Java Server Pages (JSP)

& Java Servlets

# 1. Introduction

Java Server Pages (JSP) is a server-side technology for developing Webpages that supports static and dynamic content. JSP is used to construct dynamic content and it’s a convenient way of writing servlets. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>. JSP source files are represented by .jsp extension.

JSP fulfills the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

JSP also allows users to use any other scripting language that which can call Java objects by specifying as follows:

<%@ page language = “Scripting-language” %>

Using JSP, we can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

## JSP Features

Some of the key features of JSP technology are these:

1. Simple – The technology is simple and it provides an easy way to develop web applications.
2. Powerful – It is a technology that is based on Java, making it not only powerful but also secured and robust.
3. Portable – Like Java, it’s a platform independent technology. Web applications based on JSP can run on any browser.
4. Tags – It uses simple tags which are pre-defined. It also helps minimize bugs. It also allows users to customize their own tags.

Our goal is to convert our computer to both a client and a server computer. The advantage of this process is that you do not need the services of an external server. Furthermore you learn how to make a computer a server. We also learn the basics for JSP (Java Server Page) and servlet.

Note: The following installations are for 32-bit machine.

# 2. Installing the Tomcat Binary Distribution

Installation details are in a separate file.

## Testing the server

After installing Tomcat, test your installation and we will create a few examples.

### Example 1

A html file named: **Hello.html** to display word Hello in a colored background.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD><TITLE>HTML Test</TITLE></HEAD>

<BODY BGCOLOR="#FDF5E6">

<H1>This is my first html test</H1>

Hello.

</BODY>

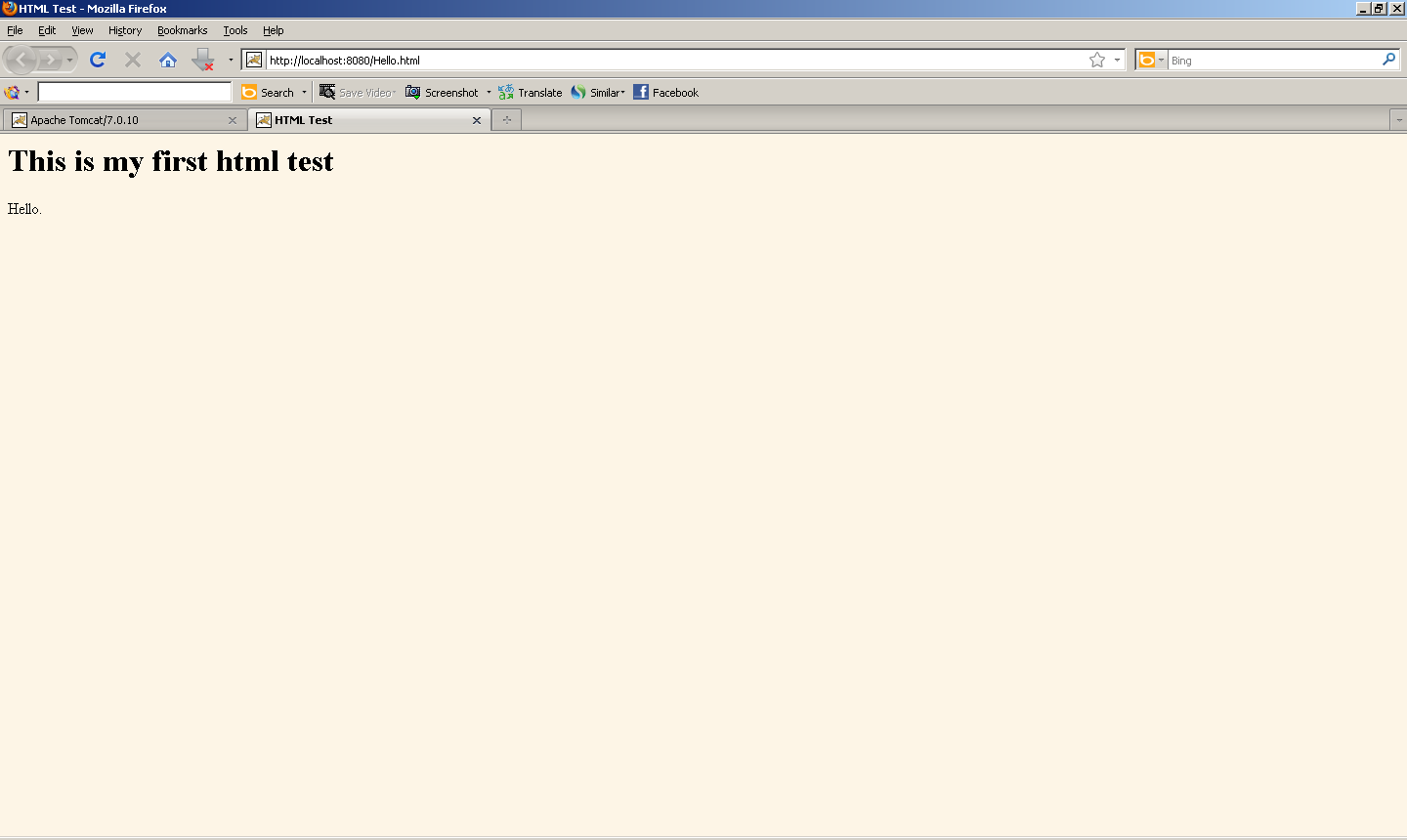
</HTML>

Save Hello.html into:

**C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\ROOT**.

In your browser enter: [**http://localhost:8080/Hello.html**](http://localhost:8080/Hello.html)

You should see:



**Description**: This is a simple html document and does not need any description.

### Example 2

A jsp named **Hello.jsp** to display the current time and date:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD><TITLE>JSP Test</TITLE></HEAD>

<BODY BGCOLOR="#FDF5E6">

<H1>This is my first JSP Test</H1>

Time: <%= new java.util.Date() %>

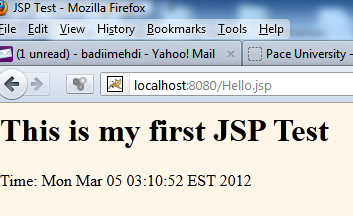
</BODY></HTML>

Save Hello.jsp into:

**C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\ROOT**.

In your browser enter: [**http://localhost:8080/Hello.jsp**](http://localhost:8080/Hello.jsp)

You should see:



**Description of the JSP code**: Everything inside the symbols <% and %> is a java instruction. In other words we write our Java code within <% and %>. Everything else is html instruction. The Java method call:

new java.util.Date()

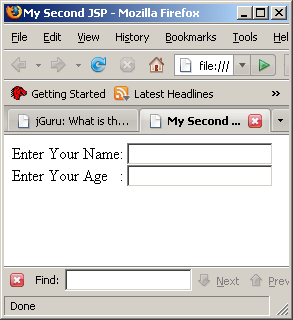
Creates an object of type class Date. This creates an object of type class Date. It is like you write the following in an ordinary java program:

System.out.println(new java.util.Date()+ "");

With that understanding, we can explore some more JSP programs which are a bit more complicated. I am assuming you know html – at least the basics. One of the most important part of html instruction is **html form**. We use html forms in almost every example.

### Example 3

We would like to make a JSP that displays:



The regular html document to display this page is:

<html>

<head>

<title>My Second JSP</title>

</head>

<body>

<form action=*"http://localhost"* method=*"post"*>

Enter Your Name: <input type=*"text"* name = *"yourName="* ><br>

Enter Your Age&nbsp;&nbsp; : <input type=*"text"* name = *"yourAge="* >

</form>

</body>

</html>

Name this html document: Second.html, save it to:

**C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\ROOT**.

In your browser enter: [**http://localhost:8080/Second.html**](http://localhost:8080/Second.html)

The JSP version of this html document is:

<html>

<head>

<title>My Second JSP</title>

</head>

<body>

<%

out.println("<form action=\"http://localhost\" method=\"post\">");

out.println("Enter Your Name: <input type=\"text\" name = \"yourName=\" </input><br>");

out.println("Enter Your Age&nbsp;&nbsp; : <input type=\"text\" name = \"yourAge=\" </input>");

%>

</form>

</body>

</html>

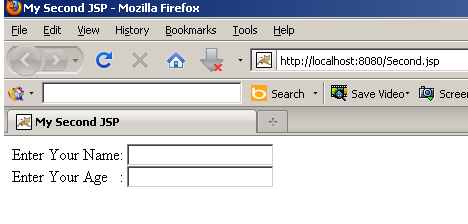
Name the JSP: Second.jsp and save it in:

**C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps\ROOT**.

In your browser enter:

<http://localhost:8080/Second.jsp>

**Display page**:



**Description**: The instruction **out.println(….)** writes a string to the webpage and is similar to the java instruction **System.out.println(….)** in an ordinary java program that writes a string on the screen. It is the process of writing a string that displays the html on the screen.

***Note:*** Here is a trick to make the html page into a JSP. It is the following changes made to the html document:

1. Converting every symbol to this format: \**"** to **\"**.

2. Inserting every line into **out.println(**"**….**"**)**..

For instance, in order to display the string **"abc"** (with the double-quotations) from an ordinary java program we must write:

System.out.println("\"abc\"").

The first and the last symbols " acts like an open and close parenthesis.

***Note:*** We can also put every line into out.println(…). That is:

<%

out.println("<html><head><title>My Second JSP</title></head><body>");

out.println("<form action=\"http://localhost\" method=\"post\">");

out.println("Enter Your Name: <input type=\"text\" name = \"yourName=\" </input><br>");

out.println("Enter Your Age&nbsp;&nbsp; : <input type=\"text\" name = \"yourAge=\" </input>");

out.println("</form></body></html>");

%>

A JSP (Java Server Page) is a mixture of html and Java instructions. The JSP in the above example (example one) there is only one java instruction new **java.util.Date()**. The java code is embedded inside html tags. Therefore a JSP looks like HTML, with Java embedded inside.

Java Server Pages (JSP) is also a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

A **Servlet**, on the other hand, is a Java program with HTML embedded inside. JSPs are more convenient for presenting a web page. Servlets are the Java programs that runs on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, and then send the response back to the web server.

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.

# 3. JSPs and Servlets with Eclipse

Since we have Java embedded in HTM (JSP) and HTML embedded in Java we need an IDE to be able to debug our program. Eclipse is the vehicle for our work.

Note: If you are using any other IDE you need to either switch to eclipse or learn how to use your IDE for the following material and through the end of the course.

The following eclipse website is good to get more information:

<http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.wst.webtools.doc.user/topics/>

In the rest of this lecture we cover:

1. Creating an Apache Tomcat server.

2. Creating a dynamic Web project

3. Creating an HTML document.

4. Creating a JSP.

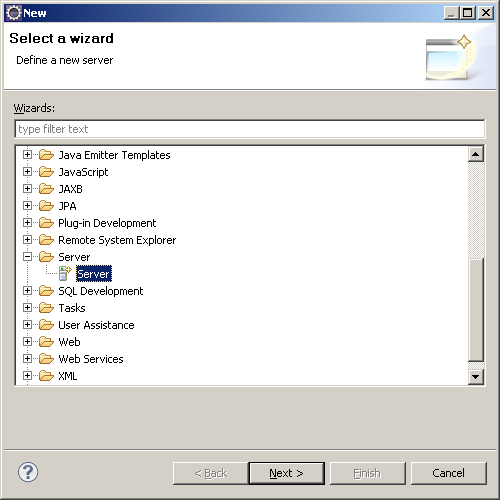
5. Creating a Servlet.

You should always do steps 1 & 2.

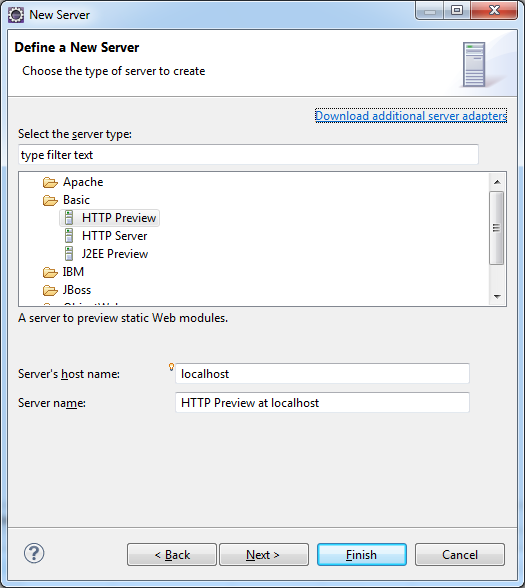
## 3.1. Creating an Apache Tomcat server

1. Make an empty folder and load eclipse for this folder. **Do not make a java project**.

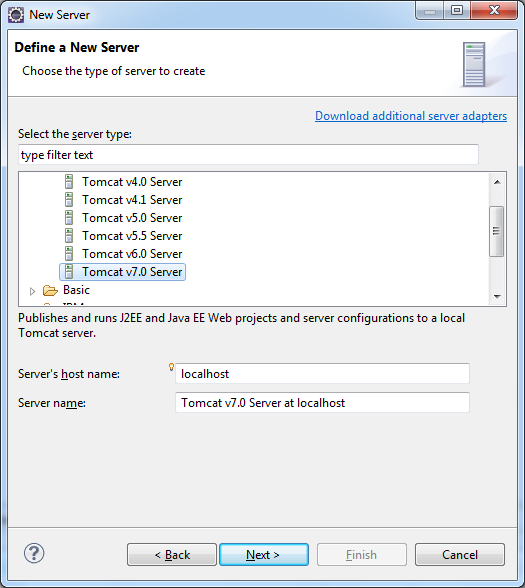
On the menu bar click: **New 🡪 Other 🡪 Server 🡪 Server** to see:



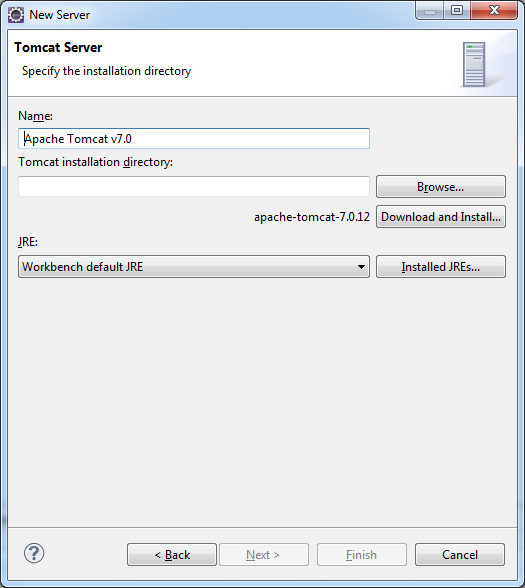
Click: **Next**.



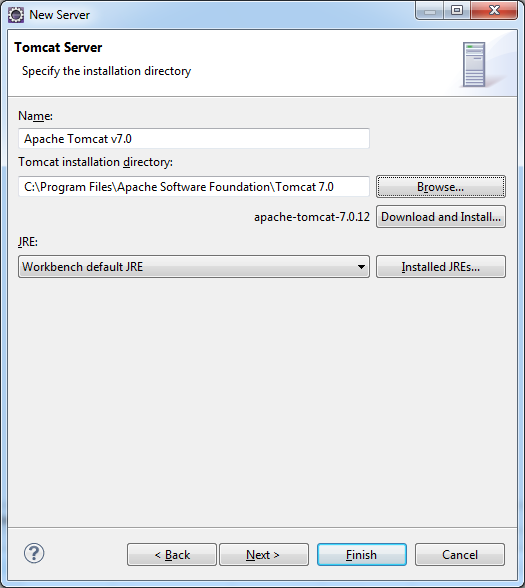
Expand the icon Apache and select **Tomcat v9.0 Server** to see:



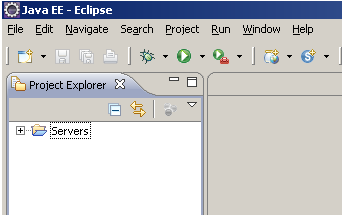
2. Click **Next**:



3. Browse, find and insert **Tomcat 9.0** to see:

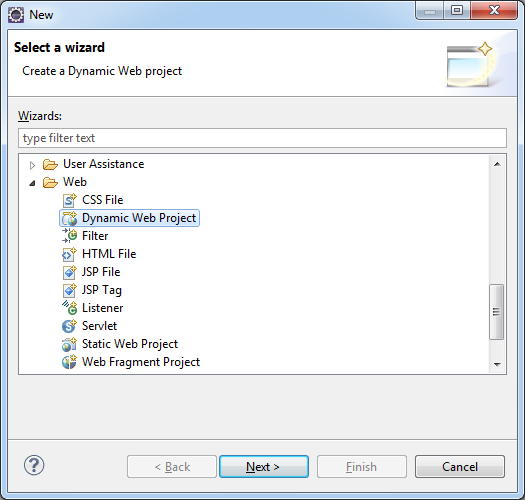


4. Click **Finish**. The Project Explorer should be:

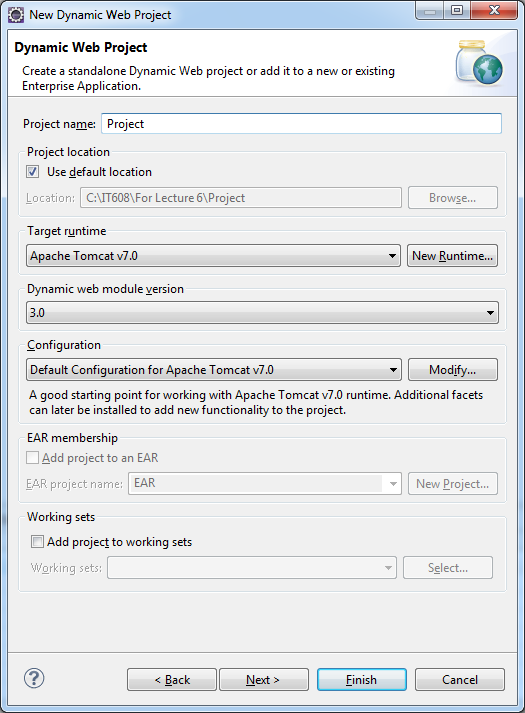


## 3.2. Creating a dynamic Web project

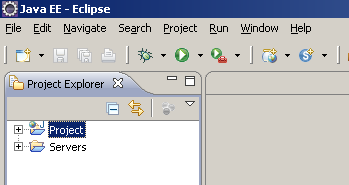
1. Click **File 🡪 New 🡪 Other 🡪 Web 🡪 Dynamic Web project**:



2. Click: **Next** and name this project. I named it: **Project**



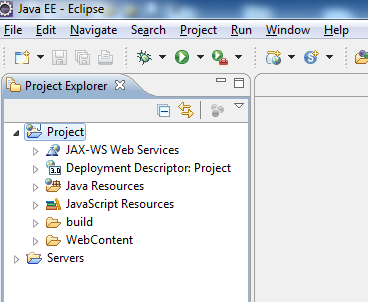
3. Click: **Finish**. The eclipse editor should be:



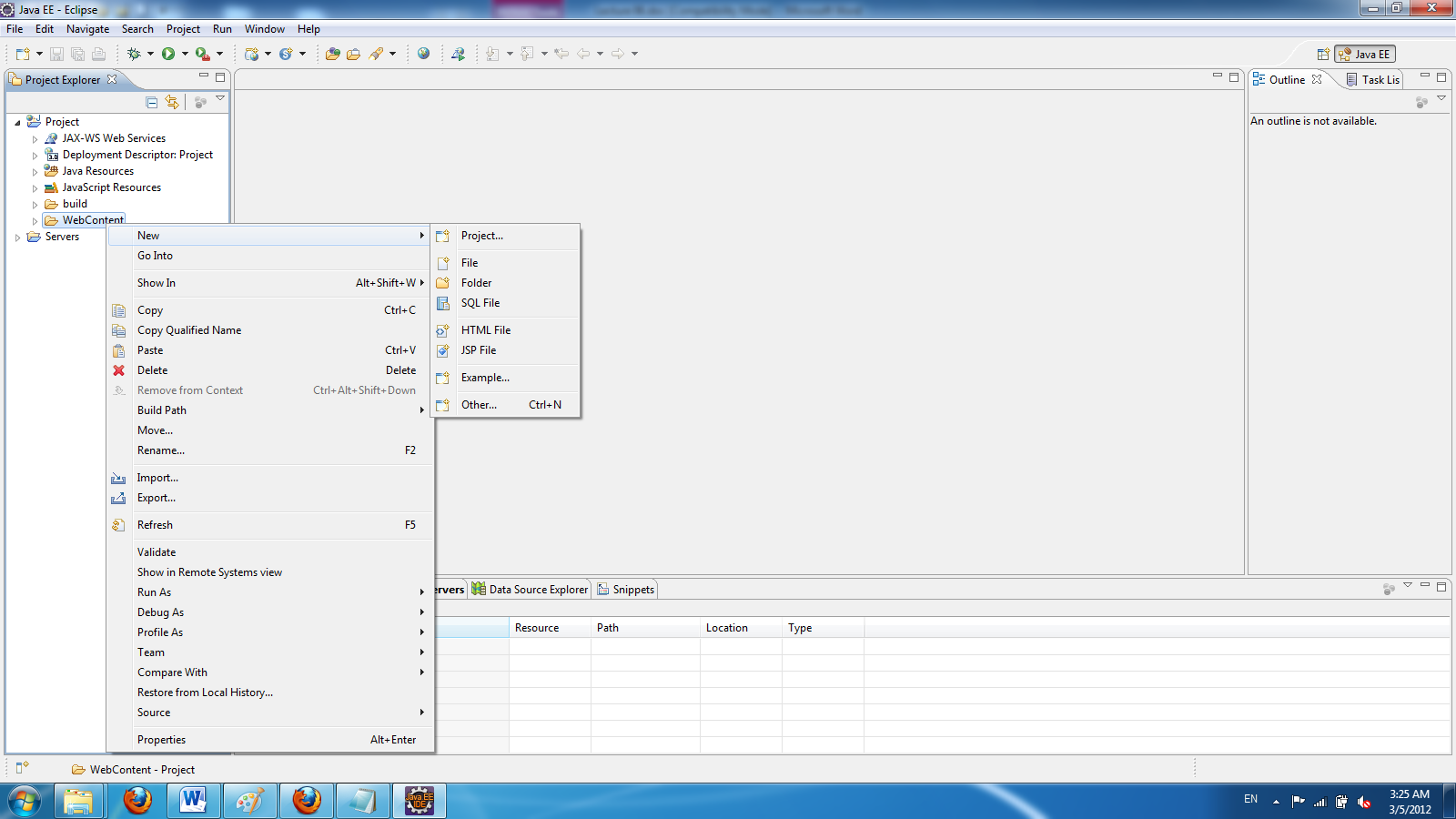
Note: *Creating a server and a dynamic project is a* ***must be*** *for HTMLs, JSPs, and Servlets*.

## 3.3. Creating an HTML document

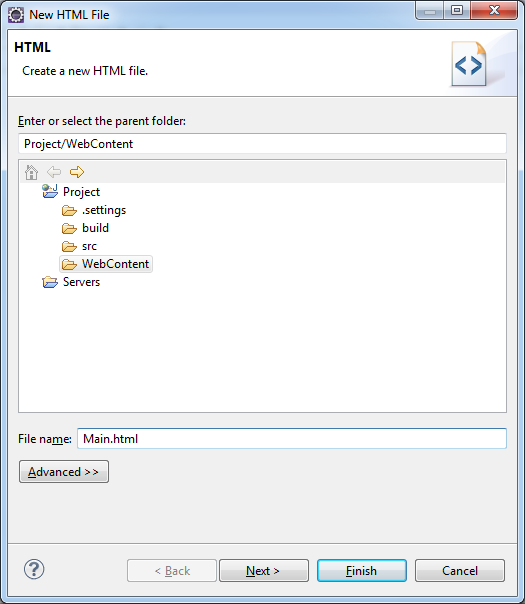
1. In the Project Explorer, expand your project to see:



2. Right click on: **WebContent** and click on: **New** to see:



3. Click on **HTML File page** to see:



4. Enter a name (I named it Main.html) and click **Finish**. The content of the editor is:

<!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>

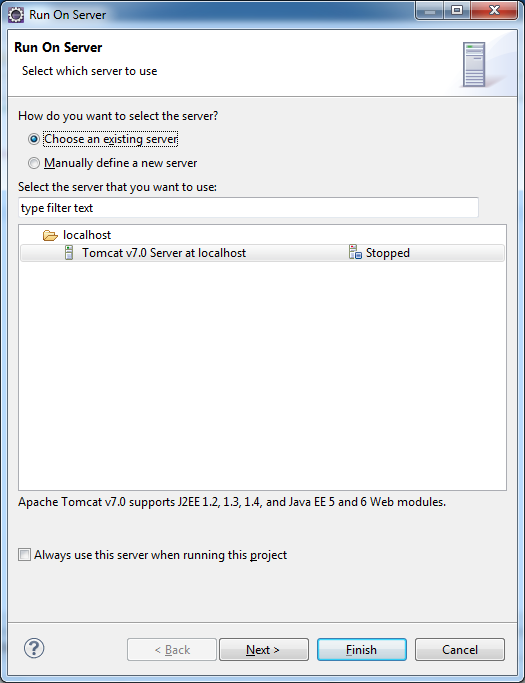
</head>

<body>

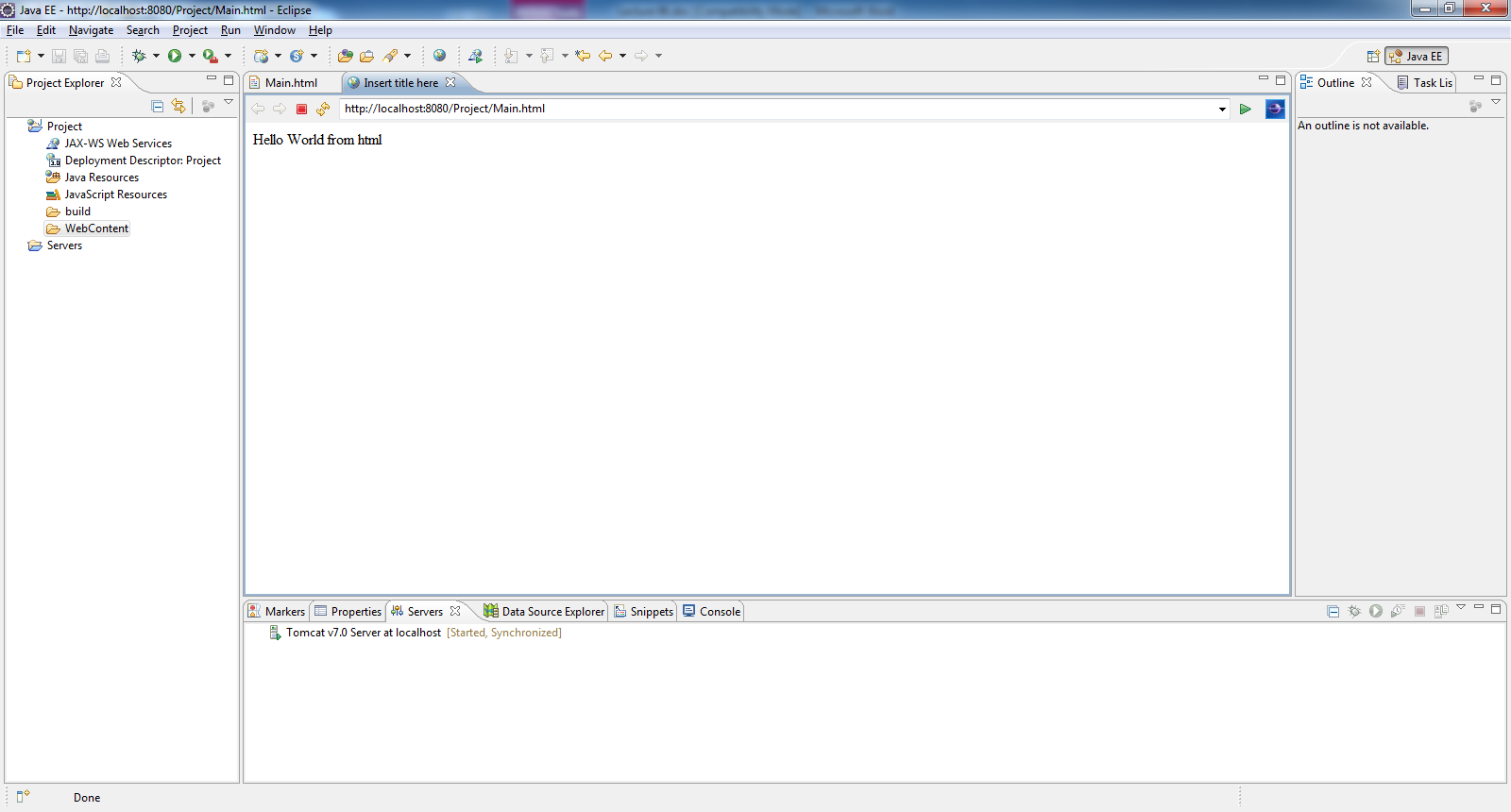
</body>

</html>

5. Write: Hello World from html between the tags <body> ...</body>. From the taskbar stop Apache. On the menu bar click on run icon A green play icon. to see:



6. Select Tomcat and click on: Finish to see:



7. Stop the tomcat server by clicking on the small red rectangle on the lower window (from: **console** window). Close the html page.

## 3.4. Creating a JSP

In the Project Explorer, expand your project and right click on your **WebContent** and click on **New 🡪 JSP**. Enter a name (I named it Main) and click **Finish**. The content of the editor is:

<%@ 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>

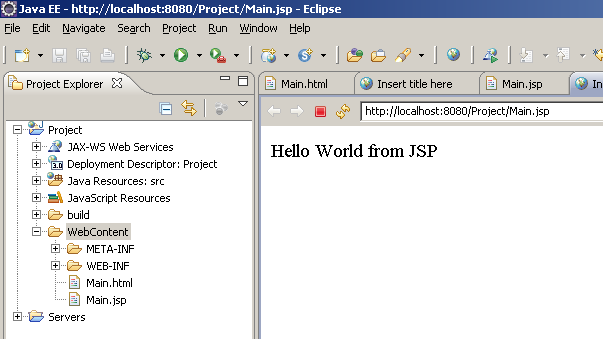
</head>

<body>

</body>

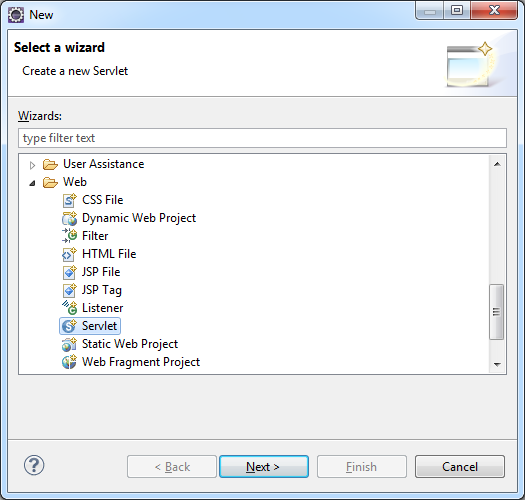
</html>

Write: Hello World from JSP between the tags <body> ...</body>. On the menu bar click on run icon A green play icon. and select the server (like in step 5 in the previous section) to see:

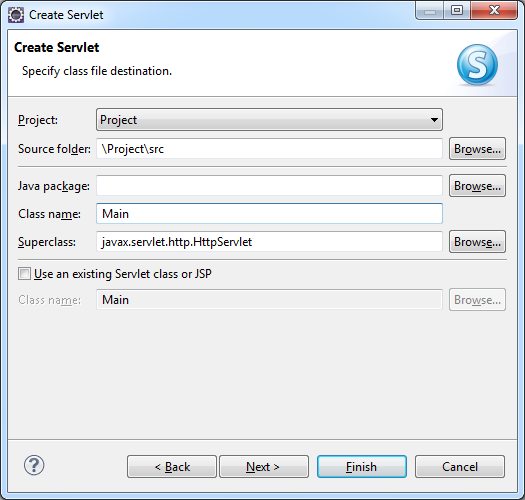


## 3.5. Creating a Servlet

1. In the Project Explorer, expand your project and right click on **WebContent 🡪 New 🡪 Others** to see:



1. Select **Servlet** and click **Next**:
2. Enter a class name (I entered Main):



Note: Since you only have one class there is no need to have a package. Therefore leave the package window blank.

1. Click **Finish**. The content of the editor should be:

**import** java.io.IOException;

**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 Main

\*/

@WebServlet("/Main")

**public** **class** Main **extends** HttpServlet {

**private** **static** **final** **long** *serialVersionUID* = 1L;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** Main() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@see** HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// **TODO** Auto-generated method stub

}

/\*\*

\* **@see** HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

}

}

Note: This is a java class. You should not get any error. Run the server. You should see a blank servlet (we later add instruction to this page). Stop the server by clicking on the small rectangle on the lower window.

Note: Always click on the tab: Console to make sure no other program is running. If you have a situation like the following screen shot click on the small red button to stop a program:

A Console tab that reads:

INFO: Starting Servlet Engine: Apache Tomcat/7.0.26
Mar 05, 2012 6:20:44 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Mar 05, 2012 6:20:44 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Mar 05, 2012 6:20:44 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 263 ms

# 4. Adding more code to the HTML document, the JSP, and the Servlet

We now add more code to the html document, JSP, and Servlet class.

## 4.1. Using Eclipse to add code to the HTML document

### Example 1.

Let us complete the HTML document of section 1.1.

1.Make the HTML document to be as follows.

<!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 action=*"http://localhost"* method=*"get"*>

Enter Your Name: <input type=*"text"* name=*"yourName"*><br>

Enter Your Age&nbsp;&nbsp;&nbsp;&nbsp;:<input type=*"text"* name=*"yourAge"*>

</form>

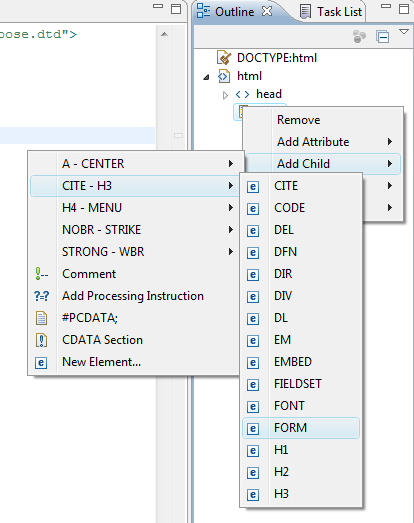
</body>

</html>

Note: If you copy/paste over an existing html and you get the page of the previous html refresh the page by clicking on: A refresh icon.

Note: If you prefer to make an html document from scratch follow the steps below to take advantage of eclipse facilities and learn.

a) On the right window right-click on the icon: **body 🡪 Add Child 🡪 CITE - H3 🡪 FORM** to see:



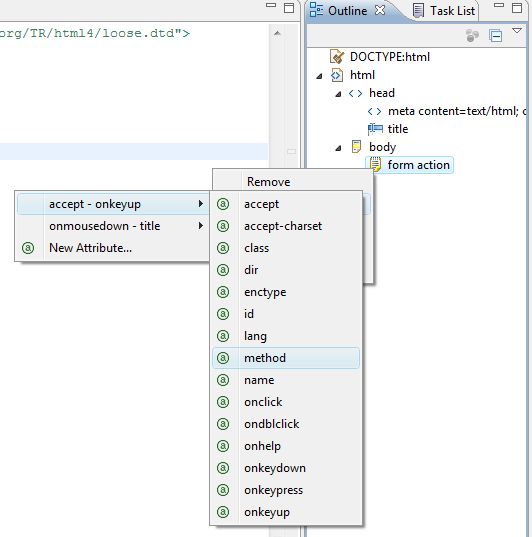
b) Select: FORM to see:

<body>

<form action=*""*></form>

</body>

c) On the right window right-click on: **form action** 🡪 **Add Attribute 🡪 accept - onkeyup 🡪 method** to see:



d) Select the method. Now the body of the html page is:

<body>

<form action=*""* method=*"get"*></form>

</body>

e) Now add: *"http://localhost"* inside: *""* for **action** to see:

<body>

<form action=*"http://localhost"* method=*"get"*></form>

</body>

You can continue until you have the above complete html document.

2. Run the html to see:

A screen capture of the Project Explorer menu outline as follows: 

1. Project
a. JAX-WS Web Services
b. Deployment Descriptor: Project
c. Java Resources: src
d. JavaScript Resources
e. build
f. Webcontent
i. META-INF
ii. WEB-INF
1. Main.html
2. Main.jsp
2. Servers

The right-hand page has two fields: "Enter Your Name" and "Enter Your Age."


## 4.2. Using Eclipse to add code to the JSP

We now try the second JSP.

### Example 2

Make the JSP to display a page similar to the one in example 1:

<%@ 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>JSP Using Eclipse</title>

</head>

<body>

<%

out.println("<form action=\"http://localhost\" method=\"post\">");

out.println("Enter Your Name: <input type=\"text\" name = \"yourName=\" </input><br>");

out.println("Enter Your Age&nbsp;&nbsp; : <input type=\"text\" name = \"yourAge=\" </input>");

%>

</body>

</html>

Remove the first semicolon to see how eclipse catches a syntax error (a red dot appears on the margin next to the line). Run the program. You should see the following message somewhere in the lower (console) window:

An error occurred at line: 11 in the jsp file: /Main.jsp

Syntax error, insert ";" to complete Statement

Put the semicolon back. Run the JSP. You should see the same page you saw in the html page.

### Example 3

Let us add a submit button to the HTML of example 1 and make a JSP to displays the message: Thank you

<!-- Main.html -->

<!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>A First page</title>

</head>

<body>

<form action="Thanks.jsp" method="get">

Enter Your Name: <input type="text" name="yourName"><br>

Enter Your Age&nbsp;&nbsp;&nbsp;&nbsp;:<input type="text" name="yourAge"><br>

<input type="submit" value="Sumitting">

</form>

</body>

</html>

Here is the JSP. Its name must be **Thanks.jsp**.

<!-- Thanks.jsp -->

<!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> A second page </title>

</head>

<body>

<h1>Thank you </h1>

</body>

</html>

Now this html and JSP communicate with each other. Look at the first html. The tag action points to the second html by:

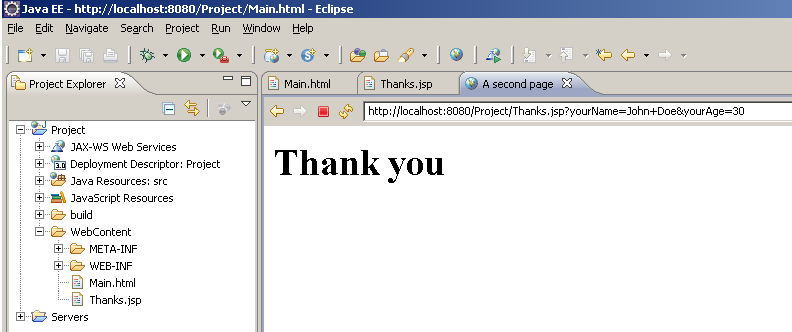
action="Thanks.jsp"

Run **Main.html** only (Do not run Thanks.html). You get the following page. Enter a name and an age in the input boxes:

The two fields from the previous screen capture have been filled in. 

Enter Your Name: John Doe
Enter Your Age: 30

Click on the submit button to see:



Interesting! What happened? We told the **Main.html** that when we click on the submit button send the HTTP *requests* a query from the server by sending contents of the input boxes to the **Thanks.jsp** which is on the Apache tomcat server by the instruction action="Thanks.jsp". Once the **Thanks.jsp** document gets this message it responds by displaying the message **Thank you**. In fact the **Main.html** does not have to be on the server.

The **Main.html** acts as a client and the **Thanks.jsp** as a server program. Take a look at the server address. It is written:

<http://localhost:8080/Project/Thanks.jsp?yourName=John+Doe&yourAge=30>

The client included the pairs (Name, Value) of each input box to the end of the link:

Name: **yourName** Value: **John+Doe**

Name: **yourAge** Value: **30**

**Methods GET and POST**

The HTTP request can be in two forms: GET and POST. In the Main.html it is: method="get". The GET/POST requester (the client) is requesting for a file on the server (in this case **Thanks.jsp**) and supplies input data to this file by sending a sequence of (Name, Value)’s. The pairs of (Name, Value) are appended at the end of the URL (for our example it is: [**http://localhost:8080/Project/Thanks.jsp**](http://localhost:8080/Project/Thanks.jsp)). Note that this is the case for the GET request. For the POST request this appending does not show up. The server processes these pairs of (Name, Value). In our example the server program does not do anything with these pairs (we did not refer to any of these pairs in **Thanks.jsp**). Therefore:

The client needs to send the input information to the server.

The server needs to use the information to create a dynamic response.

**Exercise**: In the Main.html change method="get" to method="post" and run Main.html to see: the pairs of (Name, Value) do not append. You should see:

<http://localhost:8080/Project/Thanks.jsp>

Note: Eclipse sometimes does not show this in the first run. In that case run it twice.

## 4.3. Using Eclipse to add code to the Servlet

Replace the Servlet with the following and run the server. You should get the same web page as the ones in the HTML document and JSP (sec 2.1 & 2.2):

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** java.io.\*;

@WebServlet("/Main")

**public** **class** Main **extends** HttpServlet {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**public** Main() {

**super**();

}

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse

response) **throws** ServletException, IOException {

PrintWriter out = response.getWriter(); //Is added

out.println("<form action=\"http://localhost\" method=\"post\">");

out.println("Enter Your Name: <input type=\"text\" name " +

"= \"yourName=\" </input><br>");

out.println("Enter Your Age&nbsp;&nbsp; : <input " +

"type=\"text\" name = \"yourAge=\" </input>");

out.close();

}

}

I removed the comments. I also removed the method: doPost because its body was empty.

**Description**: The calls to out.println’s makes an html document and displays on the screen. With respect to JSP we added two lines:

PrintWriter out = response.getWriter();

**import** java.io.\*;

A servlet is a java program. The first instruction is declaring an output stream. An output stream that writes to the web page. If you have had a java program that writes to an output file this should be familiar.

The JDK class **PrintWriter** is in the package **java.io.\***. Therefore we should import this package. The rest of the body of the method doGet is the same as the one in the JSP program (section 2.2).

The methods **doGet** and **doPost** are overridden to the ones in class:

javax.servlet.ServletException;

These methods are automatically called from the JDK super class **HttpServlet** to handle a GET/POST request.

The HyperText Transfer Protocol ("HTTP ") is a request-response oriented protocol. The request data is passed to the first parameter of doGet or doPost method and the response data is passed to the second argument of these methods. With a doGet request the parameters are appended at the end of the URL, but not with a doPost method.

Note that in this example we do not have a request from a client (from an html document) to this servlet.

Note: It is always better to close **out.close()** the I/O stream; especially it is necessary for the output stream.

# 5. JSPs vs Servlets

|  |  |
| --- | --- |
| **JSP** | **Servlet** |
| JSP is HTML based code. | Servlet is java code. |
| JSP is easy to code as it is java in html. | Writing code for servlet is harder than JSP as it is html in java. |
| JSP only accept http requests. | Servlet can accept all protocol requests. |
| JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to java code and then compile. | Servlet is faster than JSP. |
| In JSP business logic is separated from presentation logic by using javaBeans. | In Servlet we have to implement everything like business logic and presentation logic in just one servlet file. |