previously selected and currently
      selected computer names are different.
      */
      if (!selectedItem.equals(selectedItem2))
      {
         selectedItem = selectedItem2;
         /* Evaluates the size of the vector. */
         int size = compInfoVec.size();
         for (int j=0; j<size; j++)
```

```
                {
                   /* Creates and initializes the object of the CompInfo class. */
                   CompInfo tempObj = (CompInfo)compInfoVec.elementAt(j);
                   /* Gets the IP address using the getAddress() method of the CompInfo class. */
                   String compAddress = tempObj.getAddress();
                   if (compAddress.equals(selectedItem))
                   {
                      /*
                      Gets the computer name from the selected item.
                      */
                      String cmb_selection = (String)findComp_cmb.getSelectedItem();
                      /*
                      Gets the computer information of the selected item using the getCompInfo()
                      method of the CompInfo class. */
                      String compInfo = tempObj.getCompInfo();
                      /*
                      This section is executed when the end user selects "Echo" from the combo box.
                      */
                      if (cmb_selection.trim().equals("Echo"))
                      {
                         /* Creates and initializes the object of the CompInfoDialog class. */
                         dialogBox = new CompInfoDialog(this, "Echo Response", true, 7, compInfo);
                         /* Clears the item selection in the list. */
                         netComp_lst.clearSelection();
                      }
                      /*
                      This section is executed when the end user selects "Date Time" from the combo box.
                      */
                      else if (cmb_selection.trim().equals("Date Time"))
                      {
                         /*
                         Creates and initializes the object of the CompInfoDialog class.
                         */
                         dialogBox = new CompInfoDialog(this, "System Date & Time", true, 13, compInfo);
                         /* Clears the item selection in the list. */
                         netComp_lst.clearSelection();
                      }
                      break;
                   }
                }
             }
          }
       }
       /*
       itemStateChanged - This method is called when the end user selects the
       Service/Protocol from the combo box.
       Parameters: ie - a ListSelectionEvent object containing details of the event.
       Return Value: NA
       */
       public void itemStateChanged(ItemEvent ie)
       {
          /* This section is executed when the end user selects any value from the combo box. */
          if (ie.getSource() == findComp_cmb)
          {
             /* Calls the setListData() method of the JList class. */
             netComp_lst.setListData(listDataVec=new Vector());
          }
       }
       /*
       main() - This method creates the main window of the user interface and displays it.
       Parameters:
       args[] - Contains any command line arguments passed.
       Return Value: NA
       */
       public static void main(String[] args)
       {
          /* Creates and initializes the object of the NetCompFrame class. */
          NetCompFrame frame = new NetCompFrame("Network Information Application");
       }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the NetCompFrame class. This class generates the main window of the Network Information application, as shown in Figure 7-2:

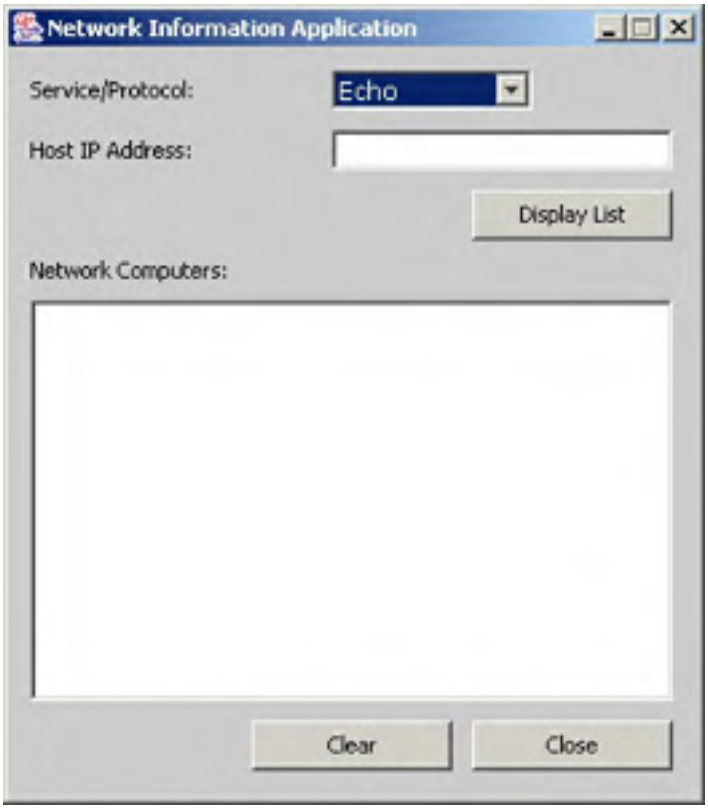


**Figure 7-2:** The Network Information Application User Interface

When the end user specifies the required service or protocol in the combo box and the host IP address in the Host IP Address text field, a list of network computers is displayed in the Network Computers list box.

The methods defined in the above code are:

- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method based on the button the end user clicks. If the end user clicks the Display List button, the actionPerformed() method retrieves the computer name from the selected item and checks the combo box value. If the end user selects Echo in the combo box, the actionPerformed() method retrieves the host IP from the Host IP Address text field and calls the getMachinesList() method with port number 7. If the end user selects the Date/Time value, the actionPerformed() method retrieves the host IP from the Host IP Address text field and calls the getMachinesList() method with port number 13. When an end user clicks the Clear button, the actionPerformed() method calls the setListData() method of the JList class. The setListData() method clears the list box. When an end user clicks the Close button, the actionPerformed() method calls the System.exit(0) method to close the application.
- `getMachinesList()`: Retrieves the IP address of the network computers and initializes the object of the NetCompConnect class. This method then calls the start() method of the NetCompConnect class and initializes the object of the CompInfo class. Next, the getMachinesList() method inserts the element at the end of the vector and calls the setListData() method of the JList class to add the list in the list box.
- `valueChanged()`: Acts as an event listener and activates an appropriate class or method based on the list value the end user selects. When the end user changes the selection of the item in the list, this method checks the selected item with the previously selected item. If both the item names are same, this method calls a break; otherwise it creates and initializes the object of the CompInfo class. This method then gets the IP address using the getAddress() method of the CompInfo class and gets the computer information of the selected item using the getCompInfo() method of the CompInfo class. Next, the valueChanged() method checks the combo box value and creates the object of the CompInfoDialog class based to the value specified in the combo box.
- `itemStateChanged()`: Acts as an event listener and activates an appropriate class or method based on the option the end user changes in the combo box. After the end user modifies the option in the combo box, this method calls the setListData() method of the JList class. The setListData() method clears the list box.


PREVIOUS NEXT

# Implementing the Network Functionality

The NetCompConnect.java file implements the functions of the Network Information application. This file reads the hosts IP address and port number on which the end user wants to create the socket channel. The NetCompConnect.java file then retrieves information, such as echo time and current system date and time of the computer the end user selects. The NetCompConnect.java file passes this information to the CompInfo.java file, which sends it to the CompInfoDialog.java file. The CompInfoDialog.java file displays the computer information in a dialog box.

Listing 7-2 shows the contents of the NetCompConnect.java file:

**Listing 7-2: The NetCompConnect.java File**

```
/* Imports java.net package classes. */
import java.net.*;
/* Imports java.io package classes. */
import java.io.*;
/* Imports java.nio package classes.*/
import java.nio.*;
import java.nio.channels.*;
import java.nio.charset.*;
import java.util.*;

/*
Class NetCompConnect - This class connects the application to the network using socket channel.
This class also performs the ping and date/time operations.
Fields:
   strCompAddr - Contains the IP address.
   mainApp - Creates the object of the NetCompFrame class.
   port - Contains the port number.
   charset - Contains the charset.
   decoder - Creates the decoder.
   buff - Creates the buffer.
   isa - Contains the object of the InetSocketAddress class.
   sc - Create the socket channel.
Methods:
   connectComp() - This method creates the socket channel at the specified port number.
   run() - This method is invoked when the thread is started.
   */
public class NetCompConnect extends Thread
{
   /* Declares the object of the String class. */
   String strCompAddr = null;
   /* Declares the object of the NetCompFrame class. */
   NetCompFrame mainApp = null;
   /* Declares the object of the Charset class. */
   Charset charset;
   /* Declares the object of the CharsetDecoder class. */
   CharsetDecoder decoder;
   /* Declares the object of the ByteBuffer class. */
   ByteBuffer buff;
   /* Declares the object of the InetSocketAddress class. */
   InetSocketAddress isa;
   /* Declares the object of the SocketChannel class. */
   SocketChannel sc = null;
   int port = 0;
   /* Defines the default constructor. */
   public NetCompConnect(String strCompAddr, NetCompFrame mainApp, int port)
   {
      this.strCompAddr = strCompAddr;
      this.mainApp = mainApp;
      this.port = port;
   }
   /*
      connectComp() - This method creates and initializes the socket at port 7 or 13 to
      retrieve the echo time or date/time of the specific computer.
      Parameter:
      strCompAddr - Stores the computer IP address.
      port - Stores the port number.
      Return Value: NA
   */
   public int connectComp(String strCompAddr, int port)
   {
      String compInfo = "";
      String temp = "";
      String time = "";
      String am_pm = "";
      String date = "";
      String date_time = "";
      /* Gets the current time of the system in milliseconds. */
      long startTime = System.currentTimeMillis();
      try
      {
         /*
         Checks the port number, if port number is 13, this code is executed.
```

```
*/
if (port == 13)
{
   /* Initializes the object of the Charset class. */
   charset = Charset.forName("US-ASCII");
   /* Initializes the object of the CharsetDecoder class. */
   decoder = charset.newDecoder();
   /* Initializes and allocates the size of the buffer. */
   buff = ByteBuffer.allocateDirect(1024);
   /* Initializes the object of the InetSocketAddress class at port 13. */
   isa = new InetSocketAddress(strCompAddr, port);
   try
   {
      /* Opens the socket channel. */
      sc = SocketChannel.open();
      /* Connects the socket channel to the specified IP address. */
      sc.connect(isa);
      /* Clears the buffer. */
      buff.clear();
      /* Reads the data from the buffer. */
      sc.read(buff);
      /* Flips the buffer. */
      buff.flip();
      /*
      Creates and initializes the object of CharBuffer and stores the decoded data from the
      */
      CharBuffer cb = decoder.decode(buff);
      temp = new String(cb.array());
      StringTokenizer st = new StringTokenizer(temp, " ");
      time = (String)st.nextToken();
      am_pm = (String)st.nextToken();
      date = (String)st.nextToken();
      date_time = date + " " + time + " " + am_pm;
      compInfo = compInfo + date_time;
   }
   finally
   {
      if (sc != null)
      /* Close the socket channel. */
      sc.close();
   }
   /*
   Creates the object of CompInfo class and Adds this object to the
   compInfoVec vector of the NetCompFrame class.
   */
   mainApp.compInfoVec.addElement(new CompInfo(strCompAddr, compInfo));
   return 1;
}
/* Checks the port number, if port number is 7, this code is executed. */
else
{
   /* Initializes the object of the InetSocketAddress class at port 7. */
   isa = new InetSocketAddress(strCompAddr, port);
   try
   {
      /* Opens the socket channel. */
      sc = SocketChannel.open();
      /* Connects the socket channel to the specified IP address. */
      sc.connect(isa);
      /* Gets the socket from the socket channel. */
      Socket t = sc.socket();
      /*
      Creates and initializes the object of DataInputStream class to retrieves
      the data from the socket.
      */
      DataInputStream is = new DataInputStream(t.getInputStream());
      /*
      Creates and initializes the object of PrintStream class to put the data to the socket
      */
      PrintStream ps = new PrintStream(t.getOutputStream());
      ps.println("ping");
      String str = is.readLine();
      if (str.equals("ping"))
      {
         /* Gets the current system time. */
         long endTime = System.currentTimeMillis();
         /* Evaluates the echo time. */
         compInfo = (endTime-startTime) + " ms";
         /*
         Creates the object of CompInfo class and Adds this object to the
         compInfoVec vector of the NetCompFrame class.
         */
         mainApp.compInfoVec.addElement(new CompInfo(strCompAddr, compInfo));
         return 1;
      }
      else
      {
         return 0;
```

```
                }
            }
            finally
            {
                if (sc != null)
                /* Close the socket channel. */
                sc.close();
            }
        }
    }
    catch (Exception e)
    {
        return 0;
    }
}
/*
run() - This method is executed when thread is started.
Parameter: NA
Return Value: NA
*/
public void run()
{
    /* Calls the connectComp() method */
    connectComp(this.strCompAddr, this.port);
}
}
```

Download this Listing.

In the above code, the constructor of the NetCompConnect class reads the host IP address, port number, and object of the NetCompFrame class. The methods defined in the above code are:

- `run()`: Calls the connectComp() method to connect the application to the network.
- `connectComp()`: Checks the port number to establish a connection. If the port number is 13, this method initializes the objects of the Charset and CharsetDecoder classes. This method then initializes and allocates the size of the buffer, initializes the object of the InetSocketAddress class at port 13, and opens a socket channel on port 13. Next, the connectComp() method connects the socket channel to the specified host address, reads the bytes into buffer, and decodes the bytes from buffer to character. This method then creates the object of the CompInfo class and adds this object to the compInfoVec vector of the NetCompFrame class. If the port number is 7, the connectComp() method initializes the object of the InetSocketAddress class at port 7 and opens the socket channel on that port. The connectComp() method then connects the socket channel to the specified IP address, gets the socket from the socket channel, and evaluates the echo time in milliseconds. Next, the connectComp() method creates the object of the CompInfo class and adds this object to the compInfoVec vector of the NetCompFrame class.

# Creating the Computer Information File

The CompInfo.java file is used to provide the computer information, such as echo time and current system date and time of a specific computer.

Listing 7-3 shows the contents of the CompInfo.java file:

**Listing 7-3: The CompInfo.java File**

```
/*
Class CompInfo - This class provides the computer information of specific computer.
Fields:
   address - Contains the IP address of the computer.
   info - Contains the computer information's, such as echo time or date/time.
Methods:
   getAddress() - This method returns the IP address of the computer.
   getCompInfo() - This method returns the computer information.
*/
public class CompInfo
{
   /* Declares and initializes the objects of String class. */
   private String address = null;
   private String info = null;
   /* Defines the default constructor. */
   public CompInfo(String address, String info)
   {
      this.address = address;
      this.info = info;
   }
   /*
   getAddress() - This method is used to retrieve the IP address of a specified computer.
   Parameter: NA
   Return Value: String
   */
   public String getAddress()
   {
      return this.address;
   }
   /*
   getCompInfo() - This method is used to retrieve the computer information of a specified computer.
   Parameter: NA
   Return Value: String
   */
   public String getCompInfo()
   {
      return this.info;
   }
}
```

Download this Listing.

In the above code, the constructor of the CompInfo class takes the host IP and computer information as input parameters. The methods defined in the above code are:

- `getAddress()`: Returns the IP address of the selected computer.
- `getCompInfo()`: Returns computer information, such as echo time and current system date and time of the selected network computer.

# Creating the Computer Information Dialog Box

The CompInfoDialog.java file creates a dialog box that displays the information of the network computer.

Listing 7-4 shows the contents of the CompInfoDialog.java file:

**Listing 7-4: The CompInfoDialog.java File**

```
/* Import java packages */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/*
Class CompInfoDialog - This class creates a computer information dialog box.
Fields:
   cont - Creates the container.
   key_lbl - Creates the label for "Echo Time" or "Date/Time".
   value_lbl - Creates the label to display the result.
Methods:
   actionPerformed() - This method is invoked when end user clicks the OK button.
*/
public class CompInfoDialog extends JDialog implements ActionListener
{
   /* Declares the object of the Container class. */
   Container cont = null;
   /* Declares the object of the JLabel class. */
   JLabel key_lbl;
   JLabel value_lbl;
   /* Declares the object of the JButton class. */
   JButton ok_btn;
   /* Defines the default constructor.   */
   public CompInfoDialog(NetCompFrame parent, String title, boolean modal, int port, String value)
   {
      super(parent, title, modal);
      /* Set the size of the Computer Information dialog box. */
      setSize(200, 140);
      /* Creates the object of the Point class to get the parent frame location. */
      Point p = parent.getLocation();
      /* Set the location of the dialog box. */
      setLocation((int)p.getX()+10, (int)p.getY()+10);
      /*
      addWindowListener - It contains the windowClosing() method.
      windowClosing: It is called when the user clicks the cancel button of the Window.
      It closes the main window.
      Parameter: we- Object of WindowEvent class.
      Return Value: NA
      */
      addWindowListener(new WindowAdapter()
      {
         public void windowClosing(WindowEvent we)
         {
            dispose();
         }
      });
      cont = this.getContentPane();
      /* Sets the layout of the application to null. */
      cont.setLayout(null);
      /* This section is executed, if port number equals to 13. */
      if (port == 13)
      {
         /* Initialize the label. */
         key_lbl = new JLabel("Date and Time", SwingConstants.CENTER);
      }
      /* This section is executed, if port number equals to 7. */
      else
      {
         /* Initialize the label. */
         key_lbl = new JLabel("Echo Time", SwingConstants.CENTER);
      }
      /* Initializes the object of the JLabel class */
      value_lbl = new JLabel(value, SwingConstants.CENTER);
      /* Initializes the object of the JButton class */
      ok_btn = new JButton("Ok");
      /* Adds the actionListener event to the ok button. */
      ok_btn.addActionListener(this);
      /* Sets the position of the swing components on the container. */
      key_lbl.setBounds(0, 10, 200, 20);
      value_lbl.setBounds(0, 40, 200, 20);
      ok_btn.setBounds(70, 70, 60, 20);
      /* Adds the components to the container. */
      cont.add(key_lbl);
      cont.add(value_lbl);
      cont.add(ok_btn);
      show();
```

```
    }
    /*
    actionPerformed() - This method is called when the user selects any menu item from the menu bar.
    Parameters:   ae - an ActionEvent object containing details of the event.
    Return Value: NA
    */
    public void actionPerformed(ActionEvent ae)
    {
       if (ae.getSource() == ok_btn)
       {
          dispose();
       }
    }
}
```

Download this Listing.

In the above code, the constructor of the CompInfoDialog class reads the object of the NetCompFrame class, title, model, port number, and value. Here, title represents a string that sets the title, model represents the Boolean value, and value represents the string that contains the network information.

When an end user clicks the OK button in the Computer Information dialog box, the actionPerformed() method is invoked. This method calls the dispose() method to close the dialog box.

# Unit Testing

To test the Network Information application:

1. 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;
   ```
2. 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;
   ```
3. Copy the NetCompFrame.java, NetCompConnect.java, CompInfo, and CompInfoDialog.java files to a folder on your computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Network Information application, specify the following command at the command prompt:
   ```
   java NetCompFrame
   ```
   The above command opens the main window of the Network Information application, as shown in Figure 7-2.
5. Select the Echo option from the Service/Protocol combo box and specify the host IP address, such as 192.168.0.36. If you do not specify the IP address, an Error dialog box appears, as shown in Figure 7-3:


**Figure 7-3:** Error Dialog Box

6. If you specify a wrong host IP address, another Error dialog box appears, as shown in Figure 7-4:


**Figure 7-4:** Error Dialog Box with Error Message

7. Click the Display List button of the Network Information application. A list of network computers appears in the Network Computer list box, as shown in Figure 7-5:

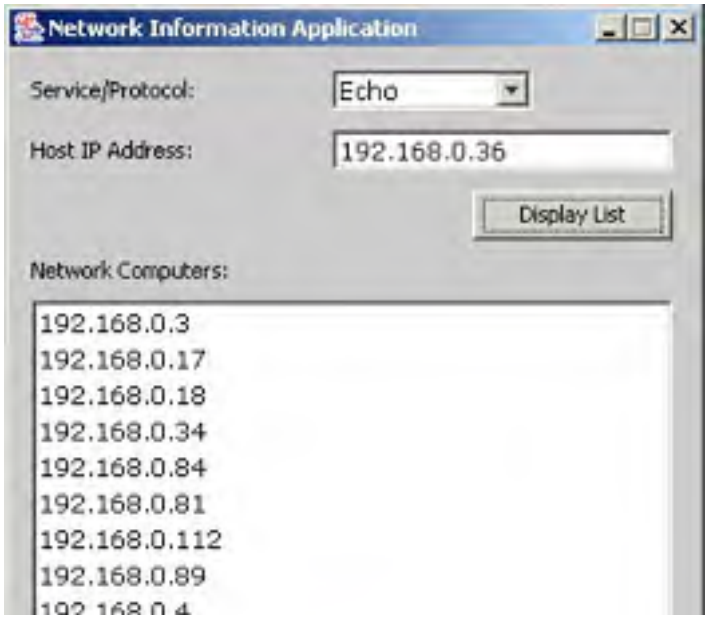


**Figure 7-5:** Network Information Application Window with Ehco Service

8. Select an IP address, such as 192.168.0.89, from the Network Computer list box. A Echo Response dialog box opens, as shown in Figure 7-6:

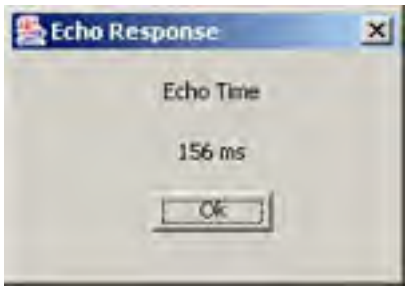


**Figure 7-6:** Echo Response Dialog Box

9. Select the Date/Time option from the Service/Protocol combo box and specify the host IP address, such as 192.168.0.36.
10. Click the Display List button of the Network Information application. A list of network computers appears in the Network Computer list box, as shown in Figure 7-7:

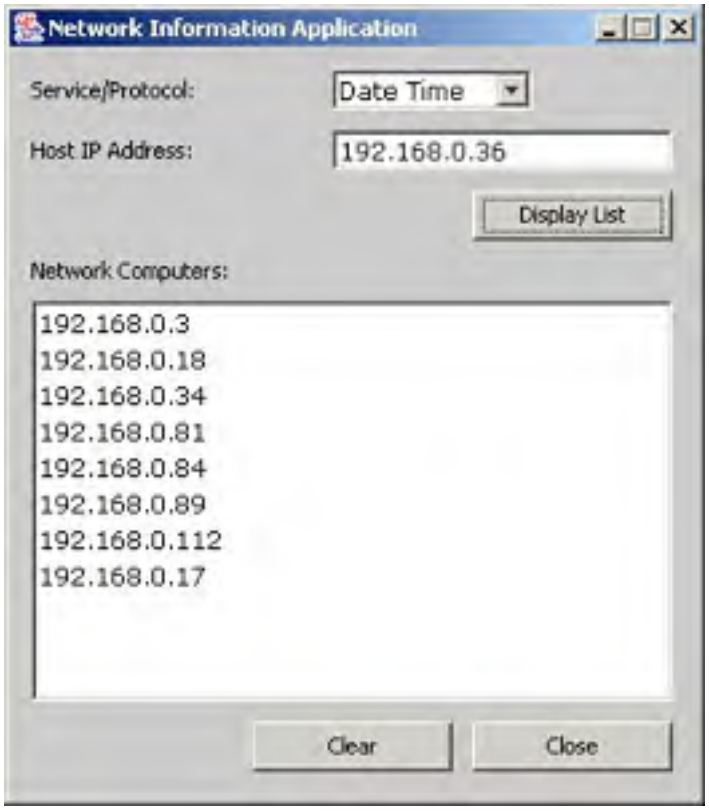


**Figure 7-7:** Network Information Application Window with Date Time Service

11. Select an IP address, such as 192.168.0.89, from the Network Computer list box. A System Date & Time dialog box opens, as shown in Figure 7-8:


**Figure 7-8:** System Date and Time Dialog Box



PREVIOUS NEXT

# Chapter 8: Creating an Encoder/Decoder Application

The New Input/Output (NIO) API provides the java.nio, java.nio.channels, and java.nio.charset packages for buffer management, socket handling, and text encoding and decoding. The java.nio package contains the ByteBuffer and CharBuffer classes that allow you to read and store the bytes and characters. To read a file, you can use the FileChannel class available in the java.nio.channel package. The java.nio.charset package provides the Charset and CharsetEncoder classes that help set the character sets. These classes encode the text using a specific encoding scheme. To decode an encoded text, you can use the CharsetDecoder class the java.nio.charset package.

An encoding scheme is a standard by which you map the coded character set to octets of eight bit sequence. The coded character set is the assignment of numeric values to a set of characters. The encoding scheme defines how a sequence of character encoding is represented as a sequence of bytes. The numeric value of the character set does not match with the encoded bytes. There are several types of encoding schemes, such as UTF-8, UTF-16, ISO-8859-1, UTF-16BE, or UTF-16LE. For example, UTF-8 encoding scheme encodes the coded character code of value less than 0x80 to a sequence of eight bytes.

This chapter explains how to develop an Encoder/Decoder application using the java.nio, java.nio.channels, and java.nio.charset packages.

## Architecture of the Encoder/Decoder Application

The Encoder/Decoder application allows an end user to encode the text using a specific encoding scheme or decode the encoded text into a readable form.

The Encoder/Decoder application uses the following files:

- `EncoderDecoder.java`: Creates a user interface that an end user can use to encode or decode a specified text.
- `EncodingSchemes.java`: Creates a dialog box to select the encoding scheme.

Figure 8-1 shows the architecture of the Encoder/Decoder application:

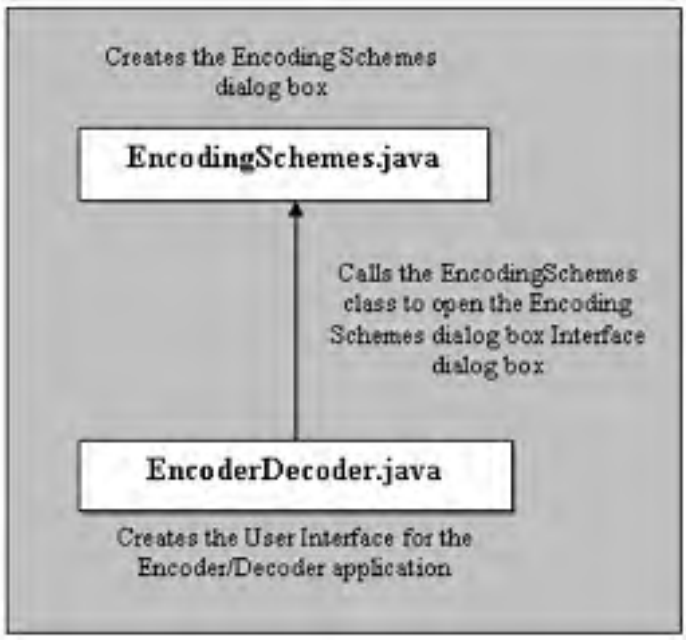


**Figure 8-1:** Architecture of the Encoder/Decoder Application

The EncoderDecoder.java file encodes and decodes a specified text. To select an encoding scheme, the EncoderDecoder.java file calls the EncodingSchemes.java file. The EncodingSchemes.java file returns the encoding scheme name to the EncoderDecoder.java file.

# Creating the User Interface for the Encoder/Decoder Application

The EncoderDecoder.java file helps create a user interface with a set of labels, text box, text areas, and buttons for the Encoder/Decoder application.

Listing 8-1 shows the contents of the EncoderDecoder.java file:

**Listing 8-1: The EncoderDecoder.java File**

```
/* Imports java.nio package classes. */
import java.nio.*;
import java.nio.charset.*;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.*;
import java.nio.channels.FileChannel;
import java.nio.charset.UnsupportedCharsetException;
/* Imports java.io package classes. */
import java.io.*;
import java.io.IOException;
/* Imports javax.swing package classes. */
import javax.swing.*;
import javax.swing.JOptionPane;
/* Imports java.awt package classes. */
import java.awt.*;
/* Imports java.awt.event package classes. */
import java.awt.event.*;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
/*
class EncoderDecoder - This class creates a GUI for the Encoder/Decoder application.
This class also provides the methods to encode the text and decode
the encoded text into a readable format.
Fields:
pan1 - Creates a panel that can contain a text box and two buttons.
titleLabel - Creates an Encoder/Decoder Application label.
selectLabel - Creates a select file label.
encodeLabel - Creates an encoded text label.
decodeLabel - Creates a decoded text label.
selectText - Creates a select file text field.
encodeArea - Creates an Encoded Text text area.
decodeArea - Creates a Decoded Text text area.
encodePane - Creates a scroll pane for the Encoded Text text area.
decodePane - Creates a scroll pane for the Decoded Text text area.
browseButton - Creates a Browse button.
encodeButton - Creates an Encode button.
selectButton - Creates a Select Encoding Scheme button.
resetButton - Creates a Reset button.
decodeButton - Creates a Decode button.
closeButton - Creates a Close button.
file_text - Represents a string that stores the contents of a file.
file - Represents a file.
jfc - Creates a file chooser dialog box.
fin - Creates a file input stream.
fchan - Creates a file channel.
buff_in - Stores the content of the opened files.
buff_out - Stores the content of the encoded file that needs to be further
encrypted in the hexadecimal format.
bbuf - Stores the encoded bytes.
cbuf - Stores the decoded bytes.
fsize - Contains the size of the file.
decoder - Creates a decoder that decodes the encoded text.
encoder - Creates an encoder that encodes the text.
charset - Creates a charset object.
encScheme - Represents an object of the EncodingSchemes class.
Methods:
keyTyped() - This method is invoked when an end user types any text in the Text
to Encode text area.
actionPerformed() - This method is invoked when an end user clicks the any
button of the Encoder/Decoder application.
open() - This method is invoked to open an existing text file that the end
user wants to encode.
encode() - This method is invoked to encode the text.
decode() - This method is invoked to decode the encoded text.
main() - This method creates the main window of the application and displays it.
*/
public class EncoderDecoder extends JDialog implements ActionListener, KeyListener
{
/* Declares the object of the JPanel class. */
JPanel pan1;
/* Declares the objects of the JLabel class. */
JLabel titleLabel;
JLabel textLabel;
```

```
JLabel selectLabel;
JLabel encodeLabel;
JLabel decodeLabel;
/* Declares the object of the JTextField class. */
JTextField selectText;
JTextField schemeText;
/* Declares the objects of the JTextArea class. */
JTextArea encodeArea;
JTextArea decodeArea;
JTextArea textArea;
/* Declares the objects of the JScrollPane class. */
JScrollPane encodePane;
JScrollPane decodePane;
JScrollPane textPane;
/* Declares the objects of the JButton class. */
JButton browseButton;
JButton encodeButton;
JButton selectButton;
JButton resetButton;
JButton decodeButton;
JButton closeButton;
/* Declares the object of the GridBagLayout class. */
GridBagLayout gbl;
/* Declares the object of the GridBagConstraints class. */
GridBagConstraints gbc;
/* Declares the objects of the String class. */
String str;
String file_text;
/* Declares the object of the File class. */
File file;
/* Declares the object of the JFileChooser class. */
JFileChooser jfc;
/* Declares the object of the FileInputStream class. */
FileInputStream fin;
/* Declares the object of the FileChannel class. */
FileChannel fchan;
/* Declares the objects of the ByteBuffer class. */
ByteBuffer buff_in;
ByteBuffer buff_out;
ByteBuffer bbuf;
/* Declares the object of the CharBuffer class. */
CharBuffer cbuf;
/* Declares the object of the CharsetDecoder class. */
CharsetDecoder decoder;
/* Declares the object of the CharsetEncoder class. */
CharsetEncoder encoder;
/* Declares the object of the Charset class. */
Charset charset;
/* Declares the object of the EncodingSchemes class. */
public EncodingSchemes encScheme;
long fsize;
/* Defines the default constructor of the EncoderDecoder class. */
public EncoderDecoder()
{
   /* Sets the title of the Encoder/Decoder application. */
   setTitle("Encoder / Decoder Application");
   /* Sets the size of the Encoder/Decoder application. */
   setSize(497,640);
   /* Sets the visibility of the Encoder/Decoder application to true. */
   setVisible(true);
   /* Sets the re-sizability of the Encoder/Decoder application to false. */
   setResizable(false);
   /* Initializes the object of the GridBagLayout class. */
   gbl = new GridBagLayout();
   /* Sets the Layout */
   getContentPane().setLayout(gbl);
   /* Creates an object of the GridBagConstraints class. */
   gbc = new GridBagConstraints();
   /* Initializes the titleLabel object and adds it to the 0, 0, 3, 1 position with CENTER alignment
   gbc.gridx = 0;
   gbc.gridy = 0;
   gbc.gridwidth = 3;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   titleLabel = new JLabel(" Encoder/Decoder Application");
   titleLabel.setFont(new Font("Verdana", Font.BOLD, 20));
   getContentPane().add(titleLabel, gbc);
   /* Initializes and adds a separator to the 0, 1, 3, 1 position with CENTER alignment. */
   gbc.gridx = 0;
   gbc.gridy = 1;
   gbc.gridwidth = 3;
   gbc.gridheight = 1;
   gbc.anchor = GridBagConstraints.CENTER;
   getContentPane().add(new JSeparator(), gbc);
   /* Initializes the selectLabel object and adds it to the 0, 2, 1, 1 position with WEST alignment.
   gbc.gridx = 0;
   gbc.gridy = 2;
```

```
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
selectLabel = new JLabel("Select File to Encode:");
getContentPane().add(selectLabel, gbc);
/* Initializes and adds a separator to the 0, 3, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 3;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the selectText text field and adds it to the 0, 4, 1, 1 position with WEST alignme
gbc.gridx = 0;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
selectText = new JTextField(30);
selectText.setFont(new Font("Verdana", Font.PLAIN, 12));
getContentPane().add(selectText, gbc);
/* Initializes the browse object and adds it to the 1, 4, 1, 1 position with CENTER alignment. */
gbc.gridx = 1;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
browseButton = new JButton(" Browse ");
browseButton.addActionListener(this);
getContentPane().add(browseButton, gbc);
/* Initializes the reset object and adds it to the 2, 4, 1, 1 position with EAST alignment. */
gbc.gridx = 2;
gbc.gridy = 4;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.EAST;
resetButton = new JButton(" Reset ");
resetButton.addActionListener(this);
getContentPane().add(resetButton, gbc);
/* Initializes and adds a separator to the 0, 5, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 5;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the selectText and adds it to the 0, 6, 1, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 6;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
schemeText = new JTextField(15);
schemeText.setEditable(false);
schemeText.setText(encScheme.encodingScheme);
schemeText.setFont(new Font("Verdana", Font.PLAIN, 12));
getContentPane().add(schemeText, gbc);
/* Initializes the selectButton object and adds it to the 1, 6, 2, 1 position with WEST alignment
gbc.gridx = 1;
gbc.gridy = 6;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
selectButton = new JButton(" Select Encoding Scheme ");
selectButton.addActionListener(this);
getContentPane().add(selectButton, gbc);
/* Initializes and adds a separator to the 0, 7, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 7;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeLabel object and adds it to the 0, 8, 1, 1 position with WEST alignment.
gbc.gridx = 0;
gbc.gridy = 8;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
textLabel = new JLabel("Text to Encode:");
getContentPane().add(textLabel, gbc);
/* Initializes and adds a separator to the 0, 9, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 9;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
```

```
/* Initializes the textArea and textPane objects. Next, this textArea adds to the textPane
and adds it to the 0, 10, 3, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 10;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
textArea = new JTextArea(7, 44);
textArea.addKeyListener(this);
textArea.setLineWrap(true);
textArea.setWrapStyleWord(true);
textArea.setFont(new Font("Verdana", Font.PLAIN, 12));
textPane = new JScrollPane(textArea);
getContentPane().add(textPane, gbc);
/* Initializes and adds a separator to the 0, 11, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 11;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeButton object and adds to it to the 1, 12, 2, 1 position with WEST align
gbc.gridx = 1;
gbc.gridy = 12;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.EAST;
encodeButton = new JButton(" Encode ");
encodeButton.setEnabled(false);
encodeButton.addActionListener(this);
getContentPane().add(encodeButton, gbc);
/* Initializes and adds a separator to the 0, 13, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 13;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeLabel object and adds it to the 0, 14, 1, 1 position with WEST alignment
gbc.gridx = 0;
gbc.gridy = 14;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
encodeLabel = new JLabel("Encoded Text:");
getContentPane().add(encodeLabel, gbc);
/* Initializes and adds a separator to the 0, 15, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 15;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the encodeArea and encodePane objects. Next, this encodeArea adds to the
encodePane and adds it to the 0, 16, 3, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 16;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
encodeArea = new JTextArea(7, 44);
encodeArea.setEditable(false);
encodeArea.addKeyListener(this);
encodeArea.setLineWrap(true);
encodeArea.setWrapStyleWord(true);
encodeArea.setFont(new Font("Verdana", Font.PLAIN, 12));
encodePane = new JScrollPane(encodeArea);
getContentPane().add(encodePane, gbc);
/* Initializes and adds a separator to the 0, 17, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 17;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the decodeButton object and adds to it to the 0, 18, 2, 1 position with EAST align
gbc.gridx = 1;
gbc.gridy = 18;
gbc.gridwidth = 2;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.EAST;
decodeButton = new JButton(" Decode ");
decodeButton.setEnabled(false);
decodeButton.addActionListener(this);
getContentPane().add(decodeButton, gbc);
/* Initializes and adds a separator to the 0, 19, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 19;
```

```
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the decodeLabel object and adds it to the 0, 20, 1, 1 position with WEST alignment
gbc.gridx = 0;
gbc.gridy = 20;
gbc.gridwidth = 1;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.WEST;
decodeLabel = new JLabel("Decoded Text:");
getContentPane().add(decodeLabel, gbc);
/* Initializes and adds a separator to the 0, 21, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 21;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the decodeArea and decodePane objects. Next, this decodeArea adds to the
decodePane and adds it to the 0, 22, 3, 1 position with WEST alignment. */
gbc.gridx = 0;
gbc.gridy = 22;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
decodeArea = new JTextArea(7, 44);
decodeArea.setEditable(false);
decodeArea.setLineWrap(true);
decodeArea.setWrapStyleWord(true);
decodeArea.setFont(new Font("Verdana",Font.PLAIN,12));
decodePane = new JScrollPane(decodeArea);
getContentPane().add(decodePane, gbc);
/* Initializes and adds a separator to the 0, 23, 3, 1 position with CENTER alignment. */
gbc.gridx = 0;
gbc.gridy = 23;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
getContentPane().add(new JSeparator(), gbc);
/* Initializes the closeButton object and adds to it to the 0, 24, 3, 1 position with CENTER alig
gbc.gridx = 0;
gbc.gridy = 24;
gbc.gridwidth = 3;
gbc.gridheight = 1;
gbc.anchor = GridBagConstraints.CENTER;
closeButton = new JButton(" Close ");
closeButton.addActionListener(this);
getContentPane().add(closeButton, gbc);
/*
addWindowListener - It contains a windowClosing() method.
windowClosing: It is called when the end user clicks the cancel button of the Window.
It closes the main window.
Parameter: we- Represents the object of the WindowEvent class.
Return Value: NA
*/
addWindowListener(new WindowAdapter()
{
   public void windowClosing(WindowEvent we)
   {
      System.exit(0);
   }
   });
   /* Initializes the object of the EncodingSchemes class. */
   encScheme = new EncodingSchemes(this);
}
/*
keyTyped() - This method is called when the end user enters any text in
the Encoded Text text area.
Parameters: ke - Represent an object of the KeyEvent class that contains
the details of the event.
Return Value: NA
*/
public void keyTyped(KeyEvent ke)
{
   if(ke.getSource()== textArea)
   {
      /* Enables the Encode button. */
      encodeButton.setEnabled(true);
   }
}
/* Defines the abstract methods defined in the KeyListener interface. */
public void keyPressed(KeyEvent ke){}
public void keyReleased(KeyEvent ke){}
/*
actionPerformed() - This method is called when the end user clicks the any button.
Parameters: ev - Represents an object of the ActionEvent class that contains
```

```
the details of the event.
Return Value: NA
*/
public void actionPerformed(ActionEvent ev)
{
   /* This section is executed when the end user clicks the Select Encoding Schemes button. */
   if(ev.getSource() == selectButton)
   {
      /* Sets the visibility of the Encoding Schemes frame to true. */
       encScheme.setVisible(true);
   }
   /* This section is executed when the end user clicks the Browse button. */
   else if(ev.getSource() == browseButton)
   {
       selectText.setText("");
       try
       {
         /* Initializes the object of the JFileChooser class. */
          jfc = new JFileChooser();
         /* Sets the file selection mode to FILES_ONLY. */
         jfc.setFileSelectionMode(JFileChooser.FILES_ONLY);
         /* Creates and sets the new file filter for selecting the text files*/
         jfc.setFileFilter(new javax.swing.filechooser.FileFilter()
         {
            /* accept() - This method allows the user to select only text files
            from the file chooser dialog box.
            Parameter: f - Represents a file object.
            Return Value: boolean
            */
            public boolean accept(File f)
            {
               if (f.isDirectory())
               {
                  return true;
               }
               /* Returns the file name */
               String name = f.getName();
               /* Checks whether the file name contains .txt or not */
               if (name.endsWith(".txt") || name.endsWith(".TXT"))
               {
                  return true;
               }
               return false;
            }
            /* getDescription() - This method sets the file description. ]
            Parameter - NA
            Returns Value - String
            */
            public String getDescription()
            {
                return ".txt";
            }
         });
         /* Displays the Open dialog box. */
         jfc.showOpenDialog(this);
         /* Gets the file from the selected location. */
         file = jfc.getSelectedFile();
         if(file==null)
         {
          }
          else
          {
            /* Checks the file object is a File type or Directory type. */
             if(file.isFile())
             {
               /* Gets the absolute path of the file. */
               str = file.getAbsolutePath();
               /* Sets the string of absolute path to the Select File to Encode text field. */
               selectText.setText(str);
               /* Calls the open() method. */
               open();
             }
             else
             {
               /* Displays an Error message dialog box. */
               JOptionPane.showMessageDialog(null, "You have selected an
               invalid file format!", "Error", _
               JOptionPane.WARNING_MESSAGE);
             }
          }
       }
       catch(Exception e)
       {
         System.out.println("Error" + e);
       }
    }
   /* This section is executed when the end user clicks the Encode button. */
    else if(ev.getSource() == encodeButton)
```

```
        {
          if (textArea.getText().equals(""))
          {
             /* Displays an Error message dialog box. */
             JOptionPane.showMessageDialog(this,"You have selected an empty file!",
             "Error", JOptionPane.WARNING_MESSAGE);
          }
          else
          {
             /* Calls the encode() method. */
              encode();
             /* Enables the Decode button. */
             decodeButton.setEnabled(true);
             /* Disables the Encode button. */
             encodeButton.setEnabled(false);
          }
       }
       /* This section is executed when the end user clicks the Decode button. */
       else if(ev.getSource()==decodeButton)
       {
          if (encodeArea.getText().equals(""))
          {
             /* Displays an Error dialog box. */
             JOptionPane.showMessageDialog(this, "The file has no content",
             "Select another File", JOptionPane.WARNING_MESSAGE);
          }
          else
          {
             /* Calls the decode() method. */
             decode();
             /* Enables the Encode button. */
             encodeButton.setEnabled(true);
             /* Disables the Decode button. */
             decodeButton.setEnabled(false);
          }
       }
       /* This section is executed when the end user clicks the Reset button. */
       else if(ev.getSource() == resetButton)
       {
          if(textArea.getText().equals("") && encodeArea.getText().equals("")
          && decodeArea.getText().equals("") &&
          selectText.getText().equals(""))
          {
          }
          else
          {
             /* Clears all the text areas, text field, byte buffer and char buffer. */
             textArea.setText("");
             encodeArea.setText("");
             decodeArea.setText("");
             selectText.setText("");
             buff_in.clear();
             bbuf.clear();
             cbuf.clear();
             /* Disables the Encode button. */
             encodeButton.setEnabled(false);
             /* Disables the Decode button. */
             decodeButton.setEnabled(false);
          }
       }
       /* This section is executed when the end user clicks the Close button. */
       else if(ev.getSource()==closeButton)
       {
          System.exit(0);
       }
    }
    /*
    open() - This method is invoked when the end user clicks the Browse button to open the file.
    Parameter - NA
    Return Value - NA
    */
    public void open()
    {
       try
       {
          /* Initializes the object of the FileInputStream class. */
          fin = new FileInputStream(file);
          /* Gets the file channel from the file input stream. */
          fchan = fin.getChannel();
          /* Gets the size of the file. */
          fsize = fchan.size();
          /* Allocates the size of the Buffer. */
          buff_in = ByteBuffer.allocate((int)fsize);
          /* Reads the buffer from the channel. */
          fchan.read(buff_in);
          /* Rewinds the data in the buffer. */
          buff_in.rewind();
          /* Stores the contents of the buff_in buffer to a string. */
```

```
            file_text = new String(buff_in.array());
            if(file_text.equals(""))
            {
               /* Displays an Error dialog box. */
               JOptionPane.showMessageDialog(null, "The file has no content",
               "Select another File", JOptionPane.WARNING_MESSAGE);
            }
            else
            {
               /* Displays the content stored in the file_text string into Encoded Text text area. */
               textArea.setText(file_text);
               /* Enables the Encode button. */
               encodeButton.setEnabled(true);
            }
            /* Closes the file channel. */
            fchan.close();
            /* Closes the file input stream. */
            fin.close();
         }
         catch(IOException ioe)
         {
            System.err.println("I/O Error on Open");
         }
         catch(UnsupportedCharsetException e)
         {
            System.err.println("The File cannot be encoded");
         }
      }
      /*
      encode() - This method is invoked when the end user clicks the Encode
      button to encode the specified text or a file.
      Parameter - NA
      Return Value - NA
      */
      public void encode()
      {
         try
         {
            /* Initializes the object of the Charset class and sets the encoding scheme. */
            charset = Charset.forName(encScheme.encodingScheme);
            /* Initializes the object of the CharsetEncoder class. */
            encoder = charset.newEncoder();
            /* Gets the text from the Encoded Text text area. */
            String str = textArea.getText();
            /* Encodes the text and stores the encoded text in the byte buffer. */
            bbuf = encoder.encode(CharBuffer.wrap(str));
            buff_out = encoder.encode(CharBuffer.wrap(str));
            /* Creates and initializes the object of the StringBuffer class. */
            StringBuffer sb = new StringBuffer();
            for (int j = 0; buff_out.hasRemaining(); j++)
            {
               /* Converts the bytes into hexadecimal string. */
               int b = buff_out.get();
               int ival = ((int) b) & 0xff;
               char c = (char) ival;
               sb.append(Integer.toHexString (ival));
            }
            String content = sb.toString();
            /* Clears the buff_out byte buffer. */
            buff_out.clear();
            /* Displays the encoded text in the Encoded Text text area. */
            encodeArea.setText(content);
         }
         catch(Exception e)
         {
            System.out.println(e);
         }
      }
      /*
      decode() - This method is invoked when the end user clicks the Decode
      button to decode the encoded text into readable format.
      Parameter - NA
      Return Value - NA
      */
      public void decode()
      {
         try
         {
            /* Initializes the object of the CharsetDecoder class. */
            decoder = charset.newDecoder();
            /* Decodes the encoded text into a readable form and stores it into a
            character buffer. */
            cbuf = decoder.decode(bbuf);
            /* Converts the data stored in cbuf to a string. */
            String s = cbuf.toString();
            /* Displays the decoded text into the Decoded Text text area. */
            decodeArea.setText(s);
            /* Clears the byte and char buffers. */
```

```
            bbuf.clear();
            cbuf.clear();
         }
         catch(Exception e)
         {
            System.out.println(e);
         }
      }
      /*
      main() - This method creates the main window of the user interface and displays it.
      Parameters:
      args[] - Contains any command line arguments passed.
      Return Value: NA
      */
      public static void main(String[] args)
      {
         try
         {
            /* Sets the look and feel of the Encoder/Decoder application to window look and feel. */
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
         }
         catch(Exception e)
         {
            System.out.println("Unknown Look and Feel." + e);
         }
         /* Creates and initializes the object of the EncoderDecoder class. */
         EncoderDecoder ed = new EncoderDecoder();
         /* Displays the main window of the Encoder/Decoder application. */
         ed.show();
      }
}
```

Download this Listing.

In the above code, the main() method creates an instance of the EncoderDecoder class. This class generates the main window of the Encoder/Decoder application, as shown in Figure 8-2:


**Figure 8-2:** The Encoder/Decoder Application User Interface

To open an existing text file, end users can use the Browse button in the user interface. The Text to Encode text area allows end users to specify the text that is to be encoded. The `Encoded Text` text area displays the encoded text whereas the Decoded Text text area displays the decoded text.

The methods defined in the above code are:

- `keyTyped()`: Acts as an event listener and activates an appropriate method based on the text that is to be encoded. When an end user specifies the text in the Text to Encode text area, this method calls the setEnabled() method to enable the Encode button.

- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method based on the button an end user clicks. If the Browse button is clicked, the actionPerformed() method initializes the object of the JFileChooser class and sets the file selection mode to FILES_ONLY. This method then creates and adds a file filter using the JFileChooser object to select only text files. The actionPerformed() method calls the showOpenDialog() method of the JFileChooser class to display the Open dialog box. Next, the actionPerformed() method retrieves the path of the selected file from the file chooser dialog box, displays the path in the Select File to Encode text field, and calls the open() method. If the end user clicks the Reset button, the actionPerformed() method clears the text displayed in the text box and text areas. This method also calls the clear() method of the ByteBuffer class to clear the byte and char buffers. If the end user clicks the Select Encoding Scheme button, the actionPerformed() method calls the setVisible() method of the JFrame class to display the Encoding Schemes window. When the end user clicks the Encode button, the actionPerformed() method calls the encode() method to encode the specified text that is displayed in the Encoded Text text area. If an end user clicks the Decode button, the actionPerformed() method calls the decode() method to decode and display the decoded text in the Decoded Text text area.
- `open()`: Initializes the object of the FileInputStream class and gets the file channel from the file input stream. The open() method gets the size of the file and allocates that size to the byte buffer. The open() method then reads the buffer from the channel and rewinds the buffered data. Next, the open() method stores the contents of the byte buffer to a string and sets it to the Text to Encode text area.
- `encode()`: Initializes the object of the Charset and CharsetEncoder classes, and sets the encoding scheme. The encode() method gets the text from the Text to Encode text area, encodes it, and stores the encoded text in the byte buffer. The open() method then creates and initializes the object of the StringBuffer class, converts the byte to a hexadecimal string, and displays this string in the Encoded Text text area.
- `decode()`: Initializes the object of the CharsetDecoder class. This method decodes the encoded text into readable form and stores it in a character buffer. The decode() method then converts the data stored in the char buffer to a string and displays the string in the Decoded Text text area.

# Creating the Encoding Schemes Dialog Box

The EncodingSchemes.java file allows you to create an Encoding Schemes dialog box. This dialog box provides options to select an encoding scheme to encode the text.

Listing 8-2 shows the contents of the EncodingSchemes.java file:

**Listing 8-2: The EncodingSchemes.java File**

```
/* Imports the java.awt package classes. */
import java.awt.*;
/* Imports the java.nio.charset package classes. */
import java.nio.charset.Charset;
/* Imports the java.util package classes. */
import java.util.SortedMap;
import java.util.Map;
import java.util.Iterator;
import java.util.Collection;
import java.lang.Object;
import java.util.Enumeration;
/* Imports the java.awt.event package classes. */
import java.awt.event.*;
/* Imports the javax.swing package classes. */
import javax.swing.*;
import javax.swing.ButtonModel;
/*
class EncodingSchemes - This class creates a window that displays various encoding schemes.
 Fields:
   title - Creates a title label.
   ok - Creates a OK button.
   cancel - Creates a Cancel button.
   rb - Creates the radio button.
   buttonPanel - Creates a panel for button.
   schemePanel - Creates a panel to display the encoding scheme.
   numberPanel - Creates a panel to display the number of encoding scheme.
   mainPanel - Creates a main panel.
   encodingScheme - Contains the encoding scheme.
   cs- Creates an object of the Charset class.
   c - Creates an object of the Collection class.
   Methods:
   getSchemes() - This method is executed when the EncodingSchemes constructor loads.
   actionPerformed() - This method is invoked when an end user clicks any button of the EncodingSchei
*/
public class EncodingSchemes extends JDialog implements ActionListener
{
   /* Declares the object of the SortedMap class. */
   SortedMap sm;
   /* Declares the object of the JLabel class. */
   JLabel title;
   /* Declares the objects of the JButton class. */
   JButton ok;
   JButton cancel;
   /* Declares the object of the String class. */
   String temp;
   /* Declares the object of the ButtonGroup class. */
   ButtonGroup bg;
   /* Declares the object of the JRadioButton class. */
   JRadioButton rb;
   /* Declares the object of the Charset class. */
   Charset cs;
   /* Declares the object of the Collection class. */
   Collection c;
   /* Declares the objects of the JPanel class. */
   JPanel buttonPanel;
   JPanel schemePanel;
   JPanel numberPanel;
   JPanel mainPanel;
   JScrollPane s_pane;
   /* Declares the object of the Font class. */
   Font f;
   int s;
   public EncoderDecoder ed;
   /* Declares the object of the String class. */
   public static String encodingScheme = "UTF-16";
   /* Defines the default constructor of the EncodingSchemes class. */
   public EncodingSchemes(EncoderDecoder ed)
   {
      this.ed = ed;
      /* Sets the title of the Encoding Schemes window. */
      setTitle("Encoding Schemes");
      /* Sets the size of the Encoding Schemes window. */
      setSize(280, 500);
      /* Sets the re-sizability of the Encoding Schemes window to false. */
```

```
    setResizable(false);
    /* Initializes the object of the ButtonGroup class. */
    bg=new ButtonGroup();
    /* Initializes the object of the Font class. */
    f=new Font("Verdana", Font.BOLD, 12);
    /* Initializes the objects of the JPanel class. */
    buttonPanel=new JPanel();
    schemePanel=new JPanel();
    numberPanel=new JPanel();
    mainPanel=new JPanel();
    /* Initializes the object of the JLabel class. */
    title = new JLabel("Select the Encoding Scheme");
    title.setFont(f);
    /* Initializes and sets the layout of the button panel to grid layout. */
    buttonPanel.setLayout(new GridLayout(1, 2));
    /* Initializes and sets the layout of the number panel to border layout. */
    numberPanel.setLayout(new BorderLayout());
    /* Initializes and sets the layout of the main panel to border layout. */
    mainPanel.setLayout(new BorderLayout());
    /* Initializes the objects of the JButton class. */
    ok=new JButton("OK");
    cancel=new JButton("Cancel");
    /* Adds the action listener events to the Ok and Cancel buttons. */
    ok.addActionListener(this);
    cancel.addActionListener(this);
    /* Calls the getSchemes() method. */
    getSchemes();
    /* Adds various swing components on the panels. */
    numberPanel.add(title, BorderLayout.NORTH);
    buttonPanel.add(ok);
    buttonPanel.add(cancel);
    mainPanel.add(numberPanel, BorderLayout.NORTH);
    mainPanel.add(s_pane, BorderLayout.CENTER);
    mainPanel.add(buttonPanel, BorderLayout.SOUTH);
    /* Adds the main panel to frame. */
    getContentPane().add(mainPanel);
    /*
    addWindowListener - It contains a windowClosing() method.
    windowClosing(): This method is called when the end user clicks the close button of the Window
    Parameter: we- Represents an object of the WindowEvent class.
    Return Value: NA
    */
    addWindowListener(new WindowAdapter()
    {
       public void windowClosing(WindowEvent we)
       {
          dispose();
       }
    });
}
/*
getSchemes() - This method returns the selected encoding scheme.
Parameter - NA
Return Value - NA
*/
public void getSchemes()
{
    try
    {
       /* Retrieves the available character schemes. */
       sm=cs.availableCharsets();
       /* Gets the size of sorted map object. */
       s=sm.size();
       /* Sets the layout of the schemePanel panel to the GridLayout. */
       schemePanel.setLayout(new GridLayout(s, 1));
       /* Retrieves the value from the sorted model object. */
       c=sm.values();
       /* Creates and initializes the object of the Iterator class. */
       Iterator i=c.iterator();
       while(i.hasNext())
       {
          temp = i.next().toString();
          /* Initializes the object of the JRadioButton class. */
          rb = new JRadioButton(temp);
          if(temp.equals("UTF-16"))
          {
             rb.setSelected(true);
          }
          /* Adds the radio button to the button group. */
          bg.add(rb);
          /* Adds the radio button group to the schemePanel panel. */
          schemePanel.add(rb);
          s_pane = new JScrollPane(schemePanel);
          }
       }catch(Exception e)
       {
          System.out.println(e);
       }
```

```
        }
        /*
        actionPerformed() - This method is called when the end user clicks the any button.
        Parameters: ev - Represents an objects of the ActionEvent class that contains the details of t
        Return Value: NA
        */
        public void actionPerformed(ActionEvent ev)
        {
           /* This section is executed when the end user clicks the OK button. */
           if(ev.getSource()==ok)
           {
              /* Creates and initializes the object of the Enumeration class to
              get the elements from the button group. */
              Enumeration enum=bg.getElements();
              while(enum.hasMoreElements())
              {
                 JRadioButton button = (JRadioButton)enum.nextElement();
                 /* Checks the status of a radio button. */
                 if (button.isSelected())
                 {
                    /* Retrieves the text from the radio button. */
                    encodingScheme = button.getText();
                    ed.schemeText.setText(encodingScheme);
                 }
              }
              /* Disposes the window. */
              dispose();
           }
           /* This section is executed when the end user clicks the Cancel button. */
           else if(ev.getSource()==cancel)
           {
              /* Disposes the window. */
              dispose();
           }
     }
}
```

Download this Listing.

In the above code, the constructor of this class creates the Encoding Schemes dialog box for the Encoder/Decoder application, as shown in Figure 8-3:

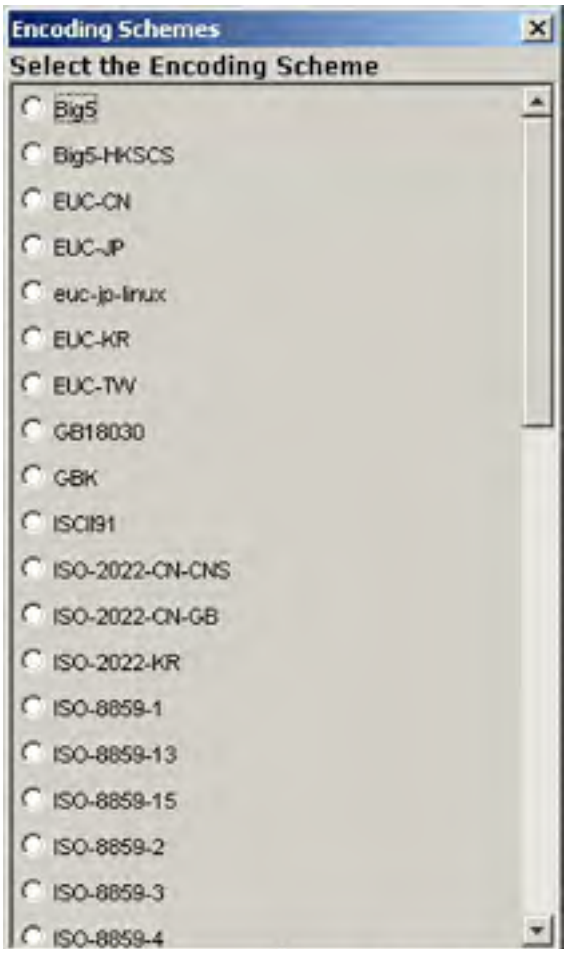


**Figure 8-3:** The Encoding Schemes Dialog Box

The methods defined in the above code are:

- `getSchemes()`: Retrieves the available character schemes and stores it to the sorted map. The getSchemes() method gets the size of the sorted map in the integer variable, s, and sets the layout of the schemePanel panel to sX1 grid layout. This method then retrieves the value from the sorted model object and initializes the object of the JRadioButton class. Finally, this method adds the radio button to the button group and the radio button group to the schemePanel panel.

- `actionPerformed()`: Acts as an event listener and activates an appropriate class or method based on the button the end user clicks. If the OK button is clicked, the actionPerformed() method checks the status of the radio button. This method then gets the text associated with this radio button and stores it to the static variable, encodingScheme. When an end user clicks the Cancel button, the actionPerformed() method calls the dispose() method to close the Encoding Schemes window.



◄ PREVIOUS NEXT ►

## Unit Testing

To test the Encoder/Decoder application:

1. 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;
   ```
2. 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;
   ```
3. Copy the EncoderDecoder.java and EncodingSchemes.java files to a folder on your computer. On the command prompt, use the cd command to move to the folder where you have copied the Java files. Compile the files using the following javac command:
   ```
   javac *.java
   ```
4. To run the Encoder/Decoder application, specify the following command at the command prompt:
   ```
   java EncoderDecoder
   ```
   The above code opens the main window of the Encoder/Decoder application, as shown in Figure 8-2.
5. Click the Browse button and select the test.txt file from the C drive of the computer.

   Figure 8-4 shows the contents of the test.txt file:


**Figure 8-4:** Displaying test.txt File in the Encoder/Decoder Application

6. Click the Select Encoding Scheme button to open the Encoding Schemes dialog box and select the UTF-16BE encoding scheme.
7. Click the OK button of the Encoding Schemes dialog box.
8. Click the Encode button to encode the test.txt file using the UTF-16BE encoding scheme, as shown in Figure 8-5:


**Figure 8-5:** Displaying Encoded File in the Encoder/Decoder Application

9. Click the Decode button to decode the encoded file, as shown in Figure 8-6:


**Figure 8-6:** Displaying Decoded File in the Encoder/Decoder Application

10. Click the Reset button to reset the Encoder/Decoder application.
11. Specify the text you want to encode in the Text to Encode text area, as shown in Figure 8-7:


**Figure 8-7:** Displaying Text to Encode in the Encoder/Decoder Application

12. Repeat steps 6 through 9 to encode and decode the text specified in the Text to Encode text area. The results are displayed in Figure 8-8:


**Figure 8-8:** Displaying Encoded and Decoded Text

# Index

## A-J

# Index

## N-S

# Index

## T

# List of Figures

## Chapter 6: Creating a Text Editor Application



## Chapter 7: Creating a Network Information Application



## Chapter 8: Creating an Encoder/Decoder Application

# List of Examples

## Chapter 2: Creating a Chat Application



## Chapter 3: Creating a File Download Application



## Chapter 4: Creating a File Search Application



## Chapter 5: Creating a Printer Management Application



## Chapter 6: Creating a Text Editor Application



## Chapter 7: Creating a Network Information Application

# Chapter 8: Creating an Encoder/Decoder Application





PREVIOUS


**Java InstantCode: Developing Applications Using Java NIO**

SkillSoft Press © 2004

Learn about the New I/O (NIO) API introduced in J2SDK v 1.4 that provides new features and improved performance in the areas of polling, buffer management, character converters, and much more.

**Table of Contents**