COP 3330: Object-Oriented Programming
Summer 2011

Java Networking

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2011

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Java Networking

- Networking is tightly integrated in Java. The Java API provides the classes for creating sockets to facilitate program communications over the Internet.
- Sockets are the endpoints of logical connections between two hosts and can be used to send and receive data.
- Java treats socket communication similar to the way it treats I/O operations; thus applications can read from or write to sockets as easily as they can read from or write to files.

Java Networking

- The Internet Protocol (IP) is a low-level protocol for delivering data from one computer to another across the Internet in packets. Two higher-level protocols used in conjunction with IP are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).
- TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees that all packets sent are delivered in the same order in which they were sent.
- UDP is a low-overhead, connectionless, host-to-host protocol that allows a datagram to be sent from one host to another. No connection is established and no guarantees are offered.

Java Networking

- Network programming typically involves a server and one or more clients. The client sends requests to the server, and the server responds to those requests.
- The client begins by attempting to establish a connection to the server. The server can accept or deny the connection. Once a connection is established, the client and the server communicate through sockets.

Server Sockets

- To establish a server, you need to create a server socket and attach it to a port, which is where the server will listen for connections.
- The port identifies the TCP service on the socket.
- Port numbers range from 0 to 65536 (2^{16}), but most OS reserve port numbers 0 to 1024 for privileged services. For example, email servers run on port 25, and the Web server usually runs on port 80.
- You can choose any port number that is not currently used by any other process.

Server Sockets

- The following statement creates a server socket named `serverSocket`:

```
ServerSocket serverSocket = new ServerSocket(portNumber);
```

Example:

```
ServerSocket server = new ServerSocket(8000);
```

- Attempting to create a server socket on a port already in use would cause a `java.net.BindException`.

Server Sockets

- Once a server socket is created, the server can use the following statement to listen for connection attempts:

```
Socket socket = ServerSocket.accept();
```

- This statement waits until a client connects to the server socket. How this is implemented is somewhat system dependent, in general the server blocks itself until a connection is attempted at which time the server unblocks and begins to deal with the connection attempt.

Client Sockets

- A client will issue the following statement to request a connection to a server:

```
Socket socket = new Socket(serverName, port);
```

- This statement opens a socket so that the client can begin communications with the server.

  `serverName` is the server's Internet host name or IP address.

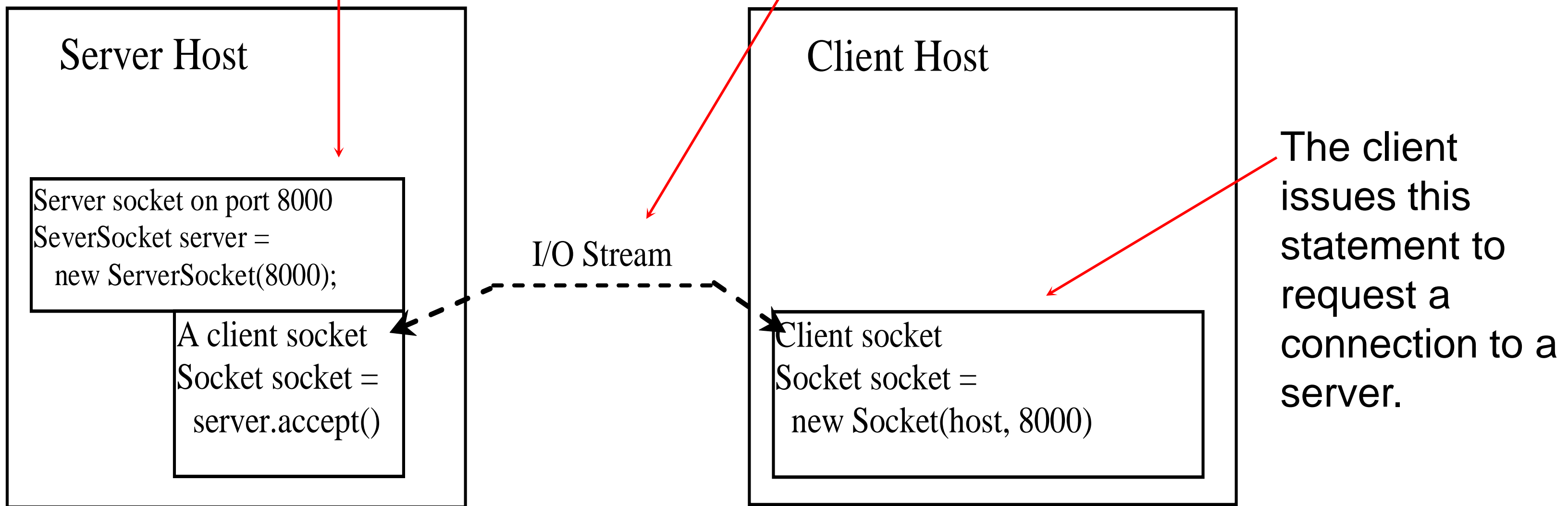
  `port` is the port number on which the socket is to be created.

- When you create a socket using the host name, the JVM asks the DNS to translate the host name into an IP address.
- The host name `localhost` or the IP address `127.0.0.1` refer to the machine on which a client is running.

Overview of Client-Server Operations

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.



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

Overview of Client-Server Operations

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);
```

Connection Request

I/O Streams

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();
```

An Overview Of Client-Server Operations

- To get an input stream and an output stream, you'll use the `getInputStream()` and `getOutputStream()` methods on a socket object (see bottom of previous slide).
- The `InputStream` and `OutputStream` streams are used to read or write bytes.
- You can use the `DataInputStream`, `DataOutputStream`, `BufferedReader`, and `PrintWriter` to wrap the `InputStream` and `OutputStream` to read or write data, such as int, double, or String.

An Overview Of Client-Server Operations

- The following statements, for instance, create a `DataInputStream`, `input`, and a `DataOutputStream`, `output`, to read and write primitive data values:

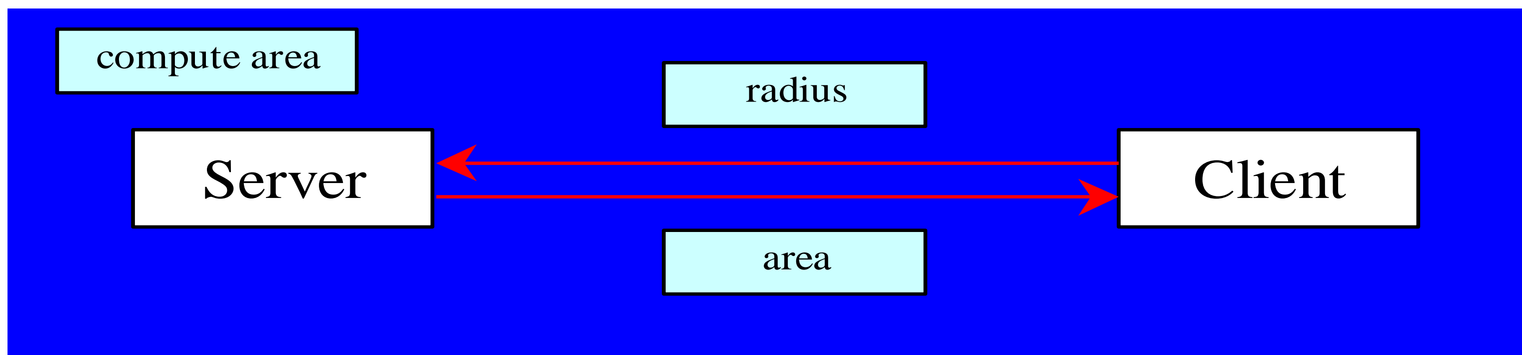
```
DataInputStream input =
    new DataInputStream(socket.getInputStream());

DataOutputStream output =
    new DataOutputStream(socket.getOutputStream());
```

- The server can use `input.readDouble()` to receive a `double` value from the client, and `output.writeDouble(d)` to send `double` value `d` to the client.

A Client-Server Example

- Let's develop a complete client/server solution to a problem.
- The client will send data to a server, the server will receive the data from the client, process the data and then send a result back to the client. The client will display the results to the user.
- In this example, the client will send data that represents the radius of a circle. The server will return to the client the area of a circle that has that radius.



A Client-Server Example

- The client will send a radius value through a DataOutputStream on the output stream socket, and the server will receive the radius through the DataInputStream on the input stream socket, as shown on the next page.
- The server will compute the area of the circle and send this result to the client through a DataOutputStream on the output stream socket, and the client receives the calculated area through a DataInputStream on the input stream socket as shown on the next page.

A Client-Server Example

Server

radius

DataInputStream

socket.getInputStream

socket

Client

radius

DataOutputStream

socket.getOutputStream

socket

Network

Server

area

DataOutputStream

socket.getOutputStream

socket

Client

area

DataOutputStream

socket.getOutputStream

socket

Network

Client Sends Request To Server

Client Receives Response From Server

FindColnagos.java | bikes.txt | tripexpenses.txt | TripExpenses.java | *Client.java

```java
//Server.java
//Networking Notes - Summer 2011
//Server receives a radius from a client and calculates area of circle.
//MJL 7/26/2011

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;

public class Server extends JFrame {
  // Text area for displaying contents
  private JTextArea jta = new JTextArea();

  public static void main(String[] args) {
    new Server();
  }//end main

  public Server() {
    // Place text area on the frame
    setLayout(new BorderLayout());
    add(new JScrollPane(jta), BorderLayout.CENTER);

    setTitle("Server");
    setSize(300, 150);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
```

Server.java

FindColnagos.java | bikes.txt | tripexpenses.txt | TripExpenses.java | *Client.java | *Server.java | »71

Server.java (continued)

```
      try {
        // Create a server socket
        ServerSocket serverSocket = new ServerSocket(8000);
        jta.append("Server started at " + new Date() + '\n');
        while (true) {
            // 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());
            jta.append("New client connection established...\n");
            // 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);
            jta.append("Radius received from client: " + radius + '\n');
            jta.append("Area found: " + area + '\n');
            jta.append("Calculated area sent to client \n");
        }//end while loop
      }//end try statement
      catch(IOException ex) {
        System.err.println(ex);
      }//end catch block
    }//end constructor
  }//end class Server
```

FindColnagos.java | bikes.txt | tripexpenses.txt | TripExpenses.java | Client.java | *Serve

```
//Client.java
//Networking Notes - Summer 2011
//Client sends a radius to the server and gets back and area for the circle
//MJL 7/26/2011

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Client extends JFrame {
  // Text field for receiving radius
  private JTextField jtf = new JTextField();

  // Text area to display contents
  private JTextArea jta = new JTextArea();

  // IO streams
  private DataOutputStream toServer;
  private DataInputStream fromServer;

  public static void main(String[] args) {
    new Client();
  }//end main

  public Client() {
    // Panel p to hold the label and text field
    JPanel p = new JPanel();
    p.setLayout(new BorderLayout());
    p.add(new JLabel("Enter circle radius"), BorderLayout.WEST);
    p.add(jtf, BorderLayout.CENTER);
    jtf.setHorizontalAlignment(JTextField.RIGHT);
```

Client.java

FindColnagos.java | bikes.txt | tripexpenses.txt | TripExpenses.java | *Client.java | *Server.java

```java
    setLayout(new BorderLayout());
    add(p, BorderLayout.NORTH);
    add(new JScrollPane(jta), BorderLayout.CENTER);
    jtf.addActionListener(new ButtonListener()); // Register listener
    setTitle("Client");
    setSize(300, 150);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setVisible(true); // It is necessary to show the frame here!
    try {
      // Create a socket to connect to the server
      Socket socket = new Socket("localhost", 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());
    }//end try statement
    catch (IOException ex) {
      jta.append(ex.toString() + '\n');
    }//end catch block
  }//end constructor
```

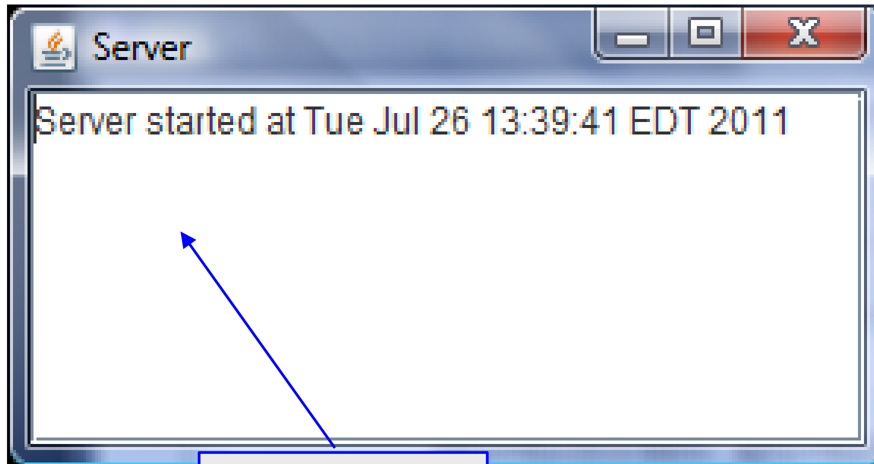
Client.java
(continued)

FindColnagos.java | bikes.txt | tripexpenses.txt | TripExpenses.java | *Client.java | *Serv

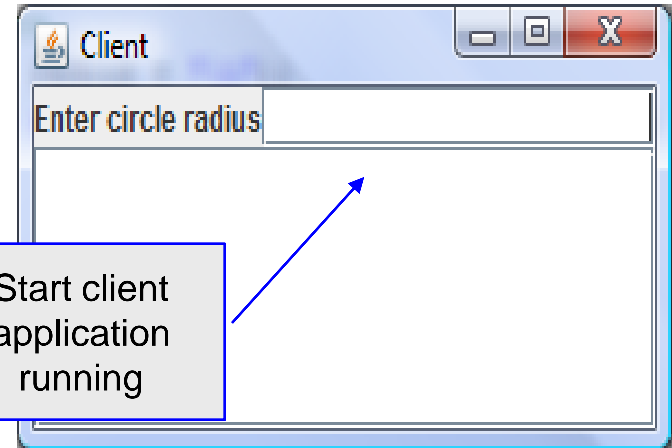
Client.java (continued)

```
  private class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      try {
        // Get the radius from the text field
        double radius = Double.parseDouble(jtf.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
        jta.append("Radius is " + radius + "\n");
        jta.append("Area received from the server is " + area + '\n');
      }//end try statement
      catch (IOException ex) {
        System.err.println(ex);
      }//end catch block
    }//end actionPerformed method
  }//end ButtonListener class
}//end Client class
```

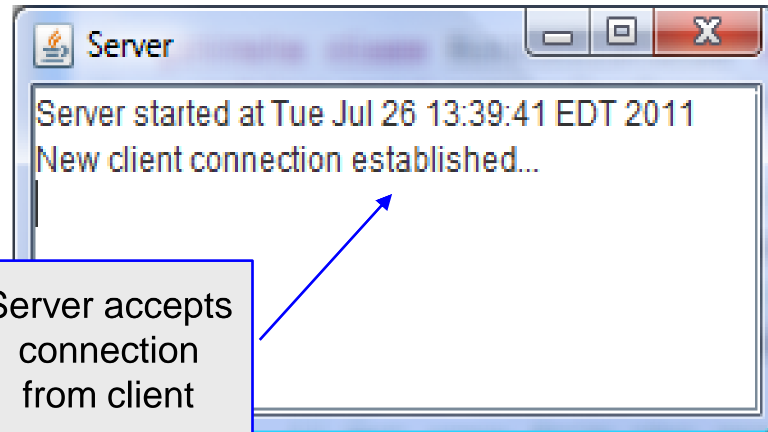
Execution of Client-Server Example



1\. Start server application running

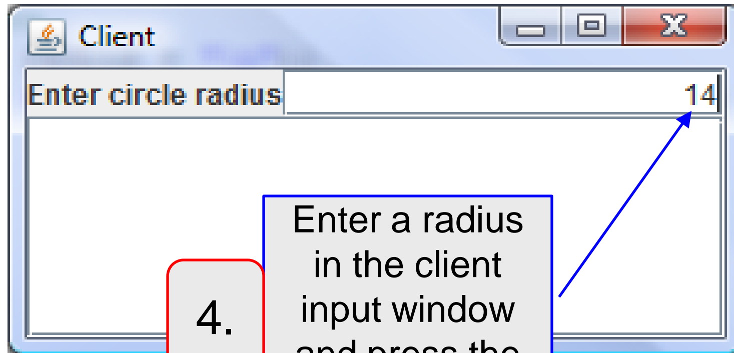


2\. Start client application running



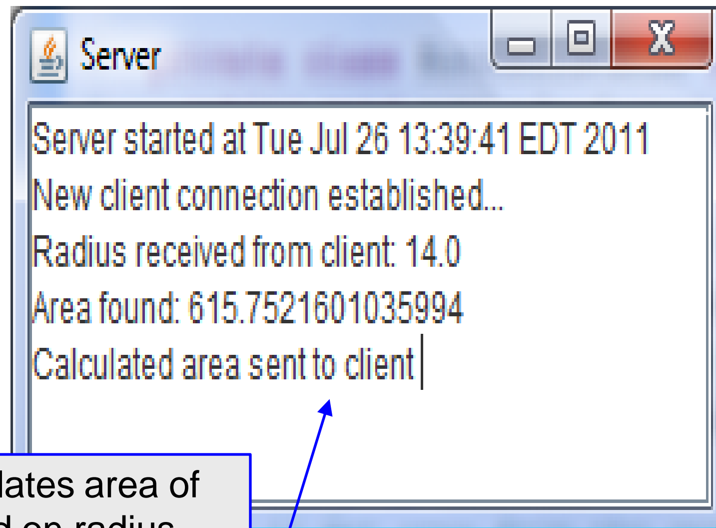
3\. Server accepts connection from client

Execution of Client-Server Example



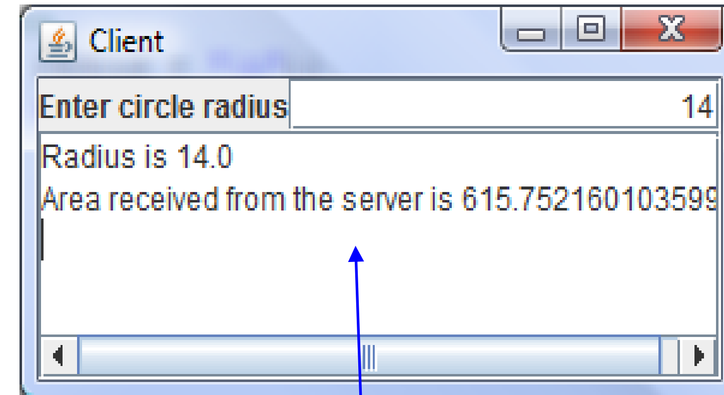
4.

Enter a radius in the client input window and press the enter key



5.

Server calculates area of circle based on radius received from client and returns area to client.



6.

Client receives area calculation from server and displays the result.

A Client-Server Example

- The client-server example is set up so that the server will loop forever, listening for clients attempting to connect. The following page illustrates multiple instances of client connections.
- I slightly modified the `Client.java` file to include a line that will print out the local port number which is automatically assigned by the JVM when the client connects to the server on port 8000. Note that the client needs to know the port number the server is listening on and the server needs to know which client it is talking to.
- The modified code (shown below) is inserted right after the socket object is created in the client application.

```
// Create an input stream to receive data from the server
// When the client connects to the server on port 8000, a socket is created dynamically
// on the client side.  This socket has its own local port number chosen automatically
// by the JVM.  If you want to see what this port number is uncomment the following line.
jta.append("The local port number is: " + socket.getLocalPort() + "\n");
```

Server

```
Server started at Tue Jul 26 13:55:10 EDT 2011
New client connection established...
Radius received from client: 10.0
Area found: 314.1592653589793
Calculated area sent to client

New client connection established...
Radius received from client: 15.0
Area found: 706.8583470577034
Calculated area sent to client

New client connection established...
Radius received from client: 22.0
Area found: 1520.53084433746
Calculated area sent to client
```

Client

```
Enter circle radius                10
The local port number is: 51588
Radius is 10.0
Area received from the server is 314.159265358979
```

Client

```
Enter circle radius                15
The local port number is: 51589
Radius is 15.0
Area received from the server is 706.858347057703
```

Client

```
Enter circle radius                22
The local port number is: 51590
Radius is 22.0
Area received from the server is 1520.53084433746
```

The `InetAddress` Class

- Sometimes, it is desirable for the server to know or identify who is attempting to connect to it. The `InetAddress` class is used to find the client's host name.
- The `InetAddress` class models an IP address.
- The statement below creates an instance of `InetAddress` for the client on a socket:

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

- You can also create an instance of `InetAddress` from a host name or IP address using the static `getbyName` method.
- The example program on the next page identifies the host name and IP address of the arguments you pas it from the command line.

tripexpenses.txt | TripExpenses.java | Client.java | Server.java | IdentifyHostNameIP.j

```java
//IdentifyHostNameIP.java
//Java Networking - Summer 2011
//Uses InetAddress class to return host name and IP address of command line arguments
// MJL 7/26/2011
import java.net.*;

public class IdentifyHostNameIP {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {
      try {
        InetAddress address = InetAddress.getByName(args[i]);
        System.out.println("Host name: " + address.getHostName());
        System.out.println("IP address: " + address.getHostAddress());
      }//end try statement
      catch (UnknownHostException ex) {
        System.err.println("Unknown host or IP address " + args[i]);
      }//end catch block
    }//end for loop
  }//end main method
}//end class
```

Console

<terminated> IdentifyHostNameIP [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jul 26, 2011 2:28:30 PM)

```
Host name: www.amazon.com
IP address: 72.21.211.176
Host name: www.my.ucf.edu
IP address: 10.171.242.31
Host name: www.cyclingnews.com
IP address: 89.167.143.53
```

Serving Multiple Clients Simultaneously

- Multiple clients are quite often connected to a single server at the same time.
- Typically, a server runs constantly on a dedicated server computer, and clients from all over the Internet may want to connect to it.
- This is handled by multithreading the server. A thread is created for each connection.

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

Serving Multiple Clients Simultaneously

- The server socket can have many connections. Each iteration of the while loop shown on the previous page 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 simultaneously.
- The diagram on the next page illustrates this concept.

Serving Multiple Clients Simultaneously

Server

A serve socket on a port

A socket for a client

A socket for a client

Client 1

Client n

• • •

- The example beginning on the next page creates a multithreaded version of the earlier server example where the server returned the area of a circle based on the radius supplied by the client.

bikes.txt | tripexpenses.txt | *Client.java | Server.java | MultiThreadServer.ja

```java
//MultithreadServer.java
//Java Networking notes - COP 3330 Summer 2011
//MJL 7/26/2011

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;

public class MultiThreadServer extends JFrame {
  // Text area for displaying contents
  private JTextArea jta = new JTextArea();

  public static void main(String[] args) {
    new MultiThreadServer();
  }//end main method

  public MultiThreadServer() {
    // Place text area on the frame
    setLayout(new BorderLayout());
    add(new JScrollPane(jta), BorderLayout.CENTER);

    setTitle("MultiThreadServer");
    setSize(500, 300);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true); // It is necessary to show the frame here!
```

MultithreadServer.java

bikes.txt | tripexpenses.txt | *Client.java | Server.java | MultiThreadServer.ja | currentWeather.java | 72

MultithreadServer.java
(continued)

```
try {
  // Create a server socket
  ServerSocket serverSocket = new ServerSocket(8000);
  jta.append("MultiThreadServer started at " + new Date() + '\n');
  // Number a client
  int clientNo = 1;
  while (true) {
    // Listen for a new connection request
    Socket socket = serverSocket.accept();
    // Display the client number
    jta.append("\nStarting thread for client " + clientNo + " at " + new Date() + '\n');
    // Find the client's host name, and IP address
    InetAddress inetAddress = socket.getInetAddress();
    jta.append("Client " + clientNo + "'s host name is " + inetAddress.getHostName() + "\n");
    jta.append("Client " + clientNo + "'s IP Address is "  + inetAddress.getHostAddress() + "\n")
    // Create a new thread for the connection
    HandleAClient task = new HandleAClient(socket);
    // Start the new thread
    new Thread(task).start();
    // Increment clientNo
    clientNo++;
  }//end while loop
}//end try statement
catch(IOException ex) {
  System.err.println(ex);
}//end catch block
}//end constructor
```

bikes.txt | tripexpenses.txt | *Client.java | Server.java | *MultiThreadServer.j | currentWeather.java

```
  // Inner class - 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;
    }//end constructor

    /** Run a thread  - implement the run method*/
    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);
          jta.append("\nRadius received from client: " + radius + '\n');
          jta.append("Area found: " + area + "\n\n");
        }//end while loop
      }//end try statement
      catch(IOException e) {
        System.err.println(e);
      }//end catch block
    }//end run method
  }//end inner class
}//end outer class
```

MultithreadServer.java (continued)

MultiThreadServer

```
MultiThreadServer started at Tue Jul 26 14:35:15 EDT 2011

Starting thread for client 1 at Tue Jul 26 14:35:22 EDT 2011
Client 1's host name is 127.0.0.1
Client 1's IP Address is 127.0.0.1

Starting thread for client 2 at Tue Jul 26 14:35:23 EDT 2011
Client 2's host name is 127.0.0.1
Client 2's IP Address is 127.0.0.1

Starting thread for client 3 at Tue Jul 26 14:35:23 EDT 2011
Client 3's host name is 127.0.0.1
Client 3's IP Address is 127.0.0.1

Starting thread for client 4 at Tue Jul 26 14:35:24 EDT 2011
Client 4's host name is 127.0.0.1
Client 4's IP Address is 127.0.0.1

Radius received from client: 11.0
Area found: 380.132711084365


Radius received from client: 13.0
Area found: 530.929158456675
```

Client

```
Enter circle radius
The local port number is: 51643
```

Client

```
Enter circle radius 11
The local port number is: 51642
Radius is 11.0
Area received from the server is 380.132711084365
```

Client

```
Enter circle radius
The local port number is: 51641
```

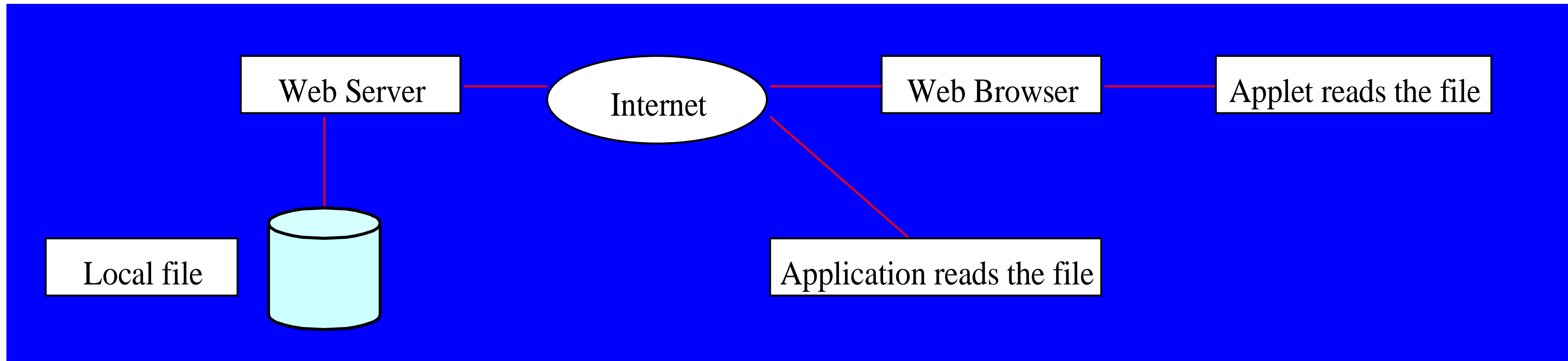
Client

```
Enter circle radius 13
The local port number is: 51640
Radius is 13.0
Area received from the server is 530.929158456675
```

Execution of `MultithreadServer.java`

Retrieving Files From Web Servers

- In the previous examples we developed client-server applications where we created both the server and the client applications.
- Java allows you to develop clients that retrieve files on a remote host through a Web server. In this case, you do not create a custom server. The Web server is used to send the files, as shown in the diagram below.



Client.java | MultiThreadServer.ja | ReadServerFileTest.j | ReadServerFile.java

ReadServerFile.java

```java
// ReadServerFile.java

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;

public class ReadServerFile extends JFrame {
    private JTextField enterField; // JTextField to enter site name
    private JEditorPane contentsArea; // to display Web site

    // set up GUI
    public ReadServerFile()    {
        super( "Simple Web Browser" );
        // create enterField and register its listener
        //enterField = new JTextField( "Enter file URL here" );
        enterField = new JTextField();
        enterField.addActionListener(
            new ActionListener() {
                // get document specified by user
                public void actionPerformed( ActionEvent event ) {
                    getThePage( event.getActionCommand() );
                } // end method actionPerformed
            } // end inner class
```

Client.java | MultiThreadServer.ja | ReadServerFileTest.j | ReadServerFile.java

ReadServerFile.java

```
// ReadServerFile.java

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;

public class ReadServerFile extends JFrame {
    private JTextField enterField; // JTextField to enter site name
    private JEditorPane contentsArea; // to display Web site

    // set up GUI
    public ReadServerFile()    {
        super( "Simple Web Browser" );
        // create enterField and register its listener
        //enterField = new JTextField( "Enter file URL here" );
        enterField = new JTextField();
        enterField.addActionListener(
            new ActionListener() {
                // get document specified by user
                public void actionPerformed( ActionEvent event ) {
                    getThePage( event.getActionCommand() );
                } // end method actionPerformed
            } // end inner class
```

Client.java | MultiThreadServer.ja | ReadServerFileTest.j | ReadServerFile.j

ReadServerFile.java (continued)

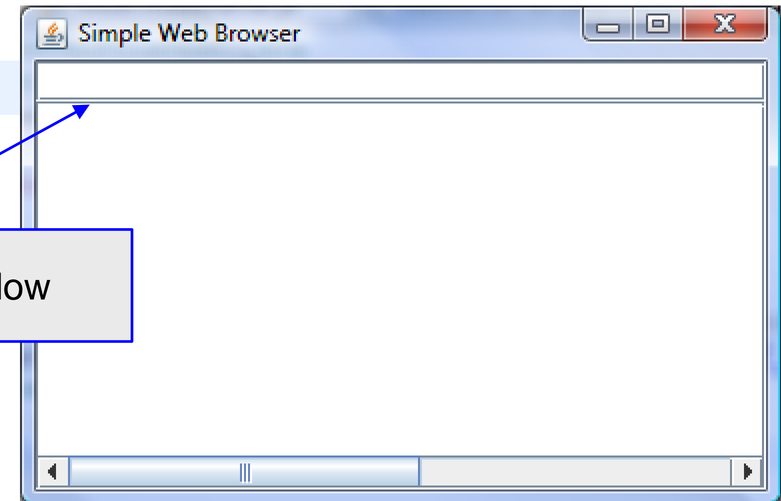
```java
        ); // end call to addActionListener
        add( enterField, BorderLayout.NORTH );
        contentsArea = new JEditorPane(); // create contentsArea
        contentsArea.setEditable( false );
        contentsArea.addHyperlinkListener(new HyperlinkListener() {
              // if user clicked hyperlink, go to specified page
              public void hyperlinkUpdate( HyperlinkEvent event ) {
                 if ( event.getEventType() == HyperlinkEvent.EventType.ACTIVATED )
                    getThePage( event.getURL().toString() );
              } // end method hyperlinkUpdate
           } // end inner class
        ); // end call to addHyperlinkListener
        add( new JScrollPane( contentsArea ), BorderLayout.CENTER );
        setSize( 700, 500 ); // set size of window
        setLocationRelativeTo(null);
        setVisible( true ); // show window
     } // end ReadServerFile constructor

     // load document
     private void getThePage( String location )  {
        try  {// load document and display location
           contentsArea.setPage( location ); // set the page
           enterField.setText( location ); // set the text
        } // end try
        catch ( IOException ioException ) {
           JOptionPane.showMessageDialog( this, "Error retrieving specified URL", "Bad URL",
              JOptionPane.ERROR_MESSAGE );
        } // end catch
     } // end method getThePage
  } // end class ReadServerFile
```

`ReadServerFileTest.java`
(driver class for `ReadServerFile.java`)

```java
// 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
```



Initial window

Simple Web Browser

http://www.cs.ucf.edu/courses/cop3330/sum2011

UCF

COP 3330 - Object-Oriented Programming - Summer 2011

Monday & Wednesday 4:00-5:50 pm HEC 125

Instructor: Dr. Mark Llewellyn
HEC 236
(407) 823-2790
Office Hours: M & W: 3:00-4:00pm, T: 12:00-2:00pm
Email to: markl@cs.ucf.edu

TA Information: (Office Hours held in HEC 308)
Ms. Jennifer Graham (lead TA) T & Th 12:00-2:00pm - Email to: jenng@knights.ucf.edu
Mr. Scott Beck M & W 12:00-1:00pm - Email: please use WebCourses email
Mr. Chintan Shah T 2:00-4:00pm - Email:cshah3.gmail.com

Course Calendar | Syllabus | TA and Supplemental Instruction Information

Lecture Notes | Solutions to the practice problems in the lecture notes

Course Assignments | Helpful Things | Code Page - Sample Code From Lecture Notes