

SSL (Secure Socket Layer) and JSSE (Java Secure Socket Extension)

61FIT3NPR -Network Programming

Faculty of Information Technology
Hanoi University
Fall 2020



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/spr2012">
    <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

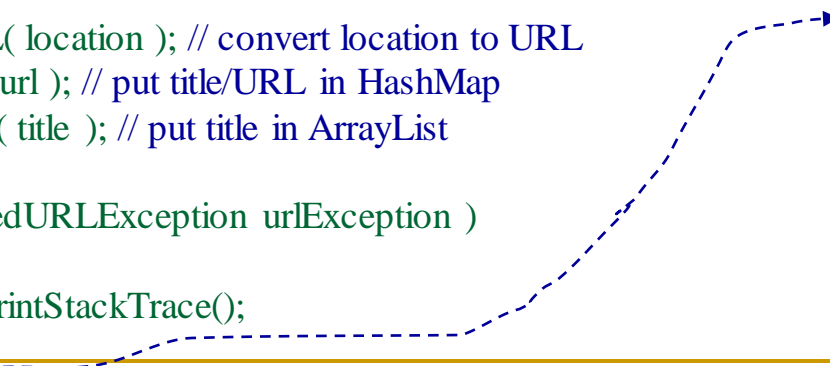
    // read HTML parameters and set up GUI
```

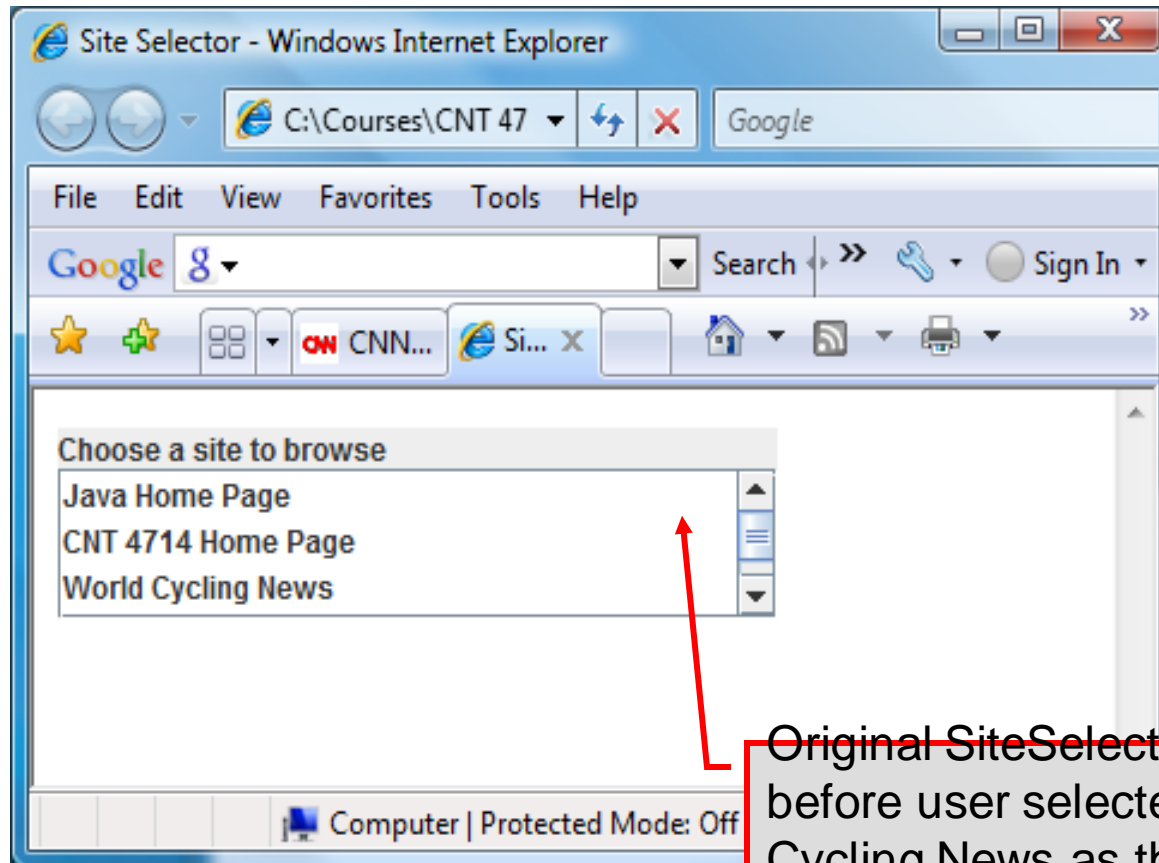
Example 1 – SiteSelector Applet (cont.)

```
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
}
```

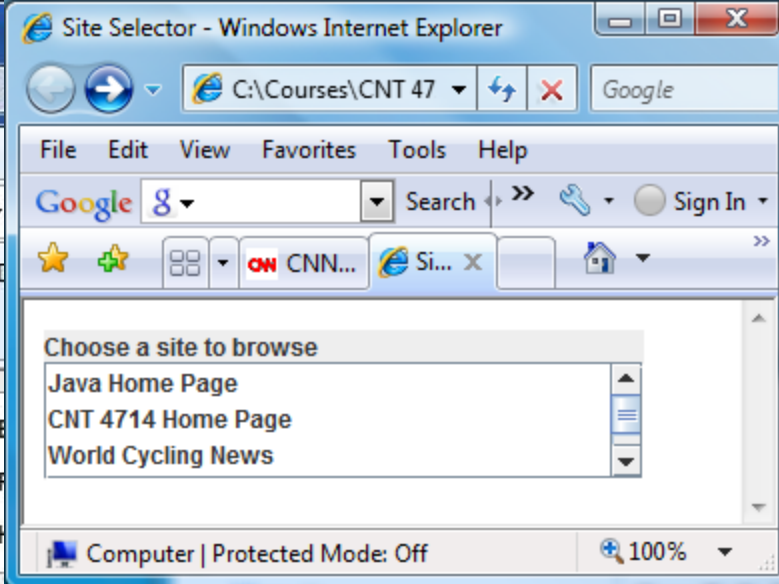
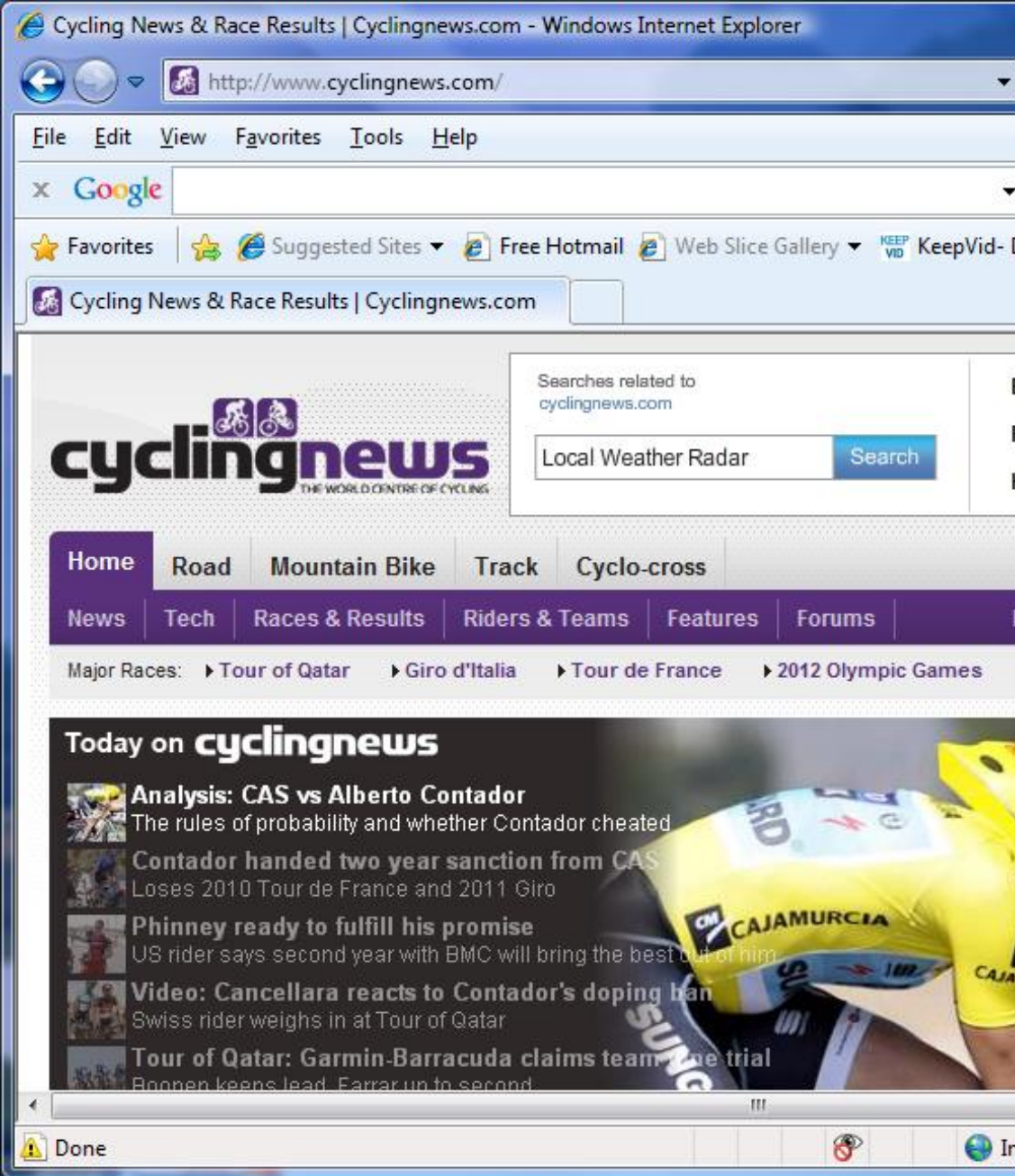
Example 1 – SiteSelector Applet (cont.)

```
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.



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.

What is SSL?

SSL (Secure Socket Layer)



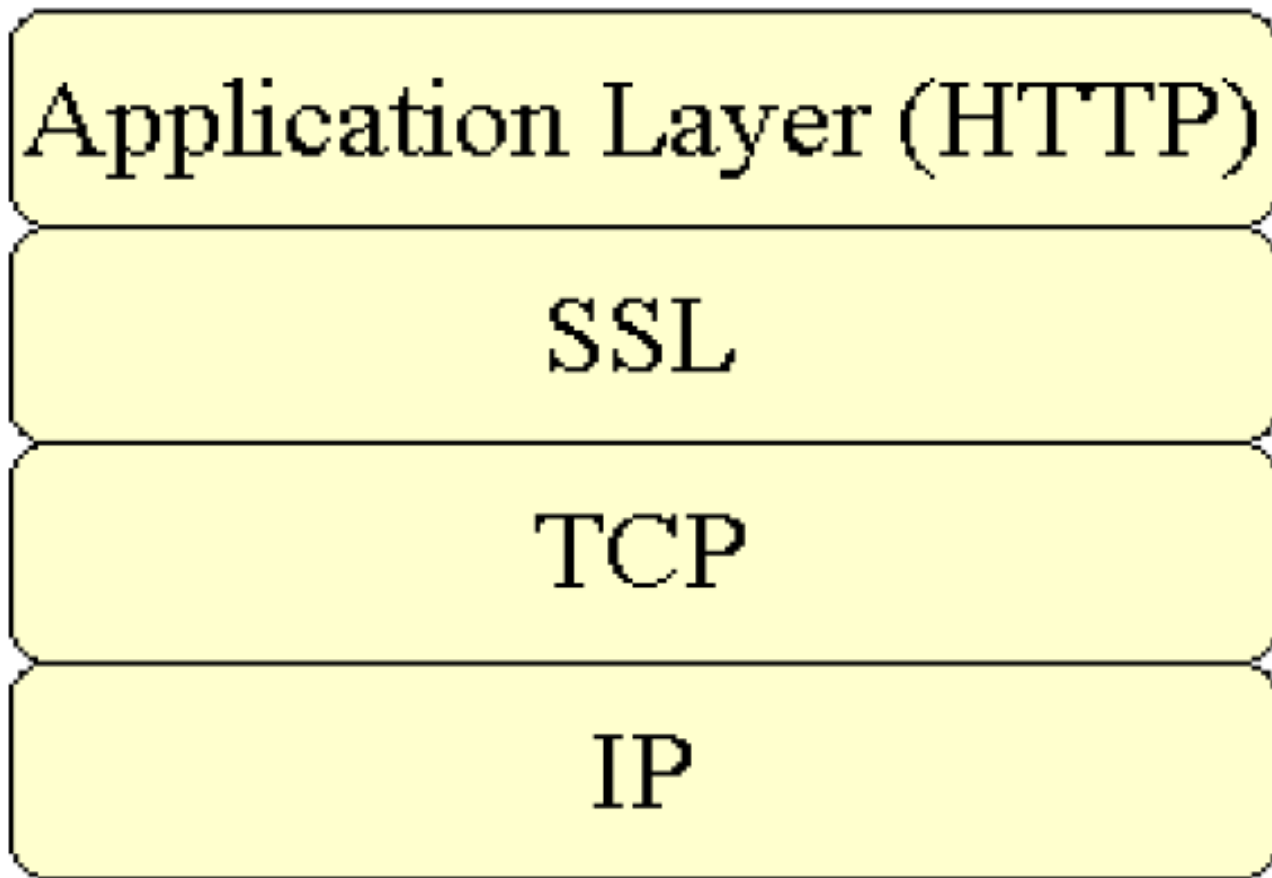
Secure Socket Layer (SSL)

- By far, the dominant security technology on the web
 - Transport layer security
 - HTTPS is HTTP over SSL
 - Responsible for the emergence of e-commerce, other security sensitive services on the web
 - Beneficiary of several years of public scrutiny
-

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.

SSL runs over TCP



Why SSL? SSL Provides ...

- Confidentiality (Privacy)
 - Data integrity (Tamper-proofing)
 - Server authentication (Proving a server is what it claims it is)
 - Used in typical B2C transaction
 - Optional client authentication
 - Would be required in B2B (or Web services environment in which program talks to program)
-

How SSL Works

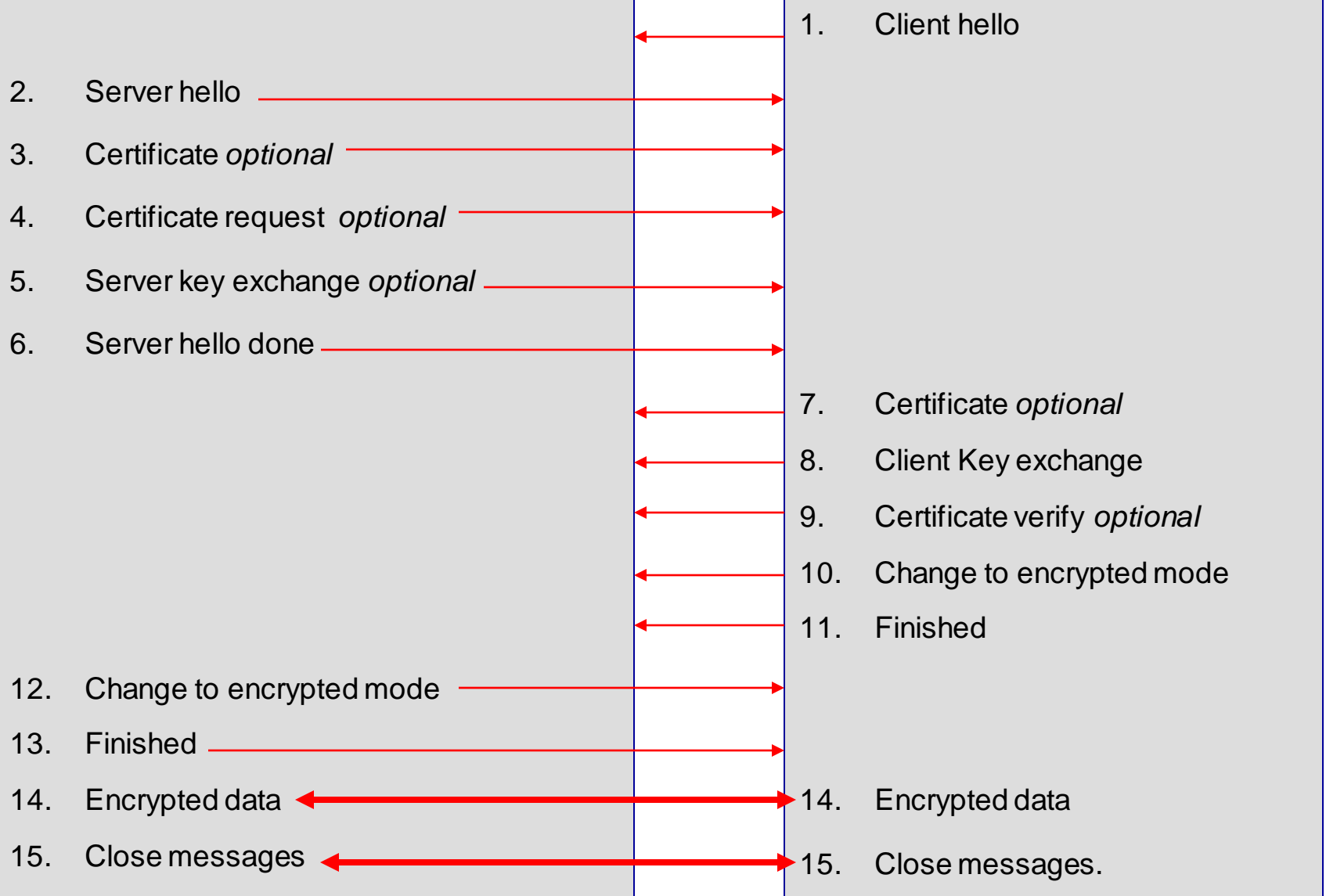
- Initially, a client sends a message to a server.
- The server responds and sends its digital certificate to the client for authentication.
- Using public-key cryptography to communicate securely, the client and server negotiate **session keys** to continue the transaction.
- Once the session keys are 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



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.

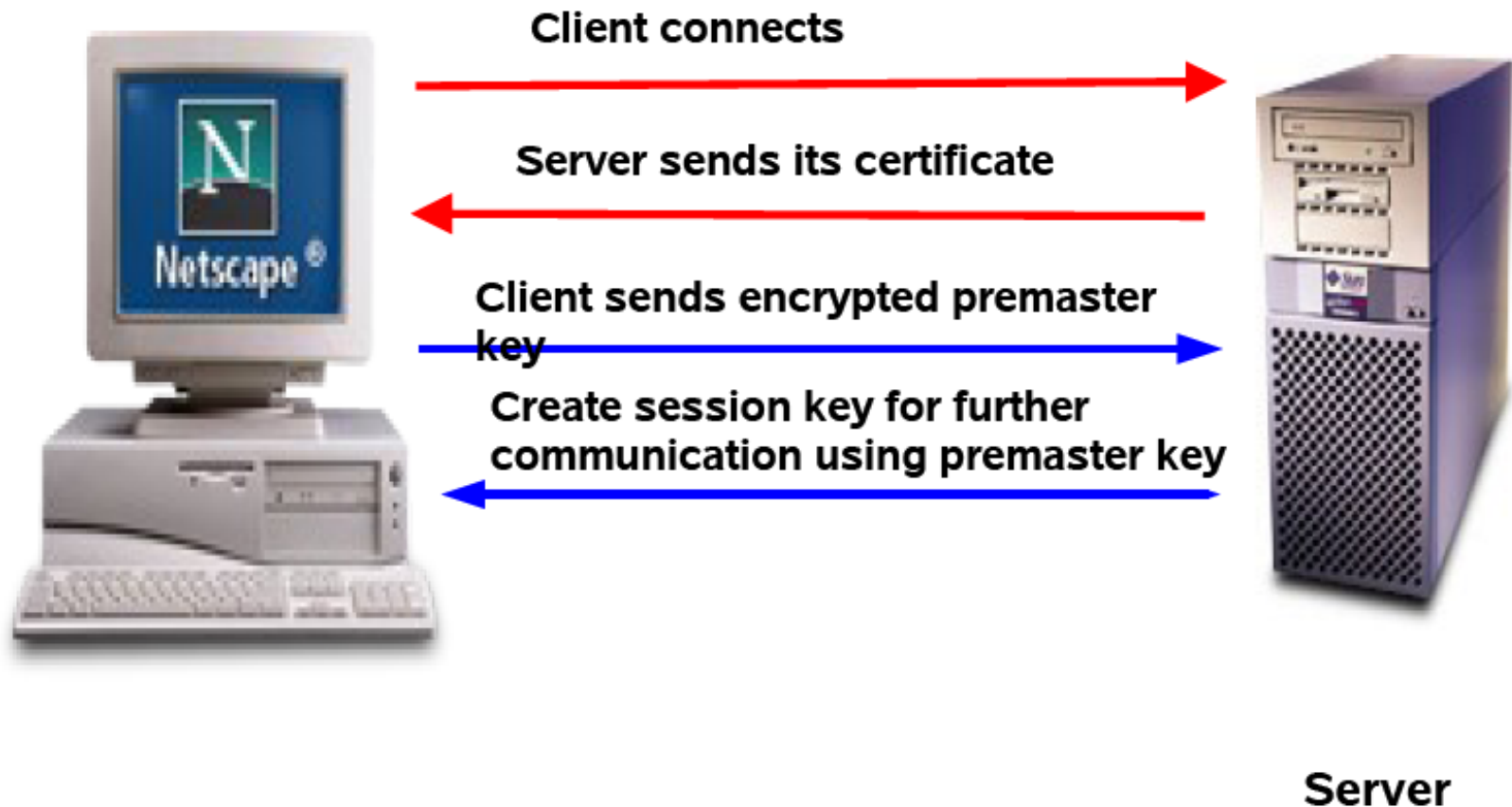
Details Of The SSL Protocol (cont.)

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.
-

SSL and Security Keys

- Uses public/private key (asymmetric) scheme to create secret key (symmetric)
 - Secret key is then used for encryption of data
 - SSL operation is optimized for performance: Using symmetric key for encryption is a lot faster than using asymmetric keys
-

SSL Key Exchange (Simplified)



SSL and Encryption

- You need only server's certificate in order to have encrypted data transfer
 - This is the reason why you don't need to install client certificate on your browser in order to send your credit card number securely (with privacy and data integrity)

SSL and Authentication

- Server authentication
 - Server needs to provide its own certificate to a client in order to authenticate itself to the client
 - A Web server typically has a CA-signed certificate and it provides it to its clients
 - Client authentication
 - Client needs to provide its own certificate to a server in order to authenticate itself to the server
 - Mutual authentication
-

SSL and Authentication

- In a typical “browser talking to web server” communication, only server authentication is needed
 - When you send your credit card to a server, you want to make sure the server is who it claims it is
 - In the future of B2B environment, client certification would be also required
 - The server wants to make sure it is talking to a client whose identity is verified
-

SSL and Web-tier Security

- Encrypted password move **from** the browser to the web server
- Encrypted data move **between** the browser and the web server
- Server authentication
 - Done before encrypted data transfer occurs
- Client Authentication
 - Not used in most cases

Certificates & Keytool Utility

What is a Certificate?

- A certificate is like **digital driver license**
 - A certificate is **cryptographically signed** and is practically impossible for anyone else to forge
 - A certificate can be **purchased from (signed by) a well-known CA** (Certificate Authority) like Verisign (for a fee)
 - A certificate **can be self-signed** when authentication over the internet is not really a concern, that is only data privacy and integrity are important
-

What is Server Certificate?

- A certificate that contains information about the server
 - Server's public key
 - Other misc. information
- Web server must have an associated **certificate** for each external interface, or IP address, that accepts secure connections
 - HTTP service of Tomcat will not run unless a server certificate has been installed

Why Server Certificate is Needed?

- Enables server authentication
 - ❑ Verifies the server's identity to the client
 - ❑ Client would need to have an access to the server certificate
 - ❑ Server sends server certificate as part of SSL key handshake
 - ❑ HTTPS service of Tomcat would not work unless a server certificate is installed
-

keytool Utility of JDK

- A key and certificate management utility
 - Enables users to **create** and administer their own public/private key pairs and associated certificates
 - Ships with JDK (Uses RSA-based JCE provider as default)
 - Allows users to **cache** the public keys (in the form of certificates) of their communicating peers
 - Stores the keys and certificates in a so-called **keystore**
-

JSSE

What is JSSE?

- Java API for Secure Sockets Layer (SSL)
 - Now standard part of J2SE 1.4
 - SSL 3.0 and TLS 1.0
 - Supports
 - Encryption
 - Server authentication
 - Optional client authentication
 - Data integrity
-

Why JSSE?

- 100% pure Java implementation
 - Abstracts the complex underlying cryptographic algorithms and thus minimizes the risk of creating subtle and dangerous security vulnerabilities
 - Uses algorithms, keys transparently
 - Simple to use to create secure application
-

JSSE Framework

- Supplements `java.security` and `java.net` packages
 - Provides `javax.net` and `javax.net.ssl` packages
 - Extends networking socket classes, trust and key managers, and a socket factory framework for encapsulating socket creation behavior
-

SunJSSE Provider

- JSSE provider that comes with JDK 1.4.1
 - Installed and pre-registered with the Java Cryptography Architecture
 - Supplies implementations of the SSL v3.0 and TLS v1.0 as well as most common SSL and TLS cipher suites
 - ❑ `getSupportedCipherSuites`
 - ❑ `getEnabledCipherSuites`
 - ❑ `setEnabledCipherSuites`
-

JSSE Programming example: Server Side

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

public class Server {
    int port = portNumber;
    SSLServerSocket server;
    try {
        SSLServerSocketFactory factory =
            (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
        server =
            (SSLServerSocket)factory.createServerSocket(portNumber);
        SSLSocket client = (SSLSocket) server.accept();

        // Create input and output streams as usual
        // send secure messages to client through the
        // output stream
        // receive secure messages from client through
        // the input stream
    } catch (Exception e) {
    }
}
```


JSSE Programming example: Client Side

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

public class Client {
    ...
    try {
        SSLSocketFactory factory = (SSLSocketFactory)
                                   SSLSocketFactory.getDefault();

        server =
            (SSLServerSocket) factory.createServerSocket(portNumber);
        SSLSocket client =
            (SSLSocket) factory.createSocket(serverHost, port);

        // Create input and output streams as usual
        // send secure messages to server through the
        // output stream receive secure
        // messages from server through the input stream
    } catch (Exception e) {
    }
}
```

SSL Support in Tomcat

SSL on Tomcat

- You need the following modules
 - JSSE (Java Secure Socket Extension)
 - Server certificate keystore
 - An HTTPS connector
 - You have to install and configure SSL HTTPS connector over Tomcat
-

JSSE

- Included in Java WSDP
 - <JWSDP-Install>/common/jsse.jar
- Provides Java packages that support SSL/TLS (jsse.jar)
- SSL supports Encryption, server authentication, message integrity over TCP/IP
 - Data over any application level protocol (HTTP, FTP, Telnet, ...) can be securely protected
- Based on Certificate-based (Public and Private key) security scheme

Steps of Installing and Configuring SSL over Tomcat

1. Generate a key pair and a self-signed Server certificate

- `keytool -genkey -keyalg RSA -alias tomcat -keystore <keystore_filename>`
 - Enter password, fully-qualified name of your server, organizational unit, organization, etc.
 - Tomcat is looking for the keystore to have the name **.keystore** in the home directory of the machine on which Tomcat is running as a default
-

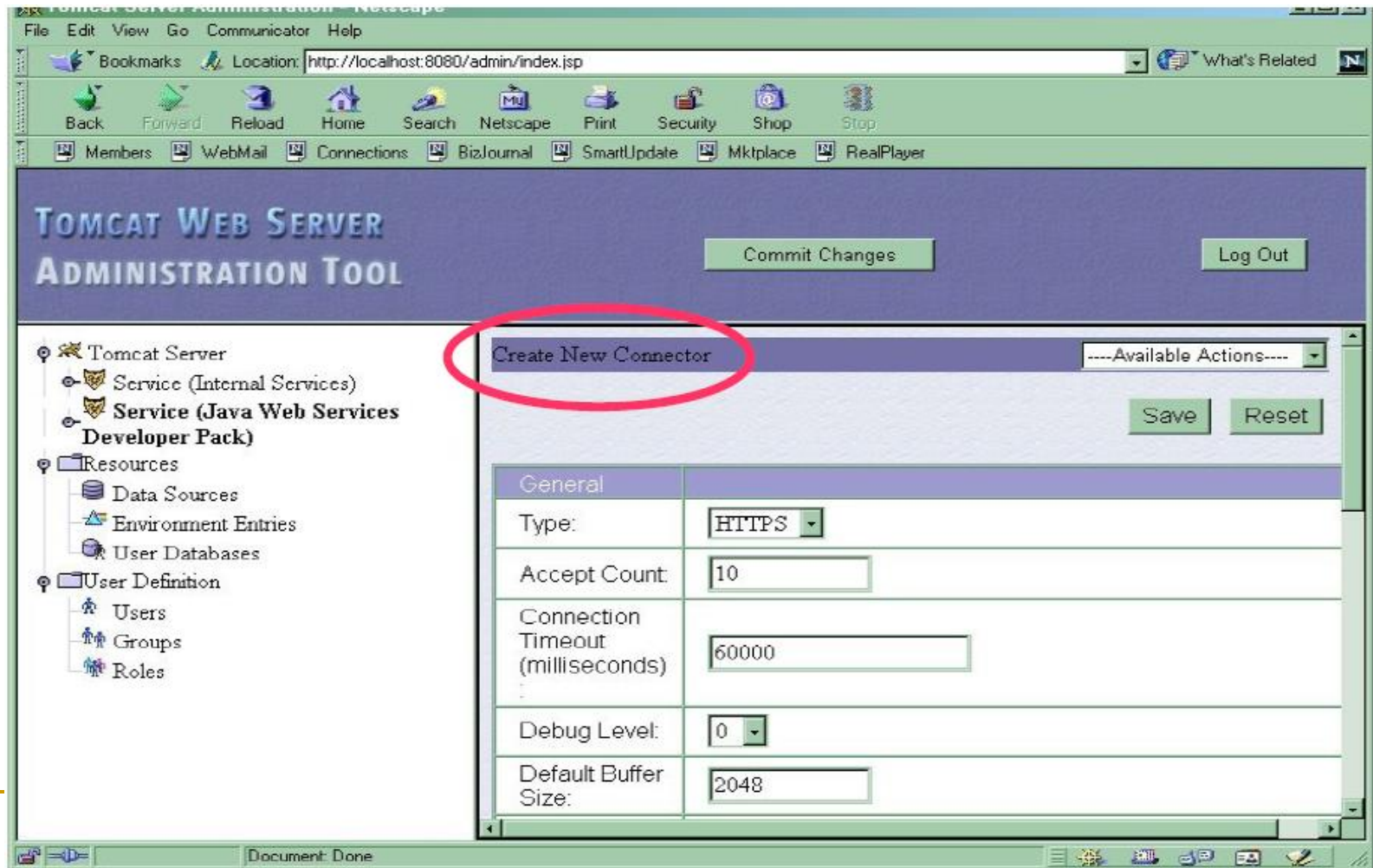
1.1 Example: keytool usage

```
C:\>keytool -genkey -keyalg RSA -alias tomcat -keystore
\tmp\keyfile.keystore
Enter keystore password: changeit
What is your first and last name?
[Unknown]: localhost
What is the name of your organizational unit?
[Unknown]: sun
What is the name of your organization?
[Unknown]: mde
What is the name of your City or Locality?
[Unknown]: burlington
What is the name of your State or Province?
[Unknown]: ma
What is the two-letter country code for this unit?
[Unknown]: us
Is CN=localhost, OU=sun, O=mde, L=burlington, ST=ma, C=us correct?
[no]: yes
Enter key password for <tomcat>
(RTURN if same as keystore password):
```

2. Configure SSL Connector & Restart Tomcat

- By default, SSL HTTPS is not enabled in Tomcat
- You enable and configure an SSL HTTPS Connector on port 8443 in one of two methods
 - via Admintool
 - Modify (actually uncomment SSL connector element) `<JWSDP_HOME>/conf/server.xml` as described in
 - `<JWSDP_HOME>/docs/tutorial/doc/WebAppSecurity6.html#68482`
- Restart Tomcat

2.1 Admintool



2.2 SSL Connector Element in server.xml

```
<!-- SSL Connector on Port 8443 -->
<Connector
  className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8443"
  minProcessors="5"
  maxProcessors="75"
  enableLookups="false"
  acceptCount="10"
  connectionTimeout="60000"
  debug="0"
  scheme="https"
  secure="true">
  <Factory
    className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
    clientAuth="false" protocol="TLS" />
</Connector>
```

3. Verify SSL Support

- From the browser, go to
 - <https://localhost:8443/>
- Port 8443 is where SSL connector is created

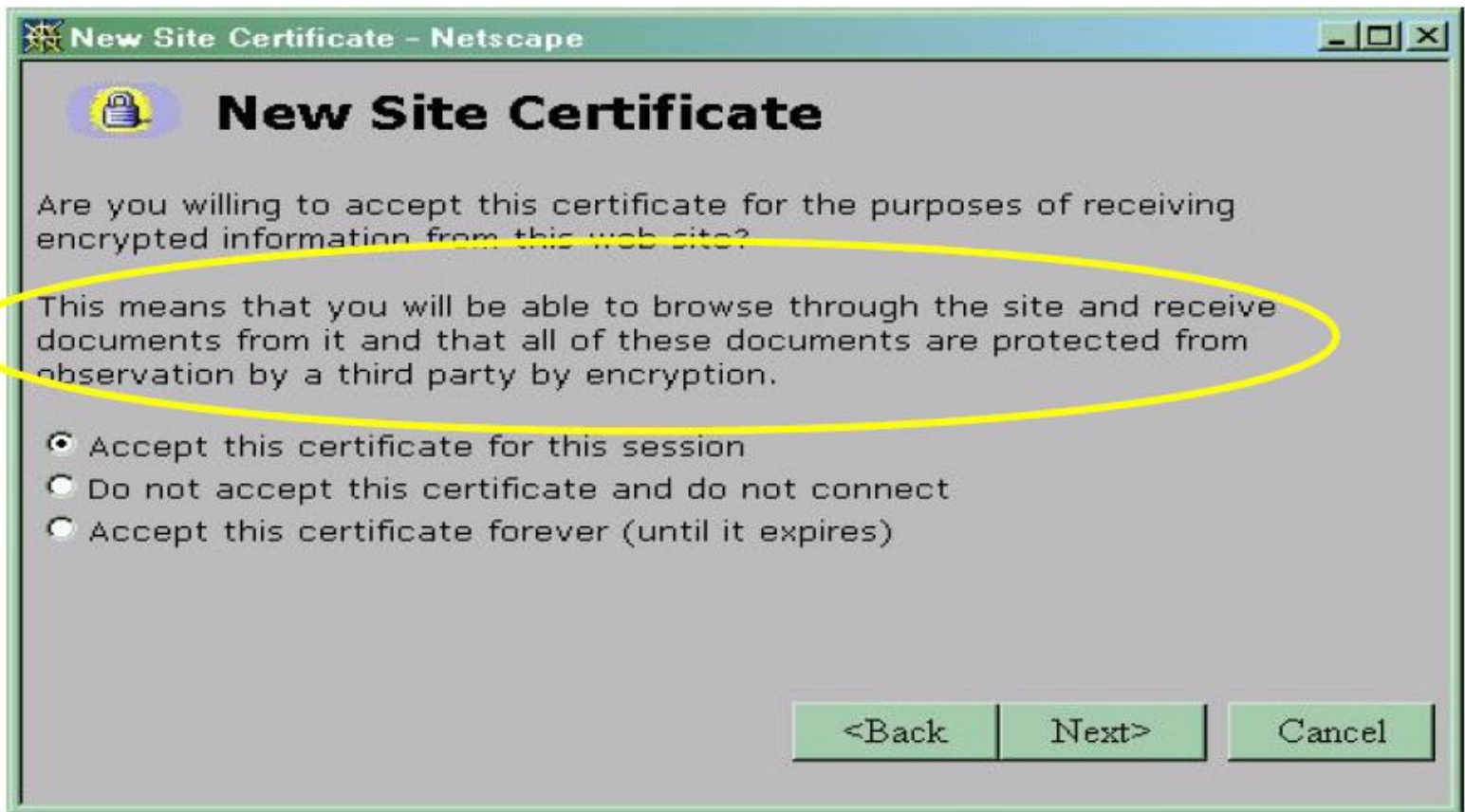
3.1 Example: Verify SSL Support



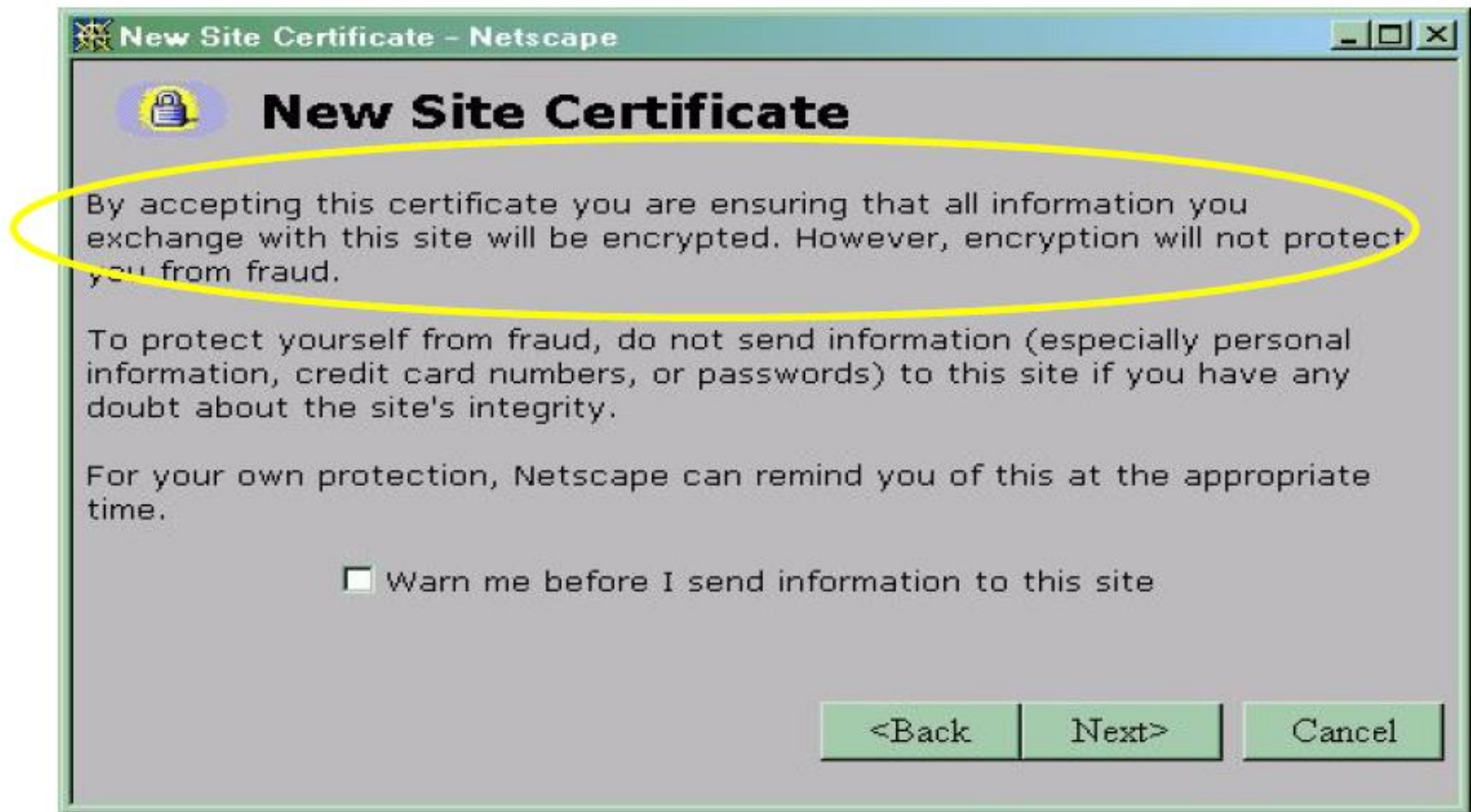
3.2 Example: Verify SSL Support



3.3 Example: Verify SSL Support



3.4 Example: Verify SSL Support



3.5 Example: Verify SSL Support



JSSE Programming example 2:

- **NOTE:** Before attempting to execute this application, look at the code first and then go to page 63 for execution details. This application will not execute correctly unless you follow the steps beginning on page 63.
-

```
// 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();
```

← Close down I/O streams and the socket

```
} // 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
```

```
// 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.*;
```

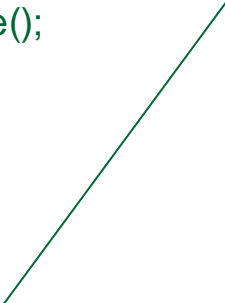
LoginClient.java
Client Class for SSL Implementation

```
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 );
        }
    }
}
```

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
```

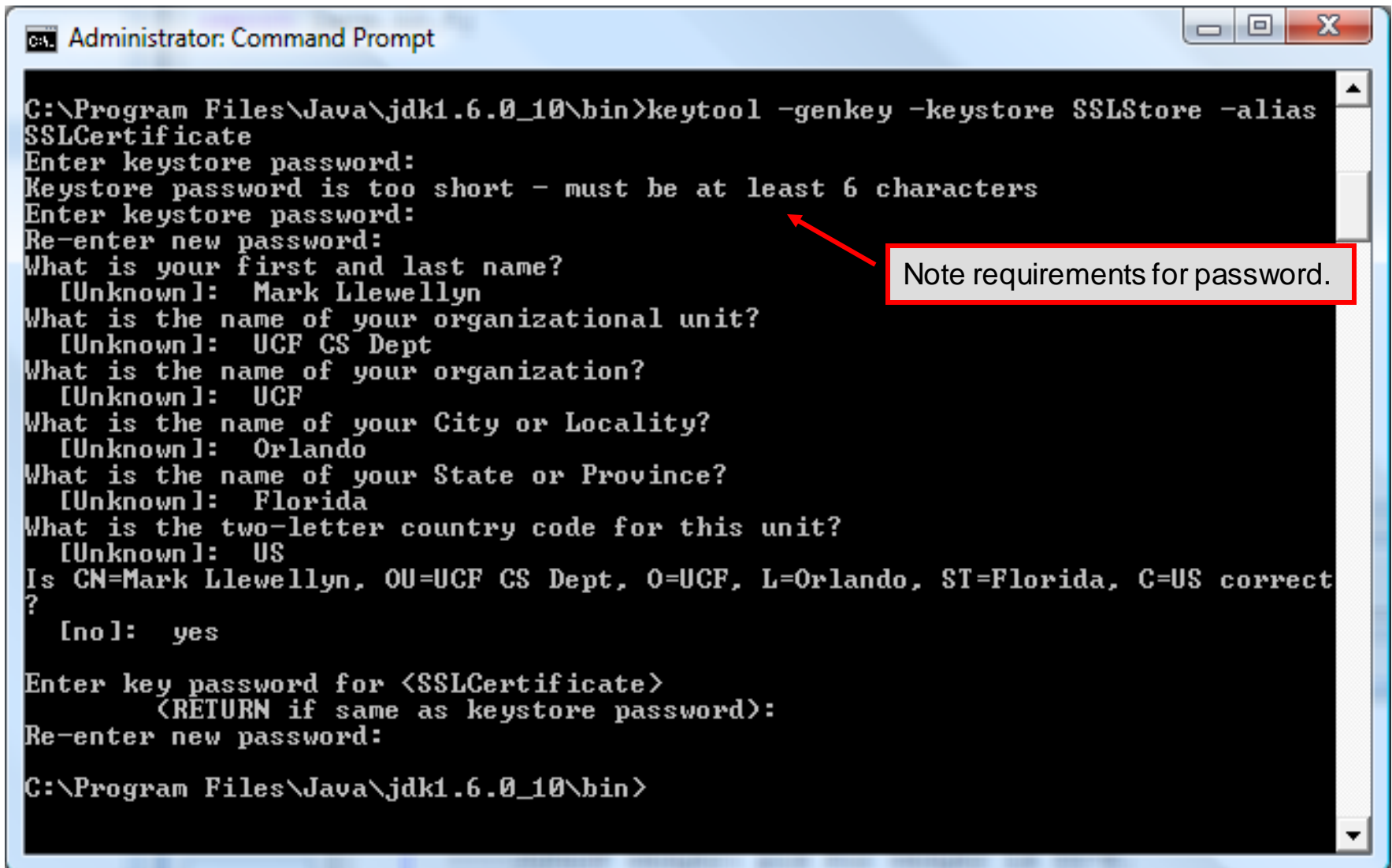


```
// execute application
public static void main( String
args[] )
{
    new LoginClient();
}
}
```

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=Florida, C=US
Issuer: CN=Mark Llewellyn, OU=School of Computer Science, O=UCF, L=Orlando, ST=Florida, 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

Administrator: Command Prompt

```
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.

Administrator: Command Prompt

```
Certificate stored in file <mycert.cer>  
C:\Program Files\Java\jdk1.6.0_10\bin>type mycert.cer  
-----BEGIN CERTIFICATE-----  
MIIDFjCCAtSgAwIBAgIESYtCNjAlBgqhkJ0OAQDBQAwbjELMAkGA1UEBhMCUUMxEDAOBgNUBAgT  
B0Zsb3JpZGExEDAOBgNUBAcTB09ybGFuZG8xDDAKBgNUBAOTa1UDRjEUMBI GA1UECxMLUUNGI ENT  
IERlcHQx FzAUBgNUBAMTDk1hcmsgTGxld2UsbHluMB4XDTA5MDIwNTE5NDcwMloXDTA5MDUwNjE5  
NDcwMlowbjELMAkGA1UEBhMCUUMxEDAOBgNUBAgTB0Zsb3JpZGExEDAOBgNUBAcTB09ybGFuZG8x  
DDAKBgNUBAOTa1UDRjEUMBI GA1UECxMLUUNGI ENTIERlcHQx FzAUBgNUBAMTDk1hcmsgTGxld2Us  
bHluMIIBuDCASwGBYqGSM44BAEwggEfAoGBAP1/U4EddRIpUt9KnC7s5Of2EbdSP09EAMMeP4C2  
USZpRU1AI1H7WT2NWPq/xfW6MPbLm1Us14E7gB00b/JmYLdrnUClpJ+f6AR7ECLCT7up1/63xhv4  
O1fnxqimFQ8E+4P208UewwI1UBNaFpEy9nXzrith1yr08iIDGZ3RSAHHAhUA12BQjxUjC8yykrnC  
ouuEC/BYHPUCgYEA9+GghdabPd7LvKtcNr hXuXmUr7v6OugC+UdMCz0HgmdRWUeOutRZT+ZxBxCB  
gLRJFpEj6EwoFh03zwkyjMim4TwWeotUfI0o4K0uHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvghR  
kImog9/hWuWfBpKLZ16Ae1U1ZAFMO/7PSSoDgYUAAoGBAPv/bmegtq4IMK46AA3PlZ2WpesJaKUK  
CM8FNmsnObgNiKYljHGxL+9KPJjyWY2uWxU79Is/TxmfGL92kdUa8+LfmDvA3iPSzQlihdLv2FKf  
2REZqwkht80tZ3Qa360829+H8hzsLwFiOUUbasKDHshMPdLx2kobbbGTLUM+DoFF4MAAsGBYqGSM44  
BAMFAAMvADAsAhRwz+MMw5Y9akly5y0KMnxJBa1UUwIUUUjEbg7GUUDuMWsWJKFBOsFDv0=  
-----END CERTIFICATE-----  
C:\Program Files\Java\jdk1.6.0_10\bin>
```

Contents of the
certificate.

Creating Keystore and Certificate

Import the certificate into a new truststore.

Administrator: Command Prompt

```
C:\Program Files\Java\jdk1.6.0_10\bin>keytool -import -alias sslcertificate -file 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>_
```

Creating Keystore and Certificate

Administrator: Command Prompt

```
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
```

```
*****
*****
```

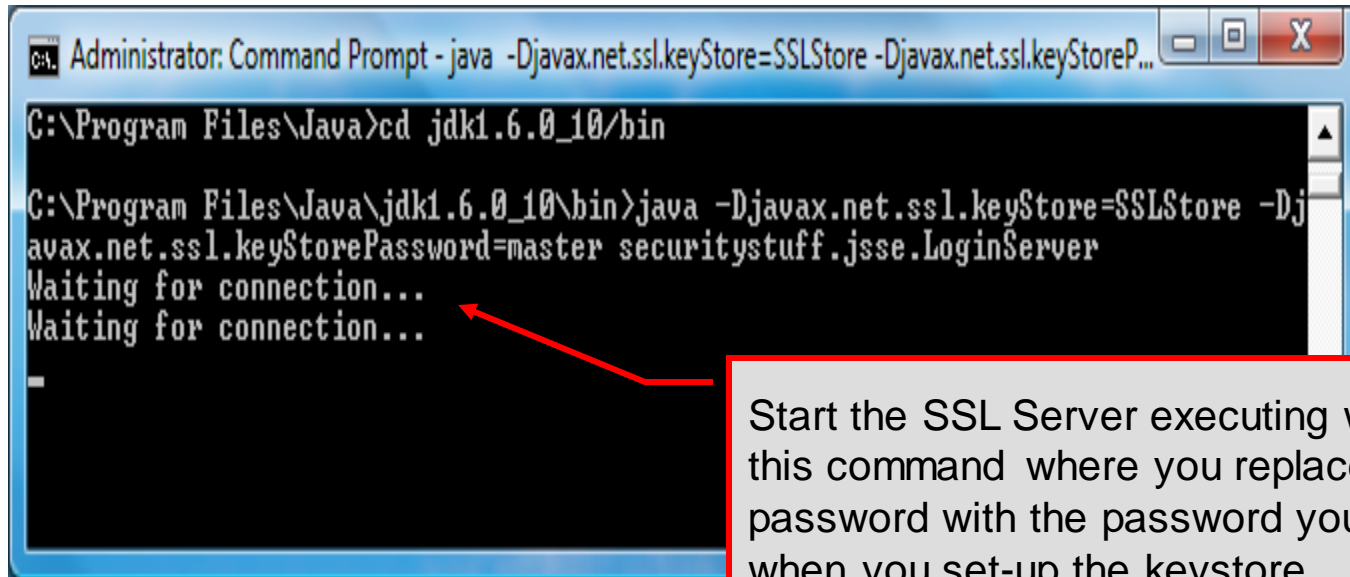
```
C:\Program Files\Java\jdk1.6.0_10\bin>_
```

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.



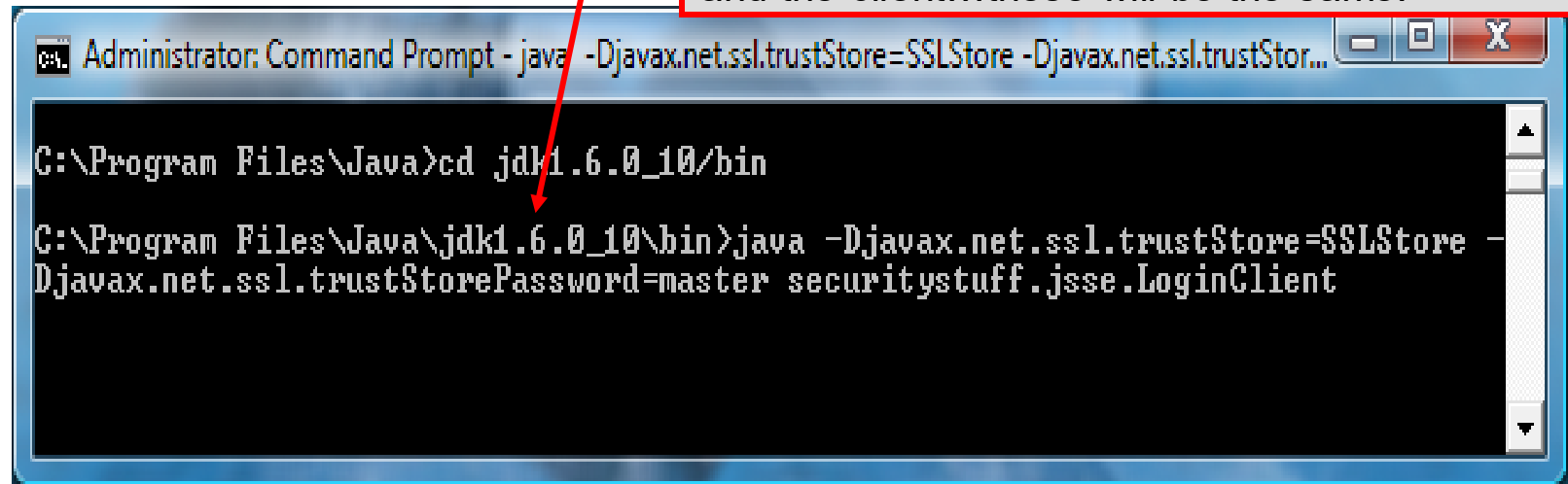
```
Administrator: Command Prompt - java -Djavax.net.ssl.keyStore=SSLStore -Djavax.net.ssl.keyStoreP...
C:\Program Files\Java>cd jdk1.6.0_10\bin
C:\Program Files\Java\jdk1.6.0_10\bin>java -Djavax.net.ssl.keyStore=SSLStore -Dj
avax.net.ssl.keyStorePassword=master securitystuff.jsse.LoginServer
Waiting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

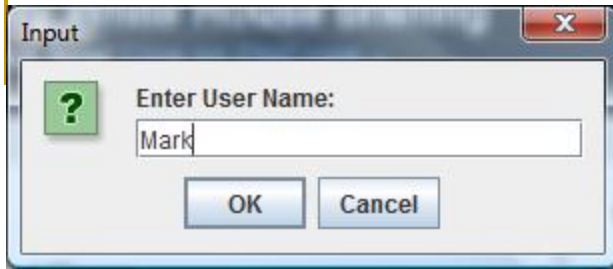
Start the SSL Client application executing with this command where you replace this password with the password you used when you set-up the keystore. Since we are using the same keystore for the server and the client...these will be the same.



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - java -Djavax.net.ssl.trustStore=SSLStore -Djavax.net.ssl.trustStorePassword=master securitystuff.jsse.LoginClient". The window has a blue title bar and standard Windows window controls. The command prompt shows the following commands and output:

```
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
```

A red arrow points from the text box above to the second command line in the command prompt.



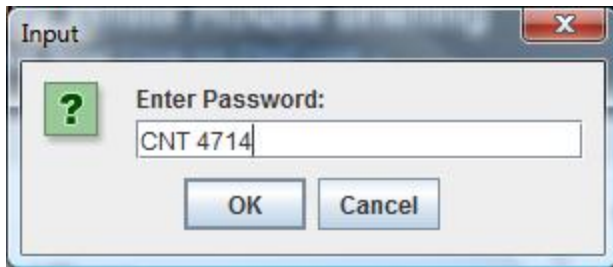
Input

Enter User Name:

Mark

OK Cancel

This is a standard Windows-style input dialog box. It has a title bar with a close button (X). The main area contains a green question mark icon, a label 'Enter User Name:', a text input field containing the text 'Mark', and two buttons at the bottom: 'OK' and 'Cancel'.



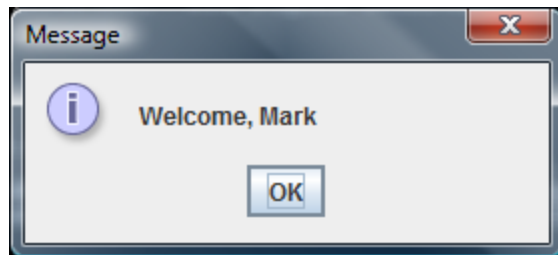
Input

Enter Password:

CNT 4714

OK Cancel

This is a standard Windows-style input dialog box. It has a title bar with a close button (X). The main area contains a green question mark icon, a label 'Enter Password:', a text input field containing the text 'CNT 4714', and two buttons at the bottom: 'OK' and 'Cancel'.



Message

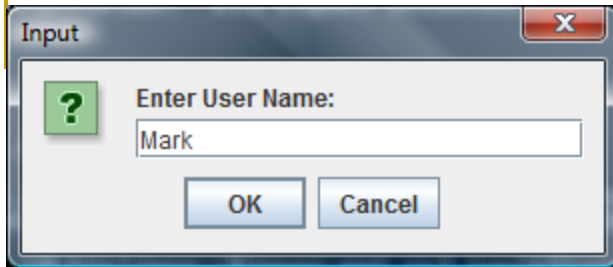
Welcome, Mark

OK

This is a standard Windows-style message dialog box. It has a title bar with a close button (X). The main area contains a blue information icon (i), the text 'Welcome, Mark', and an 'OK' button at the bottom.

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

Authentication
successful – user is
logged on.



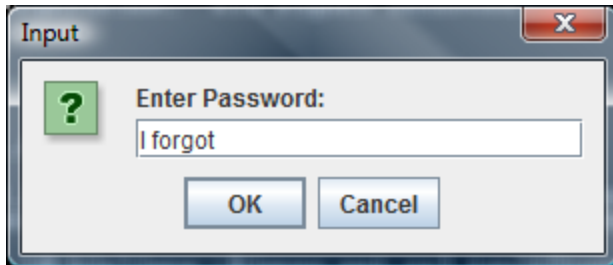
Input

Enter User Name:

Mark

OK Cancel

This is a standard Windows-style input dialog box. It has a title bar with a close button (X). The main area contains a green question mark icon, a label 'Enter User Name:', a text input field containing the text 'Mark', and two buttons at the bottom: 'OK' and 'Cancel'.



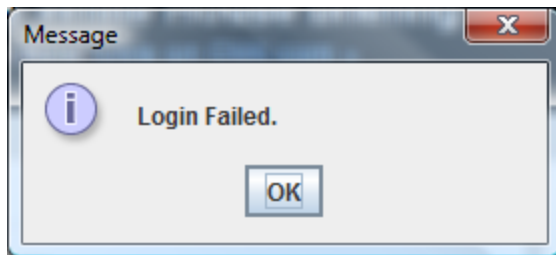
Input

Enter Password:

I forgot

OK Cancel

This is a standard Windows-style input dialog box. It has a title bar with a close button (X). The main area contains a green question mark icon, a label 'Enter Password:', a text input field containing the text 'I forgot', and two buttons at the bottom: 'OK' and 'Cancel'.



Message

Login Failed.

OK

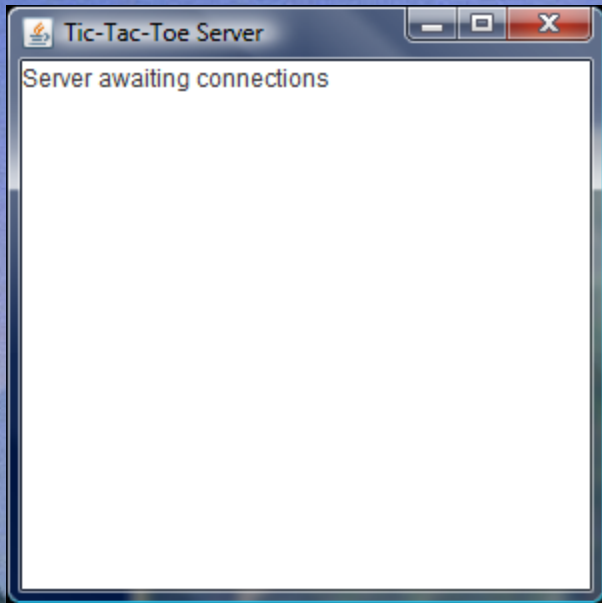
This is a standard Windows-style message dialog box. It has a title bar with a close button (X). The main area contains a purple information icon (i), a label 'Login Failed.', and a single 'OK' button at the bottom.

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.

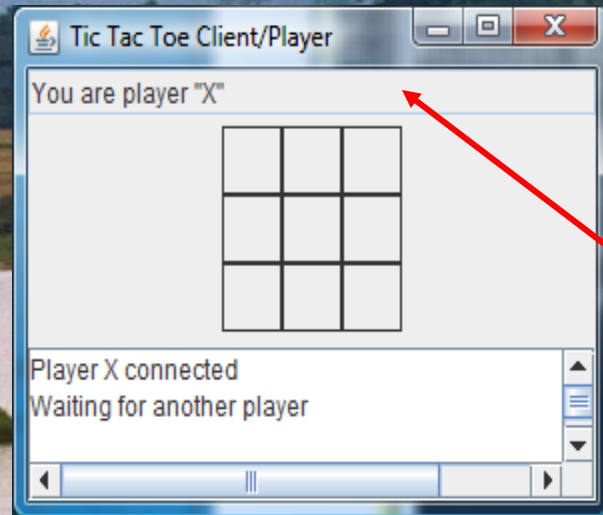
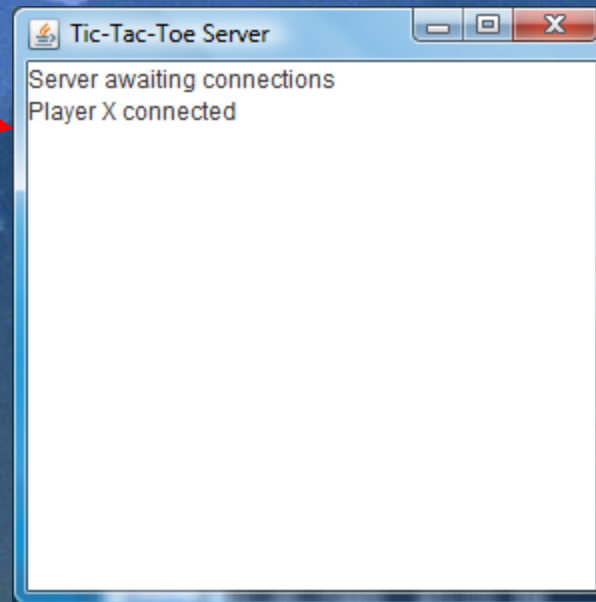
Multithreaded Socket Client/Server Example

- 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.
-

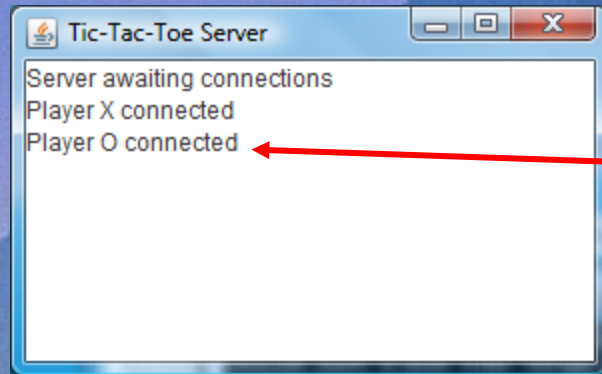


Start server
running...

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

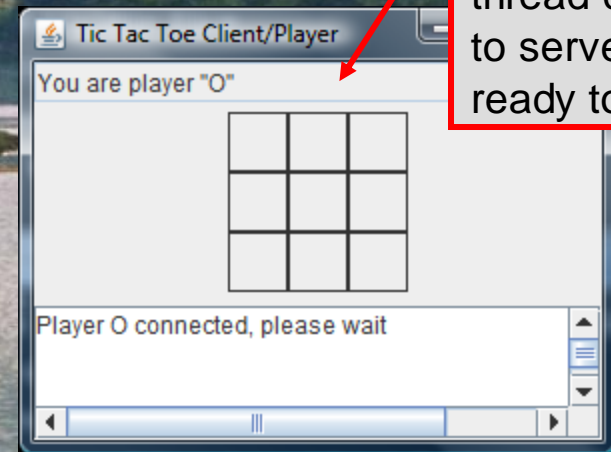
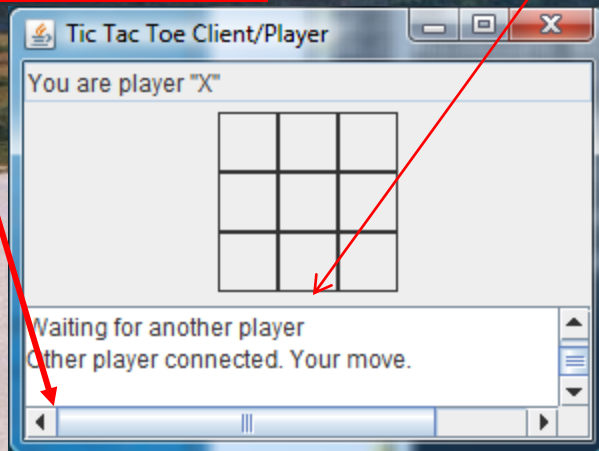


Start first player thread



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.

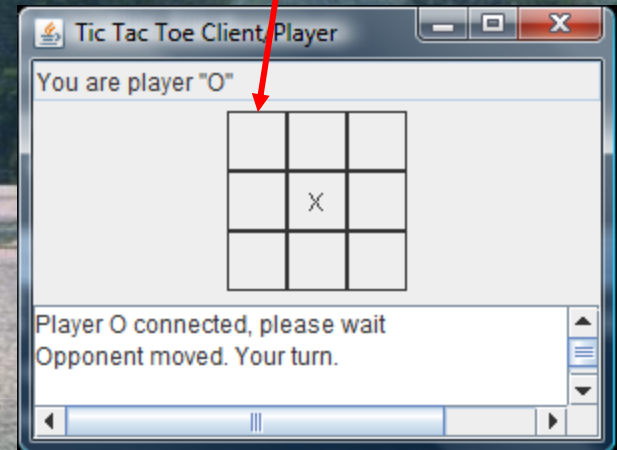
```
Tic-Tac-Toe Server
Server awaiting connections
Player X connected
Player O connected

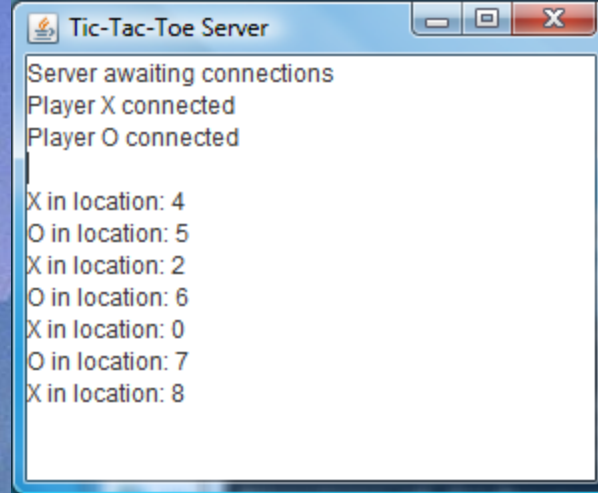
X in location: 4|
```

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

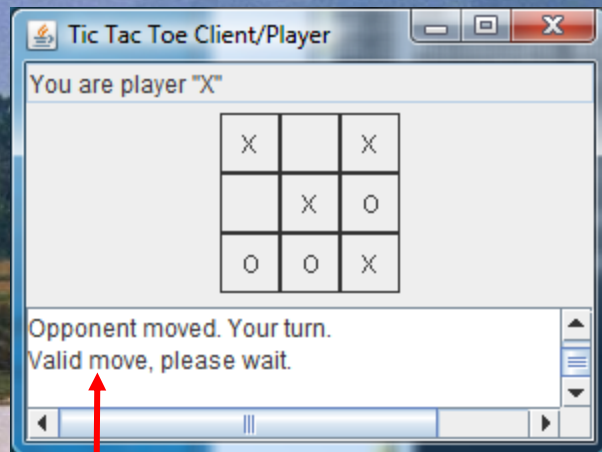


Player X makes a move by placing an "X" marker in location 4 of the game board.





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.



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

