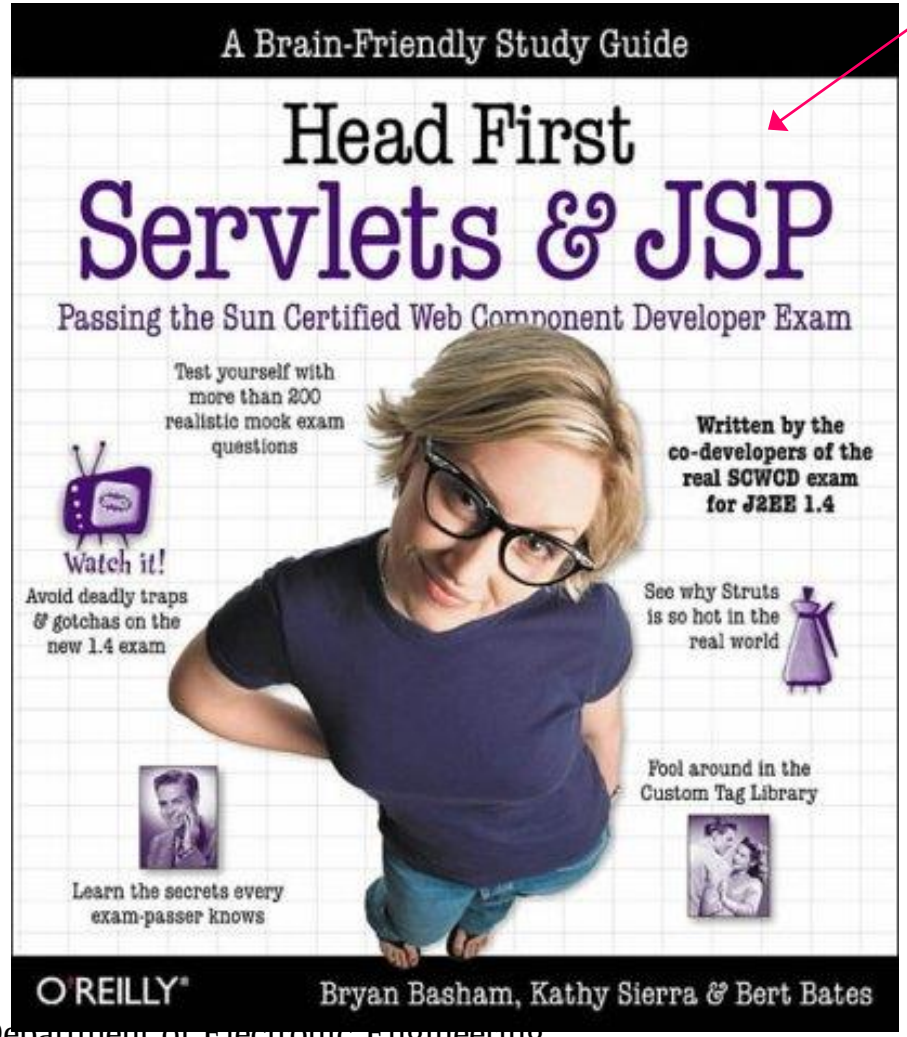
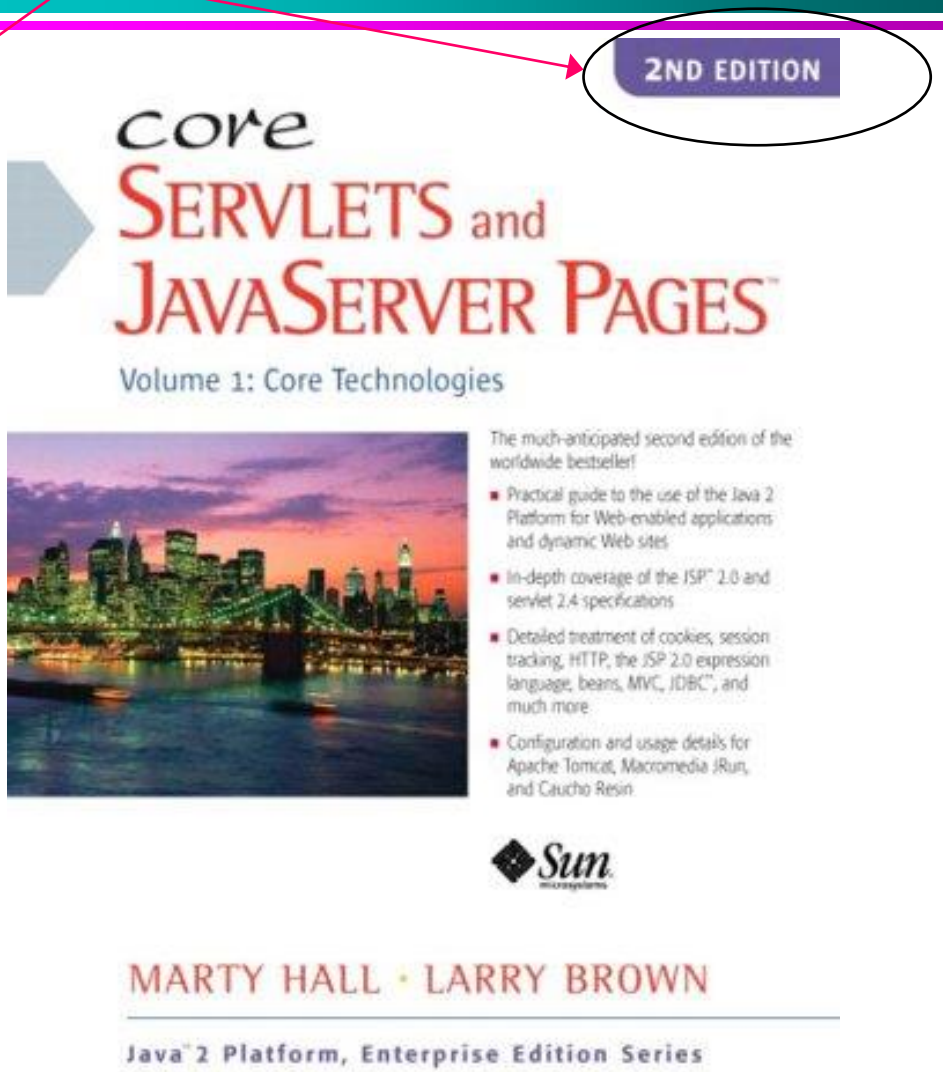
Middleware: Suggested reading



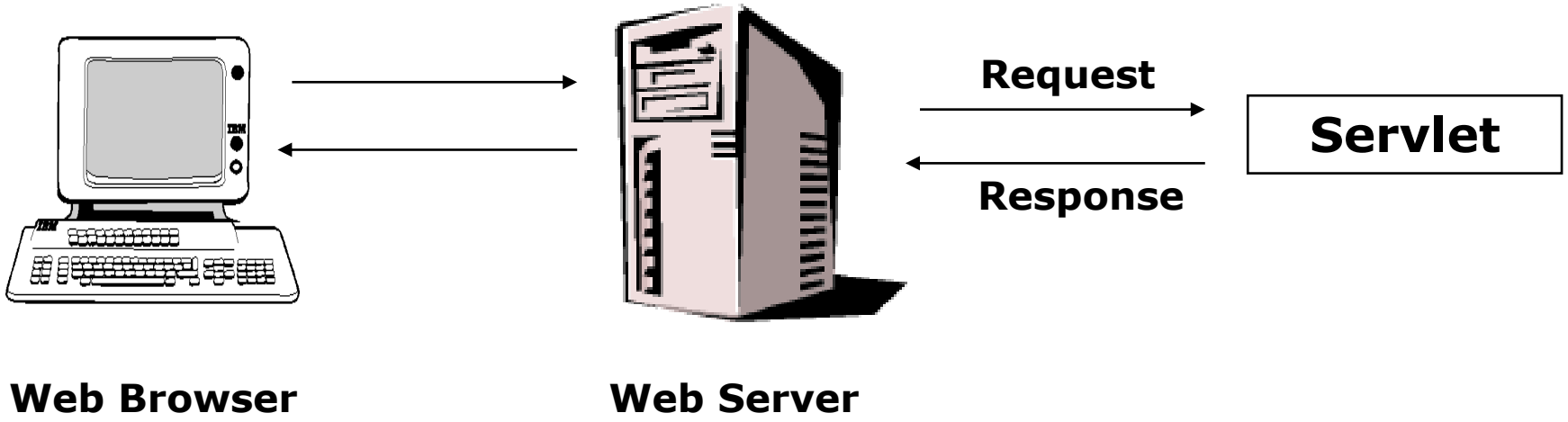


Java Servlets and the HTTP Protocol

- What are servlets and why are they useful?
- Basic servlet structure and lifecycle
- Handling request data
- HTTP request headers
- Generating the HTTP response
- Session tracking

What are servlets?

- Servlets were Java's answer to CGI (Common Gateway Interface.
  - Are programs that run on web server acting as middle layer between HTTP request and databases or other applications.
- Used for client requests that cannot be satisfied using pre-built (static) documents.
  - Used to generate dynamic web pages in response to client.



Beans, JSP and Servlets

- Although a servlet *can* be a completely self-contained program, to ease server-side programming, generating content should be split into:
  - The business logic (content generation), which governs the relationship between input, processing, and output
  - The presentation logic (content presentation, or graphic design rules), which determines how information is presented to the user
- Typically,
  - the servlet handles the HTTP protocol and coordination (controller)
  - Java Server Pages the presentation logic (view)
  - Java classes/ beans the business logic (model) …used by the above

Why are dynamic pages useful?

- Reasons for generating web pages on-the-fly include:
  - The web page is based on data submitted by the user, or
  - The web page is derived from data that changes frequently
  - The web page uses information from corporate databases or other server-side sources
- In principle, servlets could be used for requests other than HTTP, though this is not very common.

Advantages of servlets over CGI (1)

- Efficient
  - Servlets run in JVM. Each request is serviced using a thread rather than a new process (lower overhead)
    - Though some scripting languages e.g. PERL on certain web servers do this now
- Convenient
  - Provides infrastructure that parses and decodes HTML forms
- Powerful
  - Can communicate directly with web server
  - Multiple servlets can share database connections
  - Simplifies session tracking

Advantages of servlets over CGI (2)

- Portable
  - Written in Java and follow standard API
- Secure
  - CGI often executed using open source OS shells, which can cause many security breaches
  - Array checking & exception handling automatic in Java
- Inexpensive
  - Many Java web servers freely available

Uses

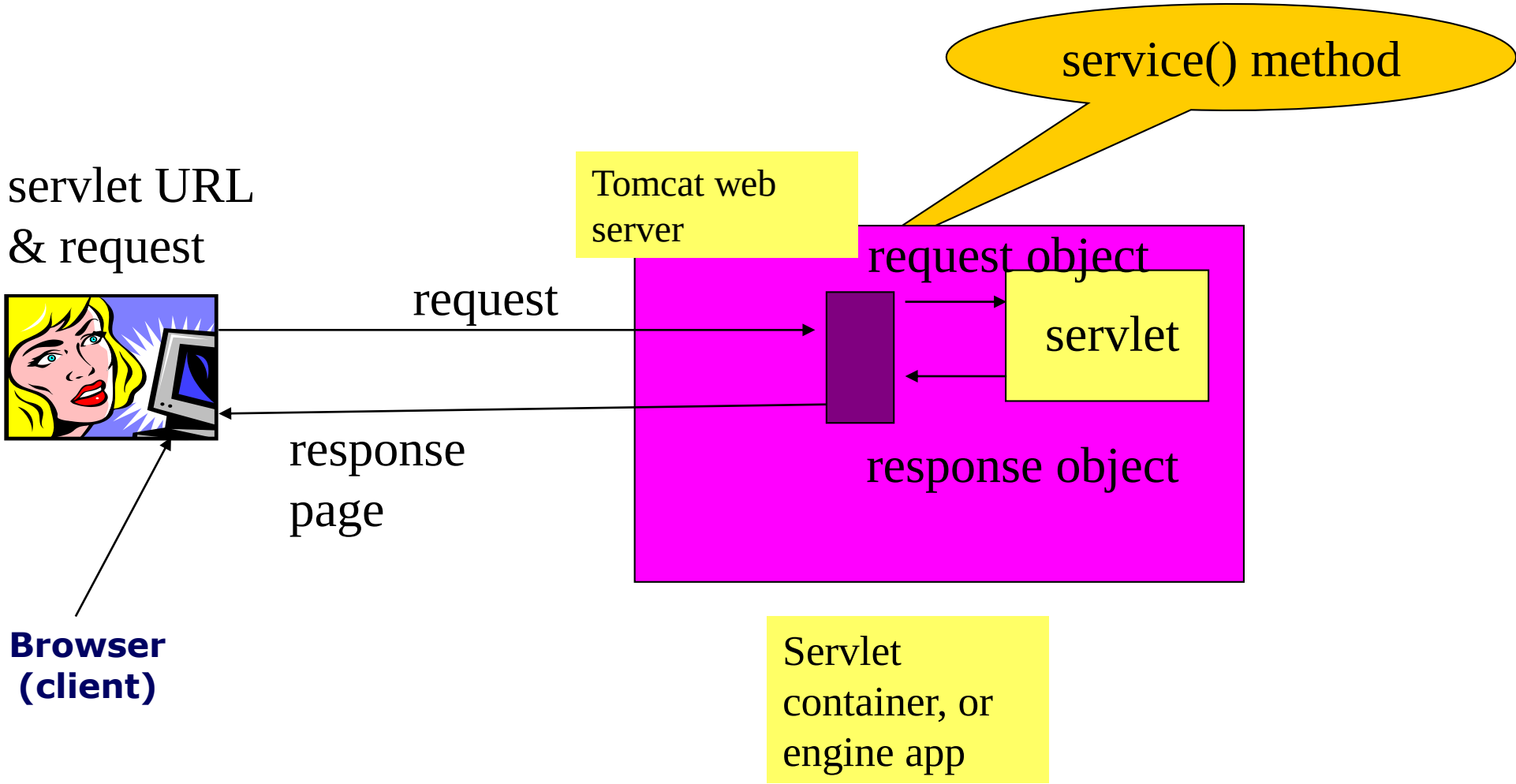
- Creating an e-commerce “shop front”
  - programmers use servlets in conjunction with JSP to create clearer and simpler applications.
- Providing web interfaces to legacy and data base systems
  - Avoids re-engineering existing system architecture
  - Can obtain access through firewalls by “HTTP tunnelling”
    - c.f. RMI

What servlets do

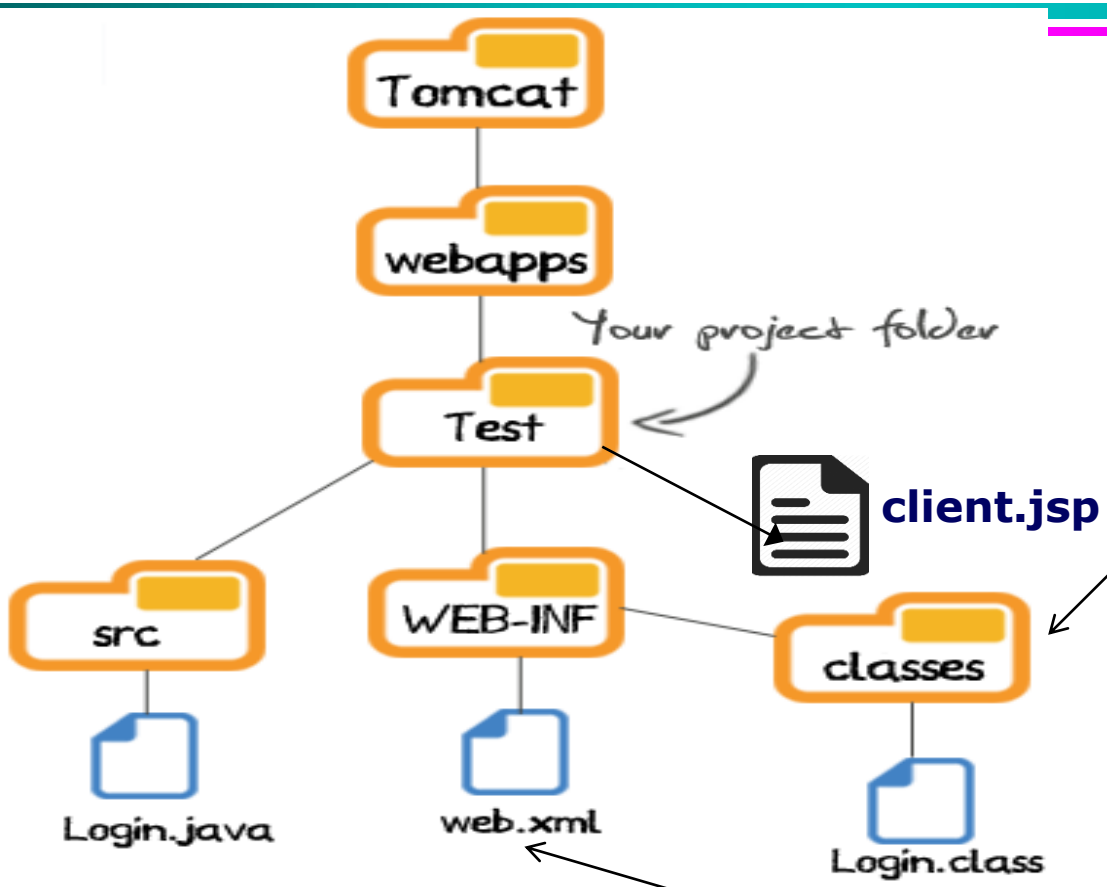
- Read any data sent by the user (explicit)
- Look up info embedded in HTTP request (implicit)
- Generate results
- Format results inside a document Format HTML or XML or GIF or Excel..
- Set appropriate HTTP response parameters
- Send the document back

  to the client

Tomcat is a web server



When you install tomcat (or similar) you create a folder in webapps



Where you place your class files (there can be many)

Where you place your DD (in lib inside this in eclipse)

Calling (or browsing) the servlet

It is a servlet

How to find

- e.g. **http://localhost:8080/servlet/myhelloservlet.HelloServlet**
- Browsing servlets differs from page browsing because you're executing a method of a servlet class instance, not looking at a page.
  - So like CGI
- The servlets handle processing, including form handing, calculation and database queries.
  - JSP is often used to format the results.
- Note: The above is simplified – fuller information later
  - A servlet can have a file path name, a name that the client uses, a name used only in the DD!

Typical generic servlet code

ONLY creates an object.

ONLY becomes a "proper" servlet after init

```
import javax.servlet.*;

public class AnyServlet extends GenericServlet {

        public AnyServlet() {}  // constructor – BUT USE THE DEFAULT
                                // NEVER ANY NEED TO WRITE ONE

        public void init(ServletConfig config) throws
        ServletException;
// The method actually called by container when servlet is first
//  created or loaded

    public void service(ServletRequest req, ServletResponse res)
                throws ServletException, IOException;
        // called by a new thread (in the container) each time a
                request is received

    public void destroy();
        // called when servlet is destroyed or removed
}
```

Servlet is "deployed" in a *container*, which is…..

- a program that receives (e.g. HTTP) requests to servlets from a web server application
  - finds the servlet (and loads, calls constructor & init if not ready)
  - creates or re-uses a thread which will call the service method of chosen servlet
  - creates & passes request and response **objects** to the chosen servlet
  - passes the response (e.g. HTTP response) back to the web server app; kills servlet thread or recycles into thread pool; and deletes request and response objects
- More generally, manages the life cycle of its servlets
  - *Calls constructor, init, service, destroy*
  - also invokes methods of *listening classes* that a servlet implements
- It has a "main" and is working "all the time"

Also provides

- declarative security using settings in Deployment Descriptor
- JSP support

init() inherited from GenericServlet

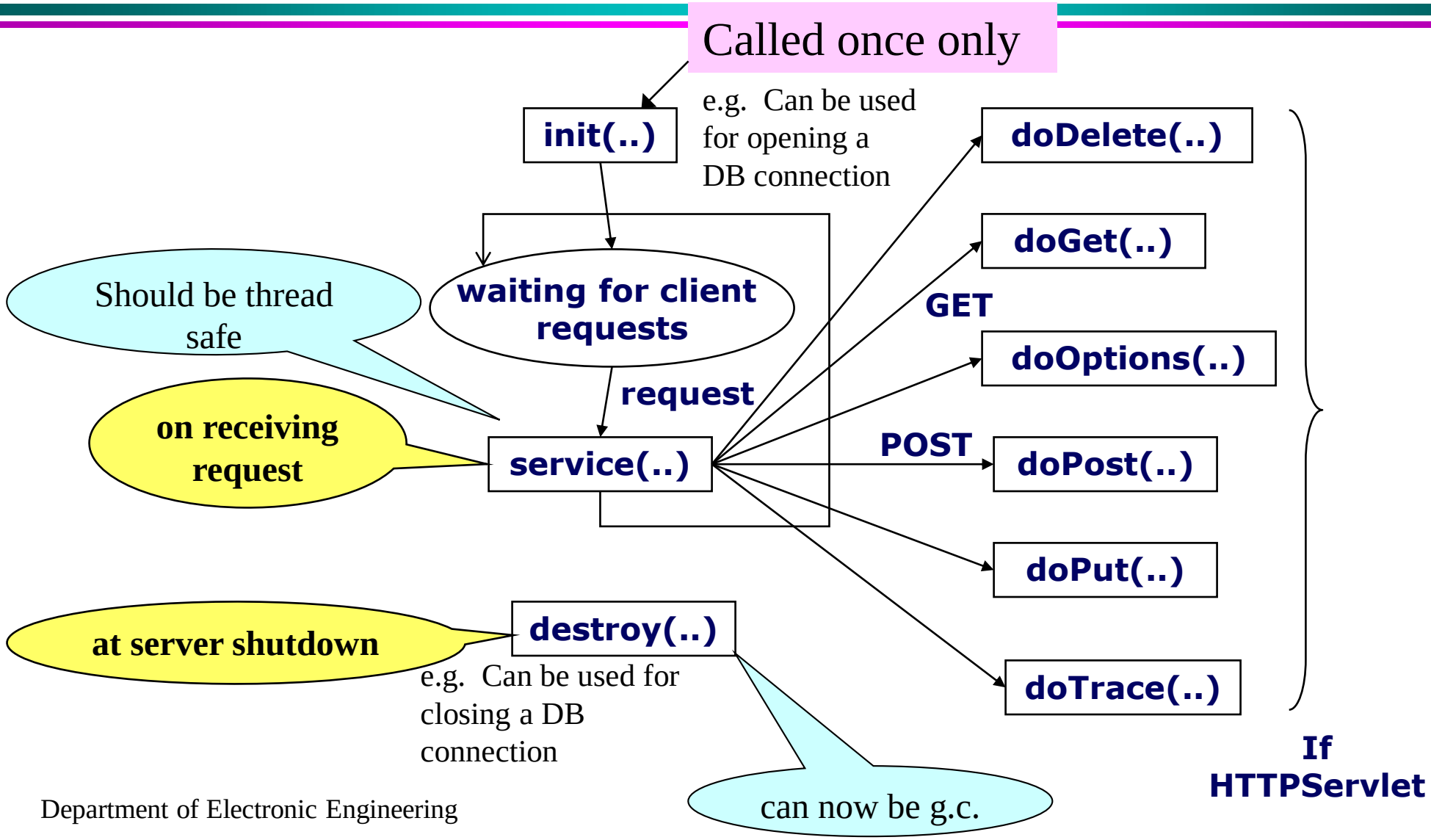
public void init(ServletConfig config) throws ServletException;

- **The method actually called by container when servlet is first created or loaded**
- **we DO NOT USUALLY OVERRIDE THIS METHOD**
- **it calls the method init()**

public void init() throws ServletException;

- **which WE DO USUALLY override with this servlet specific initialisation**

The servlet life cycle



Mapping names using the DD

```
<web-app ….>
<servlet>
    <servlet-name>….</servlet-name>
    <servlet-class>…</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>….</servlet-name>
    <url-pattern>…</url-pattern >
</servlet-mapping>
……
</web-app>
```

For each servlet in the web application

maps internal name to fully qualified class name.

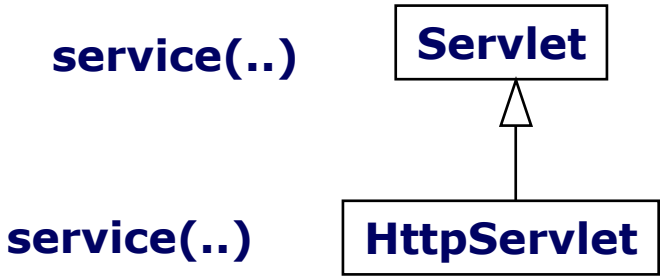
(except without .class)

maps internal name to public URL name

e.g. /makebooking

Internal name can be "anything" following XML rules

A complete servlet




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

public class S1 extends HttpServlet{
public void doGet(HttpServletRequest req,
                  HttpServletResponse res) throws IOException {
    PrintWriter out = res.getWriter();
    out.println("<html><body>Hello!</body></html>");}
}
```

Its DD

```
<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"
    web-app_2.4.xsd"
    version="2.4" >
<servlet>
    <servlet-name>Hello World Servlet</servlet-name>
    <servlet-class>S1</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Hello World Servlet</servlet-name>
    <url-pattern>/Hello</url-pattern >
</servlet-mapping>
</web-app>
```

Invocation in Eclipse

http://localhost:8080/ProjectName/Hello

Typical HTTPServlet code

```
import javax.servlet.http.*;
import javax.servlet.*;

public class AnyHttpServlet extends HttpServlet {

    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
        throws ServletException, IOException;
        // called when HTTP request uses GET

    public void doPost(HttpServletRequest req,
                       HttpServletResponse resp)
        throws ServletException, IOException;
        // called when HTTP request uses POST
}
```

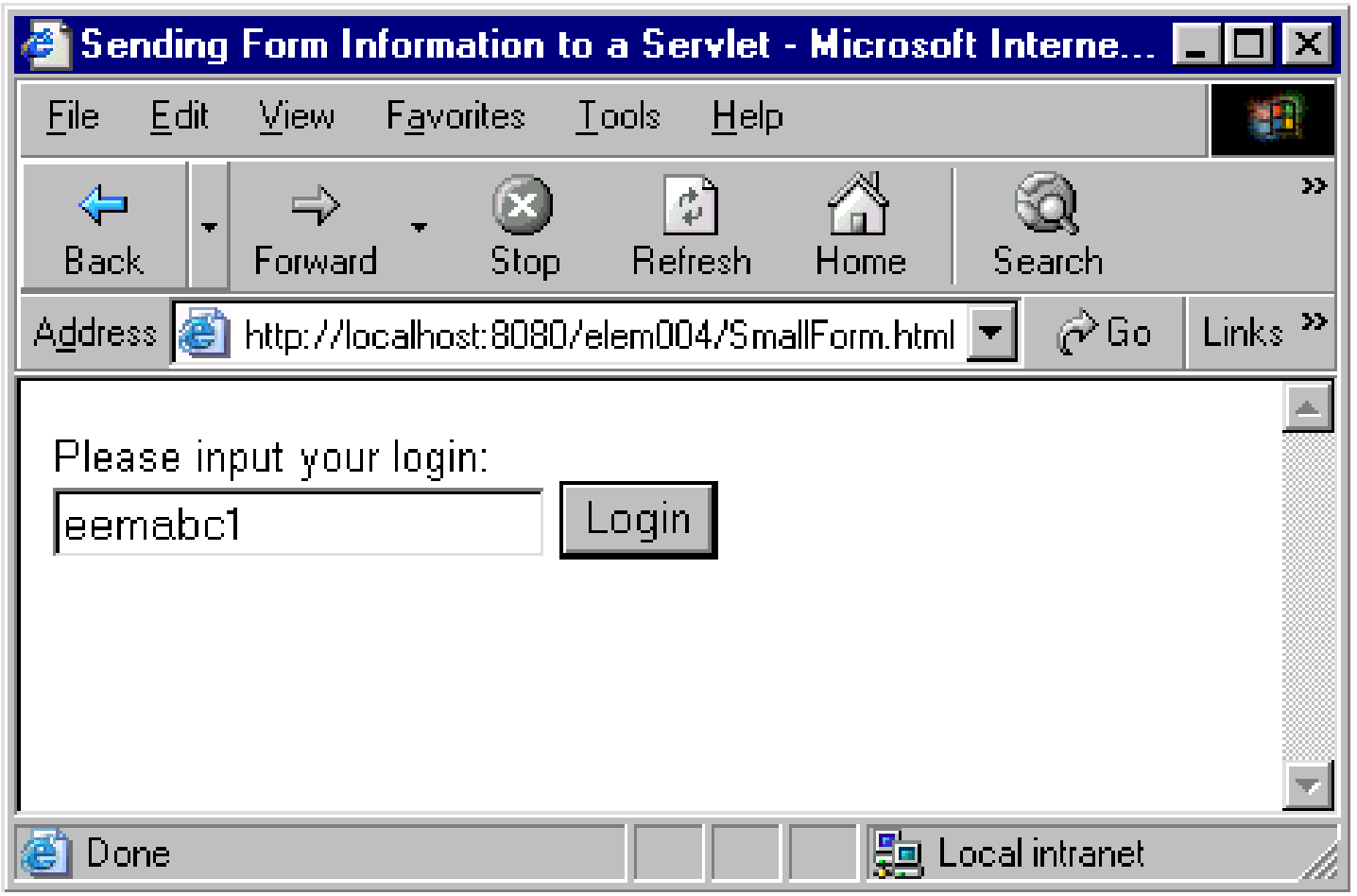
Handling the Client Request: HTML Form Data

- Form data (or query data) is used to transfer information to the server-side program via POST or GET methods.
- Servlets have built-in features to parse this data
  - no need to extract attribute-value pairs
  - no need for URL-decoding (much nicer than CGI!).

AKA key-value pairs

- ServletRequest methods:
  - `String getParameter(String);`
  - `Enumeration getParameters();`
  - `String [] getParameterValues(String);`

A Small Form


Sending Form Information to a Servlet - Microsoft Interne...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/elem004/SmallForm.html
Go Links
Please input your login:
eemabc1
Login
Done
Local intranet

Obtain single parameters
SmallForm.html

```
<html><title>Sending Form Information to a Servlet</title>
<body>

<form
action="http://localhost:8080/servlet/elem004.ProcessSmallForm"
method="post">

Please input your login: <br>
<input type="text" name="Login">
<input type="submit" value="Login">

</form>
</body>
</html>
```

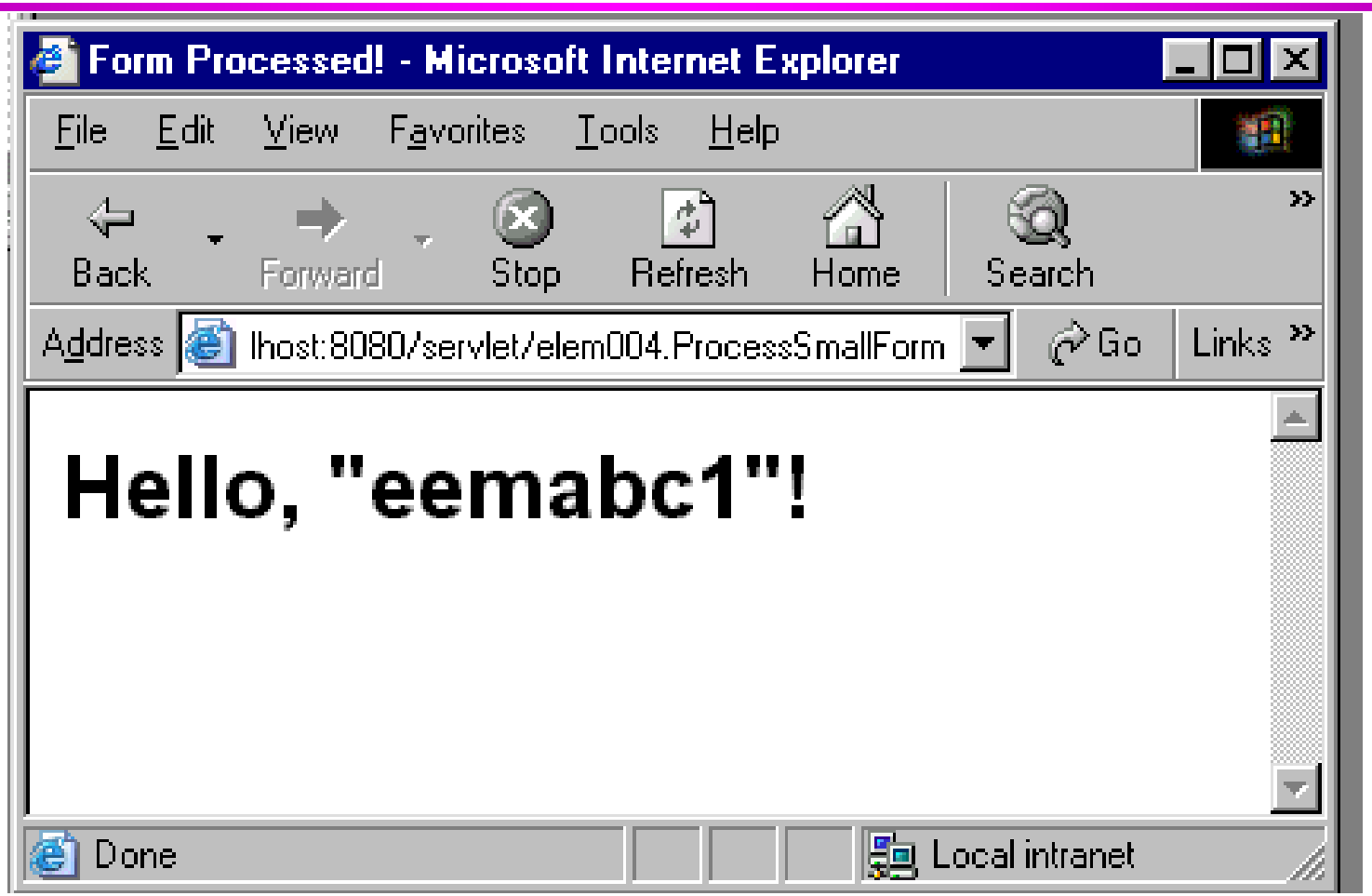
NB. Can use absolute or relative URLs or pre-configured names

Extract from ProcessSmallForm.java

```
public void doPost(…) { …
        // obtain the login from the Request object
        String loginName = req.getParameter("Login");

        out.println("<html><head>");
        out.println("<title>Form Processed!</title></head>\n");
        out.println("<h1>Hello, ");

        if(loginName != null)
                out.println(loginName);
        else
                out.println("mystery person");

        out.println("</h1></body></html>");
        out.close();
}
```

NB Almost but not complete – need PrintWriter and need to setContentType

After form is processed



Servlet initialisation & Servlet Configuration object

NB. Need to prevent race conditions (or use single thread model - deprecated)

- Only **one** servlet instance is created
  - each request serviced by a separate thread in container
- Prior to initialisation the ServletConfig object created by the container
  - One ServletConfig object per servlet
  - Container uses it to pass deploy time information to the servlet
    - Facts you do not want to hard code into the servlet, e.g. DB name
  - The names are specified in the DD
- Parameters are set in a server-specific manner, e.g.
  - in Tomcat in a file called web.xml
  - in Resin in a file call resin.config
- Parameters do not change while servlet is deployed and running
  - Like constants
  - If change need to redeploy

Example of init parameters in a DD (web.xml for tomcat)

```
<servlet>
    <servlet-name>Hello World Servlet</servlet-name>
    <servlet-class>S1</servlet-class>

<init-param>
    <param-name>lecturersEmail</param-name>
    < param-value >john@elec.qmul.ac.uk</ param-value >
</ init-param >
</servlet>
```

Container reads these and gives to ServletConfig object

Getting a parameter value from the ServletConfig object

```
out.println(
    getServletConfig().getInitParameter("lecturersEmail")
```

Returns the servlet's ServletConfig object

All servlets have this method

Extracting initialisation variables

- The `ServletConfig` object contains parameter's from the server's configuration file
  - config object retrieved using `getServletConfig()` method
- Parameters stored by `init` method for later use and are extracted from config in a portable way by
  - `String getInitParameter(String)`

```
public void init(ServletConfig config) throws ServletException {
        super.init(config);

        String initValue = config.getInitParameter("init_count");
        count = Integer.parseInt(initValue);
}
```

Since API 2.1

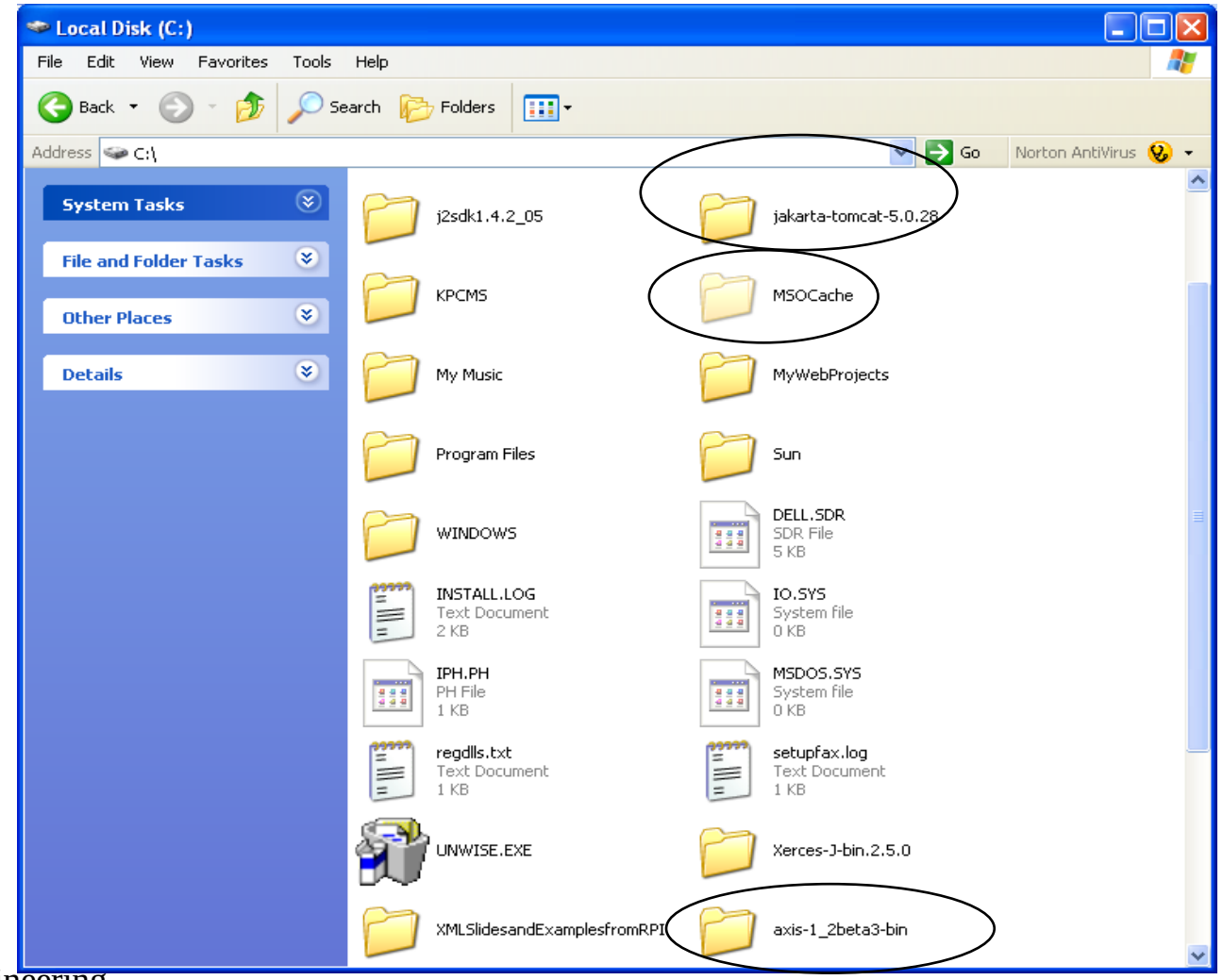
```
public void init() throws ServletException {

    String initPath=
    getServletConfig().getInitParameter("count_file");
    try {  // to read in value of counter

        BufferedReader countFile =
                                    Text.open(initPath);


        .........
    }
```

Location of servlets

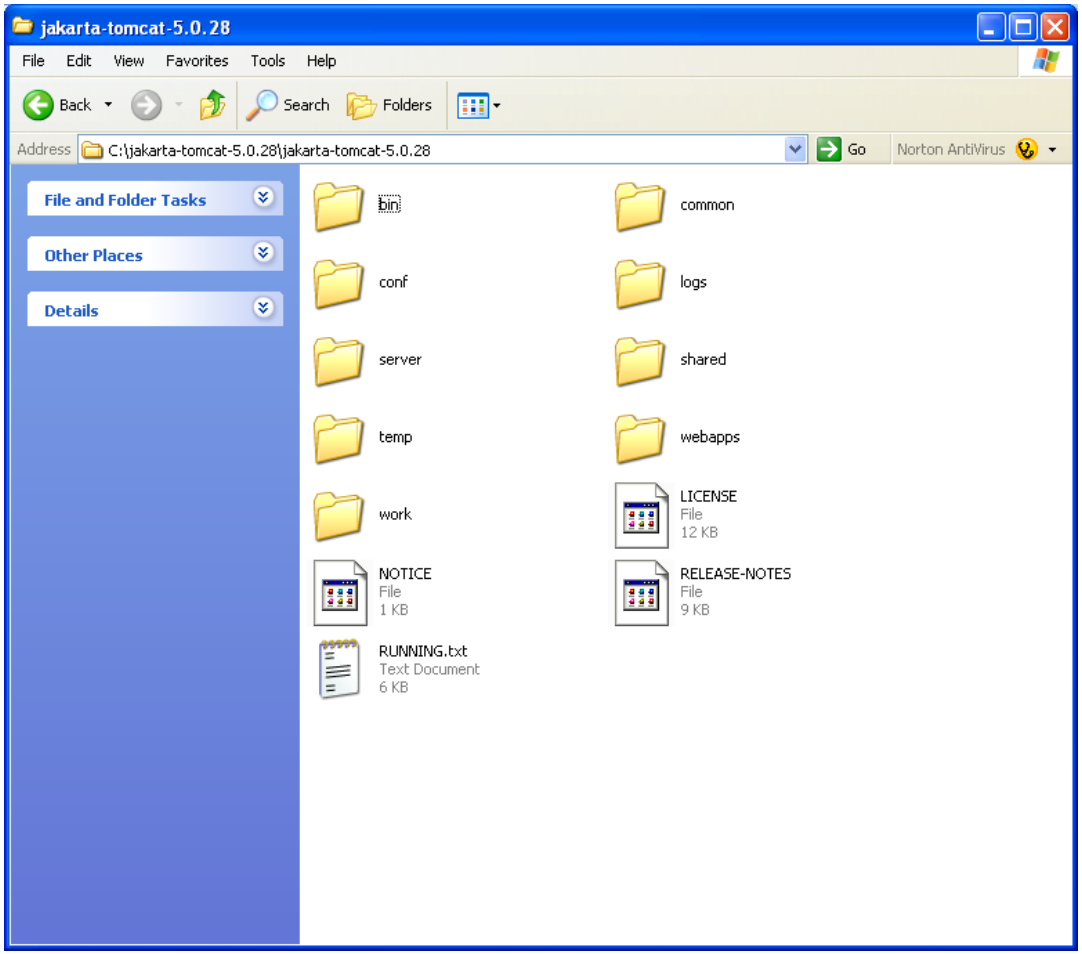
- Depends on the web server you're using but for Tomcat and Resin, servlets should be put in:

  /WEB-INF/classes

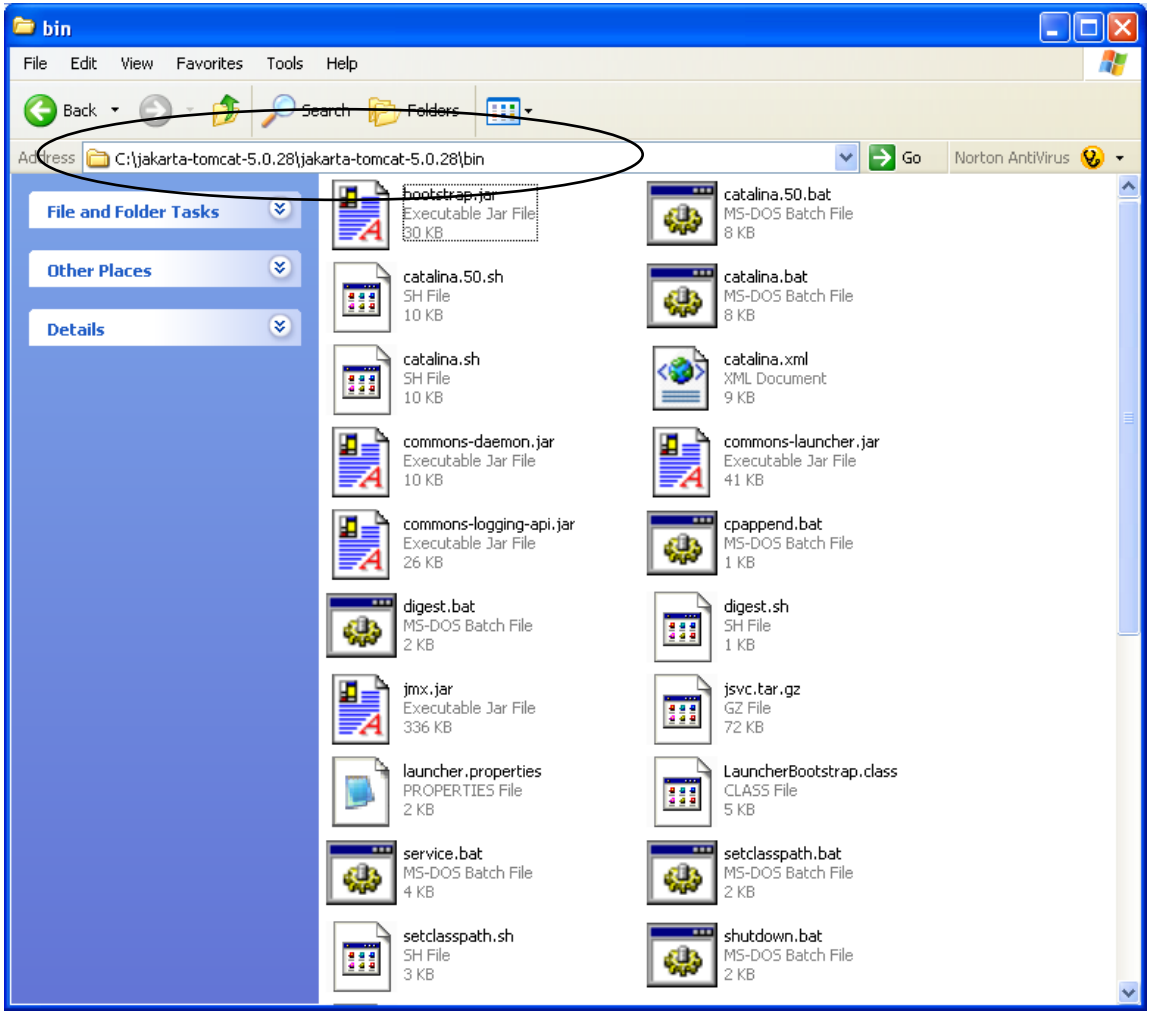
Some Important Locations



This is %CATALINA_HOME%



Tomcat Installation Path

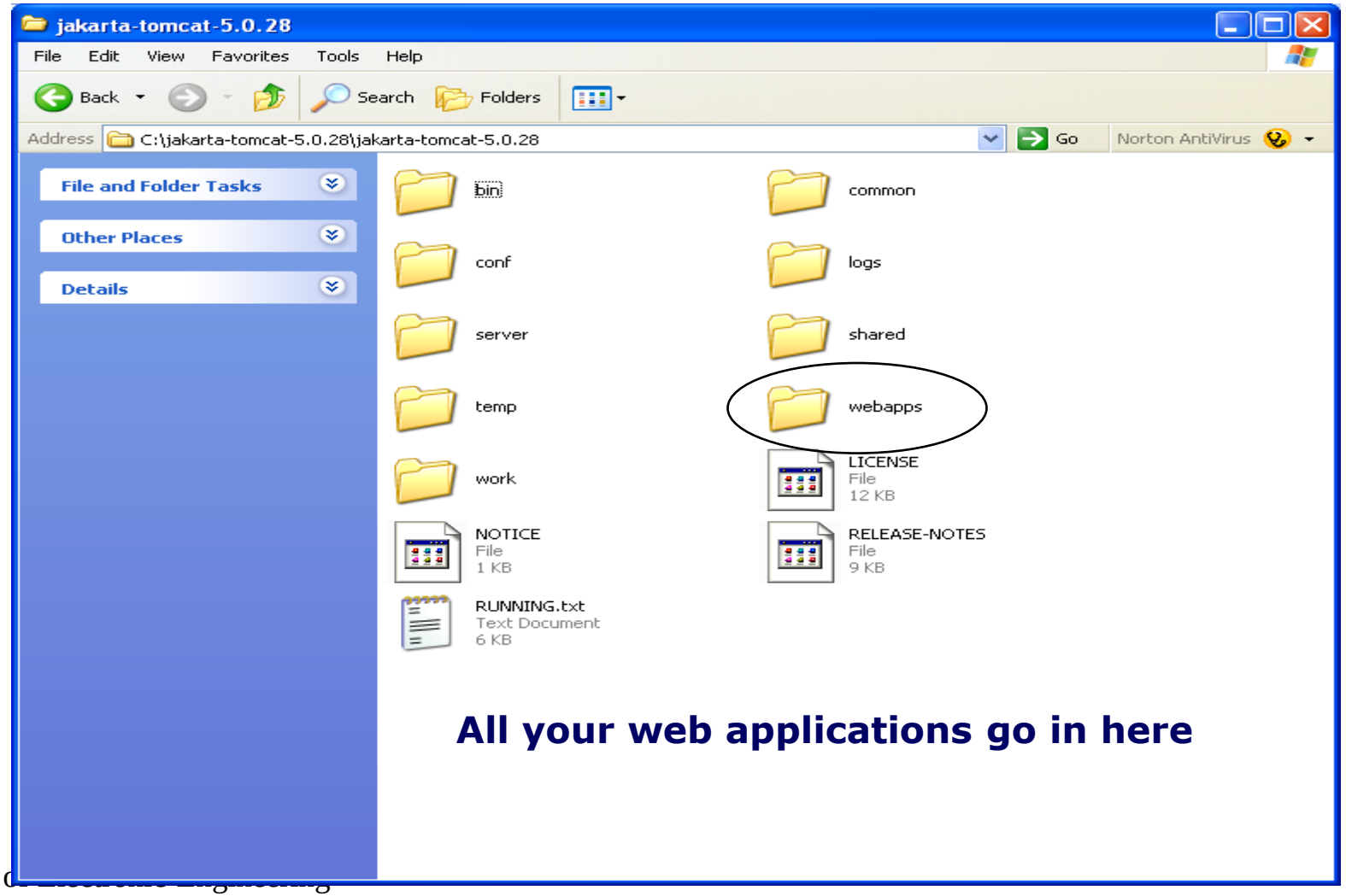


Some Command Line Tools

Command Prompt

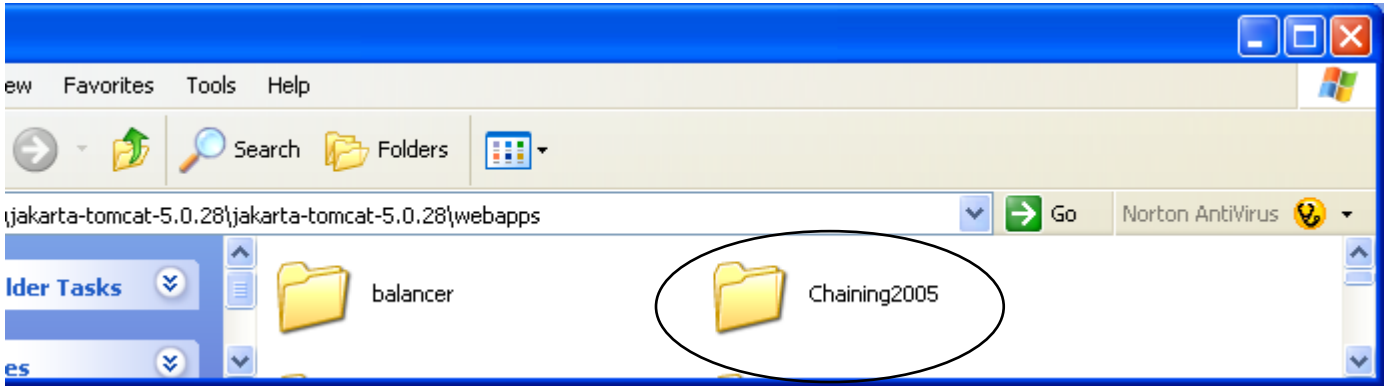
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\John>cd C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28
\bin

C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\bin>catalina start
Using CATALINA_BASE:   C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28
Using CATALINA_HOME:   C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28
Using CATALINA_TMPDIR: C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\temp
Using JAVA_HOME:       C:\j2sdk1.4.2_05

C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\bin>_
```

Where to put your Web Apps in Tomcat

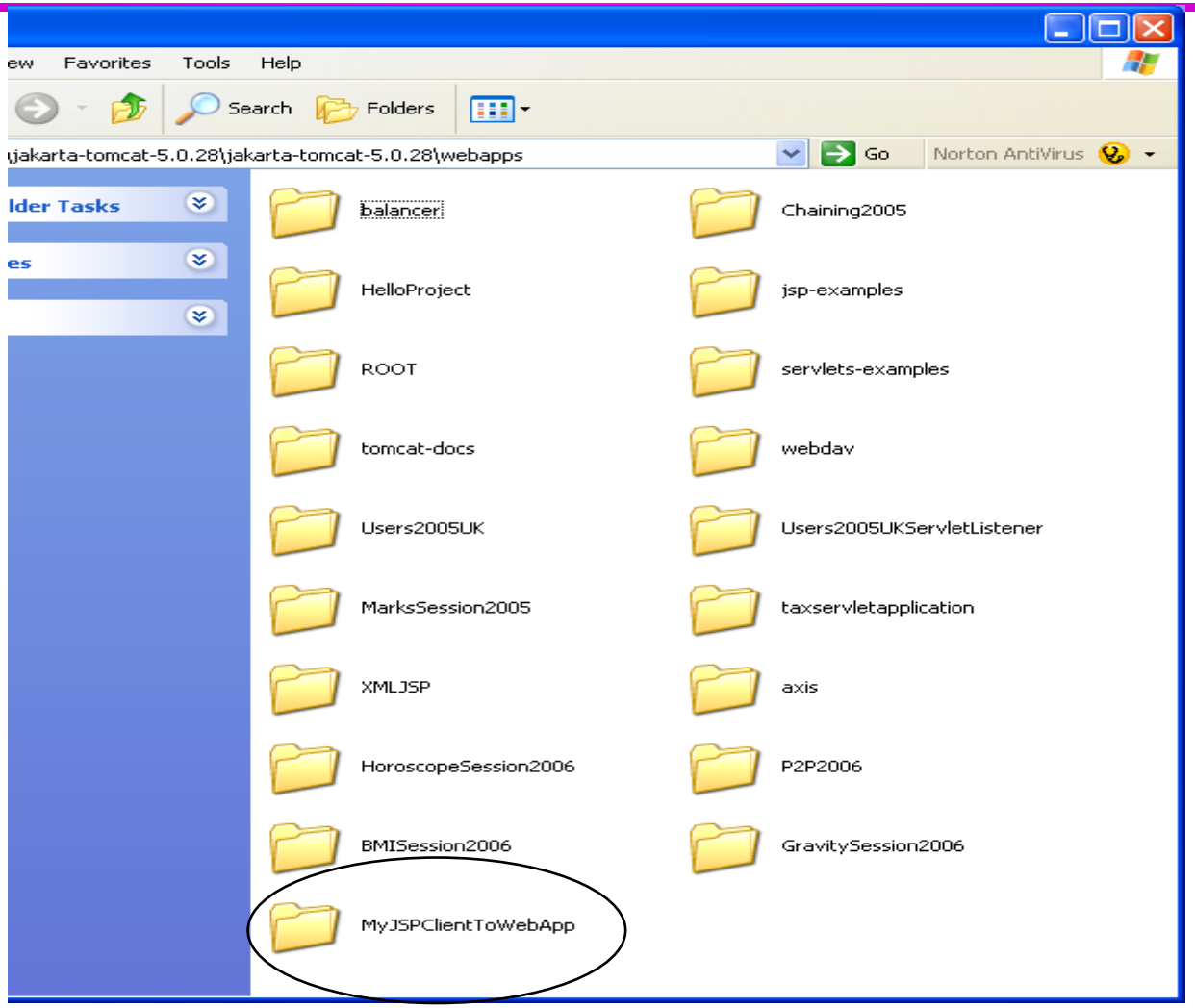


Inside webapps

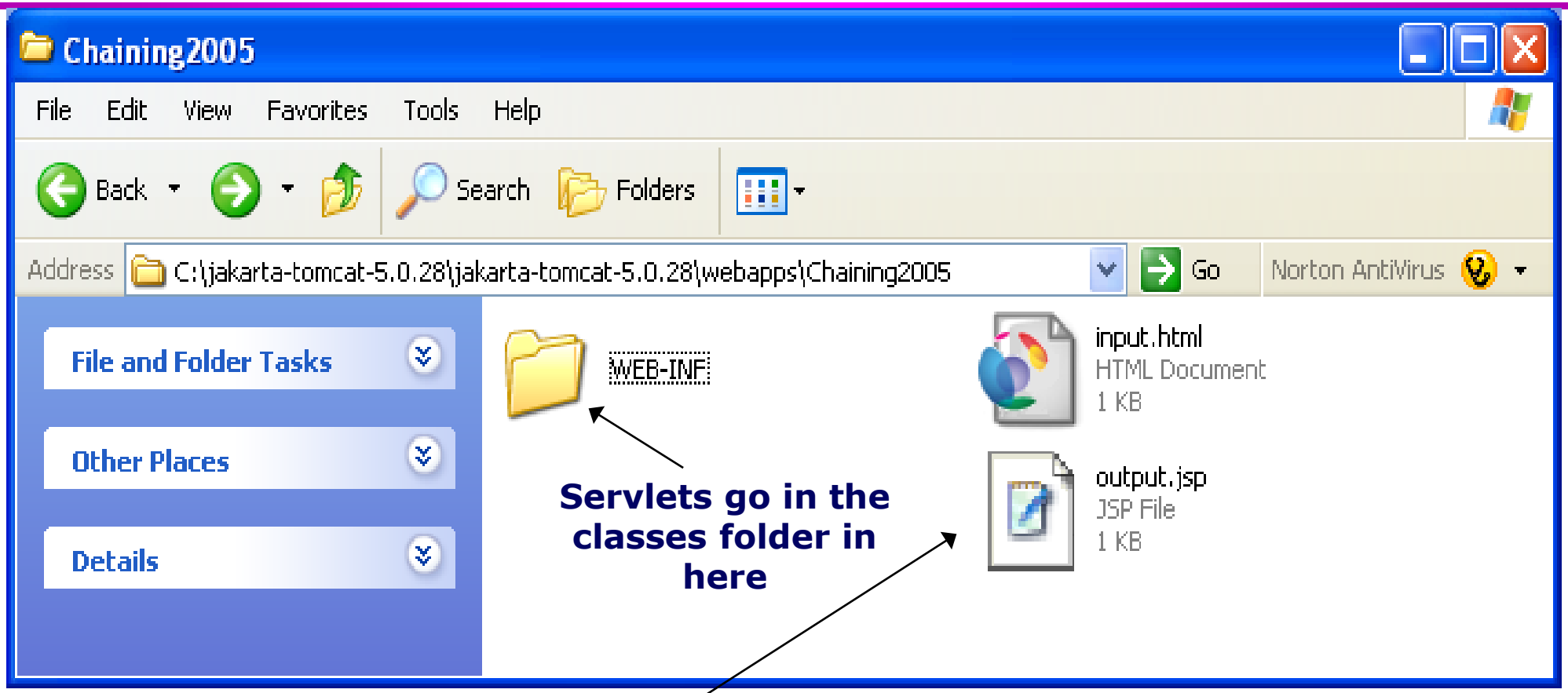
- Here are two web applications
- Usually many more



Where to Put Your Own Web App Directory



Inside *the Chaining2005* web application



Servlets go in the classes folder in here

HTML & JSP files go here

input.html

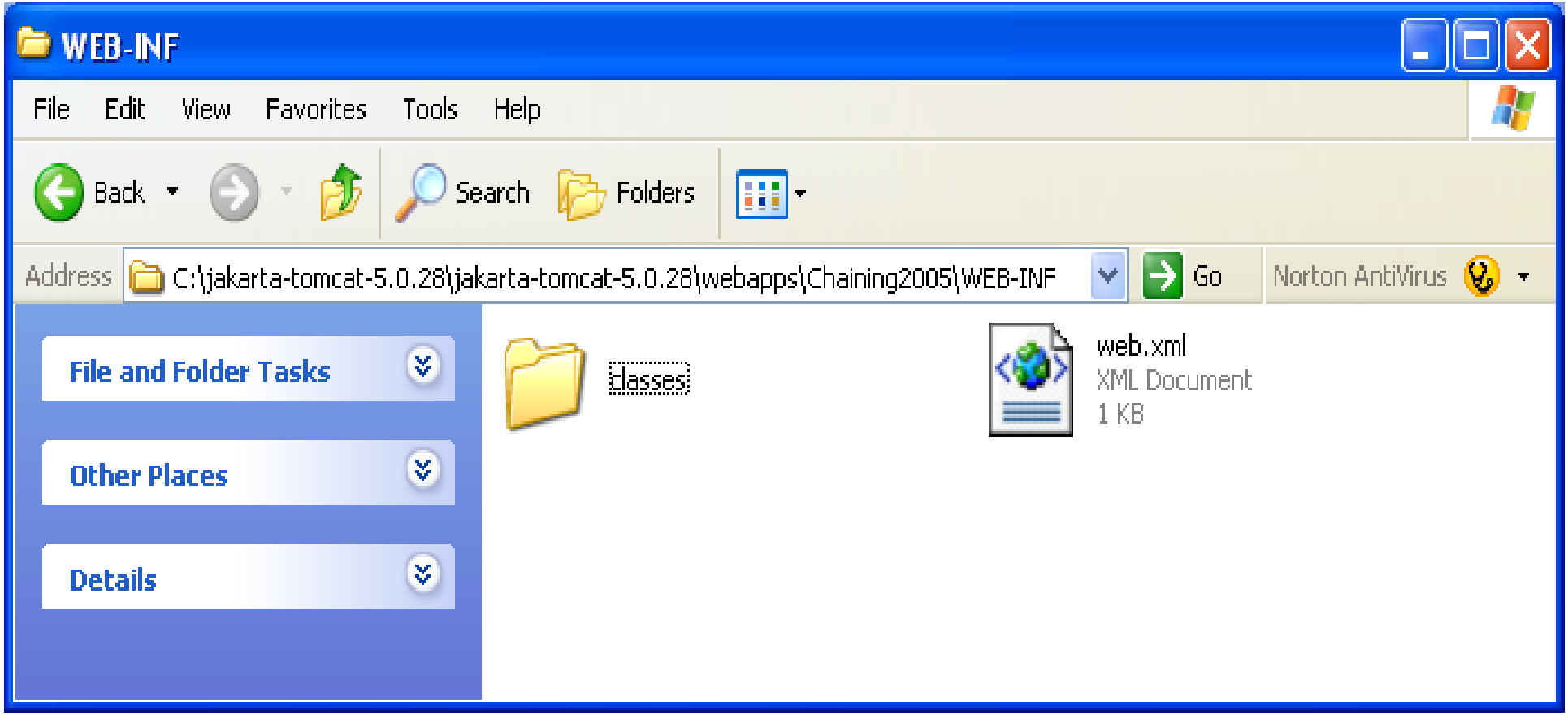
```
<html>
<head><title> Input book isbn </title> </head>
<body>
<!-- <form method="POST" action= "introspection.jsp"> -->
<form method="POST" action=
    "http://localhost:8080/Chaining2005/CH">

<input type="text" name="isbn" value="">
<input type="submit" >
</form>
</body>
</html>
```

It is only this because of what I have in the web,xml file

The name I have given for the servlet

The web app

Inside WEB-INF



The DD

C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\webapps\Chaining2005\WEB-INF\web.xml - Tiscali 10.0

File Edit View Favorites Tools Help

Address C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\webapps\Chaining2005\WEB-INF\web.xml

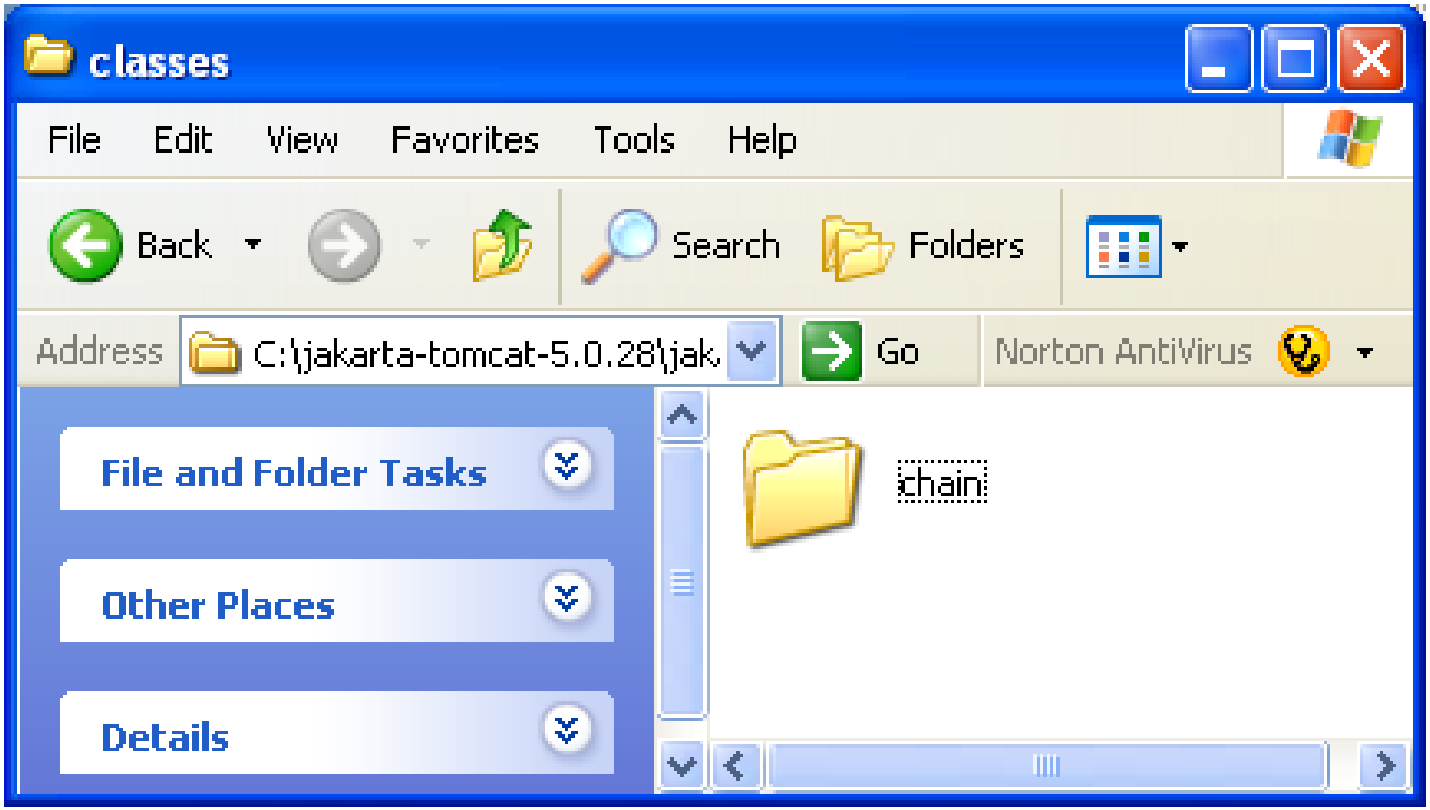
To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options...

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app (View Source for full doctype...)>
- <web-app>
  <display-name>Forwarding</display-name>
  <description>ELEM004 2005</description>
  - <servlet>
    <servlet-name>simple forwarder</servlet-name>
    <servlet-class>chain.web.AddNameServlet</servlet-class>
  </servlet>
  - <servlet>
    <servlet-name>output</servlet-name>
    <jsp-file>/output.jsp</jsp-file>
  </servlet>
  - <servlet-mapping>
    <servlet-name>simple forwarder</servlet-name>
    <url-pattern>/CH</url-pattern>
  </servlet-mapping>
</web-app>
```

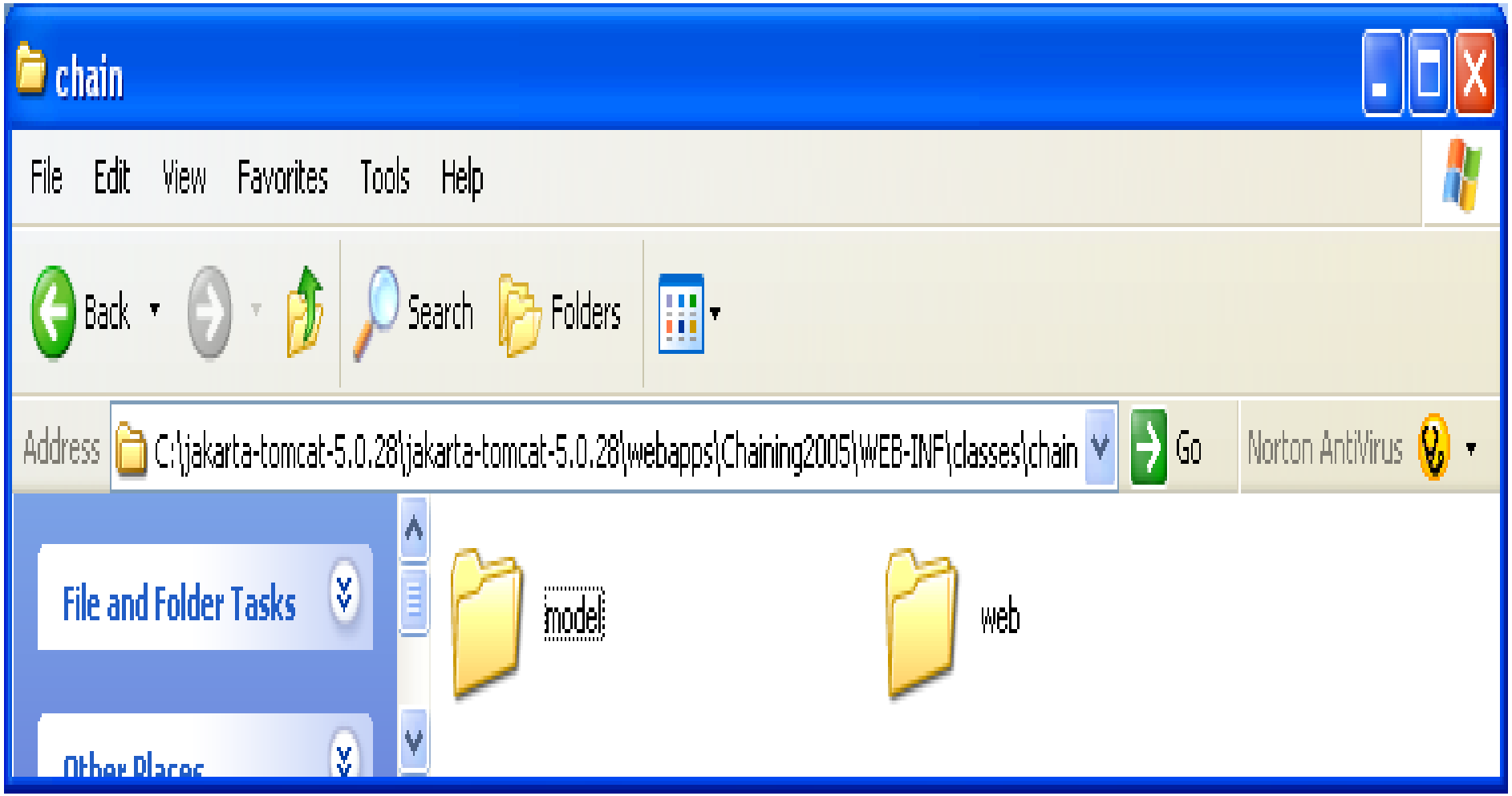
Note: full name starts from classes following package structure

Done My Computer

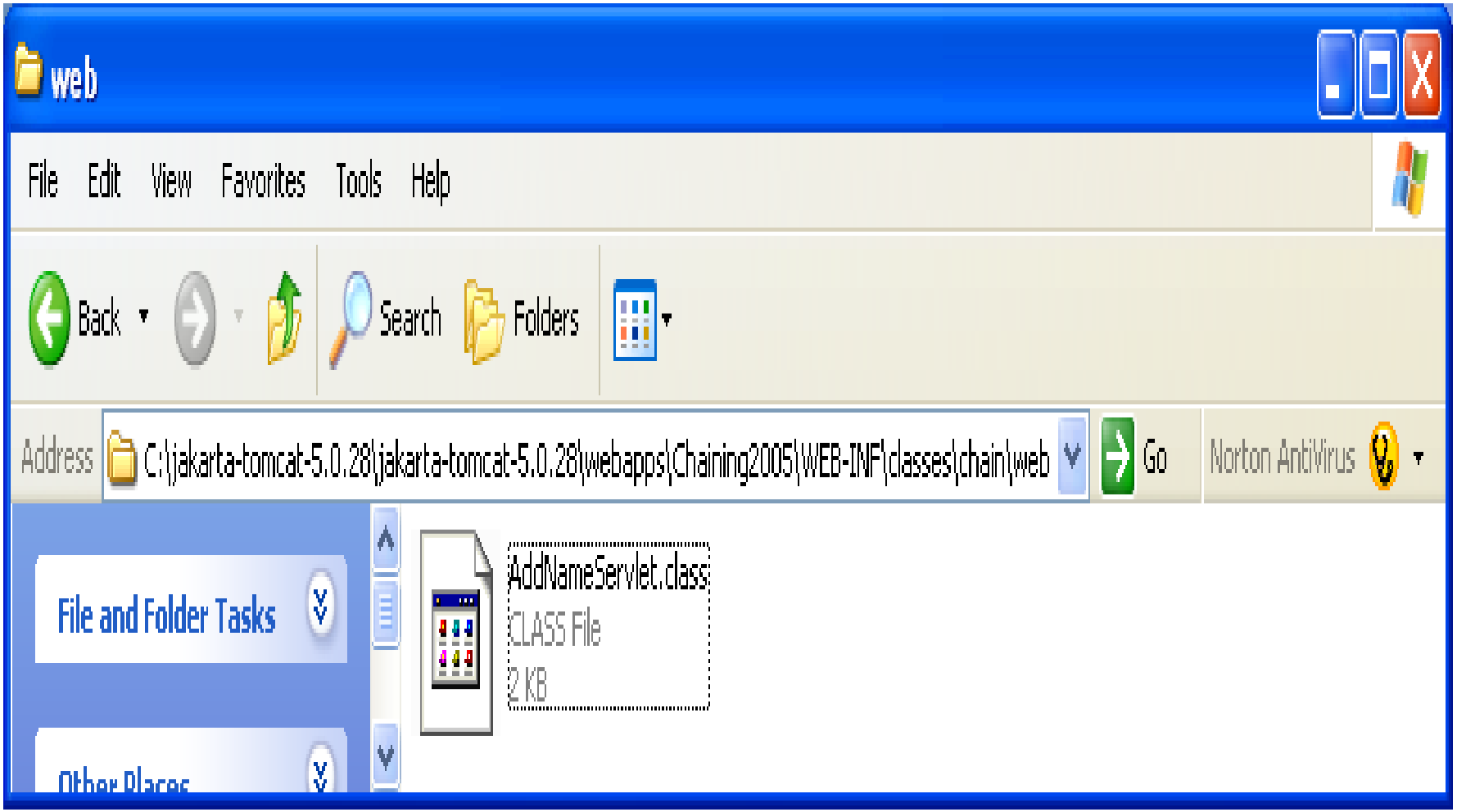
Inside classes



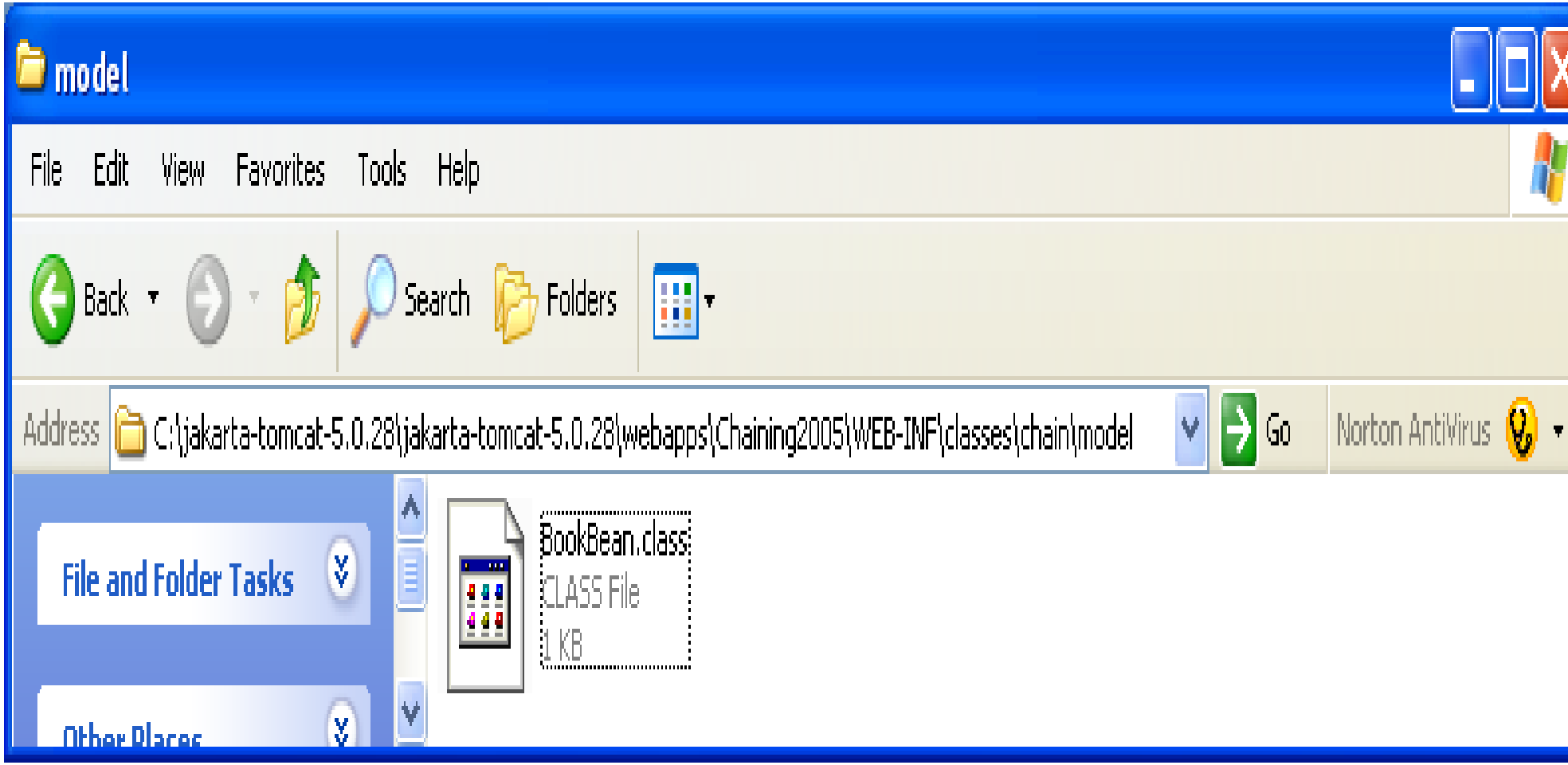
Inside chain



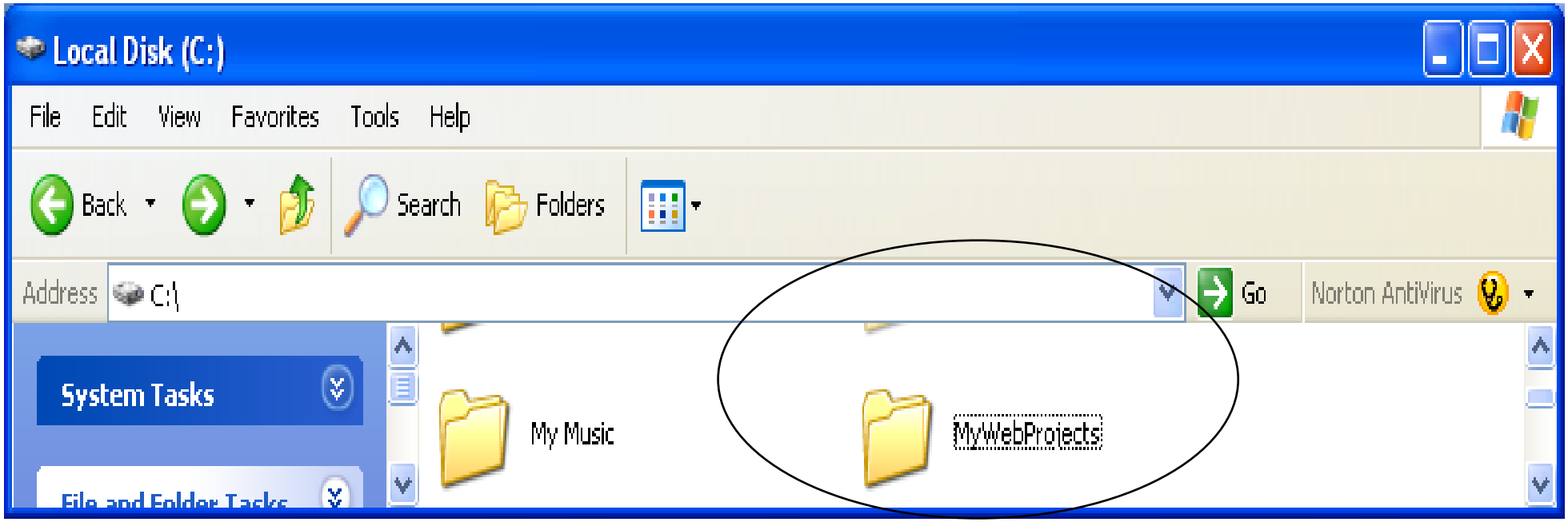
Inside classes/chain/web



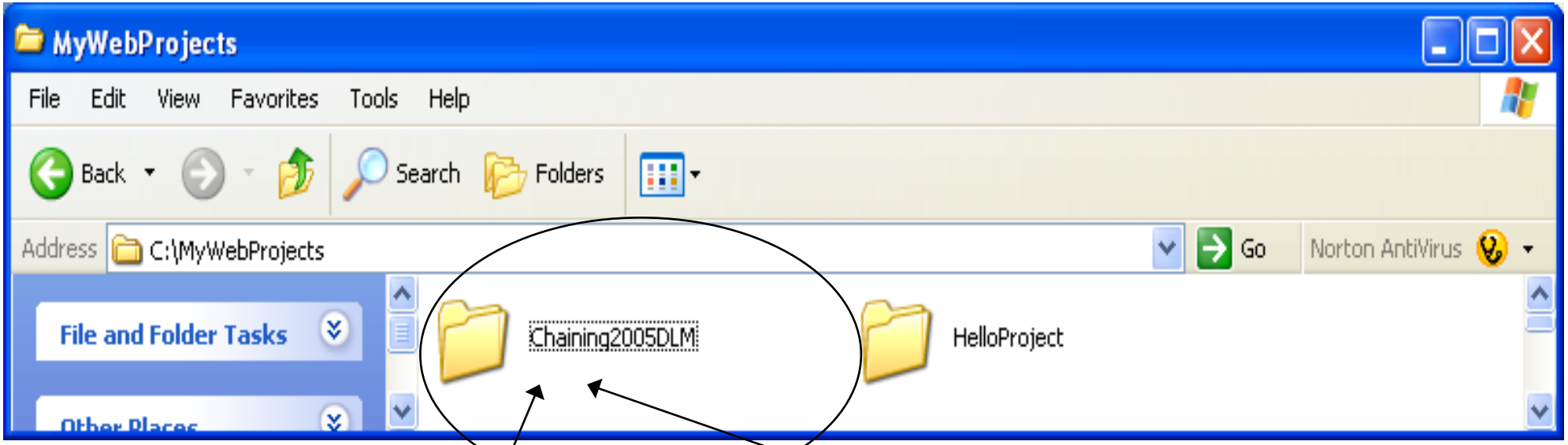
Inside classes/chain/model



Keep your servlet code in a similarly structured folder



Where we keep the java source for compilation

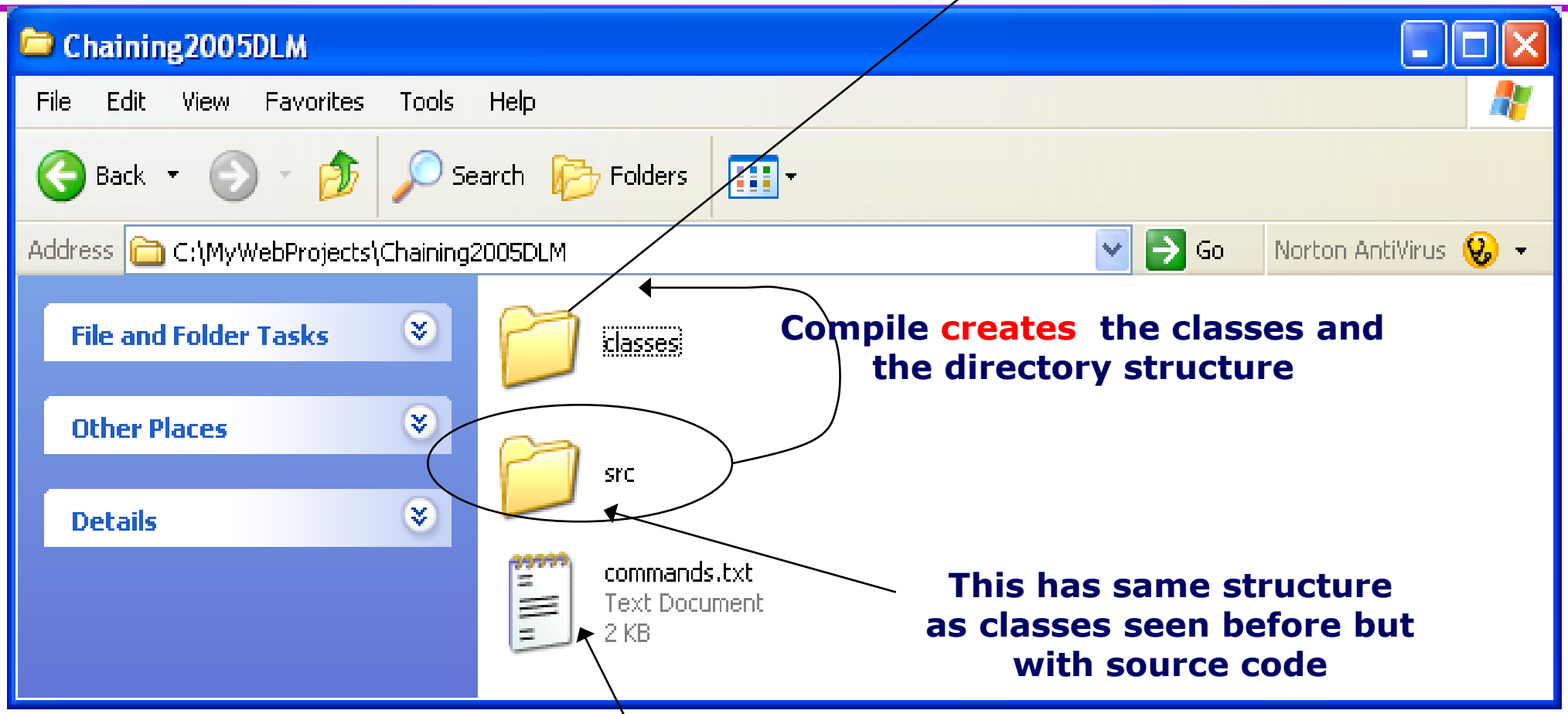


The code for the Chaining2005 web app

The corresponding folder

Queen Mary
University of London

Copy into tomcat class directory

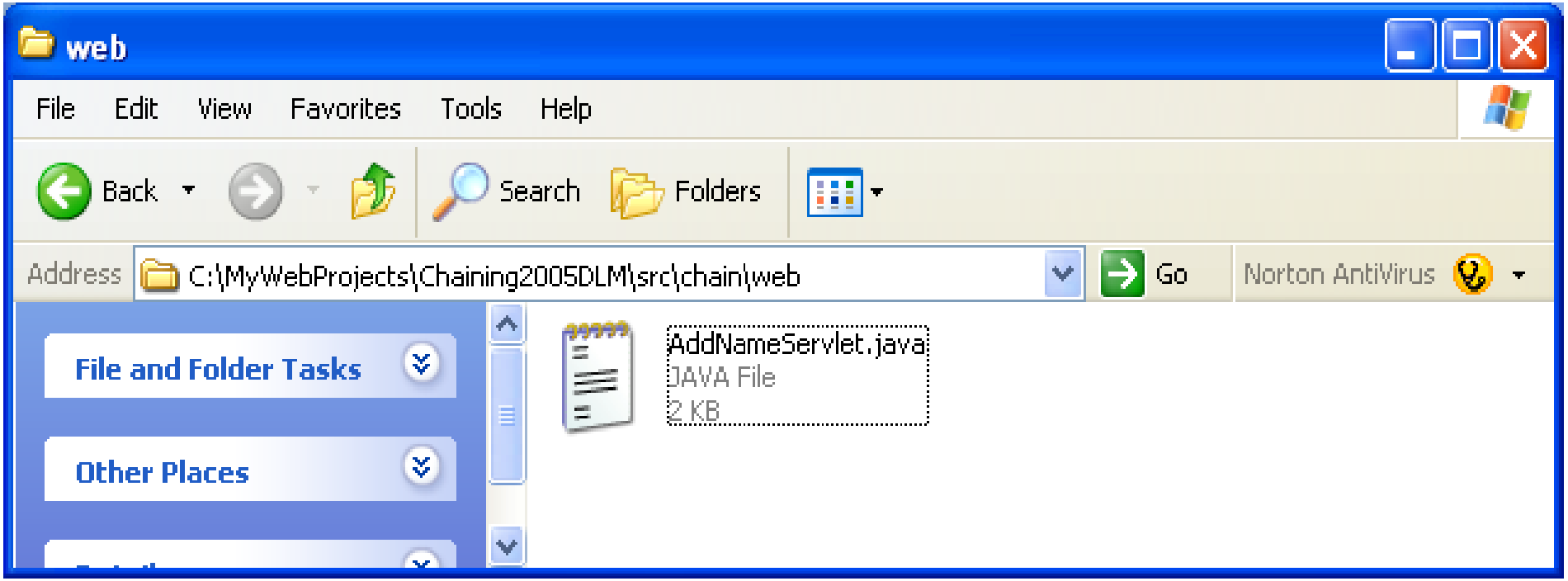


Compile creates the classes and the directory structure

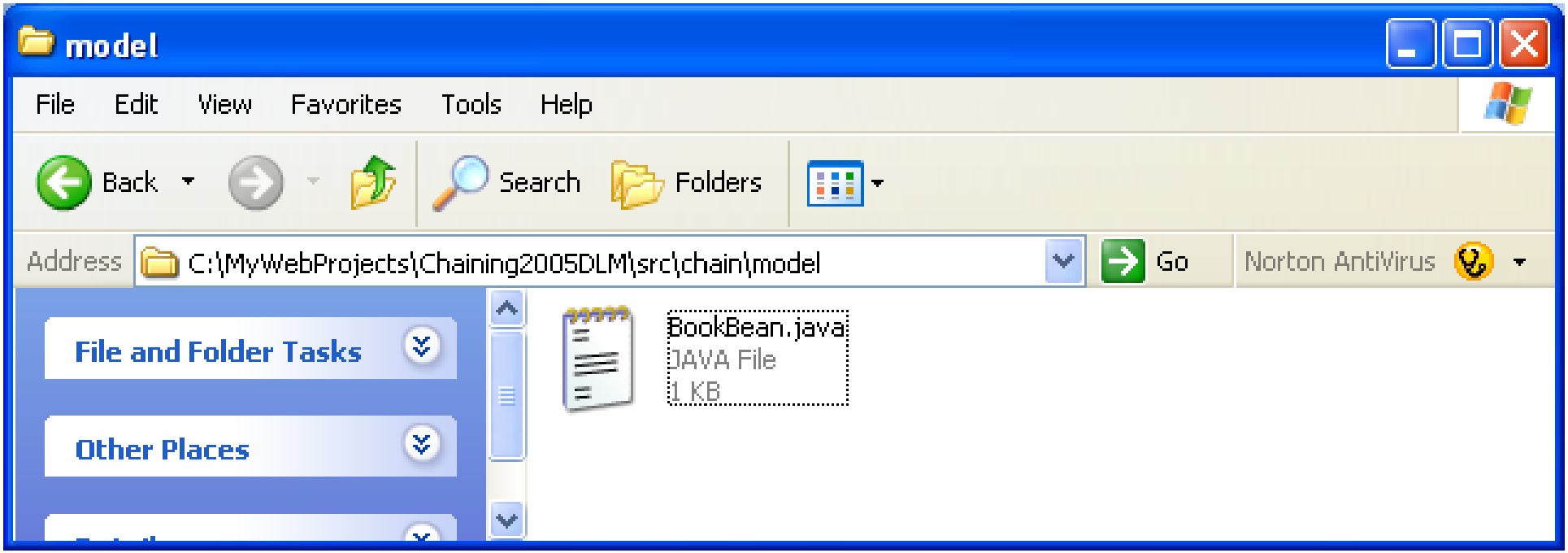
This has same structure as classes seen before but with source code

aide memoire

The Servlet Code Can Be Here



The Model Code Can Be Here



Compiling from Chaining2005 DLM

javac -classpath C:/jakarta-tomcat-5.0.28/jakarta-tomcat-5.0.28/common/lib/servlet-api.jar;classes;.

-d classes src/chain/web/AddNameServlet.java

javac -classpath C:/jakarta-tomcat-5.0.28/jakarta-tomcat-5.0.28/common/lib/servlet-api.jar:classes:.

-d classes src/chain/model/BookBean.java

AddNameServlet.java – package name

AddNameServlet.java - Notepad

File Edit Format View Help

```
package chain.web;

import chain.model.*;

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



public class AddNameServlet extends HttpServlet {
```

Do this whenever you change your recompile!

Command Prompt

```
C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\bin>catalina stop
Using CATALINA_BASE:   C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28
Using CATALINA_HOME:   C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28
Using CATALINA_TMPDIR: C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\temp
Using JAVA_HOME:       C:\j2sdk1.4.2_05

C:\jakarta-tomcat-5.0.28\jakarta-tomcat-5.0.28\bin>
```

And start again!

Extracting unknown parameters and multiple values

NB. Case sensitive

- `String getParameter(String)` used when parameter name is known
  - returns `null` if unknown parameter
  - returns `""` (empty string) if parameter has no value
- Else use `Enumeration getParameters()` to obtain parameters
- Then `String[] getParameterValues(String)` to obtain an array of values for each one
  - returns `null` if unknown parameter
  - returns a single string (“”) if parameter has no values

HTTP Request Headers (1)

- As with form data, HTTP request headers can be extracted from the HTTPServletRequest object
- Usually use `String getHeader(String)`

NB. not case sensitive

- Some common headers have their own methods
  - `getContentLength()`
  - `getContentType()`
- Can also access all headers using
  - `Enumeration getHeaderNames()`

HTTP Request Headers (2)

- Information on the main request line also has its own methods
  - `getMethod()`
  - `getRequestURI()`
  - `getProtocol()`

- Servlet to extract all information:

Servlet Example: Showing Request Headers - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit Discuss

Address http://localhost:8080/servlet/coreservlets.ShowRequestHeaders Go Links

Servlet Example: Showing Request Headers

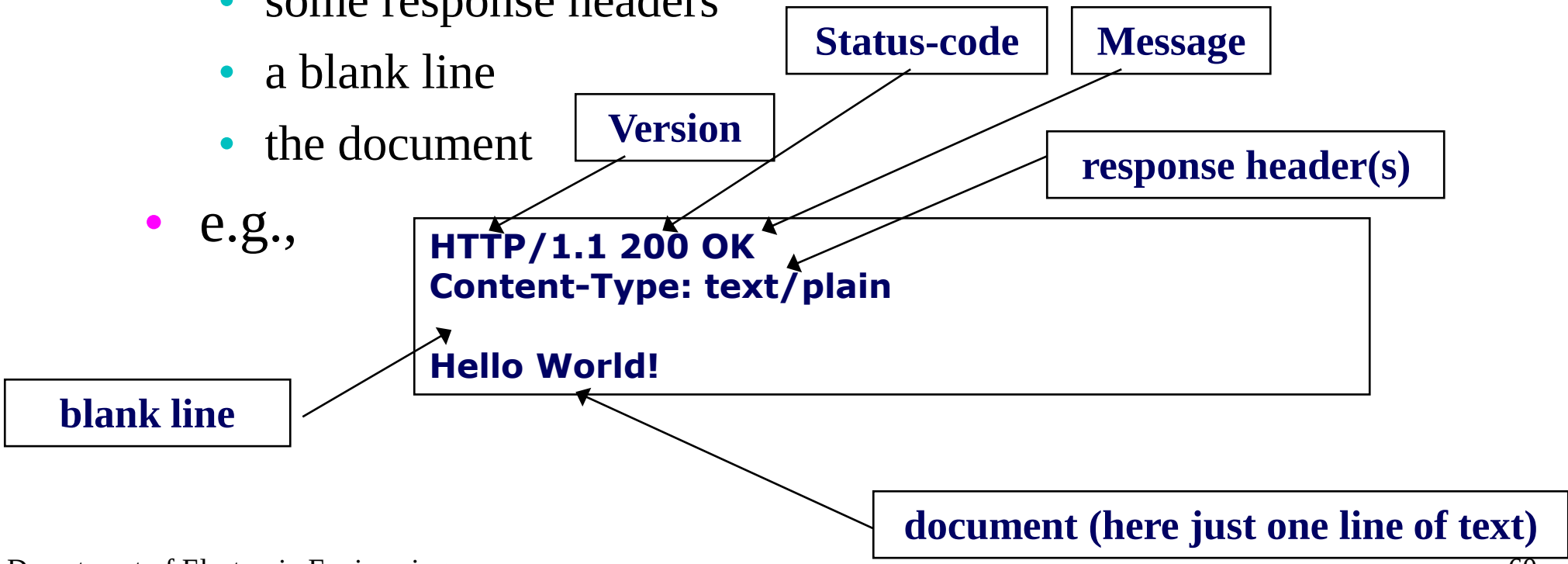
Request Method: GET
Request URI: /servlet/coreservlets.ShowRequestHeaders
Request Protocol: HTTP/1.1

Header Name	Header Value
Accept	application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/ms-bpc, */*
Accept-Language	en-gb
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)
Host	localhost:8080
Connection	Keep-Alive

Done Local intranet

Generating the Server Response

- Response typically consists of
  - a status line (containing version, status code + message)
  - some response headers
  - a blank line
  - the document
- e.g.,



HTTPServletResponse

- Servlets can perform a variety of tasks by manipulating the status line and response headers, for example
  - tell user that a password is required
  - indicate type of attached document (image, pdf, html)
  - forward user to other sites
  - etc.

HTTP Status Codes

- returned by the server to the client software to indicate the outcome of a request, e.g.
- **200 - OK**
  The request sent by the client was successful.
  301 - Moved Permanently
  The resource has permanently moved to a different URI.
- **303 - See Other**
  The requested response is at a different URI and should be accessed using a GET command at the given URI.
- **400 - Bad Request**
  The syntax of the request was not understood by the server.
- **403 - Forbidden**
  The server has refused to fulfill the request.

Setting the status code

- The servlet **only** needs to set the **status code** since the **version** is determined by the server and the **message** is associated with the status code.
- Usual method is simply to call

```
response.setStatus(int)
```

- If your response includes a special status code *and* a document you *must* call `setStatus` before returning any content via the `PrintWriter`.
  - This is because the document itself may not be buffered but sent in pieces, e.g. a large image file.

Special status codes

- Status codes are integers but it's better to use the constants defined in HTTPServletResponse class
  - e.g., `HTTPServletResponse.SC_NOT_FOUND`

Status code 404

- Two common codes have special methods

```
public void sendError(int sc, String message)
```

  - this sets the status code plus a short message

```
public void sendRedirect(String url)
```

  - generates a 301 response along with a Location header giving the URL of the new document that the browser should now request

Key part of form

```
<input type="submit" name="non-file" value="Fetch
   a non-existent file">
<input type="submit" name="redirect"
   value="Redirect to homepage">
<input type="submit" name="censored"
   value="Fetch censored content">
```

Only used to generate cases, in practice we find out that the case has arisen by looking at file system, etc.

Request Dispatch

- Servlet wants request to go to a different servlet or JSP in the web app
- Servlet calls

```
RequestDispatcher
   view=request.getRequestDispatcher("display.jsp");
view.forward(request, response);
```

- Now the JSP is invoked as if it were invoked directly by the client
  - The client will only see the URL of the original servlet

Setting Response Headers

- Response headers are are set using the method

```
res.setHeader(String header, String value);
```

- Examples include:
  - `Allow, Content-Encoding (e.g., gzip), Content-Language (e.g., en, en-us), Content-Length, Content-Type, Date, Expires, Last-Modified, Location, Refresh, Set-Cookie, WWW-Authenticate`

Support methods

- Setting some headers is so common that special support methods exist:

```
setDateHeader(String header,long msecs)
setIntHeader(String header, int value)
setContentType(String type)
setContentLength(int length)
addCookie(String cookie)
sendRedirect(String encodedURL)
```

Setting the Content-Type header

- Content-Type tells browser what sort of document is being sent.
- So far we've only used only `text/html`
- Generally of form **`maintype/subtype`** e.g.,
  - text/plain, text/html, text/css
  - image/gif, image/png, image/jpeg, image/tiff
  - application/pdf, application/msword
  - video/mpeg, video/quicktime
- Must be set before writing to the OutputStream.

The browser does not render this like html

Is any content-type acceptable?

May need to check which types browser supports

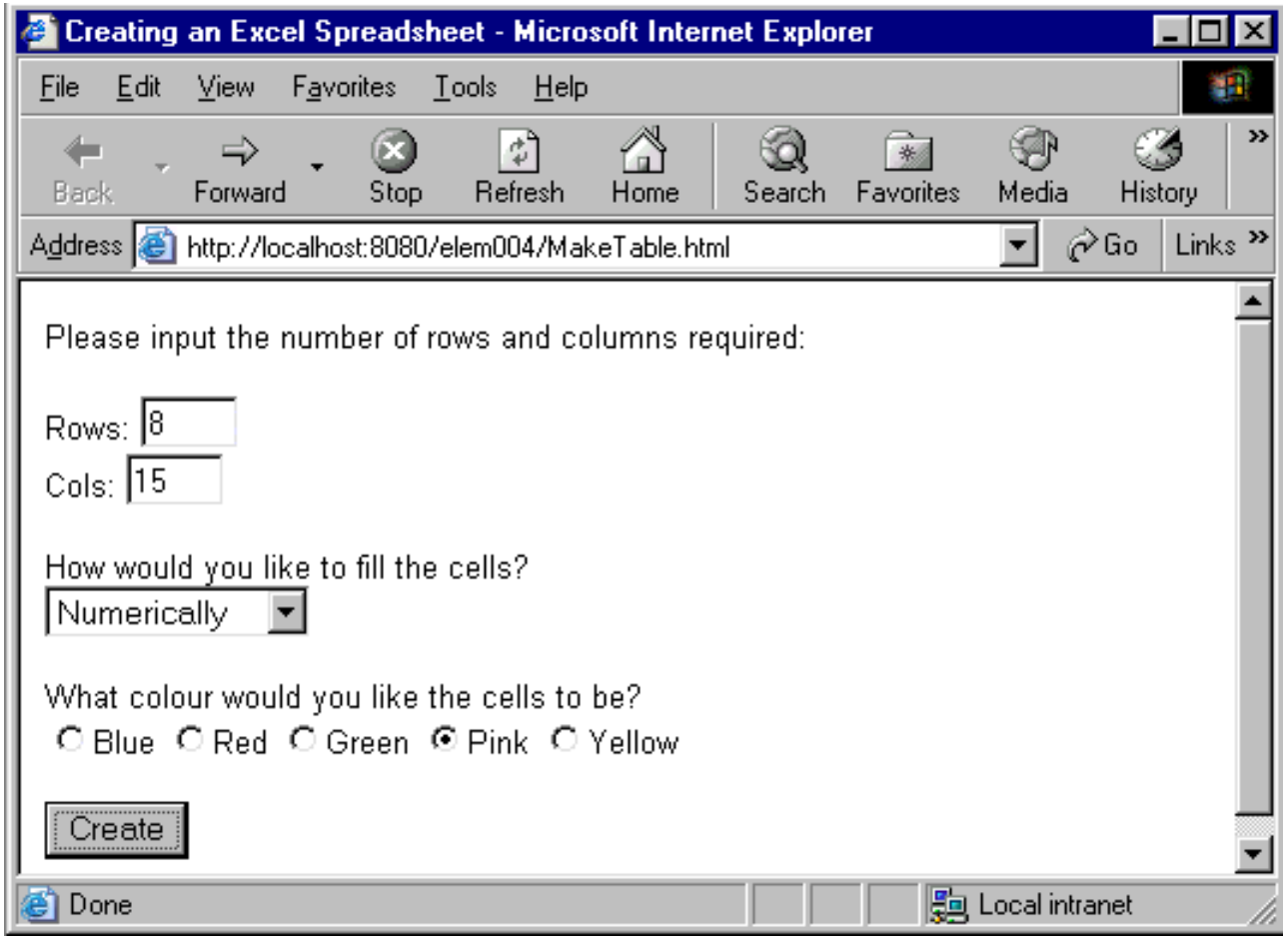
- use `String req.getHeader("Accept")` and check that output string contains the format you wish to send

```
String types = req.getHeader("Accept");

if(Utilities.contains(types, "image/jpeg")) {
        res.setContentType("image/jpeg");
        // send a jpeg file
}
else {
        res.setContentType("image/gif");
        // send a gif file
}
```

Creating an Excel spreadsheet

- As well as sending back html, servlets can dynamically create and send back more specialised content.
- The following example shows a servlet that creates an Excel spreadsheet using form data input by the user to format and fill the cells.

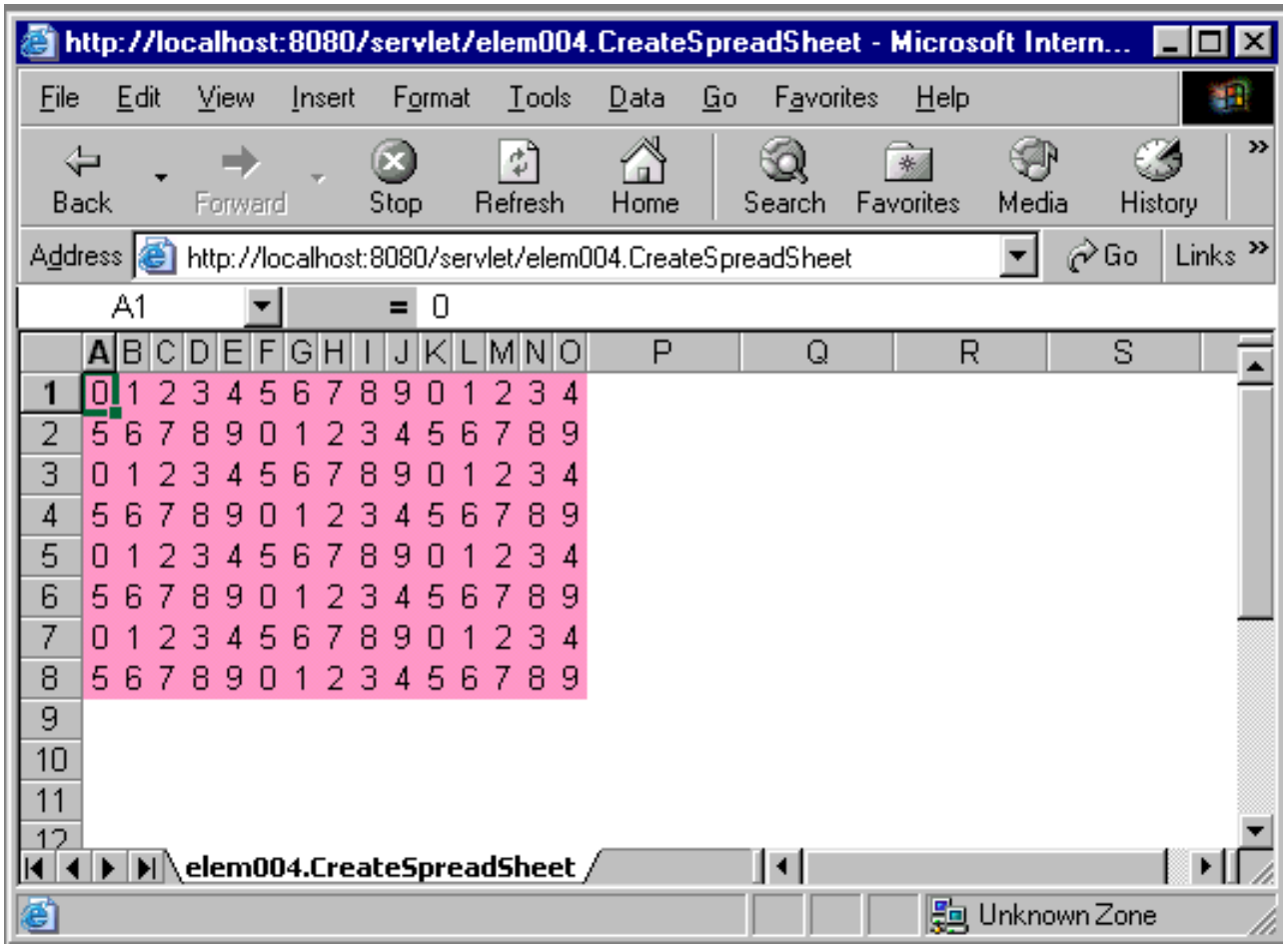
MakeTable.html



Extract from CreateSpreadSheet.java

```
res.setContentType("application/vnd.ms-excel");
PrintWriter out = res.getWriter();

out.println("<table>");
for(int r = 0; r < rows; r++) {
   out.println("<tr>");
   for(int c = 0; c < cols; c++) {
         out.println("<td bgcolor=\"" + colour + "\" >"
                     + chars.charAt(next++) + "</td>");
         next %= mod;
   }
   out.println("</tr>");
}
out.println("</table>");
out.close();
```

Output from CreateSpreadSheet



ServletContext object is created for a web app

- One per web app
  - A web app normally has several servlets (& JSPs)
- Used to access web app parameters that need to be seen by all servlets (& JSPs) in the application
  - Held as parameters in the ServletContext object
  - A misnomer as relates not to a servlet but the set of servlets and JSPsin the web app

The DD of the web app specifies the context parameters

```
<web-app …>……
<servlet>
    <servlet-name>… </servlet-name><servlet-class>… </servlet-class>
    <init-param><param-name>….</param-name>
                <param-value>….</param-value> </init-param>
</servlet>          … + other servlets

..
e>HOD_Email</param-name>
e>laurie@elec.qmul.ac.uk</param-value>
</context-param>
..
</web-app>
```

ServletContext object created and set up when web app is deployed

Note: Not inside any servlet

These are parameter name value pairs: both are strings

To access parameters in servlet code …

```
ServletContext ctx = getServletContext()
out.println(ctx.getInitParameter("HOD_Email")
);
```

Note: Same name for get method as when accessing ServletConfig object

- Context parameters generally more commonly used than Config
  - Typical use (of former) a DB lookup name

Can access ServletContext()….

- directly

getServletContext().getInitParameter(….)

- from ServletConfig

getServletConfig().getServletContext().getInitParamter(…..)

Latter useful if in a method of an auxiliary class, e.g. a bean, and only the ServletConfig object has been passed as a parameter

ServletContext also has Attributes

- Parameters are name value pairs, where both name and value are strings
- Attributes are name value pairs where the name is a string, but the value is an object (that may not be a String)
  - Accessed by getAttribute(String)
  - Set by setAttribute(String, Object)

Part of the DD

```
<servlet>
  <servlet-name>NeighbourhoodWatchServlet</servlet-name>
  <servlet-class>jb.NeighbourhoodWatchController</servlet-class>

  <init-param>
  <param-name>receivePortNumber</param-name>
  <param-value>5556</param-value>
  </init-param>
  <init-param>
  <param-name>sendPortNumber</param-name>
  <param-value>5555</param-value>
  </init-param>
  <init-param>
  <param-name>distantAppIPAddress</param-name>
  <param-value>127.0.0.1</param-value>
  </init-param>
 </servlet>
```

```
<servlet-mapping>
        <servlet-name>NeighbourhoodWatchServlet</servlet-name>
        <url-pattern>/NHWC</url-pattern>
</servlet-mapping>
```

Calling the servlet

http://127.0.0.1:8080/NHWWeb/NHWC

init parameters re web master held in the web.xml file

```
..
<context-param>
    <param-name>sex</param-name>
    <param-value>female</param-value>
</context-param>
<context-param>
    <param-name>age</param-name>
    <param-value>21</param-value>
</context-param>
```

Remember: when retrieve from a container it is of type object

```
WebMaster w =
(WebMaster ) getServletContext().getAttribute("webmaster");
```

Remember to cast

How does the container know there is a listener?

```
<web-app   …   >
…
<listener>
   <listener-class>
         packagename.scl
   </listener-class>
</listener>
</web-app>
```

The listener class

The need for session tracking: keeping the state of a user over a sequence of requests

- HTTP is a “stateless” protocol
  - each time a client retrieves a page it opens a separate connection with web server
  - no contextual information about the client is stored
- So how can we manage
  - shopping carts and checkout facilities?
  - personalised pages (e.g., customer portfolios)?
- That is, how does the server know to associate old client information with a new request?

Ways to perform session tracking

1. Cookies
   - check for cookie in request header
   - create a cookie and send back to client
2. URL-rewriting
   - attach a *session id* to end of URLs, e.g.,

Defined later

http://host/path/file.html;sessionid=1234

3. Hidden form fields
   - put hidden fields in generated forms, e.g.,

<input type="hidden" name="session" value="1234" ...>

Works if interact through forms, but not if use hyperlinks <A href=...

Setting Cookies

- The `javax.servlet.http` API contains a Cookie class to get and set cookies and their attributes

- But we won't look at cookies as the servlet package also has full support for session tracking making use of cookies unnecessary.

  (and you've covered them already)

Session tracking in servlets

- The `HTTPSession` API is a high-level interface built on top of cookies or URL-rewriting
  - most servers use cookies if the browser supports them, otherwise revert to URL-rewriting
  - transparent to the servlet author
- NB. It's important to encode all URLs that reference your site (in case URL-rewriting is used)
- Do this routinely, even if the servlet being written isn't using session tracking (it might one day…)

Session tracking in servlets

- Session Tracker uses a **session ID** to match users up with **Session objects** on the server side
  - session ID is a string created and then sent as a cookie to the browser when user first accesses server.

    JSESSIONID=0ABC5019DE56
- Sometimes cookies will not work, e.g.
  - if browser does not support cookies
  - User has disabled cookies
- Session Tracker then resorts to URL rewriting
  - Tracks the users session by including the session ID in all URLs the users will communicate with the server **application**

HTTPSession objects

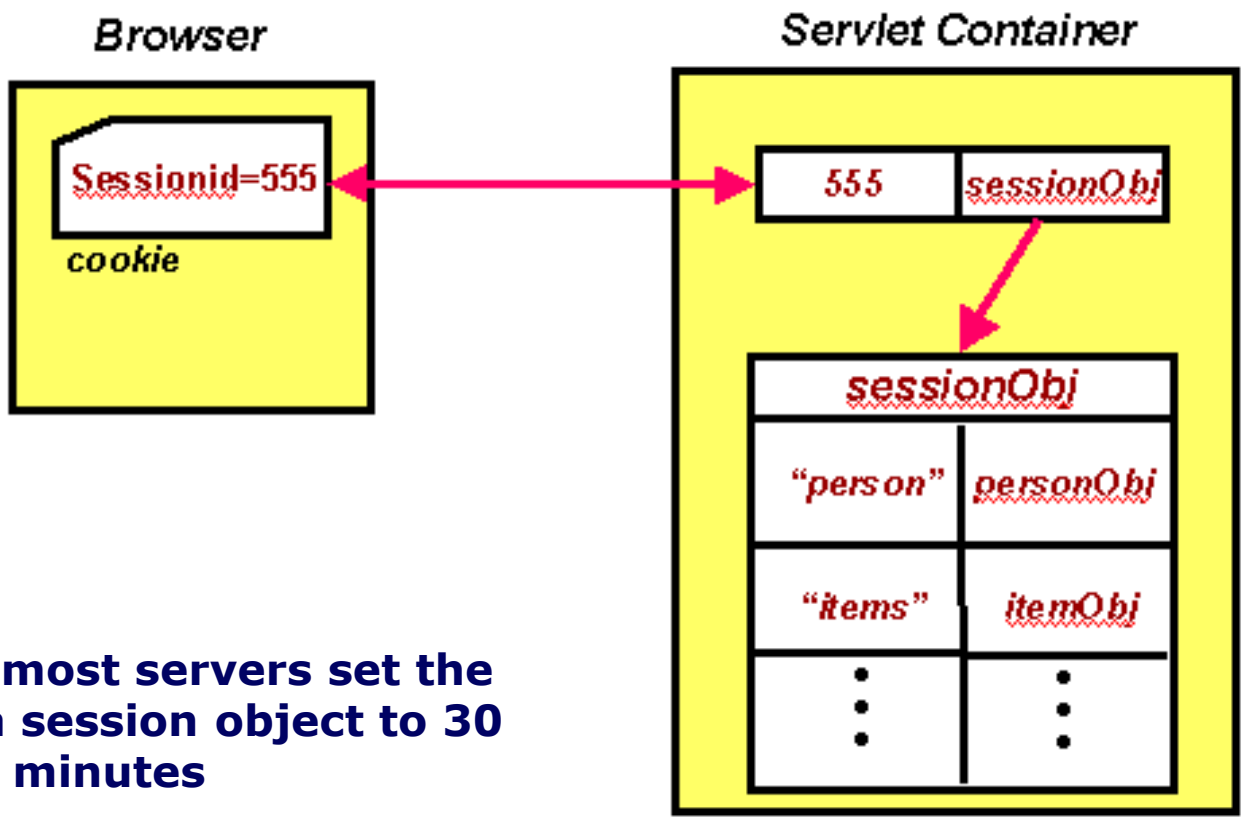
- The servlet engine keeps a table of HttpSession objects
  - Objects are found from the table using the "session id" as the key
  - This is extracted from the request object
    - a cookie or rewritten URL

EXTRACT OR CREATE A NEW SESSION OBJECT

```
HttpSession session=request.getSession()
```

- The HttpSession objects are themselves hash tables that are used to store data during the "session" with the user.

HTTPSession objects



By default, most servers set the lifetime of a session object to 30 minutes

HttpSession
session=request.getSession()

if (request includes a session ID cookie)

then extract session object with that ID

else create a new session object

- Extracts the "session id" from the request and looks up the table & returns the session object
- If no session ID found in an incoming cookie or attached URL then a new empty session object is created
  - AND also creates a cookie called JSESSIONID with a unique value (if it can)
  - Sessions are normally based on in memory cookies, not persistent cookies
    - i.e. the cookies usually have no expiry date
  - Don't have to set the Set-Cookie header
    - All done for us

Looking up the Session object for current request - alternatives

- Use the method

```
HTTPSession req.getSession()
HTTPSession req.getSession(true)
```

  - both always return a session object : either existing or new
    - Same, just saves typing
- Must be called *before* sending any document to client
- can check if just created using `boolean isNew()`
  - `session.isNew()`
  - `Strictly: true if the client has not (yet) responded with a sessionID to the web app.`

Looking up the Session object for current request - alternatives

```
HTTPSession session req.getSession(false)
```

Returns session object, if it exists already

null, if no session object exists

Looking up information stored in the session object

- Can store any information in the session object using attribute-value pairs
  - attribute is a String
  - value is an Object, so need to provide a cast on retrieval
- Methods are

```
session.setAttribute(String key, Object value)
Object session.getAttribute(String key)
```

  - returns null if no such attribute
  - Also removeAttribute(key)

Queen Mary University of London

Simple Session Tracking

```
public class Barman extends HttpServlet {

    public void doGet(HttpServletRequest
request,  HttpServletResponse response)
        throws IOException, ServletException
    {
HttpSession session = request.getSession(true);
Integer count =
(Integer)session.getAttribute("mycounter");
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

Simple Session Tracking

```
out.println("<html>");
out.println("<body bgcolor=\"#FFAACC\">");

if (count == null) {
        count = new Integer(0);
        out.print("<h1>Welcome to the 'Lilac Tree'. Please
enter your name</h1>");
         out.print("<form>");
         out.print("<input name=whoyouare>");
         out.print("</form>");
         }
```

Simple Session Tracking

```
else {
      String wanted =
            request.getParameter("whoyouare");
       if (wanted != null) {
              session.setAttribute("who",wanted);
                } else {
                    wanted =
              (String)session.getAttribute("who");
                          }
       count = new Integer(count.intValue() + 1);
       out.print("<h1>Welcome back
"+wanted+"</h1>");
       out.println("This is your visit no.
"+count+"<br>");
            }
```

Simple Session Tracking

```
session.setAttribute("mycounter",count);
        out.println("</body>");
        out.println("</html>");
    }
}
```

```
session.setAttribute("mycounter",count)
                    ;
            out.println("</body>");
            out.println("</html>");
                    }
                    }
```

Big Changes in Web Architectures

- https://en.wikipedia.org/wiki/Single-page_application
- **Thin server architecture (logic moved to client)**
  - A single fully loaded in the initial page load and then page regions are replaced or updated with new page fragments loaded from the server on demand.(AJAX)
- **Thick stateful server architecture**
  - The server keeps the necessary state in memory of the client state of the page
- **Thick stateless server architecture**
  - variant of the stateful server approach. The client page sends data representing its current state to the server, usually through AJAX requests