

CNT 4714: Enterprise Computing Spring 2009

Introduction to Servlet Technology– Part 3

Instructor : Dr. Mark Llewellyn
 markl@cs.ucf.edu
 HEC 236, 407-823-2790
 <http://www.cs.ucf.edu/courses/cnt4714/spr2009>

School of Electrical Engineering and Computer Science
University of Central Florida

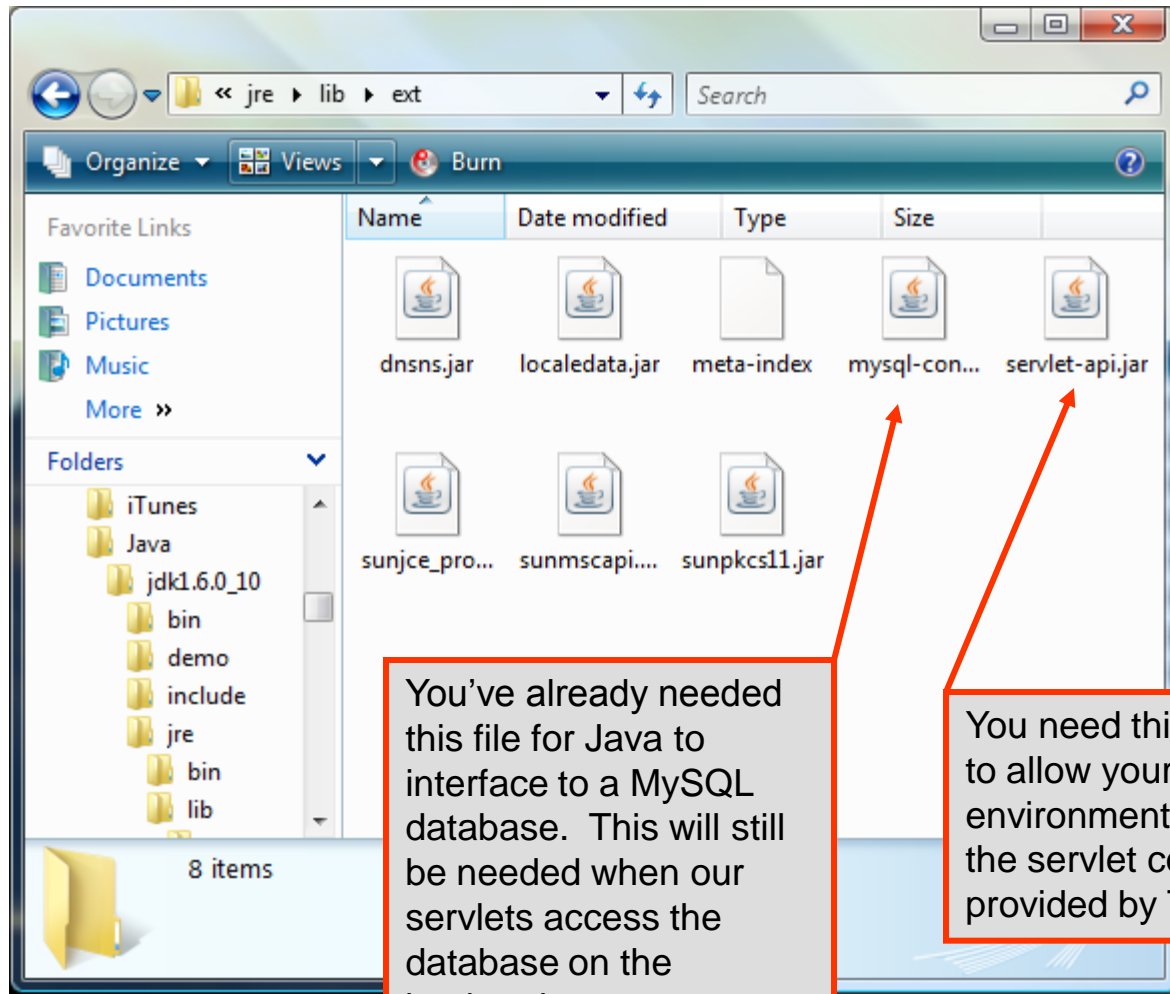


Tomcat/Java Configuration - The Servlet API

- Your Tomcat installation includes the `servlet-api.jar` file. This file can be found in the `/common/lib` folder in Tomcat. Copy this file into your `jdk/jre/lib/ext` folder to allow the java compiler access to the `javax.servlet` package.
- Your Java set-up may already have this installed.



Tomcat/Java Configuration - The Servlet API



You've already needed this file for Java to interface to a MySQL database. This will still be needed when our servlets access the database on the backend.

You need this .jar file here to allow your Java environment to interface to the servlet container provided by Tomcat.



More XHTML Document Details

- Let's look a bit closer at what happens in our servlet as it executes. (See the servlet code on page 23 of servlets-part 2 notes.)

```
protected void doGet( HttpServletRequest request,  
                    HttpServletResponse response )  
    throws ServletException, IOException {
```

- This line begins the overridden method `doGet` to respond to the get requests. In this case, the `HttpServletRequest` object parameter represents the client's request and the `HttpServletResponse` object parameter represents the server's response to the client.
- If method `doGet` is unable to handle a client's request, it throws an exception of type `javax.servlet.ServletException`. If `doGet` encounters an error during stream processing (when reading from the client or writing to the client), it throws a `java.io.IOException`.



More XHTML Document Details (cont.)

```
response.setContentType( "text/html" );  
    PrintWriter out = response.getWriter();
```

- The first line above uses the response object's `setContentType` method to specify the content type of the document to be sent as the response to the client. This enables the client browser to understand and handle the content it receives from the server. The content type is also referred to as the **MIME (Multipurpose Internet Mail Extension)** type of the data. In this servlet, the content type is `text/html` to indicate to the browser that the response is an XHTML document.
- The second line above uses the response object's `getWriter` method to obtain a reference to the `PrintWriter` object that enables the servlet to send content to the client. If the response is binary data, like an image, method `getOutputStream` would be used to obtain a reference to a `ServletOutputStream` object.



More XHTML Document Details (cont.)

```
out.println( "<?xml version = \"1.0\"?>" );
out.println( "<!DOCTYPE html PUBLIC \"- //W3C//DTD " +
    "XHTML 1.0 Strict//EN\" \"http://www.w3.org\" +
    \"/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
out.println("<html xmlns =
    \"http://www.w3.org/1999/xhtml\">" );
// head section of document
    out.println( "<head>" );
    out.println( "<title>Welcome to Servlets!</title>" );
    out.println( "</head>" );
// body section of document
    out.println( "<body>" );
    out.println( "<h1>Welcome To The World Of Servlet
        Technology!</h1>" );
    out.println( "</body>" );
// end XHTML document
    out.println( "</html>" );
```

- These lines create the XHTML document shown in the box on page 23 of *servlets-part 2 notes*.



Deploying a Web Application

- Servlets, JSPs and their supporting files are deployed as part of a Web application.
- Typically, Web applications are deployed in the `webapps` subdirectory of Tomcat.
- A Web application has a well-known directory structure in which all the files that are part of the application reside.
- This directory structure is created by the server administrator in the `webapps` directory, or the entire directory structure can be archived in a Web application archive file known as a `WAR` file (ending with a `.war` file extension) which is placed in the `webapps` directory.



Deploying a Web Application (cont.)

- The Web application directory structure contains a context root, which is the top-level directory for an entire Web application along with several subdirectories as shown below:

context root – The root directory for the Web application. All the JSPs, HTML documents, servlets and supporting files such as images and class files reside in this directory or one of the subdirectories. The name of this directory is specified by the Web application creator. To provide structure in a Web application, subdirectories can be placed in the context root. It is common to see an images subdirectory, for example.

WEB-INF – This subdirectory contains the Web application deployment descriptor **web.xml**.

WEB-INF/classes – This subdirectory contains the servlet class files and other supporting class files used in a Web application. If the classes are part of a package, the complete package directory structure would begin here.

WEB-INF/lib – This subdirectory contains Java archive (JAR) files. The JAR files can contain servlet class files and other supporting class files.



Deploying a Web Application (cont.)

- As we mentioned in the previous section of notes, Tomcat will default to a welcome page which is specified in the `web.xml` file. The standard default values were shown on page 9 in the previous set of notes.
- If you do not create one of these files, the default page for a web application is not very appealing.



Deploying a Web Application (cont.)

- Since we would like our clients to see something more appropriate than the default web application page, you should create your own web application welcome page.
- This page is simply an HTML page and I've created one for the web applications we create from this point forward. I've simply modeled the page using our course web page as a template. The code for this page is included on the course code page if you want to use it, but feel free to design your own.
- I'll utilize this page as a home page for all of our servlets and JSPs that we'll see later in the course.
- I've also created a new web application named CNT4714 that we'll use for our future servlets and JSPs.
- Now, when the client enters the URL, <http://localhost:8080/CNT4714> they will see the home page shown in the next slide.



CNT 4714 - Spring 2009 - Windows Internet Explorer

http://localhost:8080/CNT4714/

File Edit View Favorites Tools Help

Google 8 Search + - Bookmarks Find ABC Check AutoFill Sign In

Blackboard Learning System CNT 4714 - Spring 2009



UCF

CNT 4714 - Enterprise Computing Spring 2009

Instructor: [Dr. Mark Llewellyn](#)
HEC 236
(407) 823-2790
Email to: markl@cs.ucf.edu

Welcome To The CNT 4714 Servlet Home Page

SERVLET HOME PAGE

Click any of the links below to directly invoke the corresponding servlet

[Click here to invoke a simple Welcome Servlet](#)

[Click here to invoke a more personal Welcome Servlet](#)

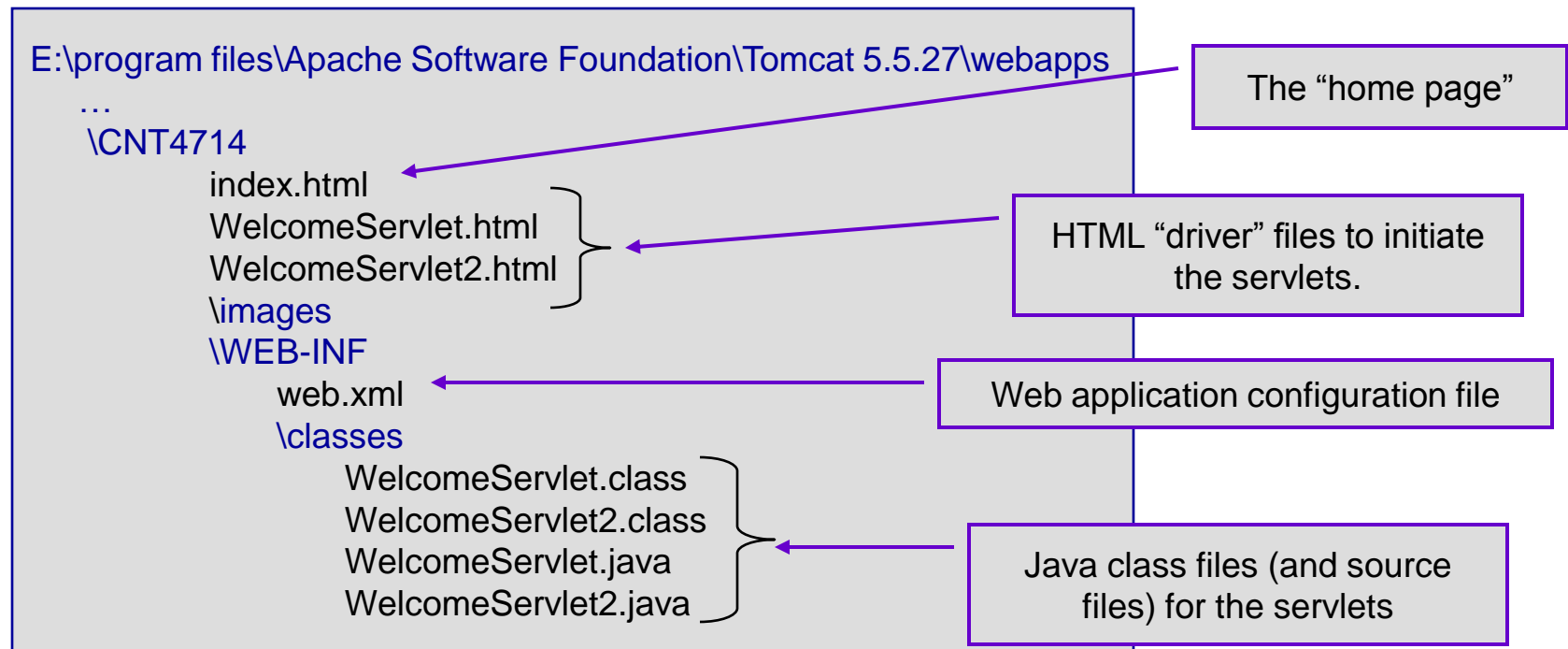
[Click here to run the Redirection Servlet](#)

Done Internet | Protected Mode: Off 100%



Deploying a Web Application (cont.)

- The Web application directory structure that I set up for the CNT4714 web application looks like the following:



A Closer Look at the `web.xml` File

The `web-app` element defines the configuration of each servlet in the Web application and the servlet mapping for each servlet.

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
```

```
<!-- General description of your Web application -->
```

```
<display-name>
  Servlet Technology
</display-name>
```

```
<description>
  This is the Web application in which we will
  demonstrate our JSP and Servlet examples.
</description>
```

The `display-name` element specifies a name which can be displayed to the server administrator on which the Web application is installed.

The `description` element specifies a description of the Web application that can also be displayed to the server administrator.



A Closer Look at the `web.xml` File

```
<!-- Servlet definitions -->
```

```
<servlet>  
  <servlet-name>welcome1</servlet-name>
```

Element `servlet` describes a servlet. There is one of these for each servlet in the Web application. Element `servlet-name` is the name chosen for the servlet.

```
  <description>  
    A simple welcome servlet that handles an HTTP get request.  
  </description>
```

Element `description` describes the servlet and can be displayed to the server administrator.

```
  <servlet-class>  
    WelcomeServlet  
  </servlet-class>  
</servlet>
```

Element `servlet-class` specifies the compiled servlet's fully qualified path name. In this case the servlet is defined by the class `WelcomeServlet`.

```
<!-- Servlet mappings -->
```

```
<servlet-mapping>  
  <servlet-name>welcome1</servlet-name>  
  <url-pattern>/welcome1</url-pattern>  
</servlet-mapping>
```

Element `servlet-mapping` specifies the `servlet-name` and `url-pattern` elements. The URL pattern helps the server determine which requests are sent to the servlet (`welcome1`). Since this web application will be installed as part of the `cnt4714` context root, the relative URL supplied to the browser to invoke the servlet is `/cnt4714/welcome1`.


```
</web-app>
```



Handling HTTP `get` Requests Containing Data

- When a requesting a document or resource from a Web server, it is often the case that data needs to be supplied as part of the request. The second servlet example in the previous set of notes responds to an HTTP `get` request that contains the name entered by the user. The servlet uses this name as part of the response to the client.
- Parameters are passes as name-value pairs in a `get` request. Within the source code for the second `WelcomeServlet2` you will find the following line (see next page):

```
String clientName = request.getParameter( "clientname" );
```



Invoke request object's
`getParameter` method



```
// welcomeServlet2.java
// Processing HTTP get requests containing data.

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class welcomeServlet2 extends HttpServlet {

    // process "get" request from client
    protected void doGet( HttpServletRequest request,
        HttpServletResponse response )
        throws ServletException, IOException
    {
        String clientName = request.getParameter( "clientname" );

        response.setContentType( "text/html" );
        PrintWriter out = response.getWriter();

        // send XHTML document to client

        // start XHTML document
        out.println( "<?xml version = \"1.0\"?>" );

        out.println( "<!DOCTYPE html PUBLIC \"-//w3c//DTD \" +
            \"XHTML 1.0 Strict//EN\" \"http://www.w3.org\" +
            \"/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );

        out.println(
            "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
        out.println( "<body bgcolor=white background=images/background.jpg lang=EN-US link=blue " );
        out.println( "<body style='tab-interval:.5in'>" );
        // head section of document
        out.println( "<head>" );
        out.println(
            "<title>Processing get requests with data</title>" );
        out.println( "</head>" );

        // body section of document
        out.println( "<body>" );
        out.println( "<h1>Hello " + clientName + ",<br />" );
            out.println();
        out.println( "welcome to the Exciting world of servlet Technology!</h1>" );
        out.println( "</body>" );

        // end XHTML document
        out.println( "</html>" );
        out.close(); // close stream to complete the page
    }
}
```

Invoke request object's
getParameter method



Handling HTTP `get` Requests Containing Data

(cont.)

- The `WelcomeServlet2.html` document provides a form in which the user can input their name into the text input element `clientname` and click the `Submit` button to invoke the servlet.
- When the user clicks the `Submit` button, the values of the input elements are placed in name-value pairs as part of the request to the server.
- Notice in the screen shot on the next page that the Tomcat server has appended `?clientname=Mark` to the end of the `action` URL. The `?` separates the **query string** (i.e., the data passed as part of the `get` request) from the rest of the URL in a `get` request. The name-value pairs are passed with the name and value separated by `=`. If there is more than one name-value pair, each pair is separated by an `&`.



Handling HTTP `get` Requests Containing Data

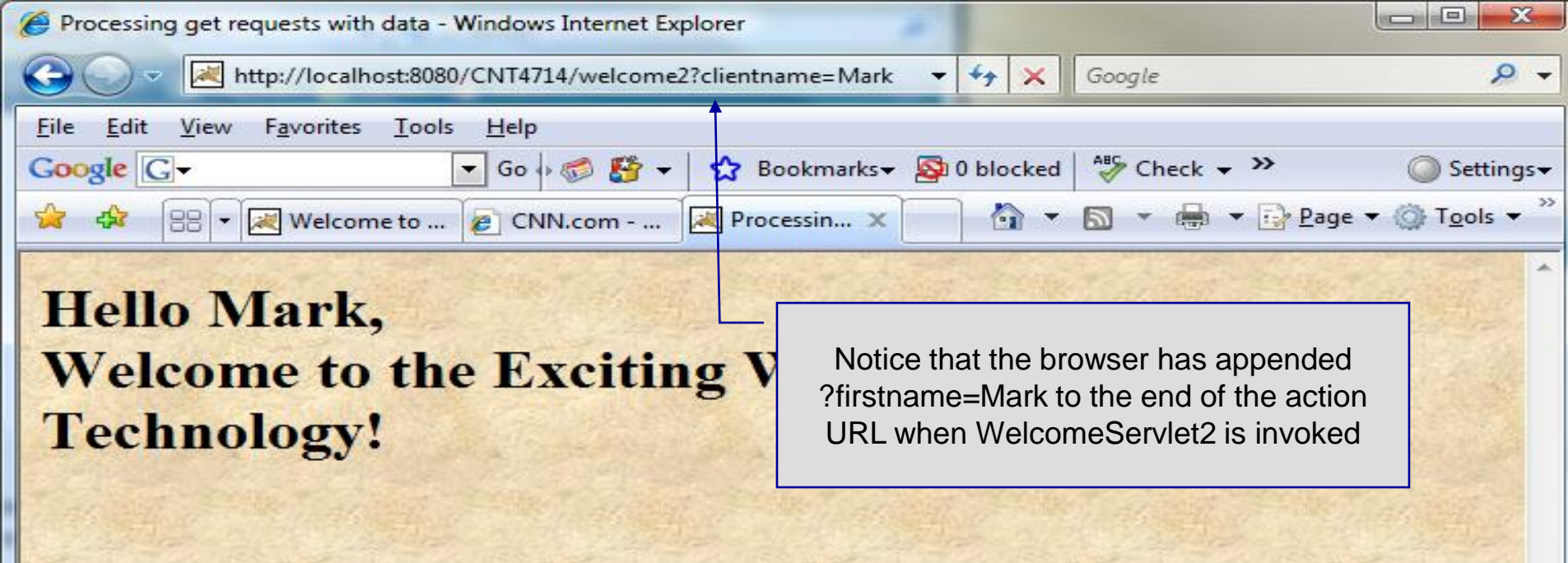
```
File Edit Format View Help
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- welcomeServlet2.html -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>A more personal welcome servlet - contains input data</title>
</head>
<body>
<font size = 4>
<body bgcolor=white background=images/background.jpg lang=EN-US
link=blue vlink=blue style='tab-interval:.5in' >
<br>
<form action = "/CNT4714/welcome2" method = "get">
<p><label>
Enter your name and click the submit button to run a more personal welcome servlet
<input type = "text" name = "clientname" />
<input type = "submit" value = "submit" /></label> (uses an HTTP Get request)
</p></label>
</form>
</body>
</html>
```

Context root is /CNT4714
Servlet alias is welcome2

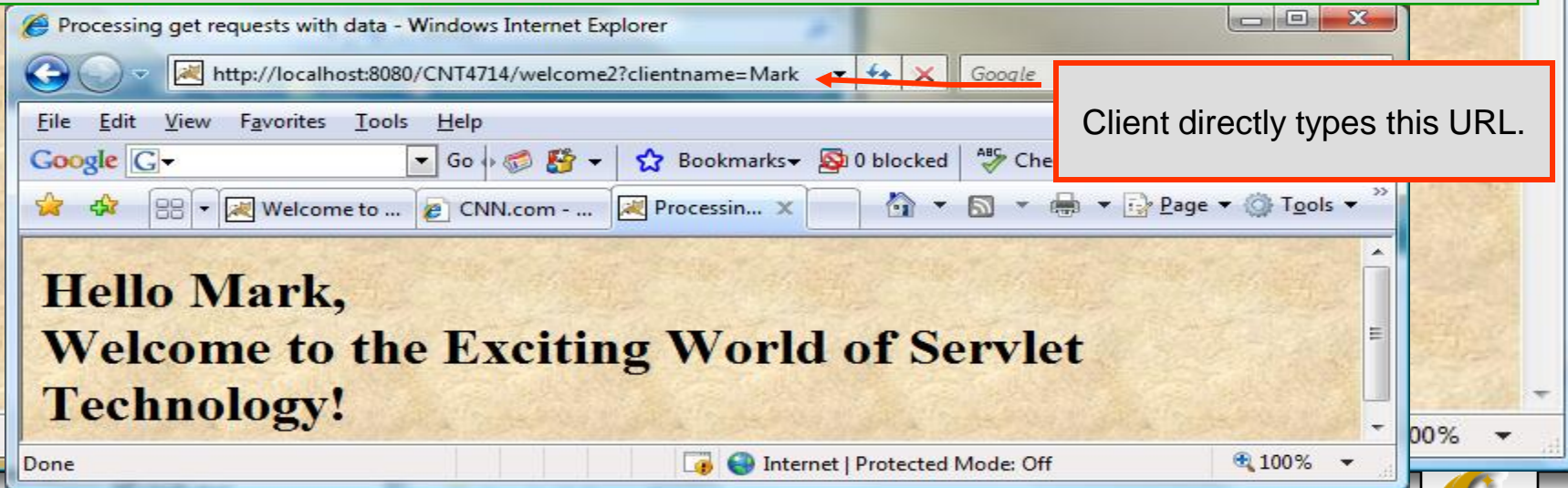
Form in WelcomeServlet2.html that specifies an input whose type is "text" and whose name is "clientname"







Note: The same servlet could have been invoked directly by typing in directly to the browsers Address or Location field. This is shown in the overlay below



Handling HTTP `post` Requests

- An HTTP `post` request is typically used to send data from an HTML form to a server-side form handler that processes the data. For example, when you respond to a Web-based survey, a `post` request normally supplies the information you entered into the form to the Web server.
- If you were to replace the `doGet` method in `WelcomeServlet2` with a `doPost` method, nothing would change in the apparent execution of the servlet with the exception that the values passed to the server are **not** appended to the request URL.
- This is illustrated by `WelcomeServlet3` which is exactly the same as `WelcomeServlet2` except that it uses the `doPost` method. Notice how the URL differs between the two versions.





WelcomeServlet2 uses the get method to supply the data to the form whereas WelcomeServlet3 uses the post method to do the same. Notice that the data is appended to the URL when the get method is used but it is not appended to the URL when the post method is used.



Modifications Necessary to `web.xml` File For Handling Additional Servlets

- In addition to modifying our `index.html` (homepage) file to include descriptors for launching the additional `WelcomeServlet2` and `WelcomeServlet3` servlets, we also need to modify the `web.xml` configuration file to register these servlets with Tomcat.
- We will need to include servlet definitions and servlet mappings for both `WelcomeServlet2` and `WelcomeServlet3`.
- The additional statements that must be included in this file are shown on the next slide.
- You must also include the Java class files for these servlets in the `classes` folder.



```
<servlet>
  <servlet-name>welcome2</servlet-name>

  <description>
    A more personal welcome servlet
  </description>

  <servlet-class>
    WelcomeServlet2
  </servlet-class>
</servlet>

<servlet>
  <servlet-name>welcome3</servlet-name>

  <description>
    A more personal welcome servlet - using a post action
  </description>

  <servlet-class>
    WelcomeServlet3
  </servlet-class>
</servlet>
```

Servlet descriptions

Servlet mappings

```
<servlet-mapping>
  <servlet-name>welcome2</servlet-name>
  <url-pattern>/welcome2</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>welcome3</servlet-name>
  <url-pattern>/welcome3</url-pattern>
</servlet-mapping>
```



Redirecting Requests to Other Resources

- Sometimes it is useful to redirect a request to a different resource. For example, a servlet's job might be to determine the type of the client's browser and redirect the request to a Web page that was designed specifically for that browser.
- The same technique is used when redirecting browsers to an error page when the handling of a request fails.
- Shown on the next two pages is the source code for a `ReDirectionServlet` (available on the course website) which redirects the client to another resource selected from a list of resources.



RedirectionServlet.java

```
// Redirecting a client to a different Web page.  
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;
```

```
public class ReDirectionServlet extends HttpServlet {  
    // process "get" request from client  
    protected void doGet( HttpServletRequest request, HttpServletResponse response )  
        throws ServletException, IOException  
    {  
        String location = request.getParameter( "page" );  
        if ( location != null )  
            if ( location.equals( "CNT 4714" ) )  
                response.sendRedirect( "http://www.cs.ucf.edu/courses/cnt4714/spr2009" );  
            else  
                if ( location.equals( "welcome1" ) )  
                    response.sendRedirect( "welcome1" );  
                else  
                    if ( location.equals ( "error" ) )  
                        response.sendRedirect( "error" );
```

sendRedirect is a method within the HttpServletResponse Interface. The string parameter is utilized as the URL to which the client's request is redirected.



```

// code that executes only if this servlet does not redirect the user to another page
response.setContentType( "text/html" );
PrintWriter out = response.getWriter();

// start XHTML document
out.println( "<?xml version = \"1.0\"?>" );
out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD \" +
    \"XHTML 1.0 Strict//EN\" \"http://www.w3.org\" + \"/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );

// head section of document
out.println( "<head>" );
out.println( "<title>Invalid page</title>" );
out.println( "</head>" );

// body section of document
out.println( "<body>" );
out.println( "<h1>Invalid page requested</h1>" );
out.println( "<p><a href = \" +
    \"\"RedirectionServlet.html\">" );
out.println( "Click here for more details</a></p>" );
out.println( "</body>" );

// end XHTML document
out.println( "</html>" );
out.close(); // close stream to complete the page
}
}

```



```

<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- ReDirectionServlet.html -->

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Redirecting a Request to Another Site</title>
</head>

<body>
<font size = 4><body bgcolor=white background=images/background.jpg lang=EN-US
link=blue vlink=blue style='tab-interval:.5in'>
<br>
<p><b>CLICK A LINK TO BE REDIRECTED TO THE APPROPRIATE RESOURCE</b></p>
<p>
<a href = "/CNT4714/redirect?page=CNT 4714">
www.cs.ucf.edu/courses/cnt4714/spr2009</a><p>
<a href = "/CNT4714/redirect?page=welcome1">
Original welcome servlet</a><p>
<a href="/CNT4714/redirect?page=error">
Intentional error - redirected page does not exist</a><p>
</p>
</body>
</html>

```

ReDirectionServlet.html



The servlet and servlet-mapping Portions Of web.xml Modified To Handle The ReDirectionServlet

```
<servlet>
  <servlet-name>redirect</servlet-name>
  <description>
    A redirection servlet.
  </description>
  <servlet-class>
    ReDirectionServlet
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>redirect</servlet-name>
  <url-pattern>/redirect</url-pattern>
</servlet-mapping>
```



