COP 4610L: Applications in the Enterprise Spring 2005

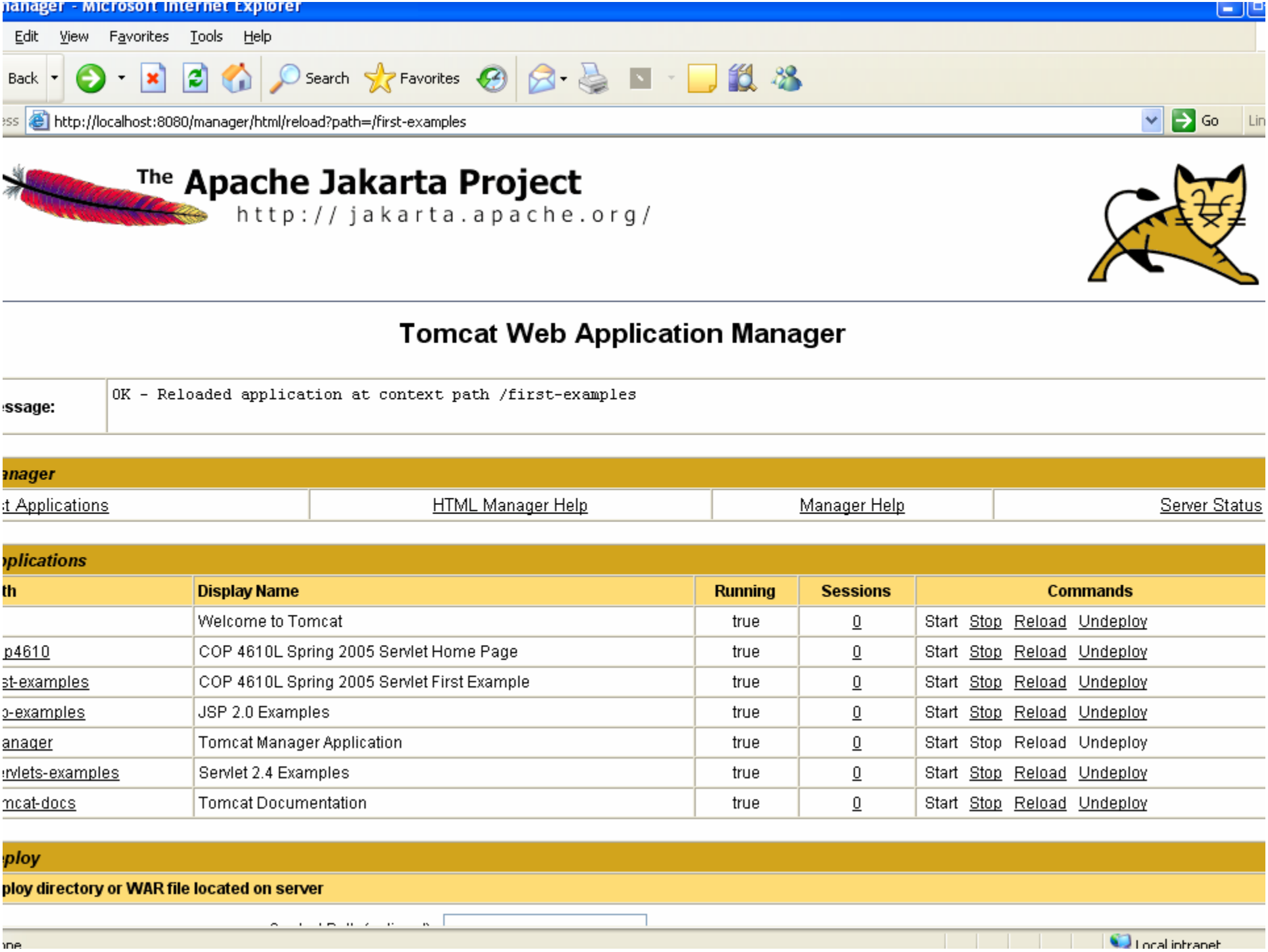
Introduction to Servlet Technology – Part 2

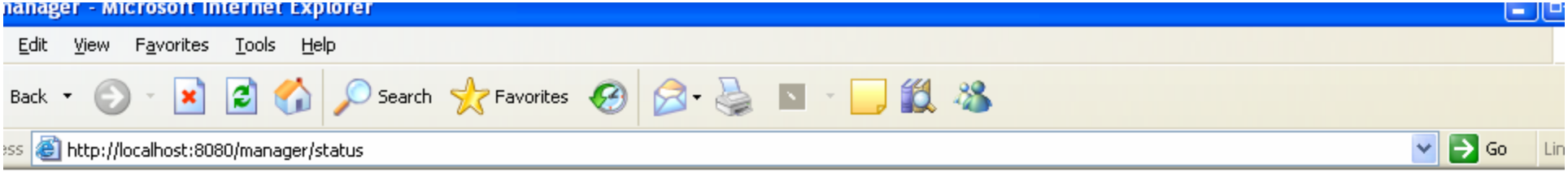
Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4610L/spr2005

School of Electrical Engineering and Computer Science
University of Central Florida

More Tomcat Details

- If your system does not recognize "localhost", enter http://127.0.0.1:8080 instead of http://localhost:8080. Address 127.0.0.1 basically means "this machine" which is the same as localhost.
- From the Tomcat homepage you can also act as the server administrator and manager. While we won't need to do anything on the administrator side (you'll need to download the administrator web application separately from apache), it is interesting to go into the manager side of things and look at the server from the server's point of view. It may also be necessary to reload applications occasionally (more on this later), which can be done from the manager application. See page 3 for an example.
- Checking the status of the server can also be accomplished from the Tomcat homepage. See page 4 for a sample.


anager - Microsoft Internet Explorer
Edit View Favorites Tools Help
Back Search Favorites
http://localhost:8080/manager/html/reload?path=/first-examples
Go
The Apache Jakarta Project
http://jakarta.apache.org/
Tomcat Web Application Manager
essage:
OK - Reloaded application at context path /first-examples
anager
t Applications
HTML Manager Help
Manager Help
Server Status
plications
th
Display Name
Running
Sessions
Commands
Welcome to Tomcat
true
0
Start Stop Reload Undeploy
p4610
COP 4610L Spring 2005 Servlet Home Page
true
0
Start Stop Reload Undeploy
st-examples
COP 4610L Spring 2005 Servlet First Example
true
0
Start Stop Reload Undeploy
p-examples
JSP 2.0 Examples
true
0
Start Stop Reload Undeploy
anager
Tomcat Manager Application
true
0
Start Stop Reload Undeploy
rvlets-examples
Servlet 2.4 Examples
true
0
Start Stop Reload Undeploy
mcat-docs
Tomcat Documentation
true
0
Start Stop Reload Undeploy
ploy
ploy directory or WAR file located on server
Local intranet



The Apache Jakarta Project
http://jakarta.apache.org/

Server Status

anager

t Applications	HTML Manager Help	Manager Help	Complete Server Status

rver Information

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture
Apache Tomcat/5.5.7	1.5.0-b64	Sun Microsystems Inc.	Windows XP	5.1	x86

VM

e memory: 0.12 MB Total memory: 4.69 MB Max memory: 63.56 MB

tp-8080

threads: 150 Min spare threads: 25 Max spare threads: 75 Current thread count: 25 Current thread busy: 2
processing time: 303 ms Processing time: 0 s Request count: 16 Error count: 2 Bytes received: 0.00 MB Bytes sent: 0.08 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
S	37 ms	0 KB	0 KB	127.0.0.1	localhost	GET /manager/status HTTP/1.1
R	?	?	?	?	?	?
R	?	?	?	?	?	?

A Tour of Tomcat

- Before we look into creating our own servlets, we need to look more closely at Tomcat. This will help you better understand how web applications are developed and deployed.

- The directory structure within Tomcat looks like the one shown on the next page. It contains, among other things, nine directories named, `bin`, `conf`, `logs`, `common`, `server`, `shared`, `webapps`, `work`, and `temp`.

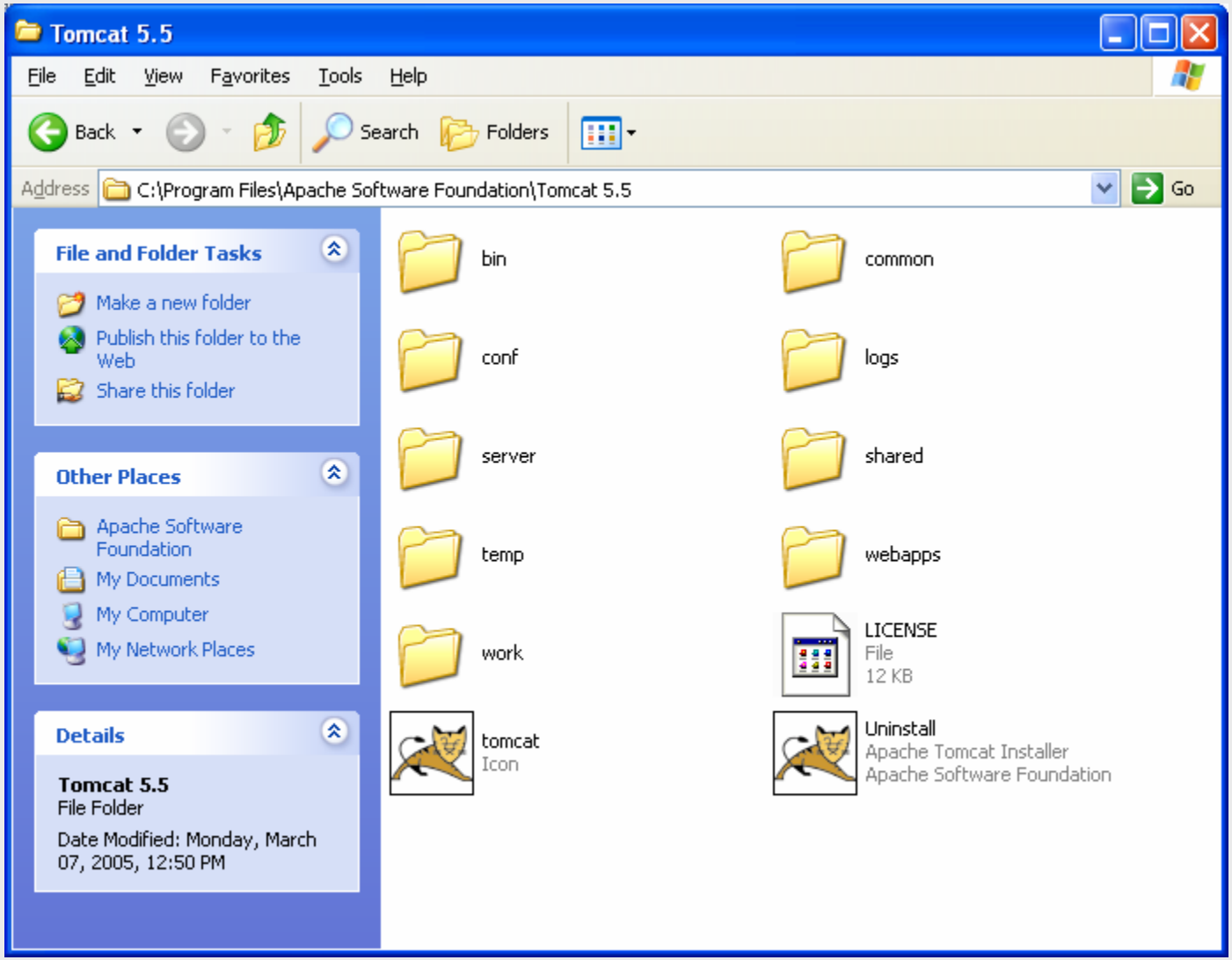
`bin`

- Directory `bin` contains scripts for starting and stopping Tomcat as well as some additional tools.

`conf`

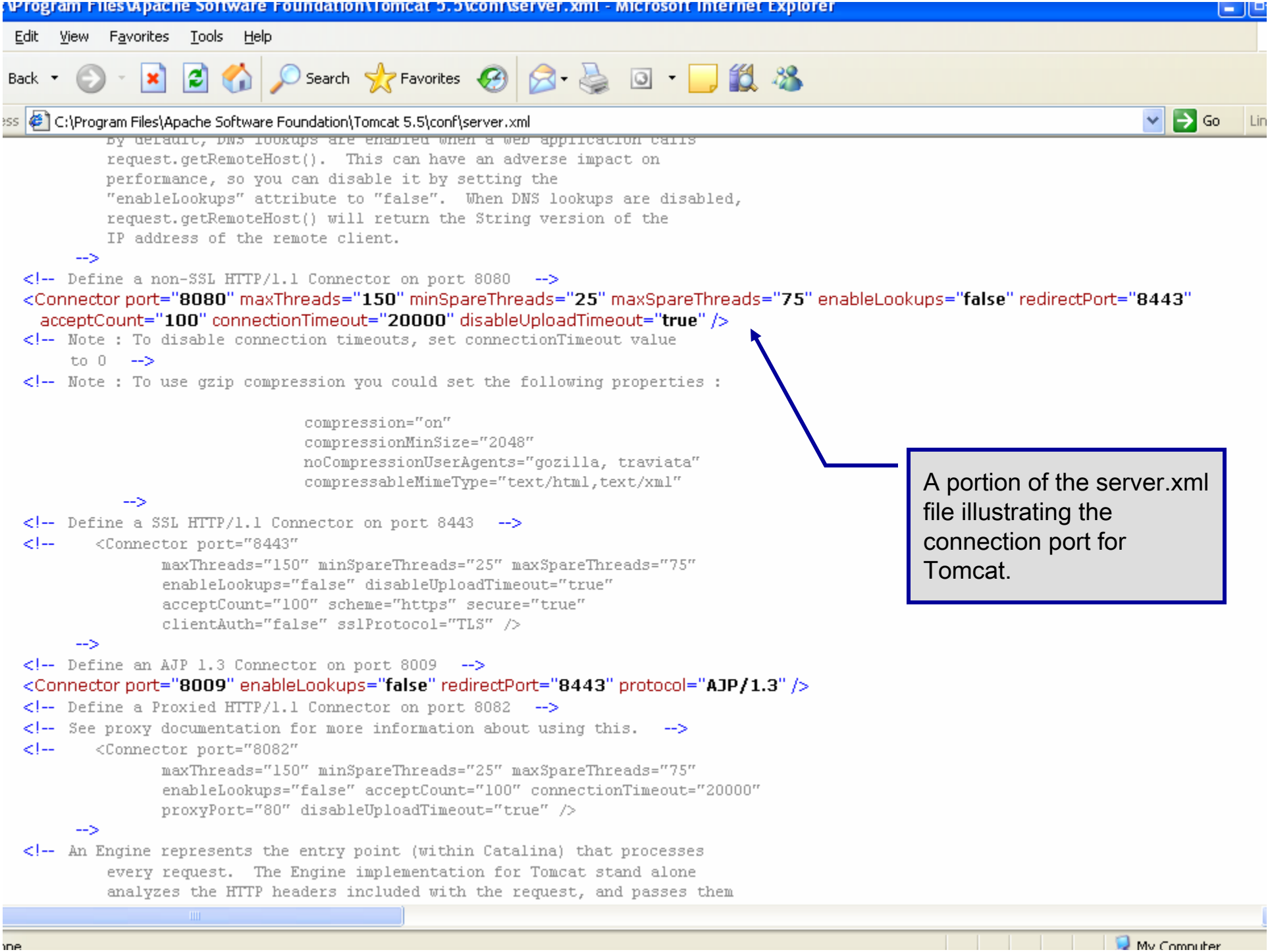
- Directory `conf` contains files used to configure Tomcat at the global level, although it is possible for each web application to override many of the values provided in this directory.

Tomcat Directory Structure

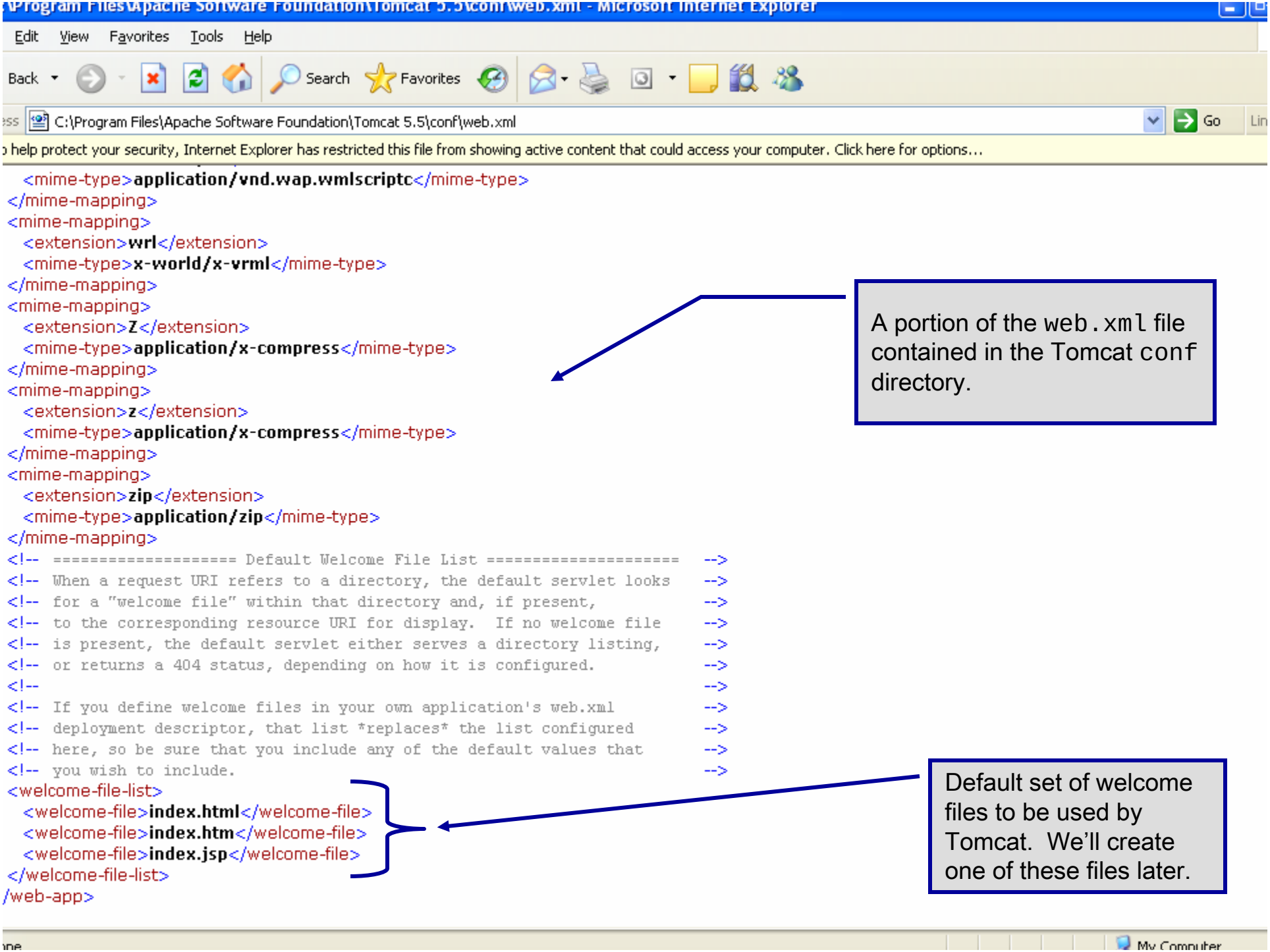


A Tour of Tomcat (cont.)

- The most important file inside the `conf` directory is `server.xml`, which tells Tomcat the set of services to run when it starts up as well as what port to listen to. This file also specifies the set of resources to make available to applications and a number of security parameters. A portion of this file (the part illustrating the non-SSL HTTP port) is shown on page 8.
- There is also a `web.xml` file in this directory, which establishes default values that may be overridden by values in each applications `web.xml` file. A portion of this file is shown on page 9.
- The file `jk2.properties` defines a set of properties that are used when Tomcat is installed as an application server in conjunction with an external web server such as Apache or IIS. In these notes we will assume that Tomcat is running in stand-alone mode, where it operates as both a web server and application server.



A portion of the server.xml file illustrating the connection port for Tomcat.


Program Files\Apache Software Foundation\Tomcat 5.5\conf\web.xml - Microsoft Internet Explorer
Edit View Favorites Tools Help
Back Search Favorites
C:\Program Files\Apache Software Foundation\Tomcat 5.5\conf\web.xml
Go
help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...
<mime-type>application/vnd.wap.wmlscriptc</mime-type>
</mime-mapping>
<mime-mapping>
<extension>wrl</extension>
<mime-type>x-world/x-vrml</mime-type>
</mime-mapping>
<mime-mapping>
<extension>Z</extension>
<mime-type>application/x-compress</mime-type>
</mime-mapping>
<mime-mapping>
<extension>z</extension>
<mime-type>application/x-compress</mime-type>
</mime-mapping>
<mime-mapping>
<extension>zip</extension>
<mime-type>application/zip</mime-type>
</mime-mapping>
<!-- ==================== Default Welcome File List ==================== -->
<!-- When a request URI refers to a directory, the default servlet looks -->
<!-- for a "welcome file" within that directory and, if present, -->
<!-- to the corresponding resource URI for display. If no welcome file -->
<!-- is present, the default servlet either serves a directory listing, -->
<!-- or returns a 404 status, depending on how it is configured. -->
<!-- -->
<!-- If you define welcome files in your own application's web.xml -->
<!-- deployment descriptor, that list *replaces* the list configured -->
<!-- here, so be sure that you include any of the default values that -->
<!-- you wish to include. -->
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
/web-app>
A portion of the web.xml file contained in the Tomcat conf directory.
Default set of welcome files to be used by Tomcat. We'll create one of these files later.
My Computer

A Tour of Tomcat (cont.)

logs

- The logs directory contains a n umber of log files created by Tomcat. The file `catalina.out` contains anything written to `System.out` and `System.err`, as well as information relevant to the server as a whole.

common

- This directory contains three subdirectories – classes, lib, and endorsed – which contain code used by Tomcat. The classes directory is effectively added to the CLASSPATH, as are any jar files in lib and endorsed. Any custom jar files that may be needed throughout Tomcat, such as a JDBC driver, are placed in these directories.

A Tour of Tomcat (cont.)

server

- This directory contains three subdirectories – classes, lib, and endorsed – which contain code used by Tomcat. The classes directory is effectively added to the CLASSPATH, as are any jar files in lib and endorsed. Any custom jar files that may be needed throughout Tomcat, such as a JDBC driver, are placed in these directories.

shared

- This is another directory containing classes and lib subdirectories. The classes and jars within these directories are available to all web applications but will not be available to the server infrastructure.

A Tour of Tomcat (cont.)

`webapps`

- This directory contains all the web applications Tomcat is configured to run, one web application per subdirectory. We will be placing the web applications that we develop into subdirectories in this directory. We'll look in more detail at the structure of these subdirectories a bit later.

`work`

- This directory is used by Tomcat to hold servlets that are built from JSP pages. Users will typically not need anything in this directory.

`temp`

- This directory is used internally by Tomcat and can be ignored.

Servlet Interface

- The servlet packages define two abstract classes that implement interface `Servlet` – class `GenericServlet` (from the package `javax.servlet`) and class `HttpServlet` (from the package `javax.servlet.http`).
- These classes provide default implementations of some `Servlet` methods.
- Most servlets extend either `GenericServlet` or `HttpServlet` and override some or all of their methods.
- The `GenericServlet` is a protocol-indpendent servlet, while the `HttpServlet` uses the HTTP protocol to exchange information between the client and server.
- We're going to focus exclusively on the `HttpServlet` used on the Web.

Servlet Interface (cont.)

- `HttpServlet` defines enhanced processing capabilities for services that extend a Web server's functionality.
- The key method in every servlet is `service`, which accepts both a `ServletRequest` object and a `ServletResponse` object. These object provide access to input and output streams that allow the servlet to read data from and send data to the client.
- If a problem occurs during the execution of a servlet, either `ServletExceptions` or `IOExceptions` are thrown to indicate the problem.

HTTPServlet Class

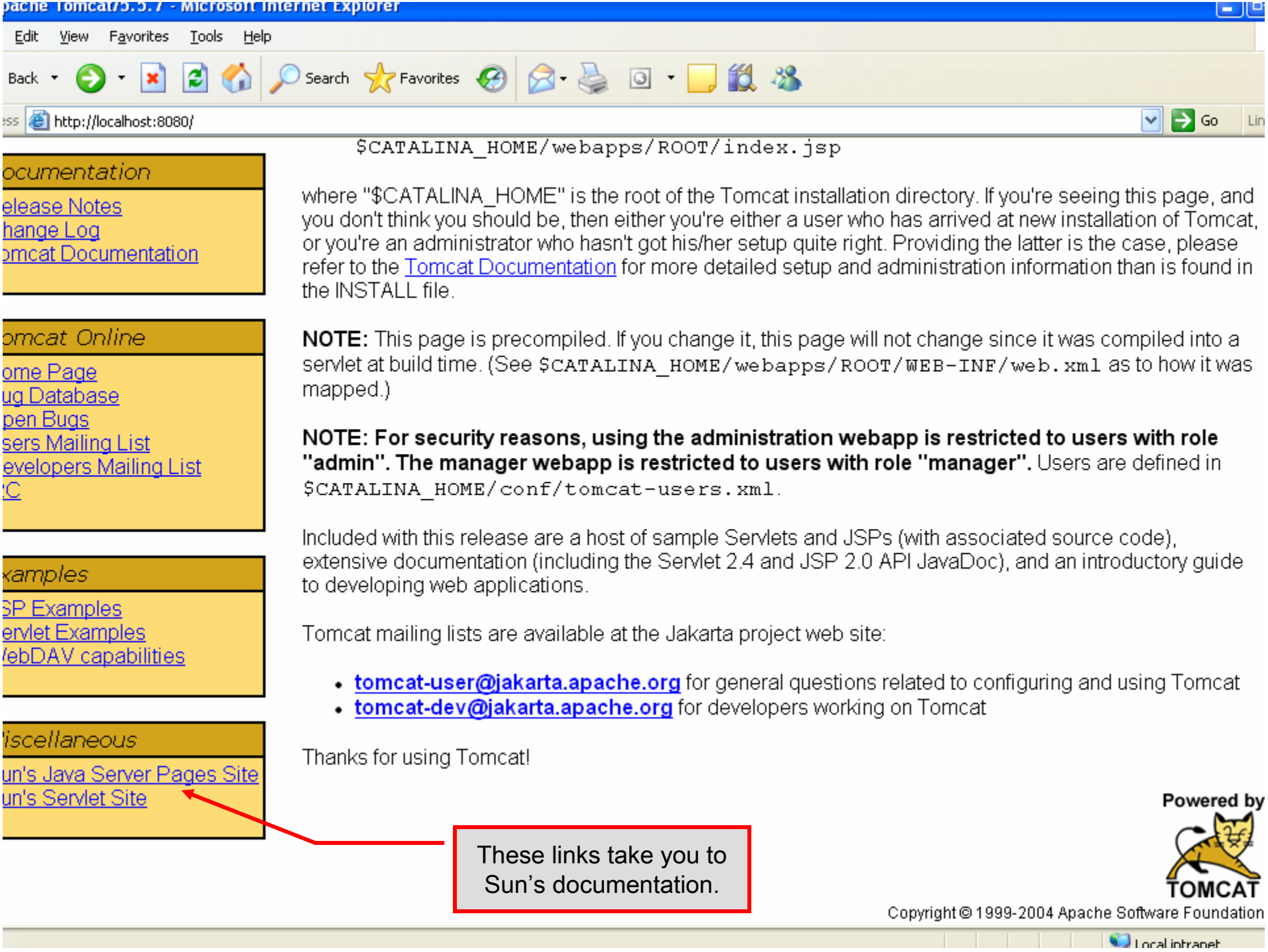
- Servlets typically extend class `HttpServlet`, which overrides method `service` to distinguish between the various requests received from a client web browser.
- The two most common HTTP request types (also known as request methods) are `get` and `post`. (See also Servlets – Part 1 notes.)
  - A get request retrieves information from a server. Typically, an HTML document or image.
  - A post request sends data to a server. Typically, post requests are used to pass user input to a data-handling process, store or update data on a server, or post a message to a news group or discussion forum.
- Class `HttpServlet` defines methods `doGet` and `doPost` to respond to get and post requests from a client.

HTTPServlet Class (cont.)

- Methods `doGet` and `doPost` are invoked by method `service`, which is invoked by the servlet container when a request arrives at the server.
- Method service first determines the request type, the invokes the appropriate method for handling such a request.
- In addition to methods `doGet` and `doPost`, the following methods are defined in class `HttpServlet`:
  - `doDelete` (typically deletes a file from the server)
  - `doHead` (client wants only response headers no entire body)
  - `doOptions` (returns HTTP options supported by server)
  - `doPut` (typically stores a file on the server)
  - `doTrace` (for debugging purposes)

HTTPServletRequest Interface

- Every invocation of `doGet` or `doPost` for an `HttpServlet` receives an object that implements interface `HttpServletRequest`.
- The servlet container creates an `HttpServletRequest` object and passes it to the servlet's `service` method, which in turn, passes it to `doGet` or `doPost`.
- This object contains the clients' request and provides methods that enable the servlet to process the request.
- The full list of HttpServletRequest methods is available at: www.java.sun.com/j2ee/1.4/docs/api/index.html, however, a few of the more common ones are shown on page 11. (Note: you can also get to them from Tomcat, see next page.)


Edit View Favorites Tools Help
Back Search Favorites
http://localhost:8080/
Go
ocumentation
elease Notes
hange Log
omcat Documentation
omcat Online
ome Page
ug Database
pen Bugs
sers Mailing List
evelopers Mailing List
C
xamples
SP Examples
ervlet Examples
ebDAV capabilities
iscellaneous
un's Java Server Pages Site
un's Servlet Site
$CATALINA_HOME/webapps/ROOT/index.jsp
where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.
NOTE: This page is precompiled. If you change it, this page will not change since it was compiled into a servlet at build time. (See $CATALINA_HOME/webapps/ROOT/WEB-INF/web.xml as to how it was mapped.)
NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in $CATALINA_HOME/conf/tomcat-users.xml.
Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation (including the Servlet 2.4 and JSP 2.0 API JavaDoc), and an introductory guide to developing web applications.
Tomcat mailing lists are available at the Jakarta project web site:
tomcat-user@jakarta.apache.org for general questions related to configuring and using Tomcat
tomcat-dev@jakarta.apache.org for developers working on Tomcat
Thanks for using Tomcat!
Powered by
TOMCAT
Copyright © 1999-2004 Apache Software Foundation
These links take you to Sun's documentation.

HTTPServletRequest Methods

- `Cookie[] getCookies()` – returns an array of Cookie objects stored on the client by the server. Cookies are used to uniquely identify clients to the server.
- `String getLocalName()` – gets the host name on which the request was received.
- `String getLocalAddr()` – gets the IP address on which the request was received.
- `int getLocalPort()` – gets the IP port number on which the request was received.
- `String getParameter( String name)` – gets the value of a parameter set to the servlet as part of a `get` or `post` request.

HTTPServletResponse Interface

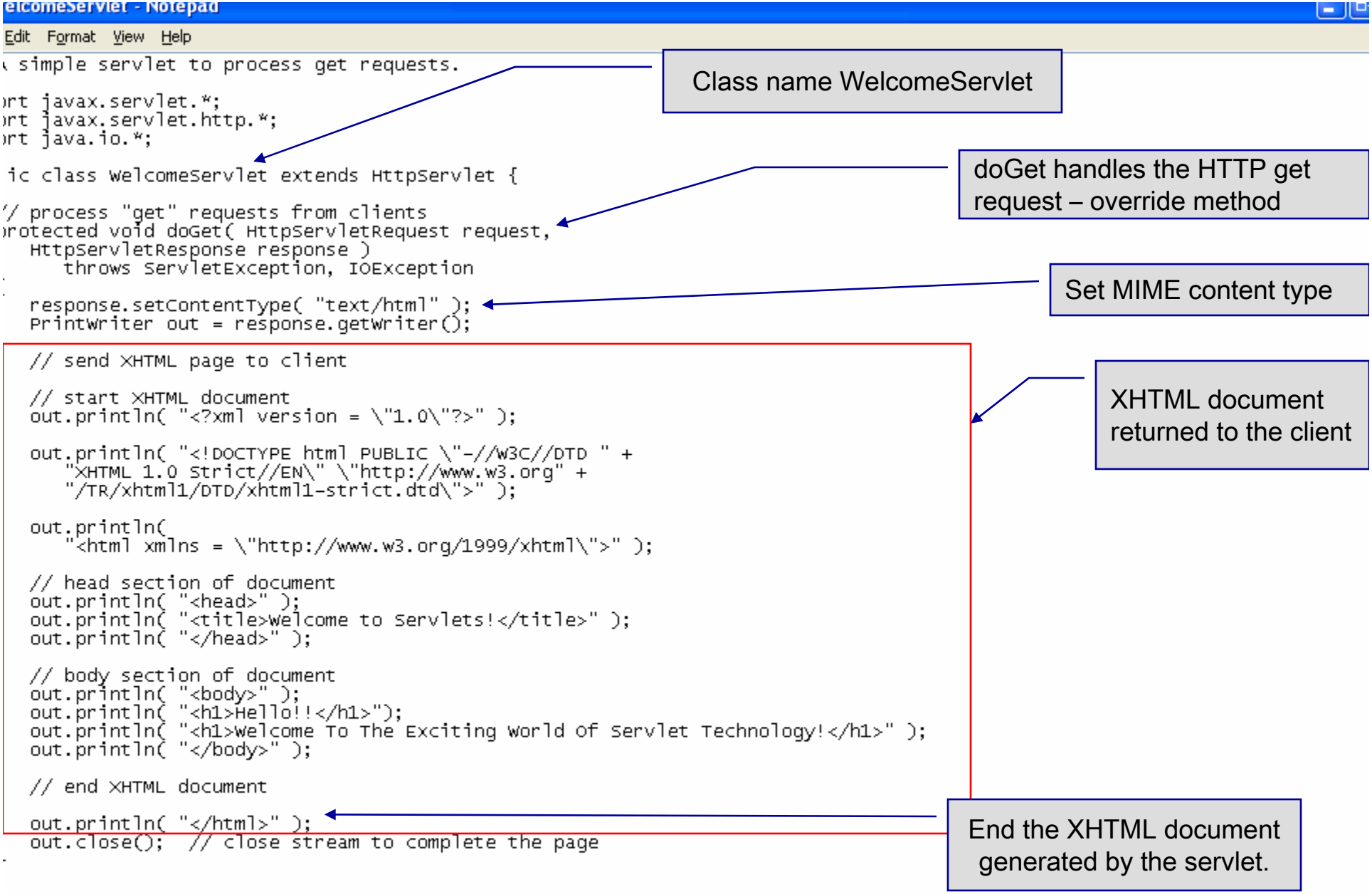
- Every invocation of `doGet` or `doPost` for an `HttpServlet` receives an object that implements interface `HttpServletResponse`.
- The servlet container creates an `HttpServletResponse` object and passes it to the servlet's `service` method, which in turn, passes it to `doGet` or `doPost`.
- This object provides methods that enable the servlet to formulate the response to the client.
- The full list of HttpServletRequest methods is available at: www.java.sun.com/j2ee/1.4/docs/api/index.html, however, a few of the more common ones are shown on the next page. (Also accessible from Tomcat.)

HTTPServletResponse Methods

- `void addCookie (Cookie cookie)` – adds a Cookie to the header of the response to the client.
- `ServletOutputStream getOutputStream()` – gets a byte-based output stream for sending binary data to the client.
- `PrintWriter getWriter()` – gets a character-based output stream for sending text data (typically HTML formatted text) to the client.
- `void SetContentType (String type)` – specifies the content type of the response to the browser to assist in displaying the data.
- `void getContentType()` – gets the content type of the response.

Handling HTTP `get` Requests

- The primary purpose of an HTTP `get` request is to retrieve the contents of a specified URL, which is typically an HTML or XHTML document.
- Before we look at a complete implementation of a servlet execution, let's examine the Java code that is required for a basic servlet.
- Shown on the next page is a servlet that responds to an HTTP `get` request. This is a simple welcome servlet and is about as simple a servlet as is possible.
- Note: Tomcat will look for an index.html, or welcome.html files to run as a default "home page". At this point we haven't set one up so the initial screen for our web application will not be too pretty.


elcomeServlet - Notepad
Edit Format View Help
\ simple servlet to process get requests.
ort javax.servlet.*;
ort javax.servlet.http.*;
ort java.io.*;
ic class WelcomeServlet extends HttpServlet {
// process "get" requests from clients
rotected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
response.setContentType("text/html");
PrintWriter out = response.getWriter();
// send XHTML page 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\">");
// 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>Hello!!</h1>");
out.println("<h1>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
Class name WelcomeServlet
doGet handles the HTTP get request – override method
Set MIME content type
XHTML document returned to the client
End the XHTML document generated by the servlet.

Handling HTTP `get` Requests (cont.)

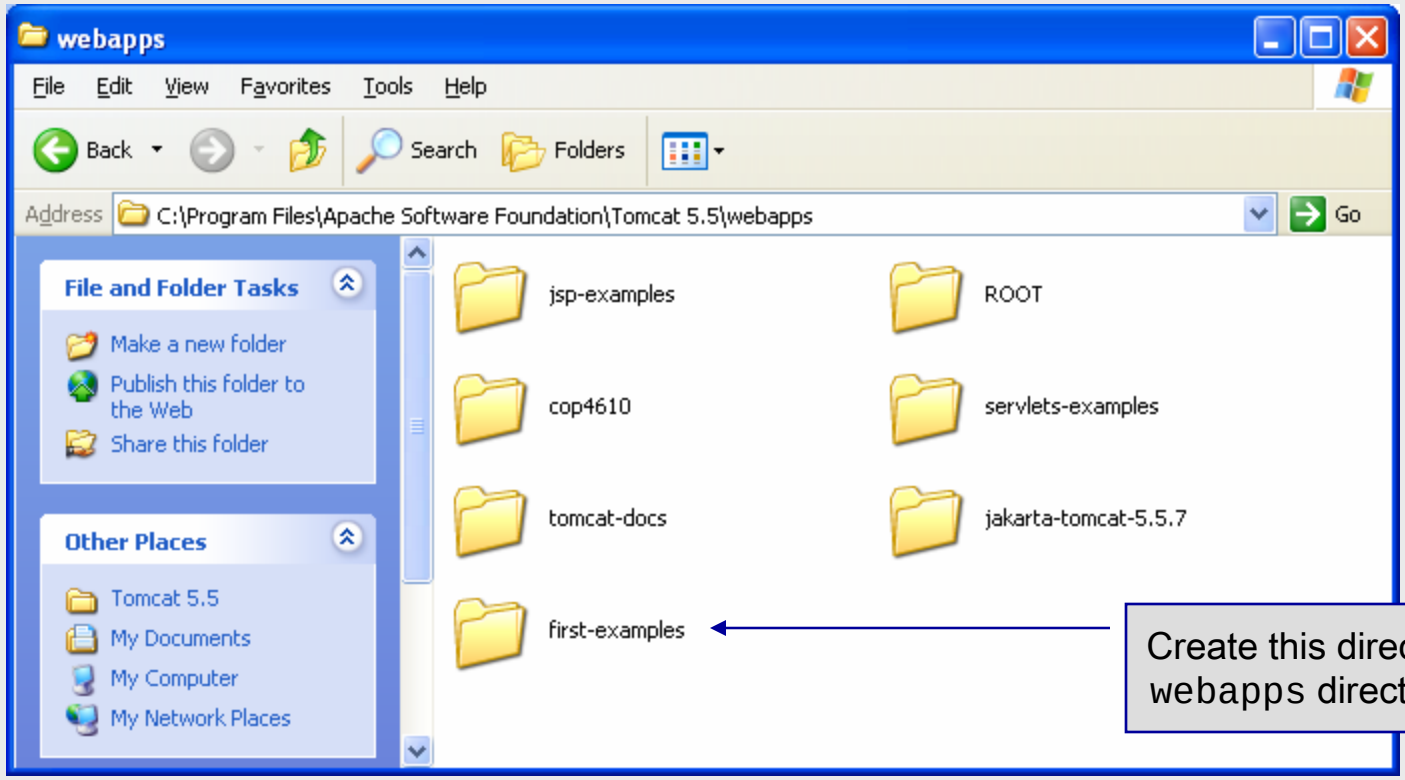
- The servlet creates an XHTML document containing the text "Hello! Welcome to the Exciting World of Servlet Technology!"
- This text is the response to the client and is sent through the `PrintWriter` object obtained from the `HttpServletRepsonse` object.
- The response object's setContentType method is used to specify the type of data to be sent as the response to the client. In this case it is defined as text/html, we'll look at other types later. In this case the browser knows that it must read the XHTML tags and format the document accordingly.
- The content type is also known as the MIME (Multipurpose Internet Mail Extension) type of the data.

Creating a Web Application

- One of the fundamental ideas behind Tomcat is that of a web application.
- A web application is a collection of pages, code, and configurations that is treated as a unit.
- Normally a web application will map to a particular URL, so URLs such as http://somesite.com/app1 and http://somesite.com/app2 will invoke different web applications called `app1` and `app2` respectively.
- Tomcat can contain an arbitrary number of web applications simultaneously.
- While web applications can be extremely complex, we'll start out with a minimal web application and build from there.

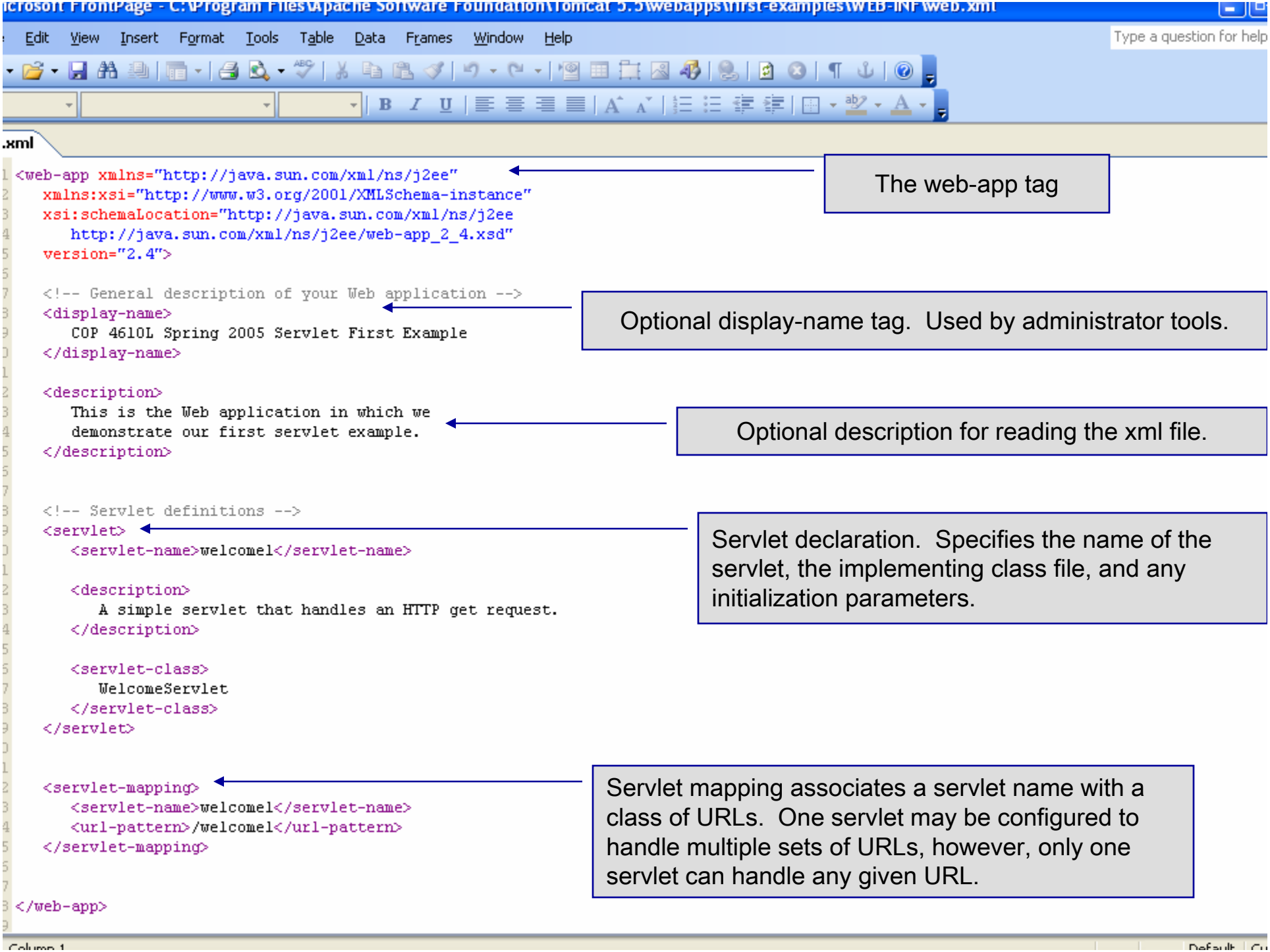
Creating a Web Application (cont.)

- The most basic web application in Tomcat will require the creation of a directory inside the `webapps` directory to hold the web application. For this first example, we'll create a subdirectory called `first-examples`.



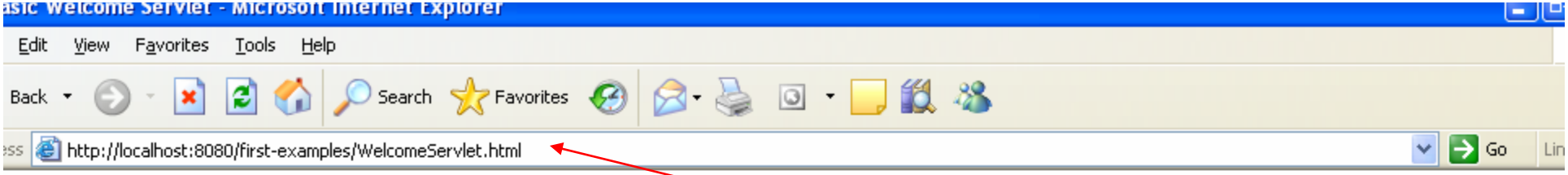
Creating a Web Application (cont.)

- Within the `first-examples` directory we need to create a directory that will hold the configuration and all of the resources for the web application. This directory must be called `WEB-INF`.
- The most important and only required element in `WEB-INF` is the file `web.xml`. The `web.xml` file controls everything specific to the current web application. We'll look at this file in more detail later as we add to it, but for now we'll look only at the components of this file that are essential for a very simple web application.
- The next page illustrates our initial `web.xml` file.

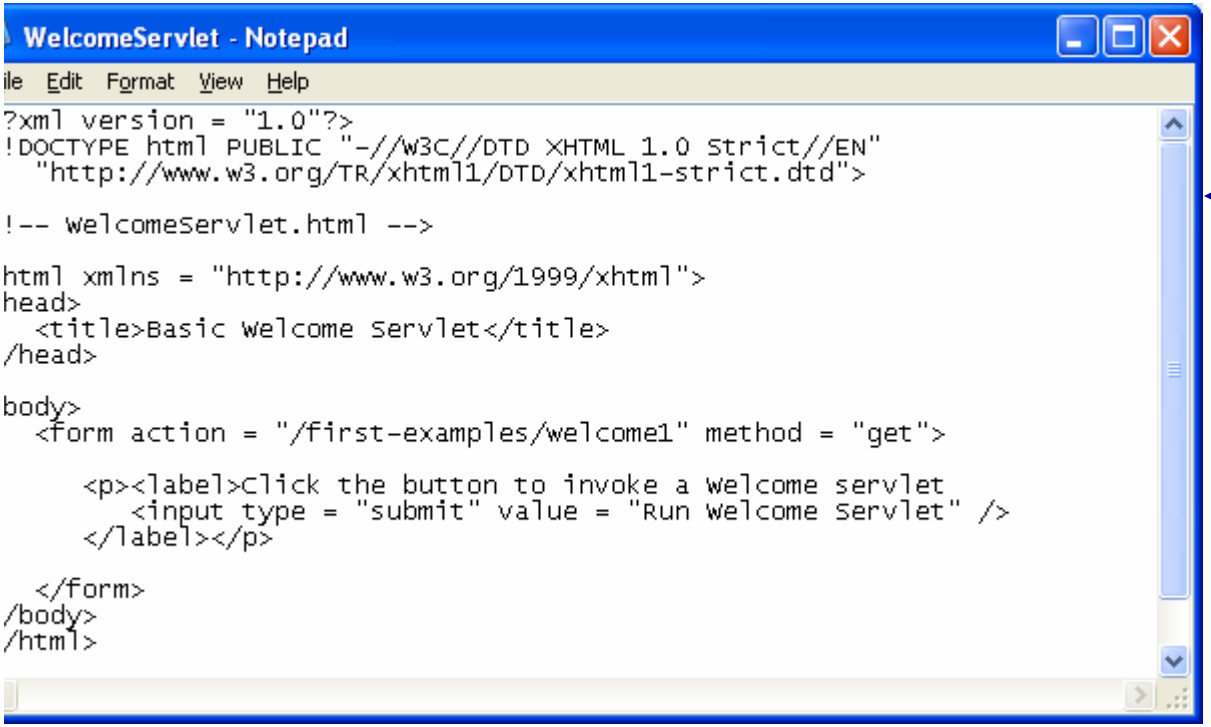

Microsoft FrontPage - C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\first-examples\WEB-INF\web.xml
Edit View Insert Format Tools Table Data Frames Window Help
Type a question for help
.xml
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
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>
COP 4610L Spring 2005 Servlet First Example
</display-name>
<description>
This is the Web application in which we
demonstrate our first servlet example.
</description>
<!-- Servlet definitions -->
<servlet>
<servlet-name>welcome1</servlet-name>
<description>
A simple servlet that handles an HTTP get request.
</description>
<servlet-class>
WelcomeServlet
</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>welcome1</servlet-name>
<url-pattern>/welcome1</url-pattern>
</servlet-mapping>
</web-app>
The web-app tag
Optional display-name tag. Used by administrator tools.
Optional description for reading the xml file.
Servlet declaration. Specifies the name of the servlet, the implementing class file, and any initialization parameters.
Servlet mapping associates a servlet name with a class of URLs. One servlet may be configured to handle multiple sets of URLs, however, only one servlet can handle any given URL.

Creating a Web Application (cont.)

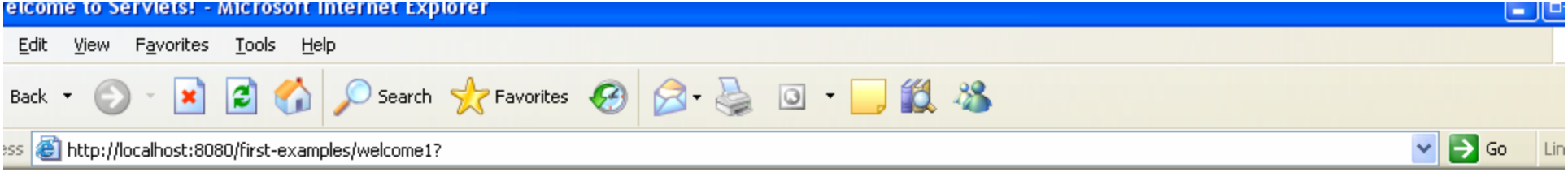
- With these directories and files in place, Tomcat will be able to respond to a request for the page from a client at http://localhost:8080/first-examples/WelcomeServlet.html.
- Other HTML and JSP pages can be added at will, along with images, MP3 files, and just about anything else.
- Although what we have just seen is all that is required to create a minimal web application, much more is possible with a knowledge of how web applications are arranged and we will see this as we progress through this technology.
- The next few slides illustrate the execution of our simple web application (a welcome servlet).



Client invokes the WelcomeServlet page from the web application named first-examples.



This is the XHTML file that generates the output shown above which informs the client how to invoke the servlet.



ello!!

/elcome To The Exciting World Of Servlet Technology!

Execution of the WelcomeServlet servlet

An XHTML Document

- The XHTML document shown on page 30 provides a `form` that invokes the servlet defined on page 23.
- The form's `action` attribute (/first-examples/welcome1) specifies the URL path that invokes the servlet.
- The form's method attribute indicates that the browser sends a get request to the server, which results in a call to the servlet's `doGet` method.
  - We'll look at how to set-up the URL's and deployment structure in the next set of notes.

Set-Up For First Web Application

- The exact set-up you need to use for setting up your web application in Tomcat is summarized on the next couple of pages.

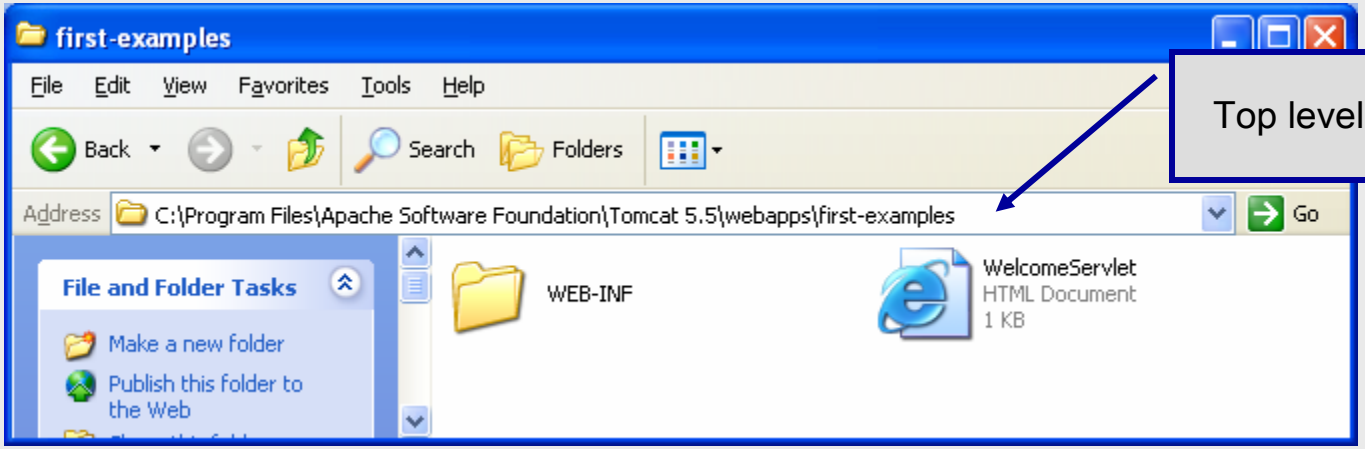
1. In the Tomcat `webapps` folder create a directory named `first-examples`.

2. In the top level of `first-examples` copy the `WelcomeServlet.html` file from the course code page.

3. In the top level of `first-examples` create a directory named `WEB-INF`.

4. When steps 2 and 3 are complete the top level of first-examples should look like the picture at the top of the next page.

Set-Up For First Web Application (cont.)

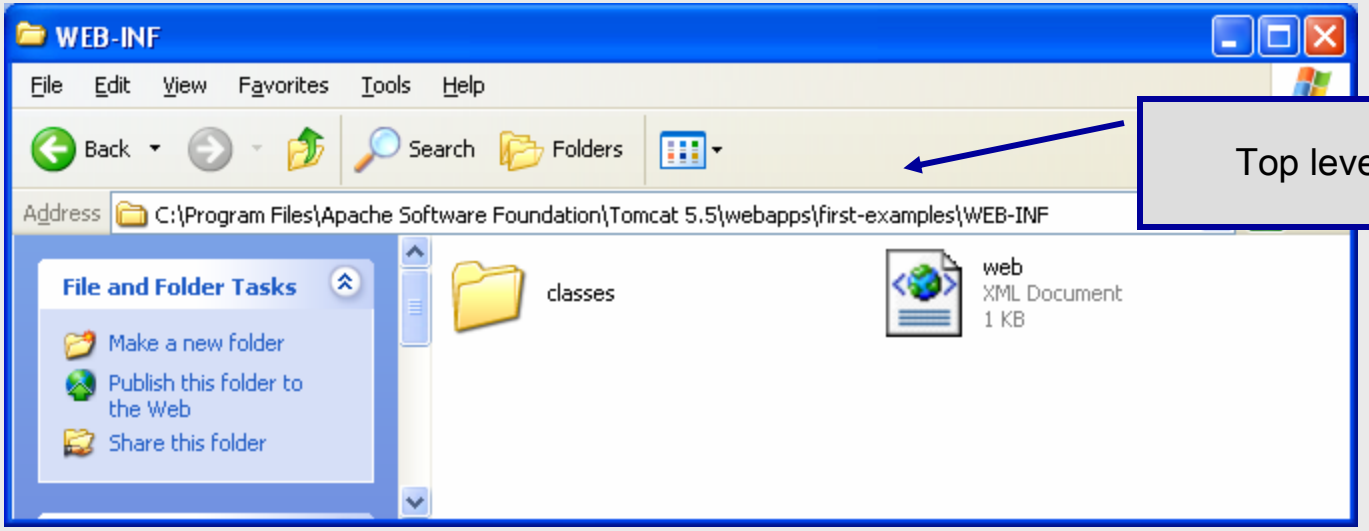


5. Copy the `web.xml` configuration file from the course code page into the `WEB-INF` directory.

6. At the top level of the `WEB-INF` directory create a directory named `classes`.

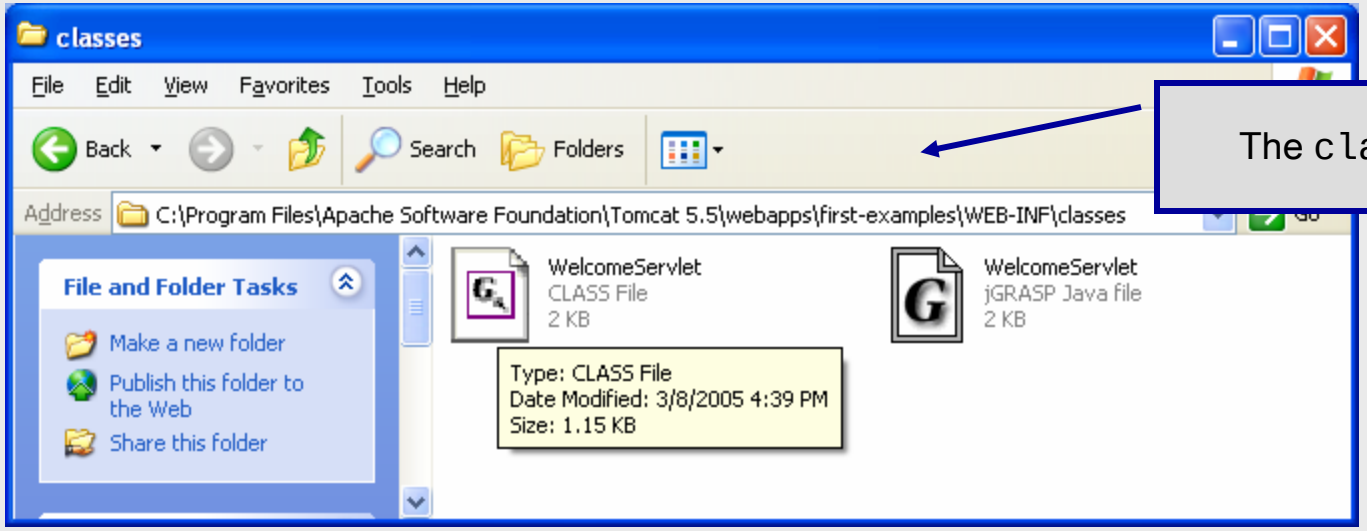
7. When steps 5 and 6 are complete, the WEB-INF directory should look like the picture on the top of the next page.

Set-Up For First Web Application (cont.)



8. Copy the `WelcomeServlet.java` file from the course code page into the `classes` directory and compile it to produce the `WelcomeServlet.class` file which should also reside in the classes directory. (The `.java` file does not need to reside in this directory for a servlet, but it is handy to keep the source in the same place.)

Set-Up For First Web Application (cont.)



9. Once the `classes` directory looks like the one shown above. You are ready to invoke the servlet from a web browser. Start Tomcat and enter the URL http://localhost:8080/WelcomeServlet.html. Tomcat and the servlet will do the rest. If all goes well you should see the output that was shown on page 31.