

# 26 Networking

Chapter 30@8e

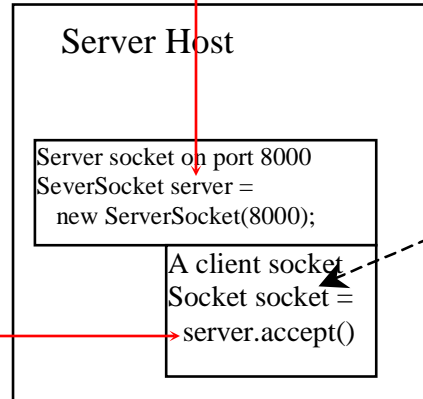
# Client/Server Communications

The server must be running when a client starts.

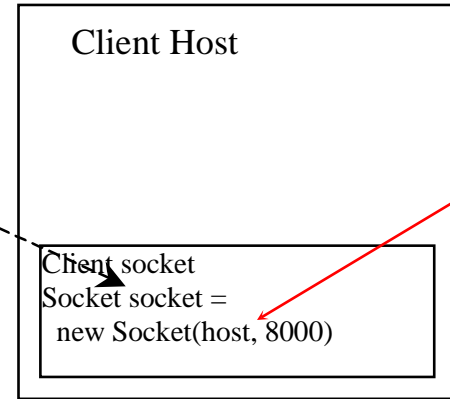
The server **waits** for a connection request from a client. To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.

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

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

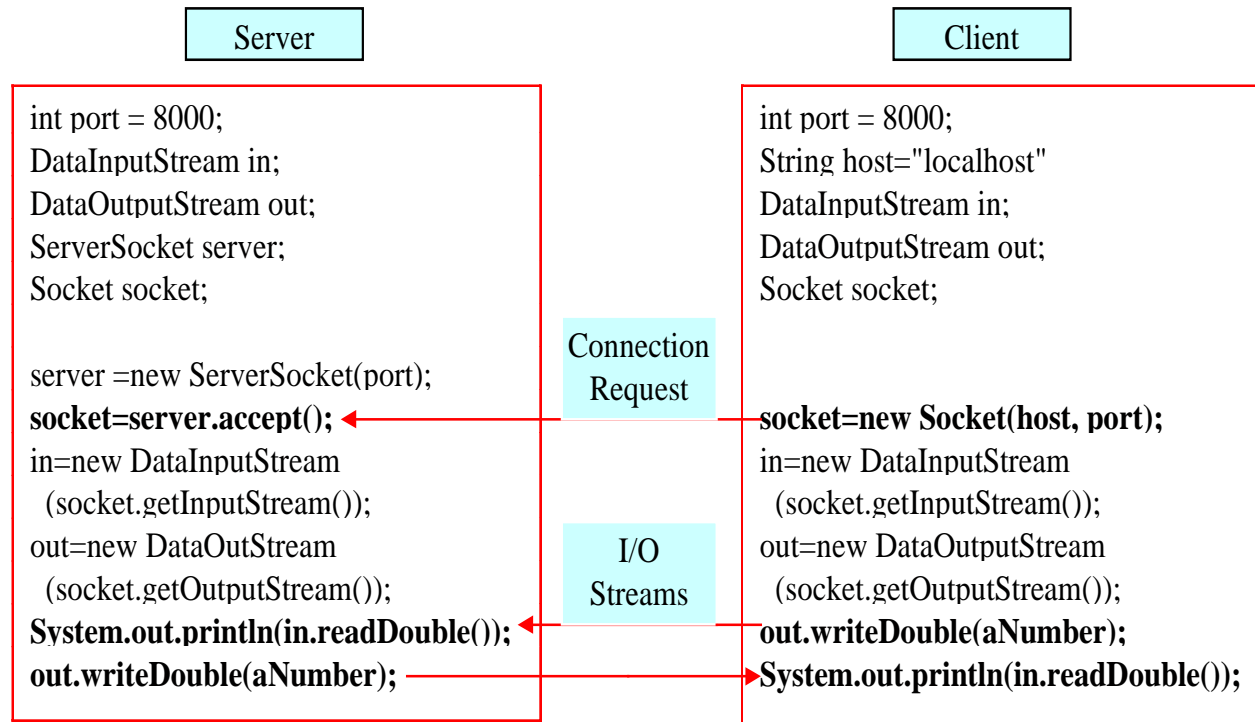


I/O Stream



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

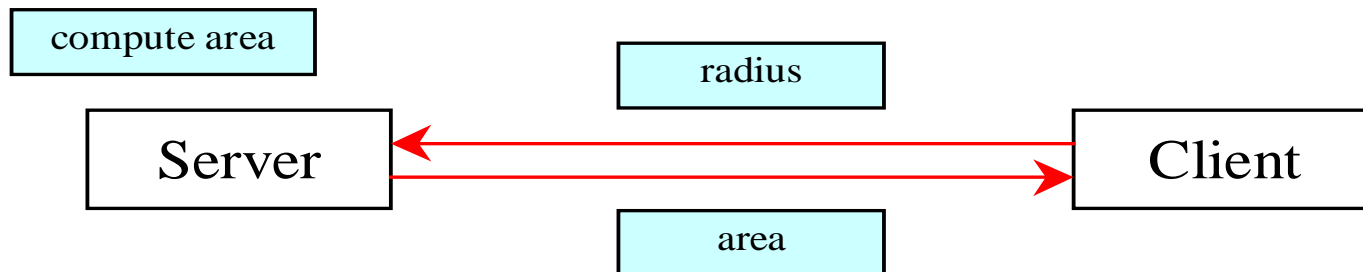
# Data Transmission through Sockets



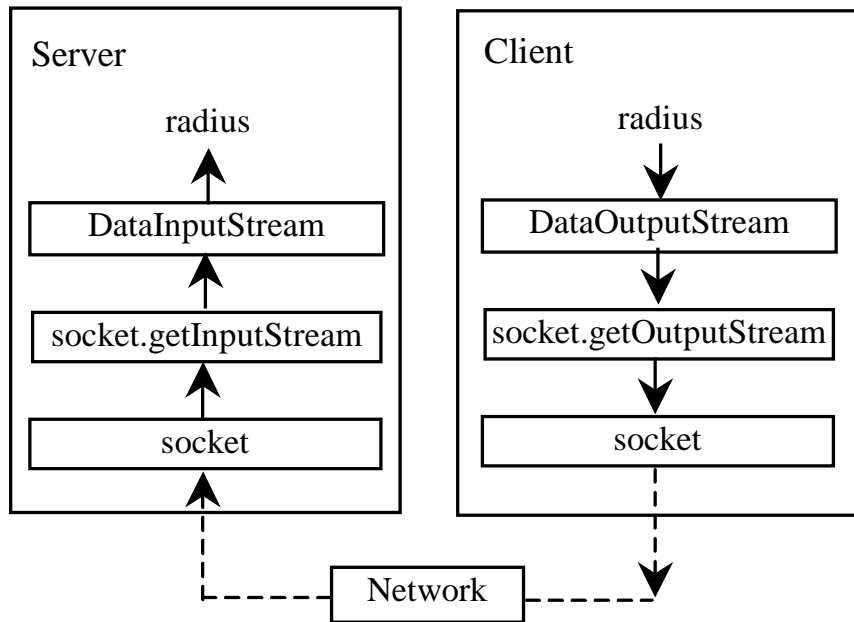
```
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputStream();
```

# A Client/Server Example

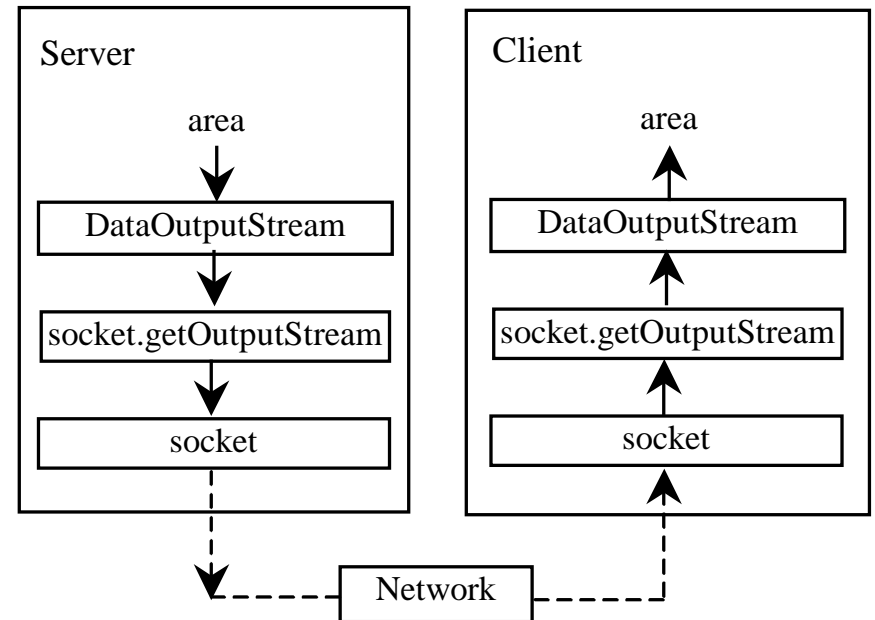
- Problem: Write a client to send data to a server. The server receives the data, uses it to produce a result, and then sends the result back to the client. The client displays the result on the console. In this example, the data sent from the client is the radius of a circle, and the result produced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



# A Client/Server Example, cont.

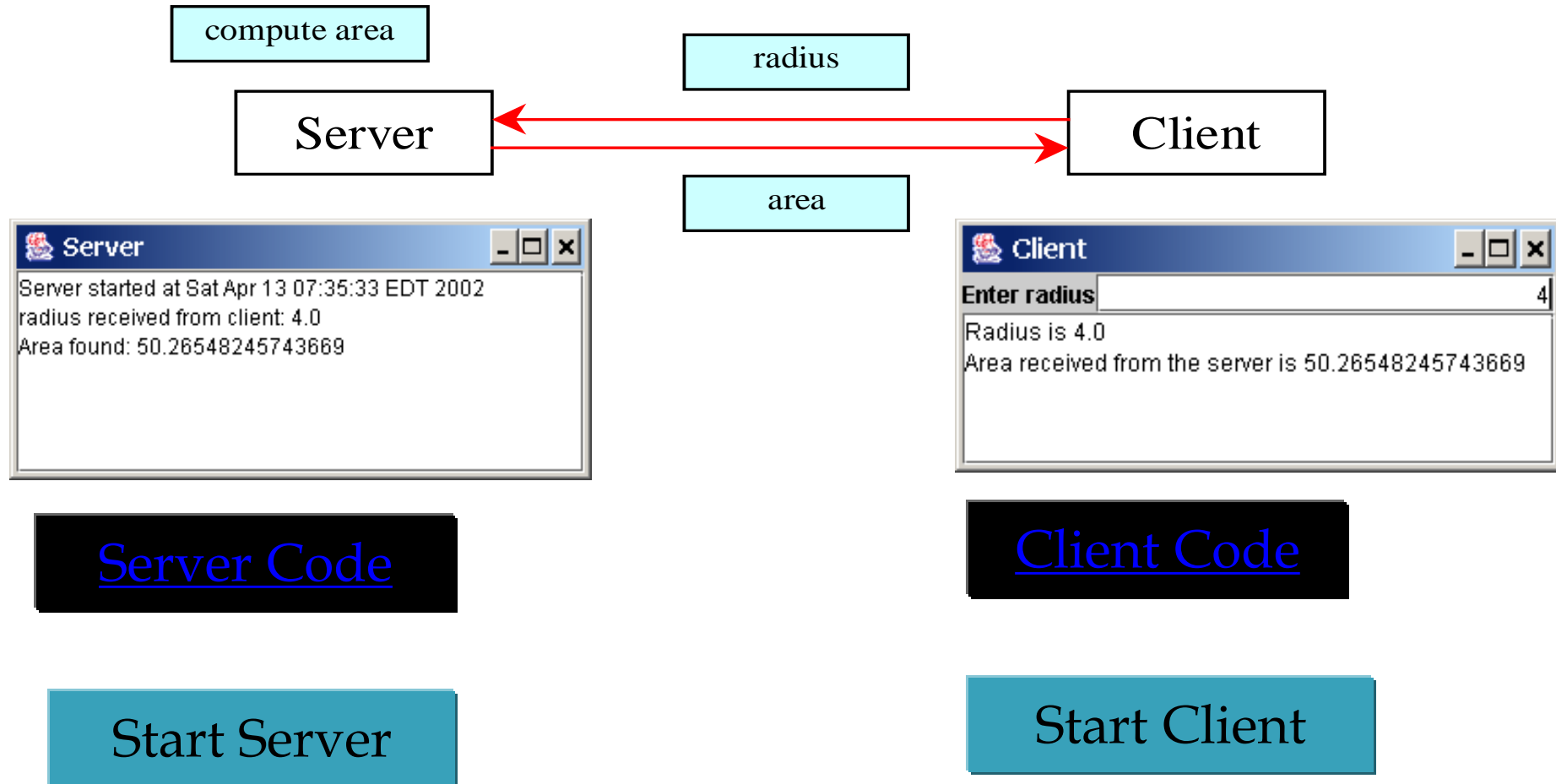


(A)



(B)

# A Client/Server Example, cont.



Note: Start the server, then the client.

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

```
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();
```

## 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();  
}
```

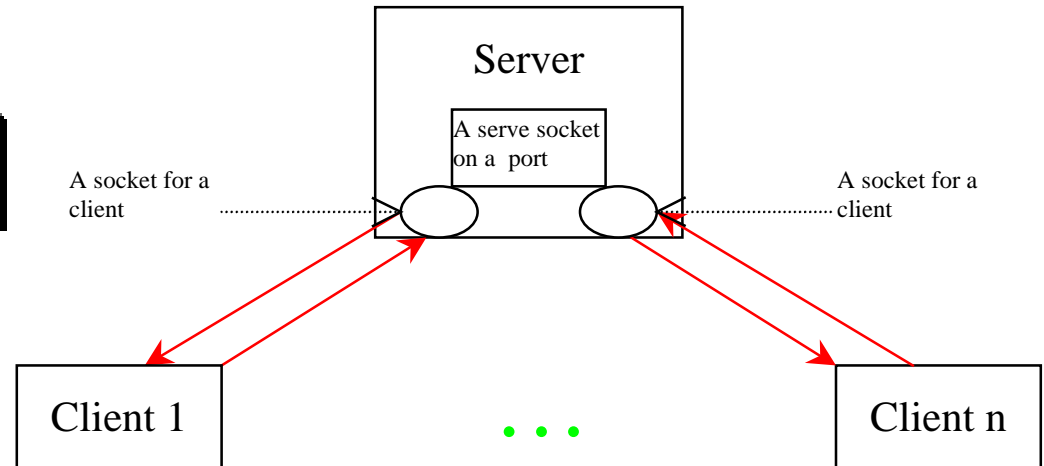
The server socket can have many connections. Each iteration of the while loop creates a new connection. Whenever a connection is established, a new thread is created to handle communication between the server and the new client; and this allows multiple connections to run at the same time.

# Example: Serving Multiple Clients

## Server for Multiple Clients

Start Server

Start Client



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

MultiThreadServer

Client

```

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();

```

```

// 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) {

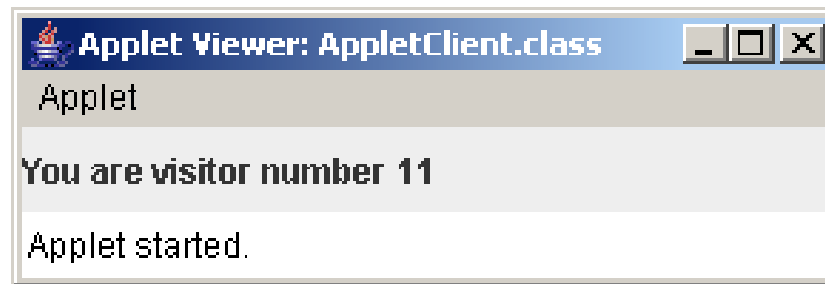
```

# Applet Clients

Due to security constraints, **applets can only connect to the host from which they were loaded**. Therefore, the HTML file must be located on the machine on which the server is running.

# Example: Creating Applet Clients

Write an applet that shows the number of visits made to a Web page. The count should be stored in a file on the server side. Every time the page is visited or reloaded, the applet sends a request to the server, and the server increases the count and sends it to the applet. The applet then displays the new count in a message, such as **You are visitor number 11**.



[CountServer](#)

[AppletClient](#)

Start Server

Start Client

```

5 public class AppletClient extends JApplet {
6     // Label for displaying the visit count
7     private JLabel jlblCount = new JLabel();
8
9     // Indicate if it runs as application
10    private boolean isStandAlone = false;
11
12    // Host name or ip
13    private String host = "localhost";
14
15    /** Initialize the applet */
16    public void init() {
17        add(jlblCount);
18
19        try {
20            // Create a socket to connect to the server
21            Socket socket;
22            if (isStandAlone)
23                socket = new Socket(host, 8001);
24            else
25                socket = new Socket(getCodeBase().getHost(), 8001);
26
27            // Create an input stream to receive data from the server
28            DataInputStream inputFromServer =
29                new DataInputStream(socket.getInputStream());
30
31            // Receive the count from the server and display it on lab
32            int count = inputFromServer.readInt();
33            jlblCount.setText("You are visitor number " + count);
34
35            // Close the stream
36            inputFromServer.close();
37        }
38        catch (IOException ex) {
39            ex.printStackTrace();
40        }
41    }
42

```

```

43    /** Run the applet as an application */
44    public static void main(String[] args) {
45        // Create a frame
46        JFrame frame = new JFrame("Applet Client");
47
48        // Create an instance of the applet
49        AppletClient applet = new AppletClient();
50        applet.isStandAlone = true;
51
52        // Get host
53        if (args.length == 1) applet.host = args[0];
54
55        // Add the applet instance to the frame
56        frame.add(applet, java.awt.BorderLayout.CENTER);
57
58        // Invoke init() and start()
59        applet.init();
60        applet.start();
61
62        // Display the frame
63        frame.pack();
64        frame.setVisible(true);
65    }
66

```

```

4 public class CountServer {
5     private RandomAccessFile raf;
6     private int count; // Count the access to the server
7
8     public static void main(String[] args) {
9         new CountServer();
10    }
11
12    public CountServer() {
13        try {
14            // Create a server socket
15            ServerSocket serverSocket = new ServerSocket(8001);
16            System.out.println("Server started ");
17
18            // Create or open the count file
19            raf = new RandomAccessFile("count.dat", "rw");
20
21            // Get the count
22            if (raf.length() == 0)
23                count = 0;
24            else
25                count = raf.readInt();
26
27            while (true) {
28                // Listen for a new connection request
29                Socket socket = serverSocket.accept();
30
31                // Create a DataOutputStream for the socket
32                DataOutputStream outputToClient =
33                    new DataOutputStream(socket.getOutputStream());
34
35                // Increase count and send the count to the client
36                count++;
37                outputToClient.writeInt(count);
38
39                // Write new count back to the file
40                raf.seek(0);
41                raf.writeInt(count);
42            }
43        }
44        catch(IOException ex) {
45            ex.printStackTrace();
46        }
47    }

```



# 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.

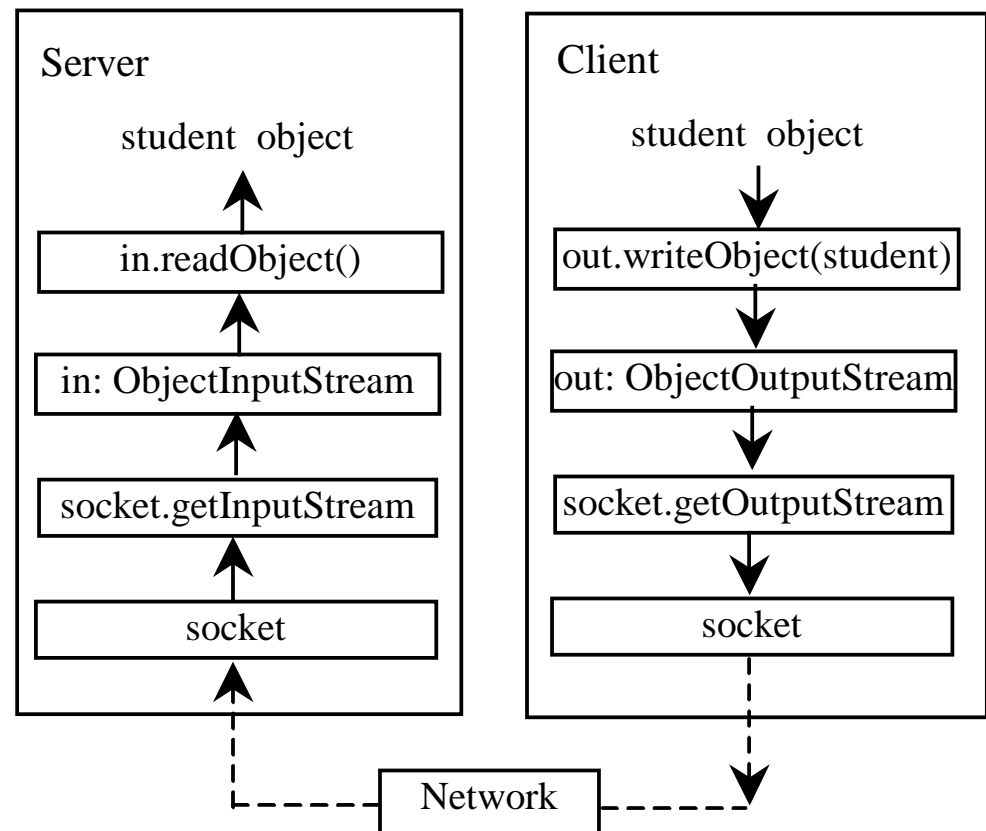
[Student Class](#)

[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 output 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());

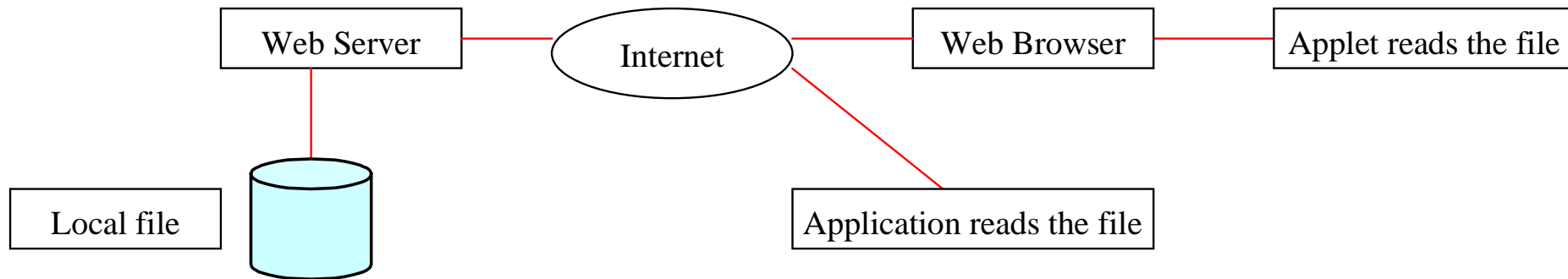
            // Read from input
            Object object = inputFromClient.readObject();

            // 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](http://java.net) 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`.

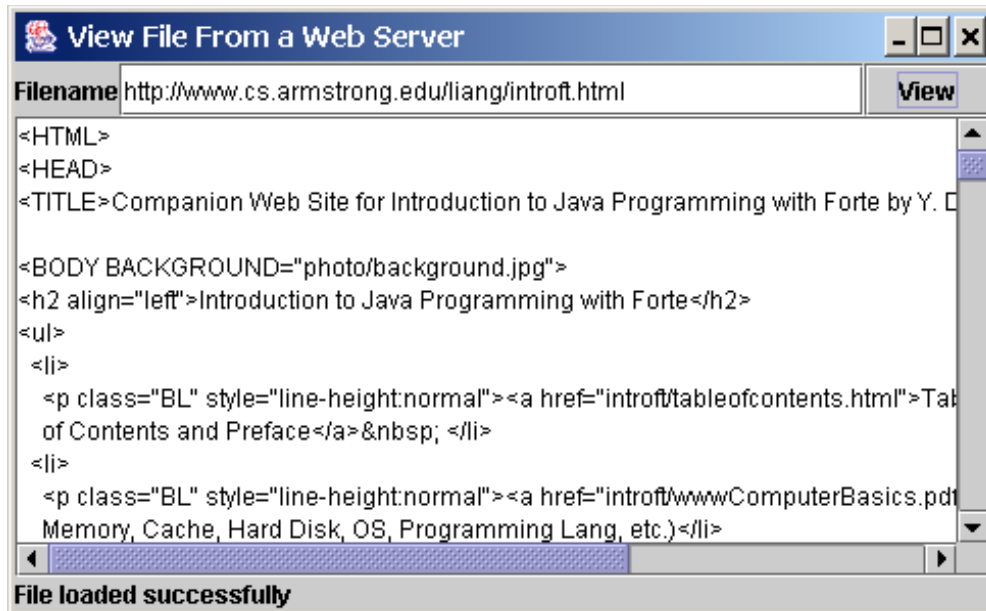
```
URL url = new URL("http://www.cs.armstrong.edu/liang/index.html");
```

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**.



[ViewRemoteFile](#)

Run

```

35 // REGISTER LISTENER TO HANDLE THE VIEW BUTTON
36 jbtView.addActionListener(new ActionListener() {
37     @Override
38     public void actionPerformed(ActionEvent e) {
39         showFile();
40     }
41 });
42 }
43
44 private void showFile() {
45     java.util.Scanner input = null; // Use Scanner for getting text input
46     URL url = null;
47
48     try {
49         // Obtain URL from the text field
50         url = new URL(jtfURL.getText().trim());
51
52         // Create a Scanner for input stream
53         input = new java.util.Scanner(url.openStream());
54
55         // Read a line and append the line to the text area
56         while (input.hasNext()) {
57             jtaFile.append(input.nextLine() + "\n");
58         }
59
60         jlblStatus.setText("File loaded successfully");
61     }
62     catch (MalformedURLException ex) {
63         jlblStatus.setText("URL " + url + " not found.");
64     }
65     catch (IOException e) {
66         jlblStatus.setText(e.getMessage());
67     }
68     finally {
69         if (input != null) input.close();
70     }
71 }

```

# 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 explicitly 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

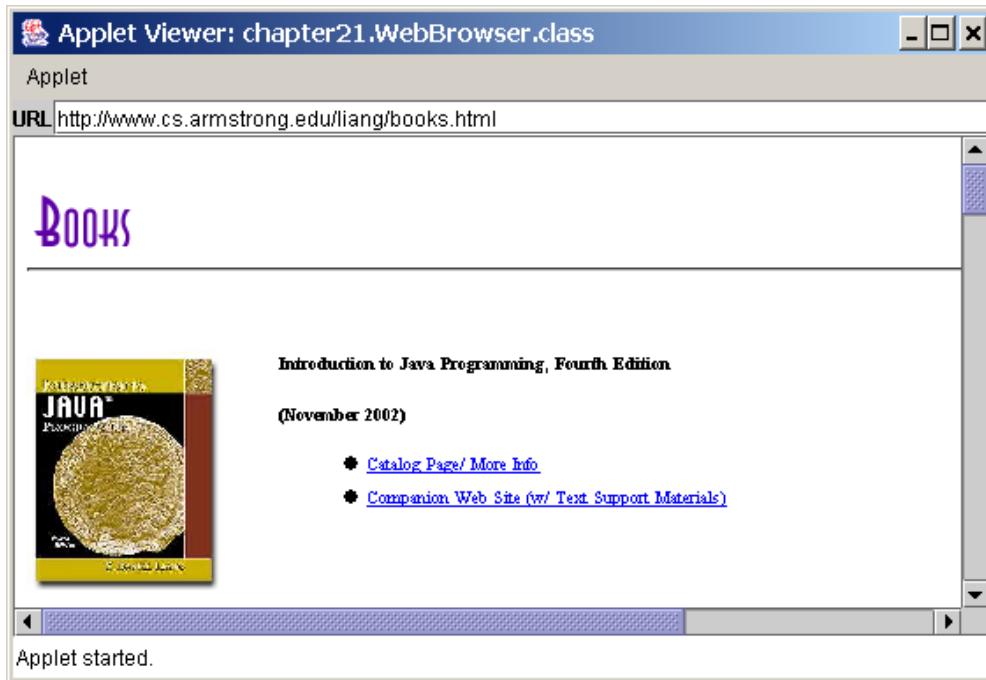
JEditorPane generates javax.swing.event.HyperlinkEvent when a hyperlink in the editor pane is clicked. Through this event, you can get the URL of the hyperlink and display it using the setPage(url) method.



# Example: Creating a Web Browser

Viewing HTML Files Using the JEditorPane.

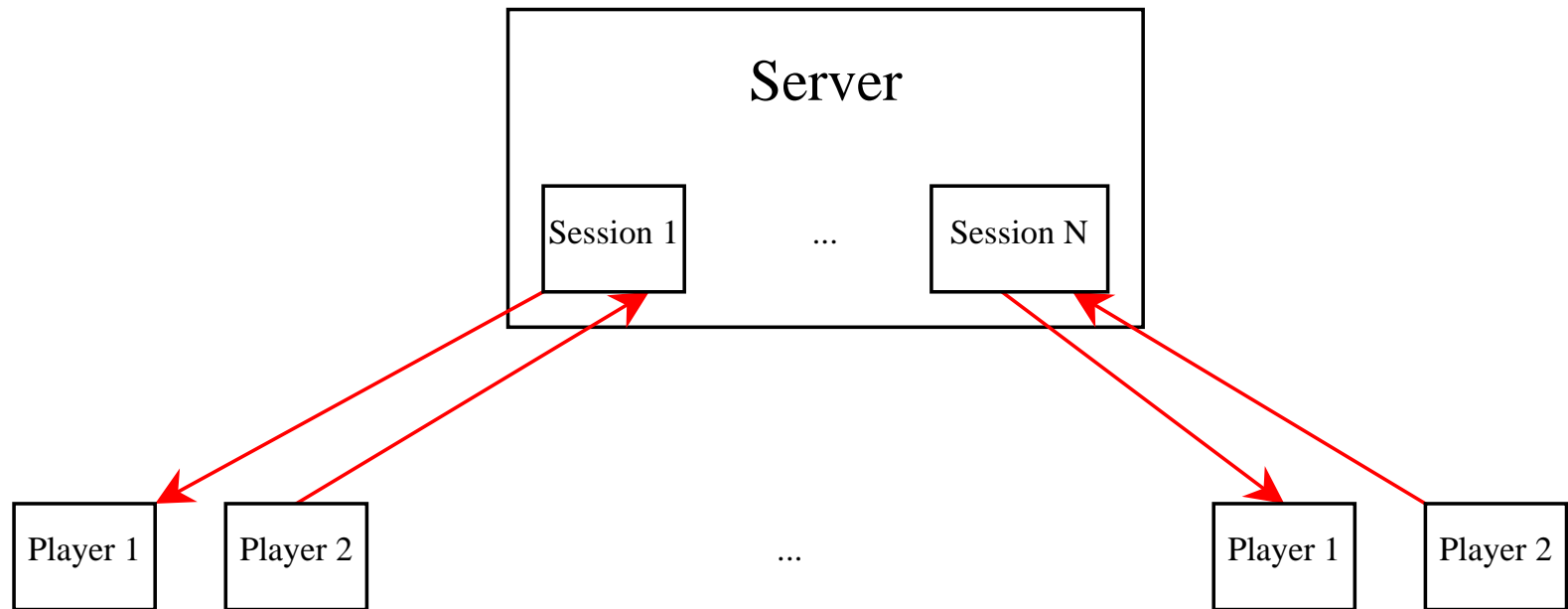
JEditorPane can be used to display HTML files.



WebBrowser

Run

# Case Studies: Distributed TicTacToe Games



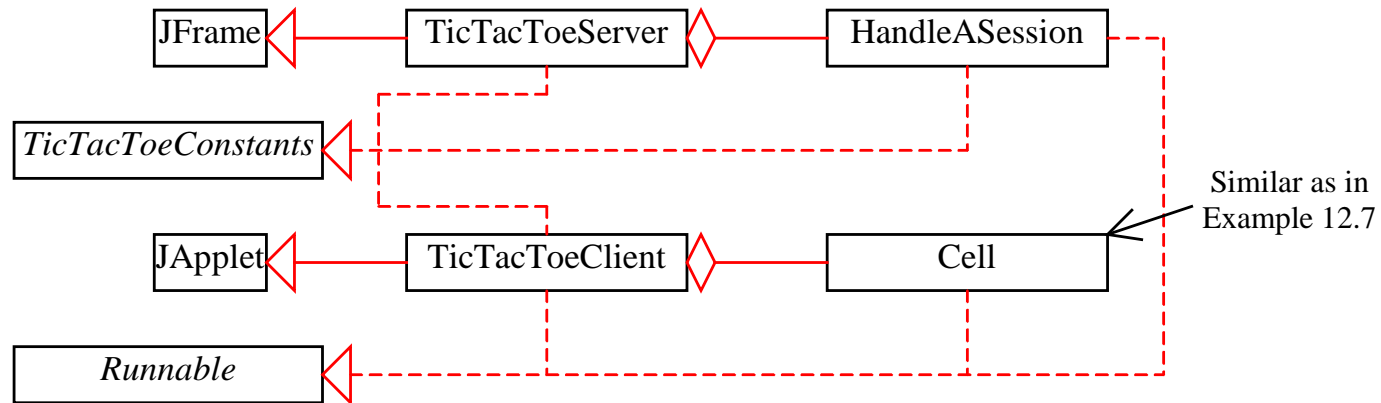
[TicTacToeServer](#)

Run Server

[TicTacToeClient](#)

Run Client

# Distributed TicTacToe, cont.



TicTacToeServer
+main(args: String[]): void

<i>TicTacToeConstants</i>
+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: DataOutputStream, 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 be 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	A JavaBeans property to specify the length of buffer.
address: InetAddress	A JavaBeans property to specify the address of the machine where the package is sent or received.
port: int	A JavaBeans property to specify the port of the machine where the package is sent or received.
+DatagramPacket(buf: byte[], length: int, host: InetAddress, port: int)	Constructs a datagram packet in a byte array <u>buf</u> of the specified <u>length</u> with the <u>host</u> and the <u>port</u> for which the packet is sent. This constructor is often used to construct a packet for delivery from a client.
+DatagramPacket(buf: byte[], length: int)	Constructs a datagram packet in a byte array <u>buf</u> of the specified <u>length</u> .
+getData(): byte[]	Returns the data from the package.
+setData(buf: byte[]): void	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

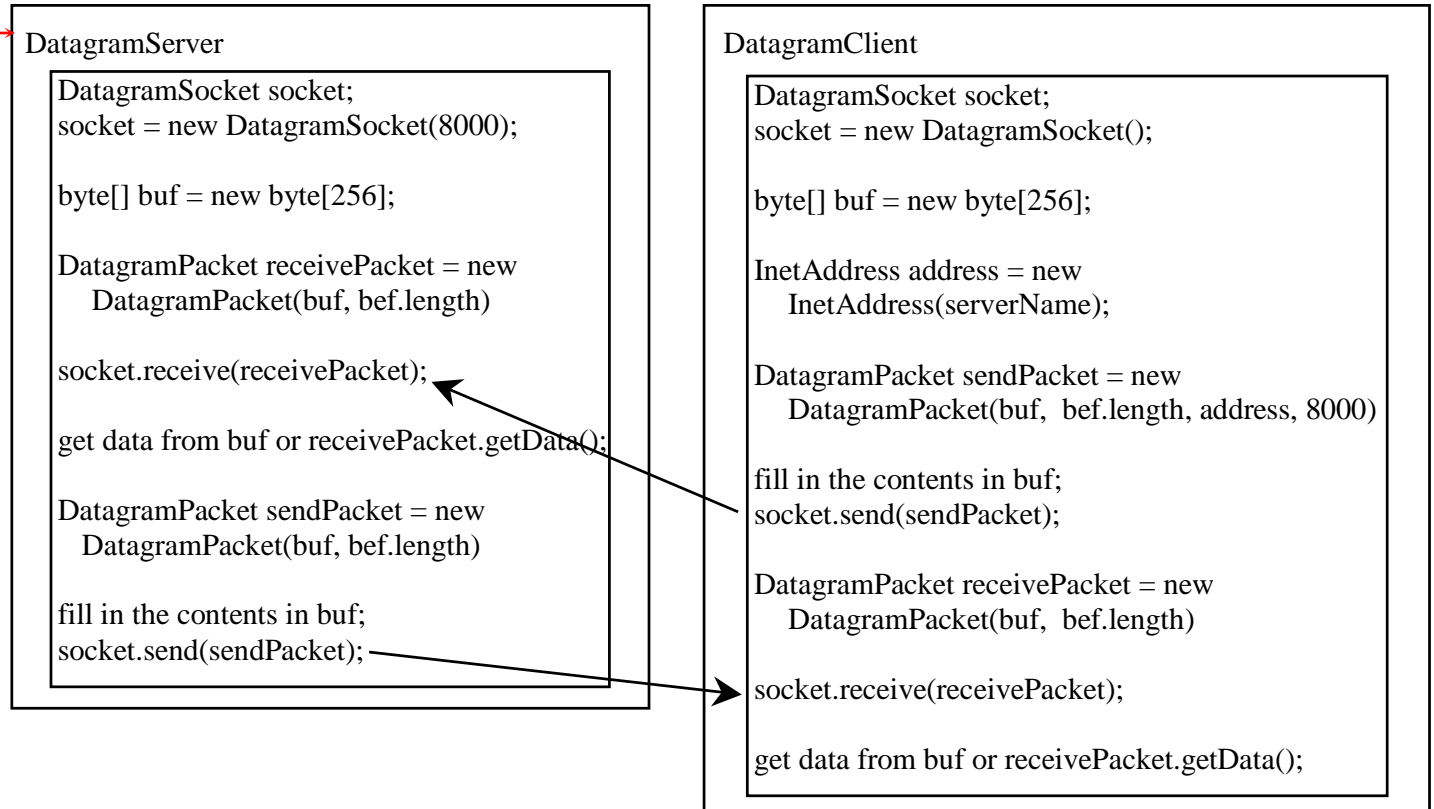
To receive data, create an empty packet and invoke the receive(packet) method on a DatagramSocket.



# Datagram Programming

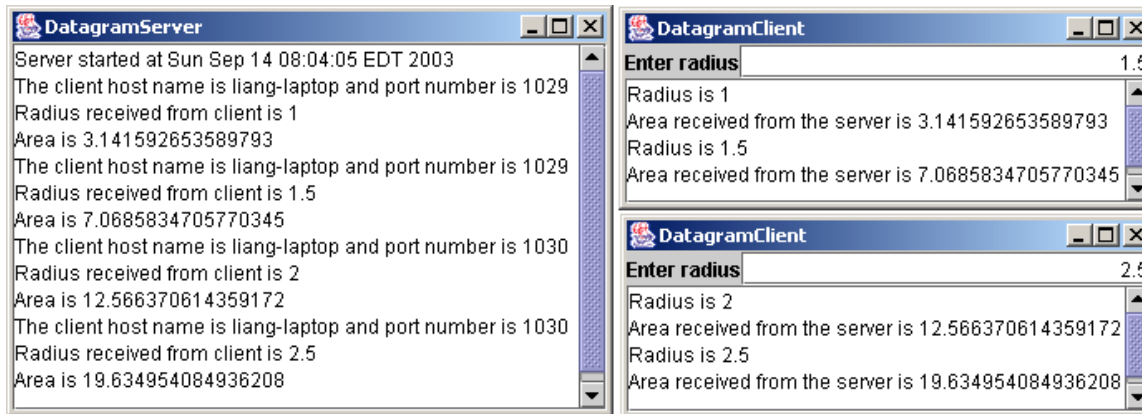
Datagram programming is different from stream socket programming in the sense that there is no concept of a ServerSocket for datagrams. Both client and server use DatagramSocket to send and receive packets.

Designate one a server



# 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.



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

```

-
3      try {
4          // get a datagram socket
5          socket = new DatagramSocket();
6          address = InetAddress.getByName("localhost");
7          sendPacket =
8              new DatagramPacket(buf, buf.length, address, 8000);
9          receivePacket = new DatagramPacket(buf, buf.length);
10     }
11     catch (IOException ex) {
12         ex.printStackTrace();
13     }
14 }
15
16 private class ButtonListener implements ActionListener {
17     @Override
18     public void actionPerformed(ActionEvent e) {
19         try {
20             // Initialize buffer for each iteration
21             Arrays.fill(buf, (byte)0);
22
23             // send radius to the server in a packet
24             sendPacket.setData(jtf.getText().trim().getBytes());
25             socket.send(sendPacket);
26
27             // receive area from the server in a packet
28             socket.receive(receivePacket);
29
30             // Display to the text area
31             jta.append("Radius is " + jtf.getText().trim() + "\n");
32             jta.append("Area received from the server is "
33                 + Double.parseDouble(new String(buf).trim()) + '\n');
34         }
35         catch (IOException ex) {
36             ex.printStackTrace();
37         }
38     }
39 }

```