JSP

You have experienced how to use out.println to write out html code in Servlets. Isn't it troublesome! JSP comes to rescue!

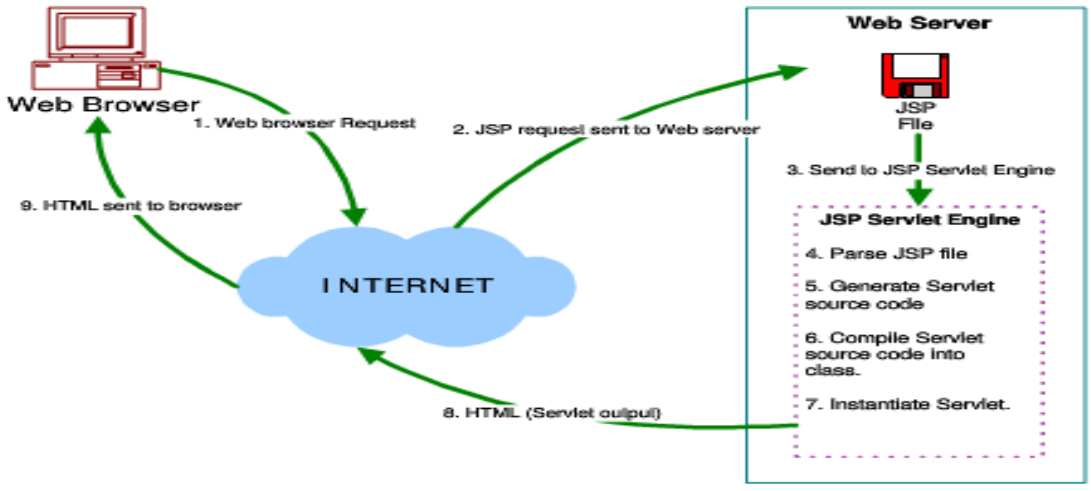
JSP (JavaServer Pages)

- Write HTML in client side instead of out.println in Servlets.
- Separate the dynamic part of your pages from the static HTML.
- Write the regular HTML in the normal manner using any tools.
- Enclose the code for the dynamic parts in special tags, most of which start with "<%" and end with "%>".

HelloWorld.jsp

```
<html>
    <head>
        <title>First JSP Program</title>
    </head>
    <body>
        <%
            out.println("Hello World! ");
        %>
    </body>
</html>
```

1 JSP architecture



Note: When a JSP page is called, it will be compiled (by the JSP engine) into a Java servlet. At this point, the servlet is handled by the servlet engine, just like any other servlet.

2 JSP Syntax Summary (incomplete)

JSP Element	Syntax	Interpretation	Example

JSP Expression	`<%= expression %>`	Expression is evaluated and placed in output.	`<%= new java.util.Date() %>`
			
			`Your hostname: <%= request.getRemoteHost() %>`
			
			Predefined variables: request, response, session

JSP Scriptlet	`<% code %>`	Code is run and put aside with the html code.	`<%`
			`String queryData = request.getQueryString();`
			`out.println("Attached GET data: " + queryData);`
			`%>`

JSP Declaration	`<%! code %>`	Code is inserted in body of servlet class.	`<%! 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>`
			`------------------------`
			`<%! private int accessCount = 0; %>`
			`Accesses to page since server reboot:`
			`<%= ++accessCount %>`

JSP `page` Directive	`<%@ page att="val" %>`	Directions to the servlet engine about general setup.	`<%@ page import="java.util.*" %>` Legal attributes, with default values in bold, are: • import="*package.class*" • contentType="*MIME-Type*" • isThreadSafe="**true**|false" • session="**true**|false" • buffer="*size*kb|none" • autoflush="**true**|false" • extends="*package.class*" • info="*message*" • errorPage="*url*" • isErrorPage="true|**false**" • language="java"

JSP `include` Directive	`<%@ include file="url" %>`	A file on the local system to be included when the JSP page is translated into a servlet.	`<BODY>`
			`<%@ include file="/navbar.html" %>`
			`<!-- Part specific to this page ... -->`
			`</BODY>`
			`</HTML>`

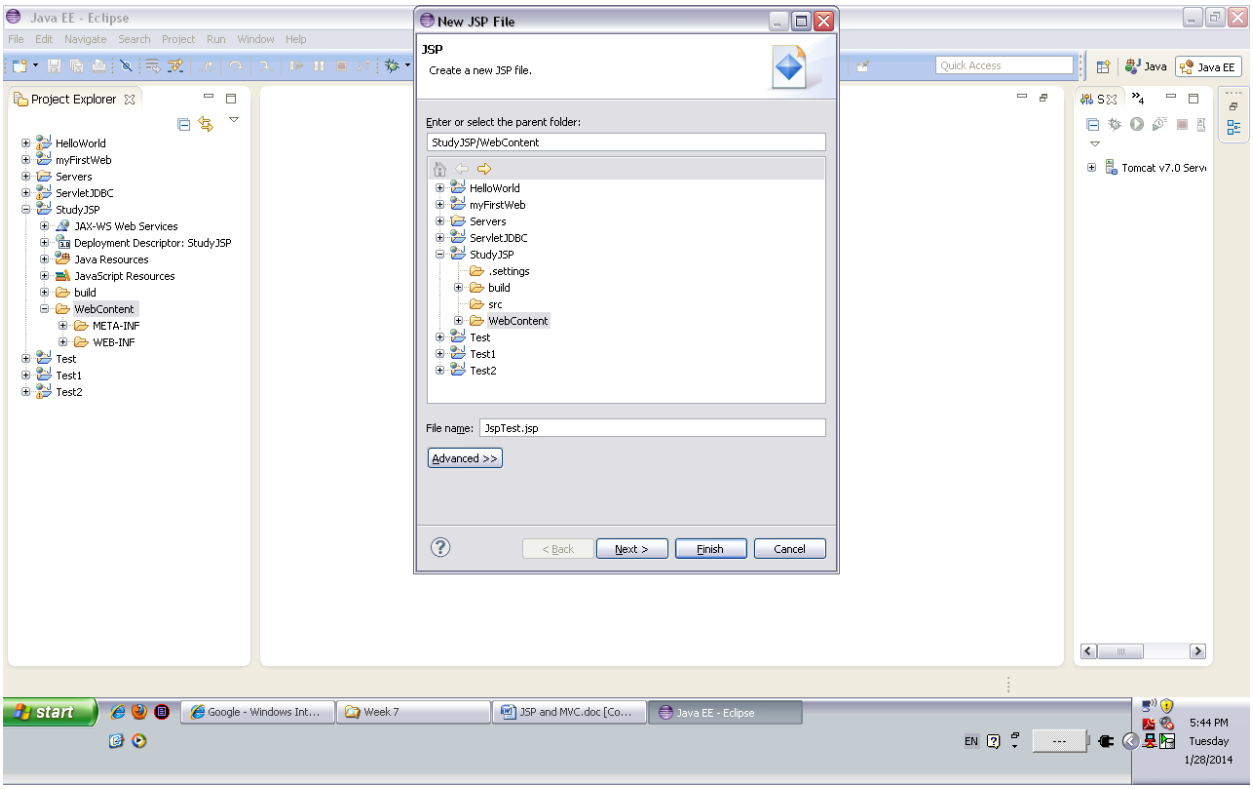
JSP Comment	`<%-- comment --%>`	Comment; ignored when JSP page is translated into servlet.	If you want a comment in the resultant HTML, use regular HTML comment syntax of `<-- comment -->`.

3 Best Practices of JSP:

- Don't overuse Java code in HTML pages
- Use include mechanism (header, footer, navigation bar)
- Separate the business logic and presentation
- Use a database for persistent information
- ...

3.1 Let’s create a jsp page:

Create a Dynamic Web Project called StudyJSP, Write click WebContent -> New -> JSP File, create a JspTest.jsp:

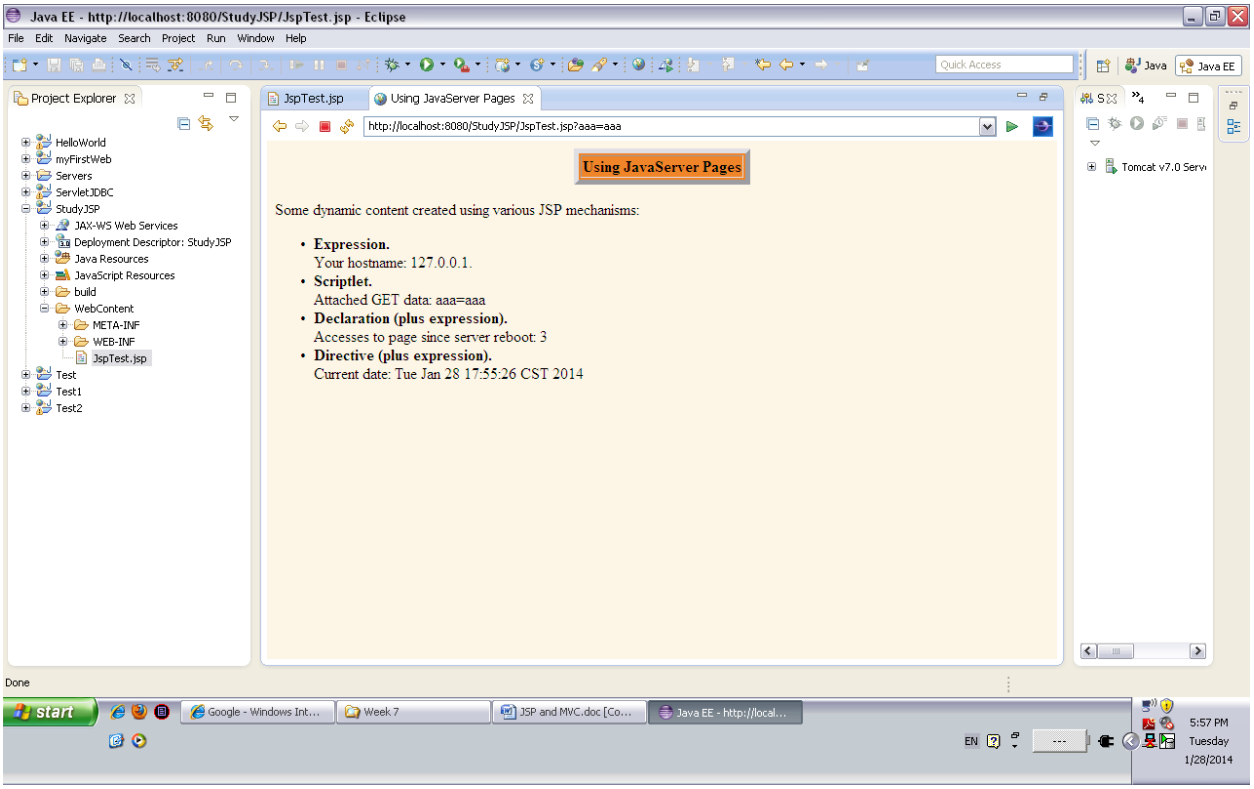


Maintain the jsp file to contain the following code:

```
<HTML>
<HEAD>
<TITLE>Using JavaServer Pages</TITLE>
</HEAD>

<BODY BGCOLOR="#FDF5E6" TEXT="#000000">

<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
  <TR><TH CLASS="TITLE">
      Using JavaServer Pages</TABLE>
</CENTER>
<P>

Some dynamic content created using various JSP mechanisms:
<UL>
  <LI><B>Expression.</B><BR>
      Your hostname: <%= request.getRemoteHost() %>.
```

```
  <LI><B>Scriptlet.</B><BR>
      <% out.println("Attached GET data: " +
                     request.getQueryString()); %>
  <LI><B>Declaration (plus expression).</B><BR>
      <%! private int accessCount = 0; %>
      Accesses to page since server reboot: <%= ++accessCount %>
  <LI><B>Directive (plus expression).</B><BR>
      <%@ page import = "java.util.*" %>
      Current date: <%= new Date() %>
</UL>

</BODY>
</HTML>
```

Run the jsp file on the server, you will see the following page. Then, read thru the jsp code carefully, and understand how to use those element in jsp file.



3.2 Another jsp file to study:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<%! int number=1; %>
<%! public int count()
    {
        return number++;
    }
 %>

 <% out.println("Welcome to WorkshopII!");
 %>
 <br>
   <%=count() %> customers have visited this websites!
</body>
</html>
```

4 Predefined Objects

To simplify code in JSP expressions and scriptlets, you are supplied with eight automatically defined objects, sometimes called *implicit objects*. They are:

- request
- response
- out
- session
- application
- config
- pageContext
- page

The request object

Each time a client requests a JSP page, the JSP engine creates a new object to represents that request called *request* object. The *request* object is an instance of class *javax.servlet.http.HttpServletRequest*. The *request* object contains all information about the

current HTTP request and the clients. Be noted that *request* object only available in a scope of the current request. It is re-created each time new request is made. By using methods of the *request* object you can access almost information such as HTTP header, query string, cookies...

The response object

JSP also creates the *response* object which is an instance of class *javax.servlet.http.HttpServletResponse.* The *response* object represents the response to the client. By using this object you can add new cookies, change MIME content type of the page. In addition, the response object also contains sufficient information on the HTTP to be able to return HTTP status codes or make the page redirect to the other page.

The session object

The session object is used to track information of a particular client between multiple requests. The session object is available in the server so it can help you to overcome the stateless of HTTP protocol. You can use session object to store arbitrary information between client requests. The session object is an instance of class *javax.servlet.http.HttpSession.*

The out object

The output stream is exposed to the JSP through the *out* object. The *out* object is an instance of class *javax.servlet.jsp.JspWriter*. The out object may refer to an output stream or a filtered stream... You can use the *out* object methods to send the data into the output stream such as *println* method. Then JSP take care the rest.

The pageContext object

The *pageContext* object represents the entire JSP page. You can use the pageContext object to get page attributes of a page. The pageContext object is an instance of class *javax.servlet.jsp.pagecontext.*

The application object

The *application* object is a representation of JSP page through its life cycle. The *application* object is created when a JSP page is initialized and removed when the JSP page is removed by *jspDestroy()* method or JSP page is recompiled. As its name imply, the information of the *application* object is accessible to any object used within the JSP page.

The config object

The *config* object allows you to access the initialization parameters for the Servlet and JSP engine. The *config* object is an instance of the class *javax.servlet.ServletConfig.*

The page object

The *page* object is an instance of a JSP page. By using the *page* object, you can call any method of the page's servlet.

The exception object

The exception object contains the exception which is thrown from the previous JSP page. You can use the exception object to generate friendly error message based on error condition to the end-user.

Examples on "Implicit Objects" from

http://www.journaldev.com/2038/jsp-implicit-objects-with-examples

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
    pageEncoding="US-ASCII"%>
<%@ page import="java.util.Date" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Index JSP Page</title>
</head>
<body>
<%-- out object example --%>
<h4>Hi There</h4>
<strong>Current Time is</strong>: <% out.print(new Date()); %><br><br>

<%-- request object example --%>
<strong>Request User-Agent</strong>: <%=request.getHeader("User-Agent") %><br><br>

<%-- response object example --%>
<%response.addCookie(new Cookie("Test","Value")); %>

<%-- config object example --%>
<strong>User init param value</strong>:<%=config.getInitParameter("User") %><br><br>

<%-- application object example --%>
<strong>User context param
value</strong>:<%=application.getInitParameter("User") %><br><br>
```

```
<%-- session object example --%>
<strong>User Session ID</strong>:<%=session.getId() %><br><br>

<%-- pageContext object example --%>
<% pageContext.setAttribute("Test", "Test Value"); %>
<strong>PageContext attribute</strong>:
{Name="Test",Value="<%=pageContext.getAttribute("Test") %>"}<br><br>

<%-- page object example --%>
<strong>Generated Servlet Name</strong>:<%=page.getClass().getName() %>

</body>

</html>
```

Read more on these objects from any online JSP tutorials.

5 JSP Action Element

Pay attention to JSP action elements. The following example shows the usage of **<jsp:forward>**

Create three files as follows: login.jsp, checklogin.jsp, success.jsp. Run the program, and understand the code below.

`login.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form method=post action=checklogin.jsp>
  <table>
    <tr>
      <td>Name:</td>
      <td><input type=text name=name value=<%=request.getParameter("user") %>></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input type=password name=password></td>
    </tr>
    <tr colspan=2>
      <td><input type=submit value=login></td>
```

```
      </tr>
  </table>
</form>
</body>
</html>
```

checklogin.jsp

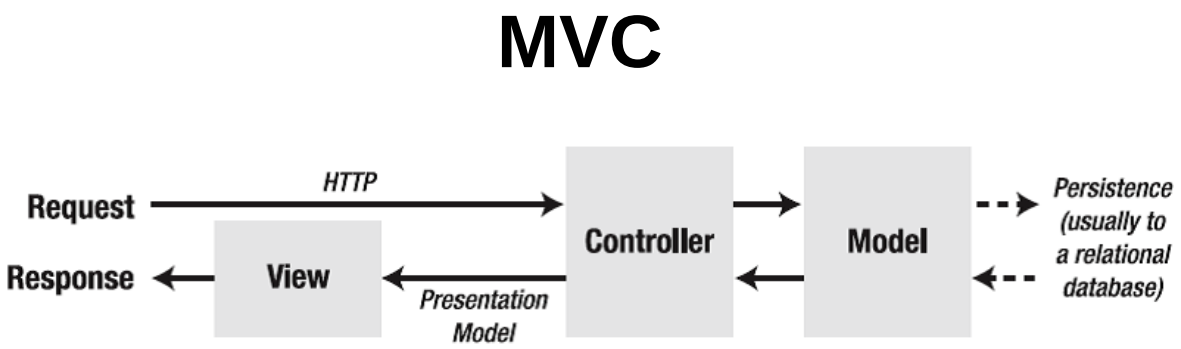
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<%
   String name = request.getParameter("name");
   String password = request.getParameter("password");

   System.out.println("111" +name+" "+password);
   if (name.equals("Gigi")  && password.equals("123456"))
   {System.out.println("222");
%>
<jsp:forward page="success.jsp">
    <jsp:param name="user" value="<%=name %>" />
</jsp:forward>
<%
    }
   else
   {
%>
          <jsp:forward page="login.jsp">
           <jsp:param name="user" value="<%=name %>" />
          </jsp:forward>

<% }
%>


</body>
</html>
```

success.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  Welcome, <%=request.getParameter("user") %>
</body>
</html>
```

6. MVC



Model

The model is a collection of Java classes that form a software application intended to store, and optionally separate, data. A single front end class that can communicate with any user interface (for example: a console, a graphical user interface, or a web application).

View

The view is represented by a **Java Server Page**, with data being transported to the page in the HttpServletRequest or HttpSession.

Controller

The Controller **servlet** communicates with the front end of the model and loads the HttpServletRequest or HttpSession with appropriate data, before forwarding the HttpServletRequest and Response to the JSP using a RequestDispatcher.

The Servlet is a Java class, and it communicates and interacts with the model, but does not need to generate HTML or XHTML output;

JSPs do not have to communicate with the model because the Servlet provides them with the information—they can concentrate on creating output.

Practice:

JSP and Servlets

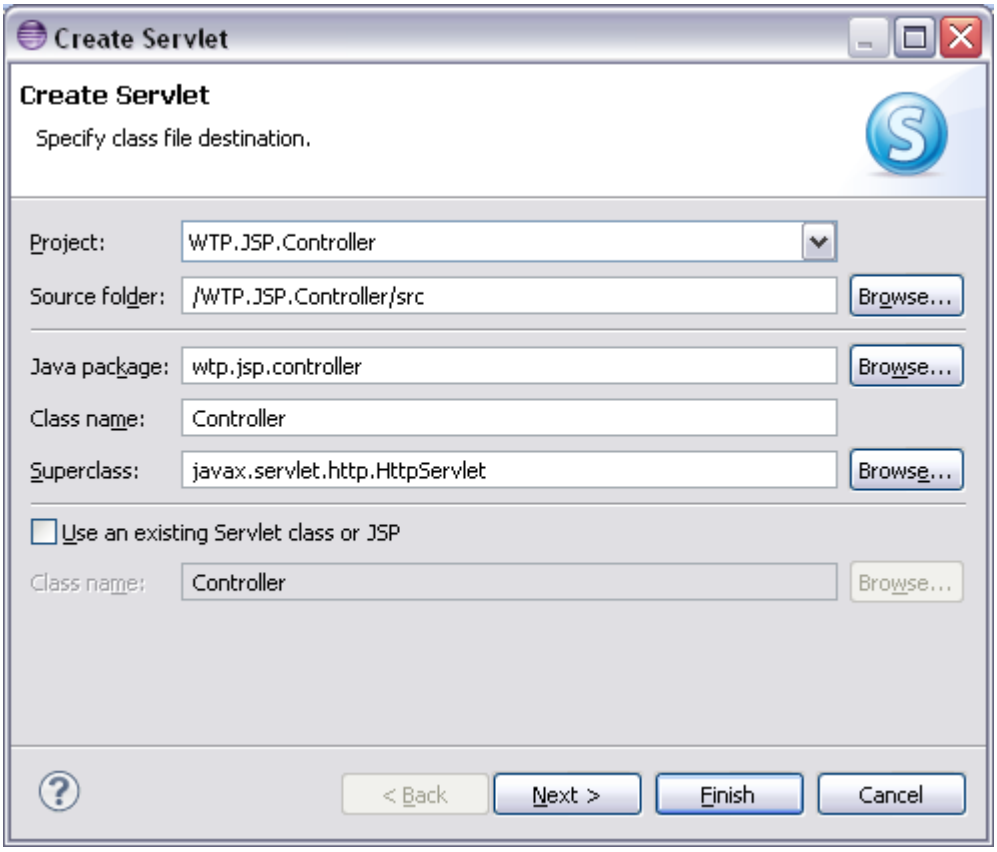
1. Create Project

This example will demonstrate the usage of JSPs for the display and a servlet as the controller for a web application. The servlet will dispatch the request to the correct JSP.

Create the Dynamic Web Project "WTP.JSP.Controller". Right click the project name -> New -> Package, name the package " wtp.jsp.controller "

2. Create the Controller (servlet)

Right click the package name -> New -> Servlet named "Controller" as follows:



This controller will check which parameters have been passed to the servlet and then forward the request to the correct JSP.

Create a Servlet, then modify the code as follows:

Note: Do not change the generated code.

```
package wtp.jsp.controller;

import java.io.IOException;
import java.util.Map;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Controller
 */

@WebServlet("/Controller")
public class Controller extends HttpServlet {
	private static final long serialVersionUID = 1L;
```

```
	private static String DELETE_JSP = "/Delete.jsp";
	private static String EDIT_JSP = "/Edit.jsp";
	private static String SHOWALL_JSP = "/ShowAll.jsp";


	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		String forward="";
		// Get a map of the request parameters
		Map<String, String[]> parameters = request.getParameterMap();
		if (parameters.containsKey("delete")){
			forward = DELETE_JSP;
		} else if (parameters.containsKey("edit")){
			forward = EDIT_JSP;
		} else {
			forward = SHOWALL_JSP;
		}
		RequestDispatcher view = request.getRequestDispatcher(forward);
		view.forward(request, response);
	}

}
```

3. Create the Views (JSP)

In the folder "WebContent" create the new JSP "ShowAll" with the following code.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Show all names</title>
</head>
<body>

<form method="GET" action='Controller' name="showall">
<table>
	<tr>
		<td><input type="checkbox" name="id1" /></td>
		<td>Jim</td>
		<td>Smith</td>
	</tr>
	<tr>
		<td><input type="checkbox" name="id2" /></td>
		<td>Jim</td>
		<td>Bean</td>
	</tr>
</table>

<p><input type="submit" name="delete" value="delete" /> 
   <input type="submit" name="edit" value="edit" /> 
	<input type="reset"
	value="reset" /></p>

</form>
```

```
</body>

</html>
```

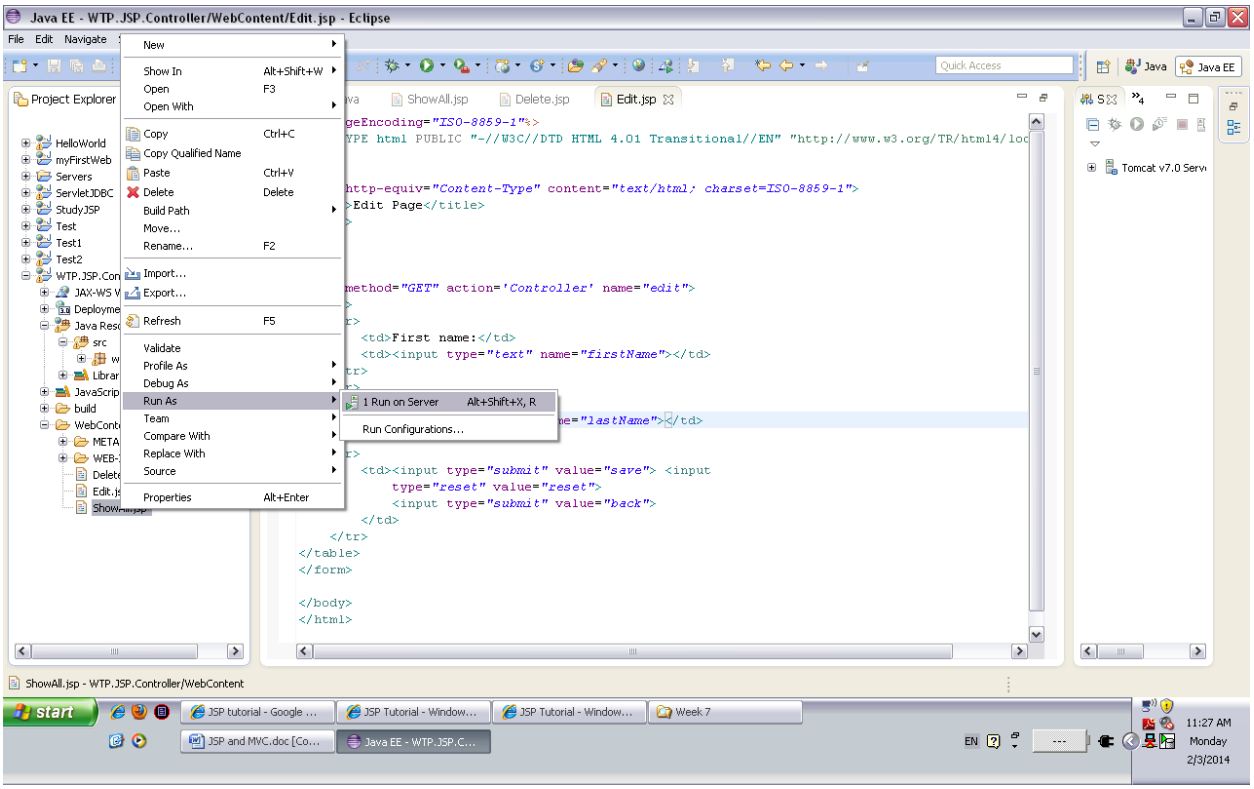
Create the JSP "Delete.jsp" with the following code:

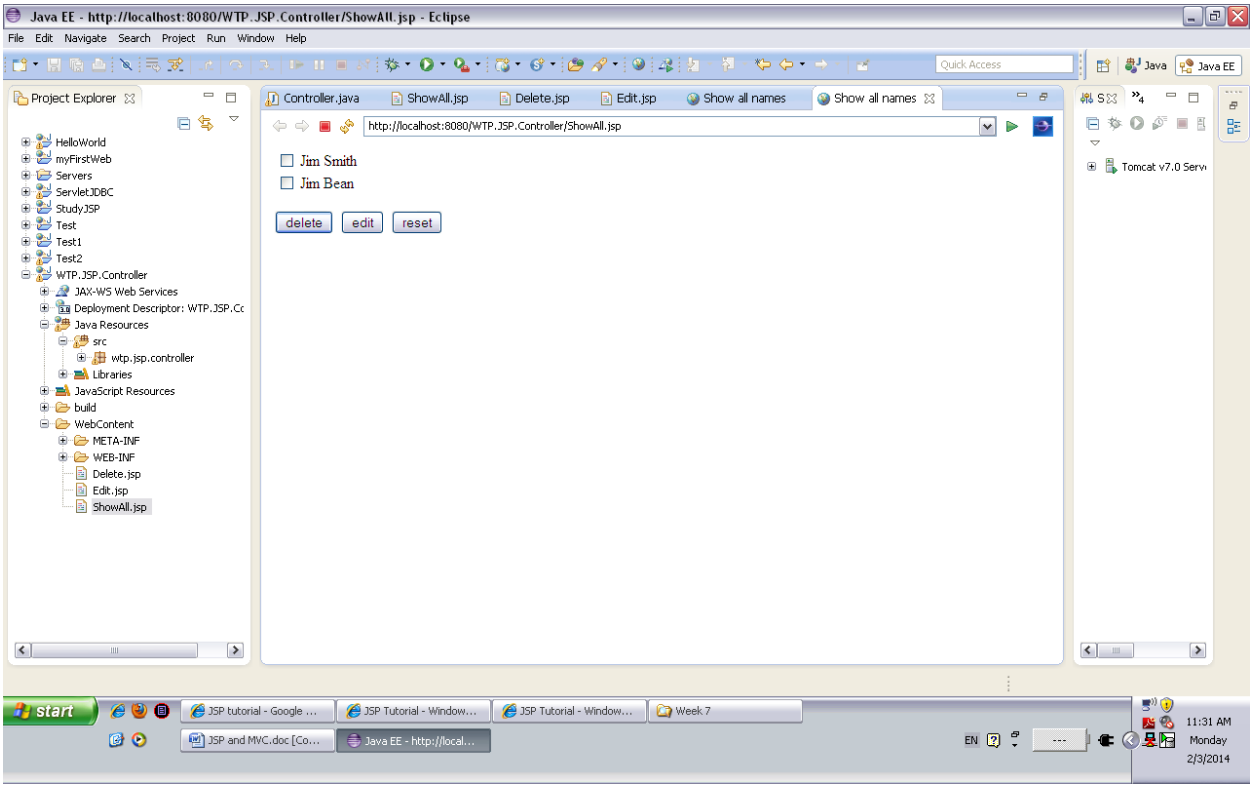
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Delete Page</title>
</head>
<body>

Delete successful
<form method="GET" action='Controller' name="delete_success"><input
	type="submit" value="back"></form>
</body>

</html>
```

Create the JSP "Edit.jsp" with the following code:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Edit Page</title>
</head>
<body>


<form method="GET" action='Controller' name="edit">
<table>
	<tr>
		<td>First name:</td>
		<td><input type="text" name="firstName"></td>
	</tr>
	<tr>
		<td>Last name:</td>
		<td><input type="text" name="lastName"></td>
	</tr>
	<tr>
		<td><input type="submit" value="save"> <input
			type="reset" value="reset">
			<input type="submit" value="back">
		</td>
	</tr>
</table>
</form>

</body>

</html>
```

Run your new application by running "ShowAll.jsp" on the server. You should be able to navigate between the pages.





Practice:

Use JSP, Servlet to make the following login pages:

1. When the user enters correct user name and password, say Smith and 12345, clicking the Login button links to the Welcome screen.
2. When the user enters anything else in the user name or password, clicking the Login button links to the Registration screen.
3. The Reset button will empty the edit box values.
4. The Back and the Register button will both take you back to the Login page.