CNT 4714: Enterprise Computing
Summer 2014

Java Networking and the Internet

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cnt4714/sum2014

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

Distributed Applications in the Enterprise

- Distributed applications are one of the latest developments of information technology, which began about 50 years ago, and is still developing at a very fast pace.

- The first electronic computers available in 1940s and 50s were reserved for special applications. Many had military applications such as the encryption and decoding of messages.

- The 1960s witnessed the advent of batch processing, in which several users could pass their tasks to the computer operator (the "server"). Once processed, the results were returned to the "client" by the operator. As there was no interactivity at that time, computers were used for primarily numerical applications that required little user input but required a high computational effort.

Distributed Applications in the Enterprise (cont.)

- With the advent of mainframes, interactive applications came into play. Several users could finally use one computer simultaneously in time-sharing mode. Tasks were no longer completed in sequence as with batch processing, but rather completed in sections.

- The next trend, beginning with the introduction of the PC in 1980, was the shift in computing power from the central mainframe to the desktop. Computer performance at levels which were undreamt of previously, was now available to employees directly at their desk. Each user could install their own applications to create an optimally configured work environment. This began the age of the standardized office packages, which enabled office automation to be driven forward considerably.

Distributed Applications in the Enterprise (cont.)

- Since the 1990s, the trend has shifted increasingly from distributed information processing to enterprise computing.
- Previously autonomously operating workstations were integrated together with central file, database, and application servers, resulting in huge decentralized clusters, which were used to handle tasks of a more complicated nature.
- The defining sentence which characterized this phase coined by Sun Microsystems reads – "The network is the computer."
- What was it that led to this ever increasing greater importance of distributed applications?
- There are several reasons commonly cited:

Distributed Applications in the Enterprise (cont.)

1. The cost of chip manufacturing dropped sharply, enabling cheap mass production of computers.

2. Simultaneously, network technologies were developed with higher bandwidths – a necessity for the quick transfer of large amounts of data between several computers.

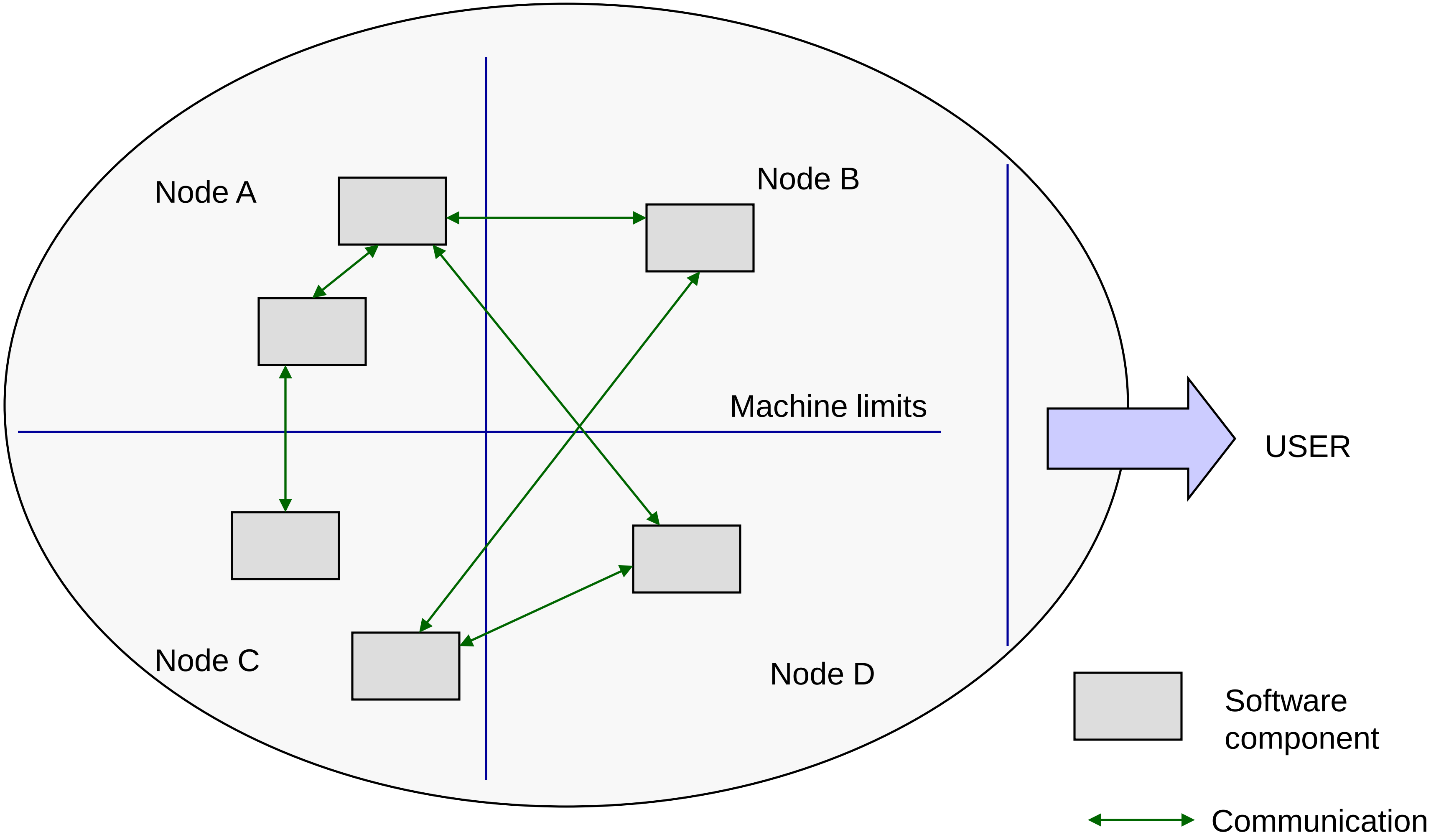
3. Response times became increasingly longer due to the heavy use of large mainframes, resulting in excessive waiting times.

4. The availability of a comparable distributed work environment gave rise to the desire for new applications that were not possible in a centralized environment. This development led from the first e-mail applications via the WWW to common use of information by people in completely different places.

What is a Distributed System?

- While many different definitions of what constitutes a distributed system have been put forth, there is general consensus that there are several central components that a distributed system must contain:
  - A set of autonomous computers.
  - A communication network, connecting those computers.
  - Software which integrates these components with a communication system.

What is a Distributed System?



What is a Distributed Application?

- A distributed application is an application A, the functionality of which is subdivided into a set of cooperating subcomponents A_1, A_2, …, A_n with $n > 1$. The subcomponents A_i are autonomous processing units which can run on different computers and exchange information over the network controlled by coordination software.

- There are typically three levels defined for a distributed system:

Level 3	Distributed Applications
Level 2	Coordination Software
Level 1	Distributed Computer System

What is a Distributed Application? (cont.)

- The application on level 3 will ideally "know" nothing of the distribution of the system, as it uses the services of level 2, the administration software that takes over the coordination of all the components and hides the complexity from the application.
- In turn, level 2 itself uses the available distributed computing environment.

As an aside, a more humorous definition of a distributed system was given by Leslie Lamport (the guy who developed LaTex), who defined a distributed system as a system "in which my work is affected by the failure of components, of which I knew nothing previously."

Important Characteristics of Distributed Systems

- Based on our simple definition, there are several important characteristics of distributed systems that need a closer look.

- All of these characteristics are based on the concept of transparency.

- In the context of information technology, the concept of transparency literally means that certain things should be invisible to the user. The manner in which the problem is solved is largely irrelevant to the user.

- The following transparency properties play a large role in achieving this result for the user:

Transparency Properties of Distributed Systems

Location Transparency – users do not necessarily need to know where exactly within the system a resource is located which they wish to utilize. Resources are typically identified by name, which has no bearing on their location.

Access Transparency – the way in which a resource is access is uniform for all resources. For example, in a distributed database system consisting of several databases of different technologies, there should also be a common user interface (such as SQL).

Replication Transparency – the face that there may be several copies of a resource is not disclosed to the user. The user has no need to know whether they are accessing the original or the copy. The altering of the resource also must occur transparently.

Transparency Properties of Distributed Systems (cont.)

Error Transparency – users will not necessarily be informed of all errors occurring in the system. Some errors may be irrelevant, and others may well be masked, as in the case of replication.

Concurrency Transparency – distributed systems are usually used by several users simultaneously. It often happens that two or more users access the same resource at the same time, such as a database table, printer, or file. Concurrency transparency ensures that simultaneous access is feasible without mutual interference or incorrect results.

Migration Transparency – using this form of transparency, resources can be moved over the network without the user noticing. A typical example is today’s mobile telephone network in which the device can be moved around freely, without any loss of communication when leaving the vicinity of a sender station.

Transparency Properties of Distributed Systems (cont.)

Process Transparency – It is irrelevant on which computer a certain task (process) is executed, provided it is guaranteed that the results are the same. This form of transparency is an important prerequisite for the successful implementation of a balanced workload between computers.

Performance Transparency – when increasing the system load, a dynamic reconfiguration may well be required. This measure for performance optimization should be unnoticed by other users.

Scaling Transparency – if a system is to be expanded so as to incorporate more computers or applications, this should be feasible without modifying the system structure or application algorithms.

Language Transparency – the programming language in which the individual subcomponents of the distributed system or application were created must not play any role in the ensemble. This is a fairly new requirement of distributed systems and is only supported by more recently developed systems.

Basic Communication Models

- Communication between the individual components of a distributed system can occur in two basic ways: using either shared memory or message passing.

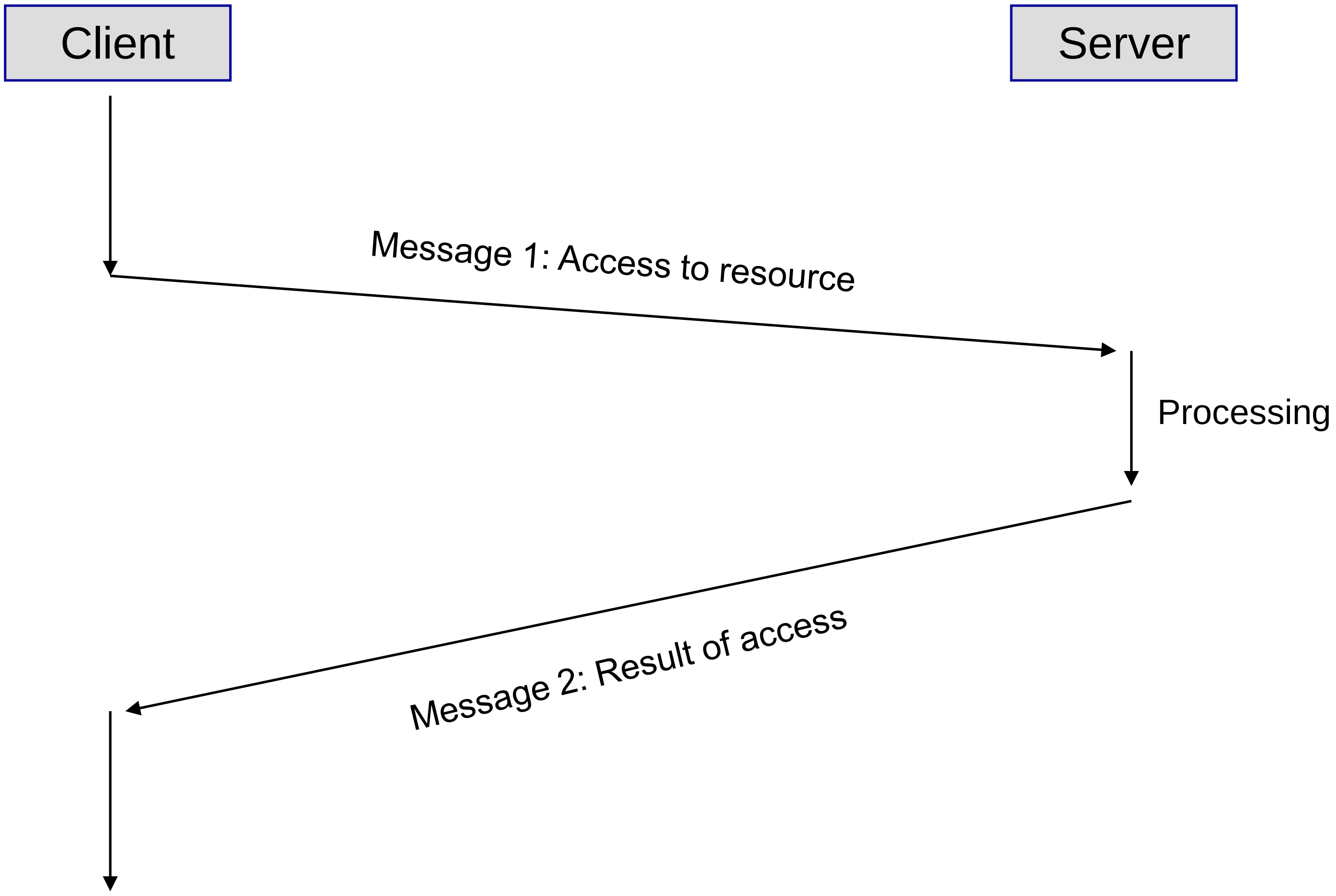
- Shared memory is an indirect form of communication, as both partners exchanging information do not communicate directly with each other, but via a third component: the shared memory.

- Message passing is a direct form of communication between the sender and receiver by means of a communication medium. Two functions are generally available for the execution of message exchange, usually called send and receive.

Basic Communication Models (cont.)

- Send is defined as: `send(r: recevier, m: message)`
  - This function sends the message *m* to the receiver *r*.
- Receive is defined as: `receive(s: sender, b: buffer)`
  - This function waits for a message from sender *s* and writes it in buffer *b* (part of the memory made available for the application process).
- The basic form of exchange of a single message can be combined with more complex models. One of the most important of these models is the client-server model.
  - In this model, the communication partners adopt the role of either a client or a server. A server is assigned to administer access to a certain resource, while a client wishes to use the resource.

Message Exchange in the Client-Server Model



Advantages and Disadvantages of Distributed Systems

- When compared to the mainframe approach, distributed systems offer the following advantages:
  - More economical – greater computing power is available at a lower cost.
  - Response times are much shorter.
  - Provide a better model of reality than a centralized computer (consider the information infrastructure of a multinational corp.).
  - Distributed systems can be made more reliable than a central system. Availability of individual components can be enhanced through replication. Also, the failure of a non-replicated component does not typically lead to total system failure as with a mainframe.
  - Distributed systems can be extended and adapted to increasing requirements far easier than can a mainframe.

Advantages and Disadvantages of Distributed Systems (cont.)

- When compared to the conventional PC approach, distributed systems offer the following advantages:
  - Generally speaking, communication between computers can only occur using a connection. Applications such as e-mail are only possible using this approach.
  - Networking PCs allows common use of both resources and data, especially hardware resources such as printers and hard drives.
  - Unless the PCs are networked, load sharing is not possible, so one user running two computationally-intensive applications will suffer even if the adjacent workstation is unused.

Advantages and Disadvantages of Distributed Systems (cont.)

- There are however, a few problems that arise with distributed systems:
  - The entire system is extremely dependent on transmission performance and the reliability of the underlying communication network. If the network is constantly overloaded, then the advantages of a distributed system are very quickly cancelled out, particularly with respect to response times.
  - Distribution and communication are always an increased security risk in many ways. Communications can be "snooped". Physical security of the system components becomes more difficult. Software issues concerning modification and piracy become more prevalent.
  - Software for both applications and the coordination of application components becomes more complex leading to greater chance for errors and higher development costs.

Technical Principles of the Internet

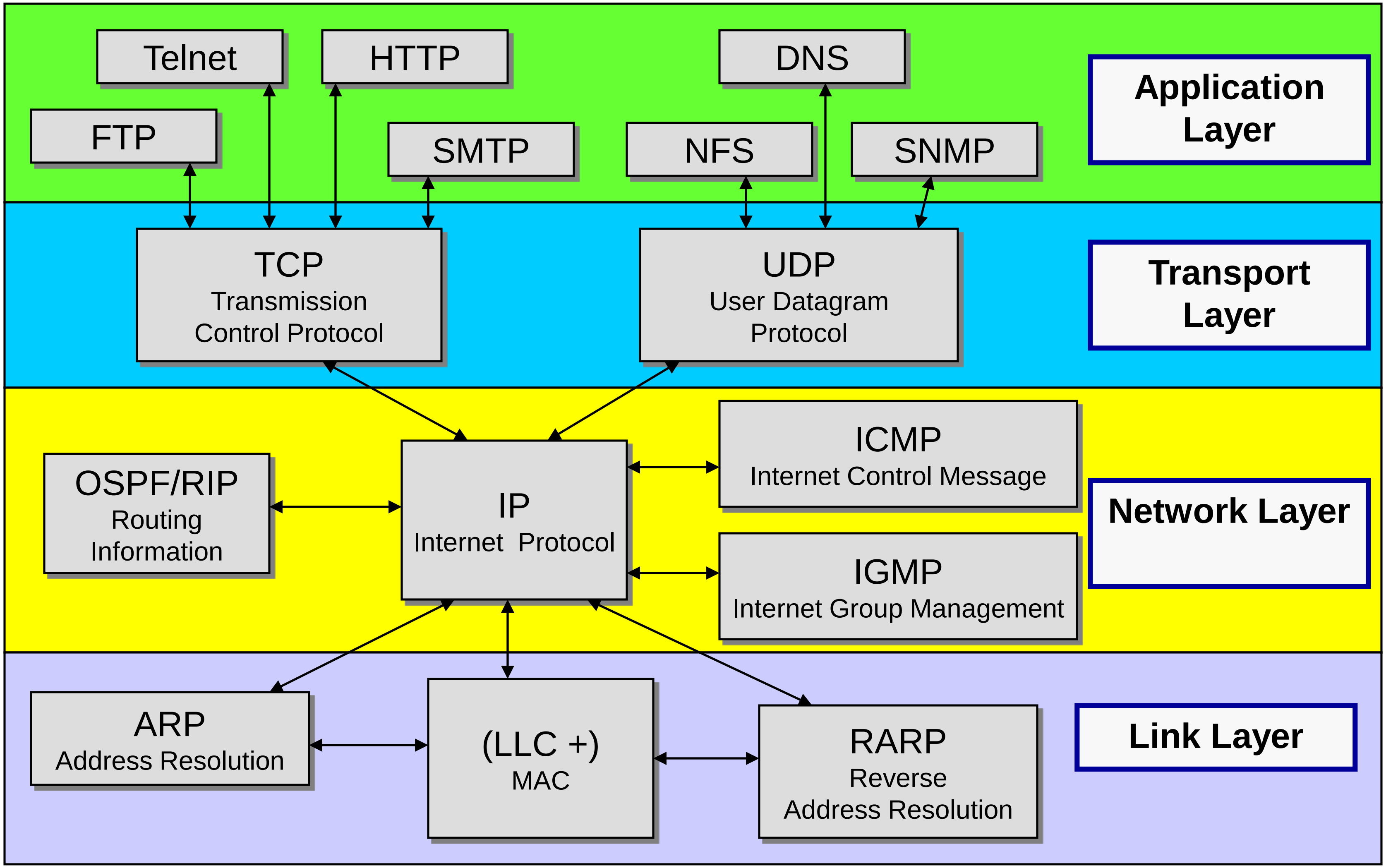
- Communications systems such as the Internet are best described using layered models because of their complexity.

- Every layer within the model has a certain task, and all layers together produce a particular communication service for the user.

- The layers are arranged in hierarchical form. Layers lower in the hierarchy produce a service used by the higher layers. The uppermost layer finally combines all lower layer services and constitutes the interface for applications.

- For the Internet, the so-called Internet reference model is used and is shown on the next slide.

Internet Reference Model



Internet Reference Model (cont.)

- The Link Layer describes the possible sub-networks of the Internet and their medium access protocols. These are, for example, Ethernets, token rings, FDDI, or ISDN networks. To its upper layer, the link layer offers communication between two computers in the same sub-network as a service.

- The Network Layer unites all the sub-networks to become the Internet. The service offered involves making communication possible between any two computers on the Internet. The network layer accesses the services of the link layer, in that a connection between two computers in different networks is put together for many small connections in the same network.

Internet Reference Model (cont.)

- The Transport Layer oversees the connection of two (or more) processes between computers communicating with each other via the network layer.
- The Application Layer makes application-specific services available for inter-process communication. These standardized services include e-mail, file transfer and the World Wide Web.
- Within the layers, protocols are used for the production of a service. Protocols are instances which can be implements either in hardware or software, and communicate with their partner instances in the same levels, but on other computers. It is only this cooperation that enables the service to be produced for the next level up.

Internet Reference Model (cont.)

- The TCP/IP Protocol constitutes the core of Internet communication technology in the transport and network layers.
- Every computer on the Internet always has an implementation of both protocols, TCP (Transmission Control Protocol) and IP (Internet Protocol).
- The task of IP is to transfer data from one Internet computer (the sender) to another (the receiver). On this basis, TCP then organizes the communication between the two processes on these two computers.

Some Important Application Layer Internet Protocols

- Telnet – makes a terminal emulation available on the remote computer. The protocol enables logins to other computers using the network.
- HTTP – (Hypertext Transport Protocol) is the underlying protocol of the World Wide Web. It is responsible for the transfer of hypertext documents.
- SMTP – (Simple Mail Transfer Protocol) is the protocol used for the transfer of e-mail messages.
- FTP – (File Transfer Protocol) is able to manage filestores on a server and enables clients to access files.
- SNMP – (Simple Network Management Protocol) is used for network management on the Internet.
- DNS – (Domain Name Service) is responsible for the mapping of symbolic names to IP addresses.
- NFS – (Network File System) makes the basic functionality for a distributed file system available.

Basic Constituents of Web Applications

- In order to access the Web, first a web server is required. The server administers the entire data material intended for publication on the Web.

- The web server is also responsible for replying to client requests, by delivering the desired documents according to the entitlement of the client.

- Web servers usually record all Web files access, so different analyses can be made using the log files created, from the simplest of tasks such as how many hits have been made in a certain time period, or an analysis of the geographical distribution of users, to more sophisticated tasks such as monitoring attempts at unauthorized access.

- Web servers might also start other programs executing, with which additional information can be obtained or generated. It is this capability that forms the basis of all distributed applications on the WWW.

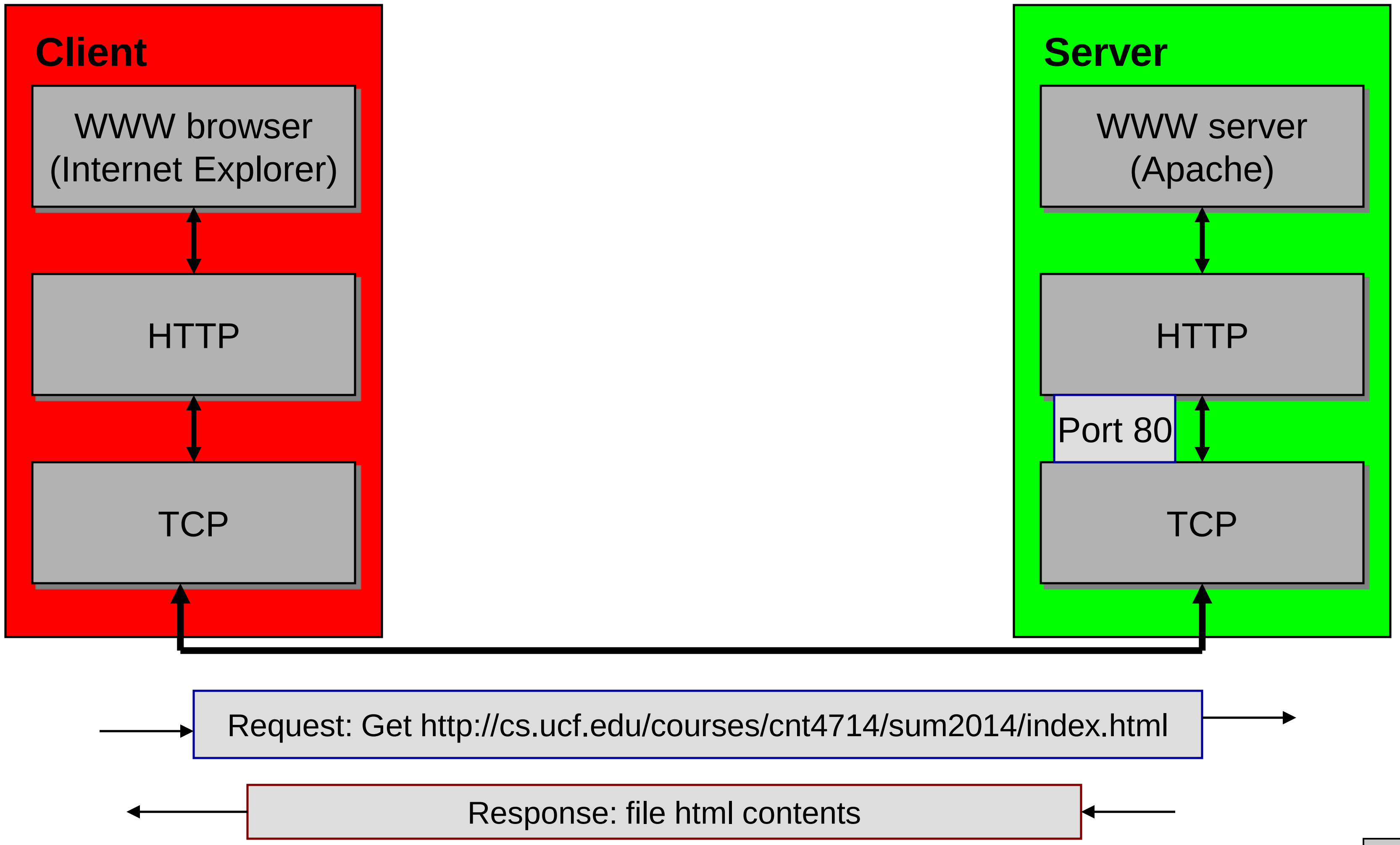
Basic Constituents of Web Applications (cont.)

- The communication between client and server on the WWW takes place using the HTTP protocol.

- HTTP is a purely text-based protocol. This means that the requests for a document are transferred by the client to the server using a “readable” command such as “get”. The server responds to the request by making the requested document available to the client, together with a header giving further information. The server may sometimes respond with an error message, such as if the requested document is not available or the user does not have the proper permission to view the document.

- This protocol is illustrated on the next page. HTTP uses the TCP service for the actual transfer of data between client and server. For every transfer of a Web document, a TCP connection is first established, via which HTTP protocol messages are transferred. (Actually, the TCP connection persists over several HTTP requests.)

The Architecture of a Web Service


Client
WWW browser (Internet Explorer)
HTTP
TCP
Server
WWW server (Apache)
HTTP
Port 80
TCP
Request: Get http://cs.ucf.edu/courses/cnt4714/sum2014/index.html
Response: file html contents

Construction of Web Applications

- The simplest form of an application on the WWW is that in which the provider places a number of static documents on a server.

- A static document is a document which can only be changed from outside, by human intervention.

- Clearly, this prototype is not flexible. As soon as information has to be modified on the server, a slow and cost-intensive process is required. For many applications, this process is simply not an option. Consider, for example, a provider that publishes current weather information. Since this information is constantly changing, an employee would need to be constantly updating the web pages.

- There are a number of approaches today which allow for the dynamic creation of web pages, some of which are already fairly old and others are relatively new. Among the newer of these are Java Servlets and Java Server Pages that we will see later in the semester.

The Architecture of Distributed Web Applications

- There are basically four major components which can or could constitute a distributed application on the Internet. These are:

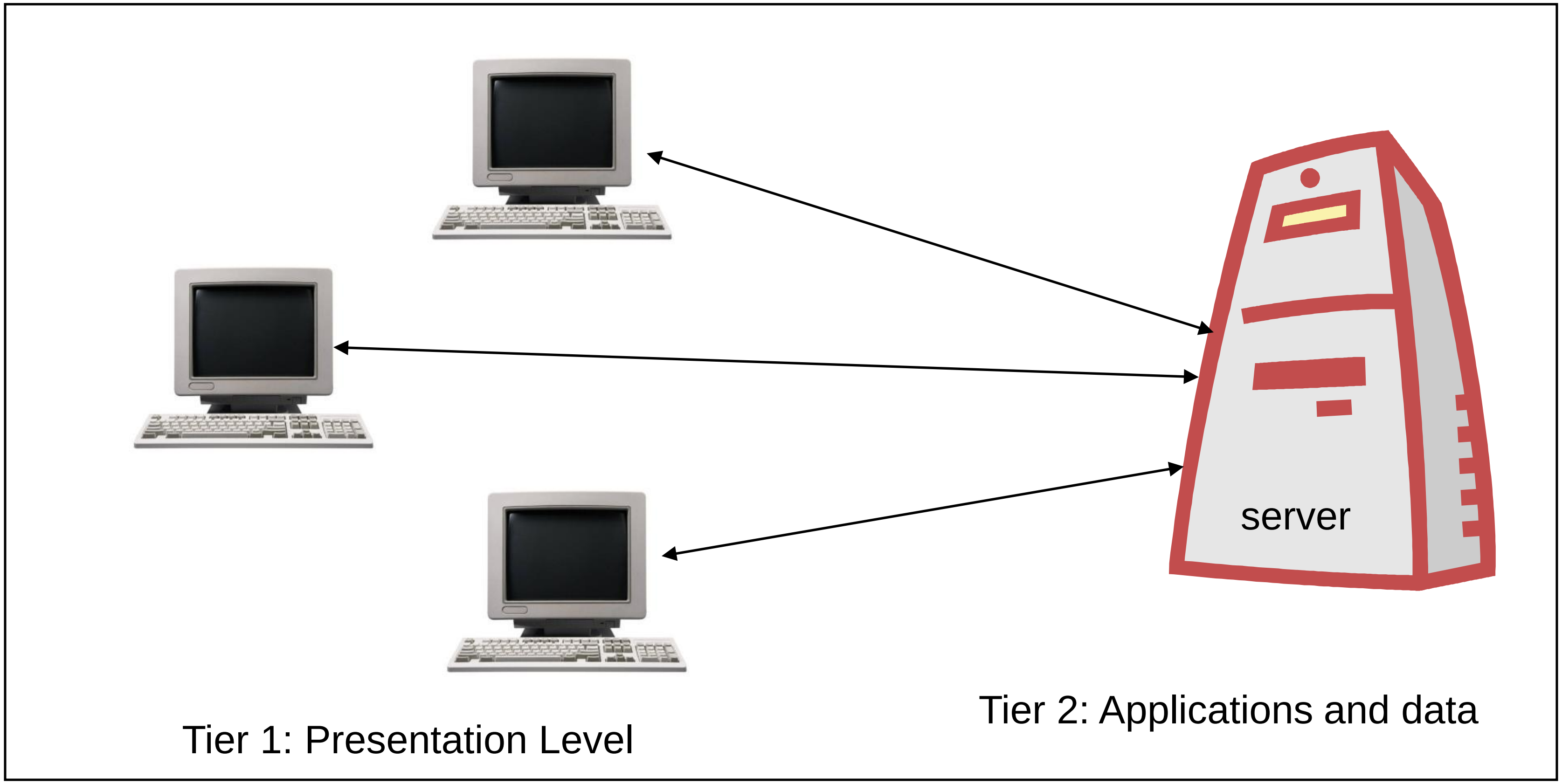
1. The presentation interface to the user, as well as access programs to server components.
2. An access interface to server components.
3. The server application logic.
4. File storage, databases, etc.

- In distributed applications, these four generic components can be distributed on the physical nodes of the system in different configurations.

- The term n-tier architecture was coined for the different variants that can be produced. The term indicates the number of levels on which the components are distributed. In practice, 2-, 3-, and 4-tier architectures are used.

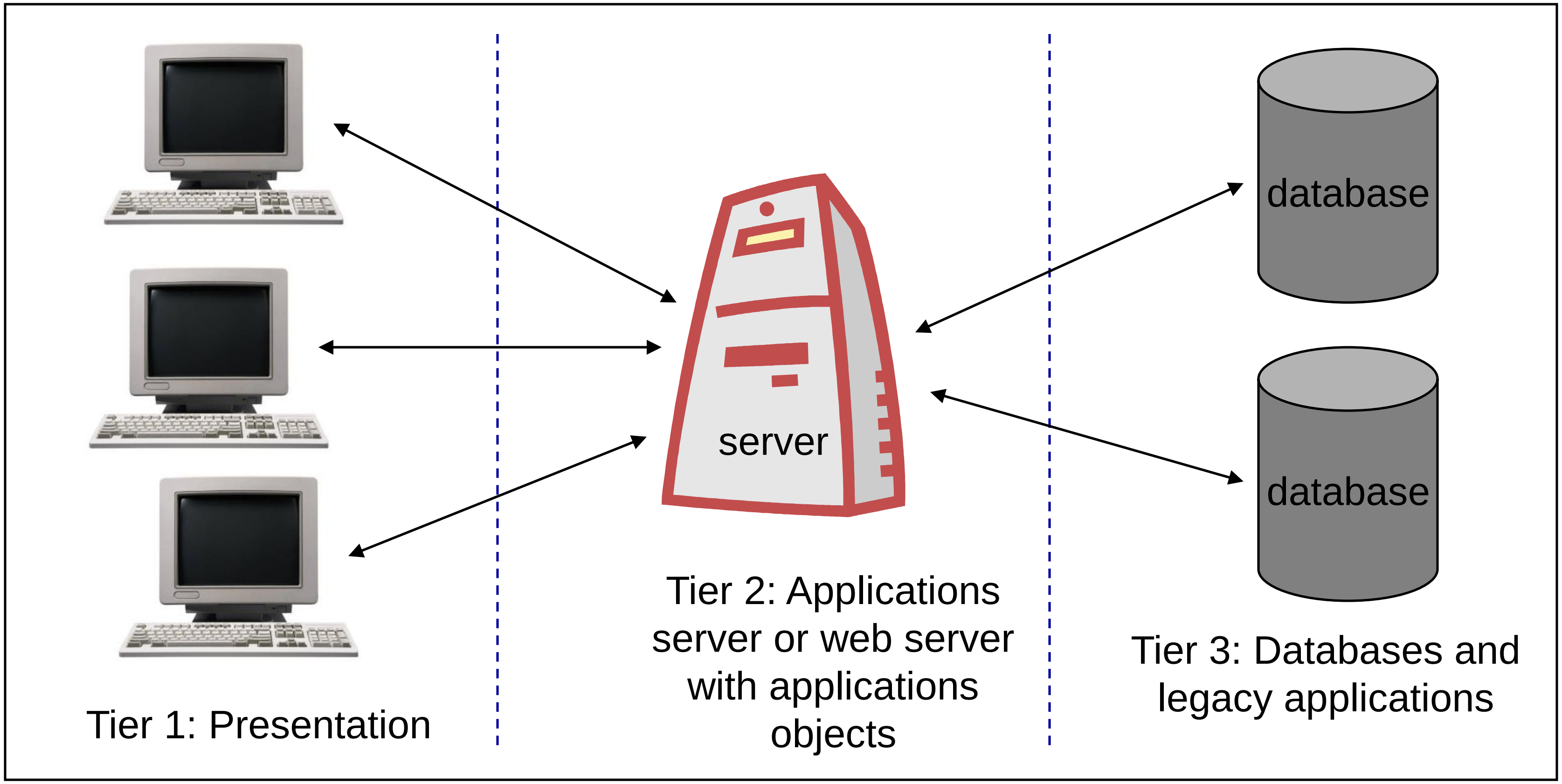
A Two-Tier Architecture

- The simplest version is the 2-tier architecture in which the presentation components are placed on the client computers, and all other components reside on one server computer. The most common example of this is TCP based client-server applications in which databases are accessed directly from the server process.



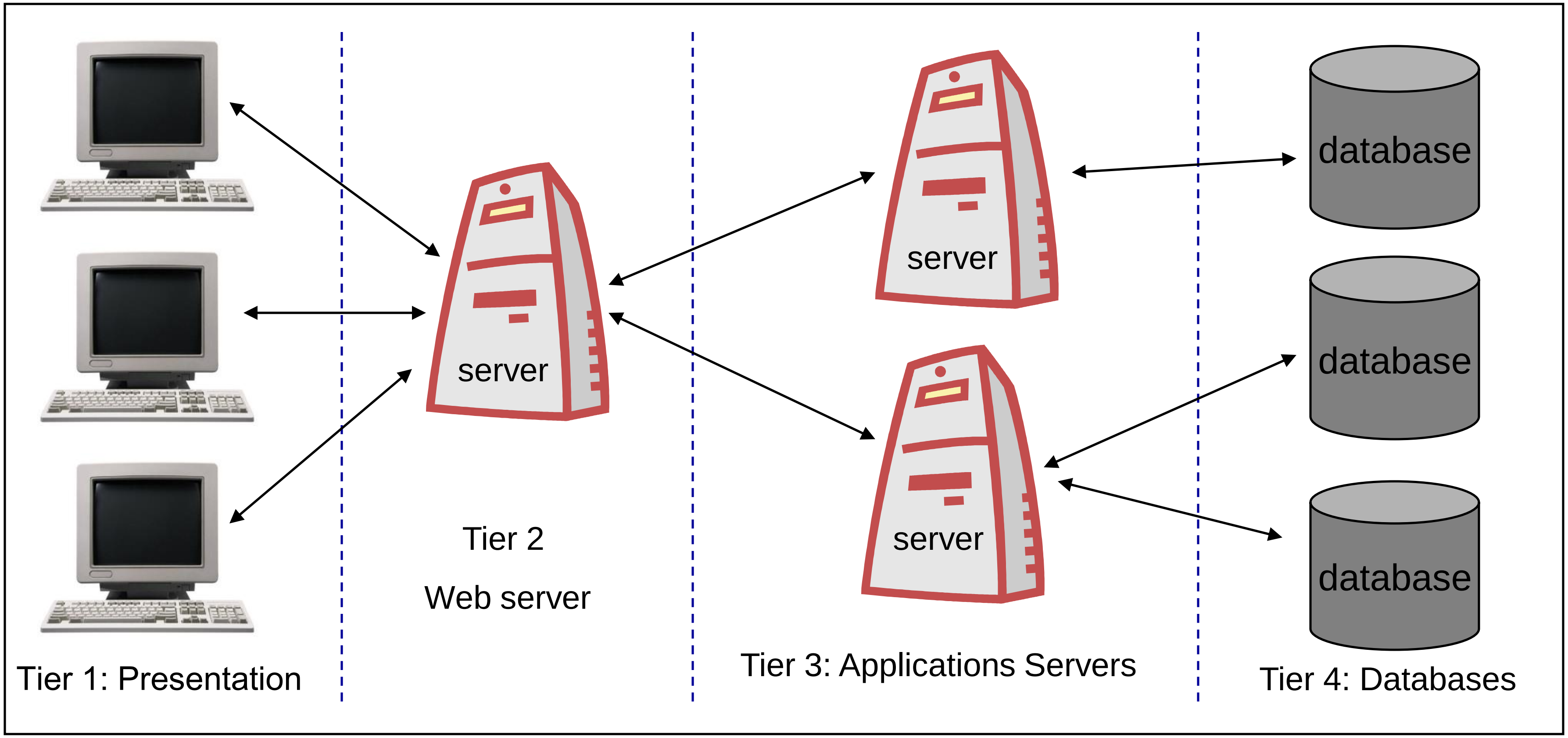
A Three-Tier Architecture

- The 3-tier architecture goes one step further, so that the actual applications are separated from data stocks. This is the common configuration for most servlet applications.


server
database
database
Tier 1: Presentation
Tier 2: Applications server or web server with applications objects
Tier 3: Databases and legacy applications

A Four-Tier Architecture

- The 4-tier architecture refines the 3-tier version by partitioning the server interface from the applications. Although not as common as the 3-tier version, this is the common configuration for many CORBA applications.



Thin Clients

- The extensive partitioning of the architecture of a distributed system into different components with respectively different areas of responsibility, basically allows for the creation of simpler and therefore more controllable individual components.

- The result of this on the client side is the development of thin clients. A thin client is a client program which contains almost no application logic, but offers only the presentation interface to the actual application program, which may run in a distributed fashion on several servers.

- While the application is executed and the graphical interface is in use, application logic is partly loaded on the client computer and executed there locally. However, it is not loaded from the local hard drive, but always by a server via the network.

- The most common version of a thin client today is a web browser. A web browser has no information whatsoever on specific applications.

Thin Clients (cont.)

- A web browser is only able to represent web pages, and possibly execute applets.

- If a certain application is to be used, then the corresponding web pages must be loaded by a web server.

- The use of thin clients has several advantages (as opposed to heavy clients):

  - The installation of program components on the client computer is unnecessary. Neither a reconfiguration of the computer nor regular updates of client software are required.

  - Users do not have to adjust to a new user interface for every distributed application. Access is always made using a well-known web browser interface. This renders a potentially large amount of training unnecessary.

  - Client computers can, on the whole, be equipped more inexpensively, as large hard drives for storing application programs are not needed.

Networking

- Java's fundamental networking capabilities are declared by classes and interfaces of the `java.net` package, through which Java offers *stream-based communications*.
- The classes and interfaces of `java.net` also offer *packet-based communications* for transmitting individual packets of information. This is most commonly used to transmit audio and video over the Internet.
- We will focus on both sides of the client-server relationship.
- The client requests that some action be performed, and the server performs the action and responds to the client.

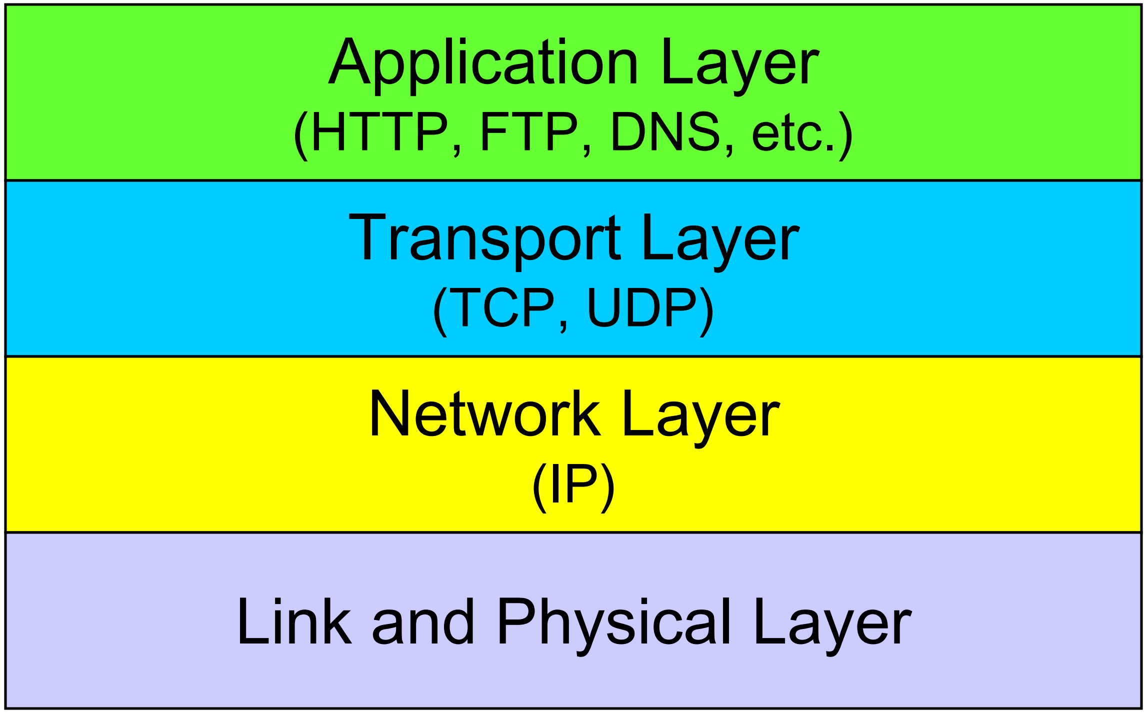
Networking (cont.)

- A common implementation of the request-response model is between Web browsers and Web servers.
  - When a user selects a Web site to browse through a browser (a client application), a request is sent to the appropriate Web server (the server application). The server normally responds to the client by sending the appropriate HTML Web page.

java.net

- "High-level" APIs
  - Implement commonly used protocols such as HTML, FTP, etc.
- "Low-level" APIs
  - Socket-based communications
    - Applications view networking as streams of data
    - Connection-based protocol
    - Uses TCP (Transmission Control Protocol)
  - Packet-based communications
    - Individual packets transmitted
    - Connectionless service
    - Uses UDP (User Datagram Protocol)

Internet Reference Model



Application Layer
(HTTP, FTP, DNS, etc.)

Transport Layer
(TCP, UDP)

Network Layer
(IP)

Link and Physical Layer

See page 21 for a more detailed version of this diagram.

Sockets

- Java's socket-based communications enable applications to view networking as if it were file I/O. In other words, a program can read from a socket or write to a socket as simply as reading from a file or writing to a file.
- A socket is simply a software construct that represents one endpoint of a connection.
- Stream sockets enable a process to establish a connection with another process. While the connection is in place, data flows between the processes in continuous streams.
- Stream sockets provide a connection-oriented service. The protocol used for transmission is the popular TCP (Transmission Control Protocol). Provides reliable , in-order byte-stream service

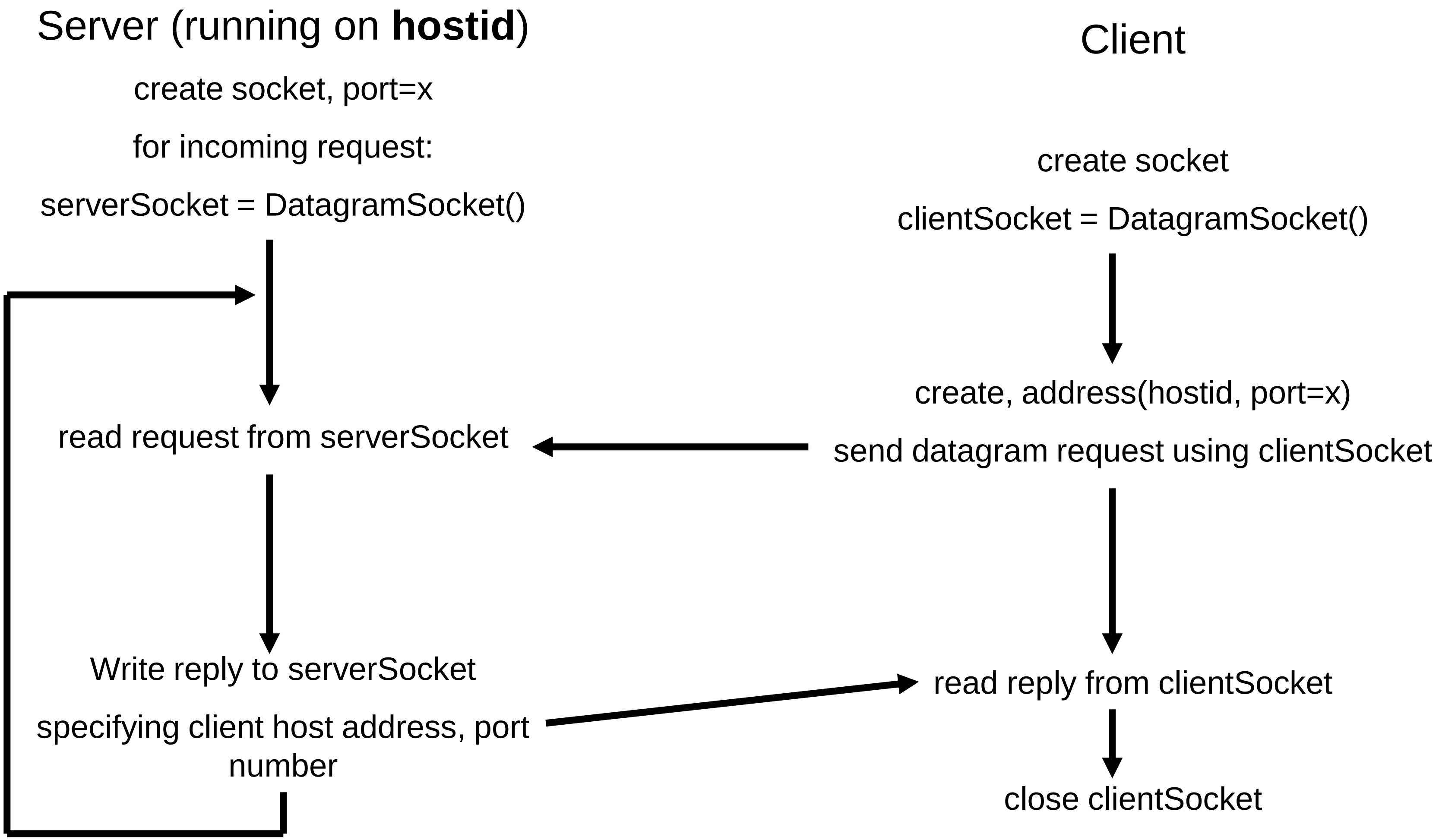
Sockets (cont.)

- Datagram sockets transmit individual packets of information. This is typically not appropriate for use by everyday programmers because the transmission protocol is UDP (User Datagram Protocol).
- UDP provides a connectionless service. A connectionless service does not guarantee that packets arrive at the destination in any particular order.
- With UDP, packets can be lost or duplicated. Significant extra programming is required on the programmer’s part to deal with these problems.
- UDP is most appropriate for network applications that do not require the error checking and reliability of TCP.

Sockets (cont.)

- Under UDP there is no “connection” between the server and the client. There is no “handshaking”.
- The sender explicitly attaches the IP address and port of the destination to each packet.
- The server must extract the IP address and port of the sender from the received packet.
- From an application viewpoint, UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server.

Example: client/server socket interaction via UDP



Server (running on **hostid**)

create socket, port=x

for incoming request:

serverSocket = DatagramSocket()

read request from serverSocket

Write reply to serverSocket

specifying client host address, port number

Client

create socket

clientSocket = DatagramSocket()

create, address(hostid, port=x)

send datagram request using clientSocket

read reply from clientSocket

close clientSocket

Example: Java server using UDP

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
            {
                        //Create datagram socket on port 9876
                        DatagramSocket serverSocket = new DatagramSocket(9876);

                        byte[] sendData = new byte[1024];
                        byte[] receiveData = new byte[1024];

                        while (true)
                        {
                                    //create space for the received datagram
                                    DatagramPacket receivePacket = new
                                                DatagramPacket(receiveData,
                                                                        receiveData.length);
                                    //receive the datagram
                                    serverSocket.receive(receivePacket);

                                    String sentence = new String(receivePacket.getData());
```

Example: Java server using UDP (cont.)

```
        //get IP address and port number of sender
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
                String capitalizedSentence =
                                        sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            //create datagram to send to client
            DatagramPacket sendPacket = new
        DatagramPacket(sendData, sendData.length, IPAddress, port);
            //write out the datagram to the socket
            serverSocket.send(sendPacket);
        } //end while loop
    }
}
```

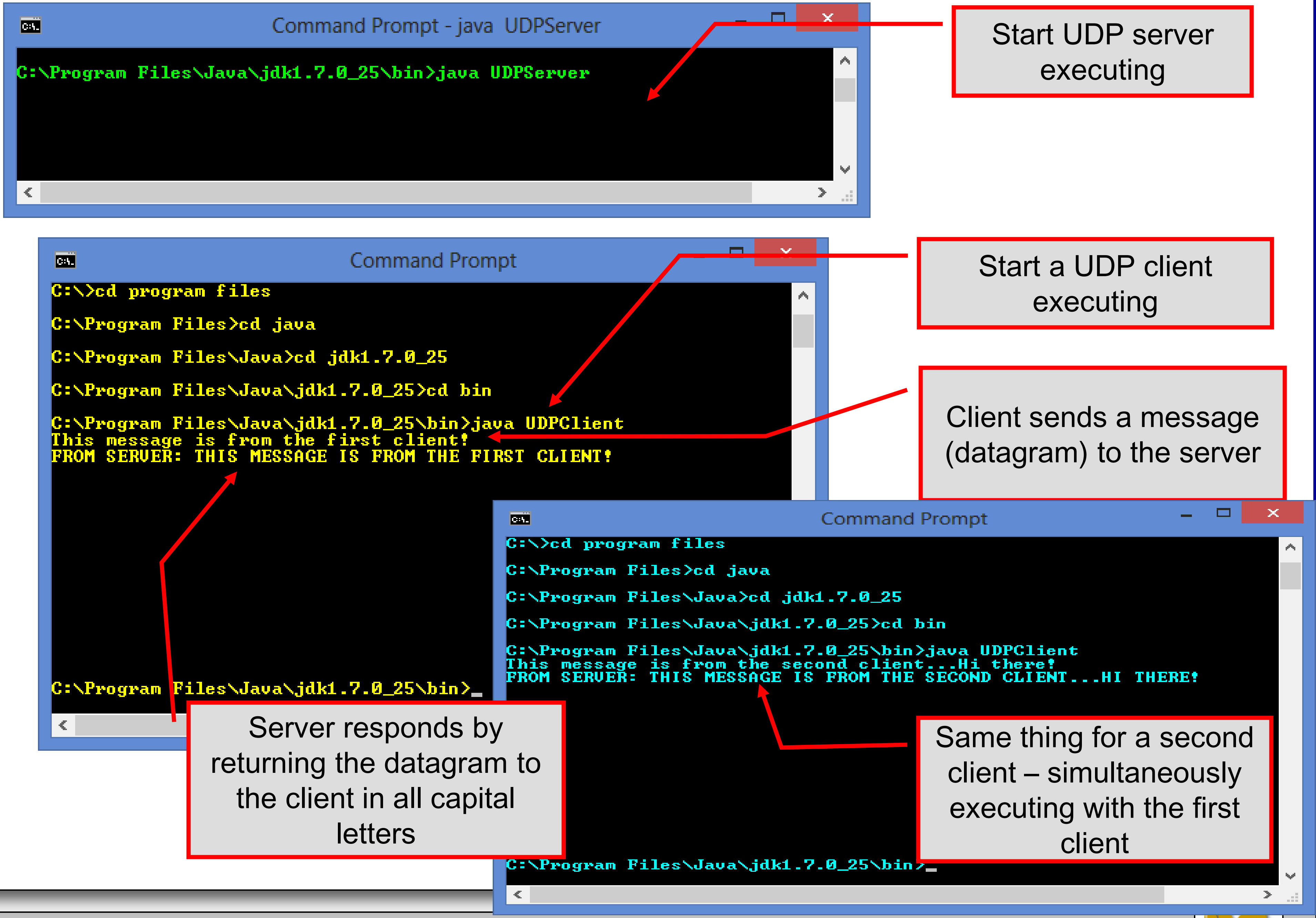
Example: Java client using UDP

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
                 {
                              //Create input stream
                              BufferedReader inFromUser = new BufferedReader(new
                                                      InputStreamReader(System.in));
                              //Create client socket
                              DatagramSocket clientSocket = new DatagramSocket();
                              //Translate hostname to IP address using DNS
                              InetAddress IPAddress = InetAddress.getByName("localhost");

                              byte[] sendData = new byte[1024];
                              byte[] receiveData = new byte[1024];

                              String sentence = inFromUser.readLine();
                              sendData = sentence.getBytes();
```

Example: Java client using UDP (cont.)

```
                DatagramPacket sendPacket = new DatagramPacket(sendData,
                        sendData.length, IPAddress, 9876);
                clientSocket.send(sendPacket);

                DatagramPacket receivePacket = new DatagramPacket(receiveData,
                        receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence = new String(receivePacket.getData());

                System.out.println("FROM SERVER: " + modifiedSentence);
                clientSocket.close();
        }
}
```

Try executing these two applications on your machine and see how it works. The code for both the server and the client are on the code page.



Socket Programming with TCP

- Server process must first be running (must have created a socket). Recall that TCP is not connectionless.
- Client contacts the server by creating client-local socket specifying IP address and port number of server process. Client TCP establishes connection to server TCP.
- When contacted by client, server TCP creates a new socket for server process to communicate with client.
  - Allows server to talk with multiple clients
  - Source port numbers used to distinguish clients
- From application viewpoint: TCP provides reliable, in-order transfer of bytes ("pipe") between client and server.

Establishing a Simple Server Using Stream Sockets

Five steps to create a simple stream server in Java:

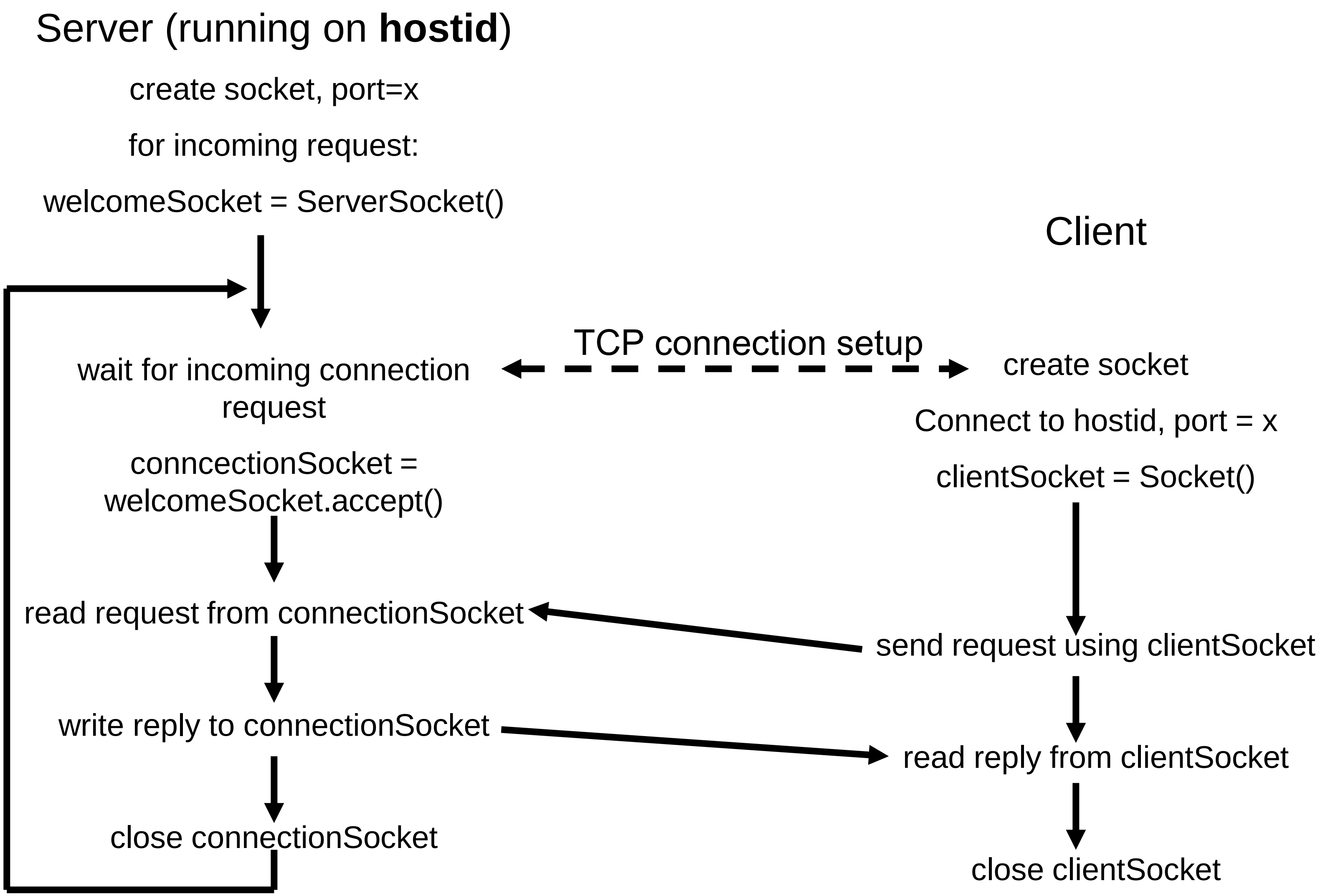
1. `ServerSocket` object. Registers an available port and a maximum number of clients.
2. Each client connection handled with a `Socket` object. Server blocks until client connects.
3. Sending and receiving data
   - `OutputStream` to send and `InputStream` to receive data.
   - Methods `getInputStream` and `getOutputStream` on `Socket` object.
4. Process phase. Server and client communicate via streams.
5. Close streams and connections.

Establishing a Simple Client Using Stream Sockets

Four steps to create a simple stream client in Java:

1. Create a `Socket` object for the client.
2. Obtains `Socket’s InputStream` and `OutputStream`.
3. Process information communicated.
4. Close streams and `Socket`.

Example: client/server socket interaction via TCP



Server (running on **hostid**)

create socket, port=x

for incoming request:

welcomeSocket = ServerSocket()

wait for incoming connection request

conncectionSocket = welcomeSocket.accept()

read request from connectionSocket

write reply to connectionSocket

close connectionSocket

Client

TCP connection setup

create socket

Connect to hostid, port = x

clientSocket = Socket()

send request using clientSocket

read reply from clientSocket

close clientSocket

Example: Java server using TCP

```
//simple server application using TCP

import java.io.*;
import java.net.*;

class TCPServer {
        public static void main (String args[]) throws Exception
        {
                String clientSentence;
                String capitalizedSentence;

                //create welcoming socket at port 6789
                ServerSocket welcomeSocket = new ServerSocket(6789);

                while (true) {
                        //block on welcoming socket for contact by a client
                        Socket connectionSocket = welcomeSocket.accept();

                        //create input stream attached to socket
                        BufferedReader inFromClient = new BufferedReader(new
                        InputStreamReader
                                (connectionSocket.getInputStream()));
```

Example: Java server using TCP (cont.)

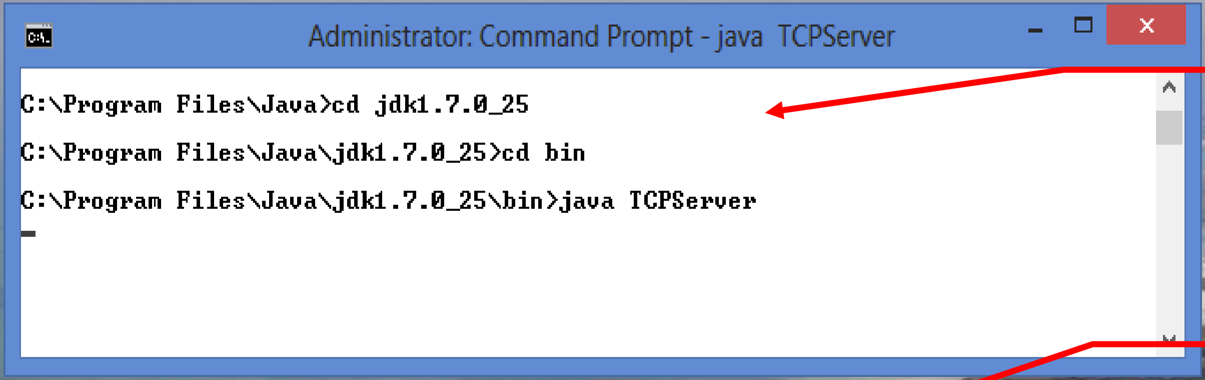
```
            //create output stream attached to socket
            DataOutputStream outToClient = new
            DataOutputStream(connectionSocket.getOutputStream());

            //read in line from the socket
            clientSentence = inFromClient.readLine();

            //process
            capitalizedSentence = clientSentence.toUpperCase() + '\n';

            //write out line to socket
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Example: Java client using TCP

```
//simple client application using TCP

import java.io.*;
import java.net.*;

class TCPClient {
        public static void main (String args[]) throws Exception
        {
                String sentence;
                String modifiedSentence;

                //create input stream
                BufferedReader inFromUser = new BufferedReader(new
                        InputStreamReader(System.in));

                //create client socket and connect to server
                Socket clientSocket = new Socket("localhost", 6789);

                //create output stream attached to socket
                DataOutputStream outToServer = new
                        DataOutputStream(clientSocket.getOutputStream());
```

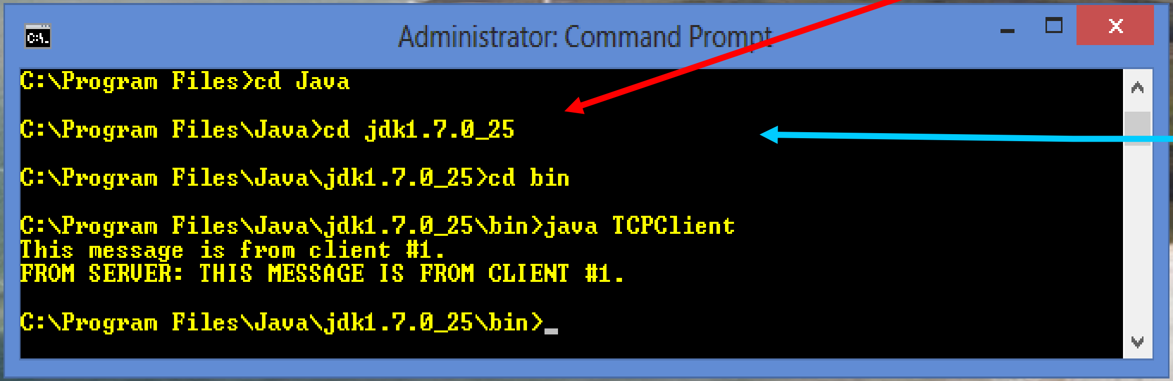
Example: Java client using TCP (cont.)

```
            //create input stream attached to socket
            BufferedReader inFromServer = new BufferedReader(new
            InputStreamReader (clientSocket.getInputStream()));

            sentence = inFromUser.readLine();

            //send line to the server
            outToServer.writeBytes(sentence + '\n');

            //read line coming back from the server
            modifiedSentence = inFromServer.readLine();

            System.out.println("FROM SERVER: " + modifiedSentence);

            clientSocket.close();
      }
}
```



```
C:\Program Files\Java>cd jdk1.7.0_25

C:\Program Files\Java\jdk1.7.0_25>cd bin

C:\Program Files\Java\jdk1.7.0_25\bin>java TCPServer
_
```

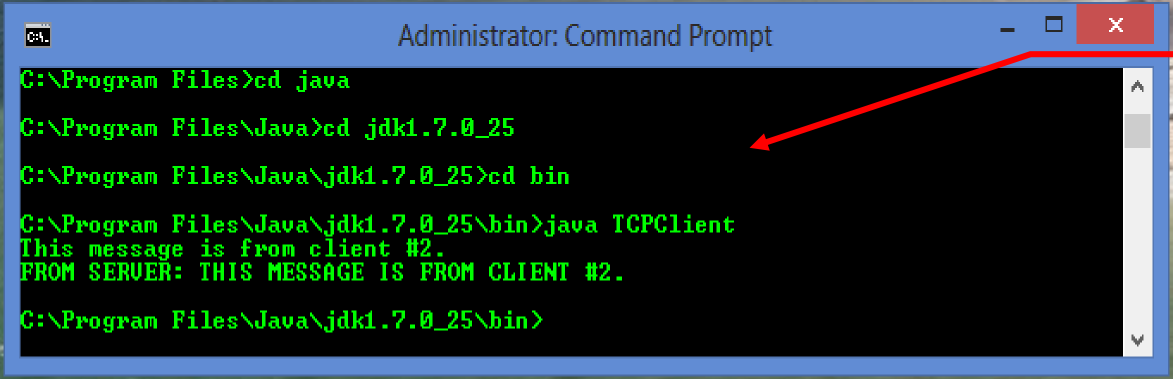
Start TCP Server executing



```
C:\Program Files>cd Java

C:\Program Files\Java>cd jdk1.7.0_25

C:\Program Files\Java\jdk1.7.0_25>cd bin

C:\Program Files\Java\jdk1.7.0_25\bin>java TCPClient
This message is from client #1.
FROM SERVER: THIS MESSAGE IS FROM CLIENT #1.

C:\Program Files\Java\jdk1.7.0_25\bin>_
```

Start a TCP Client executing and send message to server.

Server responds and client process terminates. The server is still executing.



```
C:\Program Files>cd java

C:\Program Files\Java>cd jdk1.7.0_25

C:\Program Files\Java\jdk1.7.0_25>cd bin

C:\Program Files\Java\jdk1.7.0_25\bin>java TCPClient
This message is from client #2.
FROM SERVER: THIS MESSAGE IS FROM CLIENT #2.

C:\Program Files\Java\jdk1.7.0_25\bin>
```

Another client begins execution and the cycle repeats.

A More Sophisticated TCP Client/Server Example Using GUIs

- Over the next few pages you will find the Java code for a more sophisticated client/server example.
- This example utilizes a GUI and makes things a bit more interesting from the programming point of view.
- Server process appears on pages 59-66. Server test process appears on page 75.
- Client process appears on pages 67-74. Client test process appears on page 76.

Sample Code: Java server using TCP with GUI

Page 1: Server

```
// TCPServerGUI.java
// Set up a TCP Server that will receive a connection from a client, send
// a string to the client, and close the connection.  GUI Version
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class TCPServerGUI extends JFrame
{
   private JTextField enterField; // inputs message from user
   private JTextArea displayArea; // display information to user
   private ObjectOutputStream output; // output stream to client
   private ObjectInputStream input; // input stream from client
private ServerSocket server; // server socket
private Socket connection; // connection to client
private int counter = 1; // counter of number of connections

// set up GUI
public TCPServerGUI()
{
   super( “TCP Server" );

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

   add( enterField, BorderLayout.NORTH );
```

Page 2: Server

Page 3: Server

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

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

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

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

Page 4: Server

```
            displayMessage( "\nServer terminated connection" );
         } // end catch
         finally
         {
            closeConnection(); //  close connection
            counter++;
         } // end finally
      } // end while
   } // end try
   catch ( IOException ioException )
   {
      ioException.printStackTrace();
   } // end catch
} // end method runServer

// wait for connection to arrive, then display connection info
private void waitForConnection() throws IOException
{
   displayMessage( "Waiting for connection\n" );
   connection = server.accept(); // allow server to accept connection
   displayMessage( "Connection " + counter + " received from: " +
      connection.getInetAddress().getHostName() );
} // end method waitForConnection
```

Page 5: Server

```
 // get streams to send and receive data
private void getStreams() throws IOException
{
   // set up output stream for objects
   output = new ObjectOutputStream( connection.getOutputStream() );
   output.flush(); // flush output buffer to send header information

   // set up input stream for objects
   input = new ObjectInputStream( connection.getInputStream() );

   displayMessage( "\nGot I/O streams\n" );
} // end method getStreams

// process connection with client
private void processConnection() throws IOException
{
   String message = "Connection successful";
   sendData( message ); // send connection successful message

   // enable enterField so server user can send messages
   setTextFieldEditable( true );
```

Page 6: Server

```
   do // process messages sent from client
   {
      try // read message and display it
      {
         message = ( String ) input.readObject(); // read new message
         displayMessage( "\n" + message ); // display message
      } // end try
      catch ( ClassNotFoundException classNotFoundException )
      {
         displayMessage( "\nUnknown object type received" );
      } // end catch

   } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
} // end method processConnection

// close streams and socket
private void closeConnection()
{
   displayMessage( "\nTerminating connection\n" );
   setTextFieldEditable( false ); // disable enterField
   try
   {
      output.close(); // close output stream
      input.close(); // close input stream
      connection.close(); // close socket
   } // end try
```

Page 7: Server

```
catch ( IOException ioException )
    {
      ioException.printStackTrace();
    } // end catch
  } // end method closeConnection

  // send message to client
  private void sendData( String message )
  {
    try // send object to client
    {
      output.writeObject( "SERVER>>> " + message );
      output.flush(); // flush output to client
      displayMessage( "\nSERVER>>> " + message );
    } // end try
    catch ( IOException ioException )
    {
      displayArea.append( "\nError writing object" );
    } // end catch
  } // end method sendData

  // manipulates displayArea in the event-dispatch thread
  private void displayMessage( final String messageToDisplay )
  {
    SwingUtilities.invokeLater(
      new Runnable()
```

```
         {
            public void run() // updates displayArea
            {
               displayArea.append( messageToDisplay ); // append message
            } // end method run
         } // end anonymous inner class
      ); // end call to SwingUtilities.invokeLater
   } // end method displayMessage

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

Page 8: Server

Sample Code: Java client using TCP with GUI

Page 1: Client

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

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

Page 2: Client

```
private Socket client; // socket to communicate with server

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

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

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

   add( enterField, BorderLayout.NORTH );
```

Page 3: Client

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

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

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

Page 4: Client

```
   finally
   {
      closeConnection(); // close connection
   } // end finally
} // end method runClient

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

   // create Socket to make connection to server
   client = new Socket( InetAddress.getByName( chatServer ), 12345 );

   // display connection information
   displayMessage( "Connected to: " +
      client.getInetAddress().getHostName() );
} // end method connectToServer

// get streams to send and receive data
private void getStreams() throws IOException
{
   // set up output stream for objects
   output = new ObjectOutputStream( client.getOutputStream() );
   output.flush(); // flush output buffer to send header information
   // set up input stream for objects
   input = new ObjectInputStream( client.getInputStream() );

   displayMessage( "\nGot I/O streams\n" );
} // end method getStreams

// process connection with server
private void processConnection() throws IOException
{
   // enable enterField so client user can send messages
   setTextFieldEditable( true );

   do // process messages sent from server
   {
      try // read message and display it
      {
         message = ( String ) input.readObject(); // read new message
         displayMessage( "\n" + message ); // display message
      } // end try
      catch ( ClassNotFoundException classNotFoundException )
      {
         displayMessage( "\nUnknown object type received" );
      } // end catch

   } while ( !message.equals( "SERVER>>> TERMINATE" ) );
} // end method processConnection
```

Page 5: Client

Page 6: Client

```
// close streams and socket
private void closeConnection()
{
   displayMessage( "\nClosing connection" );
   setTextFieldEditable( false ); // disable enterField

   try
   {
      output.close(); // close output stream
      input.close(); // close input stream
      client.close(); // close socket
   } // end try
   catch ( IOException ioException )
   {
      ioException.printStackTrace();
   } // end catch
} // end method closeConnection

// send message to server
private void sendData( String message )
{
   try // send object to server
   {
      output.writeObject( "CLIENT>>> " + message );
      output.flush(); // flush data to output
      displayMessage( "\nCLIENT>>> " + message );
   } // end try
```

Page 7: Client

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

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

Page 8: Client

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

Sample Code: Java server test

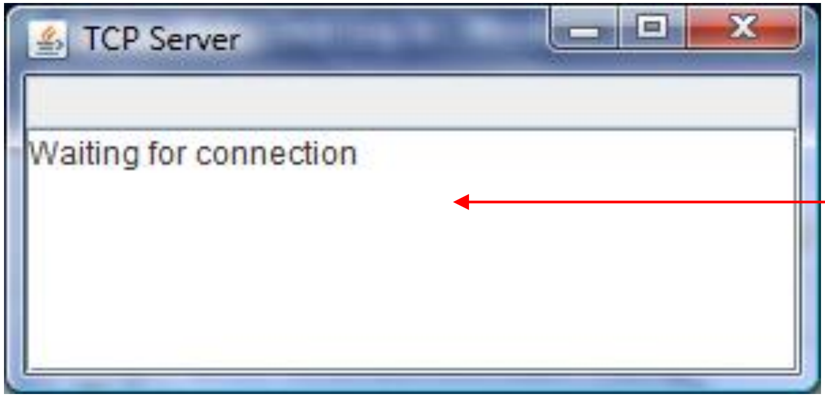
```java
// TCPServerTest.java
// Test the TCPServerGUI application.  GUI Version
import javax.swing.JFrame;

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

Sample Code: Java client test

```
// TCPClientTest.java
// Test the TCPClientGUI class.  GUI Version
import javax.swing.JFrame;

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

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

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

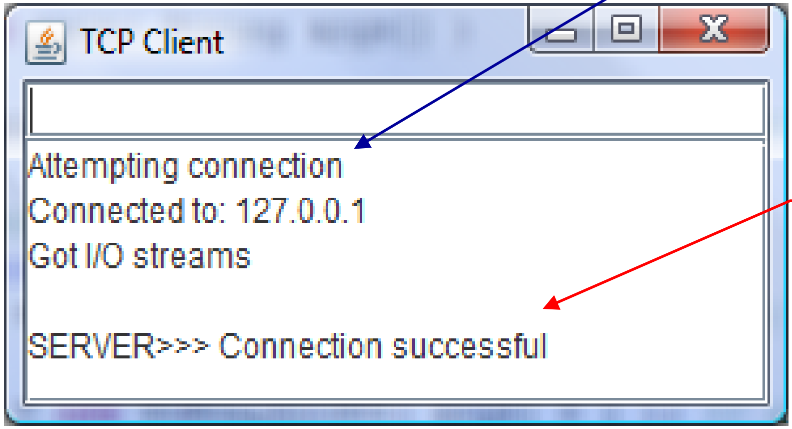
Special IP address to designate localhost.

Sample Screen Shots Illustrating Client/Server Processes



```
Waiting for connection
```

Server process initialized and waiting for a client connection.

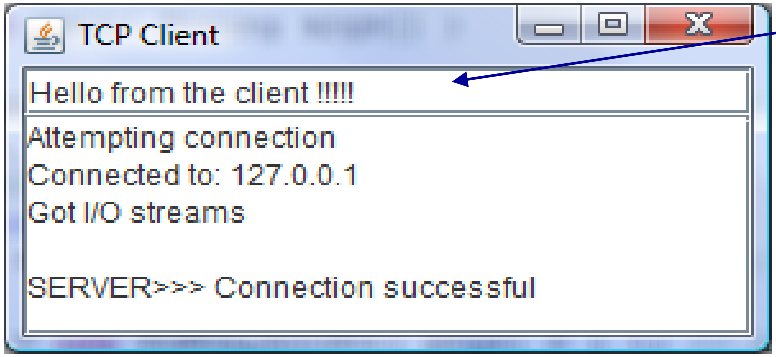


```
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
```

Client process attempts connection to localhost.

Server responds. Connection to server on localhost is successful. Stream connection is now established between server and client.

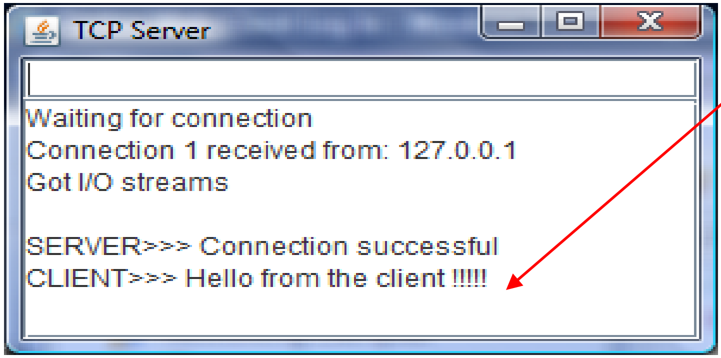
Sample Screen Shots Illustrating Client/Server Processes (cont.)



```
Hello from the client !!!!!
```

```
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
```

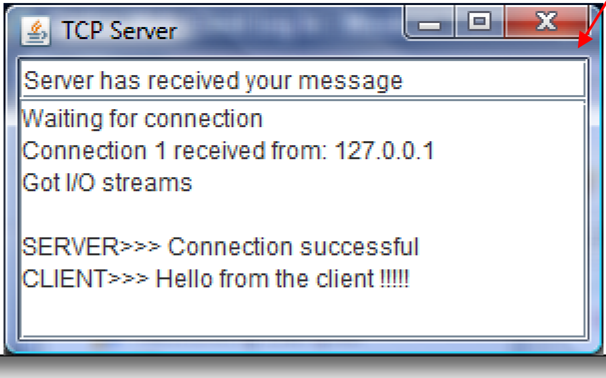
Client sends a message to the server.



```
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
```

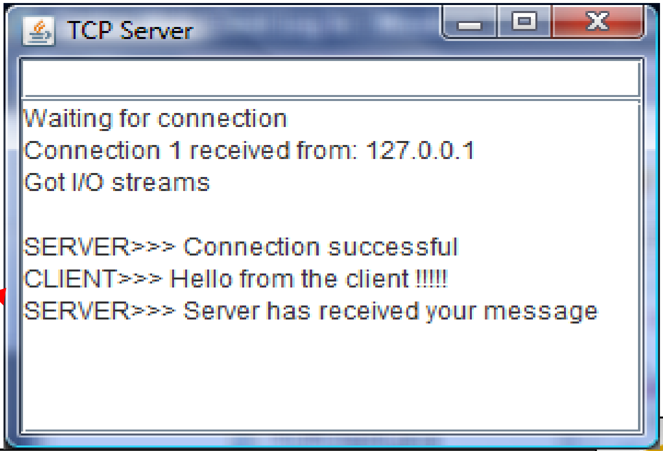
Server message from the client process.

Server responds to client.



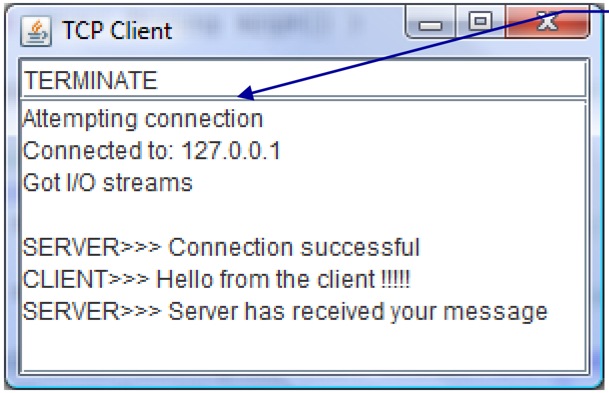
```
Server has received your message
```

```
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
```



```
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
SERVER>>> Server has received your message
```

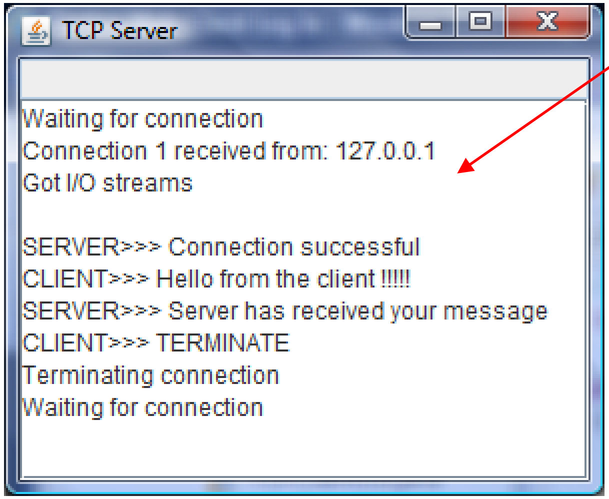
Sample Screen Shots Illustrating Client/Server Processes (cont.)



```
TERMINATE
```

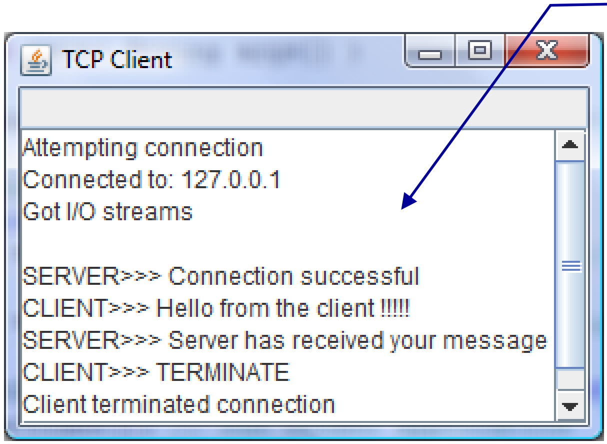
```
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
SERVER>>> Server has received your message
```

Client issues message to terminate connection.



```
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
SERVER>>> Server has received your message
CLIENT>>> TERMINATE
Terminating connection
Waiting for connection
```

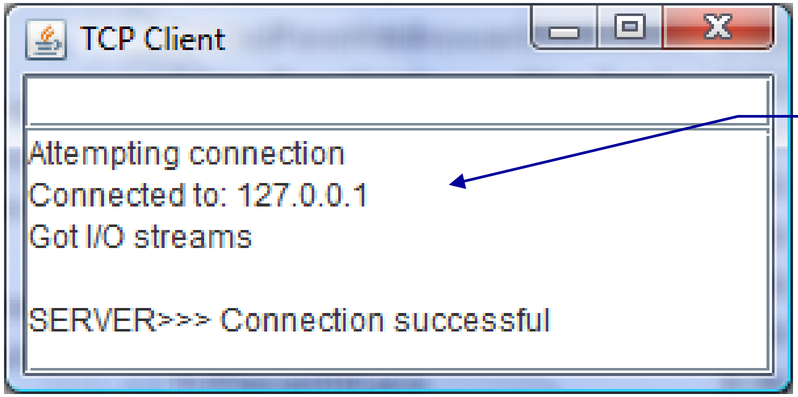
Server receives request from Client to terminate connection. Server responds by terminating connection and then blocking to await a subsequent connection.



```
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
SERVER>>> Server has received your message
CLIENT>>> TERMINATE
Client terminated connection
```

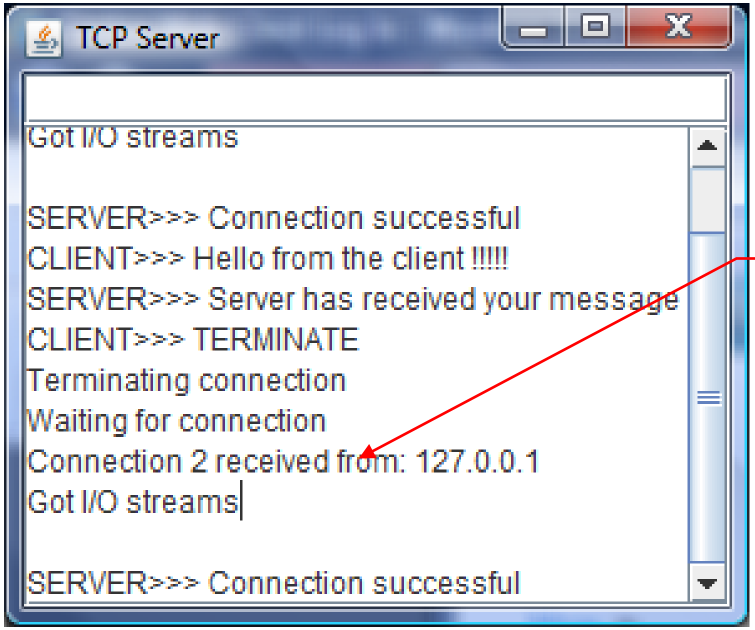
Message from Server that Client terminated connection and that the connection is now closed.

Sample Screen Shots Illustrating Client/Server Processes (cont.)



```
Attempting connection
Connected to: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
```

A subsequent connection request from another Client process is accepted by the Server. Server indicates that this is the second connection received from a client.



```
Got I/O streams

SERVER>>> Connection successful
CLIENT>>> Hello from the client !!!!!
SERVER>>> Server has received your message
CLIENT>>> TERMINATE
Terminating connection
Waiting for connection
Connection 2 received from: 127.0.0.1
Got I/O streams

SERVER>>> Connection successful
```

Server accepts a second connection and is now connected to the second client process.

Using Java's High-level Networking Capabilities

- As we saw earlier, the TCP and UDP protocols are at the transport layer within the Internet Reference Model. As far as Java is concerned, these provide "low-level" networking capability.

- Java also provides application layer networking protocol capabilities to allow for communication between applications.

- In the examples we have seen so far, it was the developer's responsibility to establish a connection between the client and the server (in the case of the UDP protocol, its more a process of establishing the sockets since there is no connection between the client and the server in this protocol).

Using Java's High-level Networking Capabilities (cont.)

- The next example illustrate Java's application layer capabilities which remove the responsibility of establishing the network connection from the developer.

- The example relies on a Web browser to establish the communication link to a Web server. (This one uses an applet to open a specific URL. Using a URL as an argument to the `showDocument` method of interface `AppletContext`, causes the browser in which the applet is executing to display that resource.)

Example 1 – SiteSelector Applet

```
<html>
<title>Site Selector</title>
<body>
   <applet code = "SiteSelector.class" width = "300" height = "75">
      <param name = "title0" value = "Java Home Page">
      <param name = "location0" value = "http://www.java.sun.com/">
      <param name = "title1" value = "CNT 47174 Home Page">
      <param name = "location1" value = "http://www.cs.ucf.edu/courses/cnt4714/spr2014">
      <param name = "title2" value = "World Cycling News">
      <param name = "location2" value = "http://www.cyclingnews.com/">
      <param name = "title3" value = "Formula 1 News">
      <param name = "location3" value = "http://www.formula1.com/">
   </applet>
</body>
</html>
```

HTML document to load the SiteSelctor Applet

Example 1 – SiteSelector Applet (cont.)

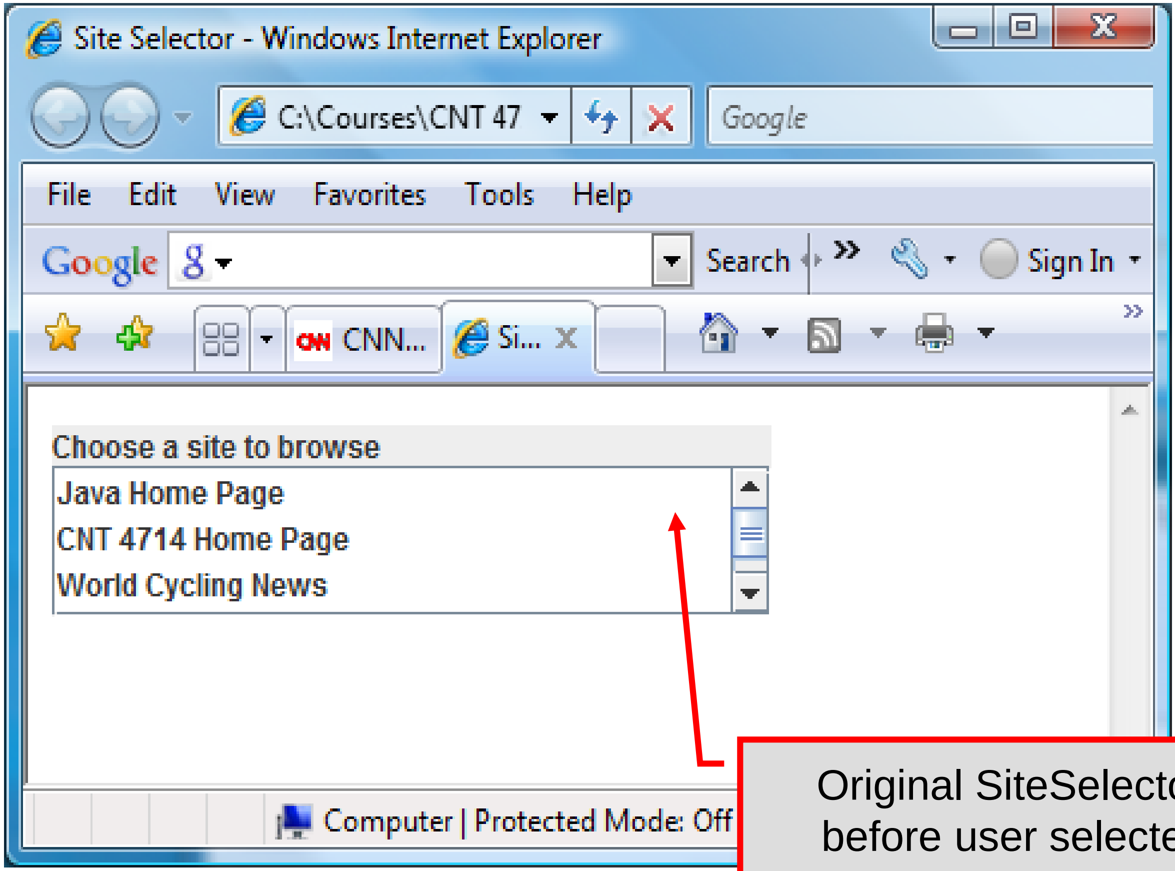
```
// SiteSelector.java
// This program loads a document from a URL.
import java.net.MalformedURLException;
import java.net.URL;
import java.util.HashMap;
import java.util.ArrayList;
import java.awt.BorderLayout;
import java.applet.AppletContext;
import javax.swing.JApplet;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

public class SiteSelector extends JApplet
{
   private HashMap< Object, URL > sites; // site names and URLs
   private ArrayList< String > siteNames; // site names
   private JList siteChooser; // list of sites to choose from

   // read HTML parameters and set up GUI
```

Example 1 – SiteSelector Applet (cont.)

```
public void init()
{
  sites = new HashMap< Object, URL >(); // create HashMap
  siteNames = new ArrayList< String >(); // create ArrayList
  // obtain parameters from HTML document
  getSitesFromHTMLParameters();
  // create GUI components and layout interface
  add( new JLabel( "Choose a site to browse" ), BorderLayout.NORTH );
  siteChooser = new JList( siteNames.toArray() ); // populate JList
  siteChooser.addListSelectionListener(
    new ListSelectionListener() // anonymous inner class
    {      // go to site user selected
      public void valueChanged( ListSelectionEvent event )
      {
        // get selected site name
        Object object = siteChooser.getSelectedValue();
        // use site name to locate corresponding URL
        URL newDocument = sites.get( object );
        // get applet container
        AppletContext browser = getAppletContext();
        // tell applet container to change pages
        browser.showDocument( newDocument );
      } // end method valueChanged
    } // end anonymous inner class
  }; // end call to addListSelectionListener
```

Example 1 – SiteSelector Applet (cont.)

```
   add( new JScrollPane( siteChooser ), BorderLayout.CENTER );
} // end method init
// obtain parameters from HTML document
private void getSitesFromHTMLParameters()
{
   String title; // site title
   String location; // location of site
   URL url; // URL of location
   int counter = 0; // count number of sites
   title = getParameter( "title" + counter ); // get first site title
   // loop until no more parameters in HTML document
   while ( title != null )
   {
      // obtain site location
      location = getParameter( "location" + counter );
      try // place title/URL in HashMap and title in ArrayList
      {
         url = new URL( location ); // convert location to URL
         sites.put( title, url ); // put title/URL in HashMap
         siteNames.add( title ); // put title in ArrayList
      } // end try
      catch ( MalformedURLException urlException )
      {
         urlException.printStackTrace();
      } // end catch
counter++;
         title = getParameter( "title" + counter
); // get next site title
      } // end while
   } // end method
getSitesFromHTMLParameters
} // end class SiteSelector
```


Site Selector - Windows Internet Explorer
C:\Courses\CNT 47
Google
File Edit View Favorites Tools Help
Search
Sign In
CNN...
Si...
Computer | Protected Mode: Off

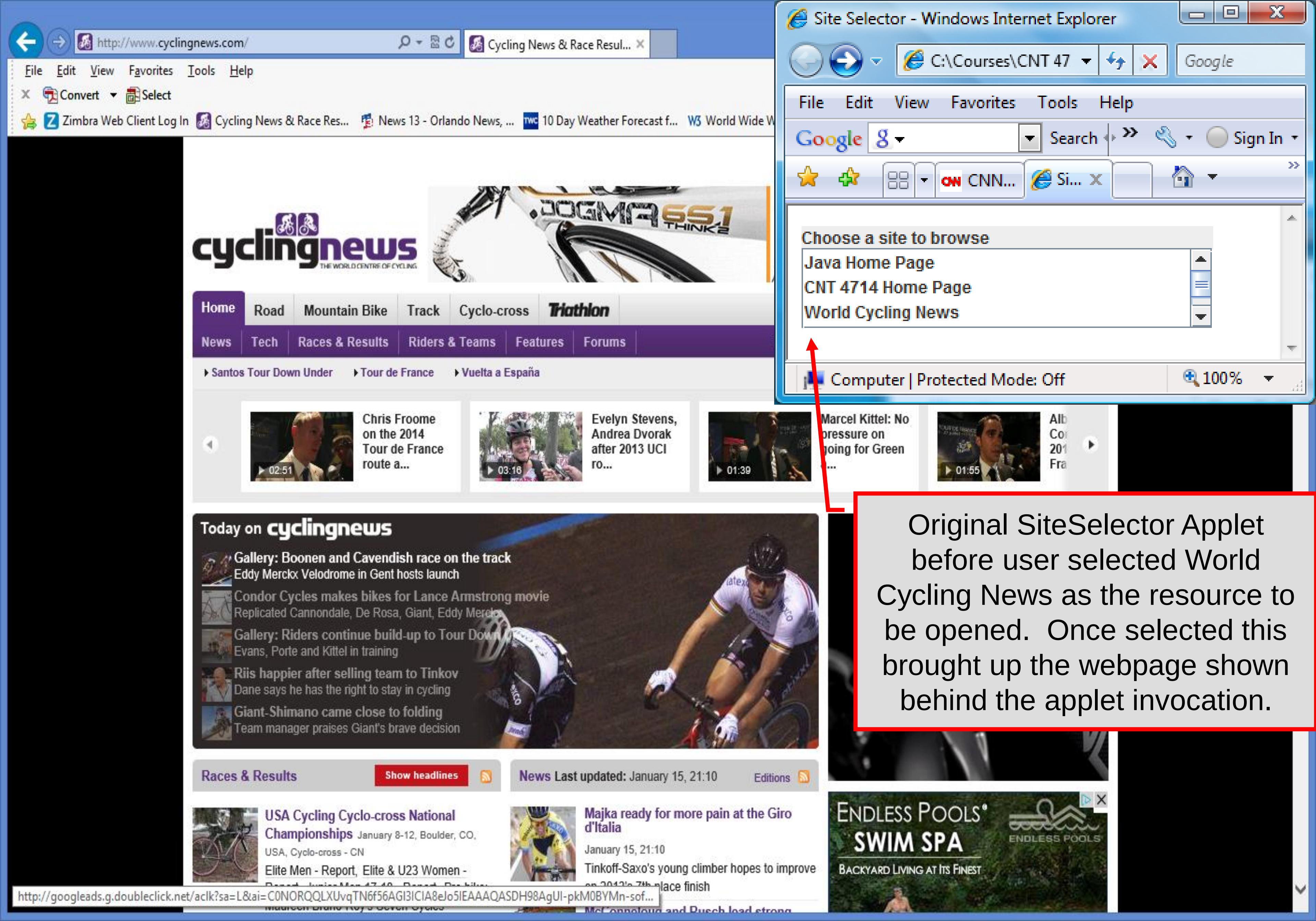
Choose a site to browse

Java Home Page

CNT 4714 Home Page

World Cycling News

Original SiteSelector Applet before user selected World Cycling News as the resource to be opened. Once selected this brought up the webpage shown behind the applet invocation.



Site Selector - Windows Internet Explorer
C:\Courses\CNT 47
Google
File Edit View Favorites Tools Help
Google
Search
Sign In
CNN...
Si...
Computer | Protected Mode: Off
100%
http://www.cyclingnews.com/
Cycling News & Race Resul...
File Edit View Favorites Tools Help
Convert
Select
Zimbra Web Client Log In
Cycling News & Race Res...
News 13 - Orlando News, ...
10 Day Weather Forecast f...
World Wide W
cyclingnews
THE WORLD CENTRE OF CYCLING
Home Road Mountain Bike Track Cyclo-cross Triathlon
News Tech Races & Results Riders & Teams Features Forums
Santos Tour Down Under
Tour de France
Vuelta a España

Choose a site to browse

Java Home Page

CNT 4714 Home Page

World Cycling News

Chris Froome on the 2014 Tour de France route a... 02:51

Evelyn Stevens, Andrea Dvorak after 2013 UCI ro... 03:16

Marcel Kittel: No pressure on going for Green ... 01:39

01:55

Today on cyclingnews

Gallery: Boonen and Cavendish race on the track
Eddy Merckx Velodrome in Gent hosts launch

Condor Cycles makes bikes for Lance Armstrong movie
Replicated Cannondale, De Rosa, Giant, Eddy Merckx

Gallery: Riders continue build-up to Tour Down Under
Evans, Porte and Kittel in training

Riis happier after selling team to Tinkov
Dane says he has the right to stay in cycling

Giant-Shimano came close to folding
Team manager praises Giant's brave decision

Races & Results Show headlines

USA Cycling Cyclo-cross National Championships January 8-12, Boulder, CO, USA, Cyclo-cross - CN
Elite Men - Report, Elite & U23 Women -

News Last updated: January 15, 21:10 Editions

Majka ready for more pain at the Giro d'Italia
January 15, 21:10
Tinkoff-Saxo's young climber hopes to improve on 2013's 7th place finish

ENDLESS POOLS® SWIM SPA BACKYARD LIVING AT ITS FINEST

http://googleads.g.doubleclick.net/aclk?sa=L&ai=C0NORQQLXUvqTN6f56AGI3ICIA8eJo5IEAAAQASDH98AgUI-pkM0BYMn-sof...

Original SiteSelector Applet before user selected World Cycling News as the resource to be opened. Once selected this brought up the webpage shown behind the applet invocation.

Secure Sockets Layer (SSL)

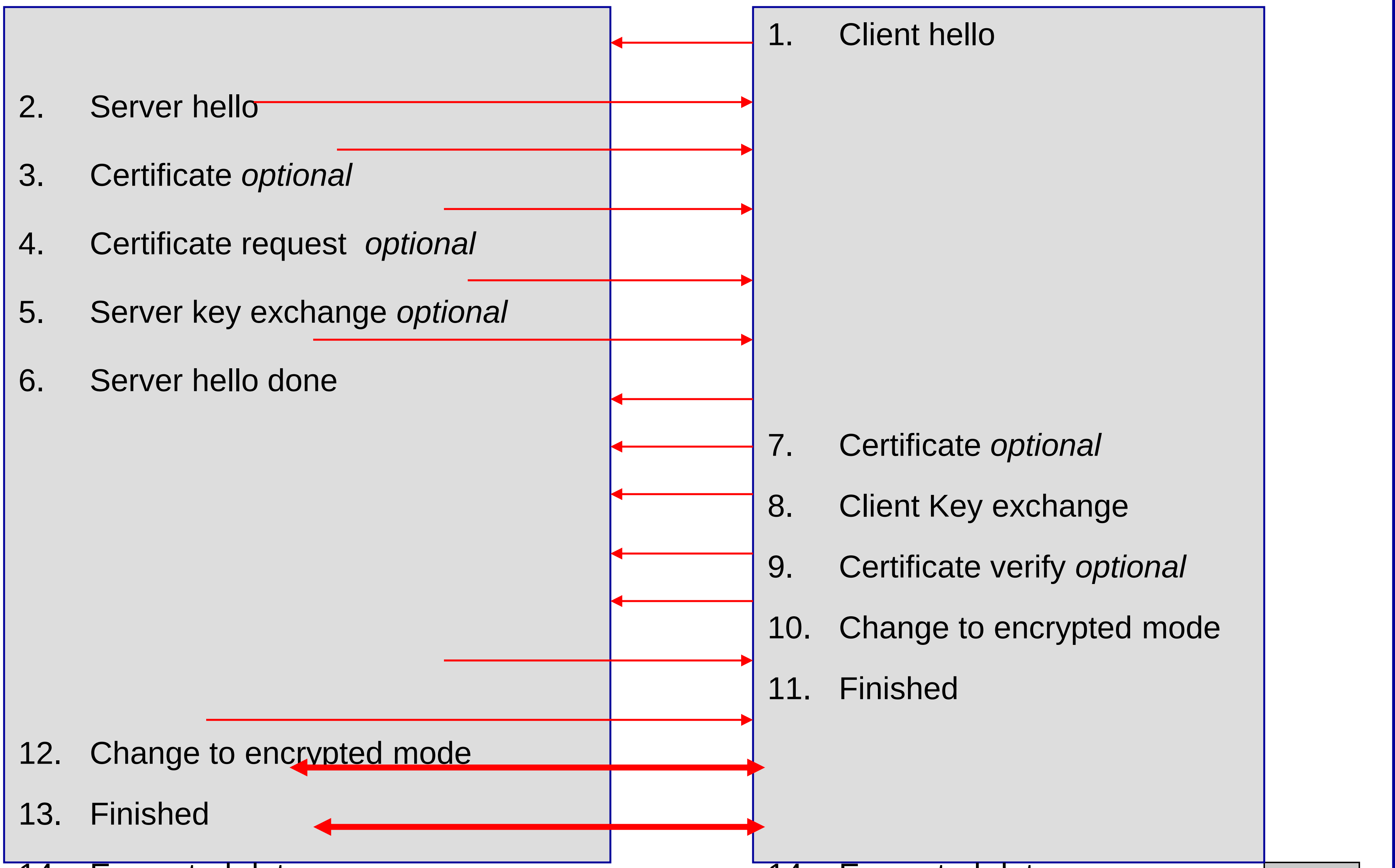
- Most e-business uses SSL for secure on-line transactions.
- SSL does not explicitly secure transactions, but rather secures connections.
- SSL implements public-key technology using the RSA algorithm (developed in 1977 at MIT by Ron Rivest, Adi Shamir, and Leonard Adleman) and digital certificates to authenticate the server in a transaction and to protect private information as it passes from one part to another over the Internet.
- SSL transactions do not require client authentication as most servers consider a valid credit-card number to be sufficient for authenticating a secure purchase.

How SSL Works

- Initially, a client sends a message to a server.
- The server responds and sends its digital certificate to the client for authentication.
- Using public-key cryptography to communicate securely, the client and server negotiate session keys to continue the transaction.
- Once the session keys are established, the communication proceeds between the client and server using the session keys and digital certificates.
- Encrypted data are passed through TCP/IP (just as regular packets over the Internet). However, before sending a message with TCP/IP, the SSL protocol breaks the information into blocks and compresses and encrypts those blocks.

How SSL Works (cont.)

- Once the data reach the receiver through TCP/IP, the SSL protocol decrypts the packets, then decompresses and assembles the data. It is these extra processes that provide an extra layer of security between TCP/IP and applications.
- SSL is used primarily to secure point-to-point connections using TCP/IP rather than UDP/IP.
- The SSL protocol allows for authentication of the server, the client, both, or neither. Although typically in Internet SSL sessions only the server is authenticated.



SERVER

2. Server hello
3. Certificate *optional*
4. Certificate request *optional*
5. Server key exchange *optional*
6. Server hello done
12. Change to encrypted mode
13. Finished
14. Encrypted data
15. Close messages

CLIENT

1. Client hello
7. Certificate *optional*
8. Client Key exchange
9. Certificate verify *optional*
10. Change to encrypted mode
11. Finished
14. Encrypted data
15. Close messages.

Details Of The SSL Protocol

- Use the diagram on the previous page to index the steps.

1. Client hello. The client sends the server information including the highest level of SSL it supports and a list of the cipher suites it supports including cryptographic algorithms and key sizes.
2. Server hello. The server chooses the highest version of SSL and the best cipher suite that both the client and server support and sends this information to the client.

Details Of The SSL Protocol (cont.)

3. Certificate. The server sends the client a certificate or a certificate chain. Optional but used whenever server authentication is required.
4. Certificate Request. If the server needs to authenticate the client, it sends the client a certificate request. In most Internet applications this message is rarely sent.
5. Server key exchange. The server sends the client a server key exchange message when the public key information sent in (3) above is not sufficient for key exchange.

Details Of The SSL Protocol (cont.)

6. Server hello done. The server tells the client that it is finished with its initial negotiation messages.
7. Certificate. If the server requests a certificate from the client in (4), the client sends its certificate chain, just as the server did in (3).
8. Client key exchange. The client generates information used to create a key to use for symmetric encryption. For RSA, the client then encrypts this key information with the server’s public key and sends it to the server.

Details Of The SSL Protocol (cont.)

9. Certificate verify. This message is sent when a client presents a certificate as above. Its purpose is to allow the server to complete the process of authenticating the client. When this message is used, the client sends information that it digitally signs using a cryptographic hash function. When the server decrypts this information with the client's public key, the server is able to authenticate the client.
10. Change to encrypted mode. The client sends a message telling the server to change to encrypted mode.
11. Finished. The client tells the server that it is ready for secure data communication to begin.

Details Of The SSL Protocol (cont.)

12. Change to encrypted mode. The server sends a message telling the client to switch to encrypted mode.
13. Finished. The server tells the client that it is ready for secure data communication to begin. This marks the end of the SSL handshake.
14. Encrypted data. The client and the server communicate using the symmetric encryption algorithm and the cryptographic hash function negotiated in (1) and (2), and using the secret key that the client sent to the server in (8).
15. Close messages. At the end of the connection, each side will send a close_notify message to inform the peer that the connection is closed.

Java Secure Socket Extension (JSSE)

- SSL encryption has been integrated into Java technology through the Java Secure Socket Extension (JSSE). JSSE has been an integral part of Java (not a separately loaded extension) since version 1.4.
- JSSE provides encryption, message integrity checks, and authentication of the server and client.
- JSSE uses keystores to secure storage of key pairs and certificates used in PKI (Public Key Infrastructure which integrates public-key cryptography with digital certificates and certificate authorities to authenticate parties in a transaction.)
- A truststore is a keystore that contains keys and certificates used to validate the identities of servers and clients.

Java Secure Socket Extension (JSSE) (cont.)

- Using secure sockets in Java is very similar to using the non-secure sockets that we have already seen.
- JSSE hides the details of the SSL protocol and encryption from the programmer entirely.
- The final example in this set of notes involves a client application that attempts to logon to a server using SSL.
- **NOTE:** Before attempting to execute this application, look at the code first and then go to page 105 for execution details. This application will not execute correctly unless you follow the steps beginning on page 105.

```
// LoginServer.java
// LoginServer uses an SSLServerSocket to demonstrate JSSE's SSL implementation.
package securitystuff.jsse;

// Java core packages
import java.io.*;

// Java extension packages
import javax.net.ssl.*;

public class LoginServer {
   private static final String CORRECT_USER_NAME = "Mark";
   private static final String CORRECT_PASSWORD = "CNT 4714";
   private SSLServerSocket serverSocket;

   // LoginServer constructor
   public LoginServer() throws Exception
   {
      // SSLServerSocketFactory for building SSLServerSockets
      SSLServerSocketFactory socketFactory =
         ( SSLServerSocketFactory )
            SSLServerSocketFactory.getDefault();
       // create SSLServerSocket on specified port
      serverSocket = ( SSLServerSocket )
         socketFactory.createServerSocket( 7070 );

   } // end LoginServer constructor
```

LoginServer.java

SSL Server Implementation

Use default SSLServerSocketFactory to create SSL sockets

SSL socket will listen on port 7070

```
// start server and listen for clients
private void runServer()
{
   // perpetually listen for clients
   while ( true ) {
      // wait for client connection and check login information
      try {
            System.err.println( "Waiting for connection..." );
             // create new SSLSocket for client
            SSLSocket socket =  ( SSLSocket ) serverSocket.accept();
           // open BufferedReader for reading data from client
           BufferedReader input = new BufferedReader(
              new InputStreamReader( socket.getInputStream() ) );
           // open PrintWriter for writing data to client
          PrintWriter output = new PrintWriter(
            new OutputStreamWriter(socket.getOutputStream() ) );
          String userName = input.readLine();
          String password = input.readLine();
           if ( userName.equals( CORRECT_USER_NAME ) &&
              password.equals( CORRECT_PASSWORD ) ) {
            output.println( "Welcome, " + userName );
         }
         else {
            output.println( "Login Failed." );
         }
```

Accept new client connection. This is a blocking call that returns an SSLSocket when a client connects.

Get input and output streams just as with normal sockets.

Validate user name and password against constants on the server.

```
            // clean up streams and SSLSocket
             output.close();
             input.close();
             socket.close();

          } // end try

          // handle exception communicating with client
          catch ( IOException ioException ) {
             ioException.printStackTrace();
          }

       } // end while

    } // end method runServer

    // execute application
    public static void main( String args[] ) throws Exception
    {
       LoginServer server = new LoginServer();
       server.runServer();
    }
 } //end LoginServer class
```

Close down I/O streams and the socket

```
// LoginClient.java
// LoginClient uses an SSLSocket to transmit fake login information to LoginServer.
package securitystuff.jsse;
// Java core packages
import java.io.*;
// Java extension packages
import javax.swing.*;
import javax.net.ssl.*;

public class LoginClient {
    // LoginClient constructor
   public LoginClient()
   {
      // open SSLSocket connection to server and send login
      try {
         // obtain SSLSocketFactory for creating SSLSockets
         SSLSocketFactory socketFactory =  ( SSLSocketFactory ) SSLSocketFactory.getDefault();
         // create SSLSocket from factory
         SSLSocket socket =   ( SSLSocket ) socketFactory.createSocket(  "localhost", 7070 );
         // create PrintWriter for sending login to server
         PrintWriter output = new PrintWriter(
            new OutputStreamWriter( socket.getOutputStream() ) );
          // prompt user for user name
         String userName = JOptionPane.showInputDialog( null, "Enter User Name:" );
          // send user name to server
         output.println( userName );
```

LoginClient.java

Client Class for SSL Implementation

Use default SSLSocketFactory to create SSL sockets

SSL socket will listen on port 7070

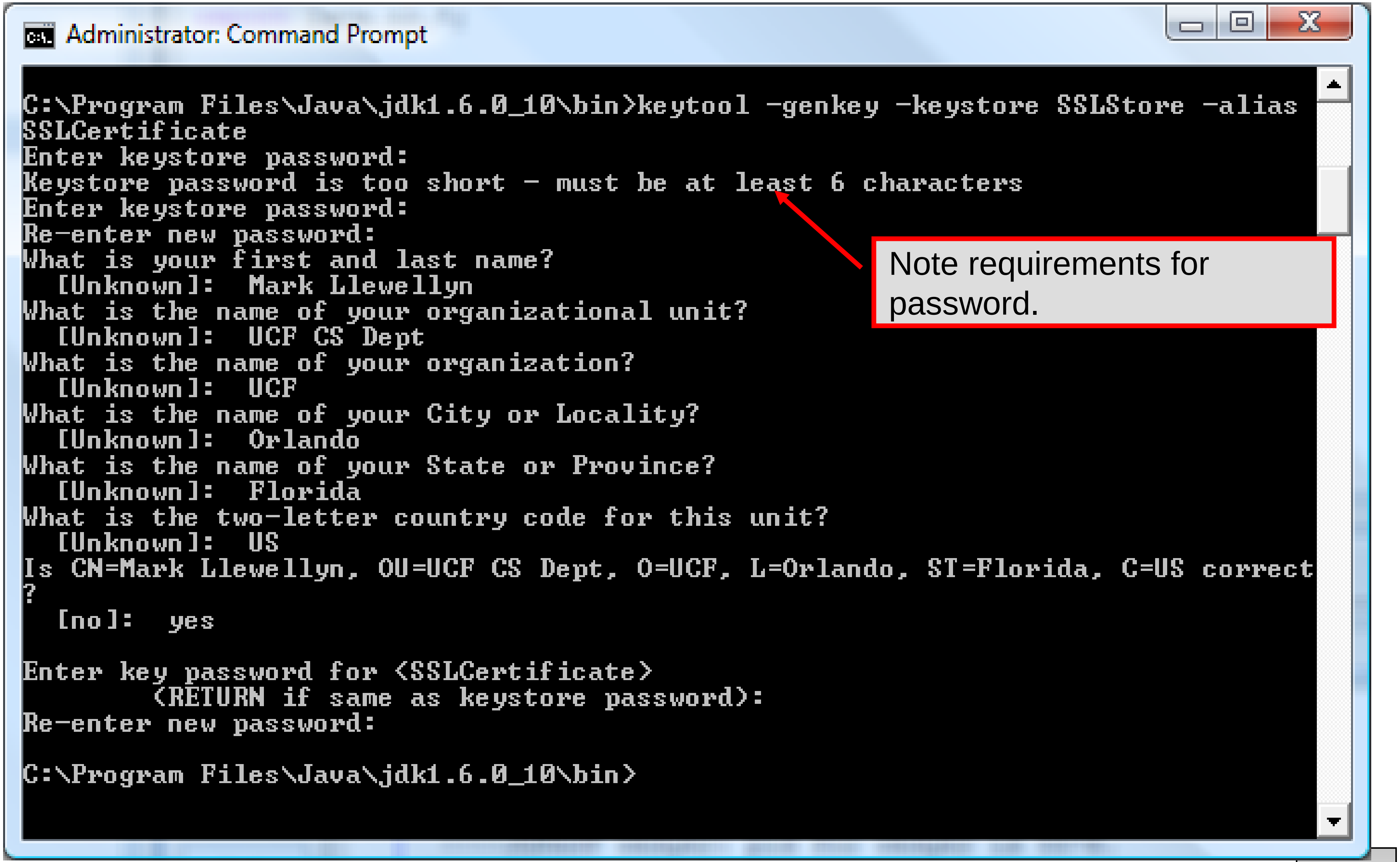
```
         // prompt user for password
         String password = JOptionPane.showInputDialog( null, "Enter Password:" );
         // send password to server
         output.println( password );
         output.flush();
          // create BufferedReader for reading server response
         BufferedReader input = new BufferedReader(
            new InputStreamReader( socket.getInputStream () ) );
          // read response from server
         String response = input.readLine();
         // display response to user
         JOptionPane.showMessageDialog( null, response );
          // clean up streams and SSLSocket
         output.close();
         input.close();
         socket.close();
      } // end try
      // handle exception communicating with server
      catch ( IOException ioException ) {
         ioException.printStackTrace();
      }
      // exit application
      finally {
         System.exit( 0 );
      }
   } // end LoginClient constructor
```

```
// execute application
   public static void main( String
args[] )
   {
      new LoginClient();
   }
}
```

Creating Keystore and Certificate

- Before you can execute the LoginServer and LoginClient application using SSL you will need to create a keystore and certificate for the SSL to operate correctly.
- Utilizing the keytool (a key and certificate management tool) in Java generate a keystore and a certificate for this server application. See the next slide for an example.
- We'll use the same keystore for both the server and the client although in reality these are often different. The client's truststore, in real-world applications, would contain trusted certificates, such as those from certificate authorities (e.g. VeriSign www.verisign.com etc.).

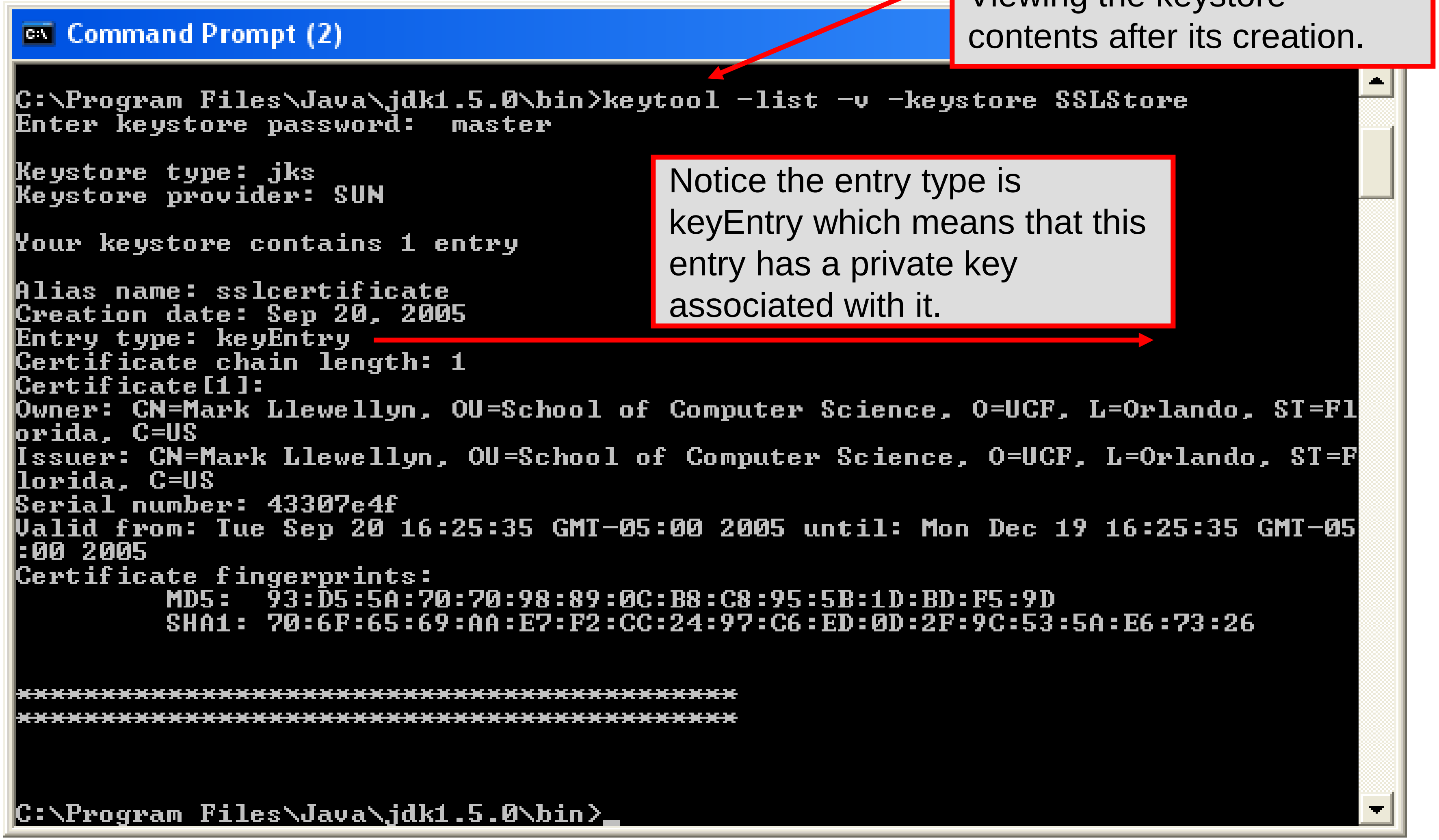
Creating Keystore and Certificate



```
C:\Program Files\Java\jdk1.6.0_10\bin>keytool -genkey -keystore SSLStore -alias
SSLCertificate
Enter keystore password:
Keystore password is too short - must be at least 6 characters
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Mark Llewellyn
What is the name of your organizational unit?
  [Unknown]:  UCF CS Dept
What is the name of your organization?
  [Unknown]:  UCF
What is the name of your City or Locality?
  [Unknown]:  Orlando
What is the name of your State or Province?
  [Unknown]:  Florida
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Mark Llewellyn, OU=UCF CS Dept, O=UCF, L=Orlando, ST=Florida, C=US correct
?
  [no]:  yes

Enter key password for <SSLCertificate>
        (RETURN if same as keystore password):
Re-enter new password:

C:\Program Files\Java\jdk1.6.0_10\bin>
```

Note requirements for password.

Creating Keystore and Certificate

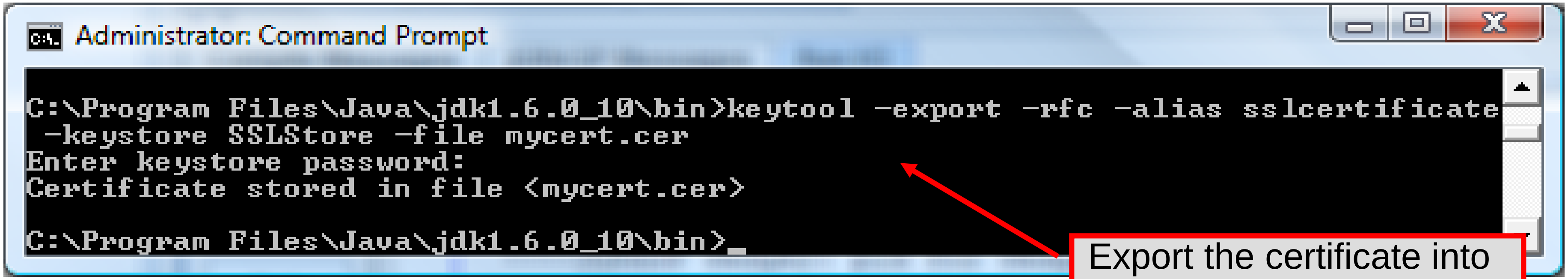


```
C:\Program Files\Java\jdk1.5.0\bin>keytool -list -v -keystore SSLStore
Enter keystore password:  master

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: sslcertificate
Creation date: Sep 20, 2005
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Mark Llewellyn, OU=School of Computer Science, O=UCF, L=Orlando, ST=Fl
orida, C=US
Issuer: CN=Mark Llewellyn, OU=School of Computer Science, O=UCF, L=Orlando, ST=F
lorida, C=US
Serial number: 43307e4f
Valid from: Tue Sep 20 16:25:35 GMT-05:00 2005 until: Mon Dec 19 16:25:35 GMT-05
:00 2005
Certificate fingerprints:
         MD5:  93:D5:5A:70:70:98:89:0C:B8:C8:95:5B:1D:BD:F5:9D
         SHA1: 70:6F:65:69:AA:E7:F2:CC:24:97:C6:ED:0D:2F:9C:53:5A:E6:73:26


*******************************************
*******************************************


C:\Program Files\Java\jdk1.5.0\bin>_
```

Viewing the keystore contents after its creation.

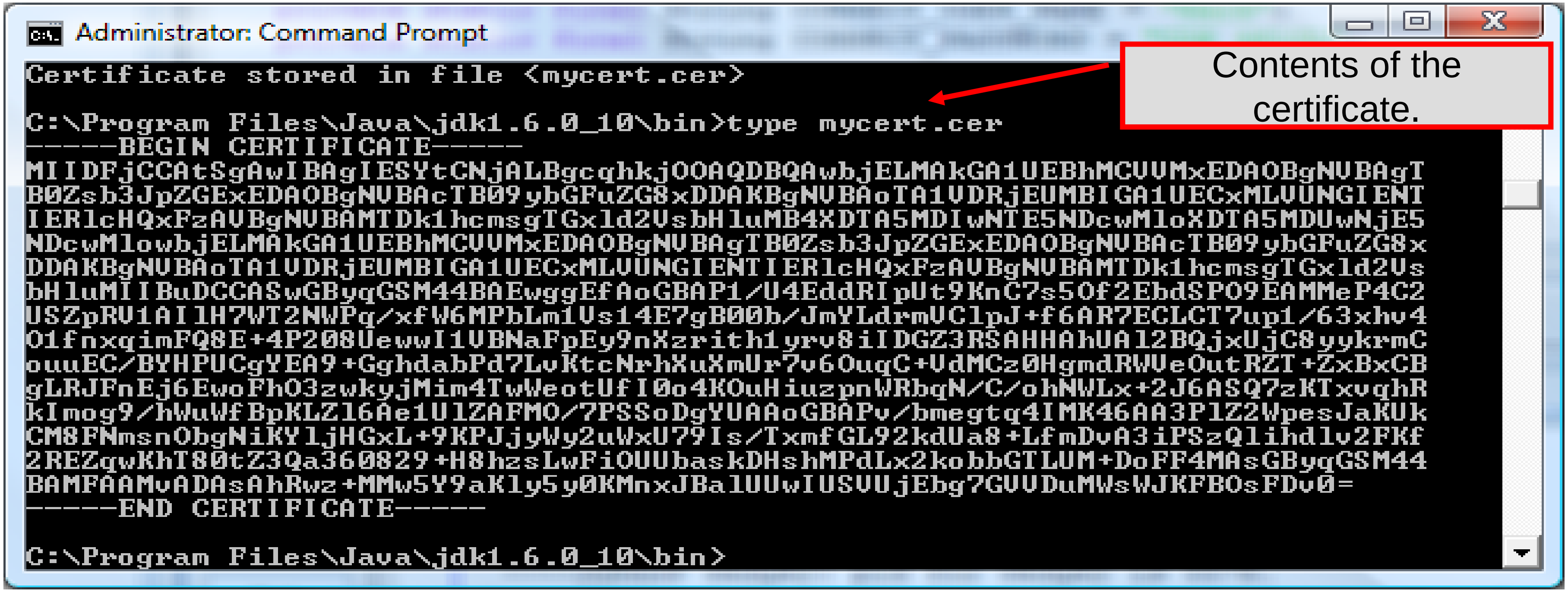
Notice the entry type is keyEntry which means that this entry has a private key associated with it.

Creating Keystore and Certificate



```
C:\Program Files\Java\jdk1.6.0_10\bin>keytool -export -rfc -alias sslcertificate
 -keystore SSLStore -file mycert.cer
Enter keystore password:
Certificate stored in file <mycert.cer>

C:\Program Files\Java\jdk1.6.0_10\bin>_
```

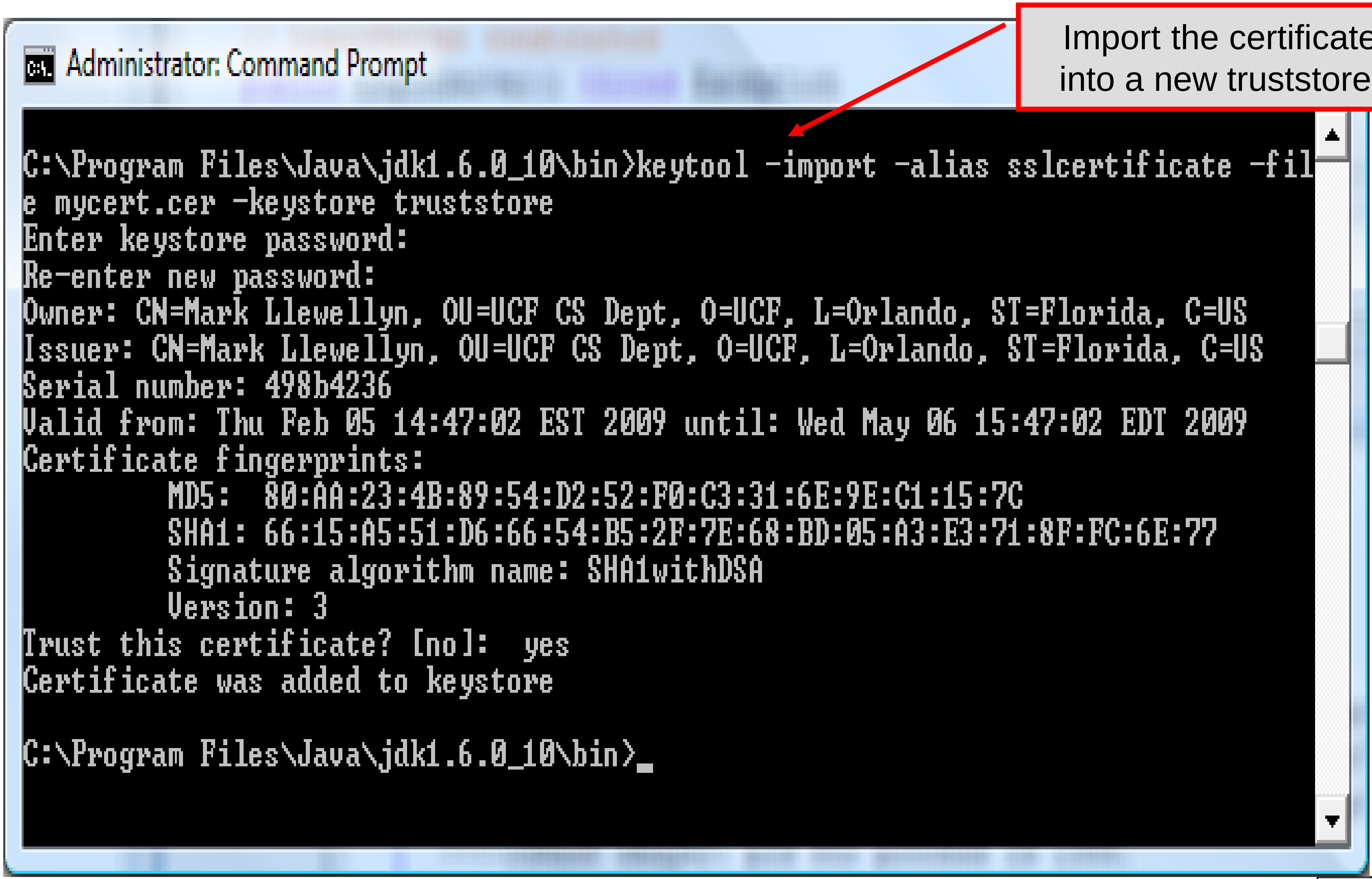
Export the certificate into a certificate file.



```
Certificate stored in file <mycert.cer>

C:\Program Files\Java\jdk1.6.0_10\bin>type mycert.cer
-----BEGIN CERTIFICATE-----
MIIDFjCCAtSgAwIBAgIESYtCNjALBgcqhkjOOAQDBQAwbjELMAkGA1UEBhMCVVMxEDAOBgNVBAgT
B0Zsb3JpZGExEDAOBgNVBAcTB09ybGFuZG8xDDAKBgNVBAoTA1VDRjEUMBIGA1UECxMLVVNGIENT
IERlcHQxFzAVBgNVBAMTDk1hcmsgTGxld2VsbHluMB4XDTA5MDIwNTE5NDcwMloXDTA5MDUwNjE5
NDcwMlowbjELMAkGA1UEBhMCVVMxEDAOBgNVBAgTB0Zsb3JpZGExEDAOBgNVBAcTB09ybGFuZG8x
DDAKBgNVBAoTA1VDRjEUMBIGA1UECxMLVVNGIENTIERlcHQxFzAVBgNVBAMTDk1hcmsgTGxld2Vs
bHluMIIBuDCCASwGByqGSM44BAEwggEfAoGBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2
USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4
O1fnxqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAhUAl2BQjxUjC8yykrmC
ouuEC/BYHPUCgYEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCB
gLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhR
kImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoDgYUAAoGBAPv/bmegtq4IMK46AA3PlZ2WpesJaKUk
CM8FNmsnObgNiKYljHGxL+9KPJjyWy2uWxU79Is/TxmfGL92kdUa8+LfmDvA3iPSzQlihdlv2FKf
2REZqwKhT80tZ3Qa360829+H8hzsLwFiOUUbaskDHshMPdLx2kobbGTLUM+DoFF4MAsGByqGSM44
BAMFAAMvADAsAhRwz+MMw5Y9aKly5y0KMnxJBalUUwIUSVUjEbg7GVVDuMWsWJKFBOsFDv0=
-----END CERTIFICATE-----

C:\Program Files\Java\jdk1.6.0_10\bin>
```

Contents of the certificate.

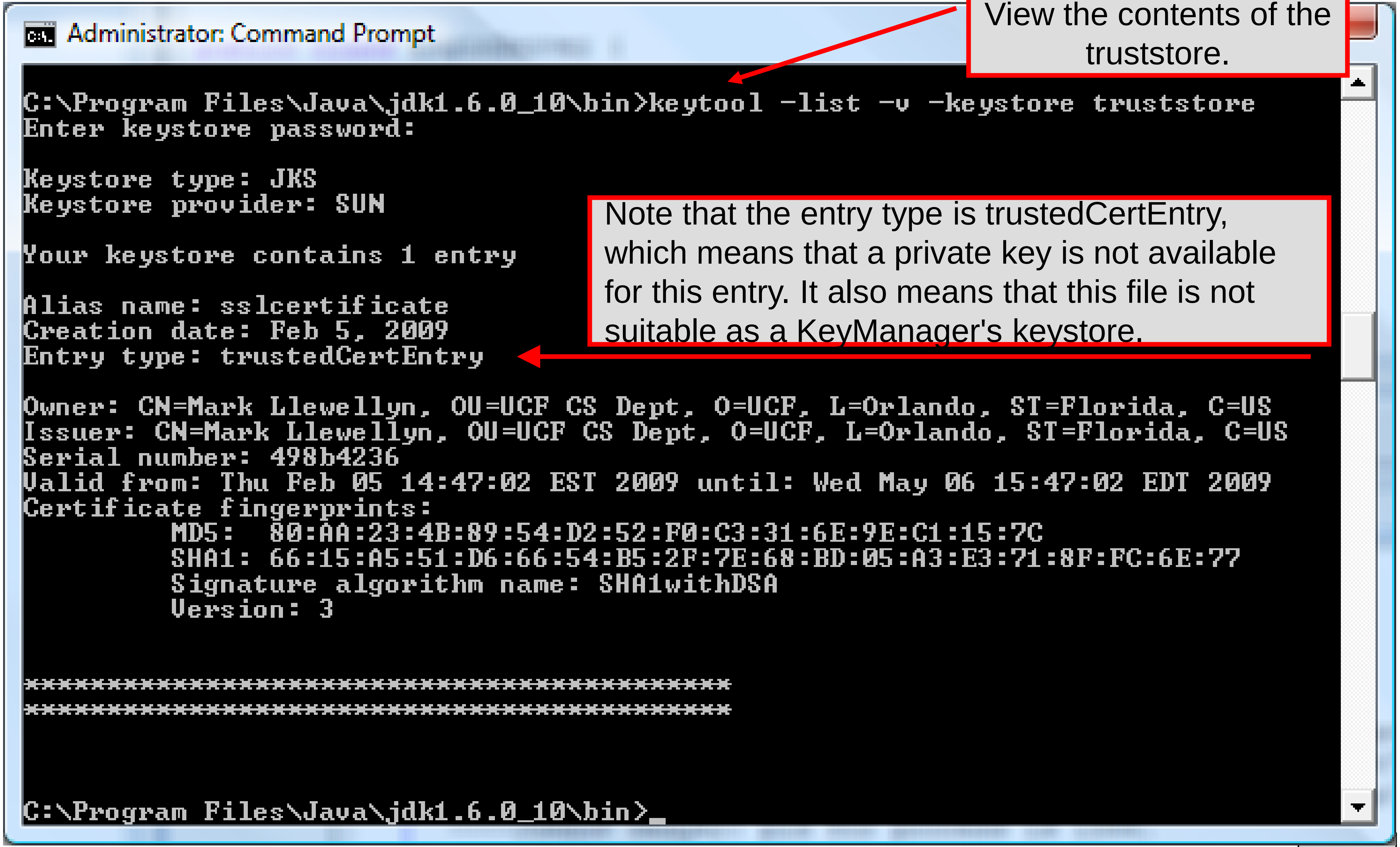
Creating Keystore and Certificate



```
C:\Program Files\Java\jdk1.6.0_10\bin>keytool -import -alias sslcertificate -fil
e mycert.cer -keystore truststore
Enter keystore password:
Re-enter new password:
Owner: CN=Mark Llewellyn, OU=UCF CS Dept, O=UCF, L=Orlando, ST=Florida, C=US
Issuer: CN=Mark Llewellyn, OU=UCF CS Dept, O=UCF, L=Orlando, ST=Florida, C=US
Serial number: 498b4236
Valid from: Thu Feb 05 14:47:02 EST 2009 until: Wed May 06 15:47:02 EDT 2009
Certificate fingerprints:
         MD5:  80:AA:23:4B:89:54:D2:52:F0:C3:31:6E:9E:C1:15:7C
         SHA1: 66:15:A5:51:D6:66:54:B5:2F:7E:68:BD:05:A3:E3:71:8F:FC:6E:77
         Signature algorithm name: SHA1withDSA
         Version: 3
Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\Program Files\Java\jdk1.6.0_10\bin>_
```

Import the certificate into a new truststore.

Creating Keystore and Certificate



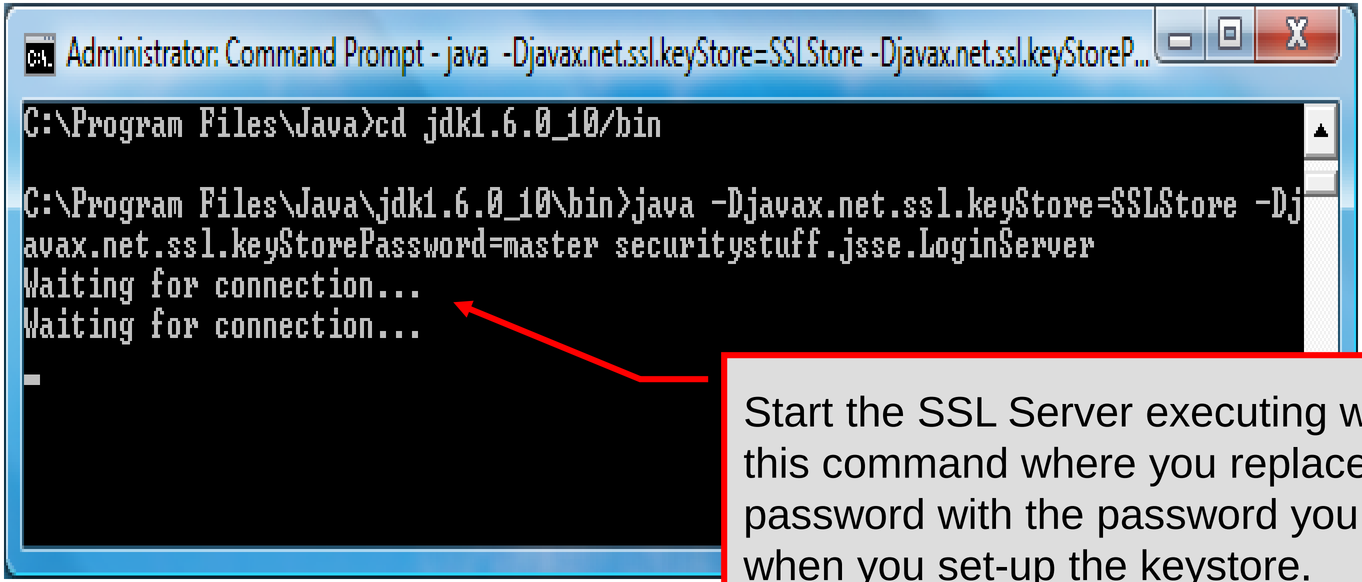
```
C:\Program Files\Java\jdk1.6.0_10\bin>keytool -list -v -keystore truststore
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: sslcertificate
Creation date: Feb 5, 2009
Entry type: trustedCertEntry

Owner: CN=Mark Llewellyn, OU=UCF CS Dept, O=UCF, L=Orlando, ST=Florida, C=US
Issuer: CN=Mark Llewellyn, OU=UCF CS Dept, O=UCF, L=Orlando, ST=Florida, C=US
Serial number: 498b4236
Valid from: Thu Feb 05 14:47:02 EST 2009 until: Wed May 06 15:47:02 EDT 2009
Certificate fingerprints:
         MD5:  80:AA:23:4B:89:54:D2:52:F0:C3:31:6E:9E:C1:15:7C
         SHA1: 66:15:A5:51:D6:66:54:B5:2F:7E:68:BD:05:A3:E3:71:8F:FC:6E:77
         Signature algorithm name: SHA1withDSA
         Version: 3


*******************************************
*******************************************



C:\Program Files\Java\jdk1.6.0_10\bin>_
```

View the contents of the truststore.

Note that the entry type is trustedCertEntry, which means that a private key is not available for this entry. It also means that this file is not suitable as a KeyManager's keystore.

Launching the Secure Server

- Now you are ready to start the server executing from a command prompt…
- Once started, the server simply waits for a connection from a client. The example below illustrates the server after waiting for several minutes.



Administrator: Command Prompt - java -Djavax.net.ssl.keyStore=SSLStore -Djavax.net.ssl.keyStoreP...

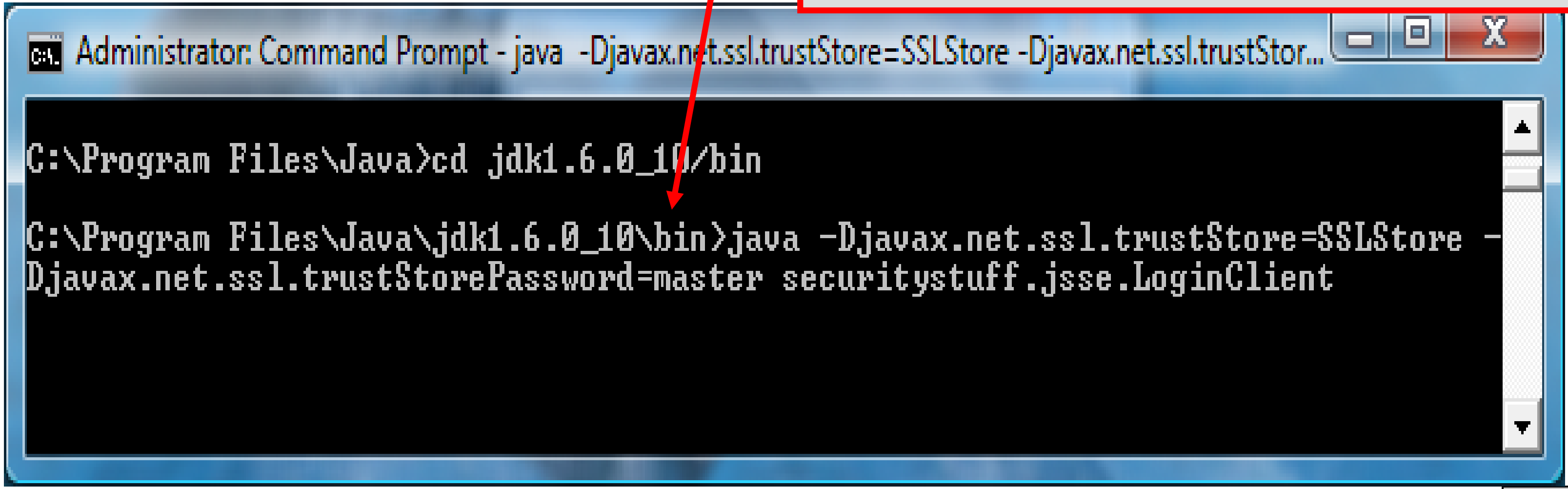
```
C:\Program Files\Java>cd jdk1.6.0_10/bin

C:\Program Files\Java\jdk1.6.0_10\bin>java -Djavax.net.ssl.keyStore=SSLStore -Dj
avax.net.ssl.keyStorePassword=master securitystuff.jsse.LoginServer
Waiting for connection...
Waiting for connection...
```

Start the SSL Server executing with this command where you replace this password with the password you used when you set-up the keystore.

Launching the SSL Client

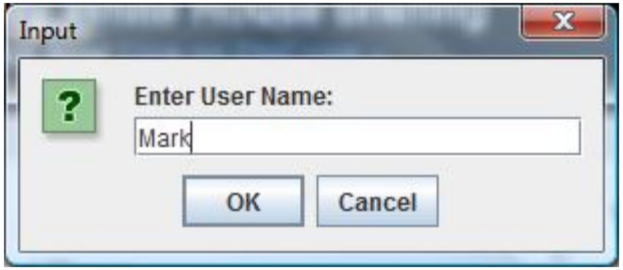
- Start a client application executing from a new command window…
- Once the client establishes communication with the server, the authentication process begins.

Start the SSL Client application executing with this command where you replace this password with the password you used when you set-up the keystore. Since we are using the same keystore for the server and the client…these will be the same.



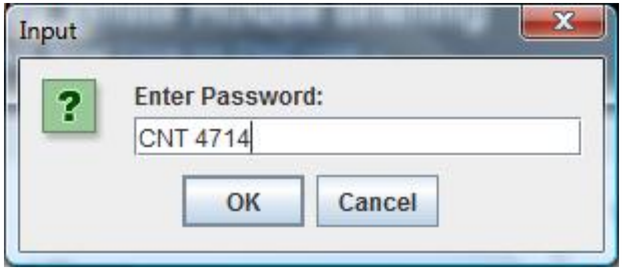
Administrator: Command Prompt - java -Djavax.net.ssl.trustStore=SSLStore -Djavax.net.ssl.trustStor...

```
C:\Program Files\Java>cd jdk1.6.0_10/bin

C:\Program Files\Java\jdk1.6.0_10\bin>java -Djavax.net.ssl.trustStore=SSLStore -
Djavax.net.ssl.trustStorePassword=master securitystuff.jsse.LoginClient
```



Enter User Name:

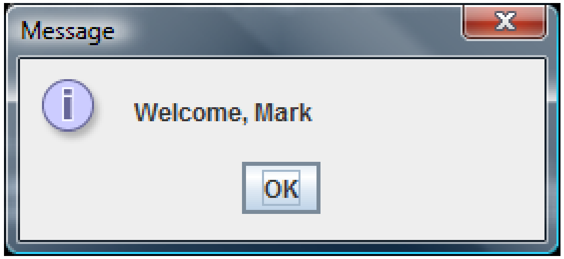
Mark



Enter Password:

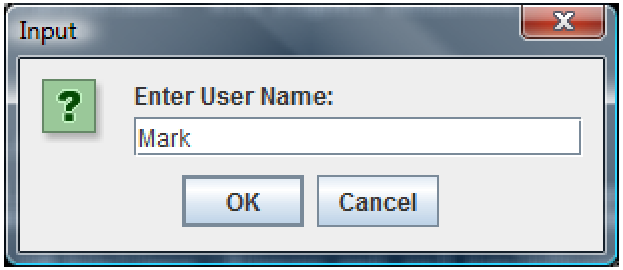
CNT 4714

User enters username and password which are sent to the server.



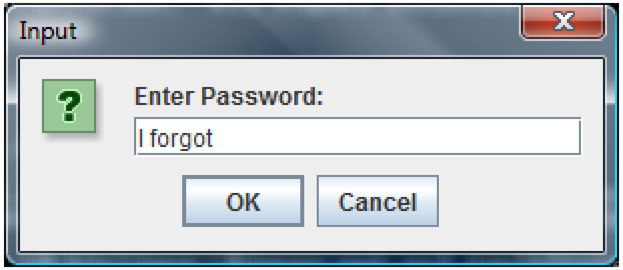
Welcome, Mark

Authentication successful – user is logged on.



Enter User Name:

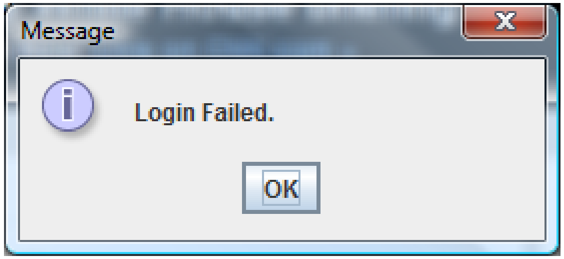
Mark



Enter Password:

I forgot

User enters username and password which are sent to the server. In this case the user enters an incorrect password.

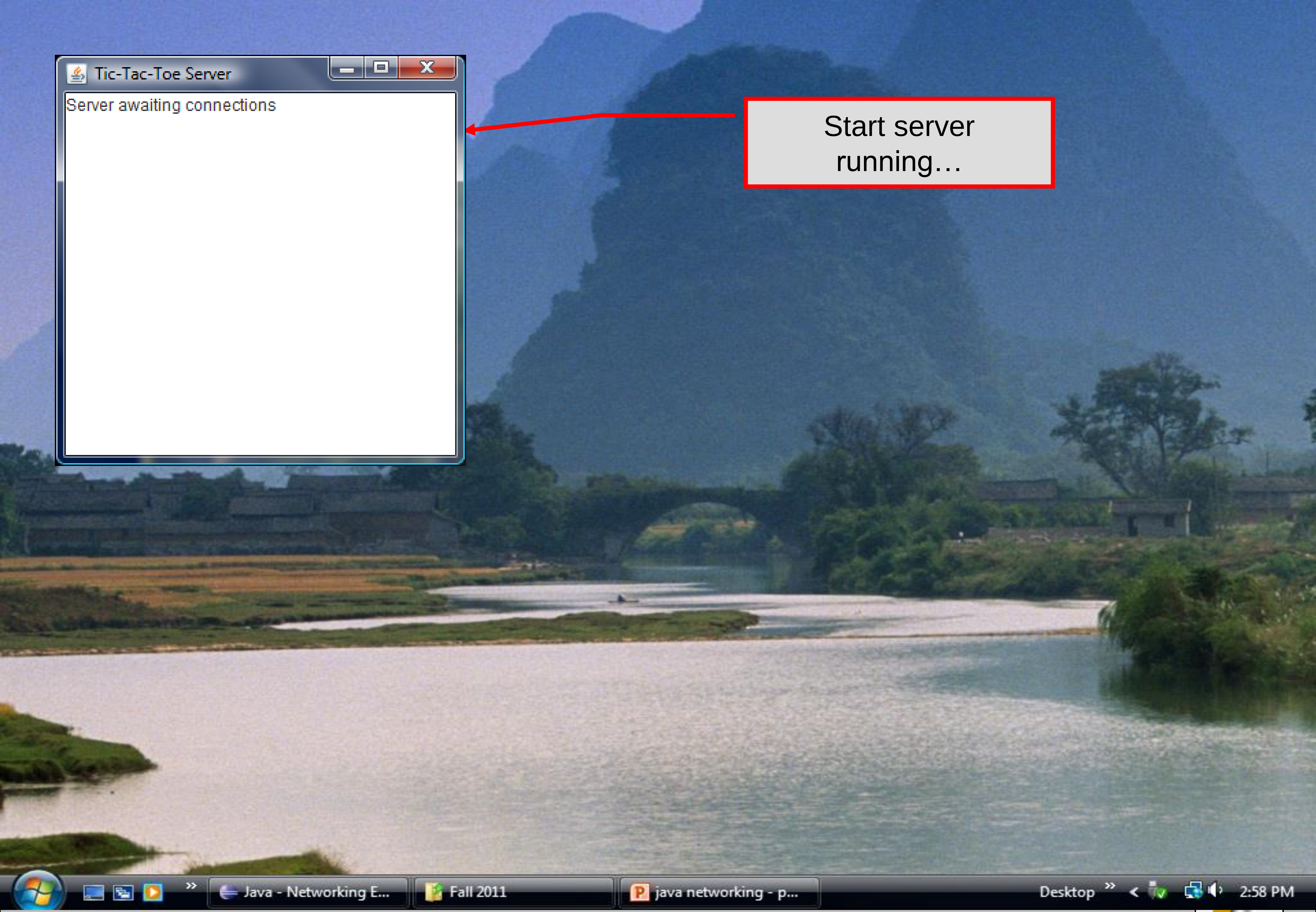


Login Failed.

Authentication not successful – user is not logged on.

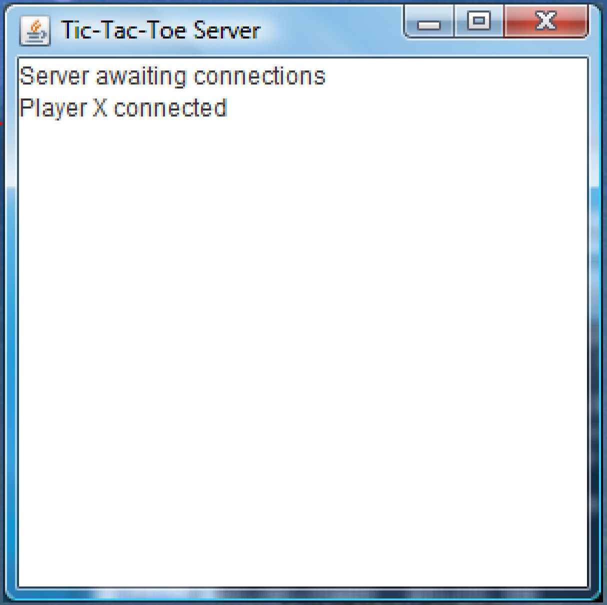
Multithreaded Socket Client/Server Example

- As a culminating example of networking and multi-threading, I’ve put together a rudimentary multi-threaded socket-based TicTacToe client/server application. The code is rather lengthy and there isn’t really anything in it that we haven’t already seen in the earlier sections of the notes. However, I did want you to see a somewhat larger example that utilizes both sockets and threading in Java. The code is on the course web page so try it out.
- This application is a multithreaded server that will allow two client’s to play a game of TicTacToe run on the server.
- To execute, open three command windows, start one server and two clients (in separate windows).
- The following few pages contain screen shots of what you should see when executing this code.



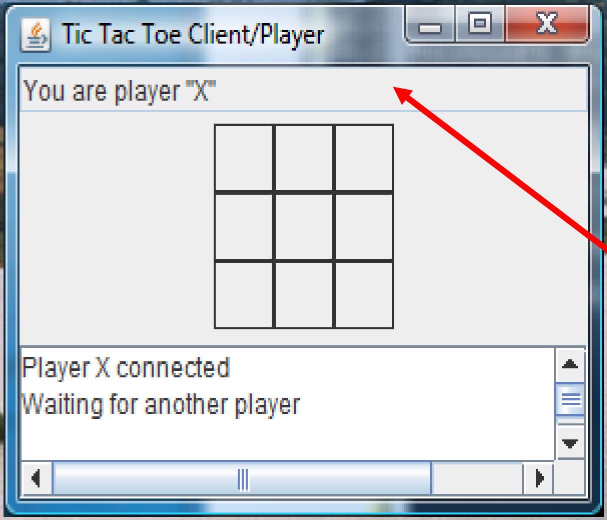
```
Server awaiting connections
```

Start server running…



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

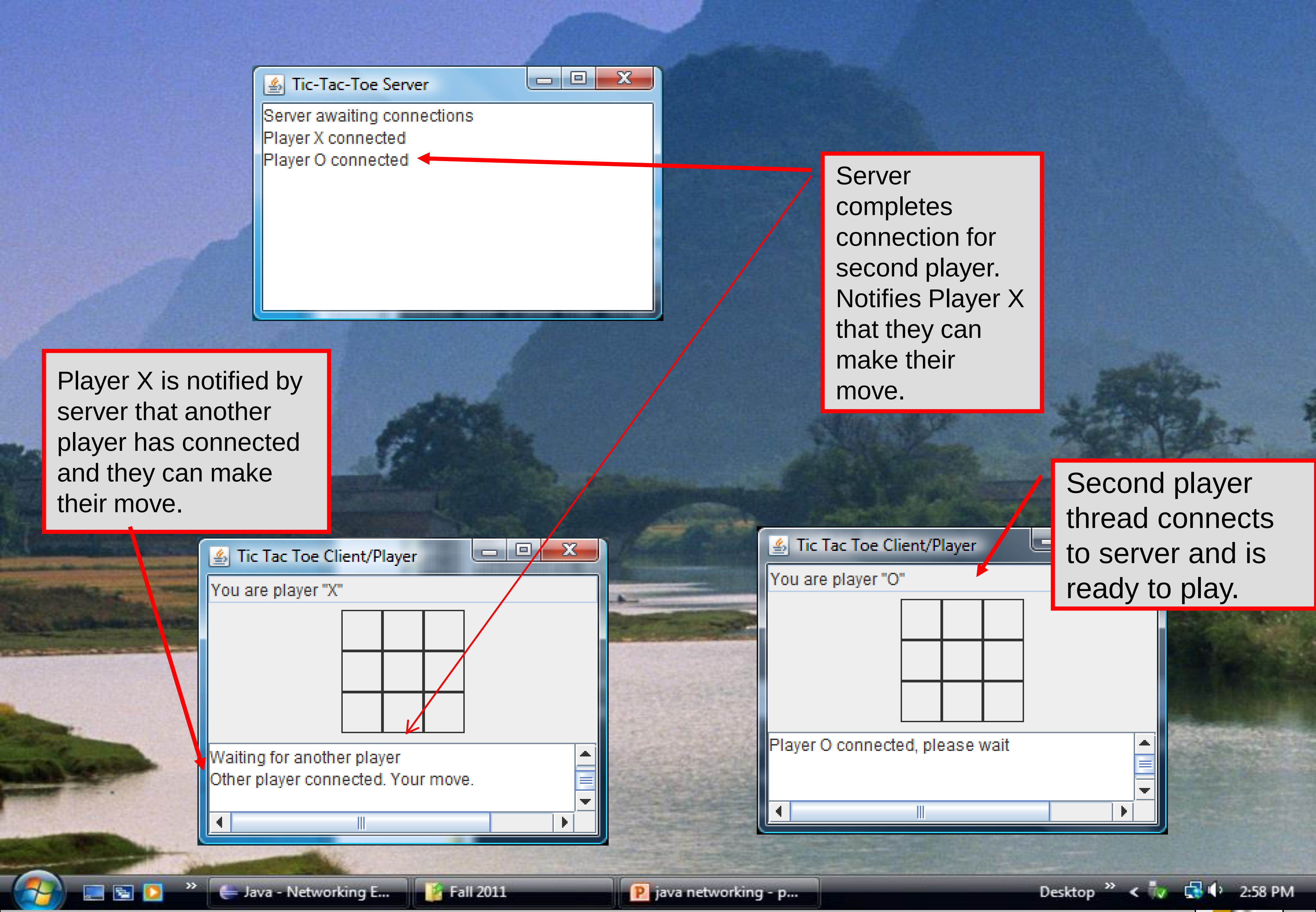
Indicate to first player that server is waiting for another player thread to connect.



```
You are player "X"
```

```
Player X connected
Waiting for another player
```

Start first player thread



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

```
You are player "X"
```

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

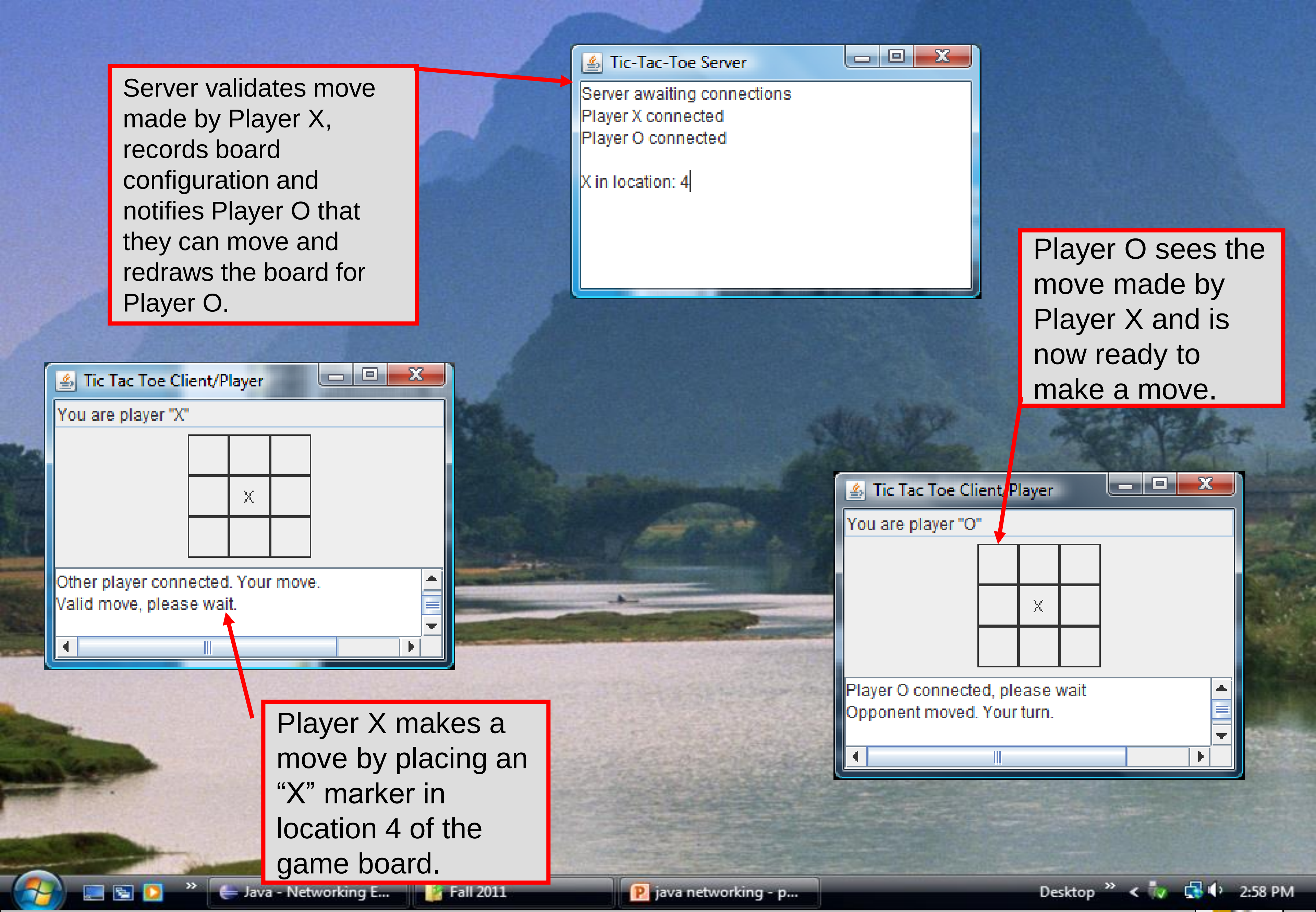
```
You are player "O"
```

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

Server completes connection for second player. Notifies Player X that they can make their move.

Player X is notified by server that another player has connected and they can make their move.

Second player thread connects to server and is ready to play.



Server validates move made by Player X, records board configuration and notifies Player O that they can move and redraws the board for Player O.

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

X in location: 4
```

Player O sees the move made by Player X and is now ready to make a move.

You are player "X"

	X	

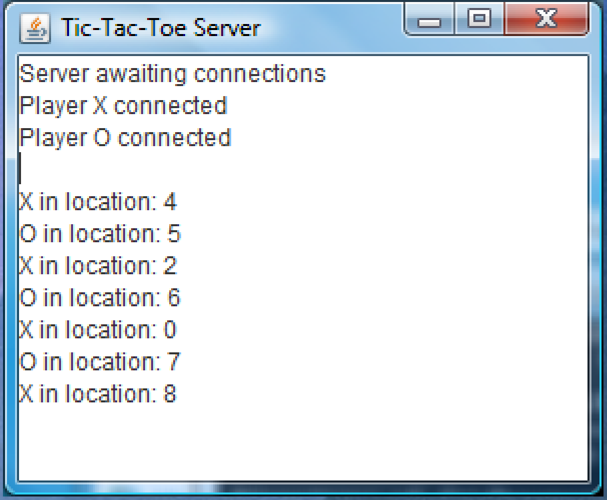
```
Other player connected. Your move.
Valid move, please wait.
```

You are player "O"

	X	

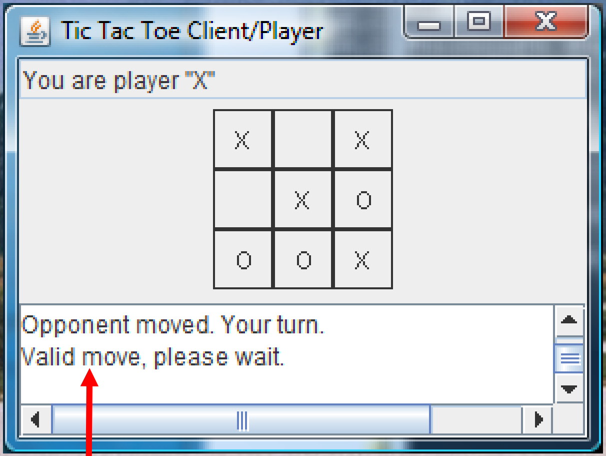
```
Player O connected, please wait
Opponent moved. Your turn.
```

Player X makes a move by placing an “X” marker in location 4 of the game board.



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

X in location: 4
O in location: 5
X in location: 2
O in location: 6
X in location: 0
O in location: 7
X in location: 8
```

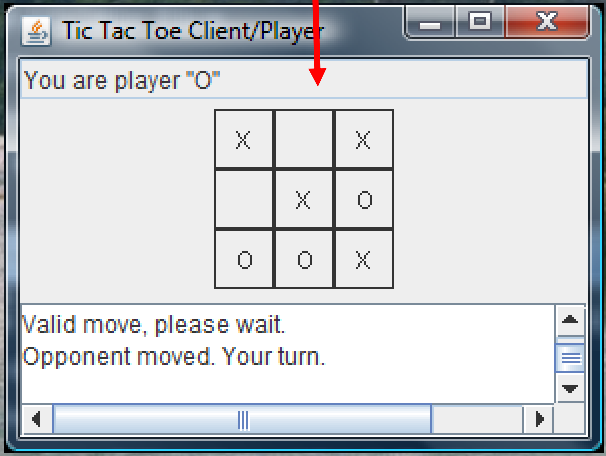
Player O is notified that Player X has made a move and is graphically shown the updated board layout. Server indicates Player O is now able to make their move. No indication is given that the game is technically over.



You are player "X"

X		X
	X	O
O	O	X

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



You are player "O"

X		X
	X	O
O	O	X

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

Although Player X has won the game, this server is too dumb to know this and allows the game to continue