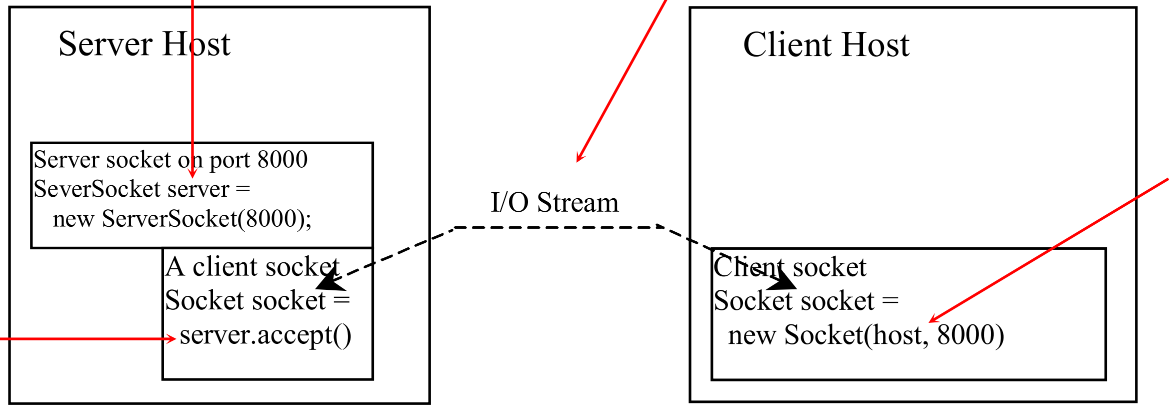
26 Networking

Chapter 30@8e

Client/Server Communications

The server must be running when a client starts. The server waits for a connection request from a client. To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.

After the server accepts the connection, communication between server and client is conducted the same as for I/O streams.



Server Host

Server socket on port 8000
SeverSocket server =
  new ServerSocket(8000);

A client socket
Socket socket =
  server.accept()

I/O Stream

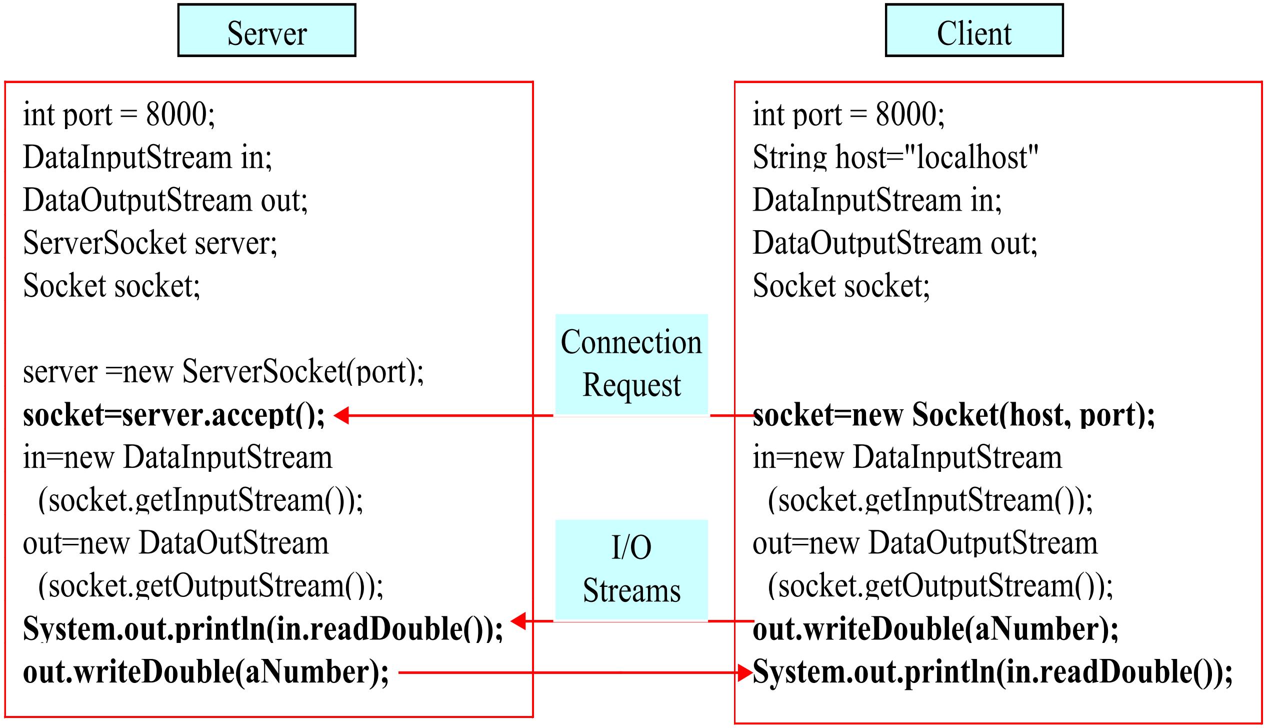
Client Host

Client socket
Socket socket =
  new Socket(host, 8000)

After a server socket is created, the server can use this statement to listen for connections.

The client issues this statement to request a connection to a server.

Data Transmission through Sockets



Server

```
int port = 8000;
DataInputStream in;
DataOutputStream out;
ServerSocket server;
Socket socket;

server =new ServerSocket(port);
socket=server.accept();
in=new DataInputStream
  (socket.getInputStream());
out=new DataOutStream
  (socket.getOutputStream());
System.out.println(in.readDouble());
out.writeDouble(aNumber);
```

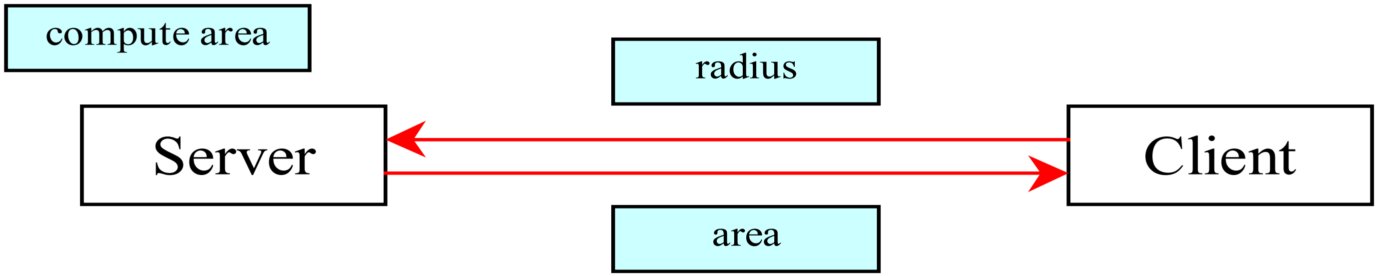
Client

```
int port = 8000;
String host="localhost"
DataInputStream in;
DataOutputStream out;
Socket socket;


socket=new Socket(host, port);
in=new DataInputStream
  (socket.getInputStream());
out=new DataOutputStream
  (socket.getOutputStream());
out.writeDouble(aNumber);
System.out.println(in.readDouble());
```

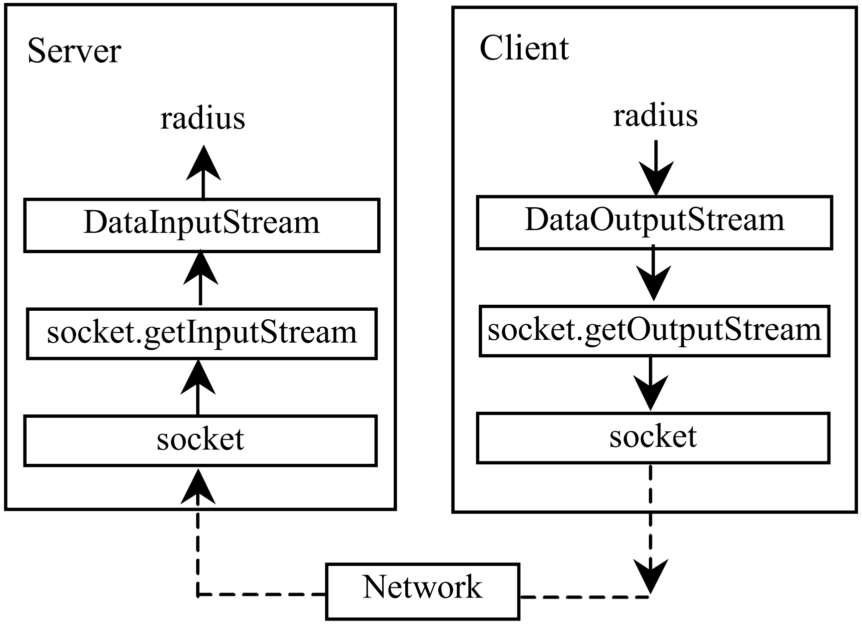
```
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputStream();
```

A Client/Server Example

- Problem: Write a client to send data to a server. The server receives the data, uses it to produce a result, and then sends the result back to the client. The client displays the result on the console. In this example, the data sent from the client is the radius of a circle, and the result produced by the server is the area of the circle.



A Client/Server Example, cont.



Server

radius

DataInputStream

socket.getInputStream

socket

Client

radius

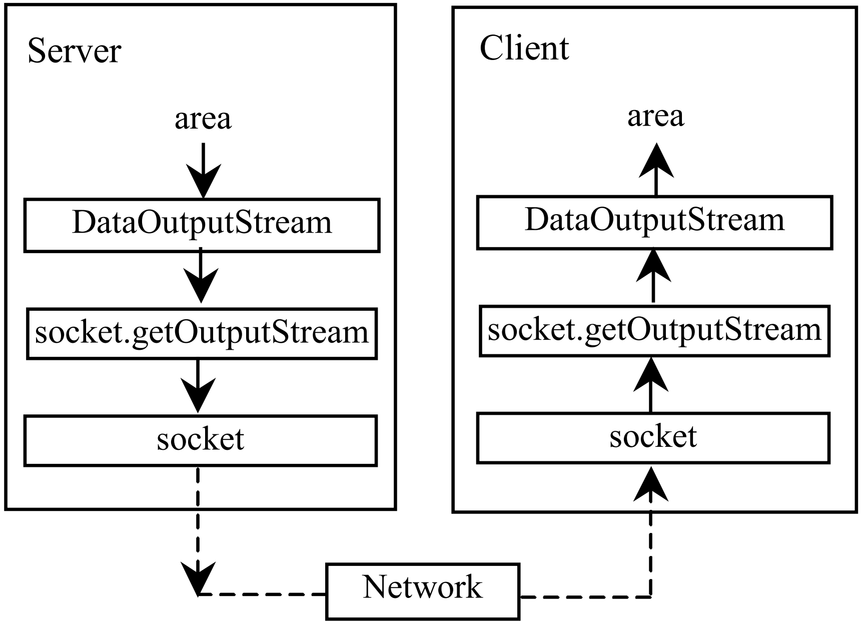
DataOutputStream

socket.getOutputStream

socket

Network

(A)



Server

area

DataOutputStream

socket.getOutputStream

socket

Client

area

DataOutputStream

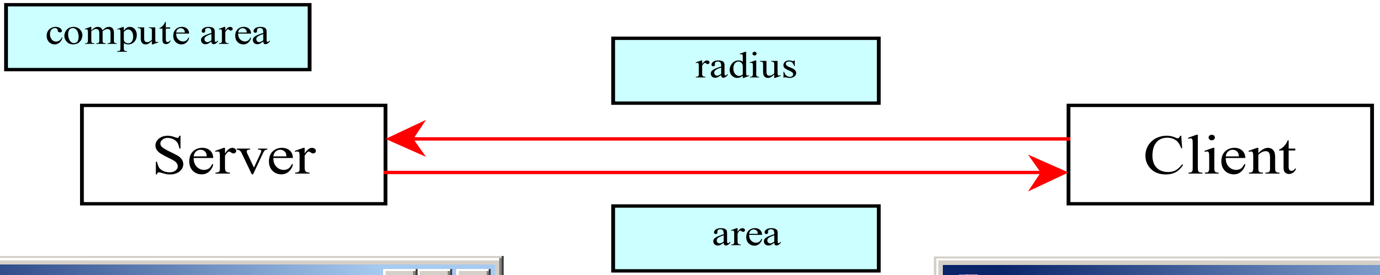
socket.getOutputStream

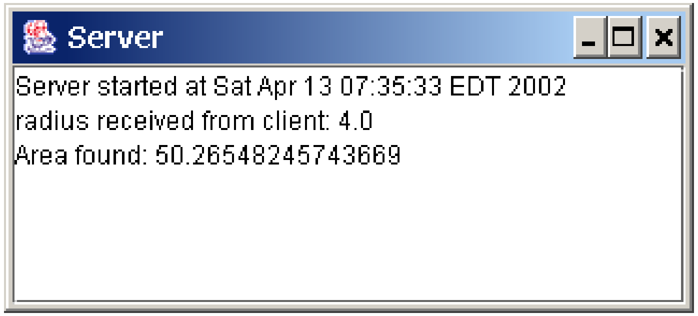
socket

Network

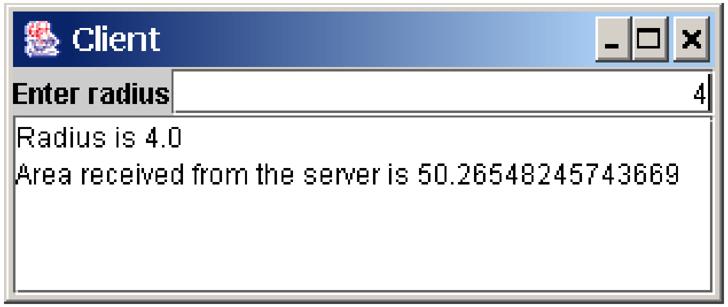
(B)

A Client/Server Example, cont.





```
Server started at Sat Apr 13 07:35:33 EDT 2002
radius received from client: 4.0
Area found: 50.26548245743669
```



Enter radius 4

```
Radius is 4.0
Area received from the server is 50.26548245743669
```

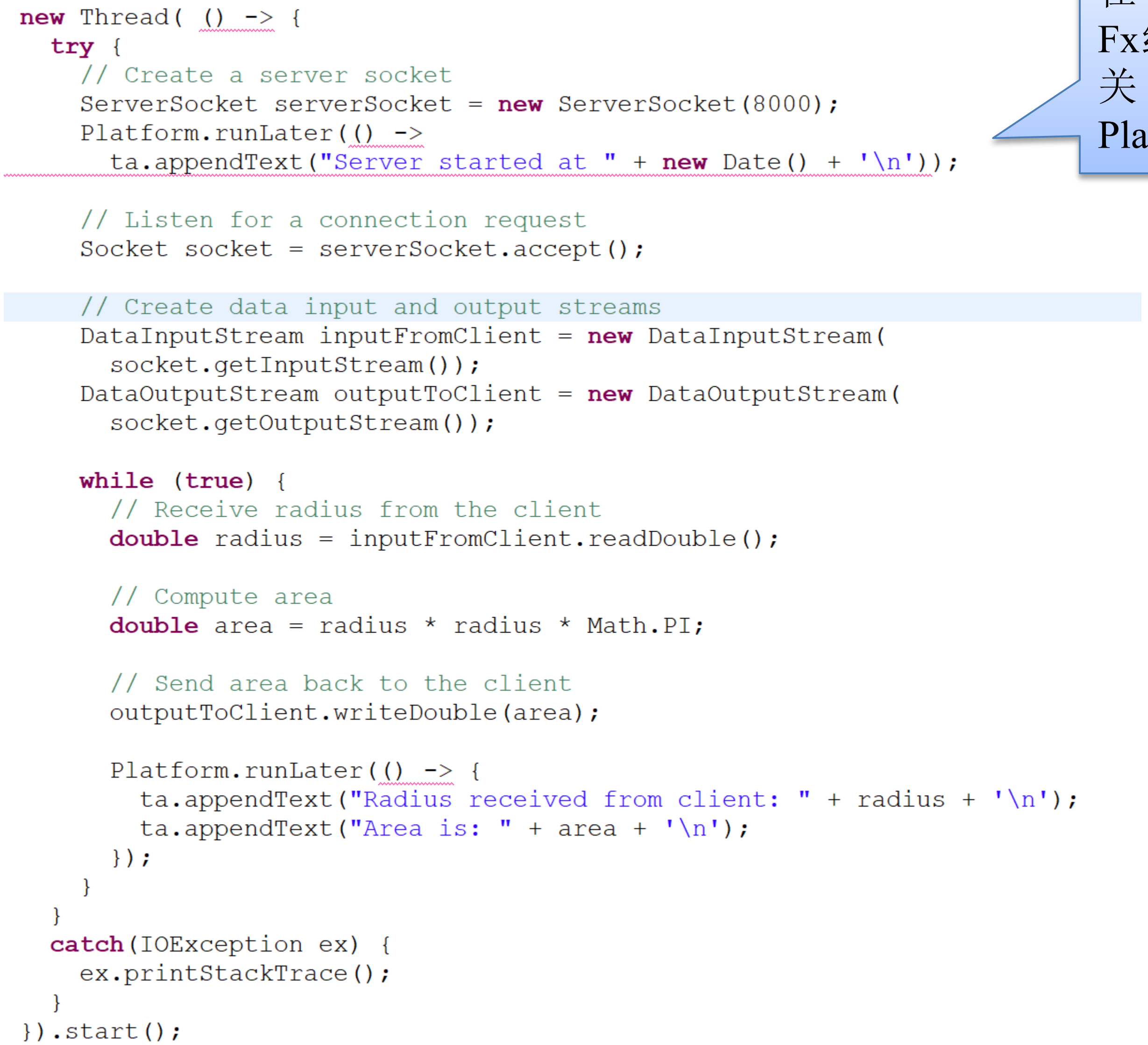
Server Code

Client Code

Start Server

Start Client

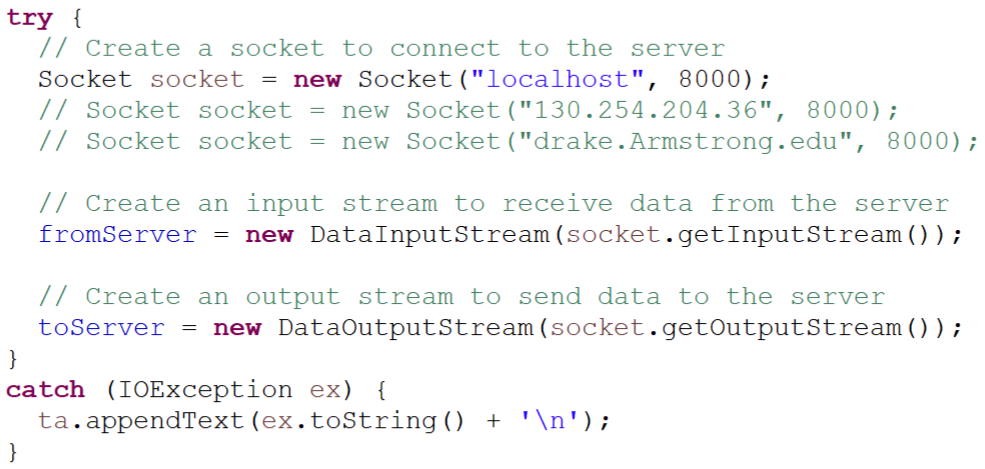
Note: Start the server, then the client.



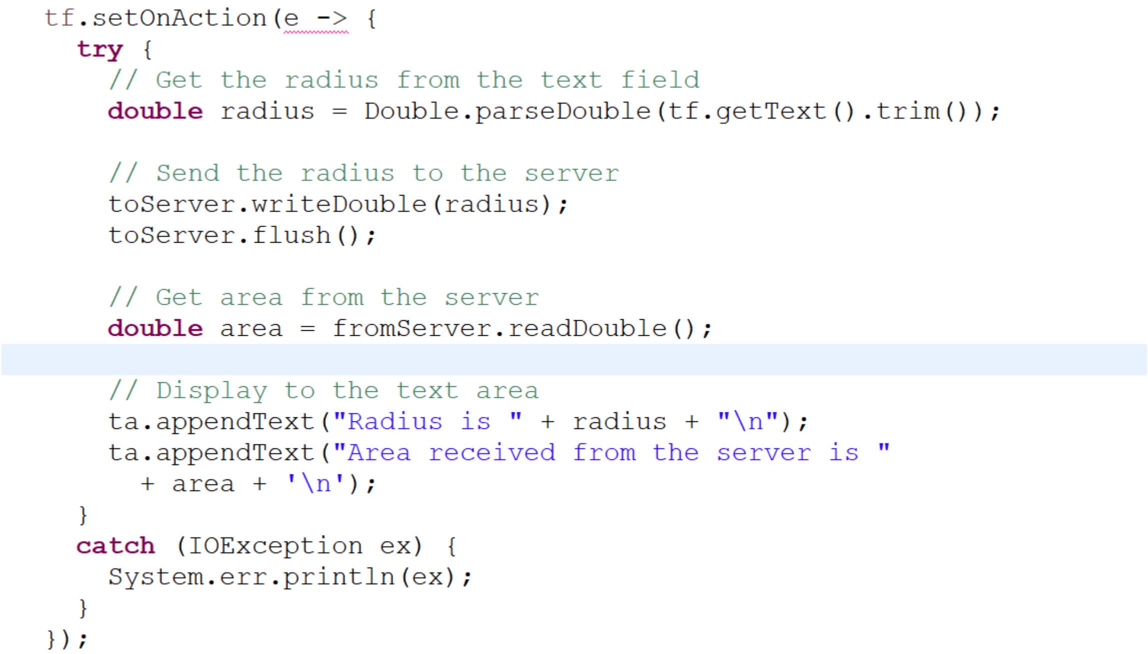
```
new Thread( () -> {
  try {
    // Create a server socket
    ServerSocket serverSocket = new ServerSocket(8000);
    Platform.runLater(() ->
      ta.appendText("Server started at " + new Date() + '\n'));

    // Listen for a connection request
    Socket socket = serverSocket.accept();

    // Create data input and output streams
    DataInputStream inputFromClient = new DataInputStream(
      socket.getInputStream());
    DataOutputStream outputToClient = new DataOutputStream(
      socket.getOutputStream());

    while (true) {
      // Receive radius from the client
      double radius = inputFromClient.readDouble();

      // Compute area
      double area = radius * radius * Math.PI;

      // Send area back to the client
      outputToClient.writeDouble(area);

      Platform.runLater(() -> {
        ta.appendText("Radius received from client: " + radius + '\n');
        ta.appendText("Area is: " + area + '\n');
      });
    }
  }
  catch(IOException ex) {
    ex.printStackTrace();
  }
}).start();
```

在 JavaFx 中，如果在非Fx线程要执行Fx线程相关的任务，必须在Platform.runlater 中执行

Server



```
try {
  // Create a socket to connect to the server
  Socket socket = new Socket("localhost", 8000);
  // Socket socket = new Socket("130.254.204.36", 8000);
  // Socket socket = new Socket("drake.Armstrong.edu", 8000);

  // Create an input stream to receive data from the server
  fromServer = new DataInputStream(socket.getInputStream());

  // Create an output stream to send data to the server
  toServer = new DataOutputStream(socket.getOutputStream());
}
catch (IOException ex) {
  ta.appendText(ex.toString() + '\n');
}
```



```
tf.setOnAction(e -> {
  try {
    // Get the radius from the text field
    double radius = Double.parseDouble(tf.getText().trim());

    // Send the radius to the server
    toServer.writeDouble(radius);
    toServer.flush();

    // Get area from the server
    double area = fromServer.readDouble();

    // Display to the text area
    ta.appendText("Radius is " + radius + "\n");
    ta.appendText("Area received from the server is "
      + area + '\n');
  }
  catch (IOException ex) {
    System.err.println(ex);
  }
});
```

Client

The InetAddress Class

Occasionally, you would like to know who is connecting to the server. You can use the InetAddress class to find the client's host name and IP address. The InetAddress class models an IP address. You can use the statement shown below to create an instance of InetAddress for the client on a socket.

```
InetAddress inetAddress = socket.getInetAddress();
```

Next, you can display the client's host name and IP address, as follows:

```
System.out.println("Client's host name is " +
  inetAddress.getHostName());
System.out.println("Client's IP Address is " +
  inetAddress.getHostAddress());
```

IdentifyHostNameIP

Serving Multiple Clients

Multiple clients are quite often connected to a single server at the same time. Typically, a server runs constantly on a server computer, and clients from all over the Internet may want to connect to it. You can use threads to handle the server's multiple clients simultaneously. Simply create a thread for each connection. Here is how the server handles the establishment of a connection:

```
while (true) {
  Socket socket = serverSocket.accept();
  Thread thread = new ThreadClass(socket);
  thread.start();
}
```

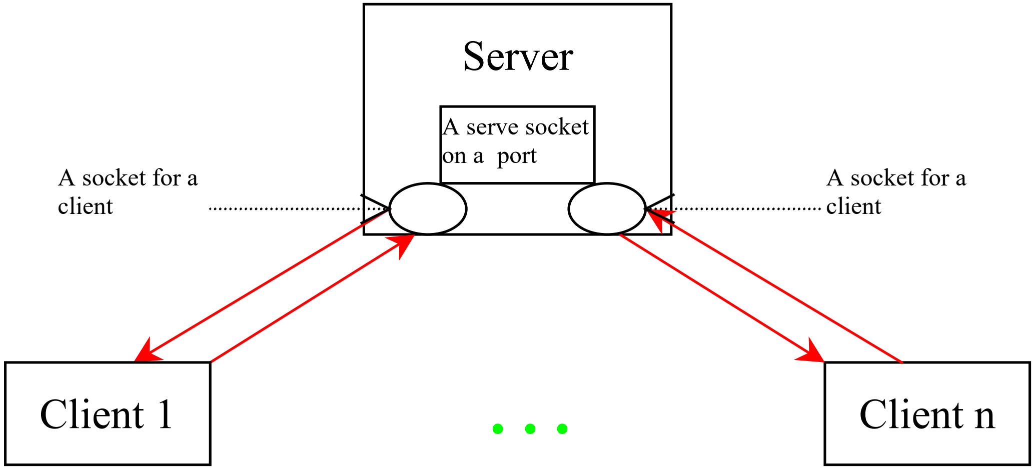
The server socket can have many connections. Each iteration of the while loop creates a new connection. Whenever a connection is established, a new thread is created to handle communication between the server and the new client; and this allows multiple connections to run at the same time.

Example: Serving Multiple Clients

Server for Multiple Clients

Start Server

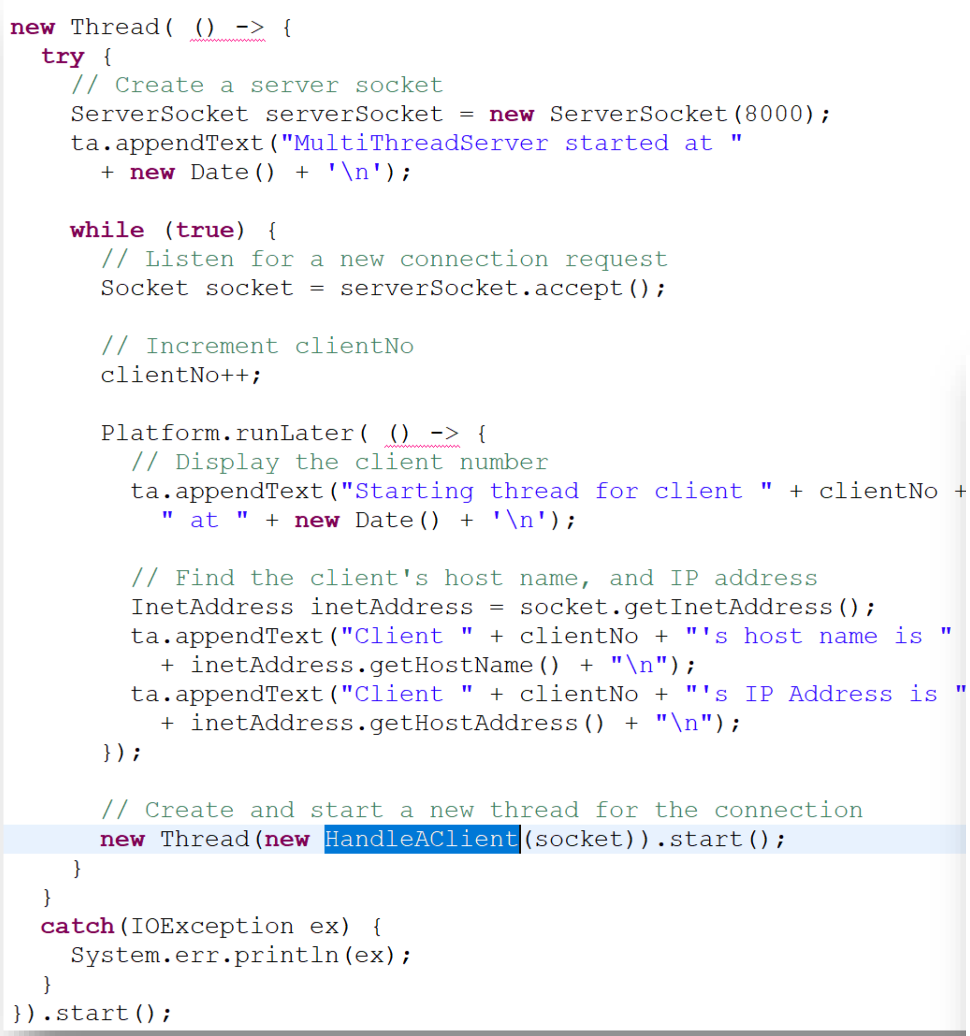
Start Client



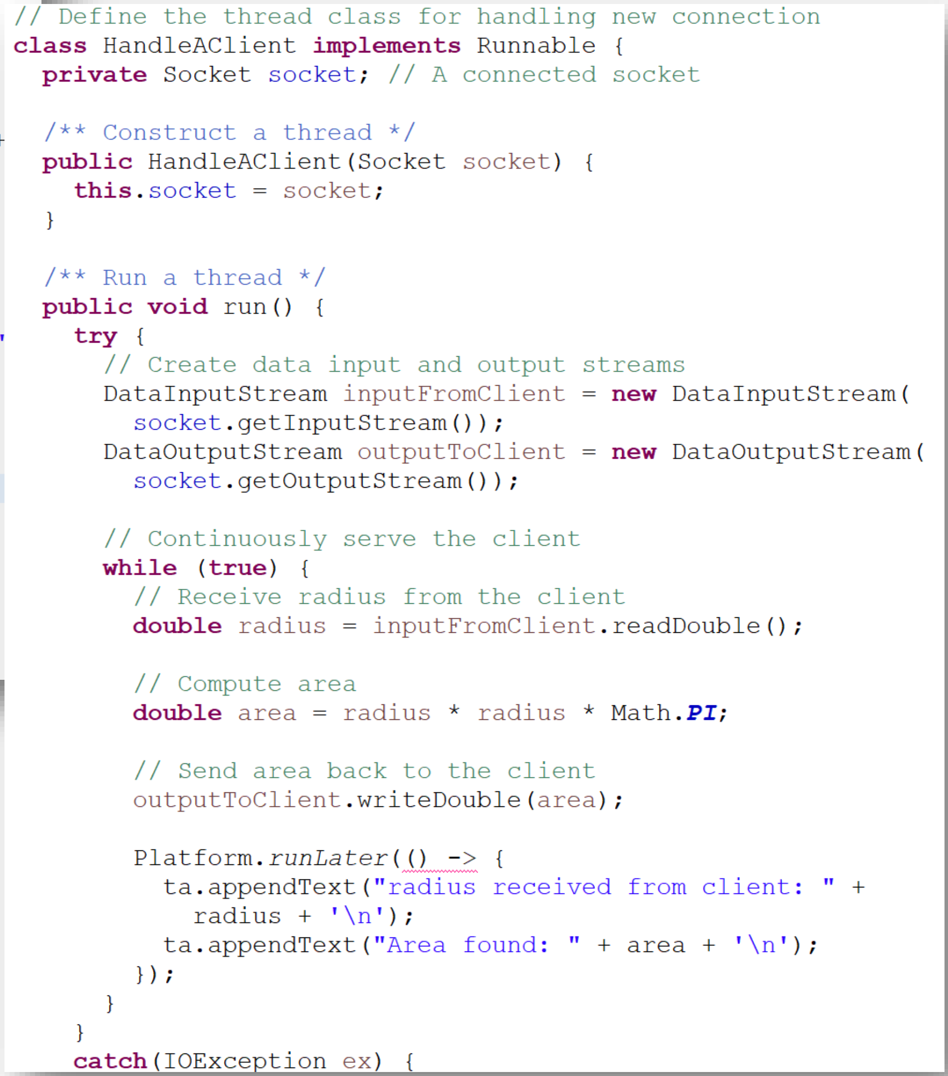
Note: Start the server first, then start multiple clients.

MultiThreadServer

Client



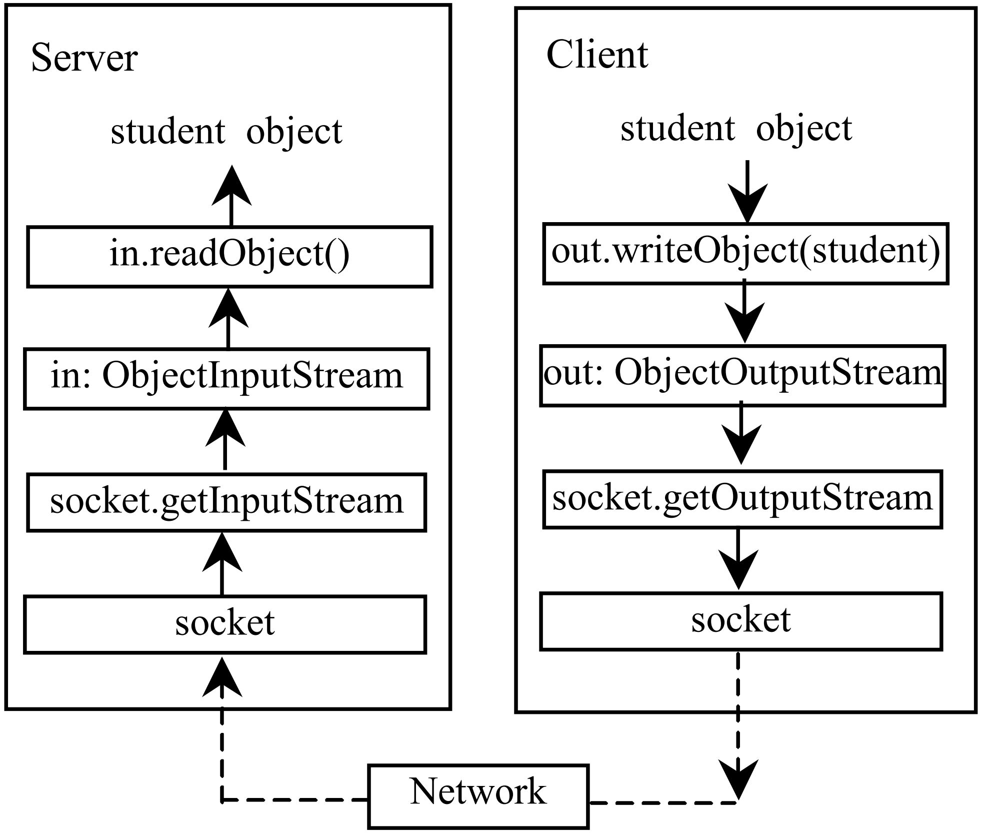
```java
new Thread( () -> {
  try {
    // Create a server socket
    ServerSocket serverSocket = new ServerSocket(8000);
    ta.appendText("MultiThreadServer started at "
      + new Date() + '\n');

    while (true) {
      // Listen for a new connection request
      Socket socket = serverSocket.accept();

      // Increment clientNo
      clientNo++;

      Platform.runLater( () -> {
        // Display the client number
        ta.appendText("Starting thread for client " + clientNo +
          " at " + new Date() + '\n');

        // Find the client's host name, and IP address
        InetAddress inetAddress = socket.getInetAddress();
        ta.appendText("Client " + clientNo + "'s host name is "
          + inetAddress.getHostName() + "\n");
        ta.appendText("Client " + clientNo + "'s IP Address is "
          + inetAddress.getHostAddress() + "\n");
      });

      // Create and start a new thread for the connection
      new Thread(new HandleAClient(socket)).start();
    }
  }
  catch(IOException ex) {
    System.err.println(ex);
  }
}).start();
```



```java
// Define the thread class for handling new connection
class HandleAClient implements Runnable {
  private Socket socket; // A connected socket

  /** Construct a thread */
  public HandleAClient(Socket socket) {
    this.socket = socket;
  }

  /** Run a thread */
  public void run() {
    try {
      // Create data input and output streams
      DataInputStream inputFromClient = new DataInputStream(
        socket.getInputStream());
      DataOutputStream outputToClient = new DataOutputStream(
        socket.getOutputStream());

      // Continuously serve the client
      while (true) {
        // Receive radius from the client
        double radius = inputFromClient.readDouble();

        // Compute area
        double area = radius * radius * Math.PI;

        // Send area back to the client
        outputToClient.writeDouble(area);

        Platform.runLater(() -> {
          ta.appendText("radius received from client: " +
            radius + '\n');
          ta.appendText("Area found: " + area + '\n');
        });
      }
    }
    catch(IOException ex) {
```

Example: Passing Objects in Network Programs

Write a program that collects student information from a client and send them to a server. Passing student information in an object.



Student Class

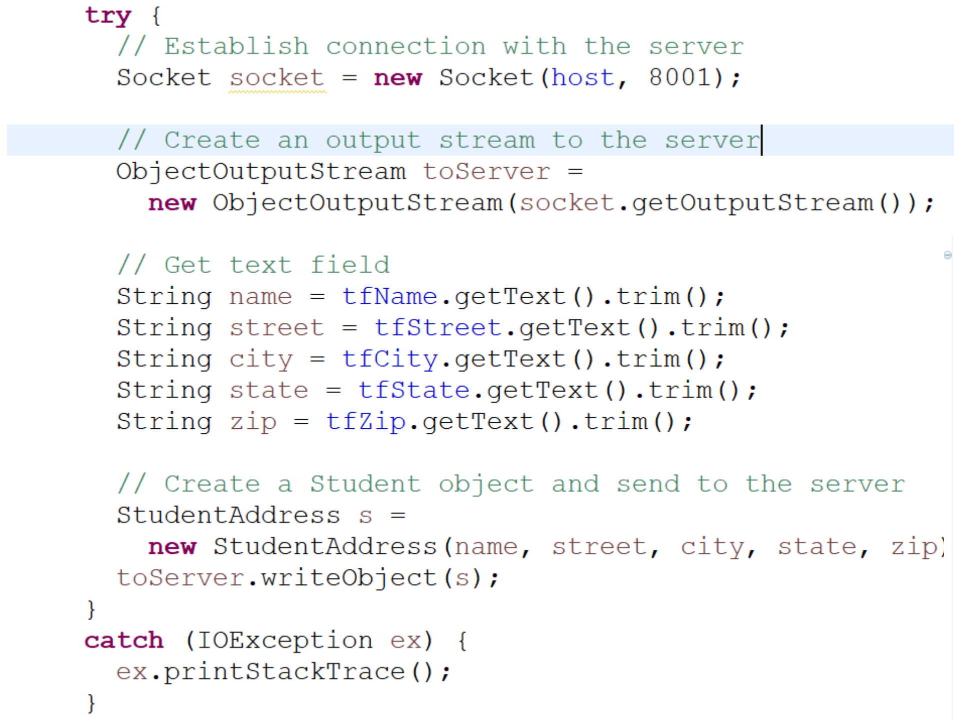
Student Sever

Student Client

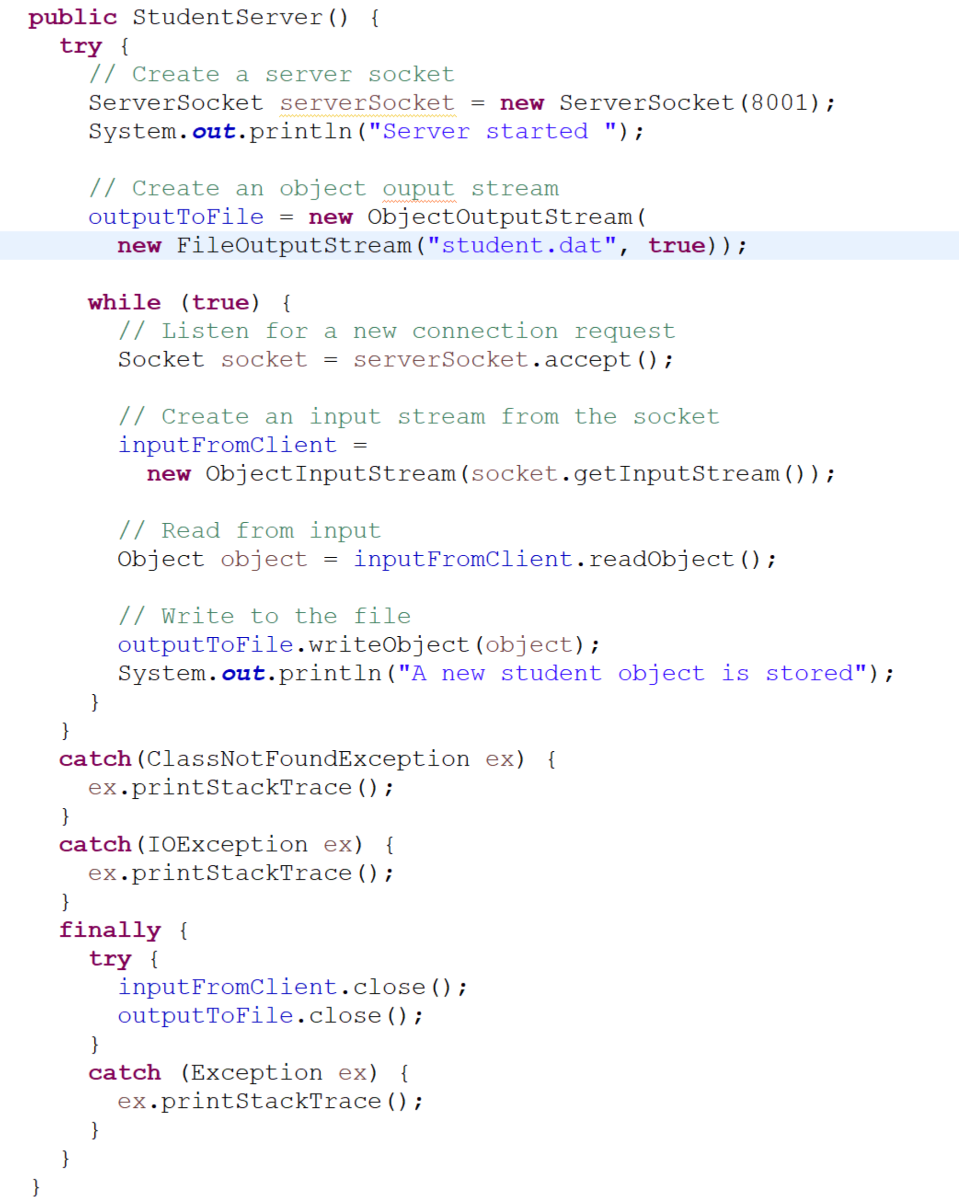
Start Server

Start Client

Note: Start the server first, then the client.



```
try {
  // Establish connection with the server
  Socket socket = new Socket(host, 8001);

  // Create an output stream to the server
  ObjectOutputStream toServer =
    new ObjectOutputStream(socket.getOutputStream());

  // Get text field
  String name = tfName.getText().trim();
  String street = tfStreet.getText().trim();
  String city = tfCity.getText().trim();
  String state = tfState.getText().trim();
  String zip = tfZip.getText().trim();

  // Create a Student object and send to the server
  StudentAddress s =
    new StudentAddress(name, street, city, state, zip)
  toServer.writeObject(s);
}
catch (IOException ex) {
  ex.printStackTrace();
}
```

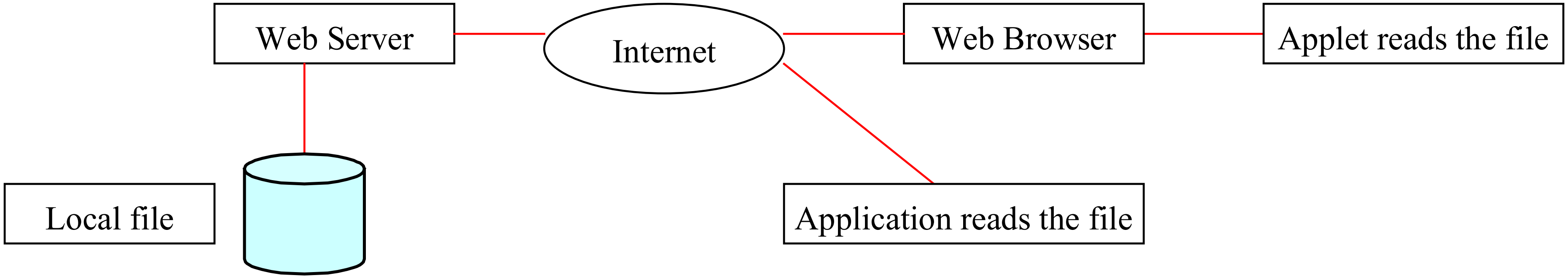


```
public StudentServer() {
  try {
    // Create a server socket
    ServerSocket serverSocket = new ServerSocket(8001);
    System.out.println("Server started ");

    // Create an object ouput stream
    outputToFile = new ObjectOutputStream(
      new FileOutputStream("student.dat", true));

    while (true) {
      // Listen for a new connection request
      Socket socket = serverSocket.accept();

      // Create an input stream from the socket
      inputFromClient =
        new ObjectInputStream(socket.getInputStream());

      // Read from input
      Object object = inputFromClient.readObject();

      // Write to the file
      outputToFile.writeObject(object);
      System.out.println("A new student object is stored");
    }
  }
  catch(ClassNotFoundException ex) {
    ex.printStackTrace();
  }
  catch(IOException ex) {
    ex.printStackTrace();
  }
  finally {
    try {
      inputFromClient.close();
      outputToFile.close();
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

Retrieving Files from Web Servers

You developed client/server applications in the previous sections. Java allows you to develop clients that retrieve files on a remote host through a Web server.

In this case, you don’t have to create a custom server program. The Web server can be used to send the files to the clients.



The URL Class

Audio and images are stored in files. The java.net.URL class can be used to identify the files on the Internet. In general, a URL (Uniform Resource Locator) is a pointer to a "resource" on the World Wide Web. A resource can be something as simple as a file or a directory. You can create a URL object using the following constructor:

```
public URL(String spec) throws MalformedURLException
```

For example, the following statement creates a URL object for http://www.sun.com:

```
try {
  URL url = new URL("http://www.sun.com");
}
catch(MalformedURLException ex) {
}
```

Creating a URL Instance

To retrieve the file, first create a URL object for the file. The java.net.URL. For example, the following statement creates a URL object for http://www.cs.armstrong.edu/liang/index.html.

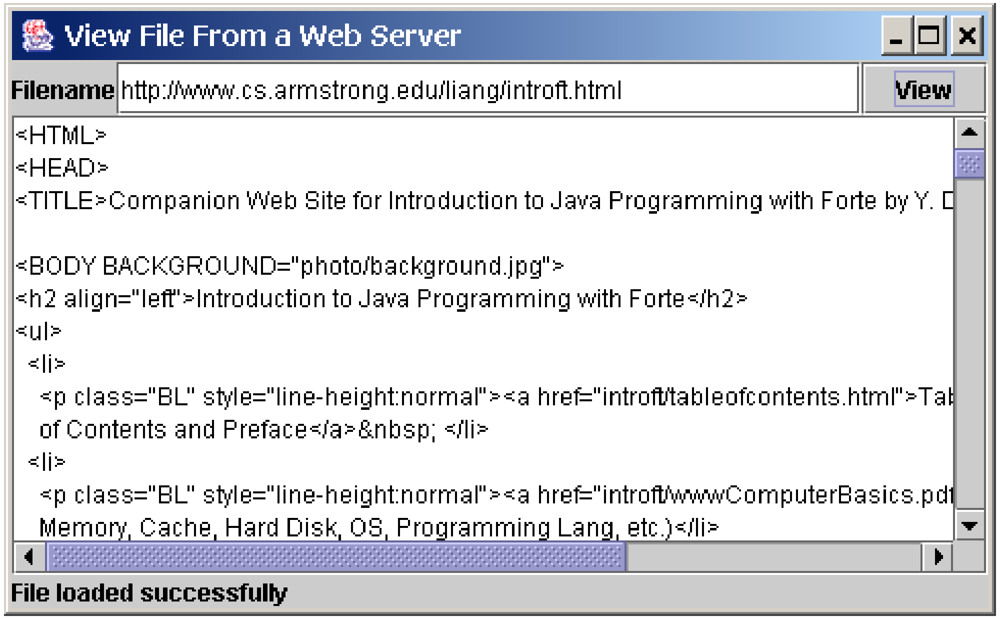
```
URL url = new URL("http://www.cs.armstrong.edu/liang/index.html");
```

You can then use the openStream() method defined in the URL class to open an input stream to the file's URL.

```
InputStream inputStream = url.openStream();
```

Example: Retrieving Remote Files

This example demonstrates how to retrieve a file from a Web server. The program can run as an application or an applet. The user interface includes a text field in which to enter the URL of the filename, a text area in which to show the file, and a button that can be used to submit an action. A label is added at the bottom of the applet to indicate the status, such as File loaded successfully or Network connection problem.



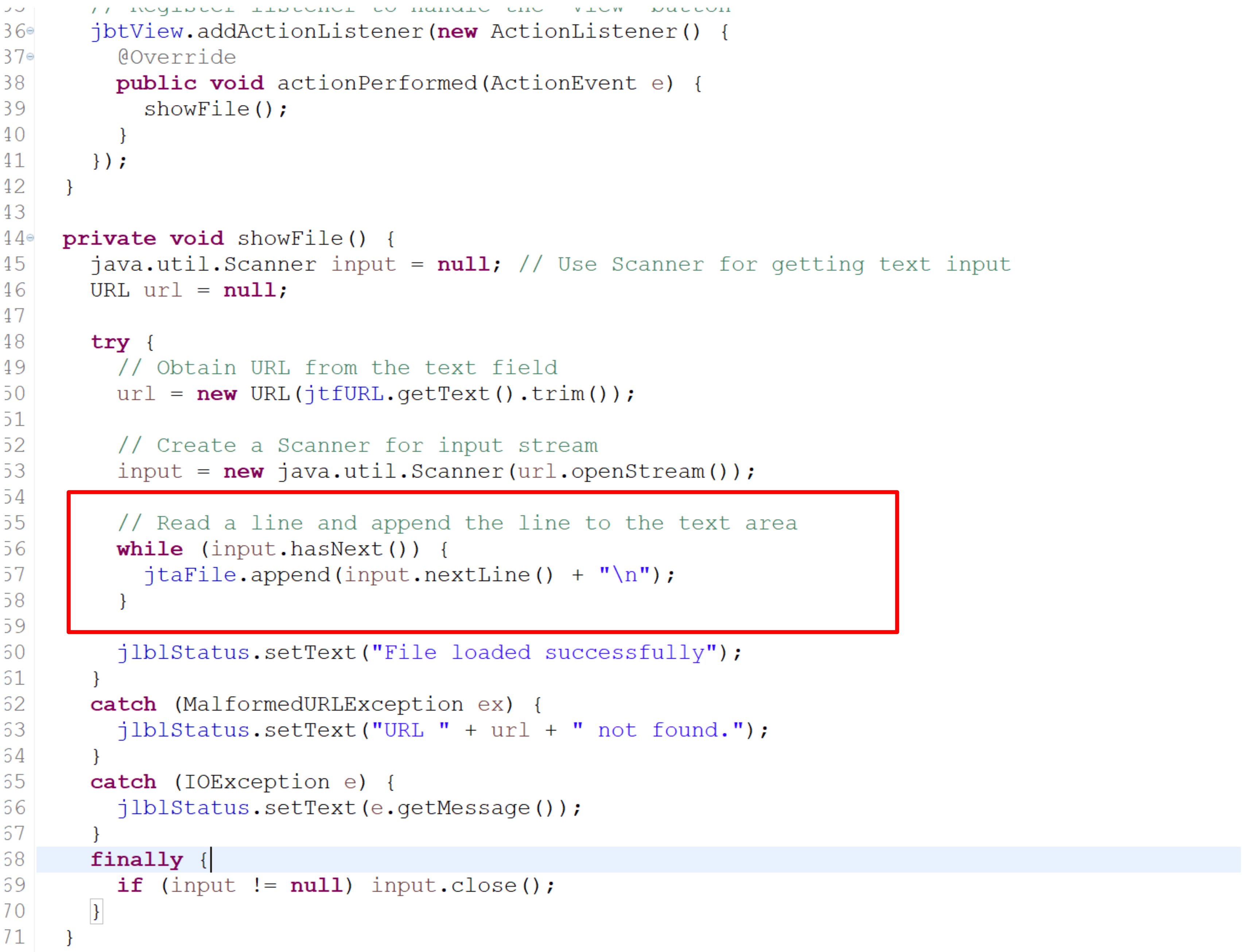
Filename http://www.cs.armstrong.edu/liang/introft.html

```
<HTML>
<HEAD>
<TITLE>Companion Web Site for Introduction to Java Programming with Forte by Y. D

<BODY BACKGROUND="photo/background.jpg">
<h2 align="left">Introduction to Java Programming with Forte</h2>
<ul>
  <li>
   <p class="BL" style="line-height:normal"><a href="introft/tableofcontents.html">Tal
   of Contents and Preface</a>&nbsp; </li>
  <li>
   <p class="BL" style="line-height:normal"><a href="introft/wwwComputerBasics.pdf
   Memory, Cache, Hard Disk, OS, Programming Lang, etc.)</li>
```

File loaded successfully

ViewRemoteFile

Run



```java
    // Register listener to handle the "View" button
    jbtView.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        showFile();
      }
    });
  }

  private void showFile() {
    java.util.Scanner input = null; // Use Scanner for getting text input
    URL url = null;

    try {
      // Obtain URL from the text field
      url = new URL(jtfURL.getText().trim());

      // Create a Scanner for input stream
      input = new java.util.Scanner(url.openStream());

      // Read a line and append the line to the text area
      while (input.hasNext()) {
        jtaFile.append(input.nextLine() + "\n");
      }

      jlblStatus.setText("File loaded successfully");
    }
    catch (MalformedURLException ex) {
      jlblStatus.setText("URL " + url + " not found.");
    }
    catch (IOException e) {
      jlblStatus.setText(e.getMessage());
    }
    finally {
      if (input != null) input.close();
    }
  }
```

JEditorPane

Swing provides a GUI component named javax.swing.JEditorPane that can be used to display plain text, HTML, and RTF files automatically. So you don't have to write code to explicit read data from the files. JEditorPane is a subclass of JTextComponent. Thus it inherits all the behavior and properties of JTextComponent.

To display the content of a file, use the setPage(URL) method as follows:

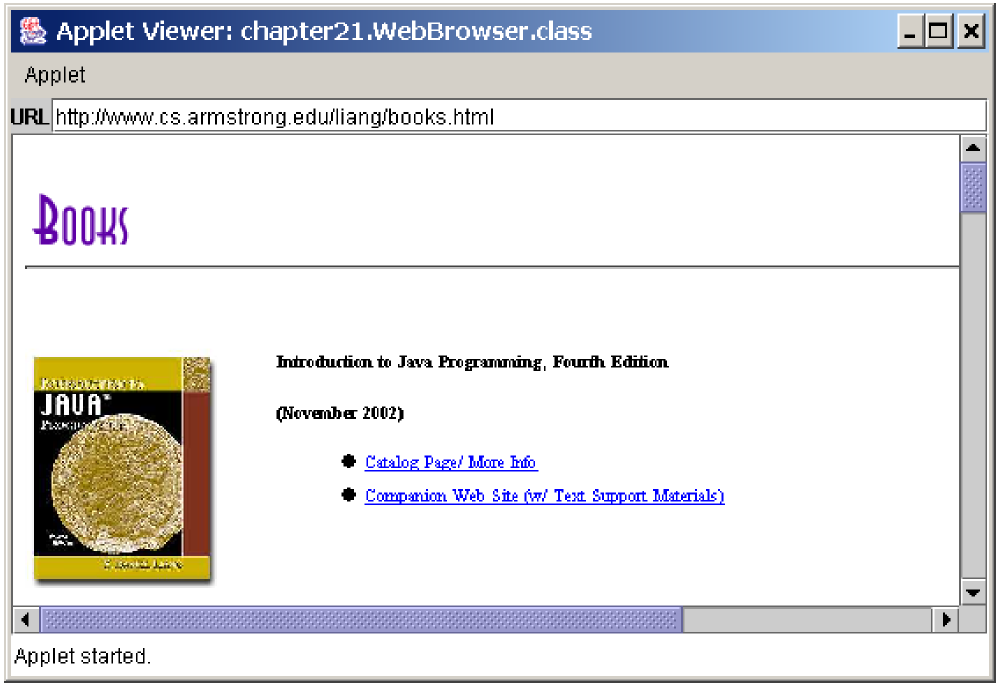
```
public void setPage(URL url) throws IOException
```

JEditorPane generates javax.swing.event.HyperlinkEvent when a hyperlink in the editor pane is clicked. Through this event, you can get the URL of the hyperlink and display it using the setPage(url) method.

Example: Creating a Web Browser

Viewing HTML Files Using the JEditorPane.

`JEditorPane` can be used to display HTML files.



http://www.cs.armstrong.edu/liang/books.html

Introduction to Java Programming, Fourth Edition

(November 2002)

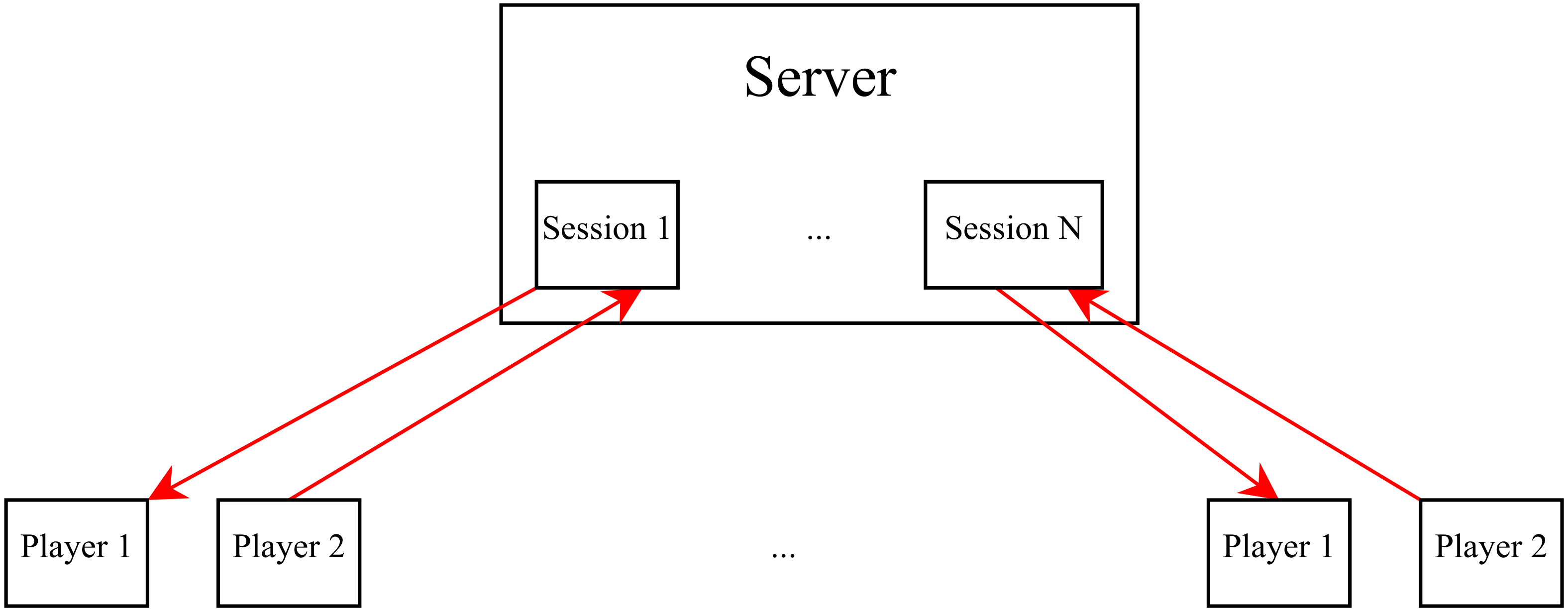
- Catalog Page/ More Info
- Companion Web Site (w/ Text Support Materials)

Applet started.

WebBrowser

Run

Case Studies: Distributed TicTacToe Games



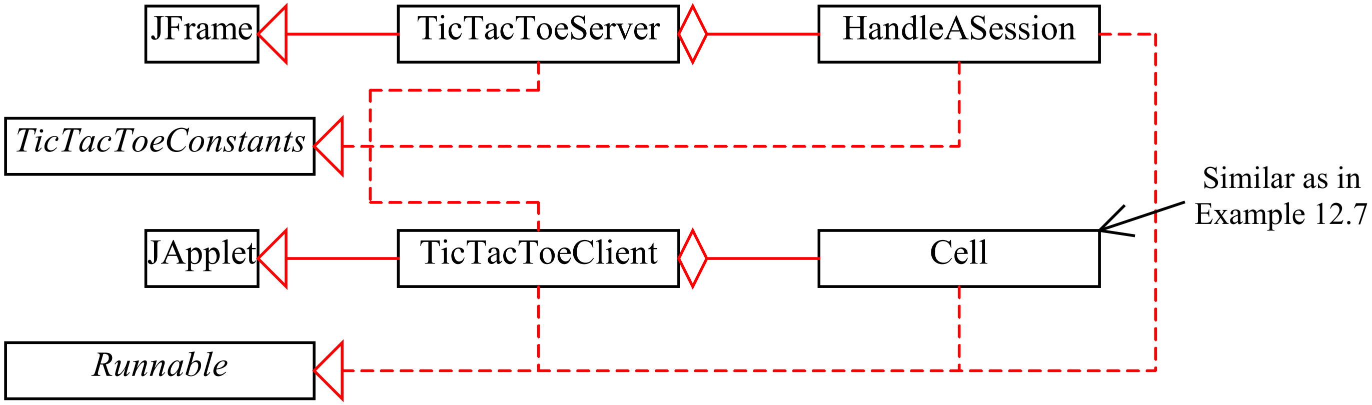
TicTacToeServer

Run Server

TicTacToeClient

Run Client

Distributed TicTacToe, cont.



Similar as in Example 12.7

TicTacToeServer

+main(args: String[]): void

TicTacToeConstants

+PLAYER1=1: int
+PLAYER2 = 2: int
+PLAYER1_WON = 1: int
+PLAYER2_WON = 2: int
+DRAW = 3: int
+CONTINUE = 4: int

HandleASession

-player1: Socket
-player2: Socket
-cell char[][]
-continueToPlay: boolean

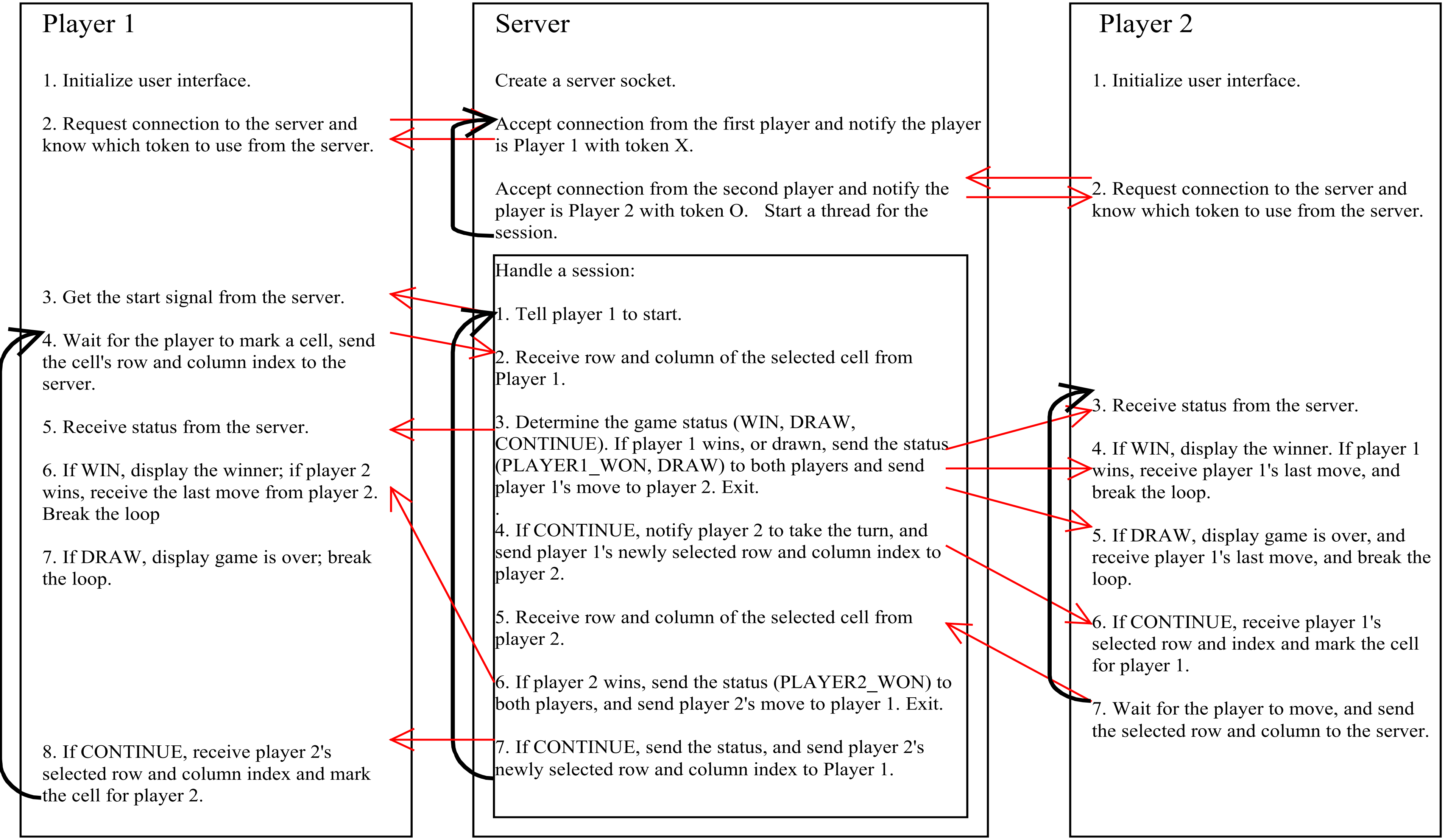
+run(): void
-isWon(): boolean
-isFull(): boolean
-sendMove(out:
  DataOuputStream, row: int,
  column: int): void

TicTacToeClient

-myTurn: boolean
-myToken: char
-otherToken: char
-cell: Cell[][]
-continueToPlay: boolean
-rowSelected: int
-columnSelected: int
-isFromServer: DataInputStream
-osToServer: DataOutputStream
-waiting: boolean

+run(): void
-connectToServer(): void
-recieveMove(): void
-sendMove(): void
-receiveInfoFromServer(): void
-waitForPlayerAction(): void

Distributed TicTacToe Game



Player 1

1. Initialize user interface.

2. Request connection to the server and know which token to use from the server.

3. Get the start signal from the server.

4. Wait for the player to mark a cell, send the cell's row and column index to the server.

5. Receive status from the server.

6. If WIN, display the winner; if player 2 wins, receive the last move from player 2. Break the loop

7. If DRAW, display game is over; break the loop.

8. If CONTINUE, receive player 2's selected row and column index and mark the cell for player 2.

Server

Create a server socket.

Accept connection from the first player and notify the player is Player 1 with token X.

Accept connection from the second player and notify the player is Player 2 with token O. Start a thread for the session.

Handle a session:

1. Tell player 1 to start.

2. Receive row and column of the selected cell from Player 1.

3. Determine the game status (WIN, DRAW, CONTINUE). If player 1 wins, or drawn, send the status (PLAYER1_WON, DRAW) to both players and send player 1's move to player 2. Exit.

.

4. If CONTINUE, notify player 2 to take the turn, and send player 1's newly selected row and column index to player 2.

5. Receive row and column of the selected cell from player 2.

6. If player 2 wins, send the status (PLAYER2_WON) to both players, and send player 2's move to player 1. Exit.

7. If CONTINUE, send the status, and send player 2's newly selected row and column index to Player 1.

Player 2

1. Initialize user interface.

2. Request connection to the server and know which token to use from the server.

3. Receive status from the server.

4. If WIN, display the winner. If player 1 wins, receive player 1's last move, and break the loop.

5. If DRAW, display game is over, and receive player 1's last move, and break the loop.

6. If CONTINUE, receive player 1's selected row and index and mark the cell for player 1.

7. Wait for the player to move, and send the selected row and column to the server.

Stream Socket vs. Datagram Socket

Stream socket

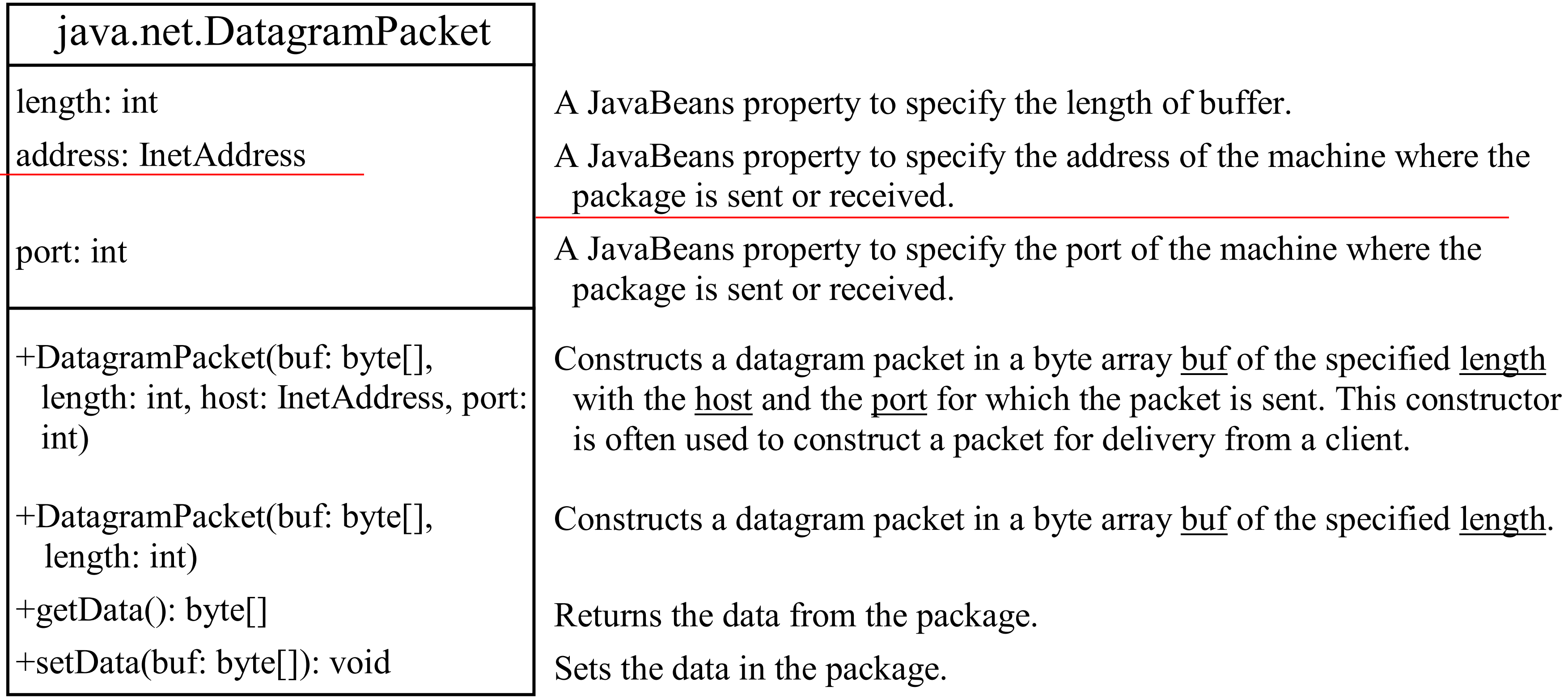
- A dedicated point-to-point channel between a client and server.
- Use TCP (Transmission Control Protocol) for data transmission.
- Lossless and reliable.
- Sent and received in the same order.

Datagram socket

- No dedicated point-to-point channel between a client and server.
- Use UDP (User Datagram Protocol) for data transmission.
- May lose data and not 100% reliable.
- Data may not received in the same order as sent.

DatagramPacket

The DatagramPacket class represents a datagram packet. Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within the packet.



java.net.DatagramPacket

length: int

address: InetAddress

port: int

+DatagramPacket(buf: byte[], length: int, host: InetAddress, port: int)

+DatagramPacket(buf: byte[], length: int)

+getData(): byte[]

+setData(buf: byte[]): void

A JavaBeans property to specify the length of buffer.

A JavaBeans property to specify the address of the machine where the package is sent or received.

A JavaBeans property to specify the port of the machine where the package is sent or received.

Constructs a datagram packet in a byte array buf of the specified length with the host and the port for which the packet is sent. This constructor is often used to construct a packet for delivery from a client.

Constructs a datagram packet in a byte array buf of the specified length.

Returns the data from the package.

Sets the data in the package.

DatagramSocket

DatagramSocket

The DatagramSocket class represents a socket for sending and receiving datagram packets. A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

Create a server DatagramSocket

To create a server DatagramSocket, use the constructor DatagramSocket(int port), which binds the socket with the specified port on the local host machine.

Create a client DatagramSocket

To create a client DatagramSocket, use the constructor DatagramSocket(), which binds the socket with any available port on the local host machine.

Sending and Receiving a DatagramSocket

Sending

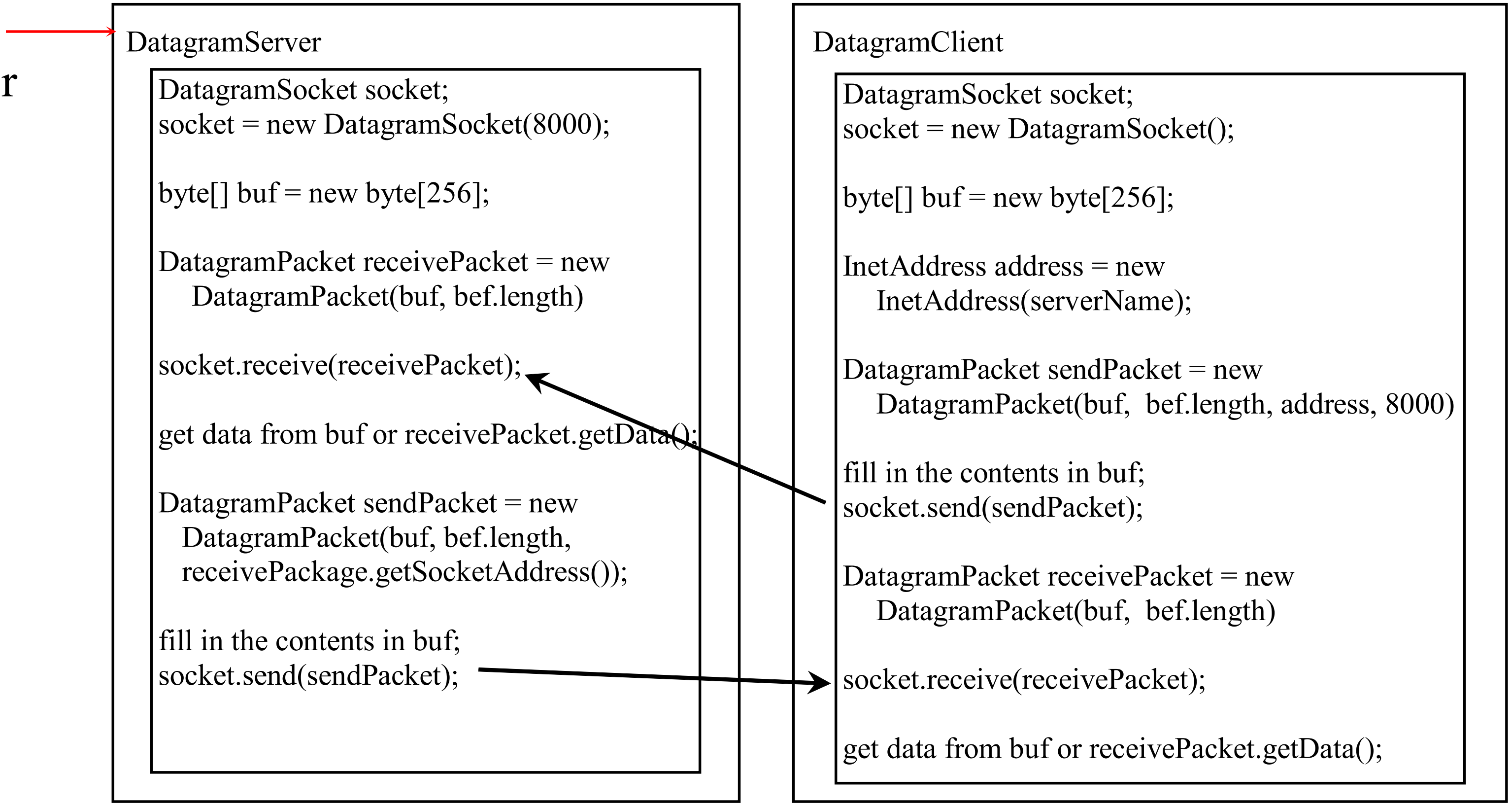
To send data, you need to create a packet, fill in the contents, specify the Internet address and port number for the receiver, and invoke the send(packet) method on a DatagramSocket.

Receiving

To receive data, create an empty packet and invoke the receive(packet) method on a DatagramSocket.

Datagram Programming

Datagram programming is different from stream socket programming in the sense that there is no concept of a <u>ServerSocket</u> for datagrams. Both client and server use <u>DatagramSocket</u> to send and receive packets.



Designate one a server

DatagramServer

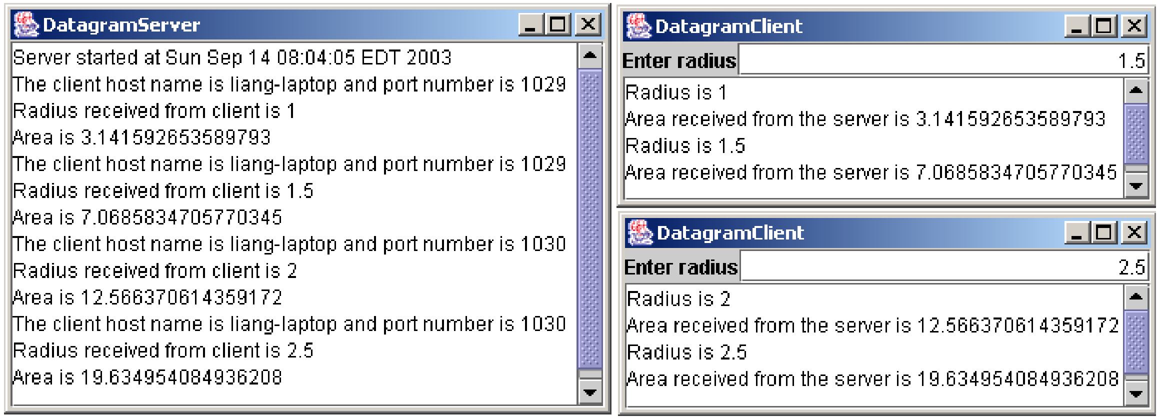
```
DatagramSocket socket;
socket = new DatagramSocket(8000);

byte[] buf = new byte[256];

DatagramPacket receivePacket = new
    DatagramPacket(buf, bef.length)

socket.receive(receivePacket);

get data from buf or receivePacket.getData();

DatagramPacket sendPacket = new
   DatagramPacket(buf, bef.length,
   receivePackage.getSocketAddress());

fill in the contents in buf;
socket.send(sendPacket);
```

DatagramClient

```
DatagramSocket socket;
socket = new DatagramSocket();

byte[] buf = new byte[256];

InetAddress address = new
    InetAddress(serverName);

DatagramPacket sendPacket = new
    DatagramPacket(buf,  bef.length, address, 8000)

fill in the contents in buf;
socket.send(sendPacket);

DatagramPacket receivePacket = new
    DatagramPacket(buf,  bef.length)

socket.receive(receivePacket);

get data from buf or receivePacket.getData();
```

Example: A Client/Server Example

Section 30.2 presents a client program and a server program using socket streams. The client sends radius to a server. The server receives the data, uses them to find the area, and then sends the area to the client. Rewrite the program using datagram sockets.



```
Server started at Sun Sep 14 08:04:05 EDT 2003
The client host name is liang-laptop and port number is 1029
Radius received from client is 1
Area is 3.141592653589793
The client host name is liang-laptop and port number is 1029
Radius received from client is 1.5
Area is 7.0685834705770345
The client host name is liang-laptop and port number is 1030
Radius received from client is 2
Area is 12.566370614359172
The client host name is liang-laptop and port number is 1030
Radius received from client is 2.5
Area is 19.634954084936208
```

Enter radius 1.5

```
Radius is 1
Area received from the server is 3.141592653589793
Radius is 1.5
Area received from the server is 7.0685834705770345
```

Enter radius 2.5

```
Radius is 2
Area received from the server is 12.566370614359172
Radius is 2.5
Area received from the server is 19.634954084936208
```

DatagramServer

DatagramClient

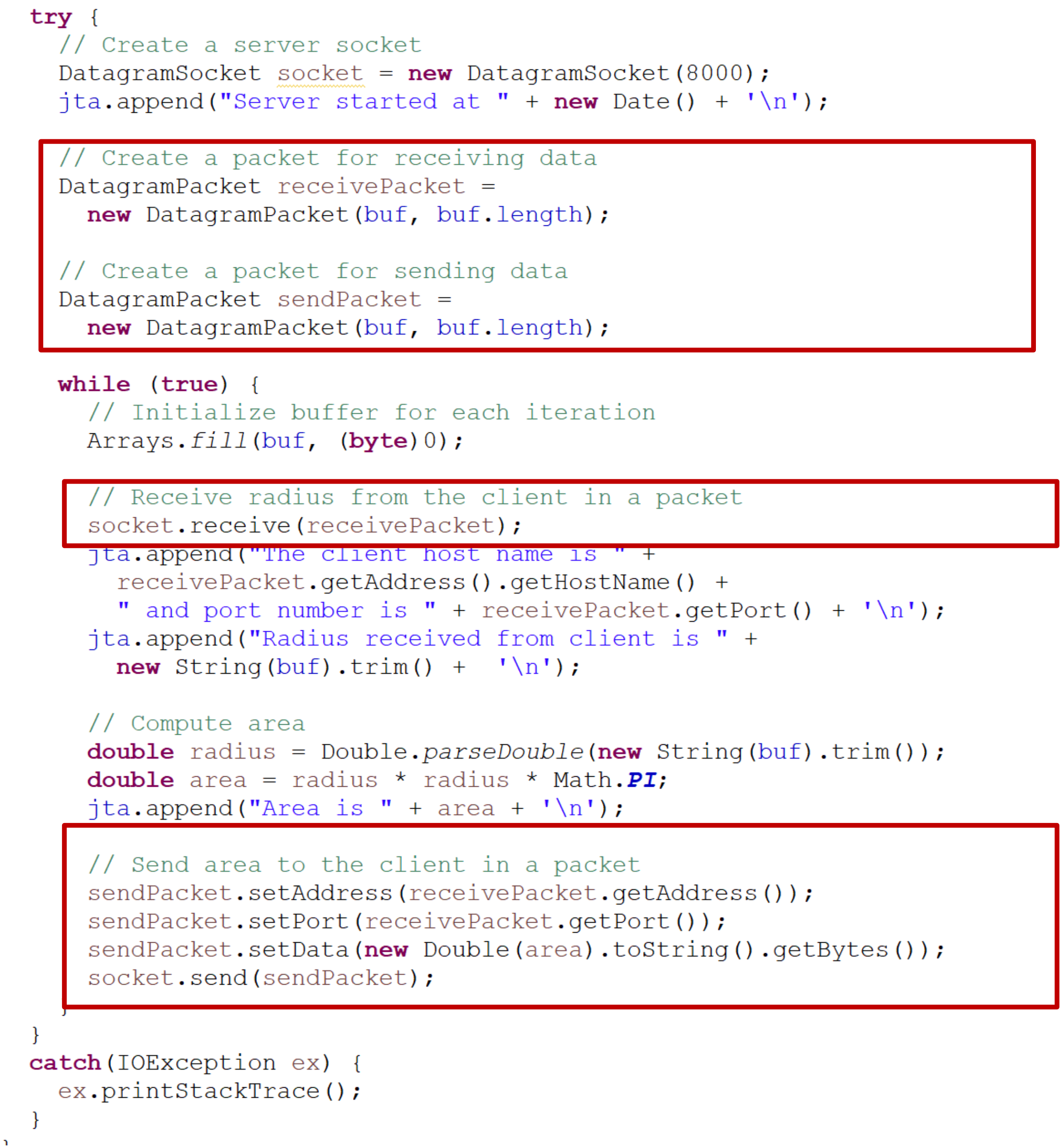
Server Code

Client Code

Start Server

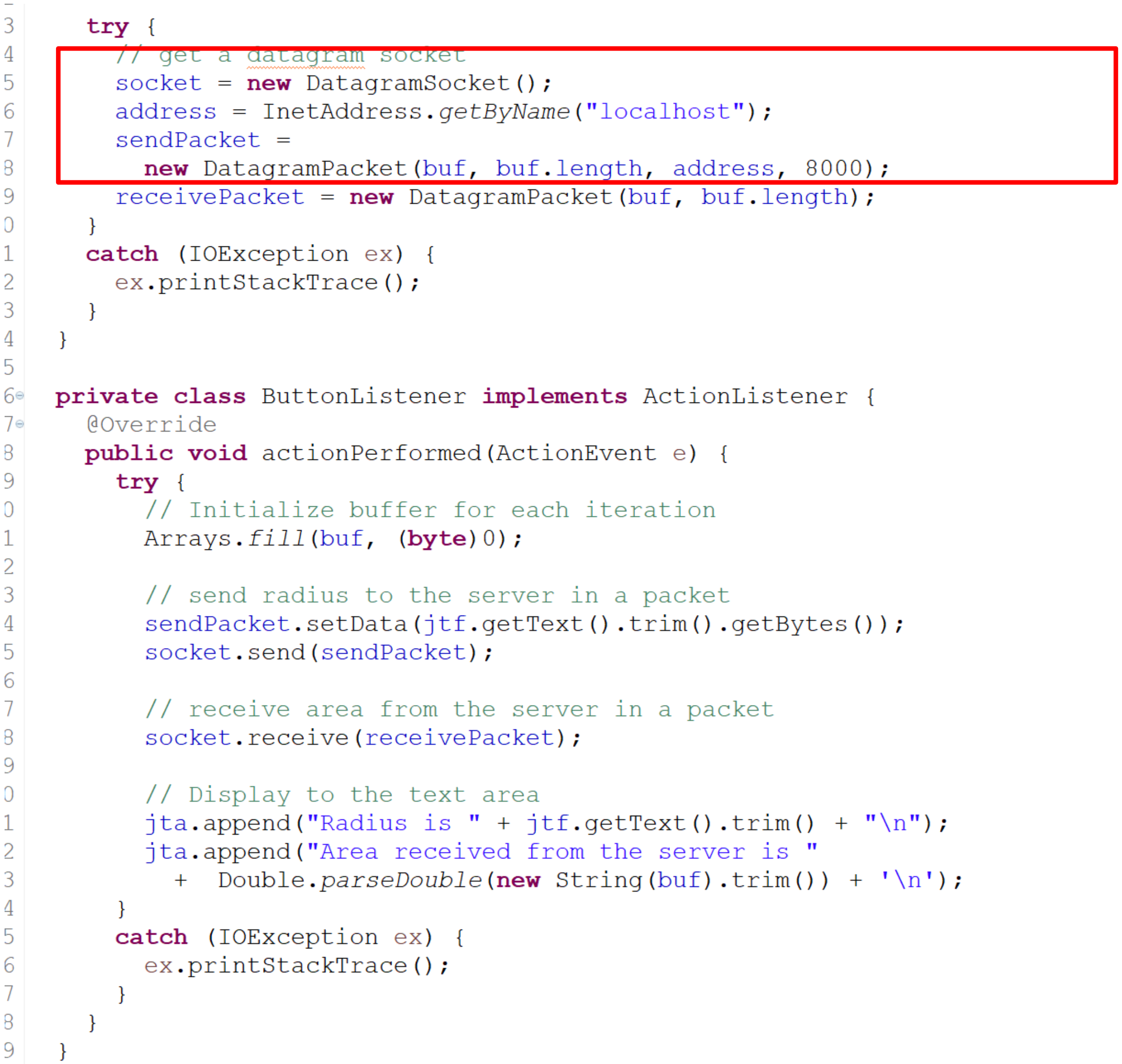
Start Client

Note: Start the server, then the client.



```
try {
  // Create a server socket
  DatagramSocket socket = new DatagramSocket(8000);
  jta.append("Server started at " + new Date() + '\n');

  // Create a packet for receiving data
  DatagramPacket receivePacket =
    new DatagramPacket(buf, buf.length);

  // Create a packet for sending data
  DatagramPacket sendPacket =
    new DatagramPacket(buf, buf.length);

  while (true) {
    // Initialize buffer for each iteration
    Arrays.fill(buf, (byte)0);

    // Receive radius from the client in a packet
    socket.receive(receivePacket);
    jta.append("The client host name is " +
      receivePacket.getAddress().getHostName() +
      " and port number is " + receivePacket.getPort() + '\n');
    jta.append("Radius received from client is " +
      new String(buf).trim() +  '\n');

    // Compute area
    double radius = Double.parseDouble(new String(buf).trim());
    double area = radius * radius * Math.PI;
    jta.append("Area is " + area + '\n');

    // Send area to the client in a packet
    sendPacket.setAddress(receivePacket.getAddress());
    sendPacket.setPort(receivePacket.getPort());
    sendPacket.setData(new Double(area).toString().getBytes());
    socket.send(sendPacket);
  }
}
catch(IOException ex) {
  ex.printStackTrace();
}
```

Server



```
  try {
    // get a datagram socket
    socket = new DatagramSocket();
    address = InetAddress.getByName("localhost");
    sendPacket =
      new DatagramPacket(buf, buf.length, address, 8000);
    receivePacket = new DatagramPacket(buf, buf.length);
  }
  catch (IOException ex) {
    ex.printStackTrace();
  }
}

private class ButtonListener implements ActionListener {
  @Override
  public void actionPerformed(ActionEvent e) {
    try {
      // Initialize buffer for each iteration
      Arrays.fill(buf, (byte)0);

      // send radius to the server in a packet
      sendPacket.setData(jtf.getText().trim().getBytes());
      socket.send(sendPacket);

      // receive area from the server in a packet
      socket.receive(receivePacket);

      // Display to the text area
      jta.append("Radius is " + jtf.getText().trim() + "\n");
      jta.append("Area received from the server is "
        +  Double.parseDouble(new String(buf).trim()) + '\n');
    }
    catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

Client