COP 4610L: Applications in the Enterprise Spring 2008

Java Networking and the Internet – Part 3

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4610L/spr2008

School of Electrical Engineering and Computer Science
University of Central Florida

More Details on Establishing a Server Using Stream Sockets

- Step 1 is to create a `ServerSocket` object.
- Invoking a `ServerSocket` constructor such as,

```
ServerSocket server =
    new ServerSocket (portNumber, queueLength);
```

registers an available TCP port number and specifies the number of clients that can wait to connect to the server (i.e., the queue length).

More Details on Establishing a Server Using Stream Sockets (cont.)

- The port number is used by the clients to locate the server application on the server computer. This is often called the handshake point.
- If the queue is full, the server refuses client connections.
- The constructor establishes the port where the server waits for connections from clients – a process known as binding the server to the port.
- Each client will ask to connect to the server on this port. Only one application at a time can be bound to a specific port on the server.

More Details on Establishing a Server Using Stream Sockets (cont.)

- Port numbers can be between 0 and 65,535. Most OS reserve port numbers below 1024 for system services such as email, and Internet servers. Generally, these ports should not be specified as connection ports in user programs. In fact, some OS require special access privileges to bind to port numbers below 1024.
- Programs manage each client connection with a `Socket` object.

More Details on Establishing a Server Using Stream Sockets (cont.)

- In Step 2, the server listens indefinitely (is said to block) for an attempt by a client to connect. To listen for a client connection, the program calls `ServerSocket` method `accept`, as in,

```
Socket connection = server.accept();
```

  which returns a Socket when a connection with a client is established.

- The Socket allows the server to interact with the client.

- The interactions with the client actually occur at a different server port from the handshake port. This allows the port specified in Step 1 to be used again in a multi-threaded server to accept another client connection. We'll see an example of this later in this set of notes.

More Details on Establishing a Server Using Stream Sockets (cont.)

- In Step 3, the `OutputStream` and `InputStream` objects that enable the server to communicate with the client by sending and receiving bytes are established.
- The server sends information to the client via an `OutputStream` and received information from the client via an `InputStream`.
- The server invokes the method `getOutputStream` on the `Socket` to get a reference to the Socket's `OutputStream` and invokes method `getInputStream` on the `Socket` to get a reference to the Socket's `InputStream`.

More Details on Establishing a Server Using Stream Sockets (cont.)

- If primitive types or serializable types (like String) need to be sent rather than bytes, wrapper classes are used to wrap other stream types around the `OutputStream` and `InputStream` objects associated with the `Socket`.

```
ObjectInputStream input =
    new(ObjectInputStream(connection.getInputStream());


ObjectOutputStream output =
    new(ObjectOutputStream(connection.getOutputStream());
```


More Details on Establishing a Server Using Stream Sockets (cont.)

- The beauty of establishing these relationships is that whatever the server writes to the `ObjectOutputStream` is set via the `OutputStream` and is available at the client's `InputStream`, and whatever the client writes to its `OutputStream` (with a corresponding `ObjectOutputStream`) is available via the server's `InputStream`.

- The transmission of the data over the network is seamless and is handled completely by Java.

More Details on Establishing a Server Using Stream Sockets (cont.)

- With Java's multithreading, you can create multithreaded servers that can manage many simultaneous connections with many clients.
- A multithreaded server can take the `Socket` returned by each call to accept and create a new thread that manages network I/O across that `Socket`.
  - Alternatively, a multithreaded sever can maintain a pool of threads (a set of already existing threads) ready to manage network I/O across the new `Sockets` as they are created. In this fashion, when the server receives a connection, it need not incur the overhead of thread creation. When the connection is closed, the thread is returned to the pool for reuse.

More Details on Establishing a Server Using Stream Sockets (cont.)

- Step 4 is the processing phase, in which the server and client communicate via the `OutputStream` and `InputStream` objects.
- In Step 5, when the transmission is complete, the server closes the connection by invoking the `close` method on the streams and on the `Socket`.

More Details on Establishing a Client Using Stream Sockets

- Step 1 is to create a `Socket` object to connect to the server. The Socket constructor established the connection with the server.
- For example, the statement

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

uses the `Socket` constructor with two arguments – the server's address and the port number.

- If the connection attempt is successful, this statement returns a `Socket`.

More Details on Establishing a Client Using Stream Sockets (cont.)

- If the connection attempt fails, an instance of a subclass of IOException, since so many program simply catch IOException.
- An UnknownHostException occurs specifically when the system is unable to resolve the server address specified in the call to the Socket constructor to a corresponding IP address.

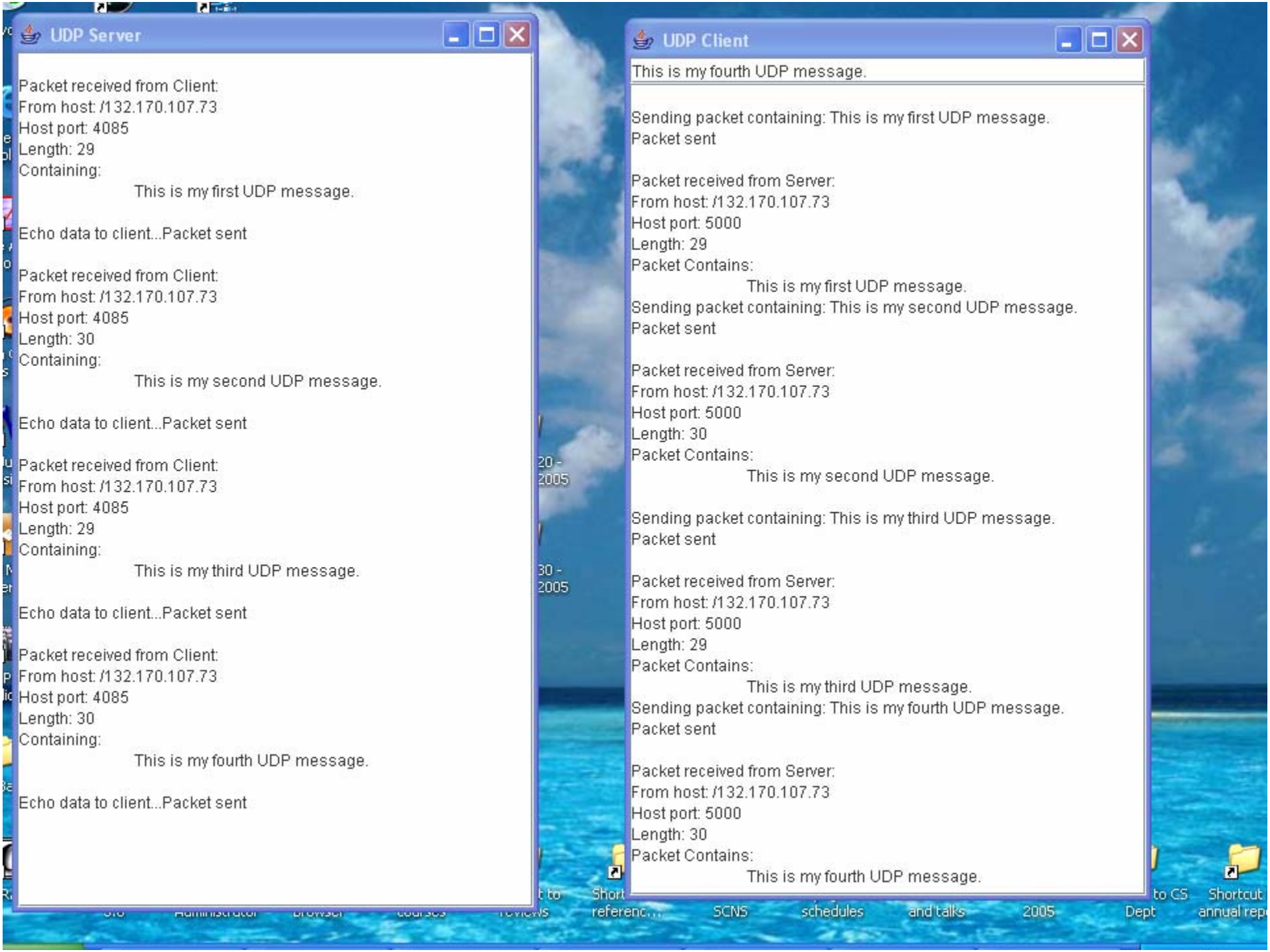
More Details on Establishing a Client Using Stream Sockets (cont.)

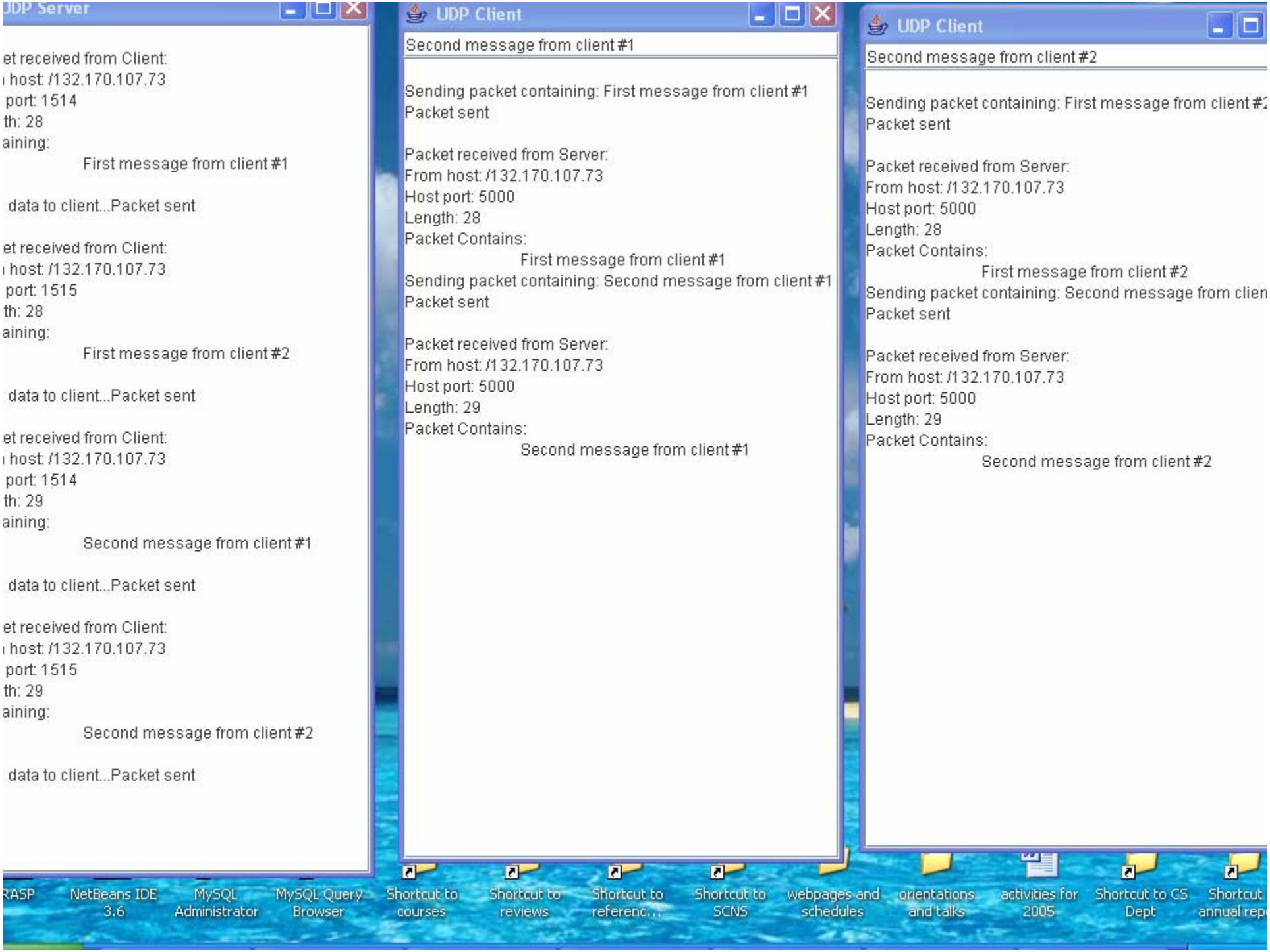
- In Step 2, the client uses Socket methods `getInputStream` and `getOutputStream` to obtain references to the `Socket`'s `InputStream` and `OutputStream`.

- If the server is sending information in the form of actual types (not byte streams) the client should receive the information in the same format. Thus, if the server sends values with an `ObjectOutputStream`, the client should read those values with an `ObjectInputStream`.

More Details on Establishing a Client Using Stream Sockets (cont.)

- Step 3 is the same as in the server, where the client and the server communicate via `InputStream` and `OutputStream` objects.
- In Step 4, the client closes the connection when the transmission is complete by invoking the close method on the streams and on the Socket.
- The client must determine when the server is finished sending information so that it can call `close` to close the Socket connection.
- For example, the `InputStream` method read returns the value -1 when it detects end-of-stream (also called EOF). If an `ObjectInputStream` is used to read information from the server, an `EOFException` occurs when the client attempts to read a value from a stream on which end-of-stream is detected.


UDP Server
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 29
Containing:
This is my first UDP message.
Echo data to client...Packet sent
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 30
Containing:
This is my second UDP message.
Echo data to client...Packet sent
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 29
Containing:
This is my third UDP message.
Echo data to client...Packet sent
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 30
Containing:
This is my fourth UDP message.
Echo data to client...Packet sent
UDP Client
This is my fourth UDP message.
Sending packet containing: This is my first UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 29
Packet Contains:
This is my first UDP message.
Sending packet containing: This is my second UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 30
Packet Contains:
This is my second UDP message.
Sending packet containing: This is my third UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 29
Packet Contains:
This is my third UDP message.
Sending packet containing: This is my fourth UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 30
Packet Contains:
This is my fourth UDP message.


UDP Server
et received from Client:
host: /132.170.107.73
port: 1514
th: 28
aining:
First message from client #1
data to client...Packet sent
et received from Client:
host: /132.170.107.73
port: 1515
th: 28
aining:
First message from client #2
data to client...Packet sent
et received from Client:
host: /132.170.107.73
port: 1514
th: 29
aining:
Second message from client #1
data to client...Packet sent
et received from Client:
host: /132.170.107.73
port: 1515
th: 29
aining:
Second message from client #2
data to client...Packet sent
UDP Client
Second message from client #1
Sending packet containing: First message from client #1
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 28
Packet Contains:
First message from client #1
Sending packet containing: Second message from client #1
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 29
Packet Contains:
Second message from client #1
UDP Client
Second message from client #2
Sending packet containing: First message from client #2
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 28
Packet Contains:
First message from client #2
Sending packet containing: Second message from clien
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 29
Packet Contains:
Second message from client #2
RASP
NetBeans IDE 3.6
MySQL Administrator
MySQL Query Browser
Shortcut to courses
Shortcut to reviews
Shortcut to referenc...
Shortcut to SCNS
webpages and schedules
orientations and talks
activities for 2005
Shortcut to CS Dept
Shortcut annual rep


UDP Server
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 29
Containing:
This is my first UDP message.
Echo data to client...Packet sent
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 30
Containing:
This is my second UDP message.
Echo data to client...Packet sent
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 29
Containing:
This is my third UDP message.
Echo data to client...Packet sent
Packet received from Client:
From host: /132.170.107.73
Host port: 4085
Length: 30
Containing:
This is my fourth UDP message.
Echo data to client...Packet sent


UDP Client
This is my fourth UDP message.
Sending packet containing: This is my first UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 29
Packet Contains:
This is my first UDP message.
Sending packet containing: This is my second UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 30
Packet Contains:
This is my second UDP message.
Sending packet containing: This is my third UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 29
Packet Contains:
This is my third UDP message.
Sending packet containing: This is my fourth UDP message.
Packet sent
Packet received from Server:
From host: /132.170.107.73
Host port: 5000
Length: 30
Packet Contains:
This is my fourth UDP message.

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 two examples illustrate Java's application layer capabilities which remove the responsibility of establishing the network connection from the developer.
- The first 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.)
- The second example uses a `JOptionPane` to perform the connection. (This example is an application that opens and reads a file on a specified web server, hence it acts as a simple web browser.)

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 = "COP 4610L Home Page">
    <param name = "location1" value = "http://www.cs.ucf.edu/courses/cop4610L/spr2008">
    <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
```

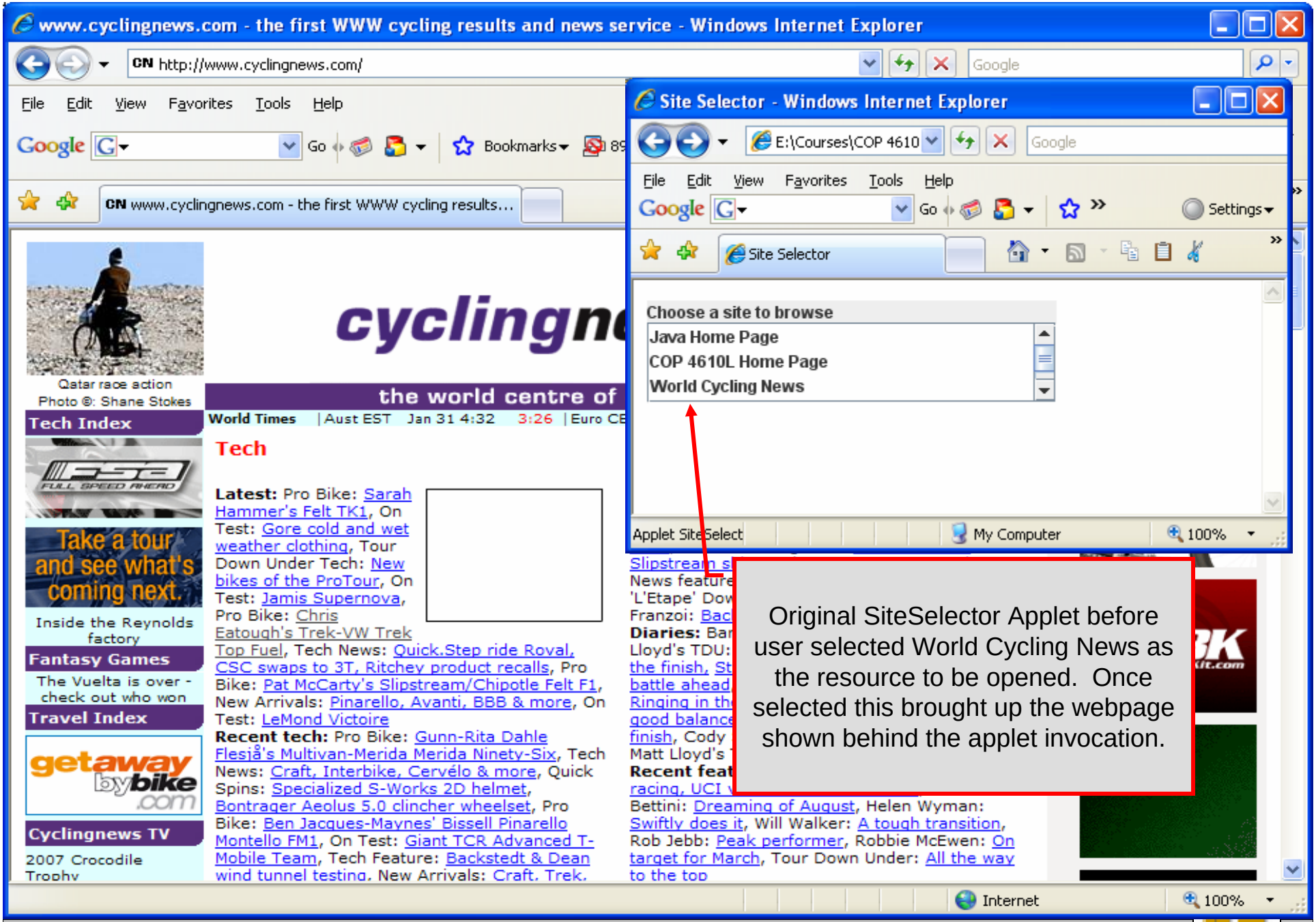

Example 1 – SiteSelector Applet (cont.)

```
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
```

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.

Example 2 – ReadServerFile Application

```
// ReadServerFile.java
// Use a JEditorPane to display the contents of a file on a Web server.
// Application showing high-level Java networking capabilities
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;

public class ReadServerFile extends JFrame
{
   private JTextField enterField; // JTextField to enter site name
   private JEditorPane contentsArea; // to display Web site
   // set up GUI
   public ReadServerFile()
   {
      super( "Simple Web Browser" );
```


Example 2 – ReadServerFile Application (cont.)

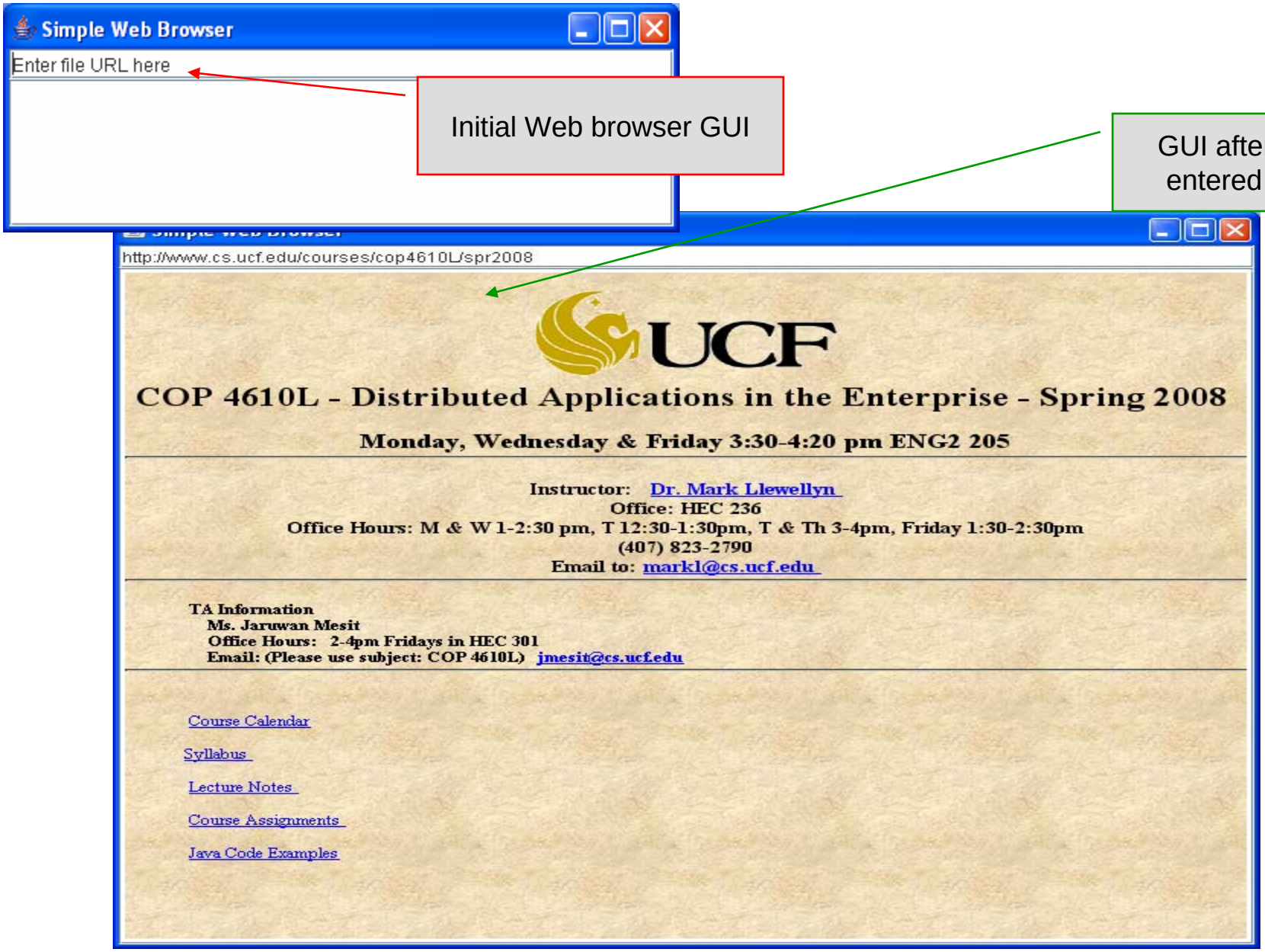
```
// create enterField and register its listener
enterField = new JTextField( "Enter file URL here" );
enterField.addActionListener(
   new ActionListener()
   {
      // get document specified by user
      public void actionPerformed( ActionEvent event )
      {
         getThePage( event.getActionCommand() );
      } // end method actionPerformed
   } // end inner class
); // end call to addActionListener
add( enterField, BorderLayout.NORTH );
contentsArea = new JEditorPane(); // create contentsArea
contentsArea.setEditable( false );
contentsArea.addHyperlinkListener(
   new HyperlinkListener()
   {
      // if user clicked hyperlink, go to specified page
      public void hyperlinkUpdate( HyperlinkEvent event )
      {
         if ( event.getEventType() ==  HyperlinkEvent.EventType.ACTIVATED )
            getThePage( event.getURL().toString() );
      } // end method hyperlinkUpdate
   } // end inner class
); // end call to addHyperlinkListener
```


Example 2 – ReadServerFile Application (cont.)

```
    add( new JScrollPane( contentsArea ), BorderLayout.CENTER );
    setSize( 400, 300 ); // set size of window
    setVisible( true ); // show window
  } // end ReadServerFile constructor

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

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

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

Driver class to execute ReadServerFile application


Simple Web Browser
Enter file URL here
Initial Web browser GUI
GUI after user entered URL
http://www.cs.ucf.edu/courses/cop4610L/spr2008
UCF
COP 4610L - Distributed Applications in the Enterprise - Spring 2008
Monday, Wednesday & Friday 3:30-4:20 pm ENG2 205
Instructor: Dr. Mark Llewellyn
Office: HEC 236
Office Hours: M & W 1-2:30 pm, T 12:30-1:30pm, T & Th 3-4pm, Friday 1:30-2:30pm
(407) 823-2790
Email to: markl@cs.ucf.edu
TA Information
Ms. Jaruwan Mesit
Office Hours: 2-4pm Fridays in HEC 301
Email: (Please use subject: COP 4610L) jmesit@cs.ucf.edu
Course Calendar
Syllabus
Lecture Notes
Course Assignments
Java Code Examples

Secure Sockets Layer (SSL)

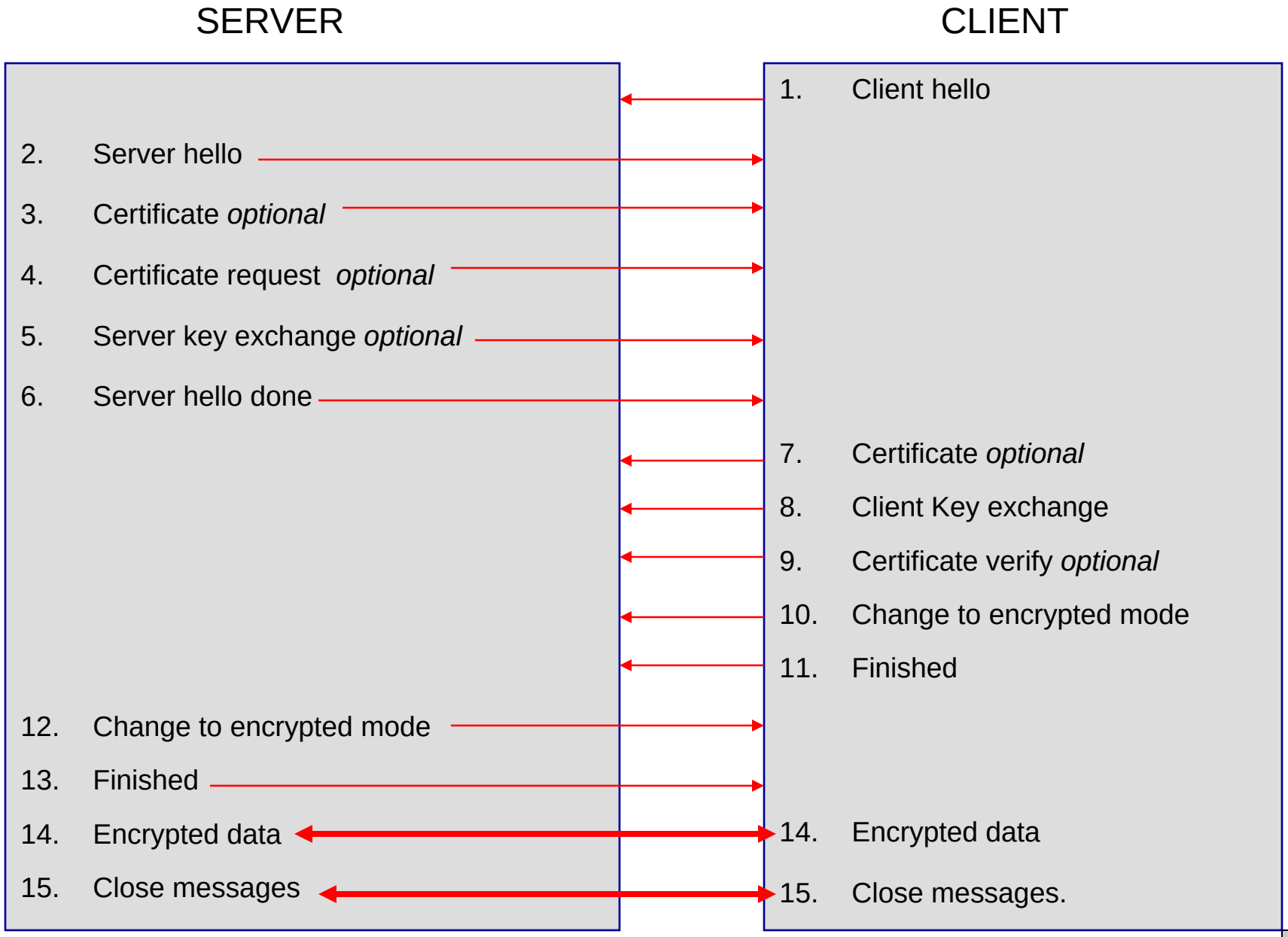
- Most e-business uses SSL for secure on-line transactions.
- SSL does not explicitly secure transactions, but rather secures connections.
- SSL implements public-key technology using the RSA algorithm (developed in 1977 at MIT by Ron Rivest, Adi Shamir, and Leonard Adleman) and digital certificates 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
CLIENT
1. Client hello
2. Server hello
3. Certificate optional
4. Certificate request optional
5. Server key exchange optional
6. Server hello done
7. Certificate optional
8. Client Key exchange
9. Certificate verify optional
10. Change to encrypted mode
11. Finished
12. Change to encrypted mode
13. Finished
14. Encrypted data
14. Encrypted data
15. Close messages
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 45 for execution details. This application will not execute correctly unless you follow the steps beginning on page 45.

```
// 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 = "COP 4610L";
   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

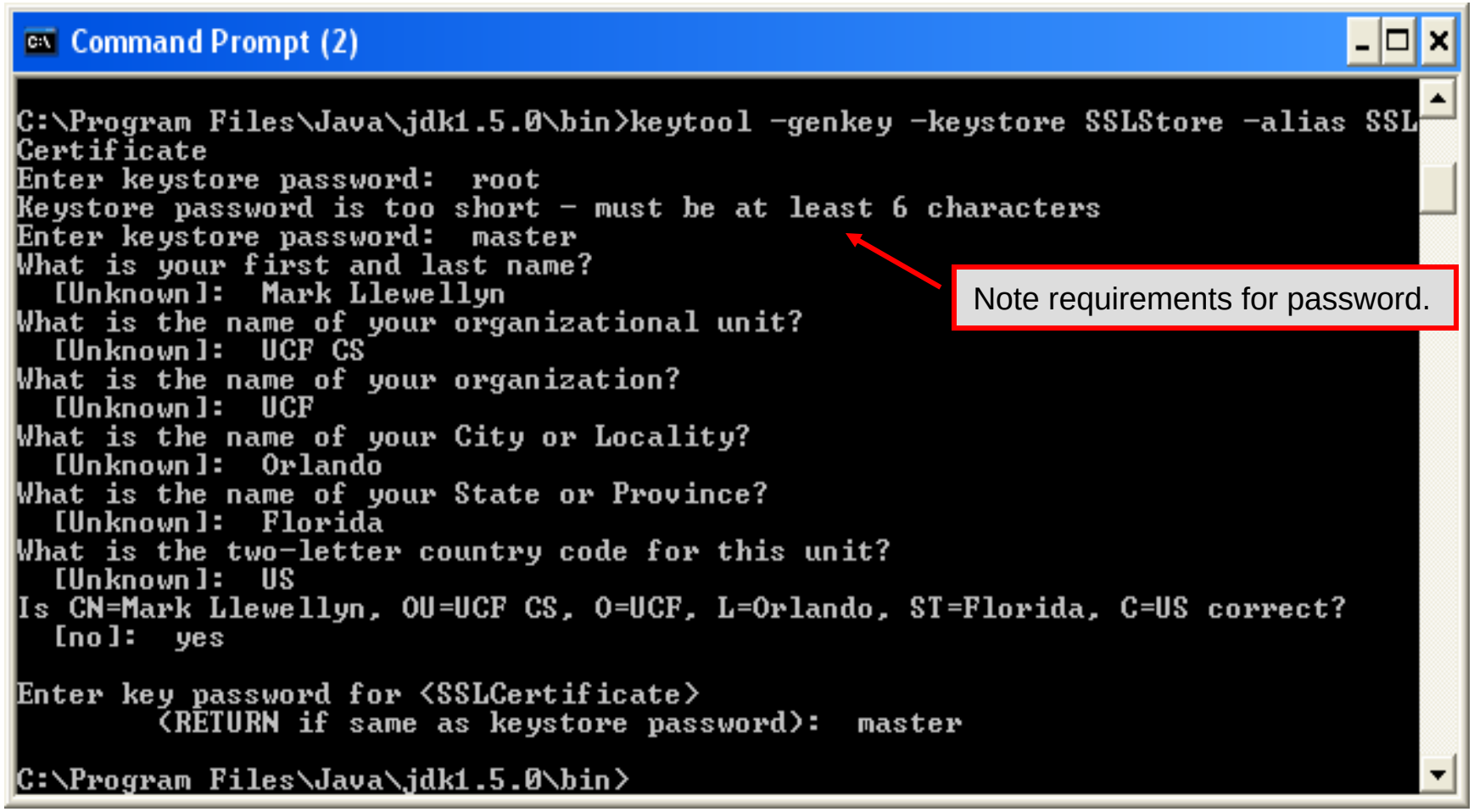
```
      // prompt user for password
      String password = JOptionPane.showInputDialog( null, "Enter Password:" );
      // send password to server
      output.println( password );
      output.flush();
       // create BufferedReader for reading server response
      BufferedReader input = new BufferedReader(
         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



Creating Keystore and Certificate

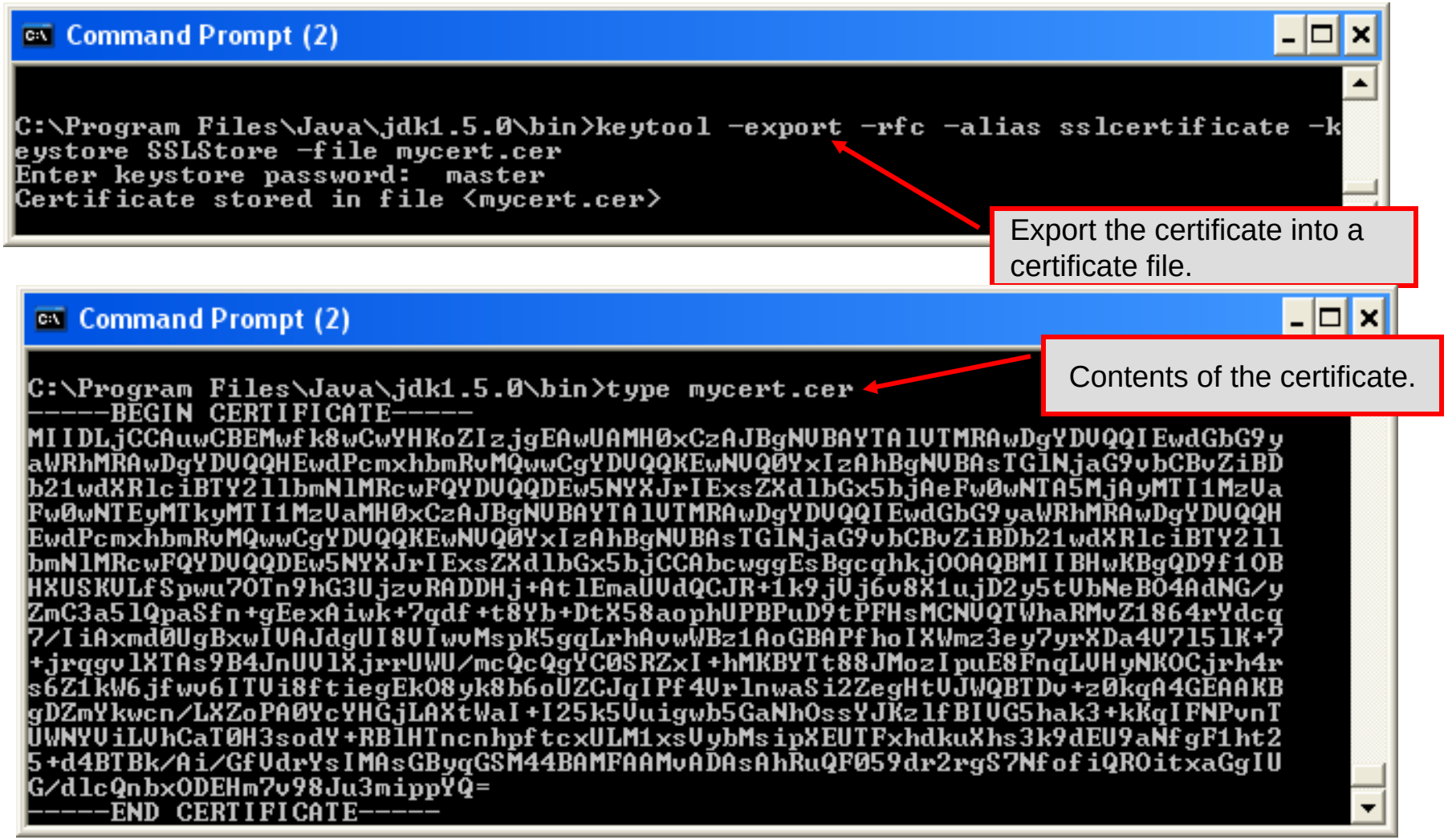
Viewing the keystore contents after its creation.

Command Prompt (2)

```
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>_
```

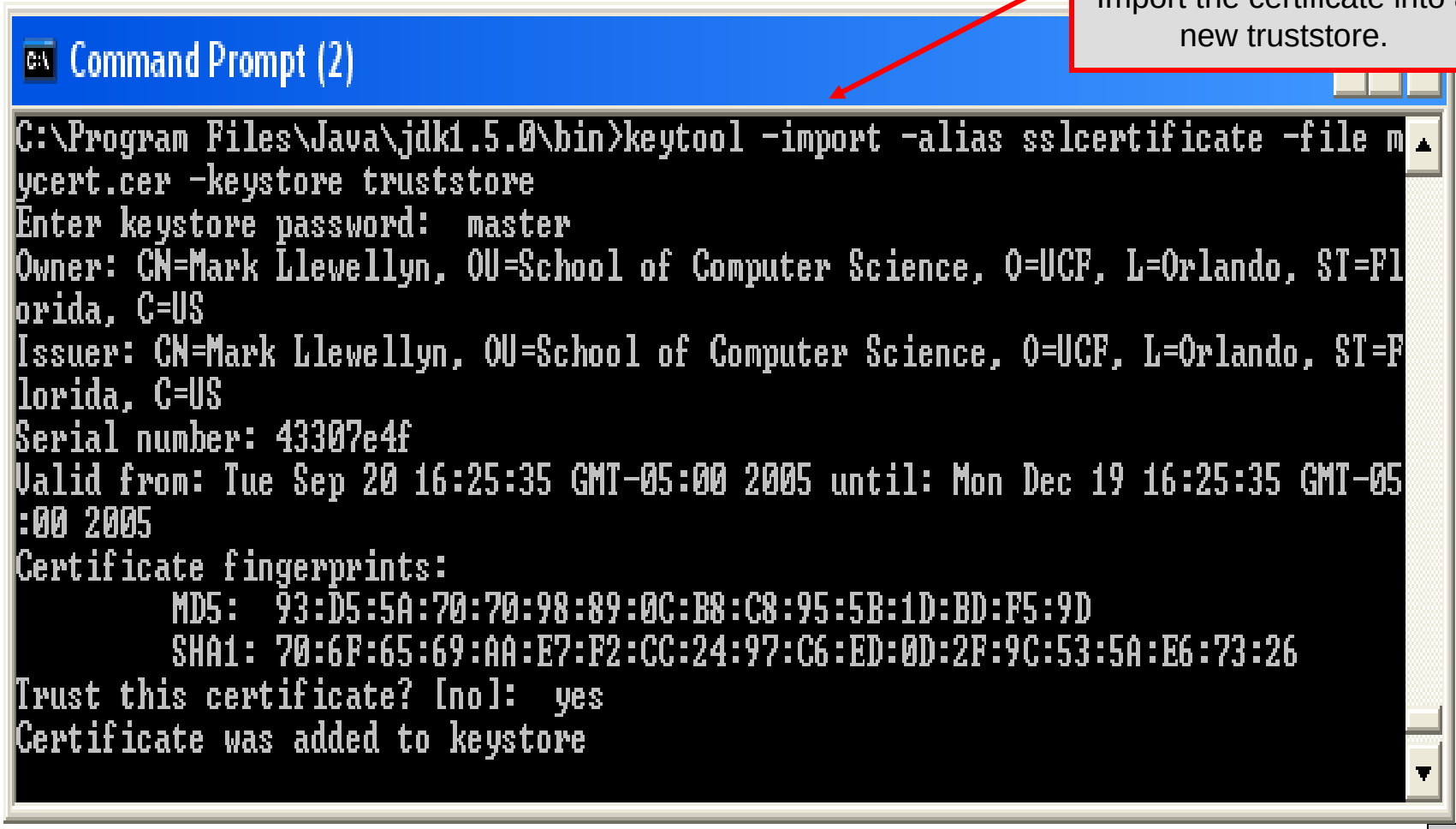
Notice the entry type is keyEntry which means that this entry has a private key associated with it.

Creating Keystore and Certificate



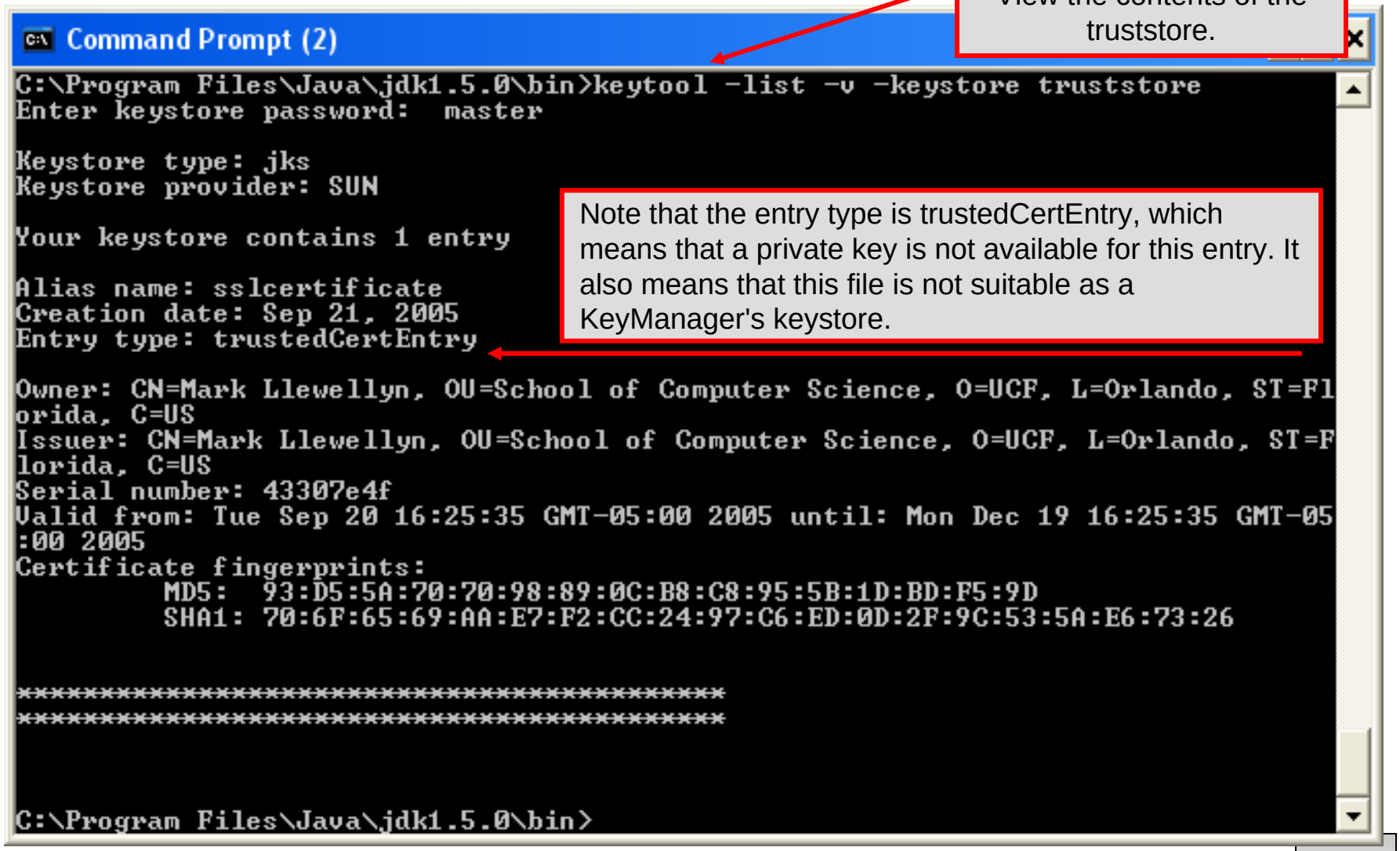
Creating Keystore and Certificate

Import the certificate into a new truststore.



Creating Keystore and Certificate

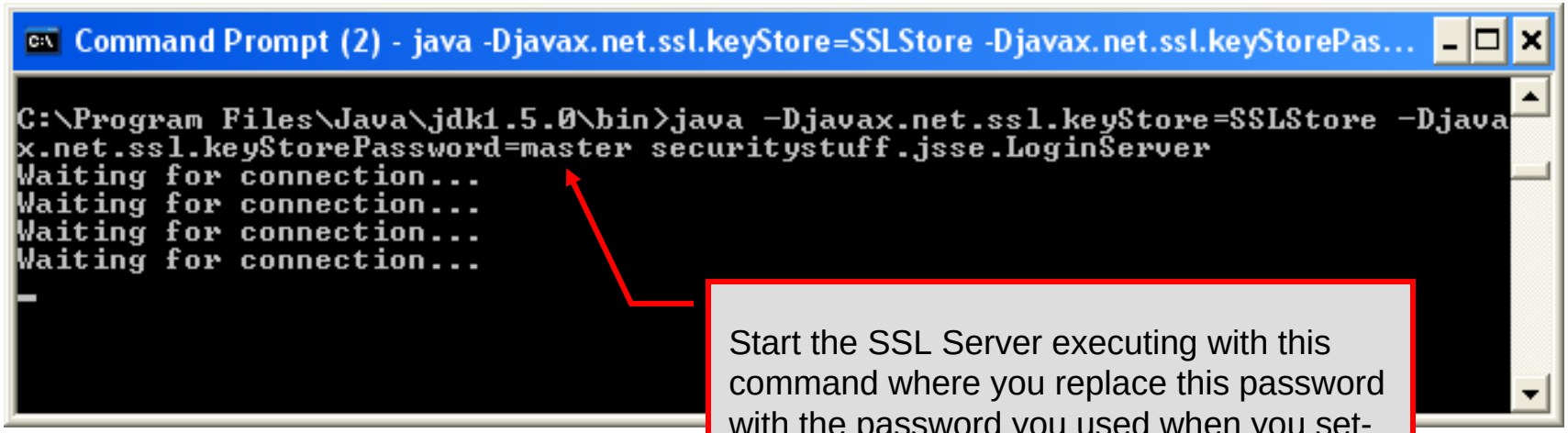
View the contents of the truststore.



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.

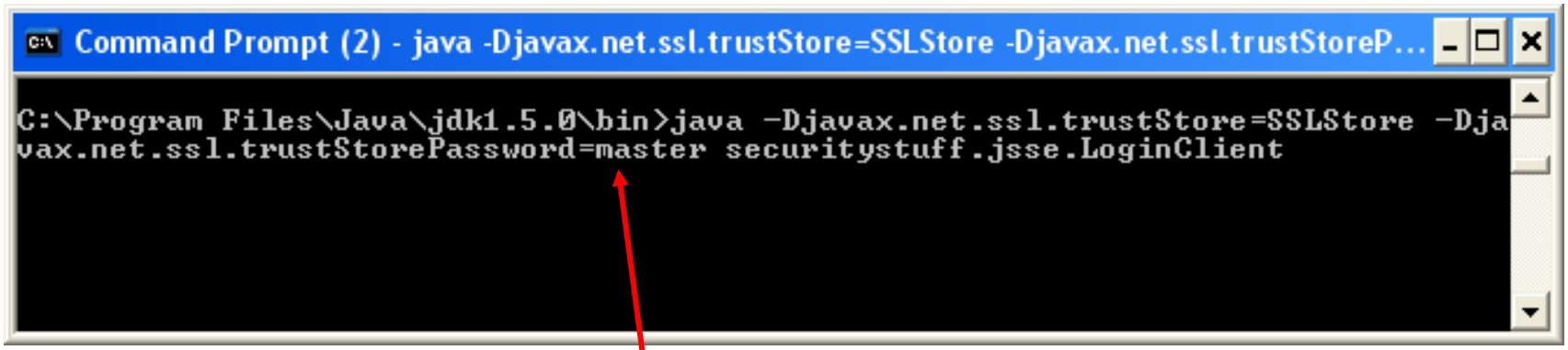
Launching the Secure Server

- 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.

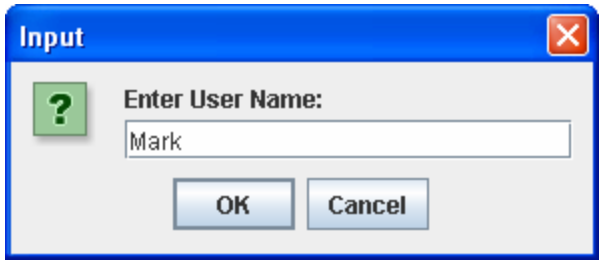


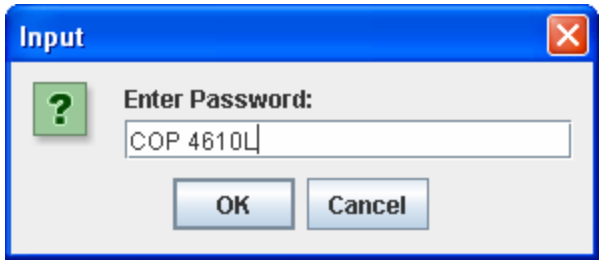
Launching the SSL Client

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



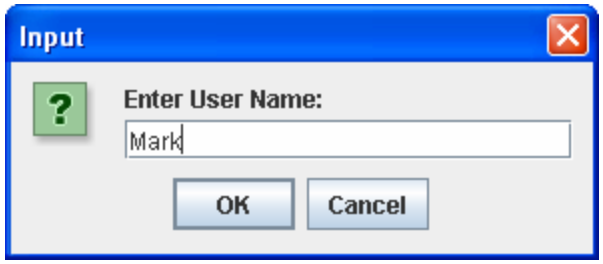
Start the SSL Client application executing with this command where you replace this password with the password you used when you set-up the keystore. Since we are using the same keystore for the server and the client…these will be the same.

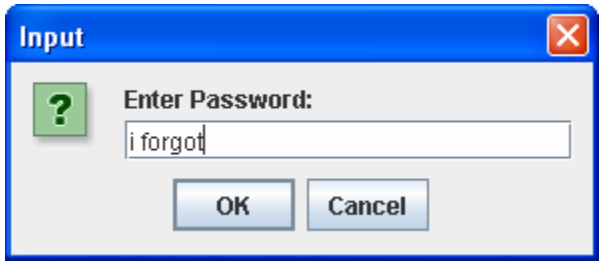


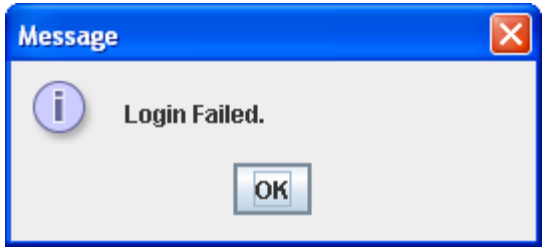


User enters username and password which are sent to the server.

Authentication successful – user is logged on.





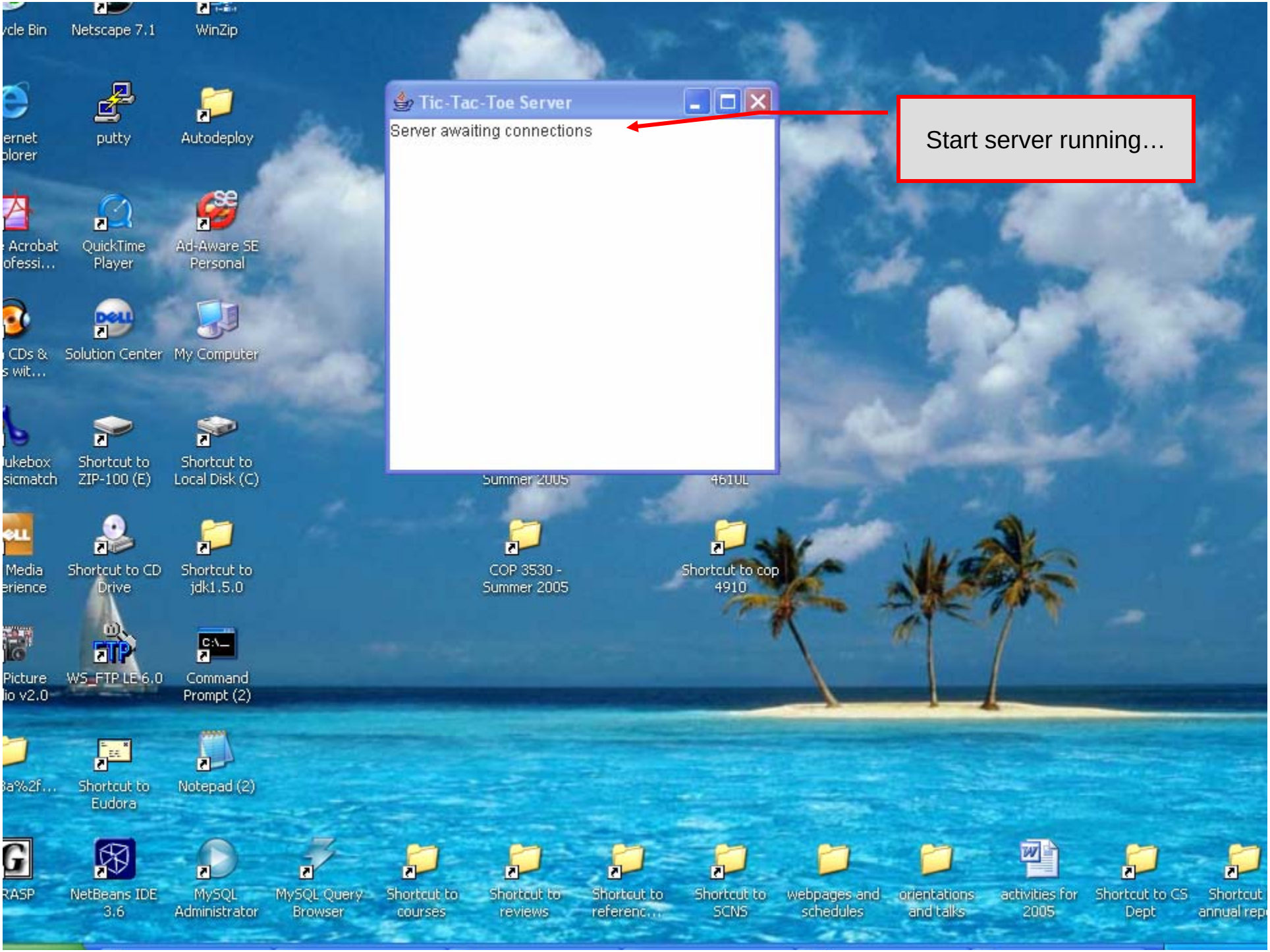


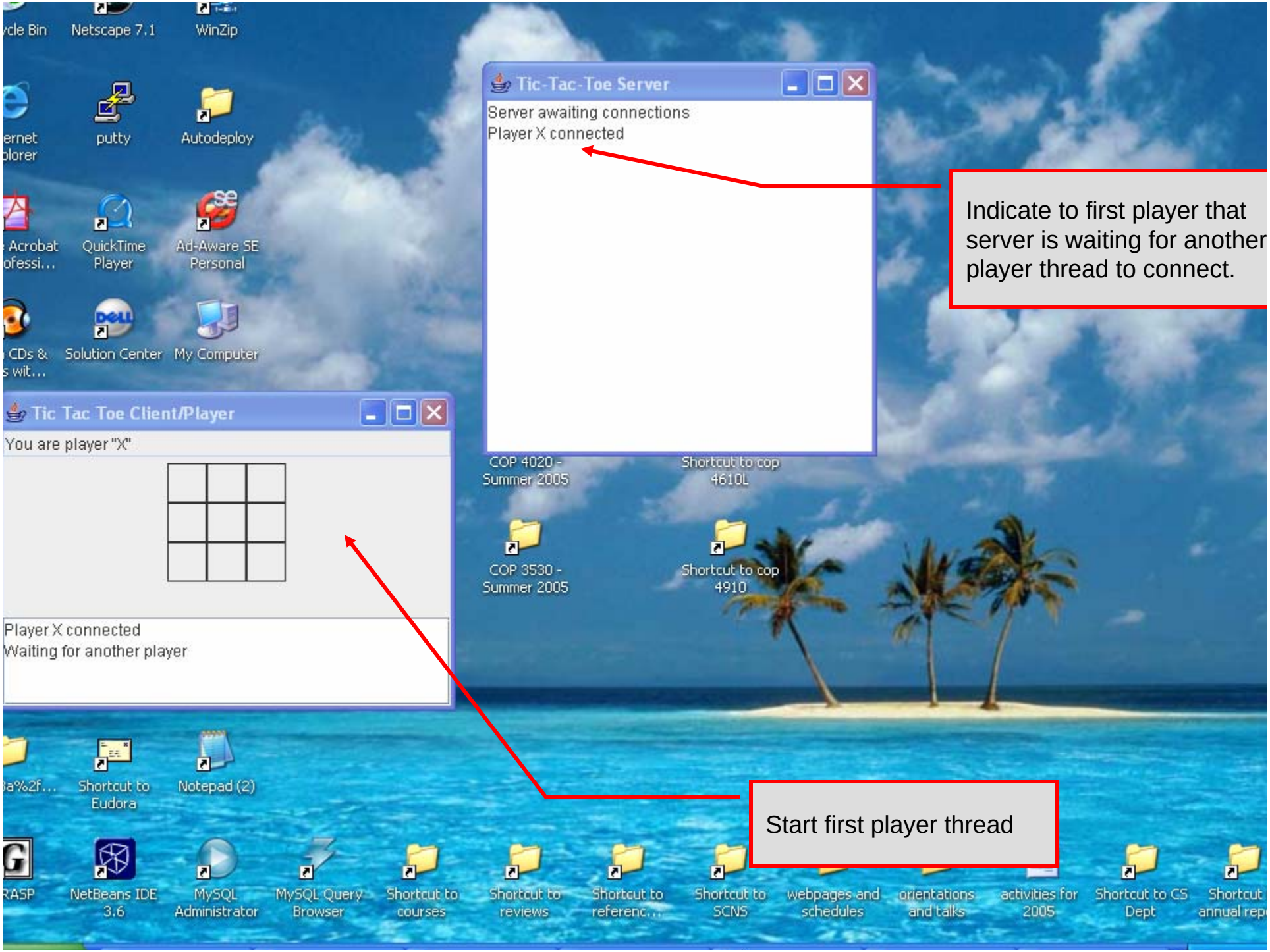
User enters username and password which are sent to the server. In this case the user enters an incorrect password.

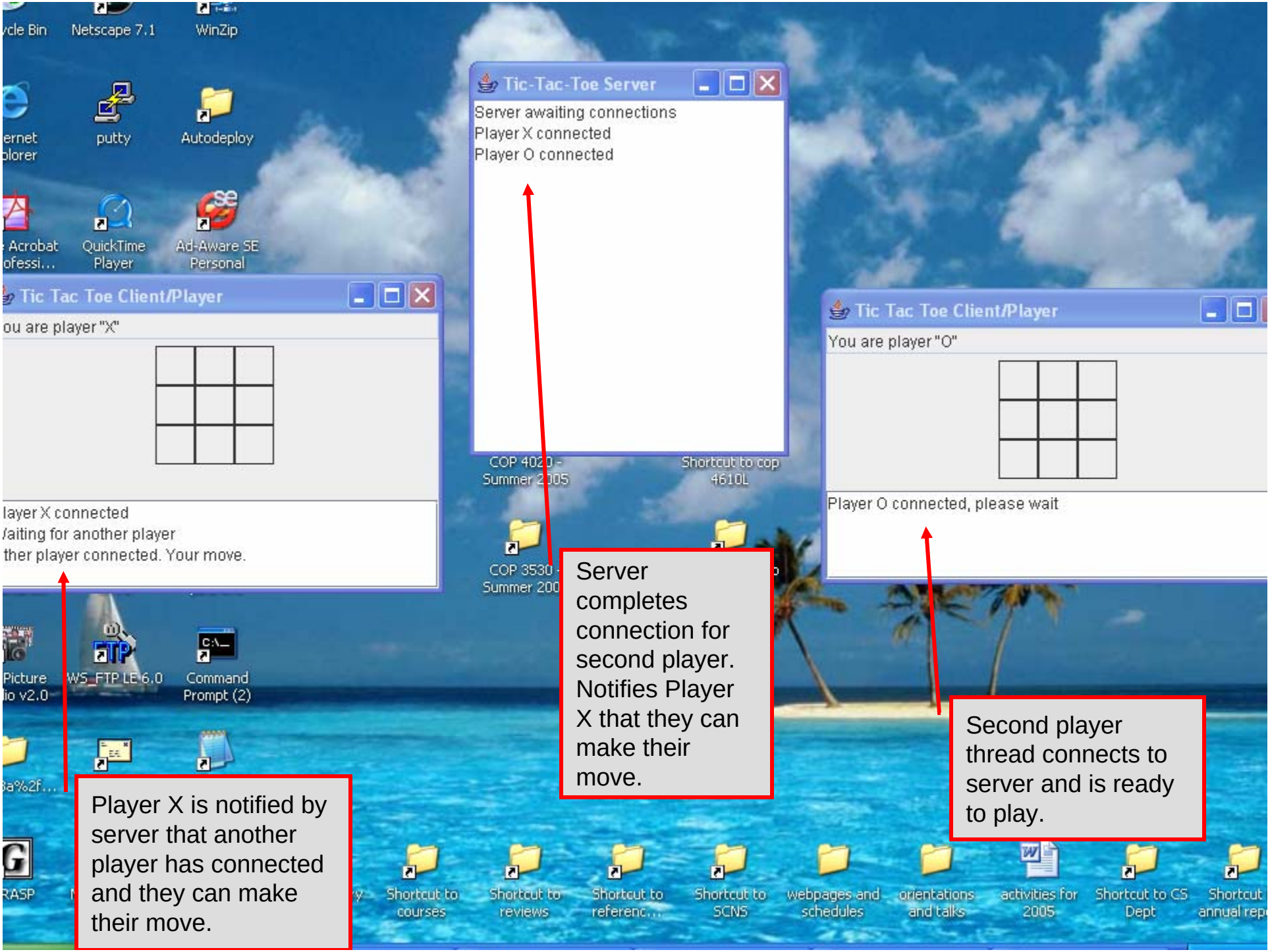
Authentication not successful – user is not logged on.

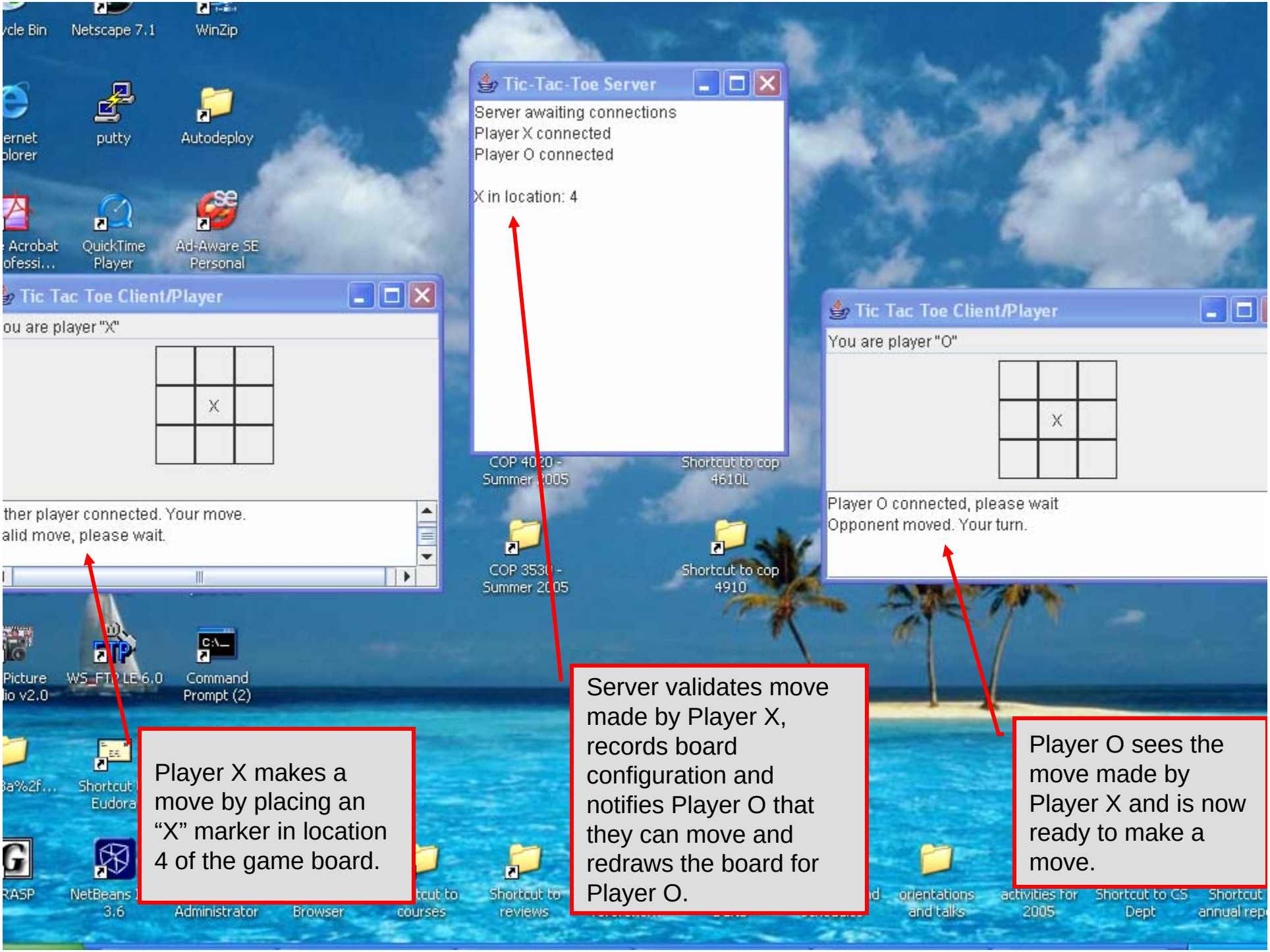
Multithreaded Socket Client/Server Example

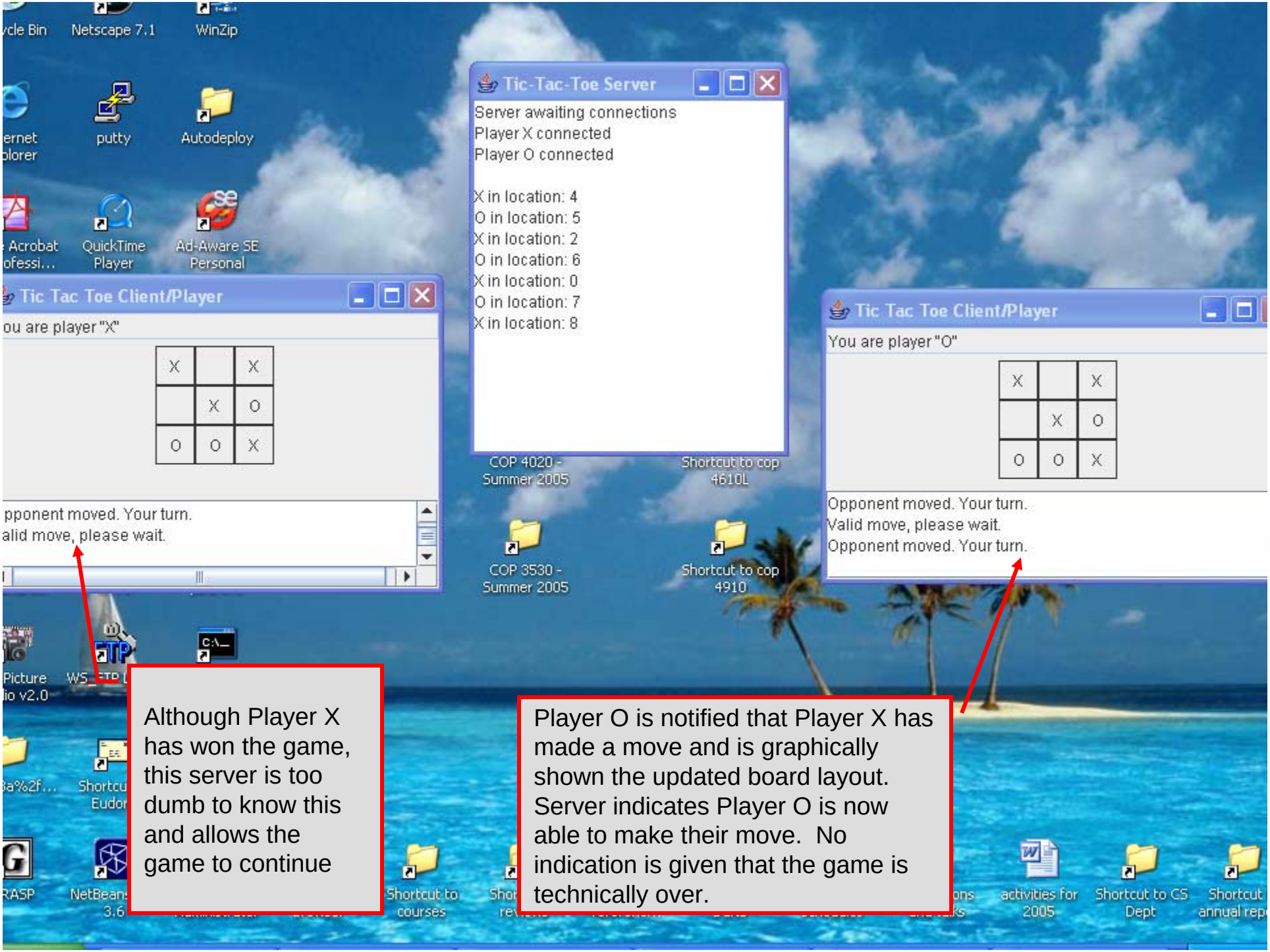
- As a culminating example of networking and multi-threading, I've put together a rudimentary multi-threaded socket-based TicTacToe client/server application. The code is rather lengthy and there isn't really anything in it that we haven't already seen in the earlier sections of the notes. However, I 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.


Tic-Tac-Toe Server
Server awaiting connections
Start server running…
Netscape 7.1
WinZip
putty
Autodeploy
QuickTime Player
Ad-Aware SE Personal
Solution Center
My Computer
Shortcut to ZIP-100 (E)
Shortcut to Local Disk (C)
Shortcut to CD Drive
Shortcut to jdk1.5.0
COP 3530 - Summer 2005
Shortcut to cop 4910
WS_FTP LE 6.0
Command Prompt (2)
Shortcut to Eudora
Notepad (2)
NetBeans IDE 3.6
MySQL Administrator
MySQL Query Browser
Shortcut to courses
Shortcut to reviews
Shortcut to SCNS
webpages and schedules
orientations and talks
activities for 2005
Shortcut to CS Dept


Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Indicate to first player that server is waiting for another player thread to connect.
Tic Tac Toe Client/Player
You are player "X"
Player X connected
Waiting for another player
Start first player thread
Netscape 7.1
WinZip
putty
Autodeploy
QuickTime Player
Ad-Aware SE Personal
Solution Center
My Computer
COP 4020 - Summer 2005
Shortcut to cop 4610L
COP 3530 - Summer 2005
Shortcut to cop 4910
Shortcut to Eudora
Notepad (2)
NetBeans IDE 3.6
MySQL Administrator
MySQL Query Browser
Shortcut to courses
Shortcut to reviews
Shortcut to SCNS
webpages and schedules
orientations and talks
activities for 2005
Shortcut to CS Dept


Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Player O connected
Tic Tac Toe Client/Player
ou are player "X"
layer X connected
/aiting for another player
ther player connected. Your move.
Tic Tac Toe Client/Player
You are player "O"
Player O connected, please wait
Server completes connection for second player. Notifies Player X that they can make their move.
Second player thread connects to server and is ready to play.
Player X is notified by server that another player has connected and they can make their move.


Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Player O connected
X in location: 4
Tic Tac Toe Client/Player
ou are player "X"
X
ther player connected. Your move.
alid move, please wait.
Tic Tac Toe Client/Player
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 validates move made by Player X, records board configuration and notifies Player O that they can move and redraws the board for Player O.
Player O sees the move made by Player X and is now ready to make a move.


Tic-Tac-Toe Server
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
Tic Tac Toe Client/Player
ou are player "X"
X | | X
| X | O
O | O | X
pponent moved. Your turn.
alid move, please wait.
Tic Tac Toe Client/Player
You are player "O"
X | | X
| X | O
O | O | X
Opponent moved. Your turn.
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
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.