24

Networking

◀ ▶

If the presence of electricity can be made visible in any part of a circuit, I see no reason why intelligence may not be transmitted instantaneously by electricity.

— Samuel F. B. Morse

Protocol is everything.

— Francois Giuliani

What networks of railroads, highways and canals were in another age, the networks of telecommunications, information and computerization … are today.

— Bruno Kreisky

The port is near, the bells I hear, the people all exulting.

— Walt Whitman

◀ ▶

OBJECTIVES

In this chapter you will learn:

- To understand Java networking with URLs, sockets and datagrams.
- To implement Java networking applications by using sockets and datagrams.
- To understand how to implement Java clients and servers that communicate with one another.
- To understand how to implement network-based collaborative applications.
- To construct a multithreaded server.

24.1 Introduction

24.2 Manipulating URLs

24.3 Reading a File on a Web Server

24.4 Establishing a Simple Server Using Stream Sockets

24.5 Establishing a Simple Client Using Stream Sockets

24.6 Client/Server Interaction with Stream Socket Connections

24.7 Connectionless Client/Server Interaction with Datagrams

24.8 Client/Server Tic-Tac-Toe Using a Multithreaded Server

24.9 Security and the Network

24.10 Case Study: DeitelMessenger Server and Client

- 24.10.1 `DeitelMessengerServer` and Supporting Classes
- 24.10.2 DeitelMessenger Client and Supporting Classes

24.11 Wrap-Up

24.1 Introduction

- Networking package is `java.net`
  - Stream-based communications
    - Applications view networking as streams of data
    - Connection-based protocol
    - Uses TCP (Transmission Control Protocol
  - Packet-based communications
    - Individual packets transmitted
    - Connectionless service
    - Uses UDP (User Datagram Protocol)

24.1 Introduction (Cont.)

- Client-server relationship
  - Client requests some action be performed
  - Server performs the action and responds to client
  - Request-response model
    - Common implementation: Web browsers and Web servers

Performance Tip 24.1

Connectionless services generally offer greater performance but less reliability than connection-oriented services.

Portability Tip 24.1

TCP, UDP and related protocols enable a great variety of heterogeneous computer systems (i.e., computer systems with different processors and different operating systems) to intercommunicate.

24.2 Manipulating URLs

- HyperText Transfer Protocol (HTTP)
  - Uses URIs (Uniform Resource Identifiers) to identify data
    - URLs (Uniform Resource Locators)
      - URIs that specify the locations of documents
      - Refer to files, directories and complex objects
- HTML document `SiteSelector.html` (Fig. 24.1)
  - `applet` element
  - `param` tag
    - `name` attribute
    - `value` attribute

```
 1 <html>
 2 <title>Site Selector</title>
 3 <body>
 4    <applet code = "SiteSelector.class" width = "300" height = "75">
 5       <param name = "title0" value = "Java Home Page">
 6       <param name = "location0" value = "http://java.sun.com/">
 7       <param name = "title1" value = "Deitel">
 8       <param name = "location1" value = "http://www.deitel.com/">
 9       <param name = "title2" value = "JGuru">
10       <param name = "location2" value = "http://www.jGuru.com/">
11       <param name = "title3" value = "JavaWorld">
12       <param name = "location3" value = "http://www.javaworld.com/">
13    </applet>
14 </body>
15 </html>
```

Outline

SiteSelector.html

Lines 5-12

Fig.24.17 | **HTML document to load `SiteSelector` applet.**

```
1  // Fig. 24.2: SiteSelector.java
2  // This program loads a document from a URL.
3  import java.net.MalformedURLException;
4  import java.net.URL;
5  import java.util.HashMap;
6  import java.util.ArrayList;
7  import java.awt.BorderLayout;
8  import java.applet.AppletContext;
9  import javax.swing.JApplet;
10 import javax.swing.JLabel;
11 import javax.swing.JList;
12 import javax.swing.JScrollPane;
13 import javax.swing.event.ListSelectionEvent;
14 import javax.swing.event.ListSelectionListener;
15
16 public class SiteSelector extends JApplet
17 {
18    private HashMap< Object, URL > sites; // site names and URLs
19    private ArrayList< String > siteNames; // site names
20    private JList siteChooser; // list of sites to choose from
21
22    // read HTML parameters and set up GUI
23    public void init()
24    {
25       sites = new HashMap< Object, URL >(); // create HashMap
26       siteNames = new ArrayList< String >(); // create ArrayList
27
28       // obtain parameters from HTML document
29       getSitesFromHTMLParameters();
30
```

Outline

SiteSelector.java

(1 of 5)

Lines 3-4

Line 8

```
31       // create GUI components and layout interface
32       add( new JLabel( "Choose a site to browse" ), BorderLayout.NORTH );
33
34       siteChooser = new JList( siteNames.toArray() ); // populate JList
35       siteChooser.addListSelectionListener(
36          new ListSelectionListener() // anonymous inner class
37          {
38             // go to site user selected
39             public void valueChanged( ListSelectionEvent event )
40             {
41                // get selected site name
42                Object object = siteChooser.getSelectedValue();
43
44                // use site name to locate corresponding URL
45                URL newDocument = sites.get( object );
46
47                // get applet container
48                AppletContext browser = getAppletContext();
49
50                // tell applet container to change pages
51                browser.showDocument( newDocument );
52             } // end method valueChanged
53          } // end anonymous inner class
54       ); // end call to addListSelectionListener
55
56       add( new JScrollPane( siteChooser ), BorderLayout.CENTER );
57    } // end method init
58
```

Outline

SiteSelector.java

(2 of 5)

Lines 39-52

Line 45

Line 48

Line 51

```
59    // obtain parameters from HTML document
60    private void getSitesFromHTMLParameters()
61    {
62       String title; // site title
63       String location; // location of site
64       URL url; // URL of location
65       int counter = 0; // count number of sites
66
67       title = getParameter( "title" + counter ); // get first site title
68
69       // loop until no more parameters in HTML document
70       while ( title != null )
71       {
72          // obtain site location
73          location = getParameter( "location" + counter );
74
75          try // place title/URL in HashMap and title in ArrayList
76          {
77             url = new URL( location ); // convert location to URL
78             sites.put( title, url ); // put title/URL in HashMap
79             siteNames.add( title ); // put title in ArrayList
80          } // end try
81          catch ( MalformedURLException urlException )
82          {
83             urlException.printStackTrace();
84          } // end catch
85
```

Outline

SiteSelector.java

(3 of 5)

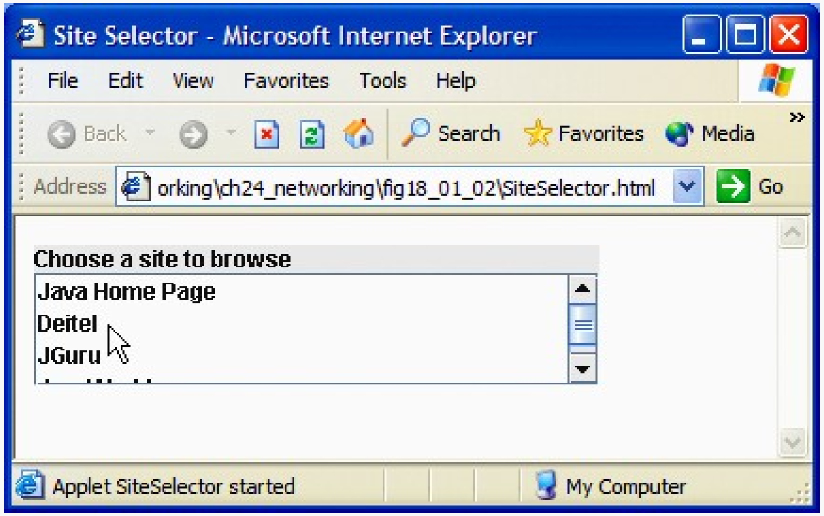
Line 67

Line 73

Line 77

Lines 81-84

```
86          counter++;
87          title = getParameter( "title" + counter ); // get next site title
88       } // end while
89    } // end method getSitesFromHTMLParameters
90 } // end class SiteSelector
```


Site Selector - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
orking\ch24_networking\fig18_01_02\SiteSelector.html
Go

Choose a site to browse

Java Home Page
Deitel
JGuru

Applet SiteSelector started

My Computer

Outline

SiteSelector.java

(4 of 5)

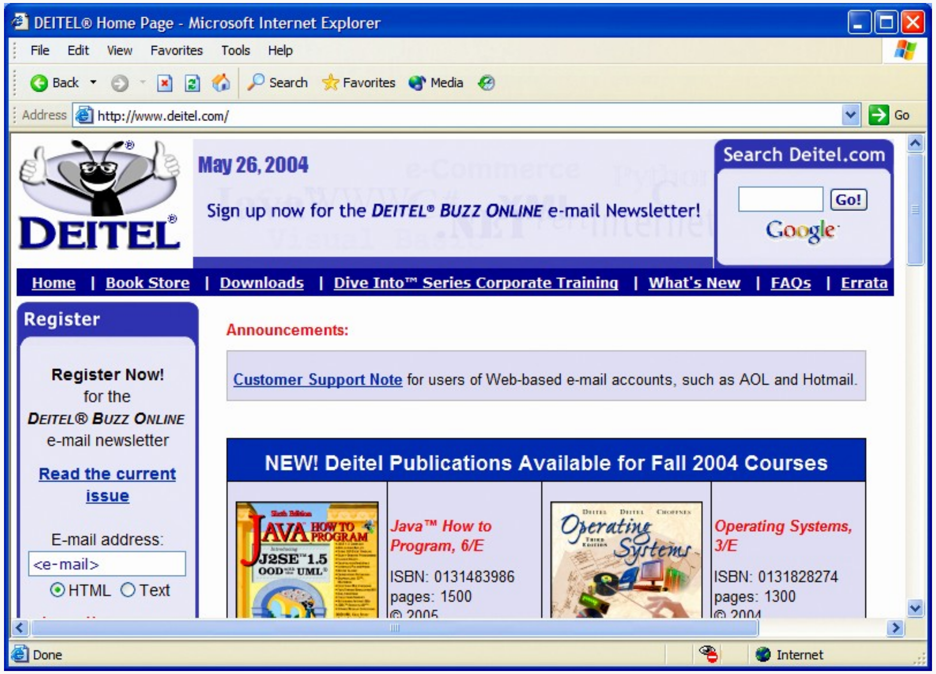
Line 87

Program output

SiteSelector.java

(5 of 5)

Program output


DEITEL® Home Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.deitel.com/
Go
Search Deitel.com
Go!
Google
Home | Book Store | Downloads | Dive Into™ Series Corporate Training | What's New | FAQs | Errata
Done
Internet

May 26, 2004

Sign up now for the DEITEL® BUZZ ONLINE e-mail Newsletter!

Register

Register Now!
for the
DEITEL® BUZZ ONLINE
e-mail newsletter

Read the current issue

E-mail address:
<e-mail>
● HTML ○ Text

Announcements:

Customer Support Note for users of Web-based e-mail accounts, such as AOL and Hotmail.

NEW! Deitel Publications Available for Fall 2004 Courses

Java™ How to Program, 6/E
ISBN: 0131483986
pages: 1500
© 2005

Operating Systems, 3/E
ISBN: 0131828274
pages: 1300
© 2004

24.2 Manipulating URLs

- **HTML frames**
  - **Specify target frame in method `showDocument`**
    - **`_blank`**
    - **`_self`**
    - **`_top`**

Error-Prevention Tip 24.1

The applet in Fig. 24.2 must be run from a Web browser, such as Mozilla or Microsoft Internet Explorer, to see the results of displaying another Web page. The `appletviewer` is capable only of executing applets—it ignores all other HTML tags. If the Web sites in the program contained Java applets, only those applets would appear in the `appletviewer` when the user selected a Web site. Each applet would execute in a separate `appletviewer` window.

24.3 Reading a File on a Web Server

- Swing GUI component `JEditorPane`
  - Render both plain text and HTML-formatted text
  - Act as a simple Web browser
    - Retrieves files from a Web server at a given URI
    - `HyperlinkEvent`s
      - Occur when the user clicks a hyperlink
      - Three event types
        - `HyperlinkEvent.EventType.ACTIVATED`
        - `HyperlinkEvent.EventType.ENTERED`
        - `HyperlinkEvent.EventType.EXITED`

```
1  // Fig. 24.3: ReadServerFile.java
2  // Use a JEditorPane to display the contents of a file on a Web server.
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.io.IOException;
7  import javax.swing.JEditorPane;
8  import javax.swing.JFrame;
9  import javax.swing.JOptionPane;
10 import javax.swing.JScrollPane;
11 import javax.swing.JTextField;
12 import javax.swing.event.HyperlinkEvent;
13 import javax.swing.event.HyperlinkListener;
14
15 public class ReadServerFile extends JFrame
16 {
17    private JTextField enterField; // JTextField to enter site name
18    private JEditorPane contentsArea; // to display Web site
19
20    // set up GUI
21    public ReadServerFile()
22    {
23       super( "Simple Web Browser" );
24
```

Outline

ReadServerFile .java

(1 of 3)

Lines 7, 12 and 13

Line 18

```
25       // create enterField and register its listener
26       enterField = new JTextField( "Enter file URL here" );
27       enterField.addActionListener(
28          new ActionListener()
29          {
30             // get document specified by user
31             public void actionPerformed( ActionEvent event )
32             {
33                getThePage( event.getActionCommand() );
34             } // end method actionPerformed
35          } // end inner class
36       ); // end call to addActionListener
37
38       add( enterField, BorderLayout.NORTH );
39
40       contentsArea = new JEditorPane(); // create contentsArea
41       contentsArea.setEditable( false );
42       contentsArea.addHyperlinkListener(
43          new HyperlinkListener()
44          {
45             // if user clicked hyperlink, go to specified page
46             public void hyperlinkUpdate( HyperlinkEvent event )
47             {
48                if ( event.getEventType() ==
49                     HyperlinkEvent.EventType.ACTIVATED )
50                   getThePage( event.getURL().toString() );
51             } // end method hyperlinkUpdate
52          } // end inner class
53       ); // end call to addHyperlinkListener
54
```

Outline

`ReadServerFile.java`

(2 of 3)

Line 40

Line 41

Lines 42-53

Lines 46-51

Lines 48-49

Line 50

```
55       add( new JScrollPane( contentsArea ), BorderLayout.CENTER );
56       setSize( 400, 300 ); // set size of window
57       setVisible( true ); // show window
58    } // end ReadServerFile constructor
59
60    // load document
61    private void getThePage( String location )
62    {
63       try // load document and display location
64       {
65          contentsArea.setPage( location ); // set the page
66          enterField.setText( location ); // set the text
67       } // end try
68       catch ( IOException ioException )
69       {
70          JOptionPane.showMessageDialog( this,
71             "Error retrieving specified URL", "Bad URL",
72             JOptionPane.ERROR_MESSAGE );
73       } // end catch
74    } // end method getThePage
75 } // end class ReadServerFile
```

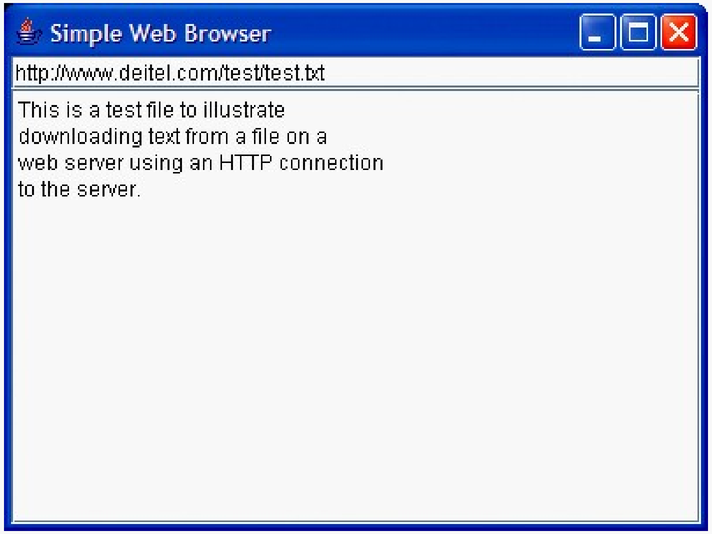
Outline

ReadServerFile.java

(3 of 3)

Line 65

```java
// Fig. 24.4: ReadServerFileTest.java
// Create and start a ReadServerFile.
import javax.swing.JFrame;

public class ReadServerFileTest
{
   public static void main( String args[] )
   {
      ReadServerFile application = new ReadServerFile();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   } // end main
} // end class ReadServerFileTest
```



http://www.deitel.com/test/test.txt

```
This is a test file to illustrate
downloading text from a file on a
web server using an HTTP connection
to the server.
```

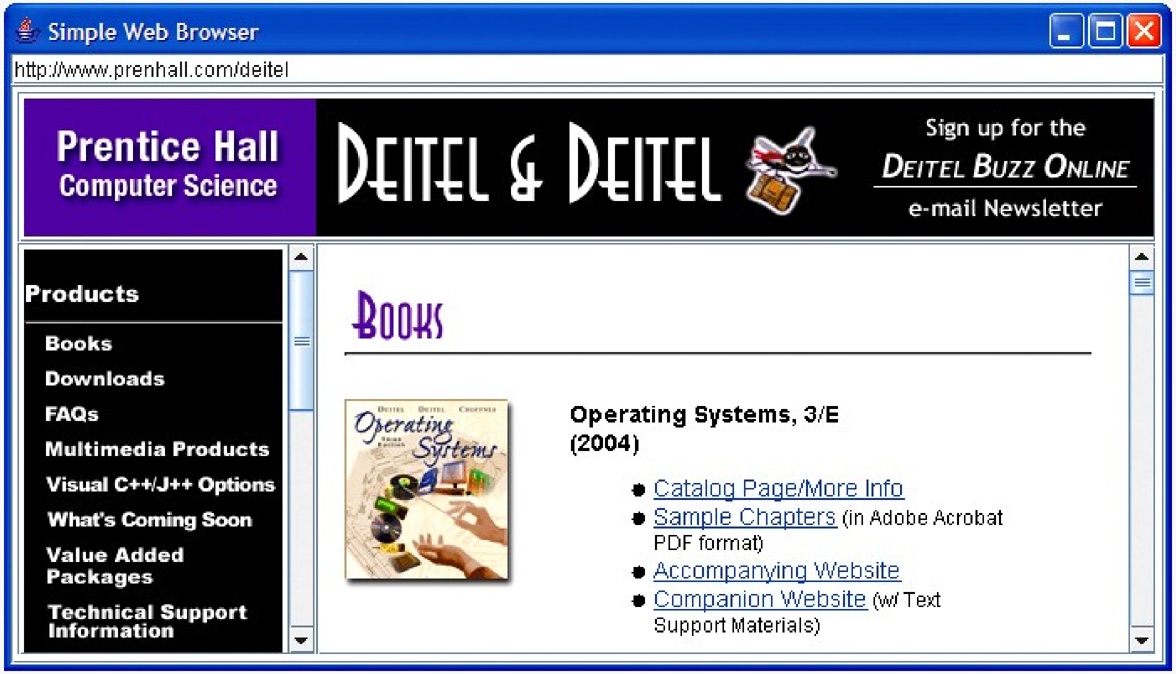
Outline

ReadServerFileTest.java

(1 of 2)

Program output

Outline



Operating Systems, 3/E
(2004)

- Catalog Page/More Info
- Sample Chapters (in Adobe Acrobat PDF format)
- Accompanying Website
- Companion Website (w/ Text Support Materials)

ReadServerFileTest.java

(2 of 2)

Program output

Look-and-Feel Observation 24.1

A `JEditorPane` generates `HyperlinkEvents` only if it is uneditable.

24.4 Establishing a Simple Server Using Stream Sockets

- **Five steps to create a simple server in Java**
  - ***Step 1*: Create `ServerSocket` object**
    - `ServerSocket server = new ServerSocket( portNumber, queueLength );`
    - **Register an available port**
    - **Specify a maximum number of clients**
    - **Handshake point**
    - **Binding the server to the port**
      - **Only one client can be bound to a specific port**

Software Engineering Observation 24.1

Port numbers can be between 0 and 65,535. Most operating systems reserve port numbers below 1024 for system services (e.g., e-mail and World Wide Web servers). Generally, these ports should not be specified as connection ports in user programs. In fact, some operating systems require special access privileges to bind to port numbers below 1024.

24.4 Establishing a Simple Server Using Stream Sockets (Cont.)

- **Five steps to create a simple server in Java**
  - **_Step 2_: Server listens for client connection**
    - **Server blocks until client connects**
    - **`Socket connection = server.accept();`**
  - **_Step 3_: Sending and receiving data**
    - **`OutputStream` to send and `InputStream` to receive data**
      - **Method `getOutputStream` returns `Socket`'s `OutputStream`**
      - **Methods `getInputstream` returns `Socket`'s `InputStream`**

24.4 Establishing a Simple Server Using Stream Sockets (Cont.)

- **Five steps to create a simple server in Java**
  - ***Step 4*: Process phase**
    - **Server and Client communicate via streams**
  - ***Step 5*: Close streams and connections**
    - **Method `close`**

Software Engineering Observation 24.2

With sockets, network I/O appears to Java programs to be similar to sequential file I/O. Sockets hide much of the complexity of network programming from the programmer.

Software Engineering Observation 24.3

With Java’s multithreading, we can create multithreaded servers that can manage many simultaneous connections with many clients. This multithreaded-server architecture is precisely what popular network servers use.

Software Engineering Observation 24.4

A multithreaded server can take the `Socket` returned by each call to `accept` and create a new thread that manages network I/O across that `Socket`. Alternatively, a multithreaded server can maintain a pool of threads (a set of already existing threads) ready to manage network I/O across the new `Socket`s as they are created. See Chapter 23 for more information on multithreading.

Performance Tip 24.2

In high-performance systems in which memory is abundant, a multithreaded server can be implemented to create a pool of threads that can be assigned quickly to handle network I/O across each new `Socket` as it is created. Thus, when the server receives a connection, it need not incur the overhead of thread creation. When the connection is closed, the thread is returned to the pool for reuse.

24.5 Establishing a Simple Client Using Stream Sockets

- **Four steps to create a simple client in Java**
  - **_Step 1_: Create a `Socket` to connect to server**

    ```
    Socket connection = new Socket (
        serverAddress, port );
    ```

  - **_Step 2_: Obtain `Socket`’s `InputStream` and `Outputstream`**
  - **_Step 3_: Process information communicated**
  - **_Step 4_: Close streams and connection**

24.6 Client/Server Interaction with Stream Socket Connections

- Client/server chat application
  - Uses stream sockets
  - Server waits for a client connection attempt
  - Client connects to the server
    - Send and receive messages
  - Client or server terminates the connection
  - Server waits for the next client to connect

Outline

```
1  // Fig. 24.5: Server.java
2  // Set up a Server that will receive a connection from a client, send
3  // a string to the client, and close the connection.
4  import java.io.EOFException;
5  import java.io.IOException;
6  import java.io.ObjectInputStream;
7  import java.io.ObjectOutputStream;
8  import java.net.ServerSocket;
9  import java.net.Socket;
10 import java.awt.BorderLayout;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13 import javax.swing.JFrame;
14 import javax.swing.JScrollPane;
15 import javax.swing.JTextArea;
16 import javax.swing.JTextField;
17 import javax.swing.SwingUtilities;
18
19 public class Server extends JFrame
20 {
21    private JTextField enterField; // inputs message from user
22    private JTextArea displayArea; // display information to user
23    private ObjectOutputStream output; // output stream to client
24    private ObjectInputStream input; // input stream from c
25    private ServerSocket server; // server socket
26    private Socket connection; // connection to client
27    private int counter = 1; // counter of number of connections
28
```

Import `ServerSocket` and `Socket` from package `java.net`

Declare `ServerSocket server`

Declare `Socket connection` which connects to the client

`Server.java`

9

Line 25

Line 26

Server.java

(2 of 8)

```
   // set up GUI
   public Server()
   {
      super( "Server" );

      enterField = new JTextField(); // create enterField
      enterField.setEditable( false );
      enterField.addActionListener(
         new ActionListener()
         {
            // send message to client
            public void actionPerformed( ActionEvent event )
            {
               sendData( event.getActionCommand() );
               enterField.setText( "" );
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener

      add( enterField, BorderLayout.NORTH );

      displayArea = new JTextArea(); // create displayArea
      add( new JScrollPane( displayArea ), BorderLayout.CENTER );

      setSize( 300, 150 ); // set size of window
      setVisible( true ); // show window
   } // end Server constructor

```

Server.java

```
   // set up and run server
   public void runServer()
   {
      try // set up server to receive connections; process connections
      {
         server = new ServerSocket( 12345, 100 ); // create ServerSocket

         while ( true )
         {
            try
            {
               waitForConnection(); // wait for a connection
               getStreams(); // get input & output streams
               processConnection(); // process connection
            } // end try
            catch ( EOFException eofException )
            {
               displayMessage( "\nServer terminated connection" );
            } // end catch
```

Create `ServerSocket` at port 12345 with queue of length 100

Wait for a client

After the connection is

Send the initial connection message to the client and process all messages received from the client

```
76             finally
77             {
78                closeConnection(); // close connection
79                counter++;
80             } // end finally
81          } // end while
82       } // end try
83       catch ( IOException ioException )
84       {
85          ioException.printStackTrace();
86       } // end catch
87    } // end method runServer
88
89    // wait for connection to arrive, then display connection info
90    private void waitForConnection() throws IOException
91    {
92       displayMessage( "Waiting for connection\n" );
93       connection = server.accept(); // allow server to accept connection
94       displayMessage( "Connection " + counter + " received from: " +
95          connection.getInetAddress().getHostName() );
96    } // end method waitForConnection
97
98    // get streams to send and receive data
99    private void getStreams() throws IOException
100   {
101      // set up output stream for objects
102      output = new ObjectOutputStream( connection.getOutputStream() );
103      output.flush(); // flush output buffer to send header information
104
```

Server.java

(4 of 8)

Line 93

Line 95

Line 102

Line 103

Outline

```
105         // set up input stream for objects
106         input = new ObjectInputStream( connection.getInputStream() );
107
108         displayMessage( "\nGot I/O streams\n" );
109      } // end method getStreams
110
111      // process connection with client
112      private void processConnection() throws IOException
113      {
114         String message = "Connection successful";
115         sendData( message ); // send connection successful message
116
117         // enable enterField so server user can send messages
118         setTextFieldEditable( true );
119
120         do // process messages sent from client
121         {
122            try // read message and display it
123            {
124               message = ( String ) input.readObject(); // read new message
125               displayMessage( "\n" + message ); // display message
126            } // end try
127            catch ( ClassNotFoundException classNotFoundException )
128            {
129               displayMessage( "\nUnknown object type received" );
130            } // end catch
131
```

Obtain `Socket`'s `InputStream` and use it to initialize `ObjectInputStream`

(5 of 8)

Line 106

Line 124

Use `ObjectInputStream` method `readObject` to read a `String` from client

```
132       } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
133    } // end method processConnection
134
135    // close streams and socket
136    private void closeConnection()
137    {
138       displayMessage( "\nTerminating connection\n" );
139       setTextFieldEditable( false ); // disable enterField
140
141       try
142       {
143          output.close(); // close output stream
144          input.close(); // close input stream
145          connection.close(); // close socket
146       } // end try
147       catch ( IOException ioException )
148       {
149          ioException.printStackTrace();
150       } // end catch
151    } // end method closeConnection
153    // send message to client
154    private void sendData( String message )
155    {
156       try // send object to client
157       {
158          output.writeObject( "SERVER>>> " + message );
159          output.flush(); // flush output to client
160          displayMessage( "\nSERVER>>> " + message );
161       } // end try
```

Method `closeConnection` closes streams and sockets

Invoke `Socket` method `close` to close the socket

Use `ObjectOutputStream` method `writeObject` to send a `String` to client

Outline

Server.java

(6 of 8)

Lines 136-151

Line 145

Outline

```java
      catch ( IOException ioException )
      {
         displayArea.append( "\nError writing object" );
      } // end catch
   } // end method sendData

   // manipulates displayArea in the event-dispatch thread
   private void displayMessage( final String messageToDisplay )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // updates displayArea
            {
               displayArea.append( messageToDisplay ); // append message
            } // end method run
         } // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

```

Server.java

(7 of 8)

```java
   // manipulates enterField in the event-dispatch thread
   private void setTextFieldEditable( final boolean editable )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // sets enterField's editability
            {
               enterField.setEditable( editable );
            } // end method run
         }  // end inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method setTextFieldEditable
} // end class Server
```

Outline

Server.java

(8 of 8)

```
// Fig. 24.6: ServerTest.java
// Test the Server application.
import javax.swing.JFrame;

public class ServerTest
{
   public static void main( String args[] )
   {
      Server application = new Server(); // create server
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.runServer(); // run server application
   } // end main
} // end class ServerTest
```

ServerTest.java

Common Programming Error 24.1

Specifying a port that is already in use or specifying an invalid port number when creating a `ServerSocket` results in a `BindException`.

Software Engineering Observation 24.5

When using an `ObjectOutputStream` and `ObjectInputStream` to send and receive data over a network connection, always create the `ObjectOutputStream` first and `flush` the stream so that the client's `ObjectInputStream` can prepare to receive the data. This is required only for networking applications that communicate using `ObjectOutputStream` and `ObjectInputStream`.

Performance Tip 24.3

A computer’s input and output components are typically much slower than its memory. Output buffers typically are used to increase the efficiency of an application by sending larger amounts of data fewer times, thus reducing the number of times an application accesses the computer’s input and output components.

```
// Fig. 24.7: Client.java
// Client that reads and displays information sent from a Server.
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class Client extends JFrame
{
   private JTextField enterField; // enters information from user
   private JTextArea displayArea; // display information to user
   private ObjectOutputStream output; // output stream to server
   private ObjectInputStream input; // input stream from server
   private String message = ""; // message from server
   private String chatServer; // host server for this application
   private Socket client; // socket to communicate with server

```

Outline

Client.java

(1 of 7)

Outline

```
   // initialize chatServer and set up GUI
   public Client( String host )
   {
      super( "Client" );

      chatServer = host; // set server to which this client connects

      enterField = new JTextField(); // create enterField
      enterField.setEditable( false );
      enterField.addActionListener(
         new ActionListener()
         {
            // send message to server
            public void actionPerformed( ActionEvent event )
            {
               sendData( event.getActionCommand() );
               enterField.setText( "" );
            } // end method actionPerformed
         } // end anonymous inner class
      ); // end call to addActionListener

      add( enterField, BorderLayout.NORTH );

      displayArea = new JTextArea(); // create displayArea
      add( new JScrollPane( displayArea ), BorderLayout.CENTER );

      setSize( 300, 150 ); // set size of window
      setVisible( true ); // show window
   } // end Client constructor

```

Client.java

(2 of 7)

```
   // connect to server and process messages from server
   public void runClient()
   {
      try // connect to server, get streams, process connection
      {
         connectToServer(); // create a Socket to make connection
         getStreams(); // get the input and output streams
         processConnection(); // process connection
      } // end try
      catch ( EOFException eofException )
      {
         displayMessage( "\nClient terminated connection" );
      } // end catch
      catch ( IOException ioException )
      {
         ioException.printStackTrace();
      } // end catch
      finally
      {
         closeConnection(); // close connection
      } // end finally
   } // end method runClient

   // connect to server
   private void connectToServer() throws IOException
   {
      displayMessage( "Attempting connection\n" );

```

Outline

Client.java

(3 of 7)

```
86       // create Socket to make connection to server
87       client = new Socket( InetAddress.getByName( chatServer ), 12345 );
88 
89       // display connection information
90       displayMessage( "Connected to: " +
91          client.getInetAddress().getHostName() );
92    } // end method connectToServer
93 
94    // get streams to send and recei
95    private void getStreams() throws
96    {
97       // set up output stream for objects
98       output = new ObjectOutputStream( client.getOutputStream() );
99       output.flush(); // flush output buffer to send header information
100 
101      // set up input stream for objects
102      input = new ObjectInputStream( client.getInputStream() );
103 
104      displayMessage( "\nGot I/O streams\n" );
105   } // end method getStreams
106 
107   // process connection with server
108   private void processConnection() throws IOException
109   {
110      // enable enterField so client user can send messages
111      setTextFieldEditable( true );
112 
```

Outline

Crea
will
12

Use InetAddress static method getByName to obtain an InetAdress object containing the IP address spec

Display a message
f the
hich
the client has connected

Obtain Socket's OutputStream and use it to initialize ObjectOutputStream

Method flu
and sends header information

Lines 90-91

Line 98

Line 99

```
113      do // process messages sent from server
114      {
115         try // read message and display it
116         {
117            message = ( String ) input.readObject(); // read new message
118            displayMessage( "\n" + message ); // display message
119         } // end try
120         catch ( ClassNotFoundException classNotFoundException )
121         {
122            displayMessage( "\nUnknown object type received" );
123         } // end catch
124
125      } while ( !message.equals( "SERVER>>> TERMINATE" ) );
126   } // end method processConnection
127
128   // close streams and socket
129   private void closeConnection()
130   {
131      displayMessage( "\nClosing connection" );
132      setTextFieldEditable( false ); // disable enterField
133
134      try
135      {
136         output.close(); // close output stream
137         input.close(); // close input stream  1
138         client.close(); // close socket
139      } // end try
```

Read a String object from server

Invoke Socket method close to close the socket

Outline

Client.java

Line 117

Line 138

```
140        catch ( IOException ioException )
141        {
142           ioException.printStackTrace();
143        } // end catch
144     } // end method closeConnection
145
146     // send message to server
147     private void sendData( String message )
148     {
149        try // send object to server
150        {
151           output.writeObject( "CLIENT>>> " + message );
152           output.flush(); // flush data to output
153           displayMessage( "\nCLIENT>>> " + message );
154        } // end try
155        catch ( IOException ioException )
156        {
157           displayArea.append( "\nError writing object" );
158        } // end catch
159     } // end method sendData
160
```

Use `ObjectOutputStream` method `writeObject` to send a `String` to server

Outline

Client.java

Line 151

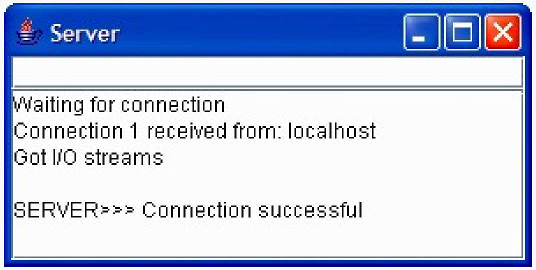
Outline

```
   // manipulates displayArea in the event-dispatch thread
   private void displayMessage( final String messageToDisplay )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // updates displayArea
            {
               displayArea.append( messageToDisplay );
            } // end method run
         }  // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

   // manipulates enterField in the event-dispatch thread
   private void setTextFieldEditable( final boolean editable )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // sets enterField's editability
            {
               enterField.setEditable( editable );
            } // end method run
         } // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method setTextFieldEditable
} // end class Client
```

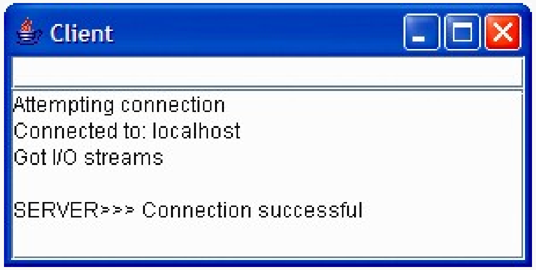
Client.java

(7 of 7)

```
// Fig. 24.8: ClientTest.java
// Test the Client class.
import javax.swing.JFrame;

public class ClientTest
{
   public static void main( String args[] )
   {
      Client application; // declare client application

      // if no command line args
      if ( args.length == 0 )
         application = new Client( "127.0.0.1" ); // connect to localhost
      else
         application = new Client( args[ 0 ] ); // use args to connect

      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.runClient(); // run client application
   } // end main
} // end class ClientTest
```



```
Waiting for connection
Connection 1 received from: localhost
Got I/O streams

SERVER>>> Connection successful
```



```
Attempting connection
Connected to: localhost
Got I/O streams

SERVER>>> Connection successful
```

ClientTest.java

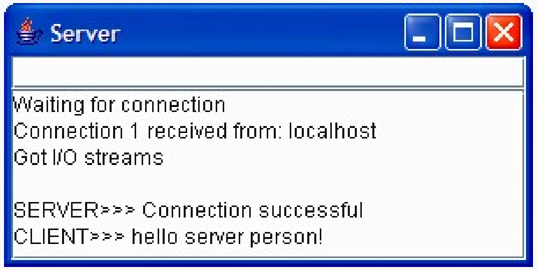
(1 of 2)

Program output

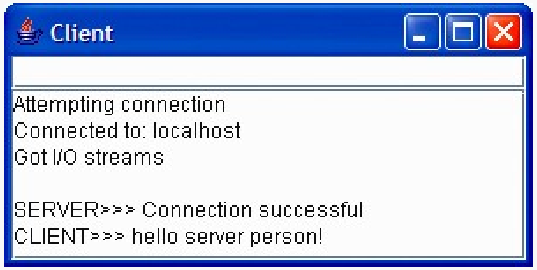
`ClientTest.java`

(2 of 2)

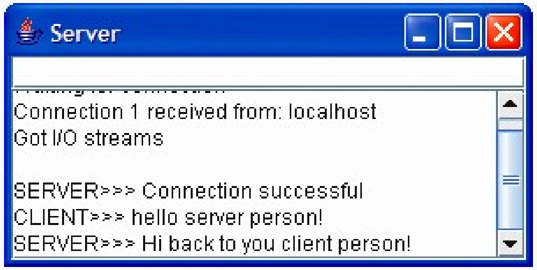
Program output



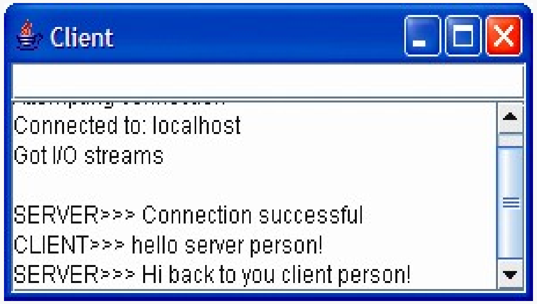
```
Waiting for connection
Connection 1 received from: localhost
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> hello server person!
```



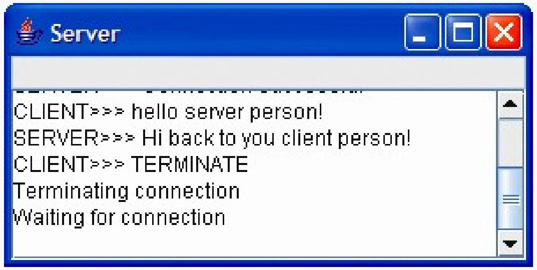
```
Attempting connection
Connected to: localhost
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> hello server person!
```



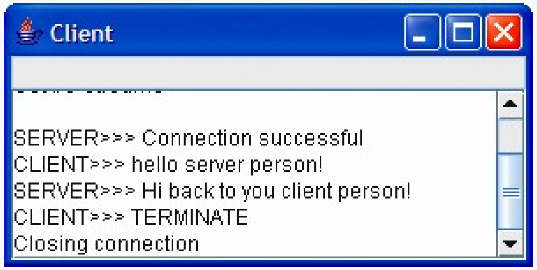
```
Connection 1 received from: localhost
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> hello server person!
SERVER>>> Hi back to you client person!
```



```
Connected to: localhost
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> hello server person!
SERVER>>> Hi back to you client person!
```



```
CLIENT>>> hello server person!
SERVER>>> Hi back to you client person!
CLIENT>>> TERMINATE
Terminating connection
Waiting for connection
```



```

SERVER>>> Connection successful
CLIENT>>> hello server person!
SERVER>>> Hi back to you client person!
CLIENT>>> TERMINATE
Closing connection
```

24.7 Connectionless Client/Server Interaction with Datagrams

- Connectionless transmission with datagrams
  - No connection maintained with other computer
  - Break message into separate pieces and send as packets
  - Message arrive in order, out of order or not at all
  - Receiver puts messages in order and reads them

```
 1 // Fig. 24.9: Server.java
 2 // Server that receives and sends packets from/to a client.
 3 import java.io.IOException;
 4 import java.net.DatagramPacket;
 5 import java.net.DatagramSocket;
 6 import java.net.SocketException;
 7 import java.awt.BorderLayout;
 8 import javax.swing.JFrame;
 9 import javax.swing.JScrollPane;
10 import javax.swing.JTextArea;
11 import javax.swing.SwingUtilities;
12
13 public class Server extends JFrame
14 {
15    private JTextArea displayArea; // displays packets received
16    private DatagramSocket socket; // socket to connect to client
17
18    // set up GUI and DatagramSocket
19    public Server()
20    {
21       super( "Server" );
22
23       displayArea = new JTextArea(); // create displayArea
24       add( new JScrollPane( displayArea ), BorderLayout.CENTER );
25       setSize( 400, 300 ); // set size of window
26       setVisible( true ); // show window
27
```

`Server.java`

(1 of 4)

Line 16

Use a `DatagramSocket` as our server

```
28       try // create DatagramSocket for sending and receiving packets
29       {
30          socket = new DatagramSocket( 5000 );
31       } // end try
32       catch ( SocketException socketException )
33       {
34          socketException.printStackTrace();
35          System.exit( 1 );
36       } // end catch
37    } // end Server constructor
38
39    // wait for packets to arrive, display data and echo packet to client
40    public void waitForPackets()
41    {
42       while ( true )
43       {
44          try // receive packet, display contents, return copy to client
45          {
46             byte data[] = new byte[ 100 ]; // set up packet
47             DatagramPacket receivePacket =
48                new DatagramPacket( data, data.length );
49
50             socket.receive( receivePacket ); // wait to receive packet
51
```

Use the DatagramSocket constructor that takes an integer port number argument to bind the server to a port where it can receive packets from clients

Create a DatagramPacket in which a received packet of information can be stored

Use DatagramSocket method receive to wait for a packet to arrive at the server

(2 of 4)

Line 30

Lines 47-48

Line 50

```
52            // display information from received packet
53            displayMessage( "\nPacket received:" +
54               "\nFrom host: " + receivePacket.getAddress() +
55               "\nHost port: " + receivePacket.getPort() +
56               "\nLength: " + receivePacket.getLength() +
57               "\nContaining:\n\t" + new String( receivePacket.get
58                  0, receivePacket.getLength() ) );
59
60            sendPacketToClient( receivePacket ); // send packet to client
61         } // end try
62         catch ( IOException ioException )
63         {
64            displayMessage( ioException.toString() + "\n" );
65            ioException.printStackTrace();
66         } // end catch
67      } // end while
68   } // end method waitForPackets
69
70   // echo packet to client
71   private void sendPacketToClient( DatagramPacket receivePacket )
72      throws IOException
73   {
74      displayMessage( "\n\nEchoing data to client..." )
75
76      // create packet to send
77      DatagramPacket sendPacket = new DatagramPacket(
78         receivePacket.getData(), receivePacket.getLength(),
79         receivePacket.getAddress(), receivePacket.getPort() );
80
```

Use DatagramPacket

Use DatagramPacket

Use DatagramPacket method getLength to obtain the number of bytes of data sent

(3 of 4)

Line 54

Use DatagramPacket method getData to obtain an byte array containing the data

Line 56

Line 57

Lines 77-79

Create a DatagramPacket, which specifies the data to send, the number of bytes to send, the client computer's Internet address and the port where the client is waiting to receive packets

```
      socket.send( sendPacket ); // send packet to client
      displayMessage( "Packet sent\n" );
   } // end method sendPacketToClient

   // manipulates displayArea in the event-dispat
   private void displayMessage( final String messageToDisplay )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // updates displayArea
            {
               displayArea.append( messageToDisplay ); // display message
            } // end method run
         } // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage
} // end class Server
```

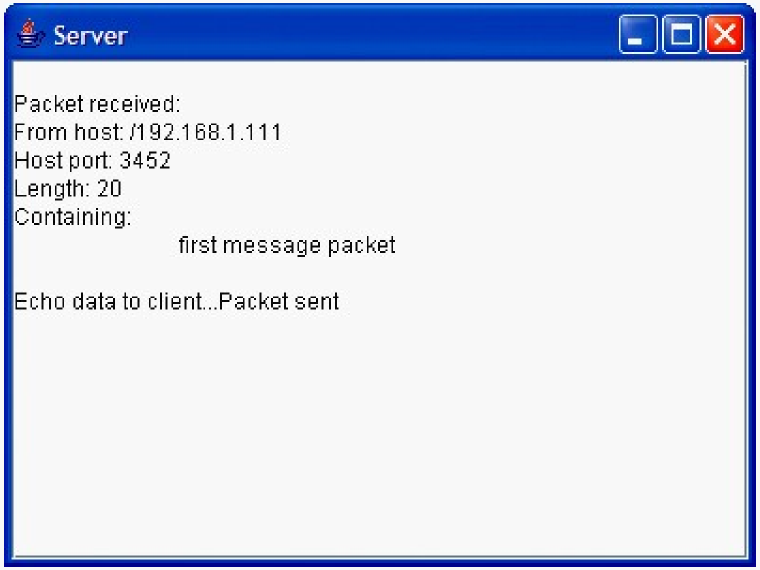
Use method send of DatagramSocket to send the packet over the network

Outline

Server.java

(4 of 4)

```java
// Fig. 24.10: ServerTest.java
// Tests the Server class.
import javax.swing.JFrame;

public class ServerTest
{
   public static void main( String args[] )
   {
      Server application = new Server(); // create server
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.waitForPackets(); // run server application
   } // end main
} // end class ServerTest
```



```
Packet received:
From host: /192.168.1.111
Host port: 3452
Length: 20
Containing:
                  first message packet

Echo data to client...Packet sent
```

Server window after packet of data is received from **client**

ServerTest.java

Program ouptut

Outline

Client.java

(1 of 5)

```java
// Fig. 24.11: Client.java
// Client that sends and receives packets to/from a server.
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class Client extends JFrame
{
   private JTextField enterField; // for entering messages
   private JTextArea displayArea; // for displaying messages
   private DatagramSocket socket; // socket to connect to server

   // set up GUI and DatagramSocket
   public Client()
   {
      super( "Client" );

```

```
28       enterField = new JTextField( "Type message here" );
29       enterField.addActionListener(
30          new ActionListener()
31          {
32             public void actionPerformed( ActionEvent event )
33             {
34                try // create and send packet
35                {
36                   // get message from textfield
37                   String message = event.g
38                   displayArea.append( "\nS
39                      message + "\n" );
40
41                   byte data[] = message.ge
42
43                   // create sendPacket
44                   DatagramPacket sendPacket = new DatagramPacket( data
45                      data.length, InetAddress.getLocalHost(), 5000 );
46
47                   socket.send( sendPacket ); // send packet
48                   displayArea.append( "Packet sent\n" );
49                   displayArea.setCaretPosition(
50                      displayArea.getText().length() );
51                } // end try
```

Create a DatagramPacket and initialize it with the byte array, the length of the string that was entered by the user, the IP address to which the packet is to be sent and the port number at which the server is waiting

Use DatagramPacket method send to send the packet

Outline

Client.java

(2 of 5)

Line 41

Outline

Client.java

(3 of 5)

Line 71

```
52                catch ( IOException ioException )
53                {
54                   displayMessage( ioException.toString() + "\n" );
55                   ioException.printStackTrace();
56                } // end catch
57             } // end actionPerformed
58          } // end inner class
59       ); // end call to addActionListener
60 
61       add( enterField, BorderLayout.NORTH );
62 
63       displayArea = new JTextArea();
64       add( new JScrollPane( displayArea ), BorderLayout.CENTER );
65 
66       setSize( 400, 300 ); // set window size
67       setVisible( true ); // show window
68 
69       try // create DatagramSocket for sending and receiving packets
70       {
71          socket = new DatagramSocket();
72       } // end try
73       catch ( SocketException socketException )
74       {
75          socketException.printStackTrace();
76          System.exit( 1 );
77       } // end catch
78    } // end Client constructor
79 
```

Create a `DatagramSocket` for sending and receiving packets

Outline

Client.java

Line 96

Line 97

```
80    // wait for packets to arrive from Server, display packet contents
81    public void waitForPackets()
82    {
83       while ( true )
84       {
85          try // receive packet and display contents
86          {
87             byte data[] = new byte[ 100 ]; // set up packet
88             DatagramPacket receivePacket = new DatagramPacket(
89                data, data.length );
90
91             socket.receive( receivePacket ); // wait for packet
92
93             // display packet contents
94             displayMessage( "\nPacket received:" +
95                "\nFrom host: " + receivePacket.getAddress() +
96                "\nHost port: " + receivePacket.getPort() +
97                "\nLength: " + receivePacket.getLength() +
98                "\nContaining:\n\t" + new String( receivePacket.get
99                   0, receivePacket.getLength() ) );
100         } // end try
101         catch ( IOException exception )
102         {
103            displayMessage( exception.toString() + "\
104            exception.printStackTrace();
105         } // end catch
106      } // end while
107   } // end method waitForPackets
108
```

Create a DatagramPacket to store received information

in

er

nt

Use DatagramPacket

Use DatagramPacket method getLength to obtain the number of bytes of data sent

Use DatagramPacket method getData to obtain an byte array containing the data

```
   // manipulates displayArea in the event-dispatch thread
   private void displayMessage( final String messageToDisplay )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // updates displayArea
            {
               displayArea.append( messageToDisplay );
            } // end method run
         }  // end inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage
}  // end class Client
```

Outline

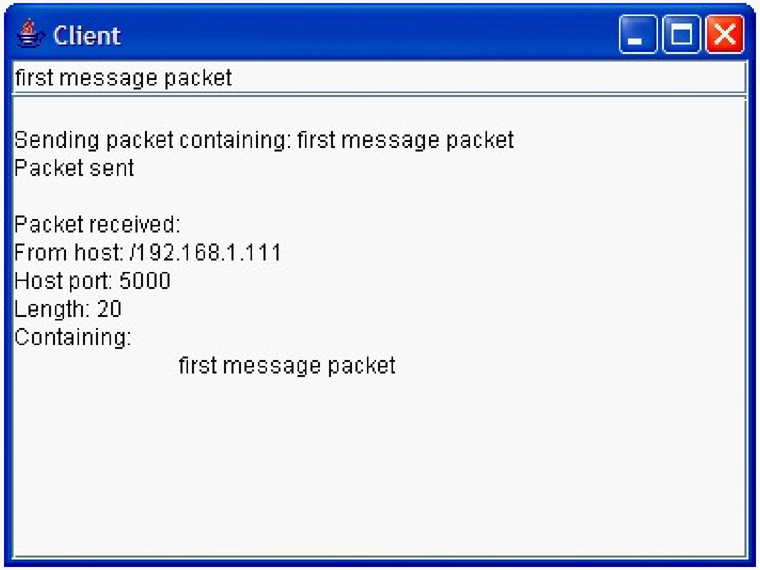
Client.java

(5 of 5)

```java
// Fig. 24.12: ClientTest.java
// Tests the Client class.
import javax.swing.JFrame;

public class ClientTest
{
   public static void main( String args[] )
   {
      Client application = new Client(); // create client
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.waitForPackets(); // run client application
   } // end main
}  // end class ClientTest
```

ClientTest.java

Program output



```
first message packet

Sending packet containing: first message packet
Packet sent

Packet received:
From host: /192.168.1.111
Host port: 5000
Length: 20
Containing:
        first message packet
```

Client window after sending packet to Server and receiving packet back from **Server**

Common Programming Error 24.2

Specifying a port that is already in use or specifying an invalid port number when creating a `DatagramSocket` results in a `SocketException`.

24.8 Client/Server Tic-Tac-Toe Using a Multithreaded Server

- **Multiple threads**
  - **Server uses one thread per player**
    - **Allow each player to play game independently**

```java
// Fig. 24.13: TicTacToeServer.java
// This class maintains a game of Tic-Tac-Toe for two clients.
import java.awt.BorderLayout;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
import java.util.Formatter;
import java.util.Scanner;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Condition;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;

```

Outline

TicTacToeServer.java

(1 of 12)

```java
public class TicTacToeServer extends JFrame
{
   private String[] board = new String[ 9 ]; // tic-tac-toe board
   private JTextArea outputArea; // for outputting moves
   private Player[] players; // array of Players
   private ServerSocket server; // server socket to connect with clients
   private int currentPlayer; // keeps track of player with current move
   private final static int PLAYER_X = 0; // constant for first player
   private final static int PLAYER_O = 1; // constant for second player
   private final static String[] MARKS = { "X", "O" }; // array of marks
   private ExecutorService runGame; // will run players
   private Lock gameLock; // to lock game for synchronization
   private Condition otherPlayerConnected; // to wait for other player
   private Condition otherPlayerTurn; // to wait for other player's turn

   // set up tic-tac-toe server and GUI that displays messages
   public TicTacToeServer()
   {
      super( "Tic-Tac-Toe Server" ); // set title of window

      // create ExecutorService with a thread for each player
      runGame = Executors.newFixedThreadPool( 2 );
      gameLock = new ReentrantLock(); // create lock for game

      // condition variable for both players being connected
      otherPlayerConnected = gameLock.newCondition();

      // condition variable for the other player's turn
      otherPlayerTurn = gameLock.newCondition();

```

Outline

TicTacToeServer .java

(2 of 12)

```
48       for ( int i = 0; i < 9; i++ )
49          board[ i ] = new String( "" ); // create tic-tac-toe board
50       players = new Player[ 2 ]; // create array of players
51       currentPlayer = PLAYER_X; // set current player to first player
52
53       try
54       {
55          server = new ServerSocket( 12345, 2 ); // set up ServerSocket
56       } // end try
57       catch ( IOException ioException )
58       {
59          ioException.printStackTrace();
60          System.exit( 1 );
61       } // end catch
62
63       outputArea = new JTextArea(); // create JTextArea for output
64       add( outputArea, BorderLayout.CENTER );
65       outputArea.setText( "Server awaiting connections\n" );
66
67       setSize( 300, 300 ); // set size of window
68       setVisible( true ); // show window
69    } // end TicTacToeServer constructor
70
```

Create array `players` with 2 elements

Create `ServerSocket` to listen on port 12345

`TicTacToeServer.java`

Line 55

Outline

```
71    // wait for two connections so game can be played
72    public void execute()
73    {
74       // wait for each client to connect
75       for ( int i = 0; i < players.length; i++ )
76       {
77          try // wait for connection, create Player, start runnable
78          {
79             players[ i ] = new Player( server.accept(), i );
80             runGame.execute( players[ i ] ); // execute player runnable
81          } // end try
82          catch ( IOException ioException )
83          {
84             ioException.printStackTrace();
85             System.exit( 1 );
86          } // end catch
87       } // end for
88
89       gameLock.lock(); // lock game to signal player X's thread
90
```

Loop twice, blocking at line 79 each time while waiting for a client connection

Create a new `Player` object to manage the connection as eparate thread

Execute the Player in the `runGame` thread pool

er

ines 75-87

Line 79

Line 80

Outline

```
      try
      {
         players[ PLAYER_X ].setSuspended( false ); // resume player X
         otherPlayerConnected.signal(); // wake up player X's thread
      } // end try
      finally
      {
         gameLock.unlock(); // unlock game after signalling player X
      } // end finally
   } // end method execute

   // display message in outputArea
   private void displayMessage( final String messageToDisplay )
   {
      // display message from event-dispatch thread of execution
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run() // updates outputArea
            {
               outputArea.append( messageToDisplay ); // add message
            } // end  method run
         } // end inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

```

TicTacToeServer .java

(5 of 12)

```java
   // determine if move is valid
   public boolean validateAndMove( int location, int player )
   {
      // while not current player, must wait for turn
      while ( player != currentPlayer )
      {
         gameLock.lock(); // lock game to wait for other player to go

         try
         {
            otherPlayerTurn.await(); // wait for player's turn
         } // end try
         catch ( InterruptedException exception )
         {
            exception.printStackTrace();
         } // end catch
         finally
         {
            gameLock.unlock(); // unlock game after waiting
         } // end finally
      } // end while

      // if location not occupied, make move
      if ( !isOccupied( location ) )
      {
         board[ location ] = MARKS[ currentPlayer ]; // set move on board
         currentPlayer = ( currentPlayer + 1 ) % 2; // change player

```

Outline

TicTacToeServer.java

(6 of 12)

Outline

```
         // let new current player know that move occurred
         players[ currentPlayer ].otherPlayerMoved( location );

         gameLock.lock(); // lock game to signal other player to go

         try
         {
            otherPlayerTurn.signal(); // signal other player to continue
         } // end try
         finally
         {
            gameLock.unlock(); // unlock game after signaling
         } // end finally

         return true; // notify player that move was valid
      } // end if
      else // move was not valid
         return false; // notify player that move was invalid
   } // end method validateAndMove

   // determine whether location is occupied
   public boolean isOccupied( int location )
   {
      if ( board[ location ].equals( MARKS[ PLAYER_X ] ) ||
         board [ location ].equals( MARKS[ PLAYER_O ] ) )
         return true; // location is occupied
      else
         return false; // location is not occupied
   } // end method isOccupied

```

TicTacToeServer .java

(7 of 12)

```
175   // place code in this method to determine whether game over
176   public boolean isGameOver()
177   {
178      return false; // this is left as an exercise
179   } // end method isGameOver
180
181   // private inner class Player manages each Player as a runnable
182   private class Player implements Runnable
183   {
184      private Socket connection; // connection to client
185      private Scanner input; // input from client
186      private Formatter output; // output to client
187      private int playerNumber; // tracks which player this is
188      private String mark; // mark for this player
189      private boolean suspended = true; // whether thread is suspended
190
191      // set up Player thread
192      public Player( Socket socket, int number )
193      {
194         playerNumber = number; // store this player's number
195         mark = MARKS[ playerNumber ]; // specify player's mark
196         connection = socket; // store socket for client
197
198         try // obtain streams from Socket
199         {
200            input = new Scanner( connection.getInputStream() );
201            output = new Formatter( connection.getOutputStream() );
202         } // end try
```

Get the streams to send and receive data

TicTacToeServer.java

(8 of 12)

Lines 200-201

```
203            catch ( IOException ioException )
204            {
205               ioException.printStackTrace();
206               System.exit( 1 );
207            } // end catch
208         } // end Player constructor
209
210         // send message that other player moved
211         public void otherPlayerMoved( int location )
212         {
213            output.format( "Opponent moved\n" );
214            output.format( "%d\n", location ); // send location of
215            output.flush(); // flush output
216         } // end method otherPlayerMoved
217
218         // control thread's execution
219         public void run()
220         {
221            // send client its mark (X or O), process messages from client
222            try
223            {
224               displayMessage( "Player " + mark + " connected\n" );
225               output.format( "%s\n", mark ); // send player's mark
226               output.flush(); // flush output
227
```

Format output notifying the
the move

Call `Formatter` method `flush` to force the output to the client

Send player’s mark

Outline

`TicTacToeServer.java`

(9 of 12)

Lines 225-226

```
228          // if player X, wait for another player to arrive
229          if ( playerNumber == PLAYER_X )
230          {
231             output.format( "%s\n%s", "Player X connected",
232                "Waiting for another player\n" );
233             output.flush(); // flush output
234
235             gameLock.lock(); // lock game to  wait for second player
236
237             try
238             {
239                while( suspended )
240                {
241                   otherPlayerConnected.await(); // wait for player O
242                } // end while
243             } // end try
244             catch ( InterruptedException exception )
245             {
246                exception.printStackTrace();
247             } // end catch
248             finally
249             {
250                gameLock.unlock(); // unlock game after second player
251             } // end finally
252
253             // send message that other player connected
254             output.format( "Other player connected. Your move.\n" );
255             output.flush(); // flush output
256          } // end if
```

Send message indicating one player connected and waiting for another player to arrive

Begin the game

Outline

er

(10 of 12)

Lines 231-233

Lines 254-255

```
257            else
258            {
259               output.format( "Player O connected, please wait\n" );
260               output.flush(); // flush output
261            } // end else
262
263            // while game not over
264            while ( !isGameOver() )
265            {
266               int location = 0; // initialize move location
267
268               if ( input.hasNext() )
269                  location = input.nextInt(); // get move location
270
271               // check for valid move
272               if ( validateAndMove( location, playerNumber ) )
273               {
274                  displayMessage( "\nlocation: " + location );
275                  output.format( "Valid move.\n" ); // notify client
276                  output.flush(); // flush output
277               } // end if
```

Send message indicating player O connected

Read a move

Check the move

Send message indicating the move is valid

Outline

TicTacToeServer.java

(11 of 12)

Lines 259-260

Line 269

Lines 275-276

```java
278               else // move was invalid
279               {
280                  output.format( "Invalid move, try again\n" );
281                  output.flush(); // flush output
282               } // end else
283            } // end while
284         } // end try
285         finally
286         {
287            try
288            {
289               connection.close(); // close connection to client
290            } // end try
291            catch ( IOException ioException )
292            {
293               ioException.printStackTrace();
294               System.exit( 1 );
295            } // end catch
296         } // end finally
297      } // end method run
298
299      // set whether or not thread is suspended
300      public void setSuspended( boolean status )
301      {
302         suspended = status; // set value of suspended
303      } // end method setSuspended
304   } // end class Player
305} // end class TicTacToeServer
```

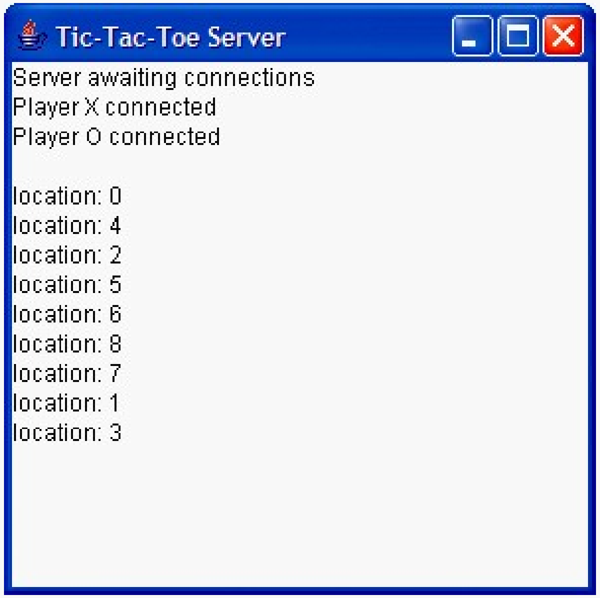
Send message indicating the move is invalid

TicTacToeServer.java

(12 of 12)

Lines 280-281

```
// Fig. 24.14: TicTacToeServerTest.java
// Tests the TicTacToeServer.
import javax.swing.JFrame;

public class TicTacToeServerTest
{
   public static void main( String args[] )
   {
      TicTacToeServer application = new TicTacToeServer();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.execute();
   } // end main
} // end class TicTacToeServerTest
```



```
Server awaiting connections
Player X connected
Player O connected

location: 0
location: 4
location: 2
location: 5
location: 6
location: 8
location: 7
location: 1
location: 3
```

Outline

TicTacToeServerTest.java

Program output

```java
// Fig. 24.15: TicTacToeClient.java
// Client that let a user play Tic-Tac-Toe with another across a network.
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.net.Socket;
import java.net.InetAddress;
import java.io.IOException;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import java.util.Formatter;
import java.util.Scanner;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

```

Outline

TicTacToeClient .java

(1 of 10)

Outline

TicTacToeClient .java

(2 of 10)

```java
public class TicTacToeClient extends JFrame implements Runnable
{
   private JTextField idField; // textfield to display player's mark
   private JTextArea displayArea; // JTextArea to display output
   private JPanel boardPanel; // panel for tic-tac-toe board
   private JPanel panel2; // panel to hold board
   private Square board[][]; // tic-tac-toe board
   private Square currentSquare; // current square
   private Socket connection; // connection to server
   private Scanner input; // input from server
   private Formatter output; // output to server
   private String ticTacToeHost; // host name for server
   private String myMark; // this client's mark
   private boolean myTurn; // determines which client's turn it is
   private final String X_MARK = "X"; // mark for first client
   private final String O_MARK = "O"; // mark for second client

   // set up user-interface and board
   public TicTacToeClient( String host )
   {
      ticTacToeHost = host; // set name of server
      displayArea = new JTextArea( 4, 30 ); // set up JTextArea
      displayArea.setEditable( false );
      add( new JScrollPane( displayArea ), BorderLayout.SOUTH );

      boardPanel = new JPanel(); // set up panel for squares in board
      boardPanel.setLayout( new GridLayout( 3, 3, 0, 0 ) );

      board = new Square[ 3 ][ 3 ]; // create board

      // loop over the rows in the board
      for ( int row = 0; row < board.length; row++ )
      {
         // loop over the columns in the board
         for ( int column = 0; column < board[ row ].length; column++ )
         {
            // create square
            board[ row ][ column ] = new Square( ' ', row * 3 + column );
            boardPanel.add( board[ row ][ column ] ); // add square
         } // end inner for
      } // end outer for

      idField = new JTextField(); // set up textfield
      idField.setEditable( false );
      add( idField, BorderLayout.NORTH );

      panel2 = new JPanel(); // set up panel to contain boardPanel
      panel2.add( boardPanel, BorderLayout.CENTER ); // add board panel
      add( panel2, BorderLayout.CENTER ); // add container panel

      setSize( 300, 225 ); // set size of window
      setVisible( true ); // show window

      startClient();
   } // end TicTacToeClient constructor

```

Outline

TicTacToeClient .java

(3 of 10)

```
79    // start the client thread
80    public void startClient()
81    {
82       try // connect to server, get streams and start outputThrea
83       {
84          // make connection to server
85          connection = new Socket(
86             InetAddress.getByName( ticTacToeHost ), 12345 );
87
88          // get streams for input and output
89          input = new Scanner( connection.getInputStream() );
90          output = new Formatter( connection.getOutputStream() );
91       } // end try
92       catch ( IOException ioException )
93       {
94          ioException.printStackTrace();
95       } // end catch
96
97       // create and start worker thread for this client
98       ExecutorService worker = Executors.newFixedThreadPool( 1 );
99       worker.execute( this ); // execute client
100   } // end method startClient
101
102   // control thread that allows continuous update of displayArea
103   public void run()
104   {
105      myMark = input.nextLine(); // get player's mark (X or O)
106
```

Outline

Connect to the server

TicTacToeClient .java

(4 of 10)

Get the streams to send and receive data

Lines 89-90

Line 105

Read mark character from server

Outline

```
107       SwingUtilities.invokeLater(
108          new Runnable()
109          {
110             public void run()
111             {
112                // display player's mark
113                idField.setText( "You are player \"" + myMark + "\"" );
114             } // end method run
115          } // end anonymous inner class
116       ); // end call to SwingUtilities.invokeLater
117 
118       myTurn = ( myMark.equals( X_MARK ) ); // determine if client's turn
119 
120       // receive messages sent to client and output them
121       while ( true )
122       {
123          if ( input.hasNextLine() )
124             processMessage( input.nextLine() );
125       } // end while
126    } // end method run
127 
128    // process messages received by client
129    private void processMessage( String message )
130    {
131       // valid move occurred
132       if ( message.equals( "Valid move." ) )
133       {
134          displayMessage( "Valid move, please wait.\n" );
135          setMark( currentSquare, myMark ); // set mark in square
136       } // end if
```

TicTacToeClient .java

(5 of 10)

Lines 121-125

Lines 123-124

Lines 132-136

Loop continually

Read and process messages from server

If valid move, write message and set mark in square

Outline

```
137      else if ( message.equals( "Invalid move, try again" ) )
138      {
139         displayMessage( message + "\n" ); // display invalid move
140         myTurn = true; // still this client's turn
141      } // end else if
142      else if ( message.equals( "Opponent moved" ) )
143      {
144         int location = input.nextInt(); // get move location
145         input.nextLine(); // skip newline after int location
146         int row = location / 3; // calculate row
147         int column = location % 3; // calculate column
148
149         setMark(  board[ row ][ column ],
150            ( myMark.equals( X_MARK ) ? O_MARK : X_MARK ) ); // mark move
151         displayMessage( "Opponent moved. Your turn.\n" );
152         myTurn = true; // now this client's turn
153      } // end else if
154      else
155         displayMessage( message + "\n" ); // display the message
156   } // end method processMessage
157
```

If opponent moves, set mark in square

ient
.java

(6 of 10)

Lines 137-141

Lines 142-153

Outline

```
   // manipulate outputArea in event-dispatch thread
   private void displayMessage( final String messageToDisplay )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run()
            {
               displayArea.append( messageToDisplay ); // updates output
            } // end method run
         }  // end inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

   // utility method to set mark on board in event-dispatch thread
   private void setMark( final Square squareToMark, final String mark )
   {
      SwingUtilities.invokeLater(
         new Runnable()
         {
            public void run()
            {
               squareToMark.setMark( mark ); // set mark in square
            } // end method run
         } // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method setMark

```

TicTacToeClient.java

(7 of 10)

Outline

```
186    // send message to server indicating clicked square
187    public void sendClickedSquare( int location )
188    {
189       // if it is my turn
190       if ( myTurn )
191       {
192          output.format( "%d\n", location ); // send location to server
193          output.flush();
194          myTurn = false; // not my turn anymore
195       } // end if
196    } // end method sendClickedSquare
197
198    // set current Square
199    public void setCurrentSquare( Square square )
200    {
201       currentSquare = square; // set current square to argument
202    } // end method setCurrentSquare
203
204    // private inner class for the squares on the board
205    private class Square extends JPanel
206    {
207       private String mark; // mark to be drawn in this square
208       private int location; // location of square
209
210       public Square( String squareMark, int squareLocation )
211       {
212          mark = squareMark; // set mark for this square
213          location = squareLocation; // set location of this square
214
```

TicTacToeClient.java

Send the move to the server

Lines 192-193

```
         addMouseListener( 
            new MouseAdapter() 
            {
               public void mouseReleased( MouseEvent e )
               {
                  setCurrentSquare( Square.this ); // set current square

                  // send location of this square
                  sendClickedSquare( getSquareLocation() );
               } // end method mouseReleased
            } // end anonymous inner class
         ); // end call to addMouseListener
      } // end Square constructor

      // return preferred size of Square
      public Dimension getPreferredSize() 
      { 
         return new Dimension( 30, 30 ); // return preferred size
      } // end method getPreferredSize

      // return minimum size of Square
      public Dimension getMinimumSize() 
      {
         return getPreferredSize(); // return preferred size
      } // end method getMinimumSize

```

Outline

TicTacToeClient .java

(9 of 10)

Outline

TicTacToeClient.java

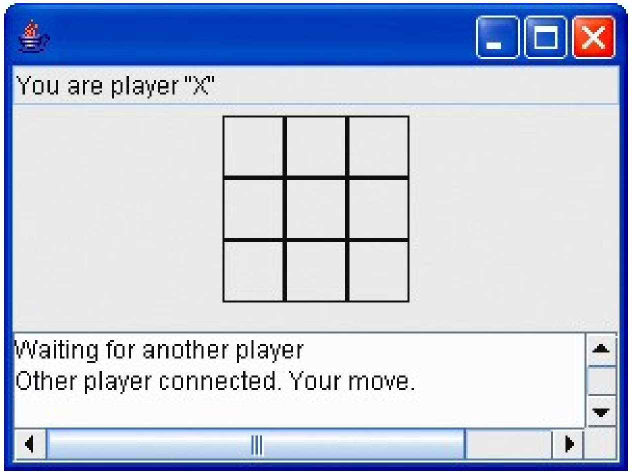
(10 of 10)

```
      // set mark for Square
      public void setMark( String newMark )
      {
         mark = newMark; // set mark of square
         repaint(); // repaint square
      } // end method setMark

      // return Square location
      public int getSquareLocation()
      {
         return location; // return location of square
      } // end method getSquareLocation

      // draw Square
      public void paintComponent( Graphics g )
      {
         super.paintComponent( g );

         g.drawRect( 0, 0, 29, 29 ); // draw square
         g.drawString( mark, 11, 20 ); // draw mark
      } // end method paintComponent
   } // end inner-class Square
} // end class TicTacToeClient
```

```java
// Fig. 24.16: TicTacToeClientTest.java
// Tests the TicTacToeClient class.
import javax.swing.JFrame;

public class TicTacToeClientTest
{
   public static void main( String args[] )
   {
      TicTacToeClient application; // declare client application

      // if no command line args
      if ( args.length == 0 )
         application = new TicTacToeClient( "127.0.0.1" ); // localhost
      else
         application = new TicTacToeClient( args[ 0 ] ); // use args

      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   } // end main
} // end class TicTacToeClientTest
```

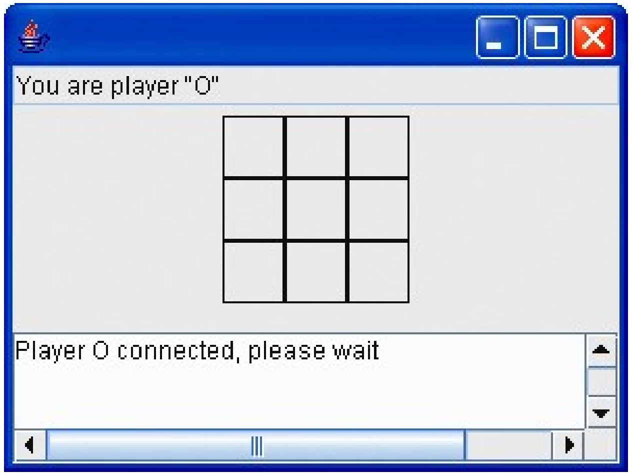
Outline

TicTacToeClientTest.java



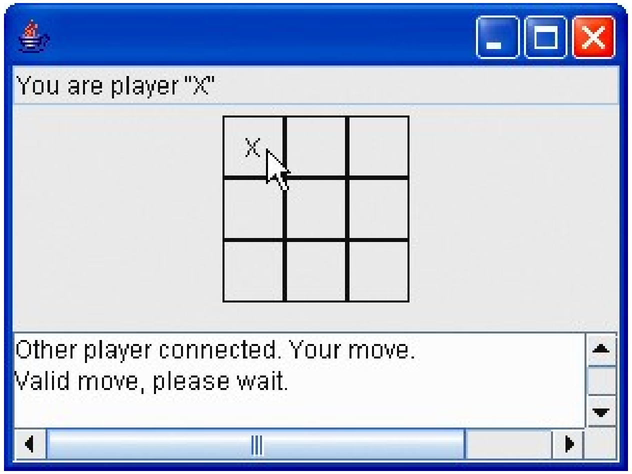
You are player "X"

```
Waiting for another player
Other player connected. Your move.
```



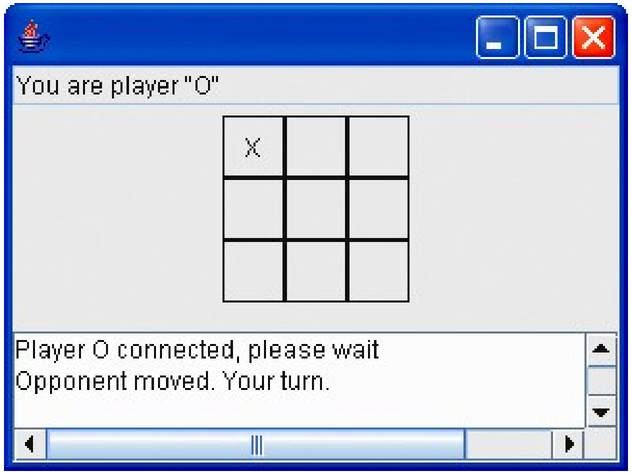
You are player "O"

```
Player O connected, please wait
```



You are player "X"

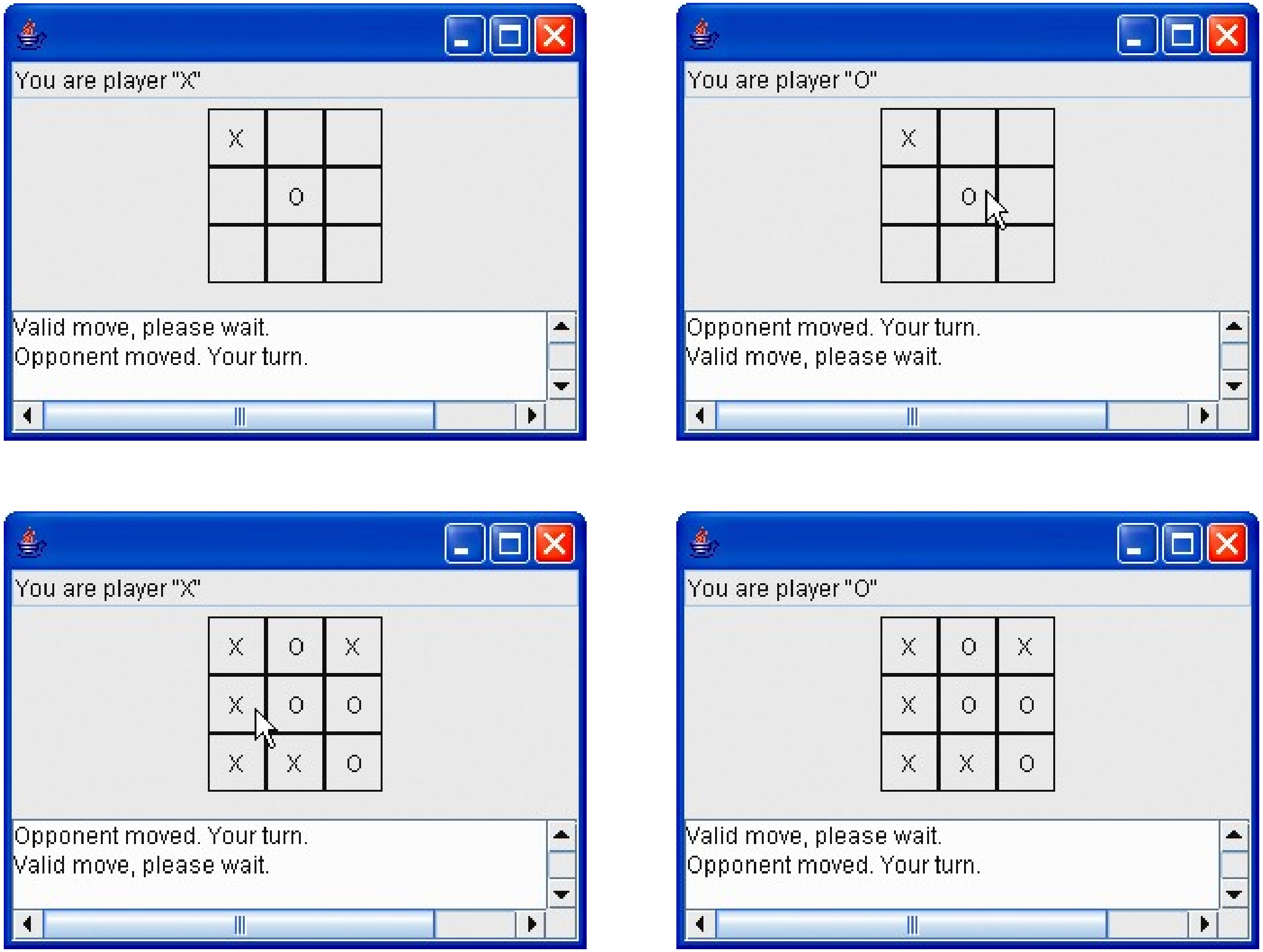
```
Other player connected. Your move.
Valid move, please wait.
```



You are player "O"

```
Player O connected, please wait
Opponent moved. Your turn.
```

Fig.24.17 | Sample outputs from the client/server Tic-Tac-Toe program. (Part 1 of 2.)



You are player "X"

X		
	O	

```
Valid move, please wait.
Opponent moved. Your turn.
```

You are player "O"

X		
	O	

```
Opponent moved. Your turn.
Valid move, please wait.
```

You are player "X"

X	O	X
X	O	O
X	X	O

```
Opponent moved. Your turn.
Valid move, please wait.
```

You are player "O"

X	O	X
X	O	O
X	X	O

```
Valid move, please wait.
Opponent moved. Your turn.
```

Fig.24.17 | **Sample outputs from the client/server Tic-Tac-Toe program. (Part 2 of 2.)**

24.9 Security and the Network

- By default, applets cannot perform file processing
- Applets often limited in machine access
  - Applets can communicate only with the machine from which it was originally downloaded
- Java Security API
  - Digitally signed applets
  - Applets given more privileges if from trusted source

24.10 Case Study: DeitelMessenger Server and Client

- Chat rooms
  - Each user can post a message and read all other messages
  - Multicast
    - Send packets to groups of clients

24.10.1 `DeitelMessengerServer` and Supporting Classes

- **`DeitelMessengerServer`**
  - Online chat system
  - Classes:
    - `DeitelMessengerServer`
      - Clients connect to this server
    - Interface `SocketMessengerConstants`
      - Defines constants for port numbers
    - Interface `MessageListener`
      - Defines method for receiving new chat messages
    - Class `MessageReceiver`
      - Separate thread listens for messages from clients
    - Class `MulticastSender`
      - Separate thread delivers outgoing messages to clients

Outline

```
1  // Fig. 24.18: DeitelMessengerServer.java
2  // DeitelMessengerServer is a multi-threaded, socket- and
3  // packet-based chat server.
4  package com.deitel.messenger.sockets.server;
5
6  import java.net.ServerSocket;
7  import java.net.Socket;
8  import java.io.IOException;
9  import java.util.concurrent.Executors;
10 import java.util.concurrent.ExecutorService;
11
12 import com.deitel.messenger.MessageListener;
13 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
14
15 public class DeitelMessengerServer implements MessageListener
16 {
17    private ExecutorService serverExecutor; // executor for server
18
19    // start chat server
20    public void startServer()
21    {
22       // create executor for server runnables
23       serverExecutor = Executors.newCachedThreadPool();
24
25       try // create server and manage new clients
26       {
27          // create ServerSocket for incoming connections
28          ServerSocket serverSocket =
29             new ServerSocket( SERVER_PORT, 100 );
30
```

Implement the `MessageListener` interface

Create a `ServerSocket` to accept incoming network connections

DeiterMessenger Server.java

(1 of 3)

Line 15

Outline

```
31          System.out.printf( "%s%d%s", "Server listening on port ",
32             SERVER_PORT, " ..." );
33
34          // listen for clients constantly
35          while ( true )
36          {
37             // accept new client connection
38             Socket clientSocket = serverSocket.accept();
39
40             // create MessageReceiver for receiving messages fro
41             serverExecutor.execute(
42                new MessageReceiver( this, clientSocket ) );
43
44             // print connection information
45             System.out.println( "Connection received from: " +
46                clientSocket.getInetAddress() );
47          } // end while
48       } // end try
```

DeiterMessenger

Invoke method

for

w

n

Create and start a new `MessageReceiver` for the client

Lines 41-42

```
49          catch ( IOException ioException )
50          {
51             ioException.printStackTrace();
52          } // end catch
53       } // end method startServer
54
55       // when new message is received, broadcast message to clients
56       public void messageReceived( String from, String message )
57       {
58          // create String containing entire message
59          String completeMessage = from + MESSAGE_SEPARATOR + message;
60
61          // create and start MulticastSender to broadcast messages
62          serverExecutor.execute(
63             new MulticastSender( completeMessage.getBytes() ) );
64       } // end method messageReceived
65    } // end class DeitelMessengerServer
```

DeiterMessenger Server.java

(3 of 3)

Lines 62-63

Create and start new **MulticastSender** to deliver completeMessage to all clients

Outline

DeitelMessengerServerTest.java

```java
// Fig. 24.19: DeitelMessengerServerTest.java
// Test the DeitelMessengerServer class.
package com.deitel.messenger.sockets.server;

public class DeitelMessengerServerTest
{
   public static void main ( String args[] )
   {
      DeitelMessengerServer application = new DeitelMessengerServer();
      application.startServer(); // start server
   } // end main
} // end class DeitelMessengerServerTest
```

Program output

```
Server listening on port 12345 ...
Connection received from: /127.0.0.1
Connection received from: /127.0.0.1
Connection received from: /127.0.0.1
```

```java
// Fig. 24.20: SocketMessengerConstants.java
// SocketMessengerConstants defines constants for the port numbers
// and multicast address in DeitelMessenger
package com.deitel.messenger.sockets;

public interface SocketMessengerConstants
{
   // address for multicast datagrams
   public static final String MULTICAST_ADDRESS = "239.0.0.1";

   // port for listening for multicast datagrams
   public static final int MULTICAST_LISTENING_PORT = 5555;

   // port for sending multicast datagrams
   public static final int MULTICAST_SENDING_PORT = 5554;

   // port for Socket connections to DeitelMessengerServer
   public static final int SERVER_PORT = 12345;

   // String that indicates disconnect
   public static final String DISCONNECT_STRING = "DISCONNECT";

   // String that separates the user name from the message body
   public static final String MESSAGE_SEPARATOR = ">>>";

   // message size (in bytes)
   public static final int MESSAGE_SIZE = 512;
} // end interface SocketMessengerConstants
```

SocketMessengerConstants.java

Address to send multicast datagrams

Port listening for multicast datagrams

Port for sending multicast datagrams

Port for socket connections to server

`String` that indicates disconnect

`String` that

Maximum message size in bytes

```java
// Fig. 24.21: MessageListener.java
// MessageListener is an interface for classes that wish to
// receive new chat messages.
package com.deitel.messenger;

public interface MessageListener
{
   // receive new chat message
   public void messageReceived( String from, String message );
} // end interface MessageListener
```

Method `messageReceived` allows an implementing class to receive chat messages

MessageListener.java

Outline

```
// Fig. 24.22: MessageReceiver.java
// MessageReceiver is a Runnable that listens for messages from a
// particular client and delivers messages to a MessageListener.
package com.deitel.messenger.sockets.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.util.StringTokenizer;

import com.deitel.messenger.MessageListener;
import static com.deitel.messenger.sockets.SocketMessengerConstants.*;

public class MessageReceiver implements Runnable
{
   private BufferedReader input; // input stream
   private MessageListener messageListener; // message listener
   private boolean keepListening = true; // when false, ends runnable

   // MessageReceiver constructor
   public MessageReceiver( MessageListener listener, Socket clientSocket )
   {
      // set listener to which new messages should be sent
      messageListener = listener;

```

MessageReceiver.java

(1 of 5)

Outline

```
28       try
29       {
30          // set timeout for reading from client
31          clientSocket.setSoTimeout( 5000 ); // five seconds
32
33          // create BufferedReader for reading incoming messages
34          input = new BufferedReader( new InputStreamReader(
35             clientSocket.getInputStream() ) );
36       } // end try
37       catch ( IOException ioException )
38       {
39          ioException.printStackTrace();
40       } // end catch
41    } // end MessageReceiver constructor
42
43    // listen for new messages and deliver them to MessageListener
44    public void run()
45    {
46       String message; // String for incoming messages
47
```

Attempt to read for five seconds

MessageReceiver.java

(2 of 5)

Line 31

Outline

```
48       // listen for messages until stopped
49       while ( keepListening )
50       {
51          try
52          {
53             message = input.readLine(); // read message from client
54          } // end try
55          catch ( SocketTimeoutException socketTimeoutException )
56          {
57             continue; // continue to next iteration to keep listen
58          } // end catch
59          catch ( IOException ioException )
60          {
61             ioException.printStackTrace();
62             break;
63          } // end catch
64
65          // ensure non-null message
66          if ( message != null )
67          {
68             // tokenize message to retrieve user name and message body
69             StringTokenizer tokenizer = new StringTokenizer(
70                message, MESSAGE_SEPARATOR );
71
```

.java

Read line of data from client

A SocketTimeOutException is thrown if the read times out

Line 55

Lines 69-70

Separate message into two tokens delimited by Message_SEPARATOR

```
            // ignore messages that do not contain a user
            // name and message body
            if ( tokenizer.countTokens() == 2 )
            {
               // send message to MessageListener
               messageListener.messageReceived(
                  tokenizer.nextToken(), // user name
                  tokenizer.nextToken() ); // message body
            } // end if
            else
            {
               // if disconnect message received, stop listening
               if ( message.equalsIgnoreCase(
                  MESSAGE_SEPARATOR + DISCONNECT_STRING ) )
                  stopListening();
            } // end else
         } // end if
      } // end while

```

Invoke method `messageReceived` of interface `MessageListener` to deliver the new message to the registered `MessageListener`

(4 of 5)

Determine whether message indicates that user wishes to leave chat room

Outline

```
      try
      {
         input.close(); // close BufferedReader (also closes Socket)
      } // end try
      catch ( IOException ioException )
      {
         ioException.printStackTrace();
      } // end catch
   } // end method run

   // stop listening for incoming messages
   public void stopListening()
   {
      keepListening = false;
   } // end method stopListening
} // end class MessageReceiver
```

`MessageReceiver .java`

(5 of 5)

```
1  // Fig. 24.23: MulticastSender.java
2  // MulticastSender broadcasts a chat message using a multicast datagram.
3  package com.deitel.messenger.sockets.server;
4
5  import java.io.IOException;
6  import java.net.DatagramPacket;
7  import java.net.DatagramSocket;
8  import java.net.InetAddress;
9
10 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
11
12 public class MulticastSender implements Runnable
13 {
14    private byte[] messageBytes; // message data
15
16    public MulticastSender( byte[] bytes )
17    {
18       messageBytes = bytes; // create the message
19    } // end MulticastSender constructor
20
21    // deliver message to MULTICAST_ADDRESS over DatagramSocket
22    public void run()
23    {
24       try // deliver message
25       {
26          // create DatagramSocket for sending message
27          DatagramSocket socket =
28             new DatagramSocket( MULTICAST_SENDING_PORT );
29
```

Create DatagramSocket for delivering DatagramPackets via multicast

Outline

MulticastSender.java

(1 of 2)

Lines 27-28

Outline

```
30             // use InetAddress reserved for multicast group
31             InetAddress group = InetAddress.getByName( MULTICAST_ADDRESS );
32
33             // create DatagramPacket containing message
34             DatagramPacket packet = new DatagramPacket( messageBytes,
35                messageBytes.length, group, MULTICAST_LISTENING_PORT );
36
37             socket.send( packet ); // send packet to mul
38             socket.close(); // close socket
39          } // end try
40          catch ( IOException ioException )
41          {
42             ioException.printStackTrace();
43          } // end catch
44       } // end method run
45    } // end class MulticastSender
```

Create an `InetAddress` object for the multicast address

`mpacket`

ssage

Close the `DatagramSocket`, and the `run` method returns, terminating the `MulticastSender`

`DatagramSocket` method `send`

`MulticastSender`

1

Lines 34-35

Line 37

Line 38

24.10.1 DeitelMessengerServer and Supporting Classes

- Execute `DeitelMessengerServerTest`
  - Change directories to the proper location
  - Type command

```
java com.deitel.mesenger.sockets.server.DeitelMessengerServerTest
```

24.10.2 `DeitelMessenger` Client and Supporting Classes

- **`DeitelMessengerServer` client**
  - Consists several components
    - Interface `MessageManager`
    - Class that implements interface `MessageManager`
      - Manages communication with server
    - `Runnable` subclass
      - Listens for messages at server’s multicast address
    - Another `Runnable` subclass
      - Sends messages from client to server
    - `JFrame` subclass
      - Provides client GUI

Outline

MessageManager
.java

```java
// Fig. 24.24: MessageManager.java
// MessageManger is an interface for objects capable of managing
// communications with a message server.
package com.deitel.messenger;

public interface MessageManager
{
   // connect to message server and route incoming messages
   // to given MessageListener
   public void connect( MessageListener listener );

   // disconnect from message server and stop routing
   // incoming messages to given MessageListener
   public void disconnect( MessageListener listener );

   // send message to message server
   public void sendMessage( String from, String message );
} // end interface MessageManager
```

Connects `MessageManager` to `DeitelMessengerServer` and routes incoming messages to

Disconnects `MessageManager` from `DeitelMessengerServer` and stops delivering messages to

Sends new message to `DeitelMessengerServer`

```
1  // Fig. 24.25: SocketMessageManager.java
2  // SocketMessageManager communicates with a DeitelMessengerServer using
3  // Sockets and MulticastSockets.
4  package com.deitel.messenger.sockets.client;
5
6  import java.net.InetAddress;
7  import java.net.Socket;
8  import java.io.IOException;
9  import java.util.concurrent.Executors;
10 import java.util.concurrent.ExecutorService;
11 import java.util.concurrent.ExecutionException;
12 import java.util.concurrent.Future;
13
14 import com.deitel.messenger.MessageListener;
15 import com.deitel.messenger.MessageManager;
16 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
17
18 public class SocketMessageManager implements MessageManager
19 {
20    private Socket clientSocket; // Socket for outgoing messages
21    private String serverAddress; // DeitelMessengerServer address
22    private PacketReceiver receiver; // receives multicast messages
23    private boolean connected = false; // connection status
24    private ExecutorService serverExecutor; // executor for server
25
```

Outline

SocketMessageManager.java

(1 of 4)

Line 20

Line 22

`Socket` for connecting and sending messages to `Deite`

`Runnable` listens for incoming messages

```
26    public SocketMessageManager( String address )
27    {
28       serverAddress = address; // store server address
29       serverExecutor = Executors.newCachedThreadPool();
30    } // end SocketMessageManager constructor
31
32    // connect to server and send messages to given MessageListener
33    public void connect( MessageListener listener )
34    {
35       if ( connected )
36          return; // if already connected, return immediately
37
38       try // open Socket connection to DeitelMessengerServer
39       {
40          clientSocket = new Socket(
41             InetAddress.getByName( serverAddress ), SERVER_PORT );
42
43          // create runnable for receiving incoming messages
44          receiver = new PacketReceiver( listener );
45          serverExecutor.execute( receiver ); // execute run
46          connected = true; // update connected flag
47       } // end try
48       catch ( IOException ioException )
49       {
50          ioException.printStackTrace();
51       } // end catch
52    } // end method connect
53
```

Outline

`SocketMessageManager.java`

(2 of 4)

Lines 40-41

Line 45

Create `Socket` to communicate with `DeitelMessengerServer`

Create a new `packetReceiver`, which listens for incoming multicast messa

Execute the `Runnable`

Outline

```
54    // disconnect from server and unregister given MessageListener
55    public void disconnect( MessageListener listener )
56    {
57       if ( !connected )
58          return; // if not connected, return immediately
59
60
61
62          // notify server that client is disconnecting
63          Runnable disconnecter = new MessageSender( clientSocket, "",
64             DISCONNECT_STRING );
65          Future disconnecting = serverExecutor.submit( disconnecter );
66          disconnecting.get(); // wait for disconnect message to be sent
67          receiver.stopListening(); // stop receiver
68          clientSocket.close(); // close outgoing Socket
69       } // end try
70       catch ( ExecutionException exception )
71       {
72          exception.printStackTrace();
73       } // end catch
74       catch ( InterruptedException exception )
75       {
76          exception.printStackTrace();
77       } // end catch
78       catch ( IOException ioException )
79       {
80          ioException.printStackTrace();
81       } // end catch
82
```

Create a new MessageSender to send DISCONNECT_STRING to the server

Start the MessageSender to deliver the message d submit of ExecutorService

Invoke Future method get to wait for the disconnect message to be delivered and the Runnable to terminate

(3 of 4)

Lines 63-64

Line 65

Line 66

Outline

SocketMessageManager.java

(4 of 4)

```
         connected = false; // update connected flag
      } // end method disconnect

      // send message to server
      public void sendMessage( String from, String message )
      {
         if ( !connected )
            return; // if not connected, return immediately

         // create and start new MessageSender to deliver message
         serverExecutor.execute(
            new MessageSender( clientSocket, from, message) );
      } // end method sendMessage
   } // end method SocketMessageManager
```

Create and start a new `MessageSender` to deliver the new message in a separate thread of execution

```java
// Fig. 24.26: MessageSender.java
// Sends a message to the chat server in a separate runnable.
package com.deitel.messenger.sockets.client;

import java.io.IOException;
import java.util.Formatter;
import java.net.Socket;

import static com.deitel.messenger.sockets.SocketMessengerConstants.*;

public class MessageSender implements Runnable
{
   private Socket clientSocket; // Socket over which to send message
   private String messageToSend; // message to send

   public MessageSender( Socket socket, String userName, String message )
   {
      clientSocket = socket; // store socket for client

      // build message to be sent
      messageToSend = userName + MESSAGE_SEPARATOR + message;
   } // end MessageSender constructor

```

Outline

MessageSender.java

(1 of 2)

Outline

va

```
24     // send message and end
25     public void run()
26     {
27        try // send message and flush PrintWriter
28        {
29           Formatter output =
30              new Formatter( clientSocket.getOutputStrea
31           output.format( "%s\n", messageToSend ); // se
32           output.flush(); // flush output
33        } // end try
34        catch ( IOException ioException )
35        {
36           ioException.printStackTrace();
37        } // end catch
38     } // end method run
39  } // end class MessageSender
```

Create a new Formatter for the cli

I voke Formatter nethod format to ormat the message

Invoke Formatter method flush to ensure that the message is sent immediately

Lines 29-30

Line 31

Line 32

```java
// Fig. 24.27: PacketReceiver.java
// PacketReceiver listens for DatagramPackets containing
// messages from a DeitelMessengerServer.
package com.deitel.messenger.sockets.client;

import java.io.IOException;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.DatagramPacket;
import java.net.SocketTimeoutException;
import java.util.StringTokenizer;

import com.deitel.messenger.MessageListener;
import static com.deitel.messenger.sockets.SocketMessengerConstants.*;

public class PacketReceiver implements Runnable
{
   private MessageListener messageListener; // receives messages
   private MulticastSocket multicastSocket; // receive broadcast messages
   private InetAddress multicastGroup; // InetAddress of multicast group
   private boolean keepListening = true; // terminates PacketReceiver

   public PacketReceiver( MessageListener listener )
   {
      messageListener = listener; // set MessageListener

```

Outline

PacketReceiver.java

(1 of 4)

Outline

```
      try // connect MulticastSocket to multicast address and port
      {
         // create new MulticastSocket
         multicastSocket = new MulticastSocket(
            MULTICAST_LISTENING_PORT );

         // use InetAddress to get multicast group
         multicastGroup = InetAddress.getByName( MULTICAST_ADDRESS );

         // join multicast group to receive messages
         multicastSocket.joinGroup( multicastGroup );

         // set 5 second timeout when waiting for new packet
         multicastSocket.setSoTimeout( 5000 );
      } // end try
      catch ( IOException ioException )
      {
         ioException.printStackTrace();
      } // end catch
   } // end PacketReceiver constructor

   // listen for messages from multicast group
   public void run()
   {
      // listen for messages until stopped
      while ( keepListening )
      {
         // create buffer for incoming message
         byte[] buffer = new byte[ MESSAGE_SIZE ];
```

MulticastSocket listens for incoming chat messages on port MULTICAST_LISTENING_PORT

PacketReceiver.java

(2 of 4)

InetAddress object to which DeitelMessengerServer

Register MulticastSocket to receive messages sent to MULTICAST_ADDRESS

Invoke MulticastSocket method setSoTimeout to specify that if no data is received in 5000 milliseconds, the MulticastSocket should issue an InterruptedIOException

Create byte array for storing DatagramPacket

```
57          // create DatagramPacket for incoming message
58          DatagramPacket packet = new DatagramPacket( buffer,
59             MESSAGE_SIZE );
60
61          try // receive new DatagramPacket (blocking call)
62          {
63             multicastSocket.receive( packet );
64          } // end try
65          catch ( SocketTimeoutException socketTimeoutException )
66          {
67             continue; // continue to next iteration to keep listening
68          } // end catch
69          catch ( IOException ioException )
70          {
71             ioException.printStackTrace();
72             break;
73          } // end catch
74
75          // put message data in a String
76          String message = new String( packet.getData() );
77
78          message = message.trim(); // trim whitespace from message
79
80          // tokenize message to retrieve user name and message body
81          StringTokenizer tokenizer = new StringTokenizer(
82             message, MESSAGE_SEPARATOR );
83
```

Create DatagramPacket for storing message

Receive incoming packet from multicast address

Invoke method trim of class String to remove

Create a StringTokenizer to separate the message body from the name of the user who sent the message

PacketReceiver

(3 of 4)

Lines 58-59

Line 63

Line 78

Lines 81-82

Outline

```
      // ignore messages that do not contain a user
      // name and message body
      if ( tokenizer.countTokens() == 2 )
      {
        // send message to MessageListener
        messageListener.messageReceived(
          tokenizer.nextToken(), // user name
          tokenizer.nextToken() ); // message body
      } // end if
    } // end while

    try
    {
      multicastSocket.leaveGroup( multicastGroup ); // leave group
      multicastSocket.close(); // close MulticastSocket
    } // end try
    catch ( IOException ioException )
    {
      ioException.printStackTrace();
    } // end catch
  } // end method run

  // stop listening for new messages
  public void stopListening()
  {
    keepListening = false;
  } // end method stopListening
} // end class PacketReceiver
```

After parsing message, deliver message to `PacketReceiver`’s `MessageListener`

.java

Invoke `MulicastSocket` method `leaveGroup` to stop receiving messages from the multicast address

Invoke `MulticastSocket` method `close` to close the `MulticastSocket`

```java
// Fig. 24.28: ClientGUI.java
// ClientGUI provides a user interface for sending and receiving
// messages to and from the DeitelMessengerServer.
package com.deitel.messenger;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import javax.swing.border.BevelBorder;

```

ClientGUI.java

(1 of 10)

Outline

`ClientGUI.java`

(2 of 10)

Line 40

Line 41

```
28 public class ClientGUI extends JFrame
29 {
30    private JMenu serverMenu; // for connecting/disconnecting server
31    private JTextArea messageArea; // displays messages
32    private JTextArea inputArea; // inputs messages
33    private JButton connectButton; // button for connecting
34    private JMenuItem connectMenuItem; // menu item for connecting
35    private JButton disconnectButton; // button for disconnecting
36    private JMenuItem disconnectMenuItem; // menu item for disconnecting
37    private JButton sendButton; // sends messages
38    private JLabel statusBar; // label for connection status
39    private String userName; // userName to add to outgoing messages
40    private MessageManager messageManager; // communicates with server
41    private MessageListener messageListener; // receives incomin
42
43    // ClientGUI constructor
44    public ClientGUI( MessageManager manager )
45    {
46       super( "Deitel Messenger" );
47
48       messageManager = manager; // set the MessageManager
49
```

`MessageListener` receives incoming messages from `MessageManager`

lles
erver

```
50       // create MyMessageListener for receiving messages
51       messageListener = new MyMessageListener();
52 
53       serverMenu = new JMenu ( "Server" ); // create Server JMenu
54       serverMenu.setMnemonic( 'S' ); // set mnemonic for server menu
55       JMenuBar menuBar = new JMenuBar(); // create JMenuBar
56       menuBar.add( serverMenu ); // add server menu to menu bar
57       setJMenuBar( menuBar ); // add JMenuBar to application
58 
59       // create ImageIcon for connect buttons
60       Icon connectIcon = new ImageIcon(
61          lus().getResource( "images/Connect.gif" ) );
62 
63       // create connectButton and connectMenuItem
64       connectButton = new JButton( "Connect", connectIcon );
65       connectMenuItem = new JMenuItem( "Connect", connectIcon );
66       connectMenuItem.setMnemonic( 'C' );
67 
68       // create ConnectListener for connect buttons
69       ActionListener connectListener = new ConnectListener();
70       connectButton.addActionListener( connectListener );
71       connectMenuItem.addActionListener( connectListener );
72 
73       // create ImageIcon for disconnect buttons
74       Icon disconnectIcon = new ImageIcon(
75          getClass().getResource( "images/Disconnect.gif" ) );
76 
```

Create an instance of `MyMessageListener`, which implements interface `MessageListener`

(3 of 10)

Line 51

Outline

ClientGUI.java

(4 of 10)

```
      // create disconnectButton and disconnectMenuItem
      disconnectButton = new JButton( "Disconnect", disconnectIcon );
      disconnectMenuItem = new JMenuItem( "Disconnect", disconnectIcon );
      disconnectMenuItem.setMnemonic( 'D' );

      // disable disconnect button and menu item
      disconnectButton.setEnabled( false );
      disconnectMenuItem.setEnabled( false );

      // create DisconnectListener for disconnect buttons
      ActionListener disconnectListener = new DisconnectListener();
      disconnectButton.addActionListener( disconnectListener );
      disconnectMenuItem.addActionListener( disconnectListener );

      // add connect and disconnect JMenuItems to fileMenu
      serverMenu.add( connectMenuItem );
      serverMenu.add( disconnectMenuItem );

      // add connect and disconnect JButtons to buttonPanel
      JPanel buttonPanel = new JPanel();
      buttonPanel.add( connectButton );
      buttonPanel.add( disconnectButton );

      messageArea = new JTextArea(); // displays messages
      messageArea.setEditable( false ); // disable editing
      messageArea.setWrapStyleWord( true ); // set wrap style to word
      messageArea.setLineWrap( true ); // enable line wrapping

```

ClientGUI.java

(5 of 10)

Lines 128-129

```
105       // put messageArea in JScrollPane to enable scrolling
106       JPanel messagePanel = new JPanel();
107       messagePanel.setLayout( new BorderLayout( 10, 10 ) );
108       messagePanel.add( new JScrollPane( messageArea ),
109          BorderLayout.CENTER );
110 
111       inputArea = new JTextArea( 4, 20 ); // for entering new messages
112       inputArea.setWrapStyleWord( true ); // set wrap style to word
113       inputArea.setLineWrap( true ); // enable line wrapping
114       inputArea.setEditable( false ); // disable editing
115 
116       // create Icon for sendButton
117       Icon sendIcon = new ImageIcon(
118          getClass().getResource( "images/Send.gif" ) );
119 
120       sendButton = new JButton( "Send", sendIcon ); // create send button
121       sendButton.setEnabled( false ); // disable send button
122       sendButton.addActionListener(
123          new ActionListener()
124          {
125             // send new message when user activates sendButton
126             public void actionPerformed( ActionEvent event )
127             {
128                messageManager.sendMessage( userName,
129                   inputArea.getText() ); // send message
130                inputArea.setText( "" ); // clear inputArea
131             } // end method actionPerformed
132          } // end anonymous inner class
133       ); // end call to addActionListener
134 
```

Send user’s name and `inputArea`’s text to `DeitelMessengerServer` as a chat message

```
135       Box box = new Box( BoxLayout.X_AXIS ); // create new box for layout
136       box.add( new JScrollPane( inputArea ) ); // add input area to box
137       box.add( sendButton ); // add send button to box
138       messagePanel.add( box, BorderLayout.SOUTH ); // add box to panel
139
140       // create JLabel for statusBar with a recessed border
141       statusBar = new JLabel( "Not Connected" );
142       statusBar.setBorder( new BevelBorder( BevelBorder.LOWERED ) );
143
144       add( buttonPanel, BorderLayout.NORTH ); // add button panel
145       add( messagePanel, BorderLayout.CENTER ); // add message panel
146       add( statusBar, BorderLayout.SOUTH ); // add status bar
147
148       // add WindowListener to disconnect when user quits
149       addWindowListener (
150          new WindowAdapter ()
151          {
152             // disconnect from server and exit application
153             public void windowClosing ( WindowEvent event )
154             {
155                messageManager.disconnect( messageListener );
156                System.exit( 0 );
157             } // end method windowClosing
158          } // end anonymous inner class
159       ); // end call to addWindowListener
160    } // end ClientGUI constructor
161
```

Disconnect from chat server when user exits client application

Outline

ClientGUI.java

(6 of 10)

Line 155

Outline

ClientGUI.java

Line 169

Lines 172-173

```
162   // ConnectListener listens for user requests to connect to server
163   private class ConnectListener implements ActionListener
164   {
165      // connect to server and enable/disable GUI components
166      public void actionPerformed( ActionEvent event )
167      {
168         // connect to server and route messages to messag
169         messageManager.connect( messageListener );
170
171         // prompt for userName
172         userName = JOptionPane.showInputDialog(
173            ClientGUI.this, "Enter user name:" );
174
175         messageArea.setText( "" ); // clear messageArea
176         connectButton.setEnabled( false ); // disable connect
177         connectMenuItem.setEnabled( false ); // disable connect
178         disconnectButton.setEnabled( true ); // enable disconnect
179         disconnectMenuItem.setEnabled( true ); // enable disconnect
180         sendButton.setEnabled( true ); // enable send button
181         inputArea.setEditable( true ); // enable editing for input area
182         inputArea.requestFocus(); // set focus to input area
183         statusBar.setText( "Connected: " + userName ); // set text
184      } // end method actionPerformed
185   } // end ConnectListener inner class
186
```

When user accesses **Connect** menu, connect to chat server

Prompt the user for a user name

```
187    // DisconnectListener listens for user requests to disconnect
188    // from DeitelMessengerServer
189    private class DisconnectListener implements ActionListener
190    {
191       // disconnect from server and enable/disable GUI components
192       public void actionPerformed( ActionEvent event )
193       {
194          // disconnect from server and stop routing messages
195          messageManager.disconnect( messageListener );
196          sendButton.setEnabled( false ); // disable send button
197          disconnectButton.setEnabled( false ); // disable disconnect
198          disconnectMenuItem.setEnabled( false ); //
199          inputArea.setEditable( false ); // disable
200          connectButton.setEnabled( true ); // enable connect
201          connectMenuItem.setEnabled( true ); // enable connect
202          statusBar.setText( "Not Connected" ); // set status bar text
203       } // end method actionPerformed
204    } // end DisconnectListener inner class
205
```

Invoke `MessageManager` method `disconnect` to disconnect from chat server

Outline

`ClientGUI.java`

(8 of 10)

Line 195

◀ ▶

Outline

ClientGUI.java

(9 of 10)

```java
// MyMessageListener listens for new messages from MessageManager and
// displays messages in messageArea using MessageDisplayer.
private class MyMessageListener implements MessageListener
{
   // when received, display new messages in messageArea
   public void messageReceived( String from, String message )
   {
      // append message using MessageDisplayer
      SwingUtilities.invokeLater(
         new MessageDisplayer( from, message ) );
   } // end method messageReceived
} // end MyMessageListener inner class

// Displays new message by appending message to JTextArea.  Should
// be executed only in Event thread; modifies live Swing component
private class MessageDisplayer implements Runnable
{
   private String fromUser; // user from which message came
   private String messageBody; // body of message

```

Display message when `MessageListener` detects that message was received

```
226      // MessageDisplayer constructor
227      public MessageDisplayer( String from, String body )
228      {
229         fromUser = from; // store originating user
230         messageBody = body; // store message body
231      } // end MessageDisplayer constructor
232
233      // display new message in messageArea
234      public void run()
235      {
236         // append new message
237         messageArea.append( "\n" + fromUser + "> " + messageBody );
238      } // end method run
239   } // end MessageDisplayer inner class
240} // end class ClientGUI
```

`ClientGUI.java`

(10 of 10)

Line 237

Append the user name, “> ” and `messageBody` to `messageArea`

Outline

```
 1 // Fig. 24.29: DeitelMessenger.java
 2 // DeitelMessenger is a chat application that uses a ClientGUI
 3 // and SocketMessageManager to communicate with DeitelMessengerServer.
 4 package com.deitel.messenger.sockets.client;
 5
 6 import com.deitel.messenger.MessageManager;
 7 import com.deitel.messenger.ClientGUI;
 8
 9 public class DeitelMessenger
10 {
11    public static void main( String args[] )
12    {
13       MessageManager messageManager; // declare MessageManager
14
15       if ( args.length == 0 )
16          // connect to localhost
17          messageManager = new SocketMessageManager( "localhost" );
18       else
19          // connect using command-line arg
20          messageManager = new SocketMessageManager( args[ 0 ] );
21
22       // create GUI for SocketMessageManager
23       ClientGUI clientGUI = new ClientGUI( messageManager );
24       clientGUI.setSize( 300, 400 ); // set window size
25       clientGUI.setResizable( false ); // disable resizing
26       clientGUI.setVisible( true ); // show window
27    } // end main
28 } // end class DeitelMessenger
```

DeitelMessenger.java

(1 of 3)

Line 17

Line 20

Line 23

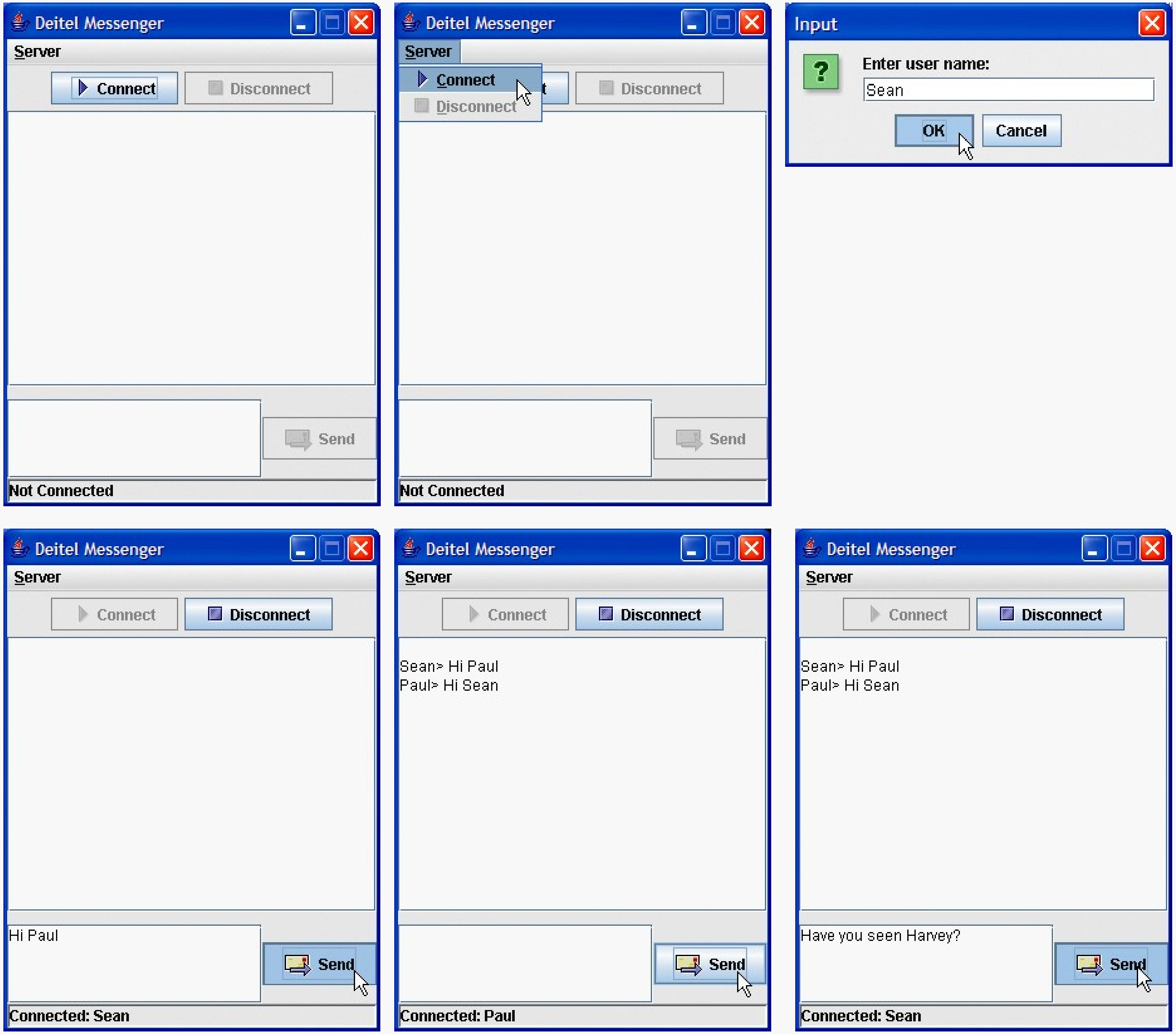
Create a client to connect to the localhost

Connect to a host supplied by the user

Create a `ClientGUI` for the `MessageManager`

DeitelMessenger.java

(2 of 3)



Not Connected

Not Connected

Enter user name: Sean

Hi Paul

Connected: Sean

Sean> Hi Paul
Paul> Hi Sean

Connected: Paul

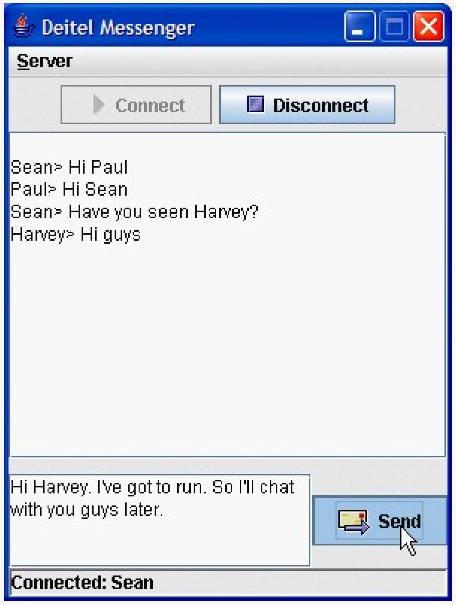
Sean> Hi Paul
Paul> Hi Sean

Have you seen Harvey?

Connected: Sean

`DeitelMessenger .java`

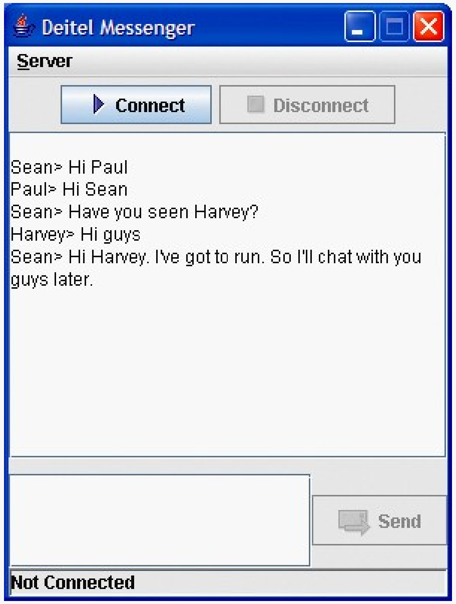
(3 of 3)



```
Sean> Hi Paul
Paul> Hi Sean
Sean> Have you seen Harvey?
Harvey> Hi guys
```

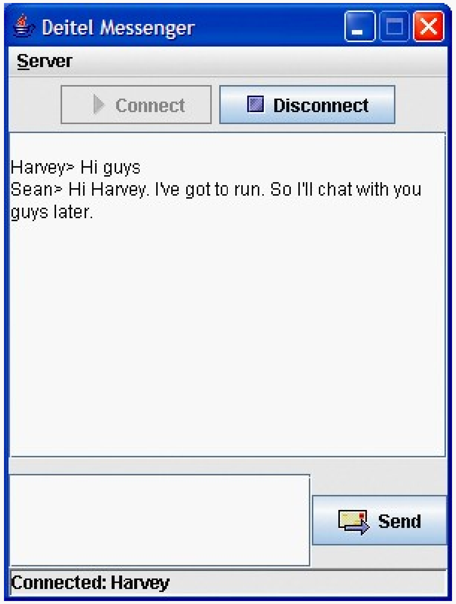
Hi Harvey. I've got to run. So I'll chat with you guys later.

Connected: Sean



```
Sean> Hi Paul
Paul> Hi Sean
Sean> Have you seen Harvey?
Harvey> Hi guys
Sean> Hi Harvey. I've got to run. So I'll chat with you
guys later.
```

Not Connected



```
Harvey> Hi guys
Sean> Hi Harvey. I've got to run. So I'll chat with you
guys later.
```

Connected: Harvey

24.10.2 `DeitelMessenger` Client and Supporting Classes

- Execute **`DeitelMessenger`** client application
  - Change directories to the proper location
  - Type command

```
java com.deitel.messenger.sockets.client.DeitelMessenger
java com.deitel.messenger.sockets.client.DeitelMessenger
localhost
java com.deitel.messenger.sockets.client.DeitelMessenger
127.0.0.1
```