Java Servlet

Presenter: Tam Phan

Introductions

- Your role
- Your background, experiences
- What do you want from this course

Course Objectives

At the end of the course, you will have acquired sufficient experience to:

- Understand about Servlet
- Explore features of Servlet API
- Understand Servlet Filter
- Write a simple servlet

Agenda

- Session 1:
  - Fundamental HTML
  - What is Java Servlet?
  - What is Servlet Container?
  - Relationship between Container and Servlet
  - Structure Deployment
  - Hello World Servlet
- Session 2:
  - Servlet Model
  - Session Management
  - Handle Server-side Exceptions
  - Using Filter
- Assignment: one week

Course Prerequisite

- The following are prerequisites:
  - Java base
  - Web programming

Assessment Disciplines

- Class Participation: 30%
- Assignment: 50%
- Final Exam: 50%
- Passing: 70%

Course Timetable

- Duration: 4 hrs
- Break 15 minutes

Further References

- servlet-3_1-final.pdf
- SCWCDExamStudyKit.pdf
- http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html

Course Environment

- Your PC must install software from: \\Qc-training\Shared folder\Servlet_Course
  - JDK 1.5 or later
  - TOMCAT 6.0 or later
  - Eclipse

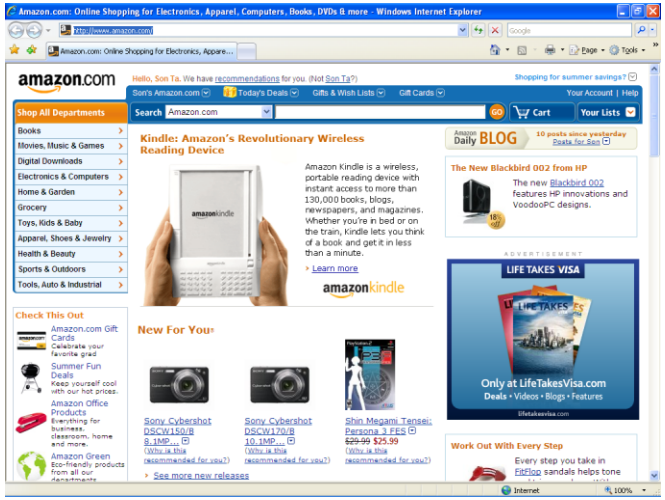
Fundamental HTML

Fundamental HTML

(http://www.amazon.com/)

request

response



Request Header

Headers Sent	Value
(Request-Line)	GET /images/G/01/gno/images/general/navAmazonLogoFooter._V28232323_.gif HTTP/1.1
Accept	*/*
Accept-Encoding	gzip, deflate
Accept-Language	en-us
Connection	Keep-Alive
Host	g-ecx.images-amazon.com
Referer	http://www.amazon.com/
UA-CPU	x86
User-Agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)

Fundamental HTML

- HTML tags are used to mark-up HTML **elements**
- HTML tags are surrounded by the **two characters < and >**
- The surrounding characters are called **angle brackets**
- HTML tags normally **come in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The text between the start and end tags is the **element content**
- HTML tags are **not case sensitive,** means the same as <B>

Basic HTML Tags

- HTML Document: <html>
- Body: <body>
- Headings: <h1>, <h2>, <h3>, <h4>, <h5>
- Paragraphs: <p>
- Line Breaks:
- Comments: <!-- comments -->
- Form: <form>
- Controls:
  - <Input>
  - <textarea>
  - <Select>

```
<html>
    <head><title>Login page</title></head>
    <script langage="javascript">
        function validateForm(){
        var myForm = document.forms['Hello'];
        if(myForm.userName.value == ''){
            alert('Please enter your User Name!');
            return false;
        }
        if(myForm.password.value == ''){
            alert('Please enter your Password!');
            return false;
        }
        return true;
        }
    </script>
    <body>
        <form action="LoginServlet" name="Hello" onsubmit="return validateForm();">
        <table align="center" border=1>
            <tr>
                <td>UserName:</td>
                <td><input name="userName" type="text" value=""></td>
            </tr>
            <tr>
                <td>Password:</td>
                <td><input name="password" type="text" value=""></td>
            </tr>
            <tr>
                <td colspan=2 align="center"><input type=submit name=submit value="submit"></td>
            </tr>
        </table>
        </form>
    </body>
</html>
```

Javascript block

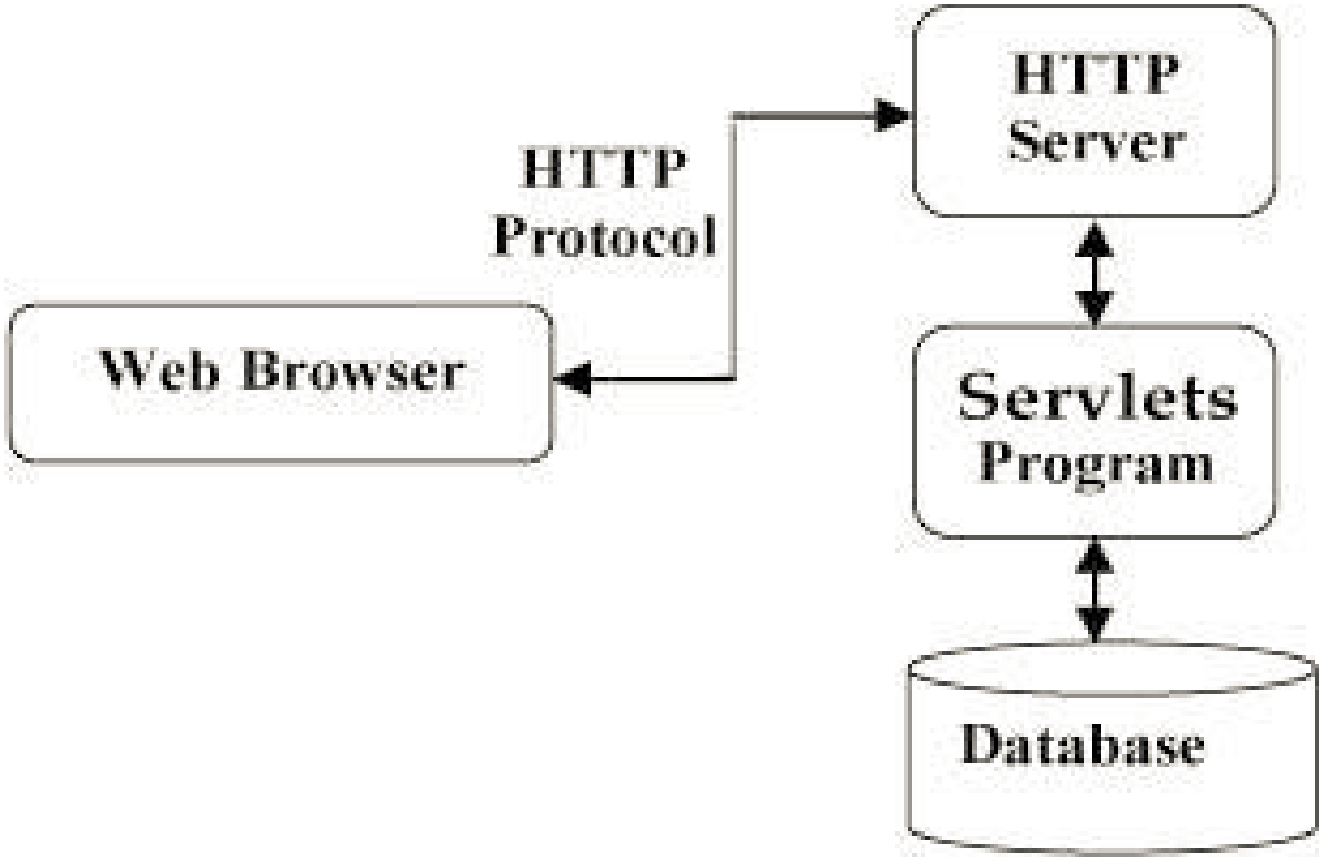
Servlet action

What is Java Servlet?

What is Java Servlet?

- A Java class like any other normal Java class
- A class that implements the javax.servlet.Servlet interface or extends javax.servlet.http.HttpServlet

What is Java Servlet?

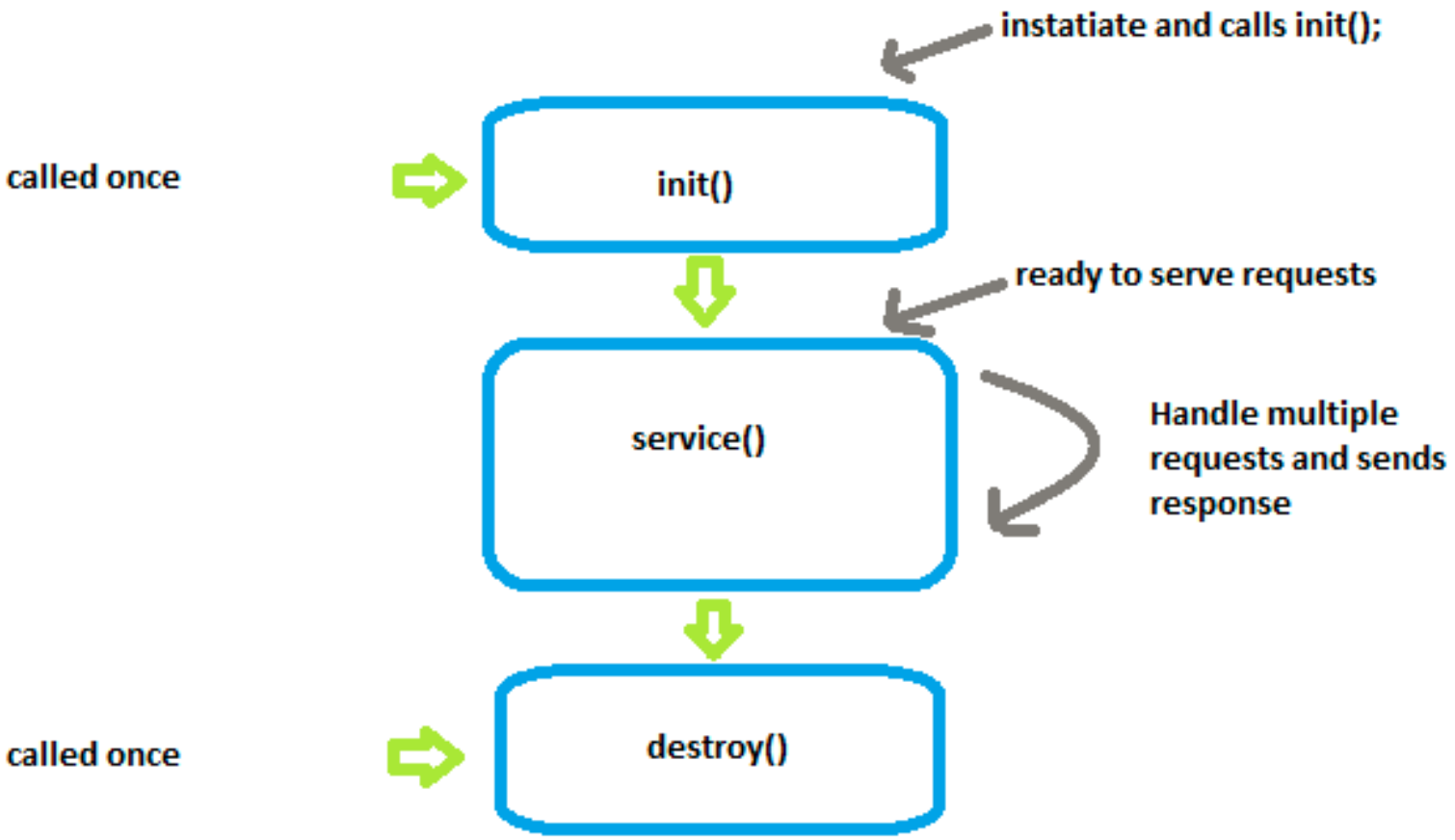


What is Java Servlet?

- Servlets perform the following major tasks:
  - Read the explicit data sent by the clients (browsers).
  - Read the implicit HTTP request data sent by the clients (browsers)
  - Process the data and generate the results.
  - Send the explicit data (i.e., the document) to the clients (browsers)
  - Send the implicit HTTP response to the clients (browsers)

Servlets Life cycle

Servlets Life cycle



Servlets Life cycle

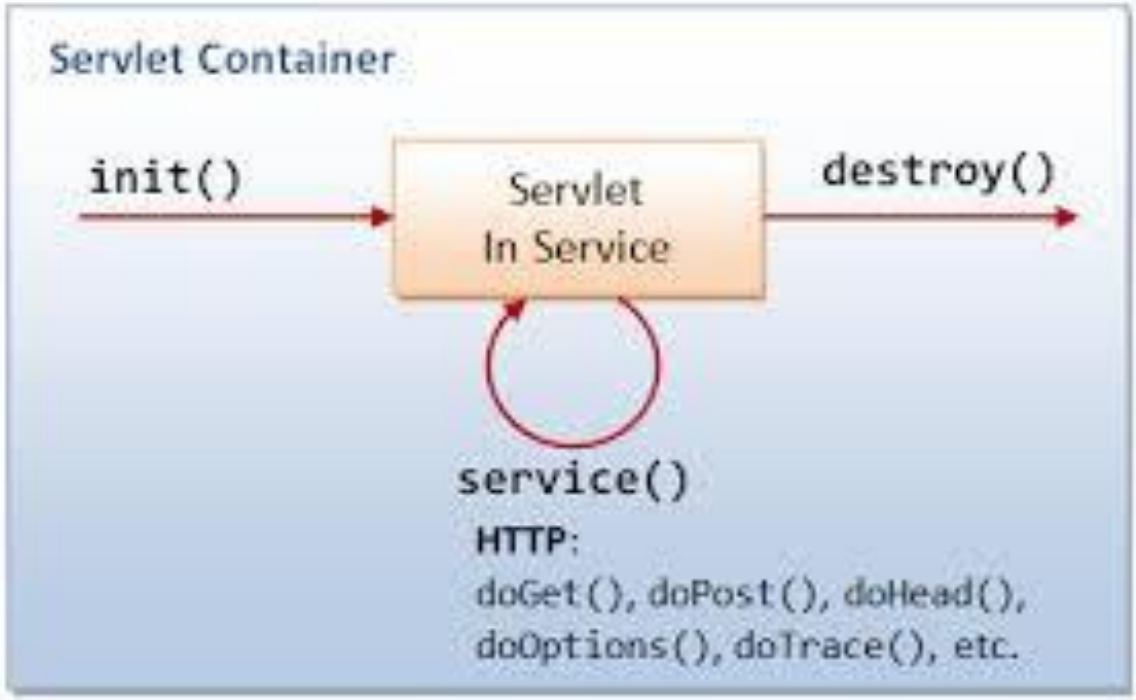
init()

- Is designed to be called only once.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread.
- Simply creates or loads some data that will be used throughout the life of the servlet.

```
public void init() throws ServletException
 { // Initialization code... }
```

Servlets Life cycle

service()




```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException { }
```

Servlets - Form Data

Get method

- Default method
- Sends the encoded user information appended to the page request
- The page and the encoded information are separated by the ? Character
- Servlet handles this type of requests using **doGet()** method

http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI

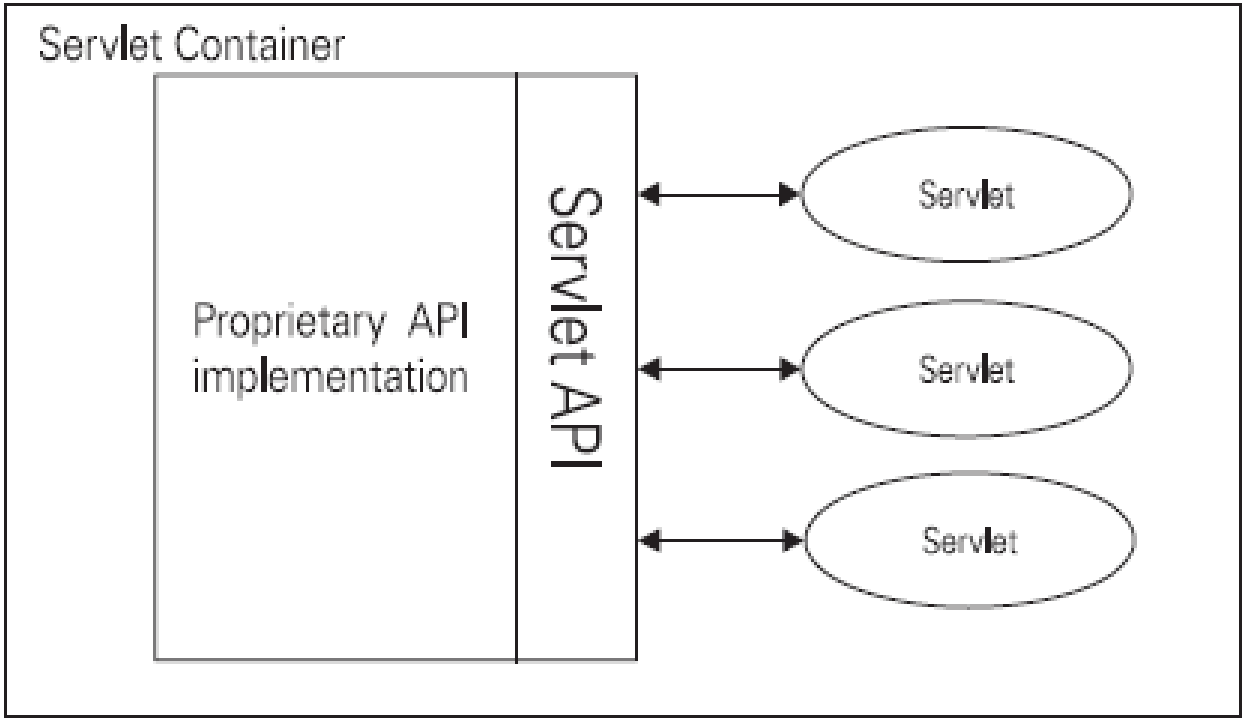
Post method

- instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input

Relationship between Servlet Container and Servlet API

Servlet API

- Servlets and Container communicate with each other via a set of Classes and Interfaces called Servlet API



Servlet API

- Divided into 2 separate packages:
  - **The javax.servlet package**
  - **The javax.servlet.http package**

The javax.servlet package

- Contains the generic servlet interfaces and classes that are independent of any protocol:
  - ***javax.servlet.Servlet***
  - ***javax.servlet.GenericServlet (extends javax.servlet.Servlet)***
  - ***javax.servlet.ServletRequest***
  - ***javax.servlet.ServletResponse***

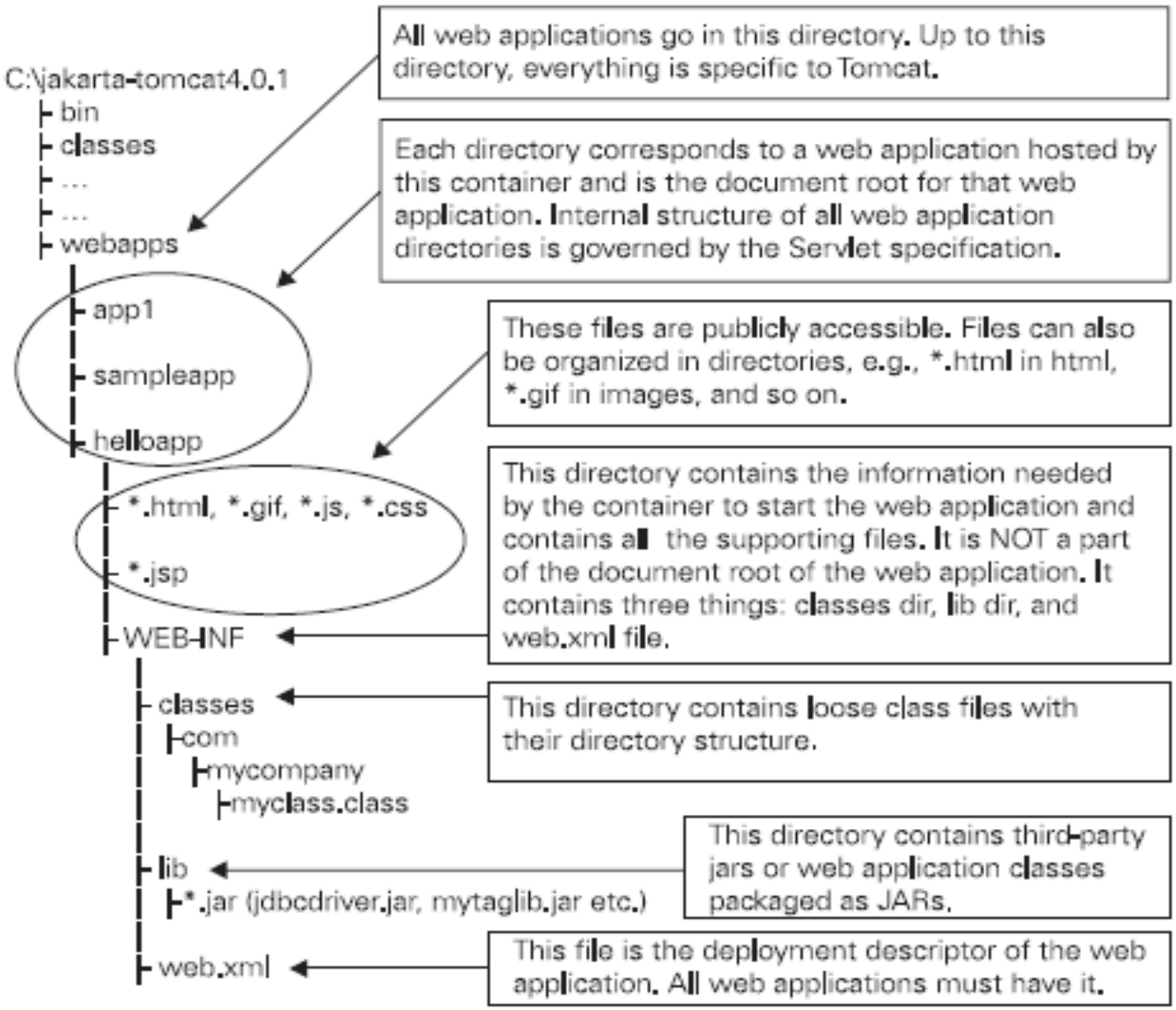
The javax.servlet.http package

- Provides the basic functionality required for HTTP servlets
- Interfaces and classes in this package extend the corresponding interfaces and classes of the javax.servlet package

  – ***javax.servlet.http.HttpServlet***
  – ***javax.servlet.http.HttpServletRequest***
  – ***javax.servlet.http.HttpServletResponse***

Structure and deployment

Structure


All web applications go in this directory. Up to this directory, everything is specific to Tomcat.
C:\jakarta-tomcat4.0.1
bin
classes
...
...
webapps
Each directory corresponds to a web application hosted by this container and is the document root for that web application. Internal structure of all web application directories is governed by the Servlet specification.
app1
sampleapp
helloapp
These files are publicly accessible. Files can also be organized in directories, e.g., *.html in html, *.gif in images, and so on.
*.html, *.gif, *.js, *.css
*.jsp
This directory contains the information needed by the container to start the web application and contains all the supporting files. It is NOT a part of the document root of the web application. It contains three things: classes dir, lib dir, and web.xml file.
WEB-INF
classes
com
mycompany
myclass.class
This directory contains loose class files with their directory structure.
lib
*.jar (jdbcdriver.jar, mytaglib.jar etc.)
This directory contains third-party jars or web application classes packaged as JARs.
web.xml
This file is the deployment descriptor of the web application. All web applications must have it.

Understanding the WEB-INF directory

- Every web application must have a WEB-INF directory directly under its root directory
- Files in the WEB-INF directory are not served to the clients
- This directory contains three things:
  - *classes directory*
  - *lib directory*
  - web.xml

THE DEPLOYMENT DESCRIPTOR

Table 5.1 Properties defined in a deployment descriptor

Web Application Properties	Short Description
Servlet Declarations	Used to specify servlet properties.
Servlet Mappings	Used to specify URL to servlet mapping.
Application Lifecycle Listener classes	Used to specify listener classes for HttpSession-Events and ServletContextAttributeEvent.
ServletContext Init Parameters	Used to specify initialization parameters for the web application.
Error Pages	Used to specify error pages for error conditions.
Session Configuration	Used to specify session timeout.
Security Constraints	Used to specify security requirements of the web application.
Tag libraries	Used to specify the tag libraries required by JSP pages.
Welcome File list	Used to specify the welcome files for the web application.
Filter Definitions and Filter Mappings	Used to specify the filter.
MIME Type Mappings	Used to specify MIME types for common file extensions.
JNDI names	Used to specify JNDI names of the EJBs.

THE DEPLOYMENT DESCRIPTOR

- **Using the <servlet> element**

```
<servlet>

   <servlet-name>us-sales</servlet-name>      <-- The servlet name

   <servlet-class>com.xyz.SalesServlet</servlet-class>   <-- The servlet class

   <init-param>                                  |
      <param-name>region</param-name>            |  The servlet
      <param-value>USA</param-value>             |  parameters
   </init-param>                                 |

   <init-param>                                  |
      <param-name>limit</param-name>             |
      <param-value>200</param-value>             |
   <init-param>

</servlet>
```

THE DEPLOYMENT DESCRIPTOR

- ***servlet-name:*** This element defines the name for the servlet
  - http://www.myserver.com/servlet/us-sales.
- ***servlet-class:*** This element specifies the Java class name that should be used by the servlet container to instantiate this servlet
- ***init-param:*** This element is used to pass initialization parameters to the servlet

THE DEPLOYMENT DESCRIPTOR

- **Using the <servlet-mapping> element**

```
<servlet-mapping>
    <servlet-name>accountServlet</servlet-name>
    <url-pattern>/account/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>accountServlet</servlet-name>
    <url-pattern>/myaccount/*</url-pattern>
</servlet-mapping>
```

Annotation

```
@WebServlet(name="us-sales",
        urlPatterns={"/myurl"},
        initParams={ @InitParam(name="region", value="USA"), @InitParam(name="limit", value="20") })
public class SalesServlet extends javax.servlet.http.HttpServlet {
        ....
}
```

THE DEPLOYMENT DESCRIPTOR

- url-pattern*:* A servlet container interprets the url-pattern according to the following rules:
  - A string beginning with a / and ending with the /* characters is used for determining a *servlet path* mapping
  - A string beginning with *. prefix is used to map the request to a servlet that handles the extension specified in the string
  - All other strings are used as exact matches only
  - A string containing only the / character indicates that servlet specified by the mapping becomes the default servlet of the application

Rule 1

- A string beginning with a / and ending with the /* characters is used for determining a *servlet path* mapping

<servlet-name>BlueServlet</servlet-name>

<url-pattern>/blue/</url-pattern>

</servlet-mapping>

Or

<servlet-name>RedBlueServlet</servlet-name>

<url-pattern>/red/blue/*</url-pattern>

</servlet-mapping>

Rule 2

- A string beginning with a *. prefix is used to map the request to a servlet that handles the extension specified in the string

```
<servlet-name>PdfServlet</servlet-name>
<url-pattern>*.pdf</url-pattern>
</servlet-mapping>
```

Rule 3

- All other strings are used as exact matches only

```
<servlet-mapping>
<servlet-name>GreenServlet</servlet-name>
<url-pattern>/green</url-pattern>
</servlet-mapping>
```

Rule 4

- A string containing only the / character indicates that servlet specified by the mapping becomes the default servlet of the application

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

Rule 4

- A string containing only the / character indicates that servlet specified by the mapping becomes the default servlet of the application

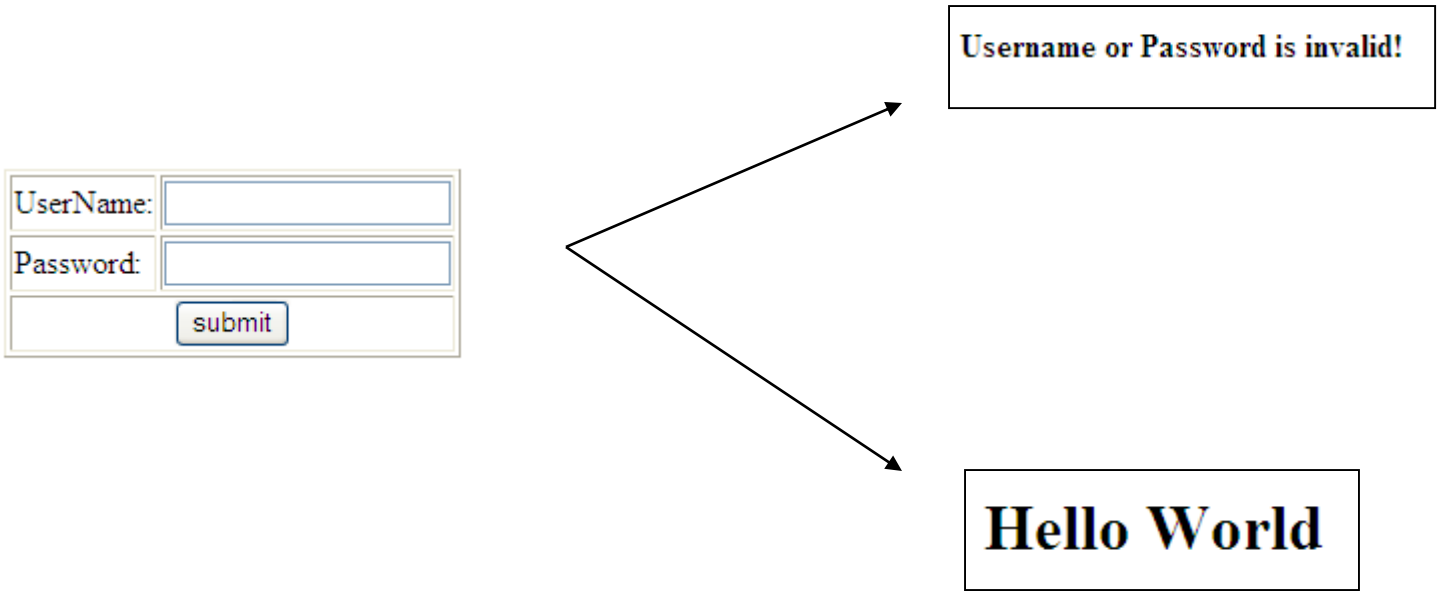
/*	http://example.com/contextPath
	http://example.com/contextPath/status/abc
/status/abc/*	http://example.com/contextPath/status/abc
	http://example.com/contextPath/status/abc/mnp
	http://example.com/contextPath/status/abc/mnp?date=today
	~~http://example.com/contextPath/test/abc/mnp~~
*.map	http://example.com/contextPath/status/abc.map
	http://example.com/contextPath/status.map?date=today
	~~http://example.com/contextPath/status/abc.MAP~~

Pattern Example

/*	http://example.com/contextPath
	http://example.com/contextPath/status/abc
/status/abc/*	http://example.com/contextPath/status/abc
	http://example.com/contextPath/status/abc/mnp
	http://example.com/contextPath/status/abc/mnp?date=today
	~~http://example.com/contextPath/test/abc/mnp~~
*.map	http://example.com/contextPath/status/abc.map
	http://example.com/contextPath/status.map?date=today
	~~http://example.com/contextPath/status/abc.MAP~~

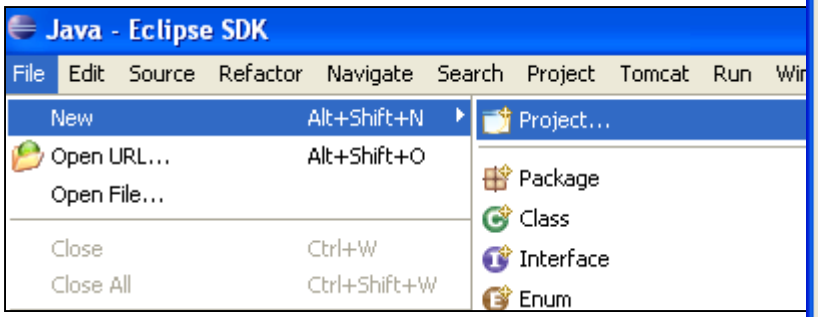
Hello World Servlet

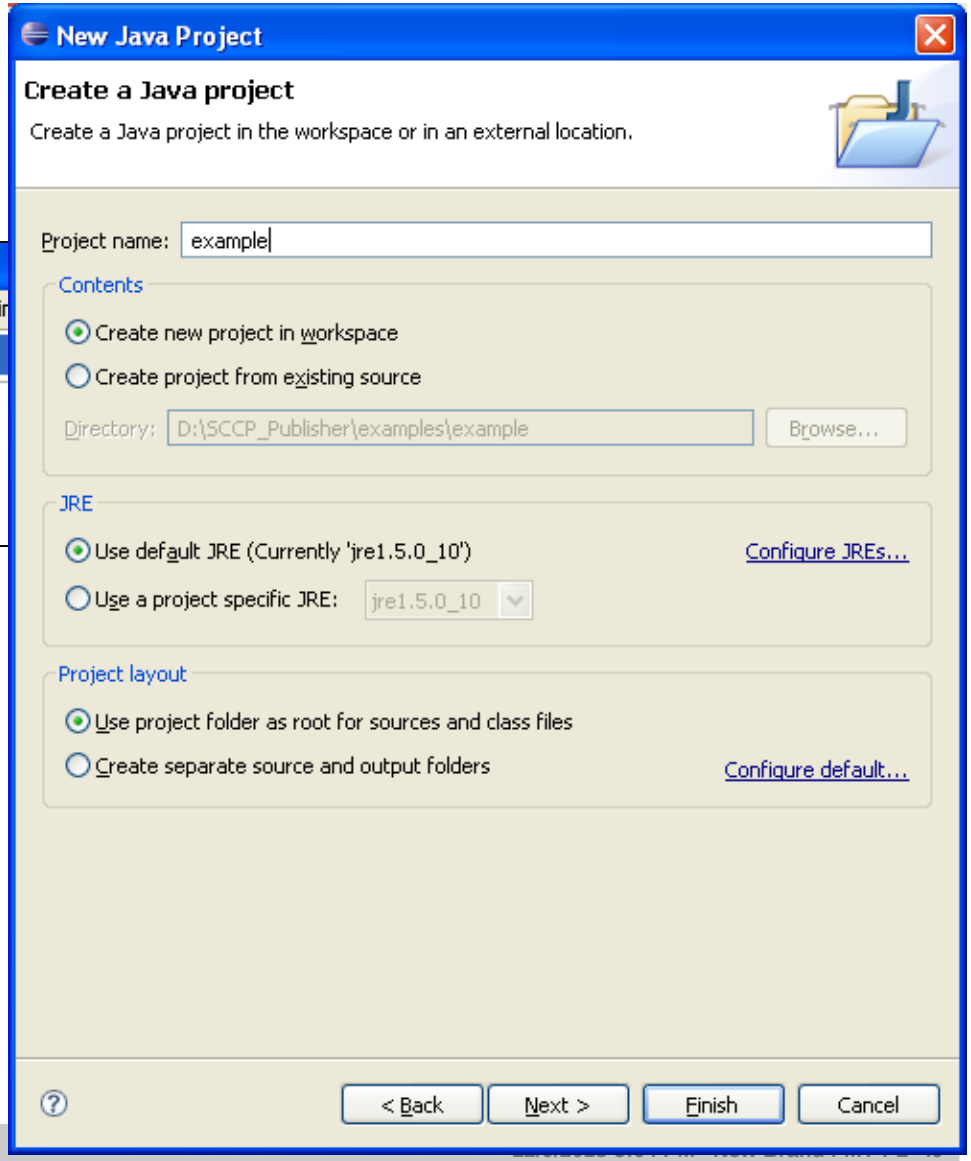
Hello World Servlet



Hello World Servlet

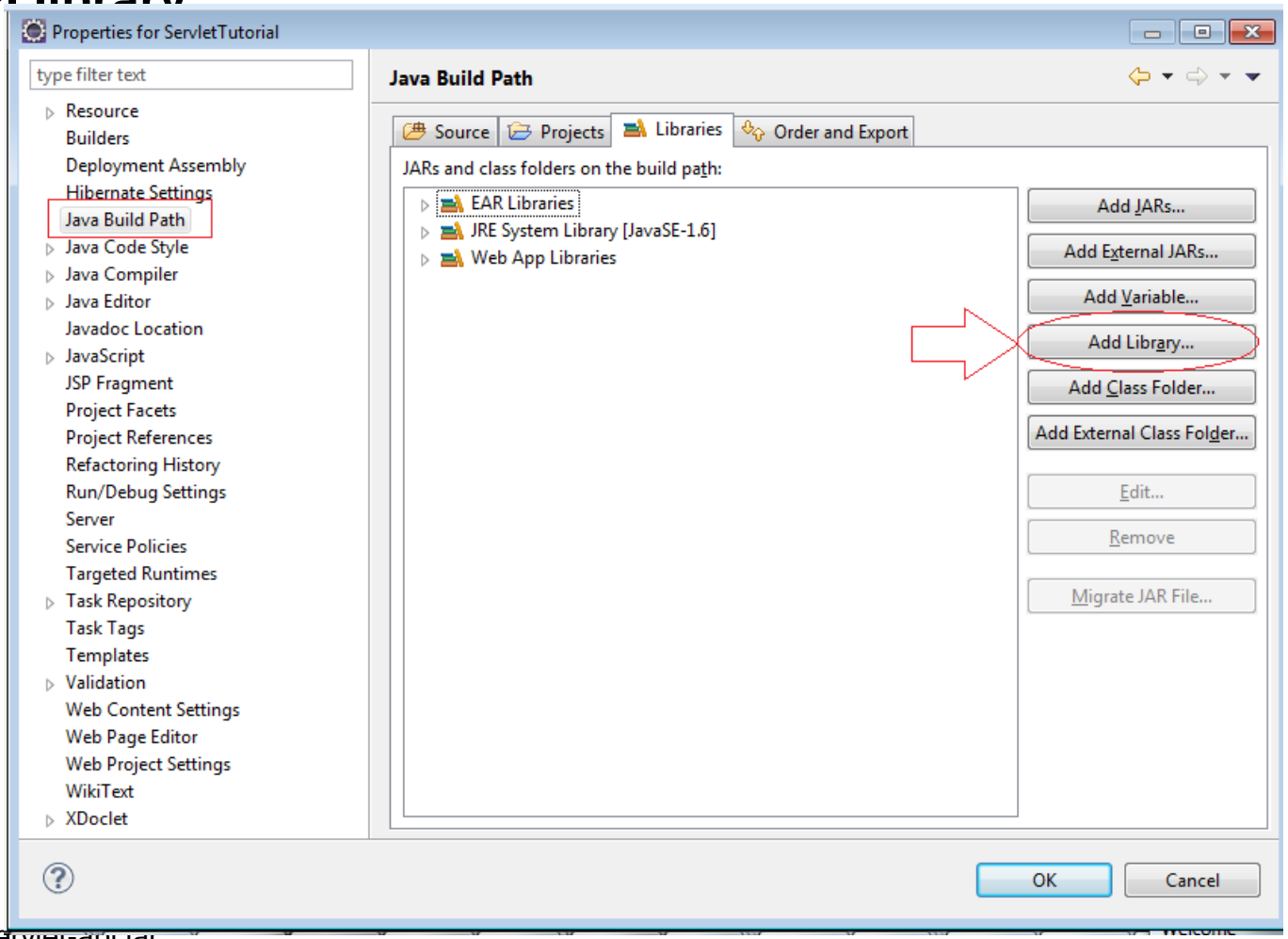
- **Open Eclipse**
- **Create a new Java Project**





Hello World Servlet

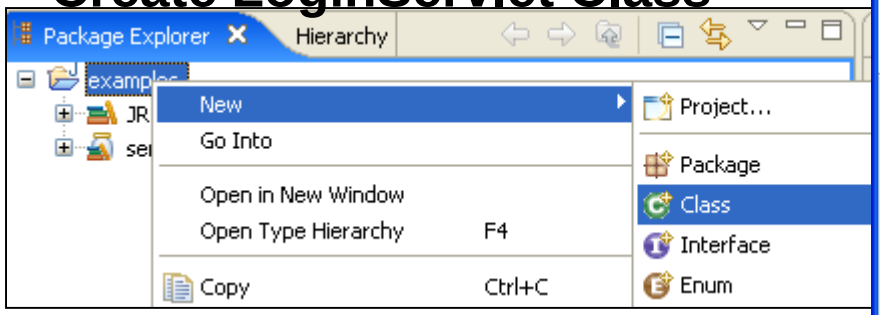
- **Add Servlet API library**

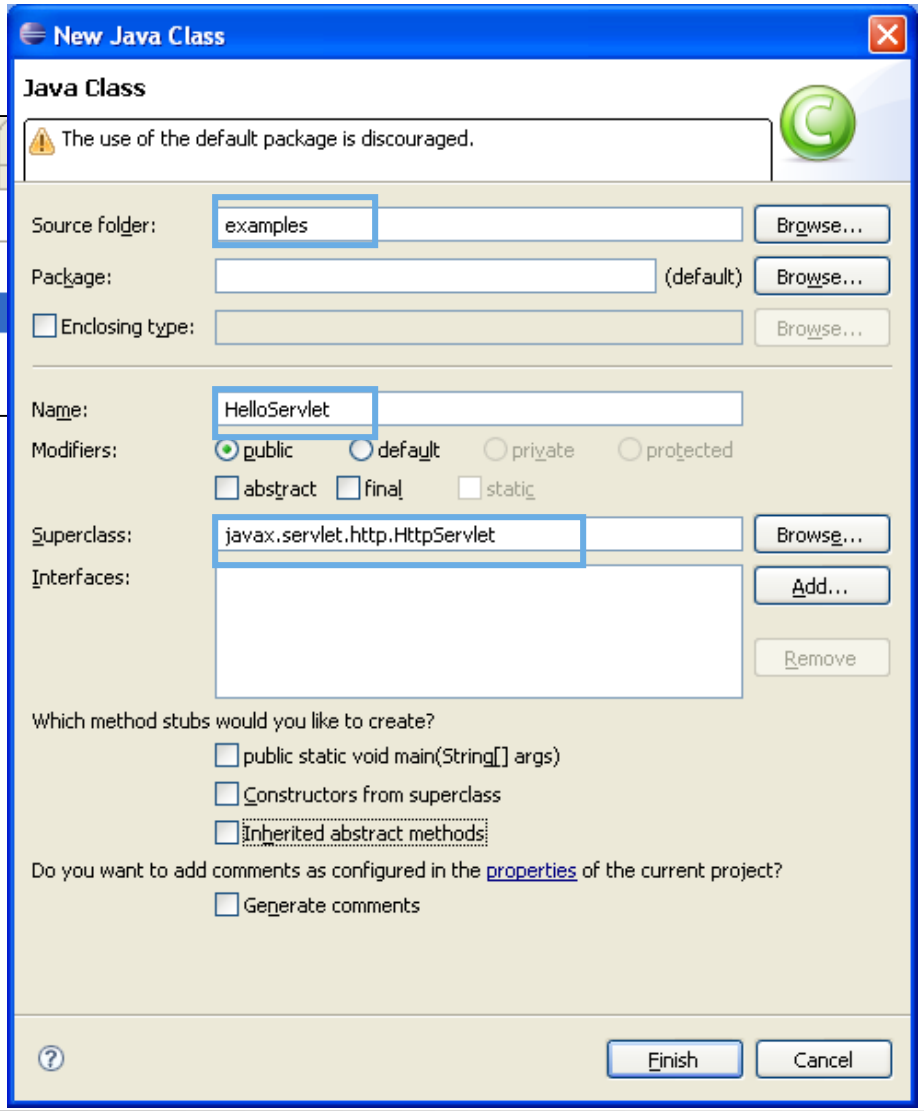


D:\Tomcat 5.5\common\lib\servlet-api.jar

Hello World Servlet

- **Create LoginServlet Class**





Hello World Servlet

*HelloServlet.java

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
    public void init(javax.servlet.ServletConfig servletConfig)
            throws javax.servlet.ServletException {
        System.out.println("Init in servlet...........");
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        //TODO Handle Post request here...
    }

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

Hello World Servlet

- **Add more code…**

```
String userName = request.getParameter("userName");
String password = request.getParameter("password");
if("admin".equals(userName) && "admin".equals(password)){
    response.sendRedirect("loginsuccess.html");
}
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Error Page</title>");
out.println("</head>");
out.println("<body>");
out.println("<b>Username or Password is invalid!</b>");
out.println("</body>");
out.println("</html>");
```

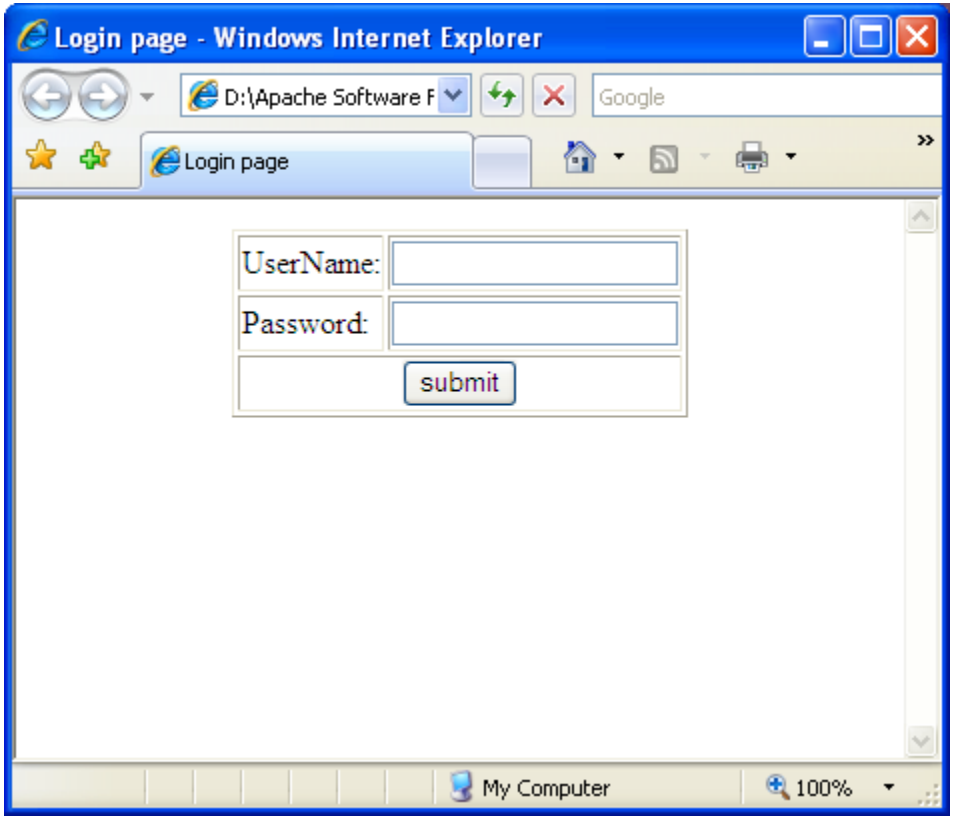
Hello World Servlet

- **Deploy HelloServlet to Tomcat**
  - Copy the HelloServlet.class file to the directory:
    [TOM_CAT]\webapps\firstservlet\WEB-INF\classes
  - Create a text file named web.xml in:
    [TOM_CAT]\webapps\firstservlet\WEB-INF directory
  - Write the following lines in the file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
<servlet>
<servlet-name>LoginServlet</servlet-name>
<servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
</web-app>
```

Hello World Servlet

- Create a login form

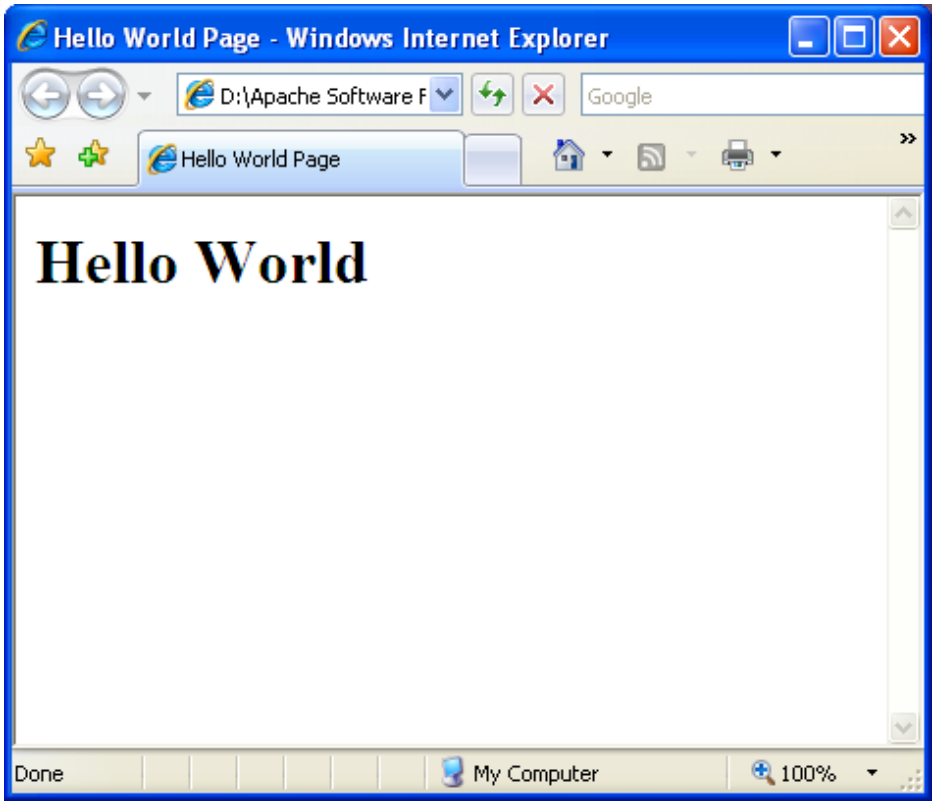


```
<html>
    <head><title>Login page</title></head>
    <script langage="javascript">
        function validateForm(){
        var myForm = document.forms['Hello'];
        if(myForm.userName.value == ''){
            alert('Please enter your User Name!');
            return false;
        }
        if(myForm.password.value == ''){
            alert('Please enter your Password!');
            return false;
        }
        return true;
        }
    </script>
    <body>
        <form action="LoginServlet" name="Hello" onsubmit="return validateForm();">
        <table align="center" border=1>
            <tr>
                <td>UserName:</td>
                <td><input name="userName" type="text" value=""></td>
            </tr>
            <tr>
                <td>Password:</td>
                <td><input name="password" type="text" value=""></td>
            </tr>
            <tr>
                <td colspan=2 align="center"><input type=submit name=submit value="submit"></td>
            </tr>
        </table>
        </form>
    </body>
</html>
```

Hello World Servlet

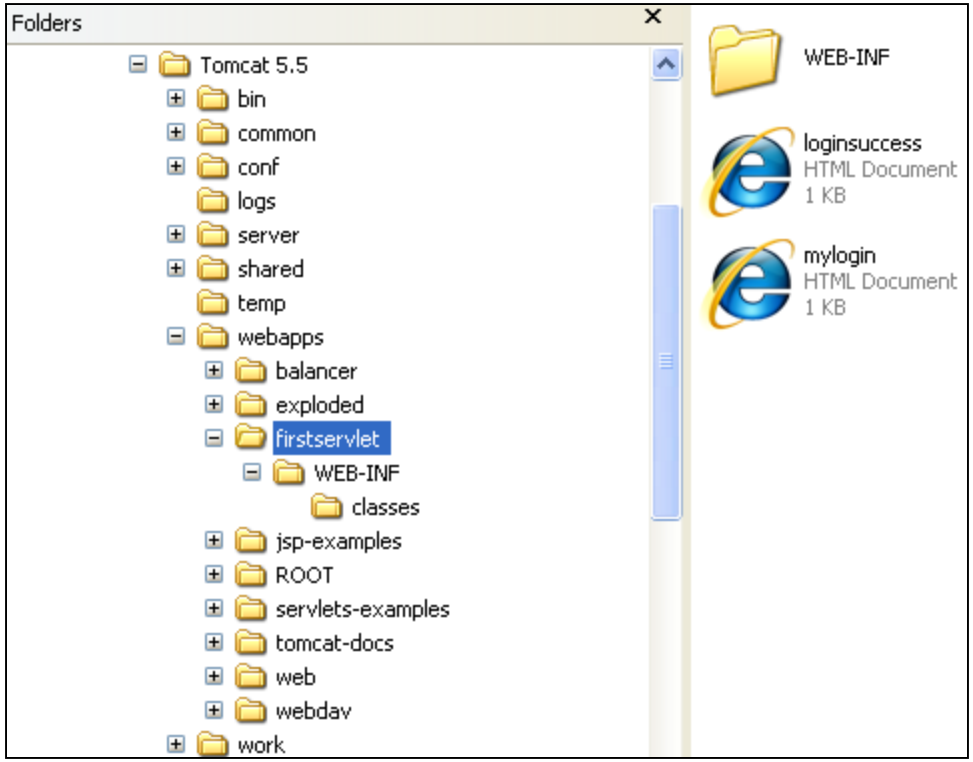
- Create the LoginSuccess page

```
1  <html>
2      <head><title>Hello World Page</title></head>
3      <body>
4          <Hl>Hello World</hl>
5      </body>
6  </html>
7
```



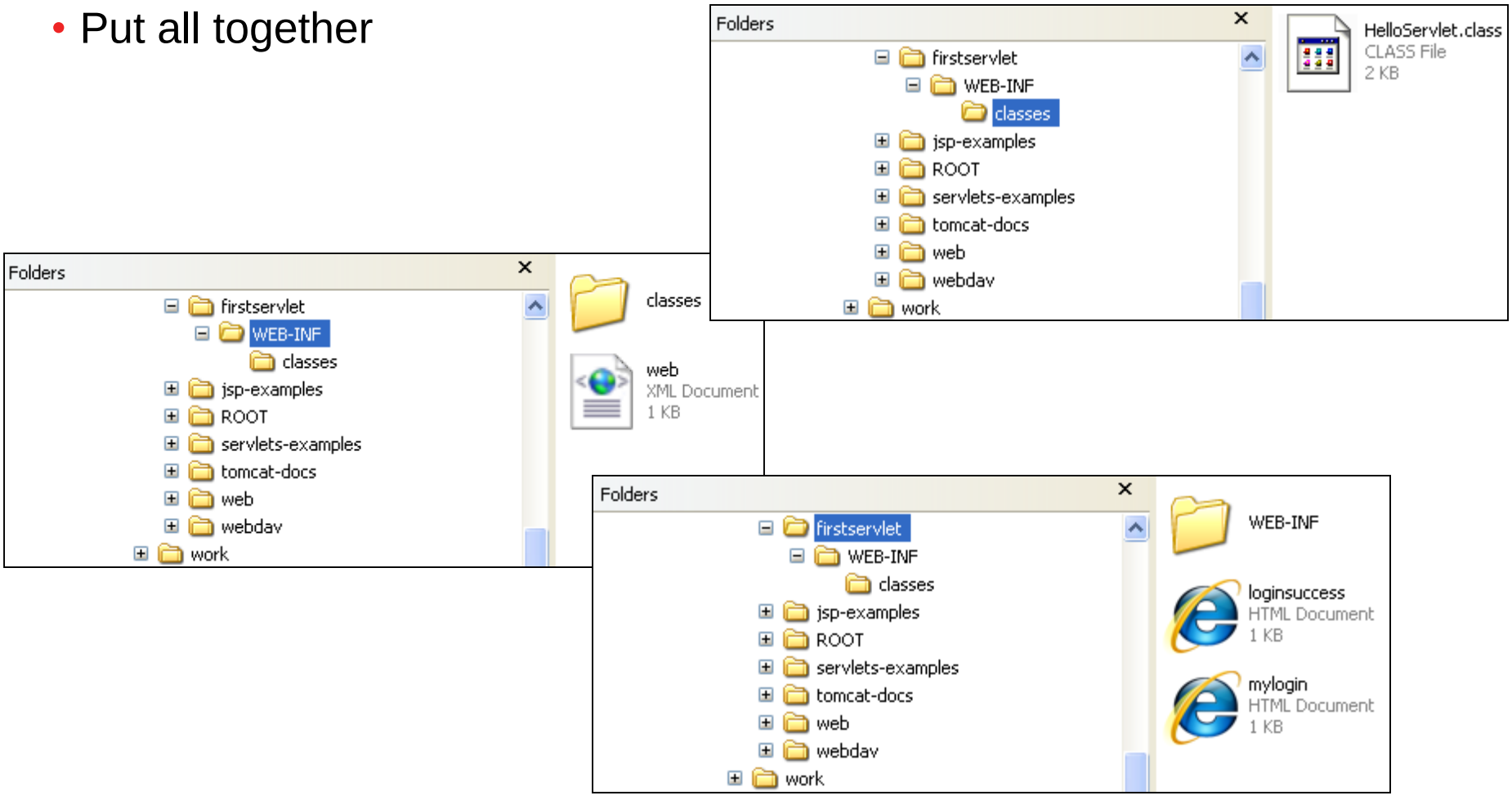
Hello World Servlet

- Copy LoginSuccess.html and MyLogin.html to:
  [TOM_CAT]\webapp\firstservlet\



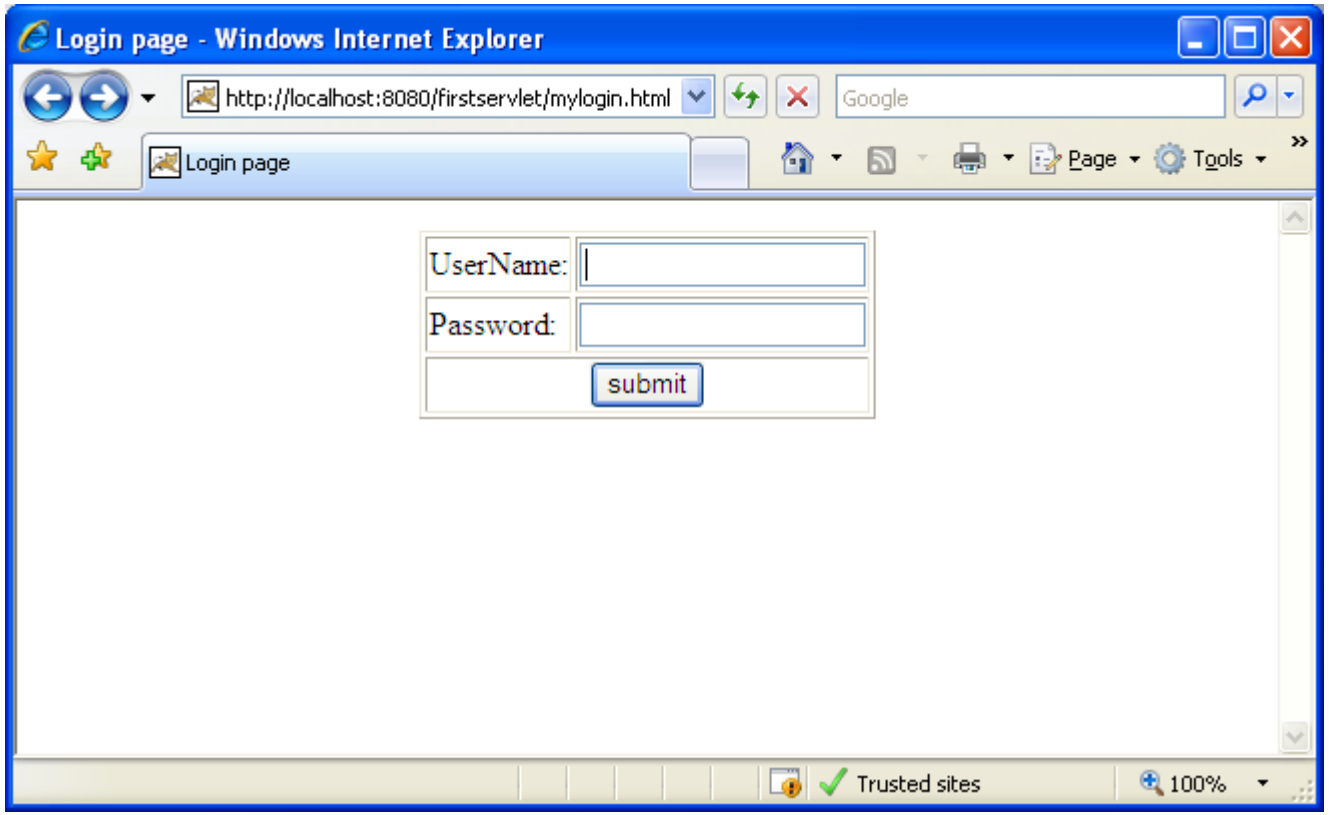
Hello World Servlet

- Put all together



Hello World Servlet

- Start TomCat Server
- Open a browser and enter the address:
  http://localhost:8080/firstservlet/mylogin.html



Servlet Model

Understanding a Request

A web browser sends an HTTP request to a web server when…

- A user clicks on a hyperlink displayed in an HTML page.
- A user fills out a form in an HTML page and submits it.
- A user enters a URL in the browser's address field and presses Enter.

HANDLING HTTP REQUESTS IN AN HTTPSERVLET

For every HTTP method, there is a corresponding method in the HttpServlet class of type:

- GET doGet()
- HEAD doHead()
- POST doPost()
- PUT doPut()
- DELETE doDelete()
- OPTIONS doOptions()
- TRACE doTrace()

Understanding ServletRequest

- String getParameter(String paramName)
- String[] getParameterValues(String paramName)
- Enumeration getParameterNames()

Understanding HttpServletRequest

- Extends from ServletRequest
- Parses and interprets HTTP messages and provides the relevant information to the servlet

Understanding HttpServletRequest

```
<form action="/servlet/TestServlet" method="POST">
Technology : <input type="text" name="searchstring" value="java">
<br><br>
State : <select name="state" size="5" multiple>
   <option value="NJ">New Jersey</option>
   <option value="NY">New York</option>
   <option value="KS">Kansas</option>
   <option value="CA">California</option>
   <option value="TX">Texas</option>
</select>
<br><br>
<input type="submit" value="Search Job">
</form>
```

Uses HTTP POST

Allows selection of multiple values

Understanding HttpServletRequest

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse res)
{
   String searchString = req.getParameter("searchstring");

   String[] stateList = req.getParameterValues("state");

   //use the values and generate appropriate response
}
```

Retrieves searchstring parameter value

Retrieves all the values selected in the state list

Understanding ServletResponse

- ***PrintWriter***
  - *Retrieved by calling* getWriter() of ServletResponse
  - Used to send character data to the client
- ***ServletOutputStream***
  - *Retrieved by calling* getOutputStream() of ServletResponse
  - Used to send a binary file to client (e.g. media, word, PDF, etc…)

Understanding ServletResponse

- Extended from ServletResponse
- Providing capabilities of:
  - Setting response header information
  - Redirecting HTTP requests to another URL
  - Adding cookies to the response

Understanding ServletConfig

- String getInitParameter(String name)
- Enumeration getInitParameterNames()
- ServletContext getServletContext()
- String getServletName()

Understanding ServletConfig

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

  <servlet>                                                    <-- Defines a servlet
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>TestServlet</servlet-class>
    <load-on-startup>1</load-on-startup>                       <-- Allows preloading of the servlet
    <init-param>
       <param-name>driverclassname</param-name>
       <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
    </init-param>
    <init-param>
       <param-name>dburl</param-name>
       <param-value>jdbc:odbc:MySQLODBC</param-value>
    </init-param>
    <init-param>
       <param-name>username</param-name>
       <param-value>testuser</param-value>
    </init-param>
    <init-param>
       <param-name>password</param-name>
       <param-value>test</param-value>
    </init-param>
  </servlet>
```

Defines a parameter and specifies its name and value

Understanding ServletConfig

```
public void init()
{
   System.out.println(getServletName()+" : Initializing...");

   ServletConfig config = getServletConfig();

   String driverClassName =
          config.getInitParameter("driverclassname");

   String dbURL = config.getInitParameter("dburl");
   String username = config.getInitParameter("username");
   String password = config.getInitParameter("password");

   //Load the driver class
   Class.forName(driverClassName);

   //get a database connection
   dbConnection =
       DriverManager.getConnection(dbURL,username,password);

   System.out.println("Initialized.");
}
```

Creates connection in init()

Retrieves parameters

Understanding ServletConfig

- java.net.URL getResource(String path)
- java.io.InputStream getResourceAsStream(String path)

  ~ getResource(String path).openStream()

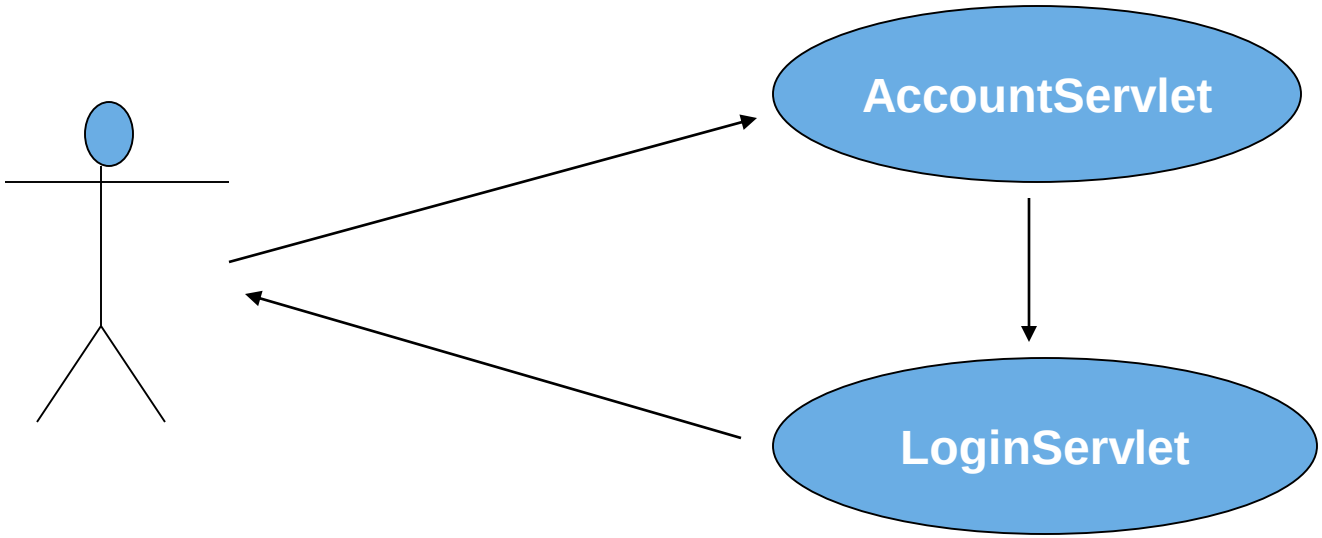
Understanding ServletContext

```
public void service(HttpServletRequest req,
                    HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
   {
      res.setContentType("application/jar");

      OutputStream os = res.getOutputStream();

      //1K buffer
      byte[] bytearray = new byte[1024];

      ServletContext context = getServletContext();
      URL url = context.getResource("files/test.jar");

      InputStream is = url.openStream();

      int bytesread = 0;
      while( (bytesread = is.read(bytearray) ) != -1 )
      {
         os.write(bytearray, 0, bytesread);
      }
      os.flush();
      is.close();

   }
```

Returns a URL object to the file

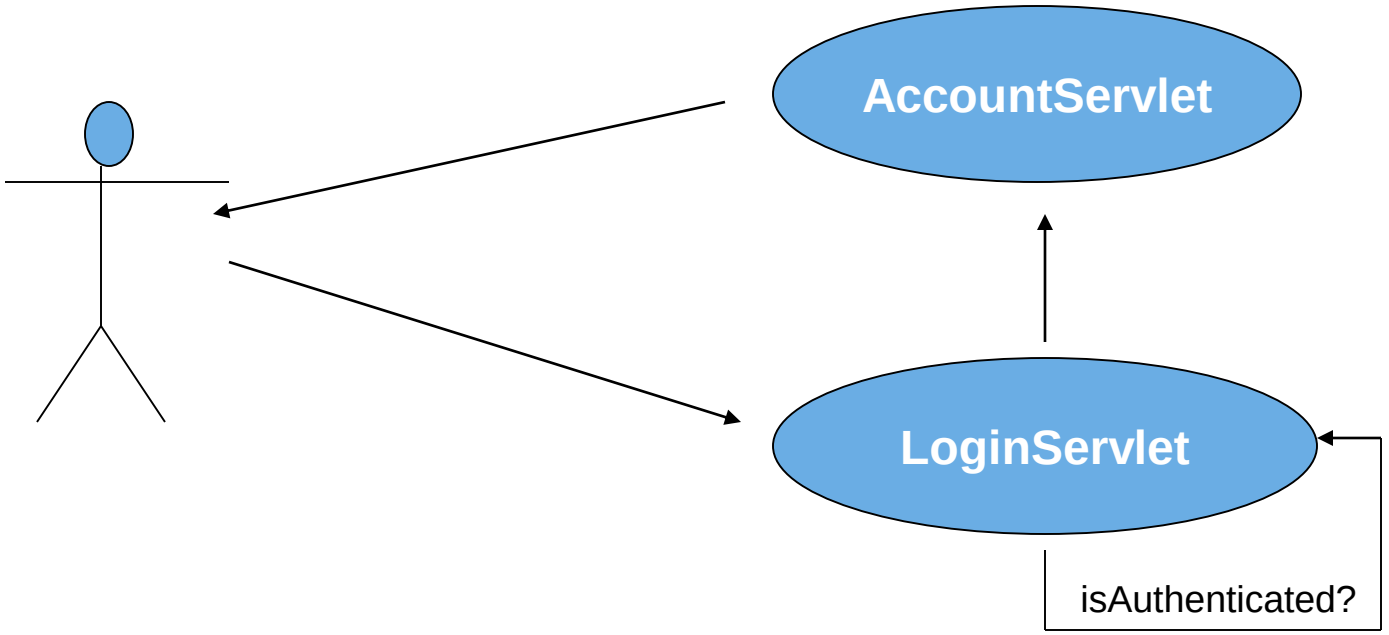
Understanding RequestDispatcher

- Scienario:
  - If a user is not logged in, AccountServlet should forward the request to LoginServlet.



Understanding RequestDispatcher

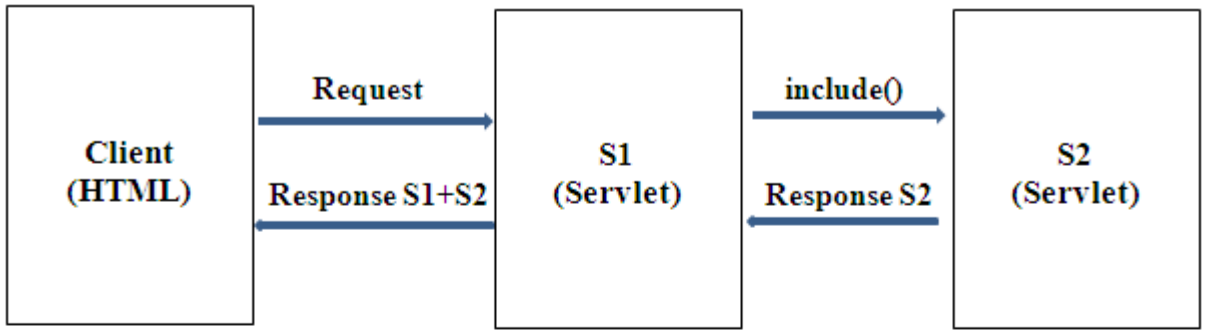
- Scienario:
  - If a user is authenticated, LoginServlet should forward the request to AccountServlet.



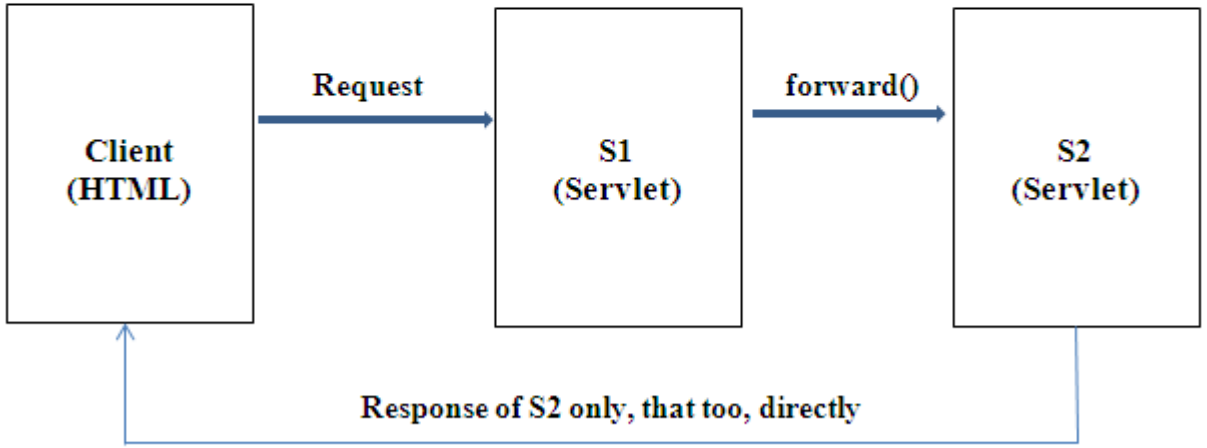
Understanding RequestDispatcher

- Methods provided:
  - void forward(ServletRequest req, ServletResponse res)
  - void include(ServletRequest req, ServletResponse res)

Understanding RequestDispatcher



RequestDispatcher – include() method



RequestDispatcher – forward() method

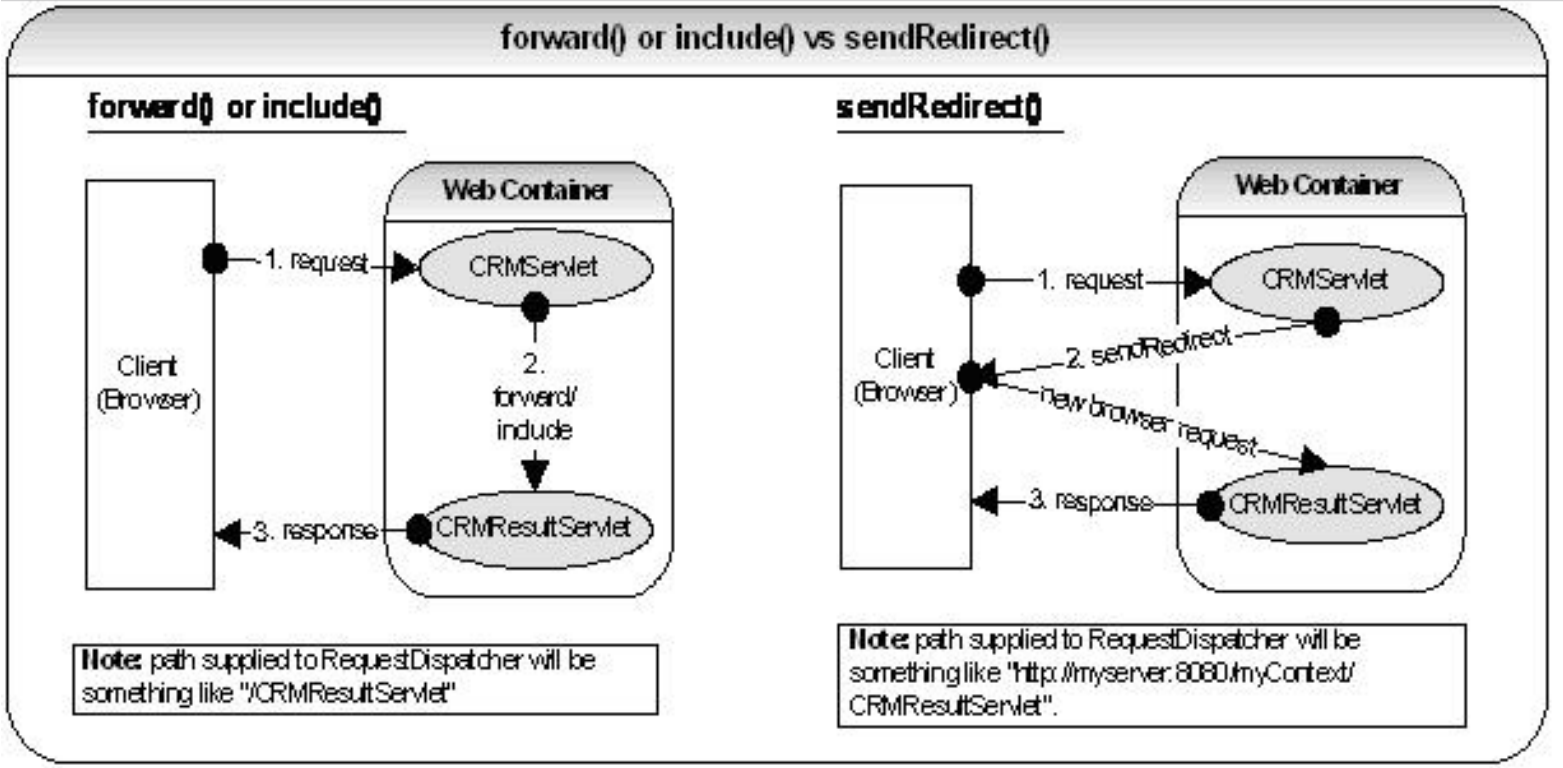
Forward vs. Redirect

- Forward
  - a forward is performed internally by the servlet
  - the browser is completely unaware that it has taken place, so its original URL remains intact
  - any browser reload of the resulting page will simple repeat the original request, with the original URL

Forward vs. Redirect

- Redirect
  - a redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original
  - a browser reload of the second URL will not repeat the *original* request, but will rather fetch the *second* URL
  - redirect is marginally slower than a forward, since it requires two browser requests, not one
  - objects placed in the *original* request scope are not available to the second request

Forward vs. Redirect



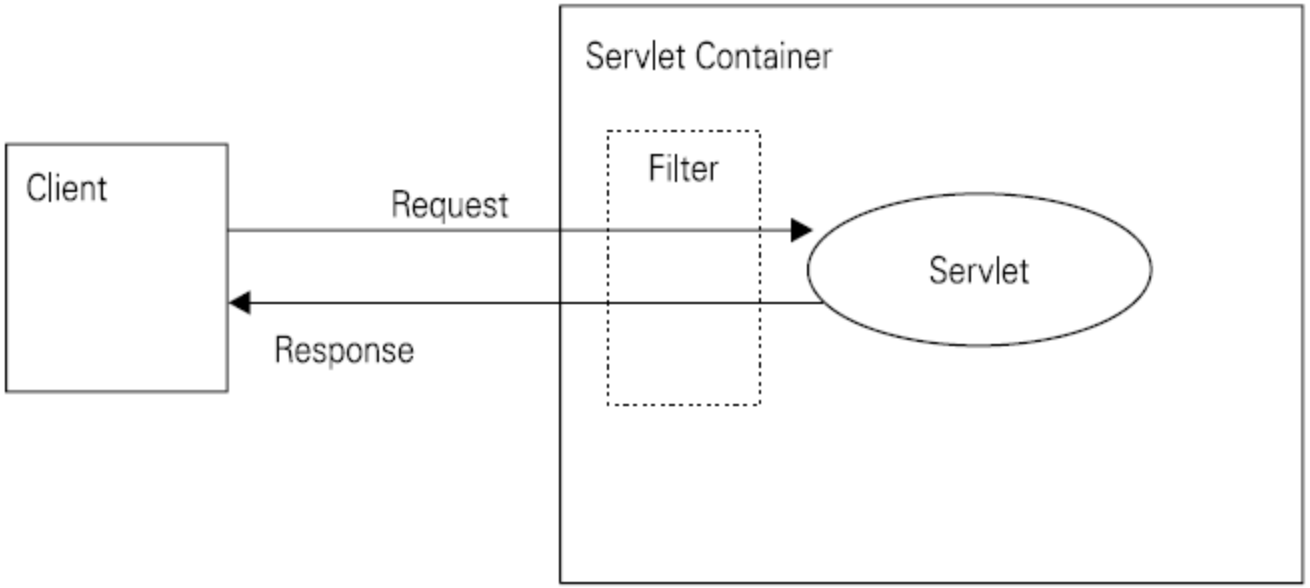
Using Filter

What is a Filter?

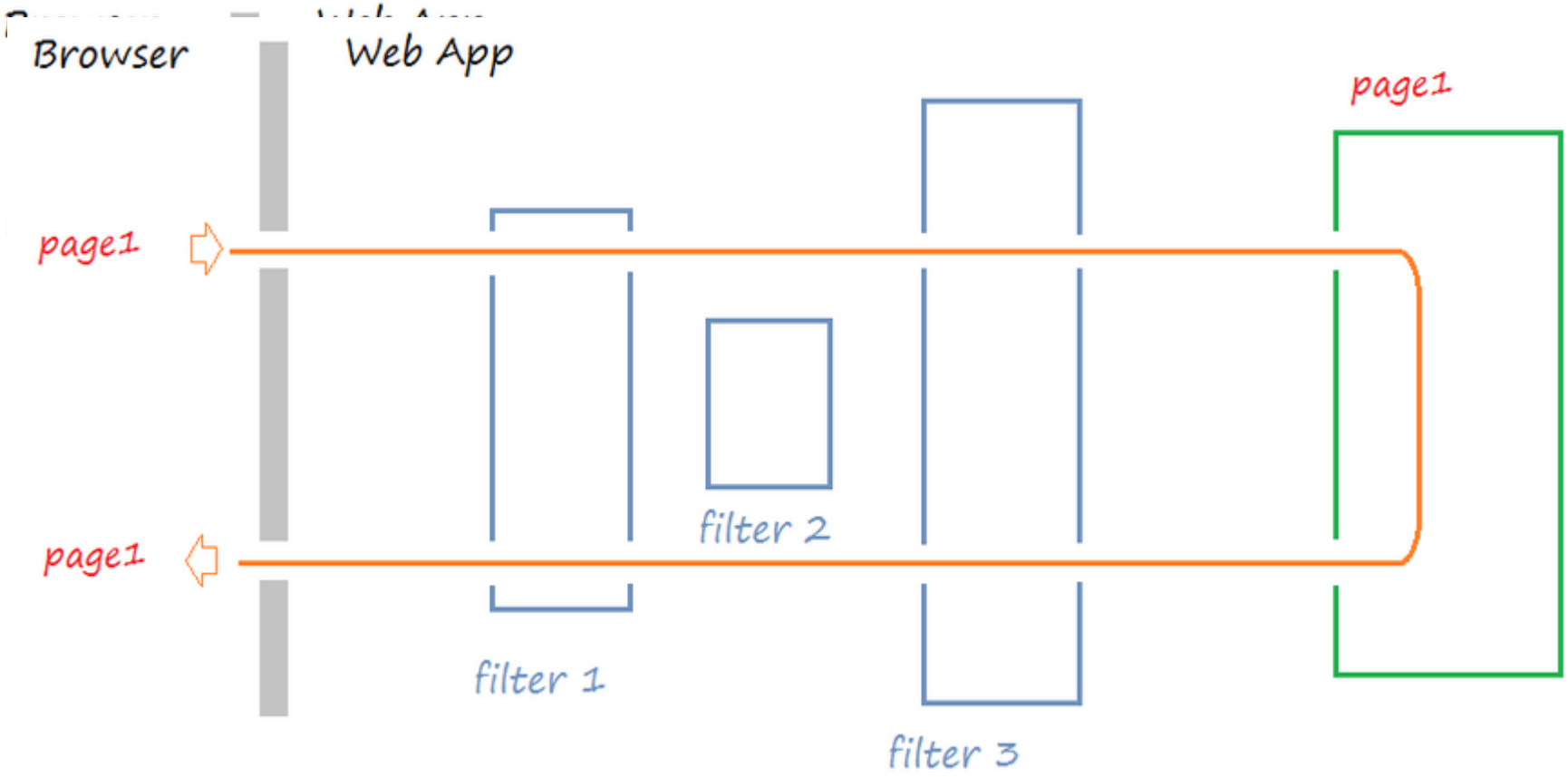
- An object that intercepts a message between a data source and a data destination
- Acts a guard to prevent undesired information from being transmitted from one point to another

What is a Filter?

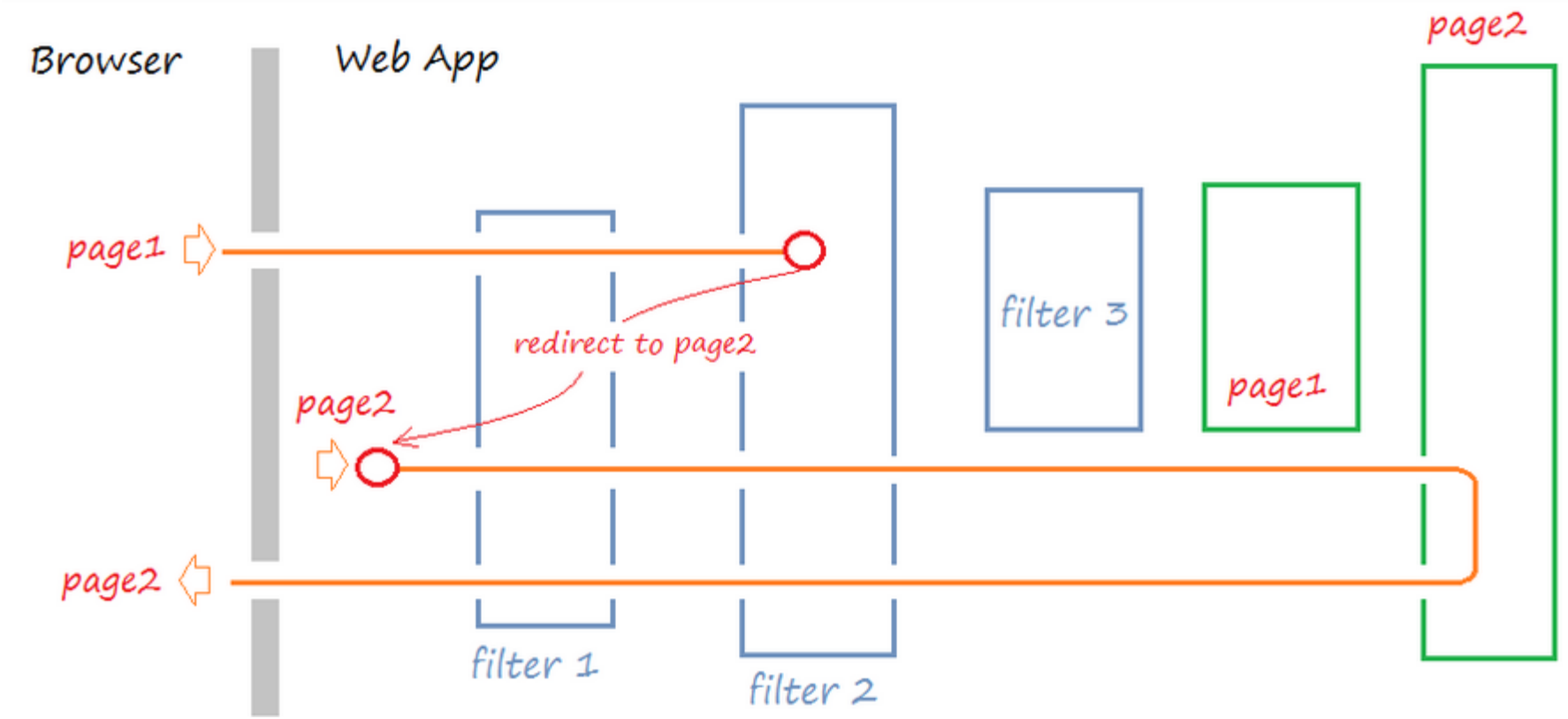
- For web application:
  - It's a web component that resides in the web server
  - It filters the request and response that are passed between a client and a resource



Chain of filters

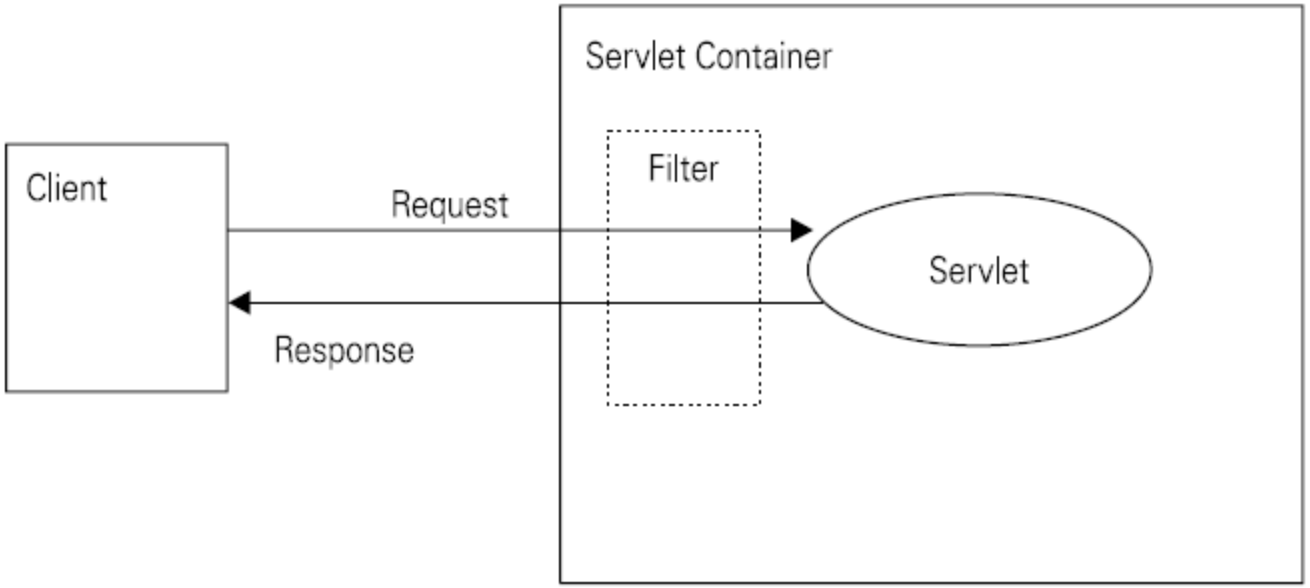


Chain of filters



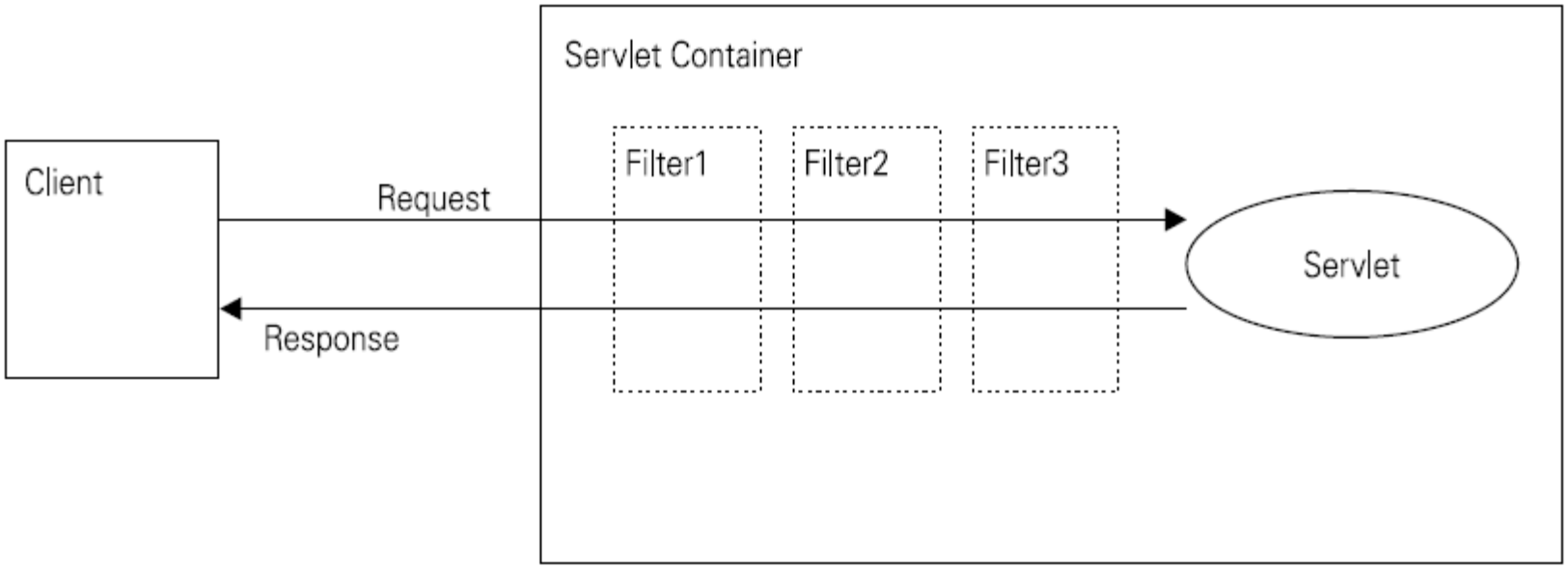
How filtering works?

- Container receives a request for a resource
- Container checks whether a filter is associated with that resource
- Container forwards the request to that filter



How filtering works?

- In turn, the filter will do one of three things:
  - Generates the response itself and returns it to the client
  - Passes on the request (modified or unmodified) to the next filter (if any) or to the designated resource if it is the last filter
  - Routes the request to a different resource



Configure a Filter

- <filter> element

```
<filter>
   <filter-name>ValidatorFilter</filter-name>
   <description>Validates the requests</description>
   <filter-class>com.manning.filters.ValidatorFilter</filter-class>
   <init-param>
      <param-name>locale</param-name>
      <param-value>USA</param-value>
   </init-param>
</filter>
```

Configure a Filter

- <filter-mapping> element

```
<filter-mapping>
    <filter-name>ValidatorFilter</filter-name>
    <url-pattern>*.doc</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>ValidatorFilter</filter-name>
    <servlet-name>reportServlet</servlet-name>
</filter-mapping>
```

A Filter Example

```
public class MyFilter implements javax.servlet.Filter {
    public void destroy() {
    }
  public void doFilter(javax.servlet.ServletRequest req,
  javax.servlet.ServletResponse resp, javax.servlet.FilterChain chain) throws
  javax.servlet.ServletException, java.io.IOException {
        System.out.println("do Filter.....");
        chain.doFilter(req, resp);
    }

    public void init(javax.servlet.FilterConfig config) throws
  javax.servlet.ServletException {
                    System.out.println("Init Filter.....");
    }
}
```

A Filter Example

```
<filter>
        <filter-name>test</filter-name>
        <filter-class>MyFilter</filter-class>
</filter>

<filter-mapping>
        <filter-name>test</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Points to Remember

Points to Remember

- What is Java Servlet?
- What is Servlet Container?
- How to write a Servlet?
- How to redirect a request?
- How to forward a request?
- How to use a Filter?

JSP Overview

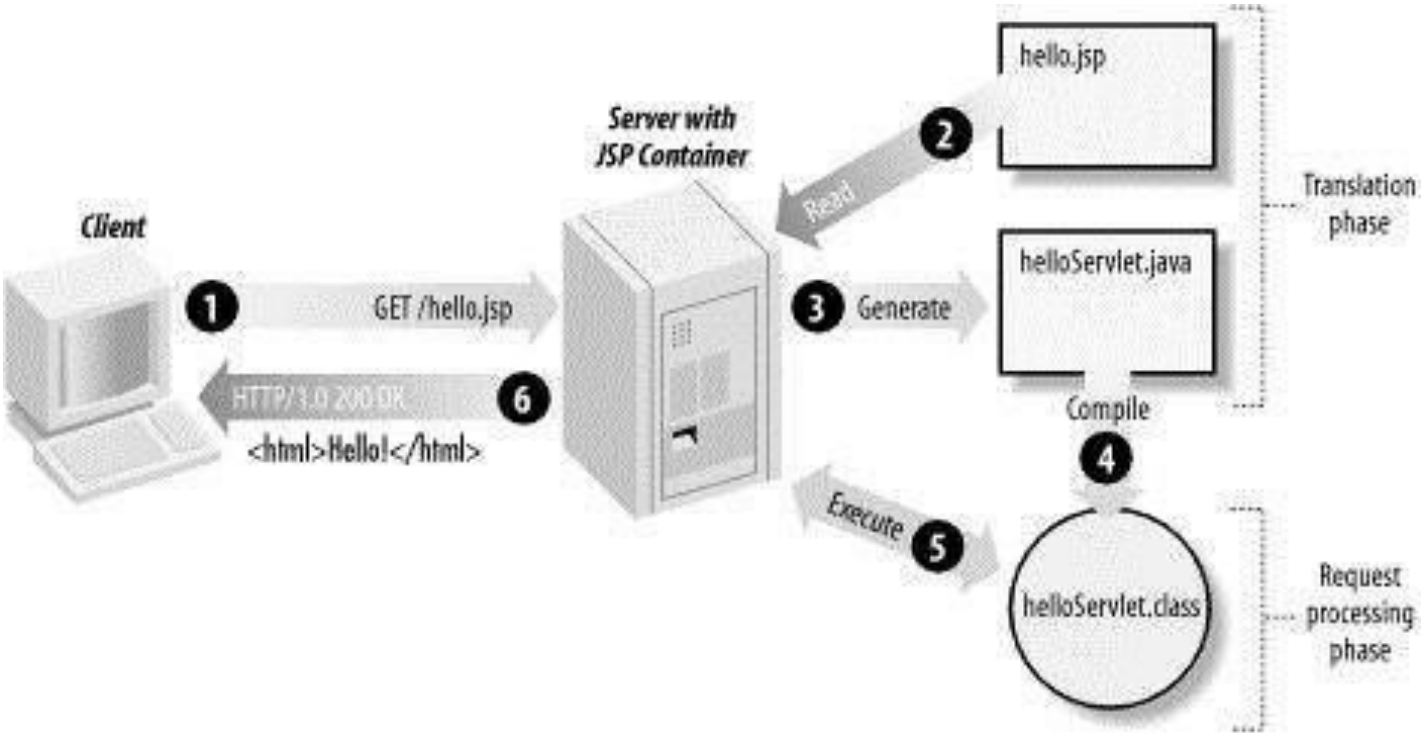
Objectives

- After finished this chapter you can understand :
  - What is the JSP?
  - JSP Architecture
  - JSP Page Semantic
  - Object Scope

What is the JSP?

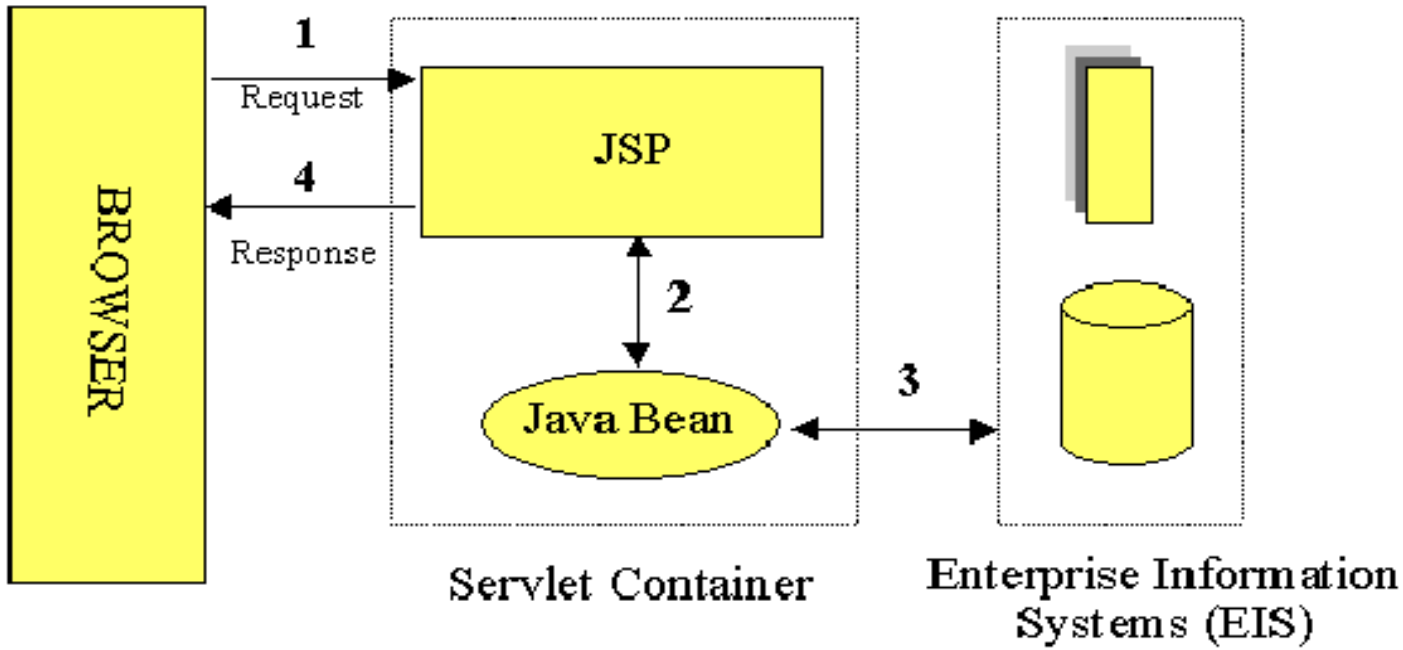
- A JSP (**J**ava **S**erver **P**age) is a web page that includes JSP technology-specific tags, declarations, and possibly scriptlets, in combination with other static (HTML or XML) tags.
- JSP is the Java platform technology for building applications containing dynamic Web content such as HTML, DHTML, XHTML and XML.

JSP Architecture



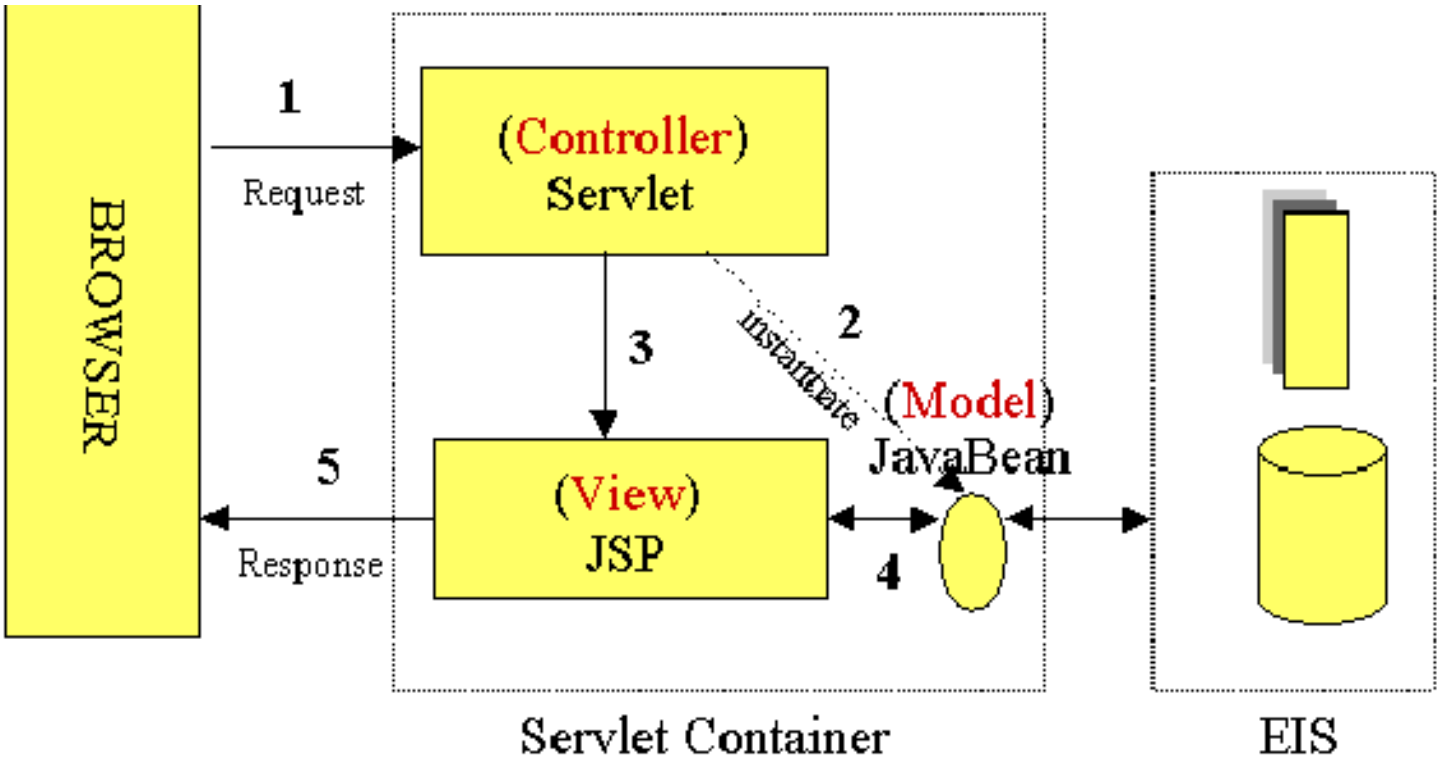
JSP Architecture

- Architecture Model 1



JSP Architecture (Cont)

- Architecture Model 2

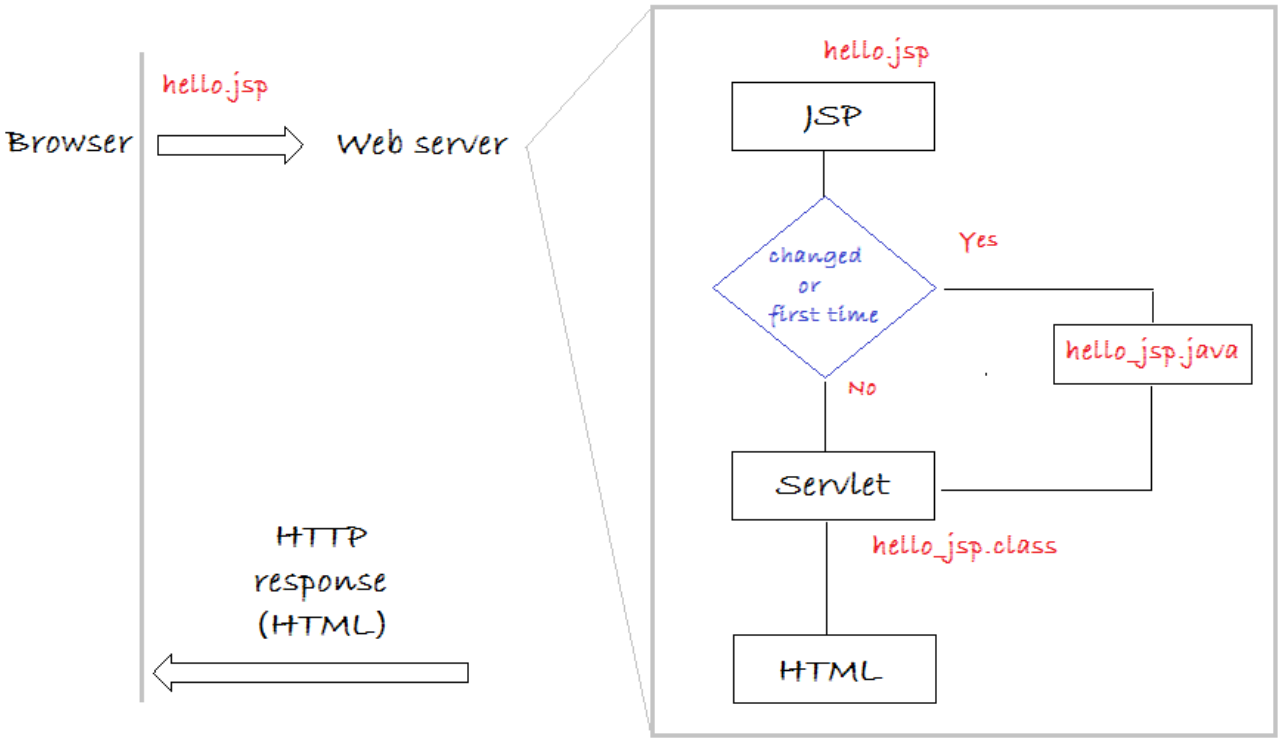


JSP Life Cycle

- Compilation
- Initialization
- Execution
- Cleanup

JSP Container

JSP Life Cycle - Compilation



JSP Container

JSP Life Cycle - Initialization

```
public void jspInit(){
  // Initialization code...
}
```

- initialize database connections
- open files
- create lookup tables in the jspInit method.

JSP Container

JSP Life Cycle - Execution

```
void _jspService(HttpServletRequest request, HttpServletResponse
response)
 { // Service handling code... }
```

- The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request.

JSP Container

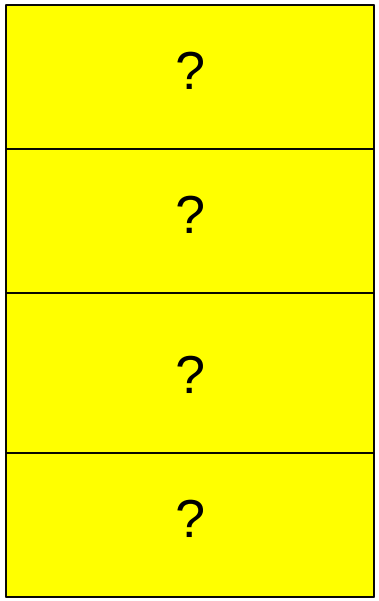
JSP Life Cycle - Cleanup

```
public void jspDestroy() { // Your cleanup code goes here. }
```

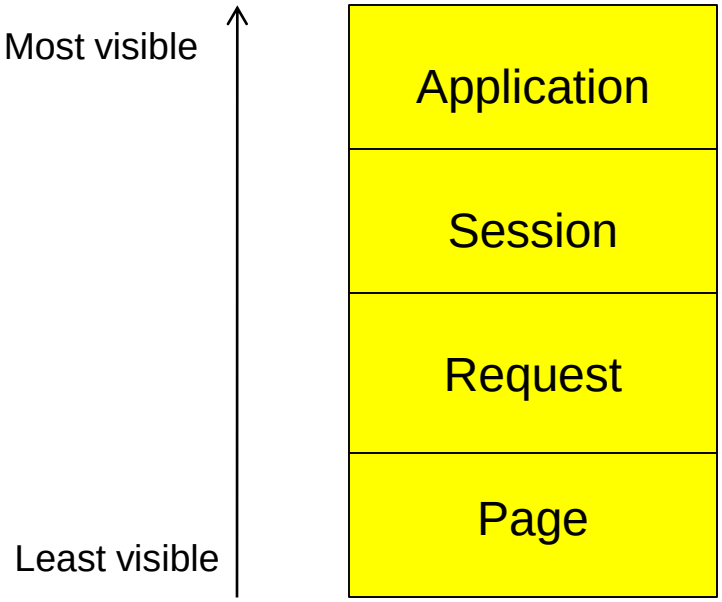
- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

JSP Container

Object Scope



Object Scope



Object accessible to pages that belong to the same application

Object accessible to pages that belong to the same session as the one in which they were created

Object accessible from pages processing the request where they were created

Object accessible only within pages where they were created

JSP Syntax

JSP Structure

- JSP structure is fairly straightforward, and can be classified into directives, declaration,scripting elements and standard actions.
- Example :

```
<%@ declaration directive %>
<%!
  declaration variables and methods
%>
<%
   java code
%>
Template static(HTML)
```

Directive

- JSP directives are messages for the JSP engine. They do not directly produce any visible output, but tell the engine what to do with the rest of the JSP page.
- JSP directives are always enclosed within the <%@ ... %> tag.

Directive (cont)

-Page directive

- The page directive is found at the top of almost all of your JSP pages.
- The attribute/value pair must be unique.
- Unrecognized attributes or values result in a translation error.
- Syntax

```
<%@ page page_directive_attr_list %>
page_directive_attr_list ::=
  { language="scriptingLanguage"}
  { import="importList"}
  { session="true|false" }
  { buffer="none| sizekb" }
  { autoFlush="true| false" }
  { isThreadSafe="true|false" }
  { info="info_text" }
  { errorPage="error_url" } %>
```

- Example

```
<%@ page import="java.util.*, com.foo.*" buffer="16k" %>
```

Directive (Cont)

- Include directive
  - Lets you separate your content into more manageable elements, such as those for including a common page header or footer.
  - The page included can be a static HTML page or more JSP content.
  - Can be to include the contents of the indicated file at any location within the JSP page.
  - Example

```
<%@ include file="copyright.html" %>
```

- **header.html**

```
<div style="background: #E0E0E0; height: 80px; padding: 5px;">
   <div style="float: left">
       <h1>My Site</h1>
   </div>
   <div style="float: right; padding: 10px;">
       Search <input name="search">
   </div>
</div>
```

- **footer.html**

```
<div
   style="background: #E0E0E0; text-align: center; padding: 5px; margin-top: 10px;">
   @Copyright mysite.com
</div>
```

- **includeDemo.jsp**

```
<%@ page import="java.util.Random,java.text.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Jsp Include Directive</title>
</head>
<body>

   <%@ include file="../fragment/header.html"%>


   <h2>Jsp tutorial for Beginners</h2>

   Hi! This is Jsp Tutorial...


   <%@ include file="../fragment/footer.html"%>
</body>
</html>
```

Directive (Cont)

Declaration

- To declare variable and methods in scripting language used in the JSP page.
- Do not produce any output into current out stream
- Declarations are initialized when the JSP page is initialized.
- Syntax

```
<%! Declaration(s) %>
```

```
<%!
   public int sum(int a, int b) {
       return a + b;
   }
%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Method in JSP</title>
</head>
<body>

   <h1>
       1 + 2 =    <%=sum(1, 2)%>
   </h1>

</body>
</html>
```

Scriptlets

- Scriptlets can contain any code fragments that are valid for the scripting language specified in the language directive.
- Scriptlets are executed at request-processing time.
- To produce any output into the *out* stream depends on the actual code in the scriptlet
- Do not limit to one line of source code.
- Syntax

```
<% scriptlets %>
```

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```

Scriptlets

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<% if (day == 1 | day == 7) { %>
      <p> Today is weekend</p>
<% } else { %>
      <p> Today is not weekend</p>
<% } %>
</body>
</html>
```

```
Today is not weekend
```

Expressions

- Expressions are used to display simple values of variables or return values by invoking a bean's getter methods.
- The results of evaluating the expression are converted to a string and directly included within the output page.
- Syntax

<%= ... %>

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

```
Today's date: 11-Sep-2010 21:24:25
```

Comments

- There are two types comments in JSP page.
  - Comment to the JSP page itself.
    - syntax

```
<%--  comment for server side only --%>
```

    - Example

```
<%-- The request method is POST --%>
```

  - Comments that are intended to appear in the generated document sent to the client.
    - Syntax

```
<!-- comments which send to client -->
```

    - Comments have dynamic data

```
<!- comment <%=expression%> more comments -->
```

    - Example

```
<!-- Send data to server -->
```

Exception Handling

- JSP provides a mechanism for handling runtime exceptions
- It is possible to forward an runtime exception to an error handling JSP page for processing
- Example
  - `<%@ page isErrorPage="false" errorPage="errorHandler.jsp"%>`
  - `<%@ page isErrorPage="true"%> (in errorHandler.jsp)`

A sample JSP

```
<%@ page import="java.util.*, java.net.*" session="true" %>
<%@ include file="header.html" %>
<%!
  int count = 0;
  String getHello() { return "Hello world";}
%>
<HTML>
  <HEAD><TITLE> A sample JSP </TITLE></HEAD>
  <%-- print string "Hello word"--%>
  <%=getHello()%>
</HTML>
<jsp:include page="footer.html" />
```

JavaBean Components

Introduce JavaBean Components

- It provides a default, no-argument constructor.
- It should be serializable and implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

get**PropertyName**()	For example, if property name is *firstName*, your method name would be getFirstName() to read that property. This method is called accessor.
set**PropertyName**()	For example, if property name is *firstName*, your method name would be setFirstName() to write that property. This method is called mutator.

Creating JavaBean Components

```
<jsp:useBean id="bean's name" scope="bean's scope" typeSpec/>
```

```
<html>
<head>
<title>useBean Example</title>
</head>
<body>

<jsp:useBean id="date" class="java.util.Date" />
<p>The date/time is <%= date %>

</body>
</html>
```

Java Bean example

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable
{
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

```
<html>
<head>
<title>get and set properties Example</title>
</head>
<body>

<jsp:useBean id="students"
                    class="com.tutorialspoint.StudentsBean">
   <jsp:setProperty name="students" property="firstName"
                    value="Zara"/>
   <jsp:setProperty name="students" property="lastName"
                    value="Ali"/>
   <jsp:setProperty name="students" property="age"
                    value="10"/>
</jsp:useBean>

<p>Student First Name:
   <jsp:getProperty name="students" property="firstName"/>
</p>
<p>Student Last Name:
   <jsp:getProperty name="students" property="lastName"/>
</p>
<p>Student Age:
   <jsp:getProperty name="students" property="age"/>
</p>

</body>
</html>
```

JSP Expression Language

- Syntax :

  ${ expression language}

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Expression Language Demo</title>
</head>
<body>


   <jsp:useBean id="emp"
       class="org.o7planning.tutorial.jsp.beans.Employee">

       <jsp:setProperty name="emp" property="empNo" value="E01" />
       <jsp:setProperty name="emp" property="empName" value="Smith" />

   </jsp:useBean>


   <br>
   Emp No: <input type="text" value = "${emp.empNo}">
   <br>
   Emp Name <input type="text" value = "${emp.empName}">


</body>
</html>
```

JSP Expression Language

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

JSP Action

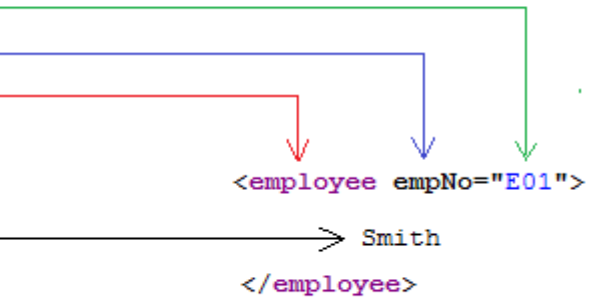
Syntax	Purpose
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

JSP Action – jsp:element, jsp:body, jsp:attribute

```
<?xml version="1.0" ?>
<%@ page pageEncoding="UTF-8"%>

<jsp:element name="data">

   <h3>Please view source of this page</h3>



   <jsp:useBean id="emp"
       class="org.o7planning.tutorial.jsp.beans.Employee">

       <jsp:setProperty name="emp" property="empNo" value="E01" />
       <jsp:setProperty name="emp" property="empName" value="Smith" />

   </jsp:useBean>


   <jsp:element name="employee">
       <jsp:attribute name="empNo" trim="true">
           <jsp:getProperty name="emp" property="empNo" />
       </jsp:attribute>
       <jsp:body>
           <jsp:getProperty name="emp" property="empName" />
       </jsp:body>
   </jsp:element>

</jsp:element>
```

JSP Action – jsp:element, jsp:body, jsp:attribute

```
<jsp:useBean id="emp"  class="org.o7planning.tutorial.jsp.beans.Employee">

<jsp:element name="employee">
    <jsp:attribute name="empNo" trim="true">
        <jsp:getProperty name="emp" property="empNo" />
    </jsp:attribute>
    <jsp:body>
        <jsp:getProperty name="emp" property="empName" />
    </jsp:body>
</jsp:element>
```



JSP Action – jsp:element, jsp:body, jsp:attribute

```
<?xml version="1.0" ?>
<%@ page pageEncoding="UTF-8"%>

<data>

   <h3>Please view source of this page</h3>

   <%--  Tạo đối tượng Employee và sét giá trị cho các field của nó --%>

   <jsp:useBean id="emp"
       class="org.o7planning.tutorial.jsp.beans.Employee">

       <jsp:setProperty name="emp" property="empNo" value="E01" />
       <jsp:setProperty name="emp" property="empName" value="Smith" />

   </jsp:useBean>

   <employee empNo="<%=emp.getEmpNo()%>">
       <%=emp.getEmpName()%>
   </employee>

</data>
```

JSP Action – jsp:forward

Following is the content of date.jsp file:

```
<p>
   Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

Here is the content of main.jsp file:

```
<html>
<head>
<title>The include Action Example</title>
</head>
<body>
<center>
<h2>The include action Example</h2>
<jsp:forward page="date.jsp" />
</center>
</body>
</html>
```

Assignment

- Each participant will do an exercise with functions:
  - Login screen
  - Change user password screen
  - Entertainment screen (mp3)
  - Checking Session screen
  - Read PDF
- Schedule
  - 1 weeks to implement

Assignment (Cont)

- Evaluation
  - send a package of source code to the trainer

Thank You!