

# **ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY**

Kanyakumari Main Road, near Anjugramam, Palkulam, Anjugramam, Tamil Nadu 629401

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATION COURSE**

**ON**

**JAVA SERVER PAGES**

**COURSE MATERIAL**

# JAVA SERVER PAGES

**JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc. A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

## **Advantages of JSP over Servlet**

There are many advantages of JSP over the Servlet. They are as follows:

### **1) Extension to Servlet**

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

### **2) Easy to maintain**

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

### **3) Fast Development: No need to recompile and redeploy**

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

### **4) Less code than Servlet**

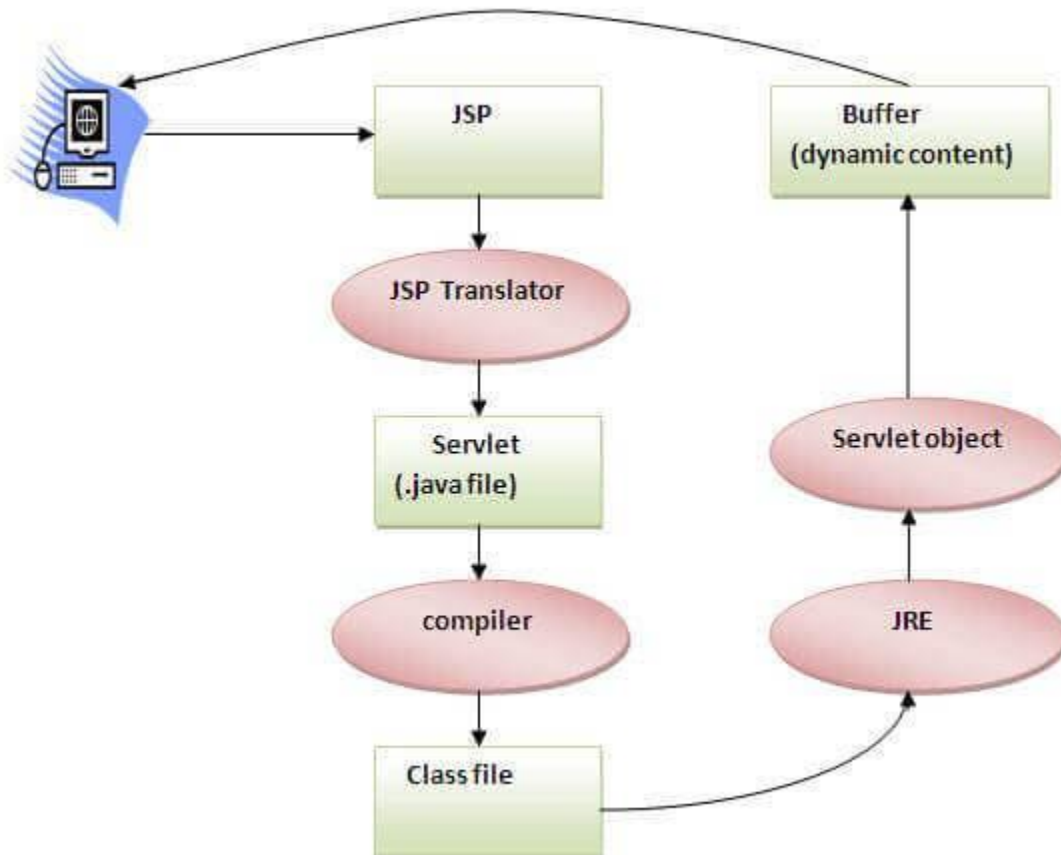
In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization ( the container invokes `jspInit()` method).
- Request processing ( the container invokes `_jspService()` method).
- Destroy ( the container invokes `jspDestroy()` method).

`jspInit()`, `_jspService()` and `jspDestroy()` are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

### **Creating a simple JSP Page**

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

#### **index.jsp**

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

It will print **10** on the browser.

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL <http://localhost:portno/contextRoot/jspfile>, for example, <http://localhost:8888/myapplication/index.jsp>

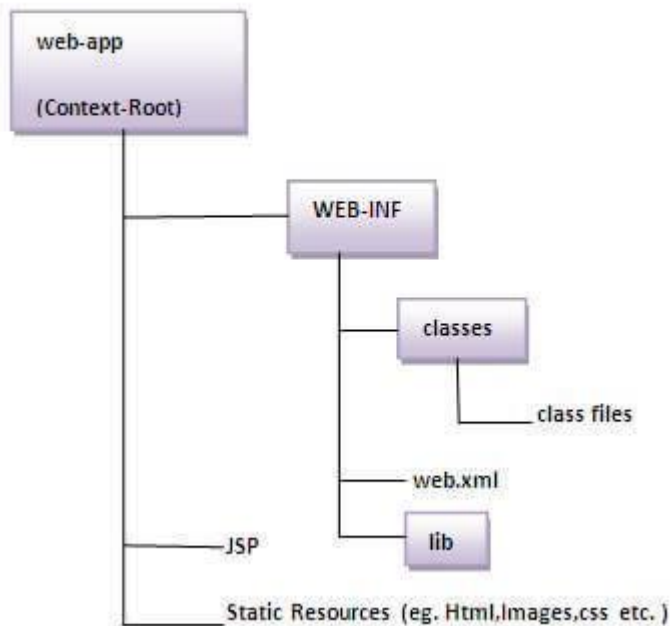
Do I need to follow the directory structure to run a simple JSP?

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

---

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



## **JSP Index**

### **JSP Tutorial**

- Life cycle of JSP
- JSP API
- JSP in Eclipse

### **JSP scripting elements**

- JSP scriptlet tag
- JSP expression tag
- JSP declaration tag

### **9 Implicit Objects**

#### JSP Out

- JSP Request
- JSP Response
- JSP Config
- JSP Application
- JSP Session
- JSP PageContext
- JSP Page
- JSP Exception

### **JSP Directive Elements**

- JSP page directive
- JSP include directive
- JSP taglib directive

## **JSP Exception**

### **Action Elements**

- jsp:forward
- jsp:include
- Java Bean class
- jsp:useBean
- set & getProperty
- Displaying applet in JSP

### **Expression Language**

### **MVC in JSP**

### **JSTL**

#### **JSP Custom tags**

- Example of Custom Tag
- Attributes
- Iteration
- Custom URI

#### **JSP Pagination**

- JSP Pagination Example

#### **JSP CRUD**

- JSP CRUD Example

#### **Development in JSP**

- Registration Form

- Login Form
- Uploading File
- Downloading File

## **The JSP API**

1. The JSP API
2. javax.servlet.jsp package
3. The JspPage interface
4. The HttpJspPage interface

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

## **javax.servlet.jsp package**

The javax.servlet.jsp package has two interfaces and classes. The two interfaces are as follows:

1. JspPage
2. HttpJspPage

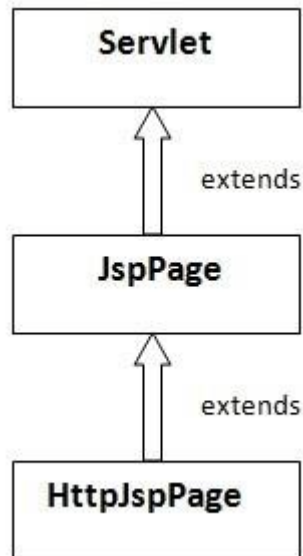
The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspError



## The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the `init()` method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

## The HttpJspPage interface

The `HttpJspPage` interface provides the one life cycle method of JSP. It extends the `JspPage` interface.

Method of `HttpJspPage` interface:

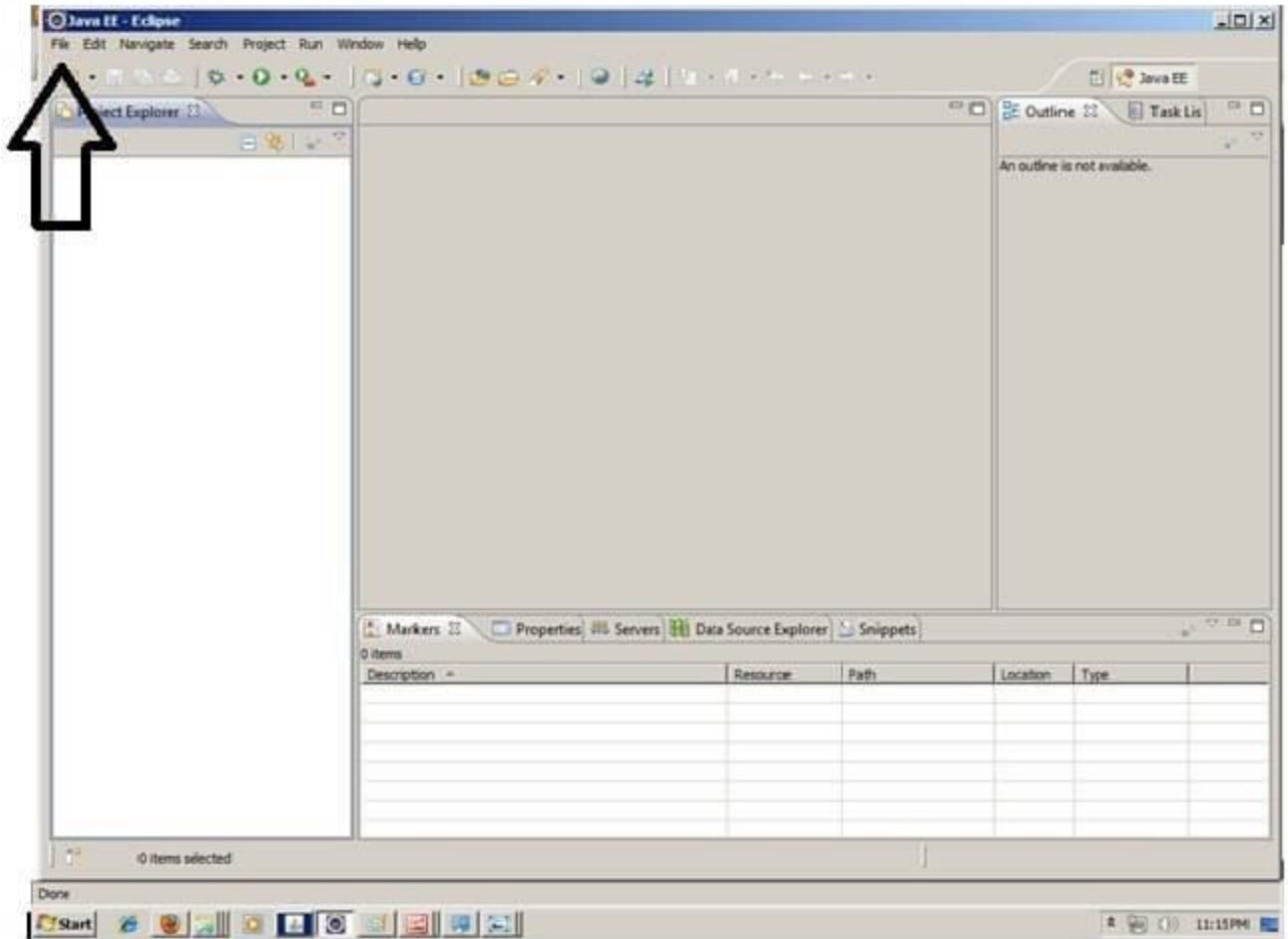
1. **public void \_jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore \_ signifies that you cannot override this method.

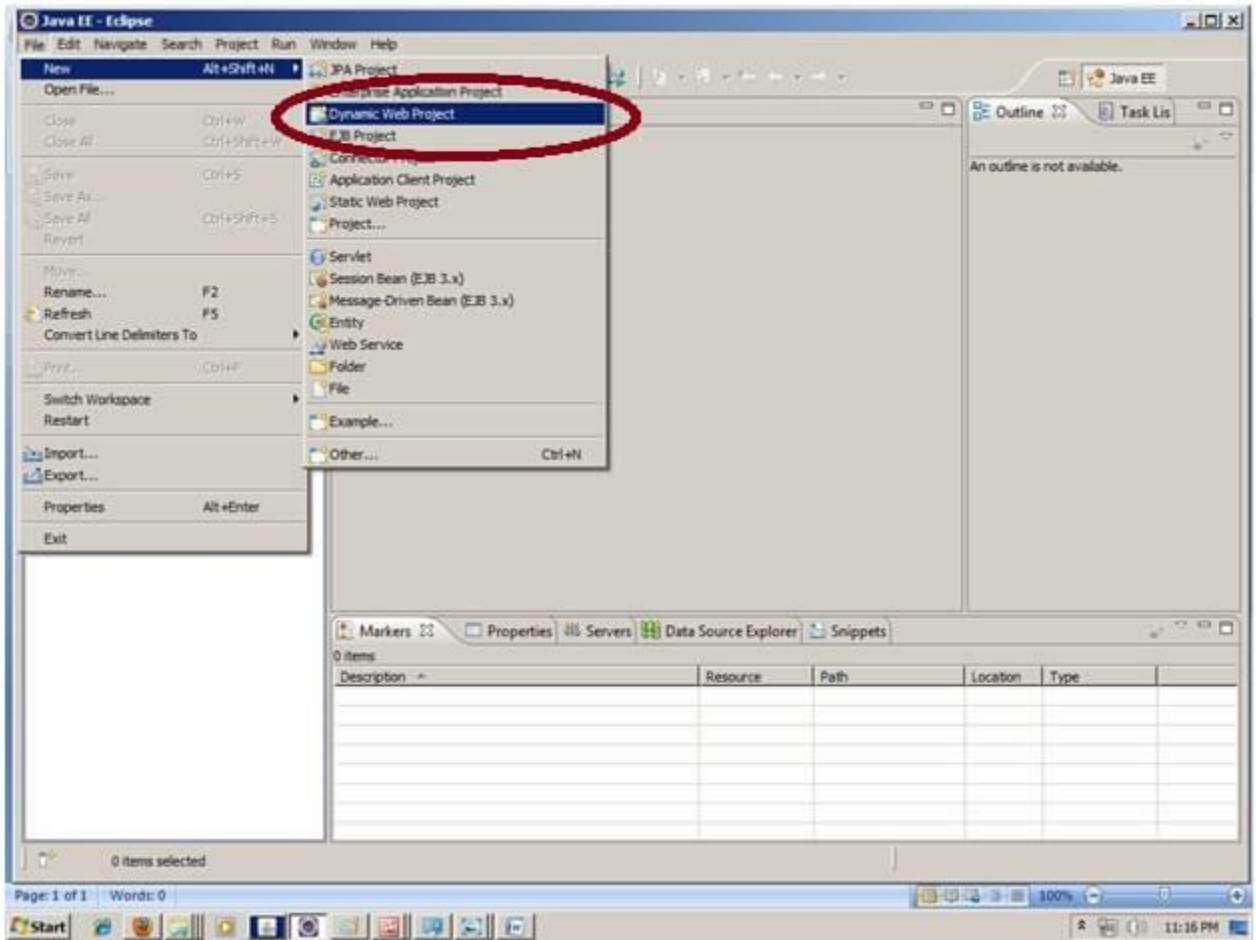
### **Creating JSP in Eclipse IDE with Tomcat server**

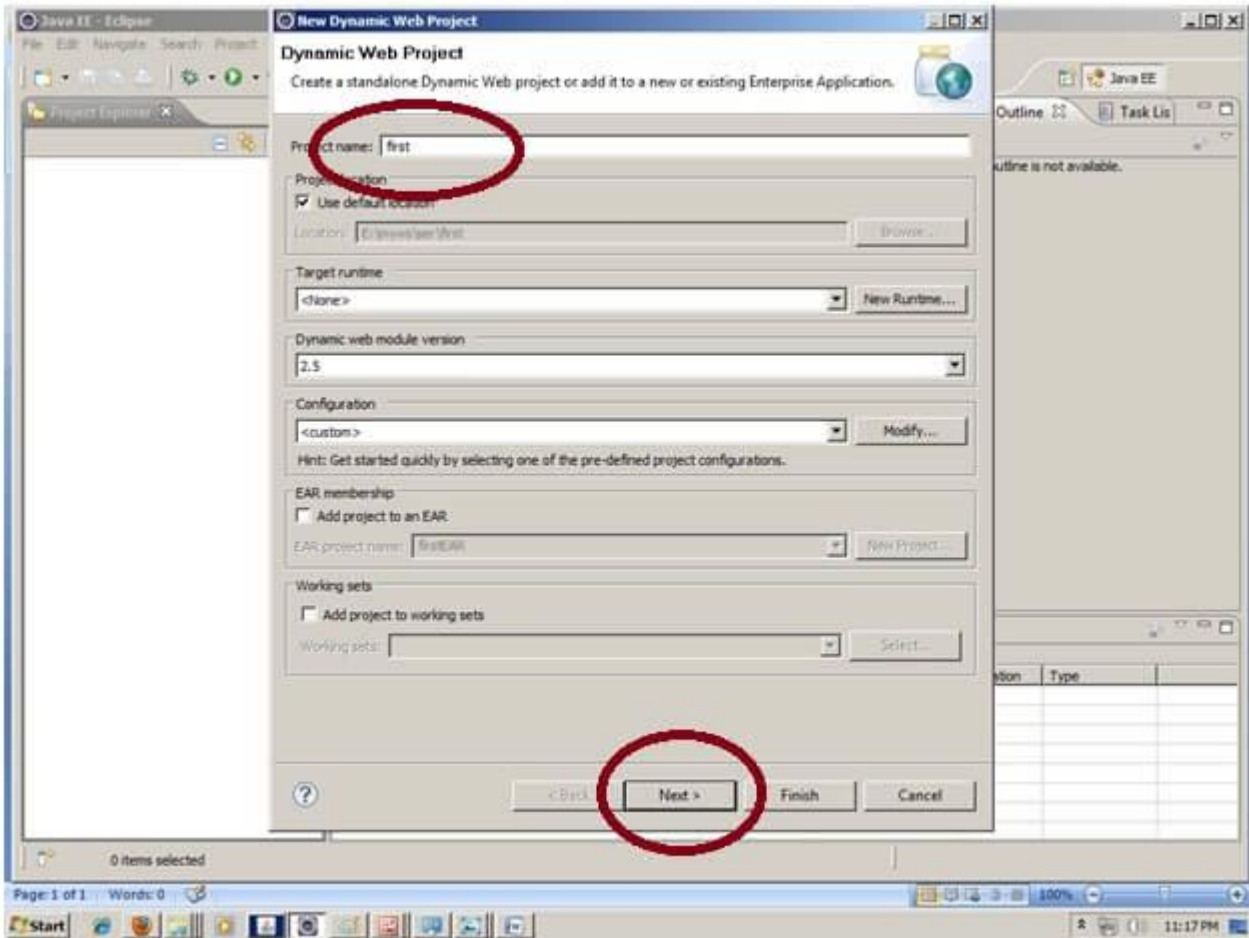
1. Creating JSP in Eclipse IDE with Tomcat
  1. Create a Dynamic web project
  2. create a jsp
  3. start tomcat server and deploy the project
    - o Create a Dynamic web project
    - o create a jsp
    - o start tomcat server and deploy the project

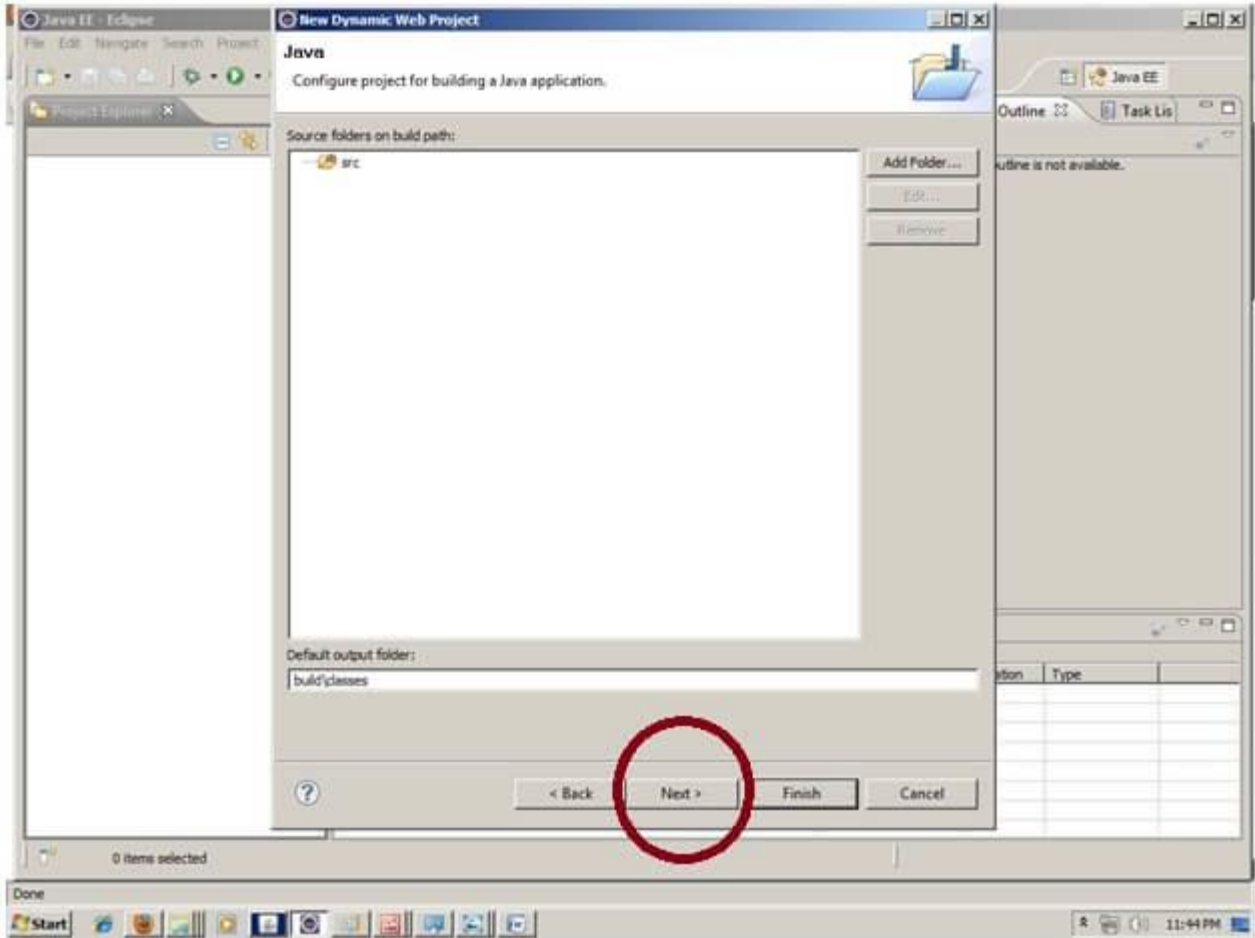
1) Create the dynamic web project

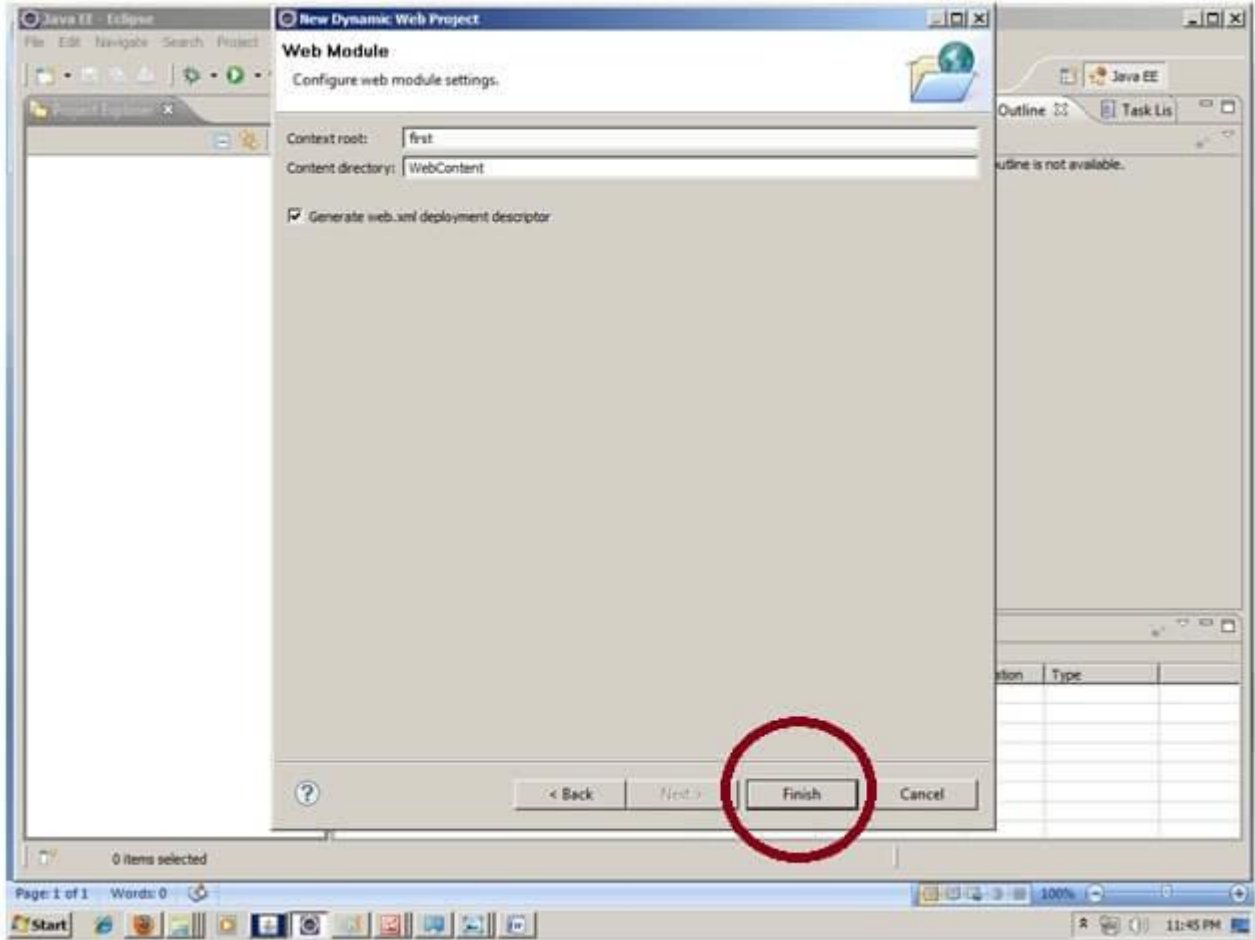
For creating a dynamic web project click on File Menu -> New -> dynamic web project -> write your project name e.g. first -> Finish.

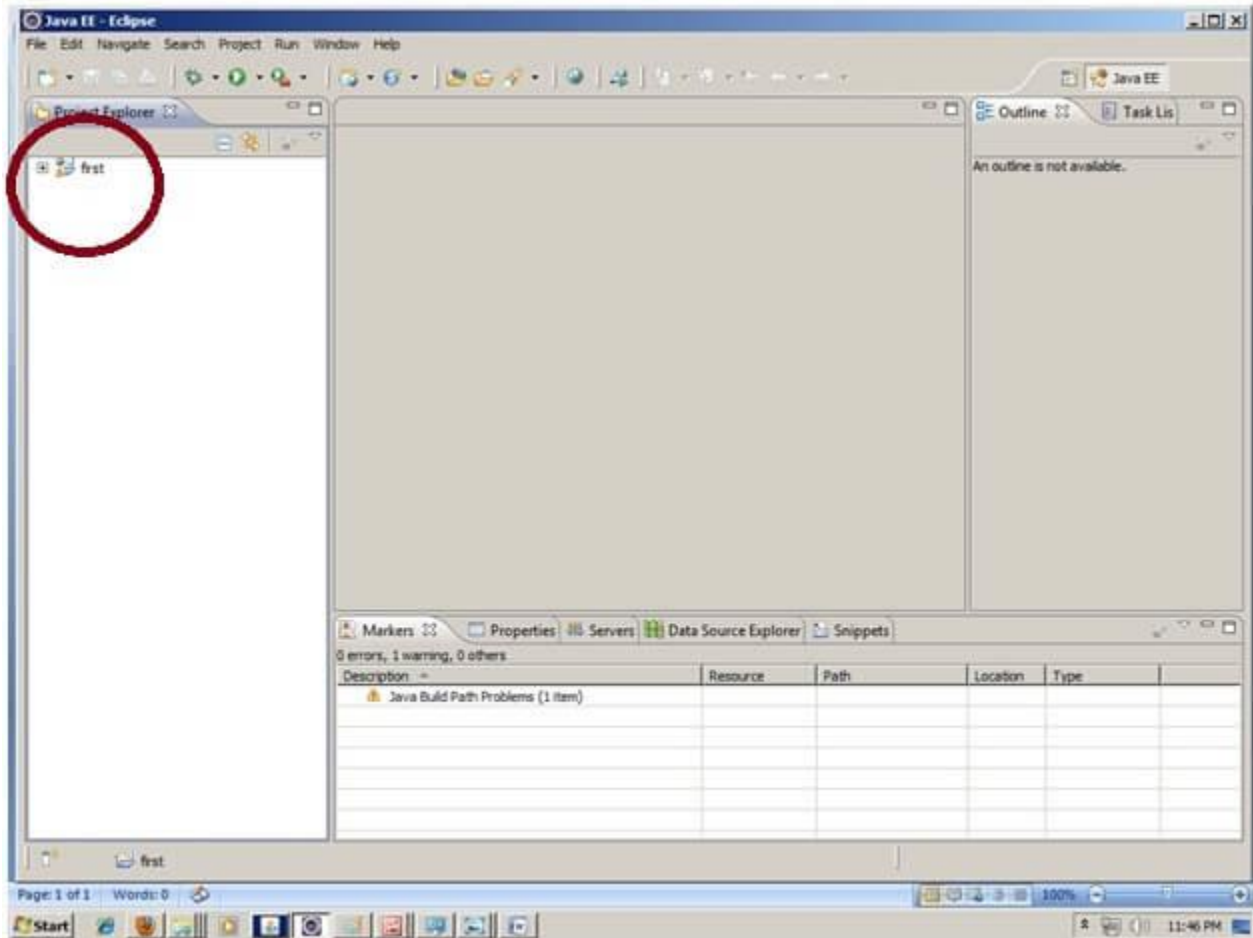








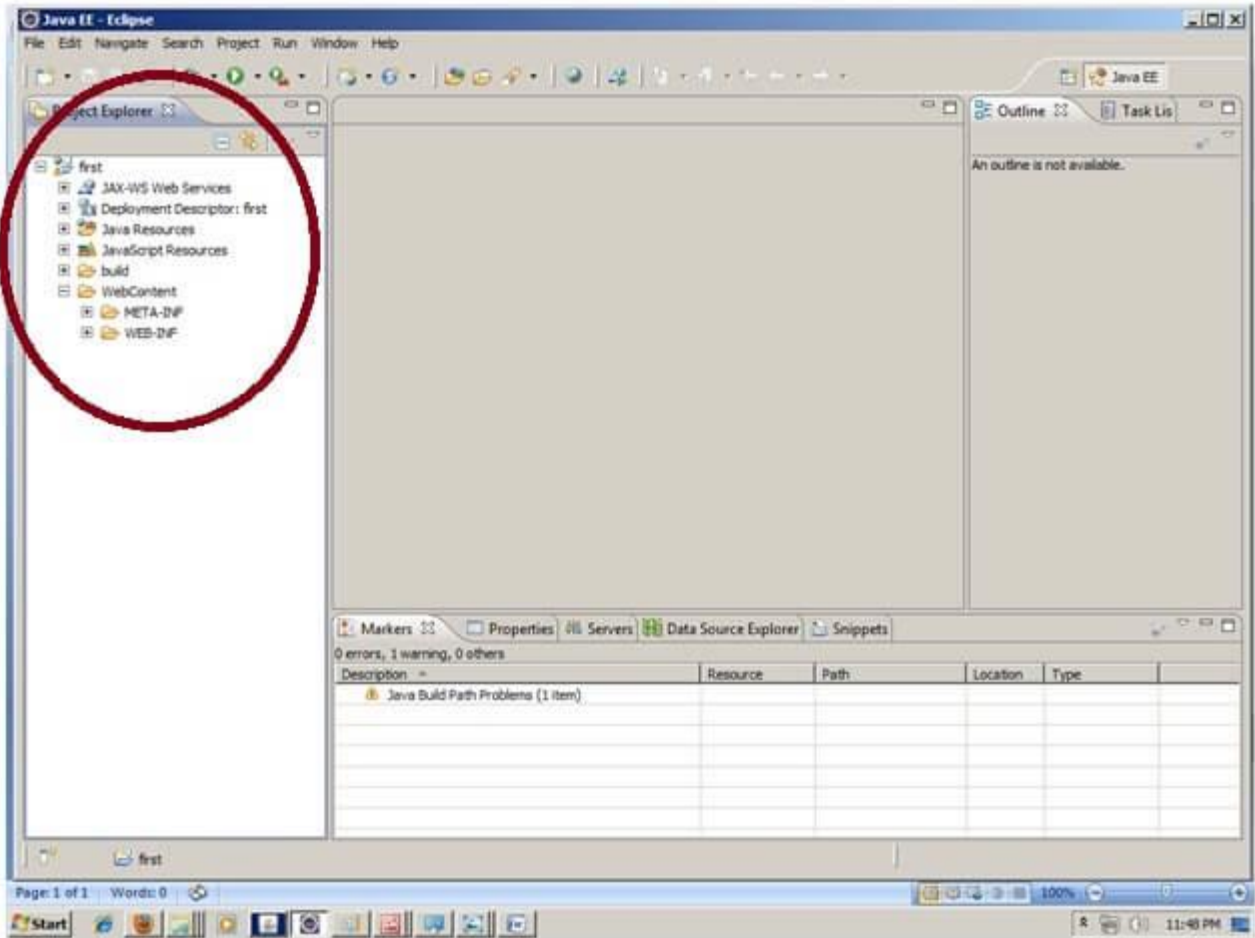


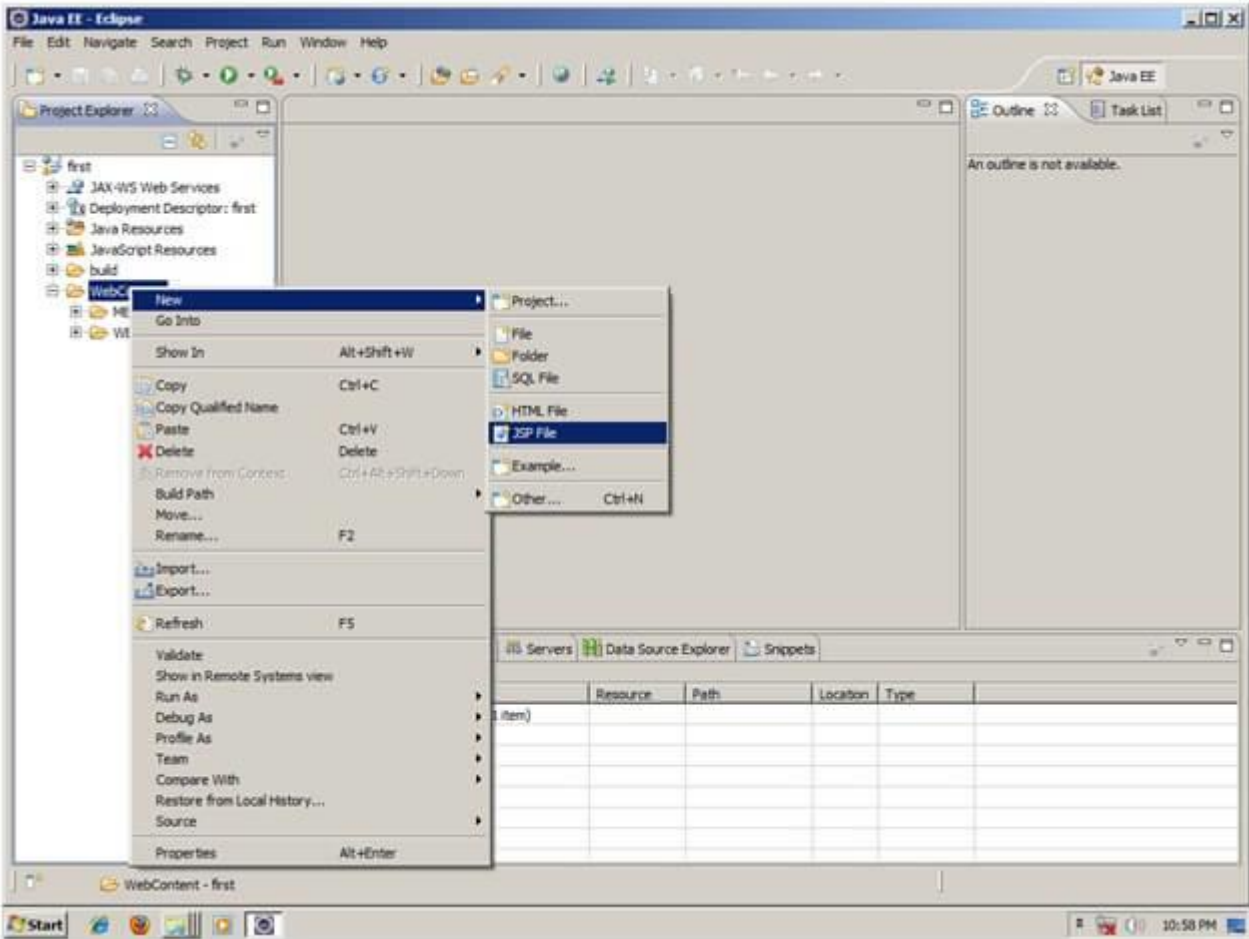


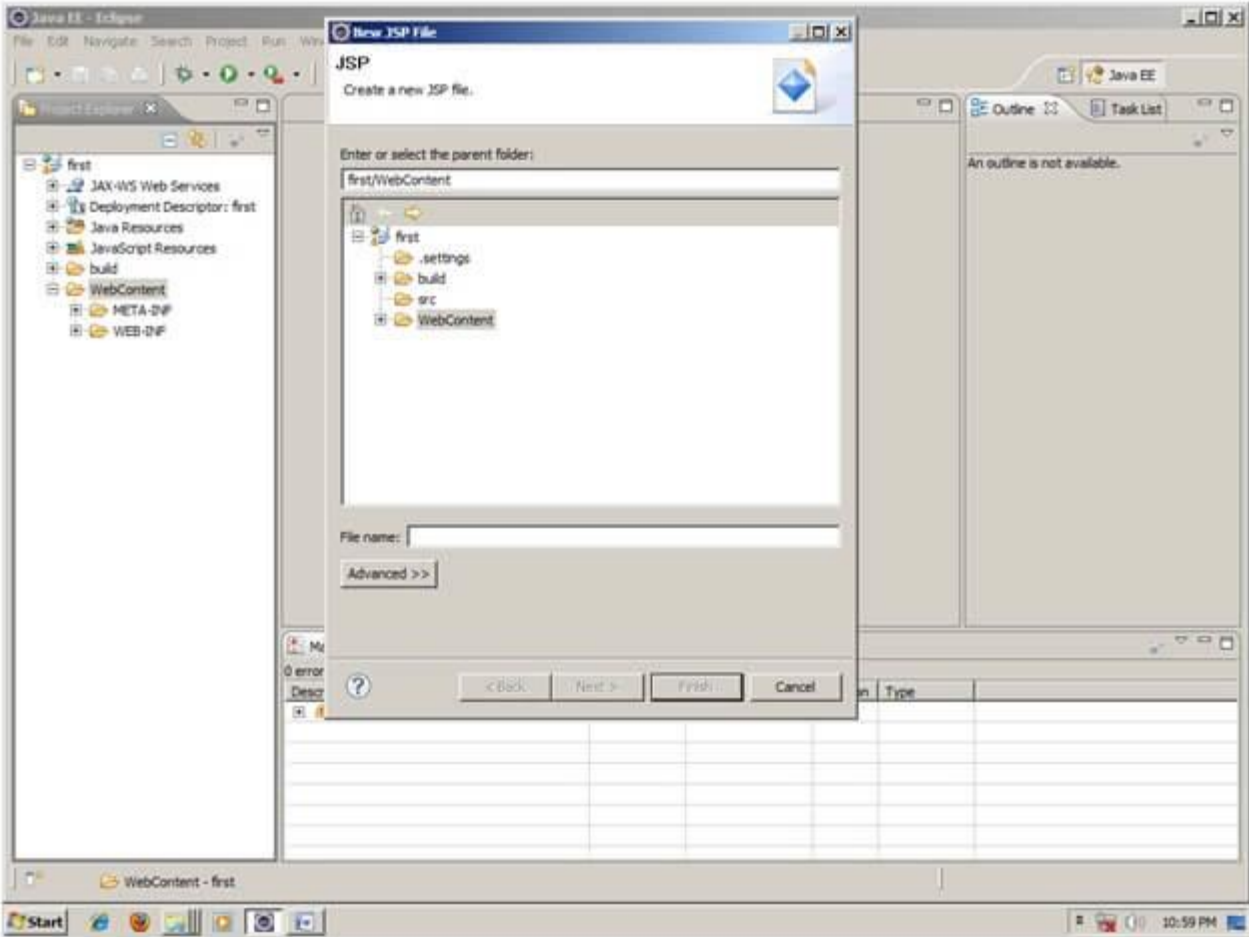
## 2) Create the JSP file in eclipse IDE

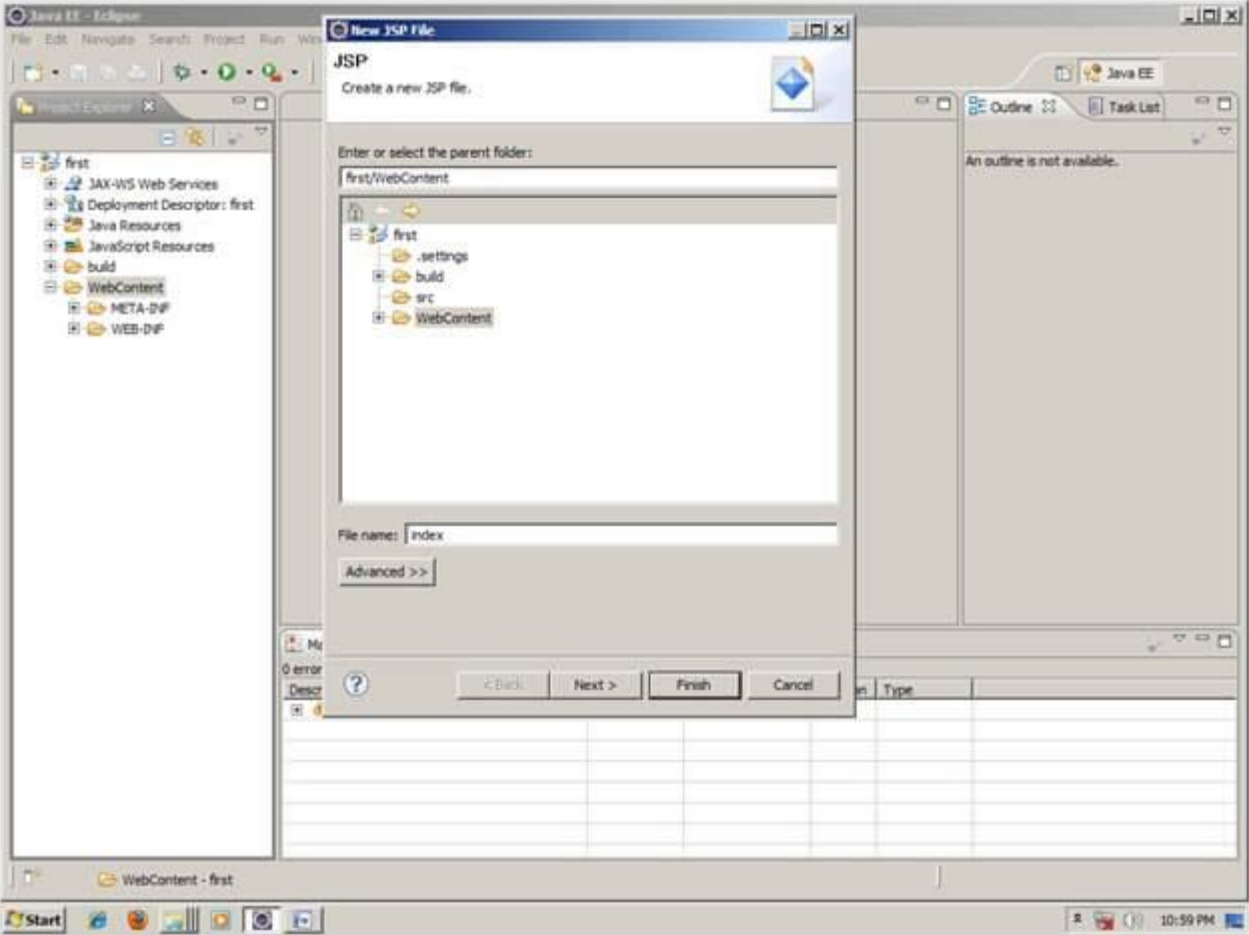
For creating a jsp file explore the project by clicking the + icon -> right click on WebContent -> New -> jsp -> write your jsp file name e.g. index -> next -> Finish.

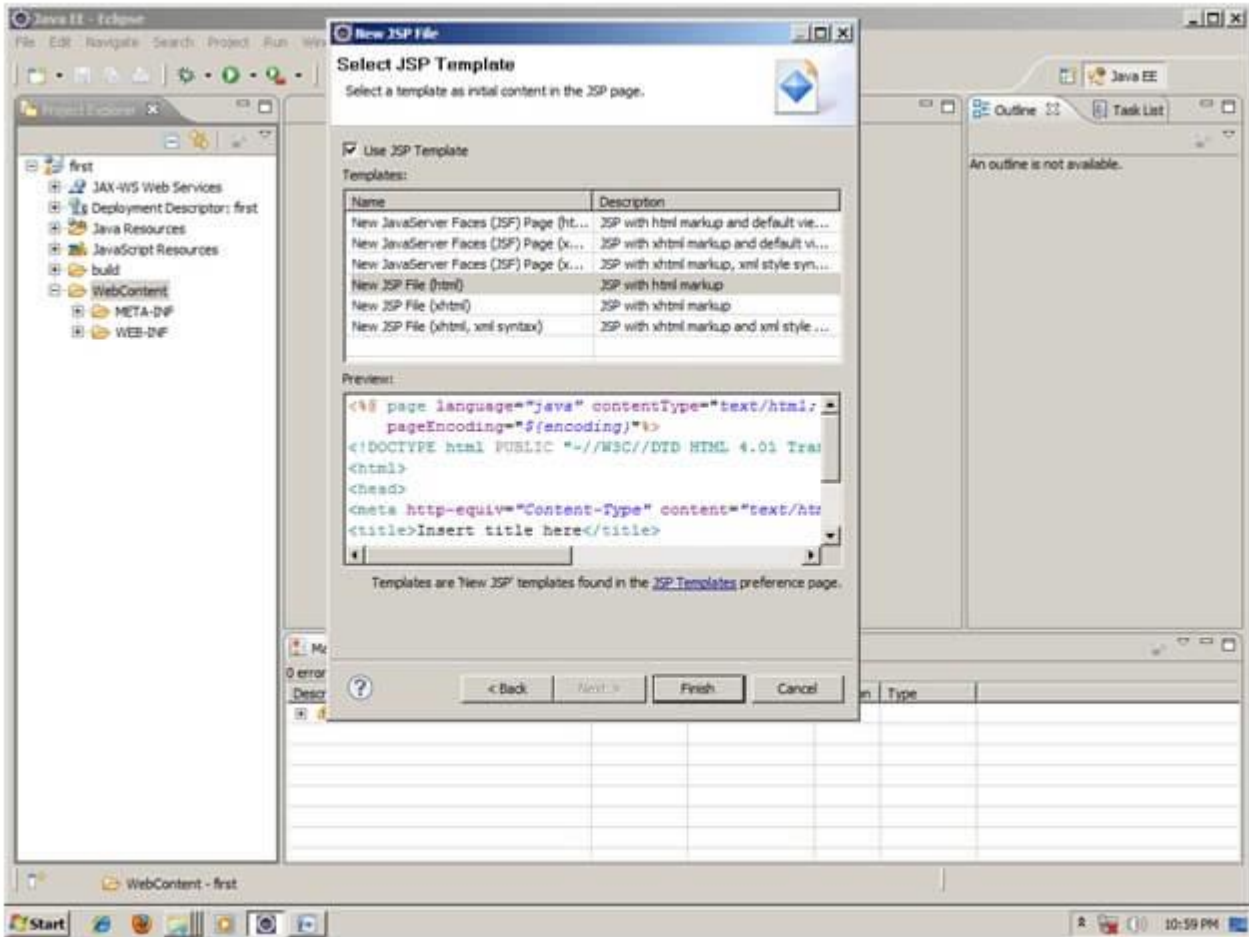


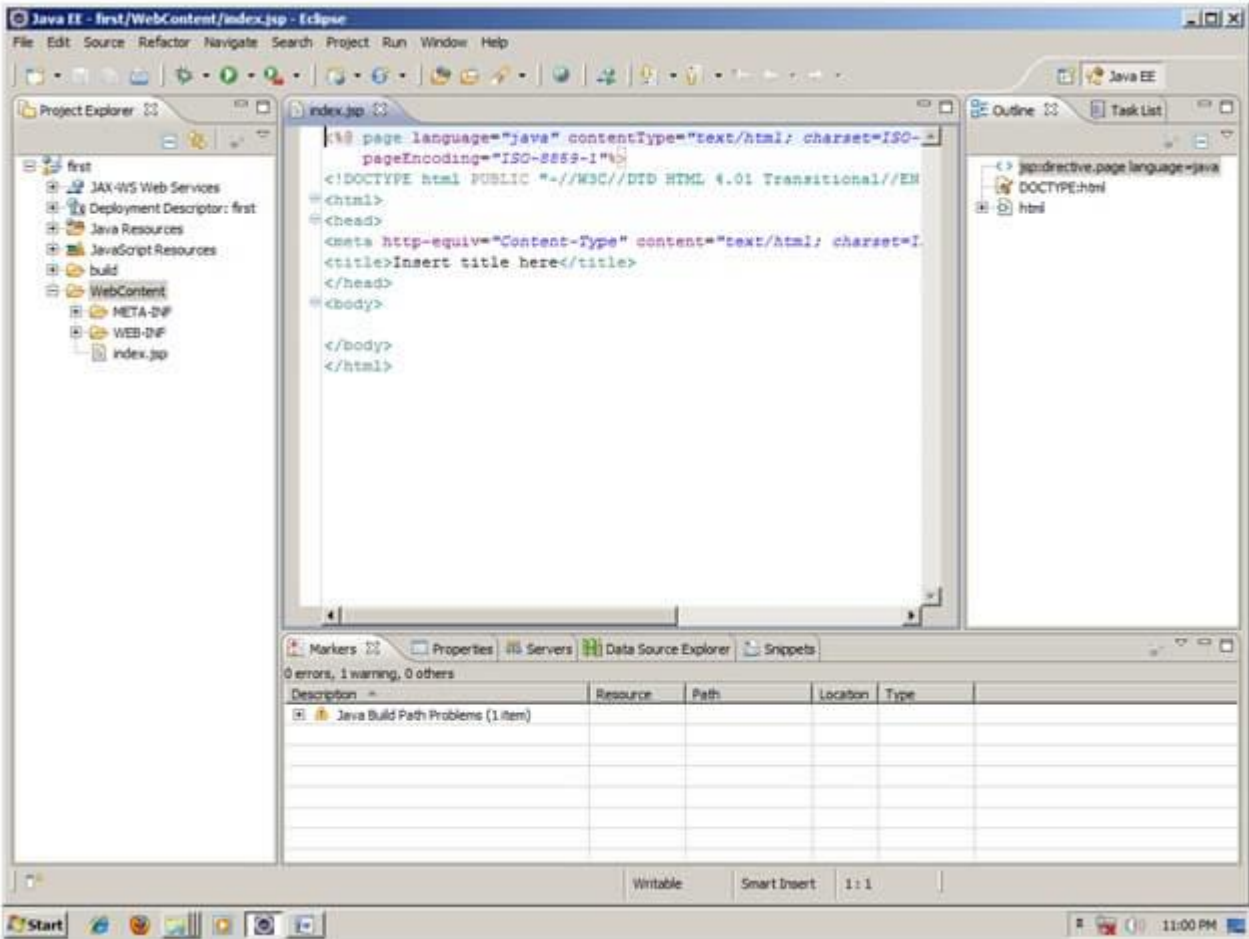




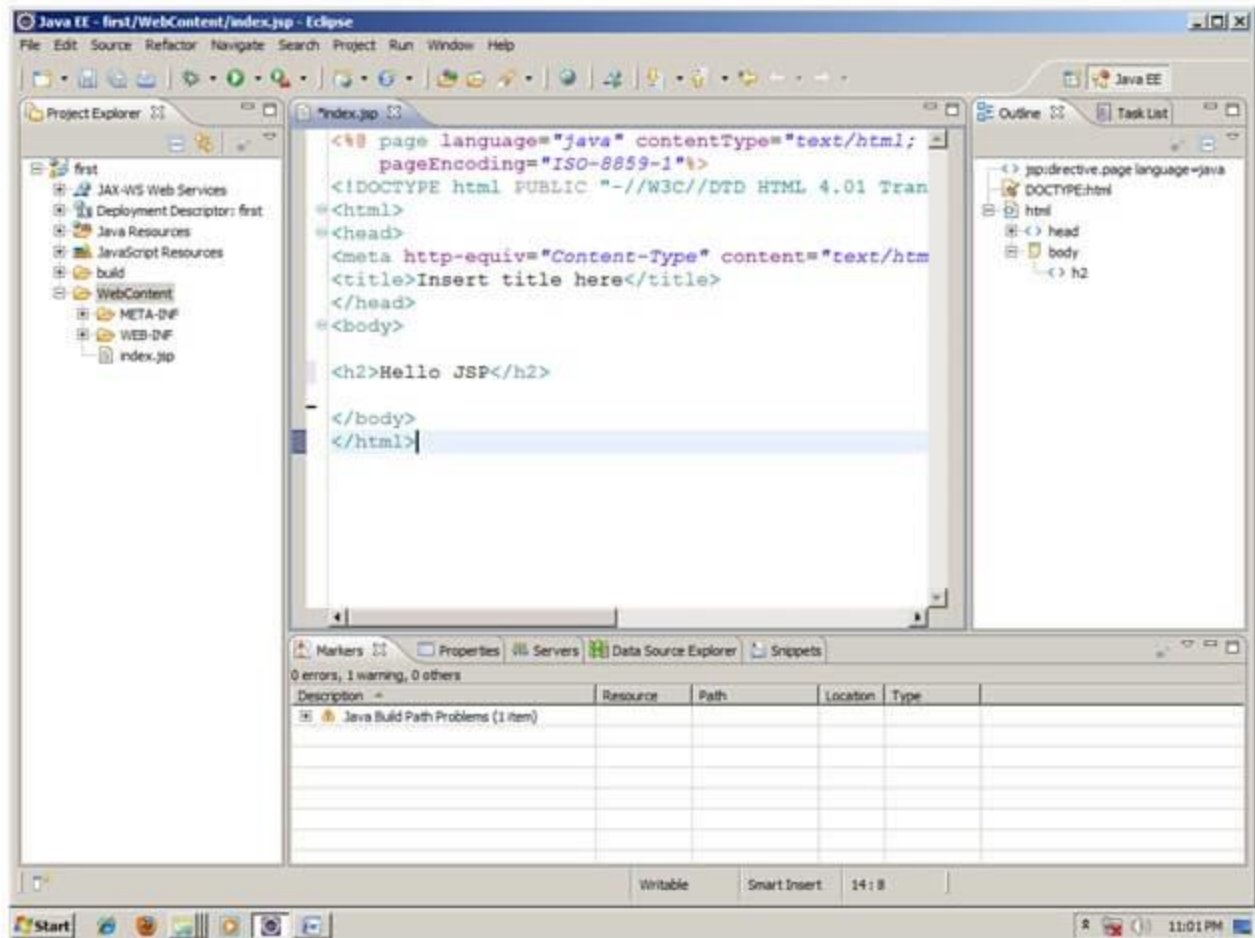








Now JSP file is created, let's write some code.



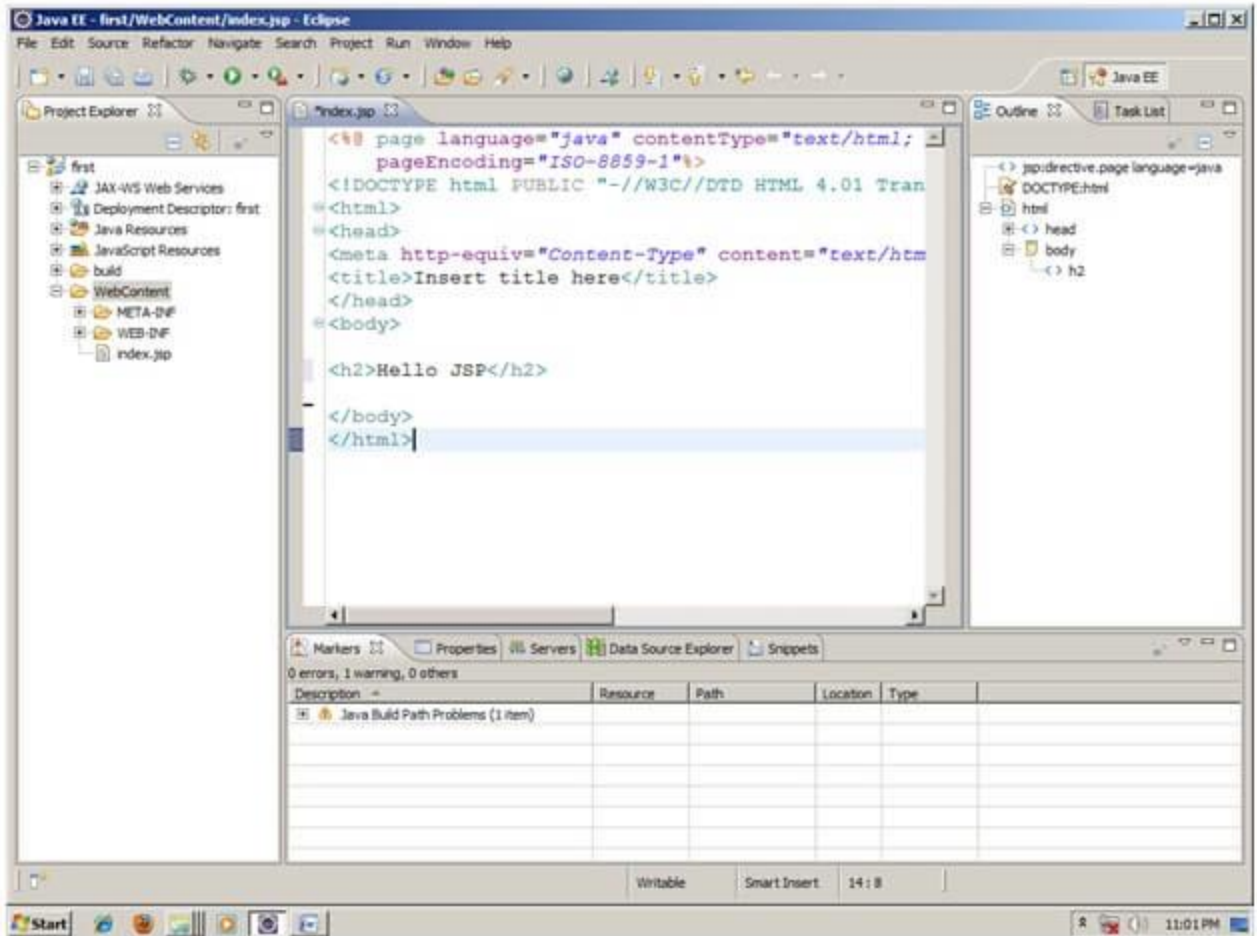
### 3) Start the server and deploy the project:

For starting the server and deploying the project in one step Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.

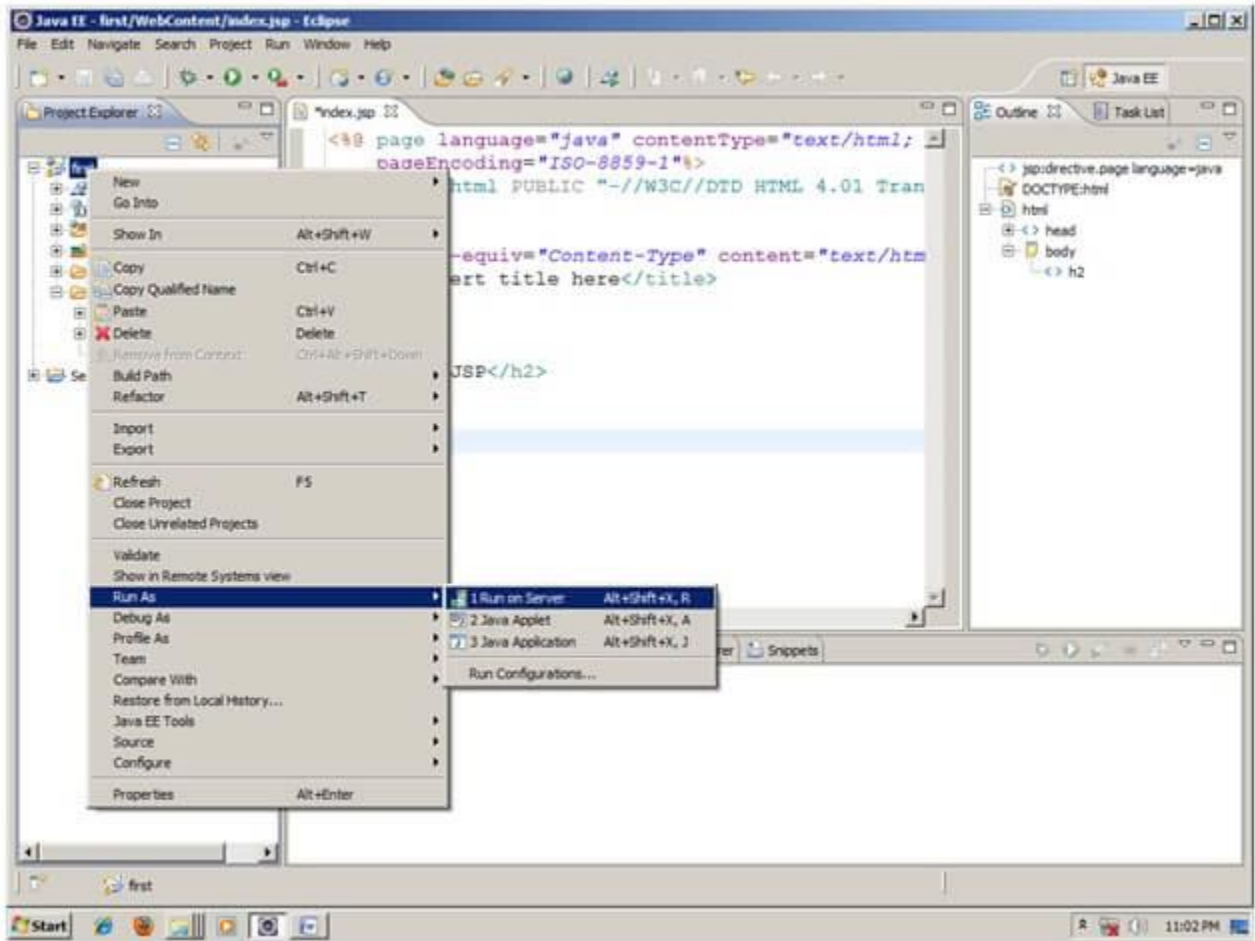
If you are using Eclipse IDE first time, you need to configure the tomcat server First. Click for How to configure tomcat server in eclipse IDE

Now start the tomcat server and deploy project

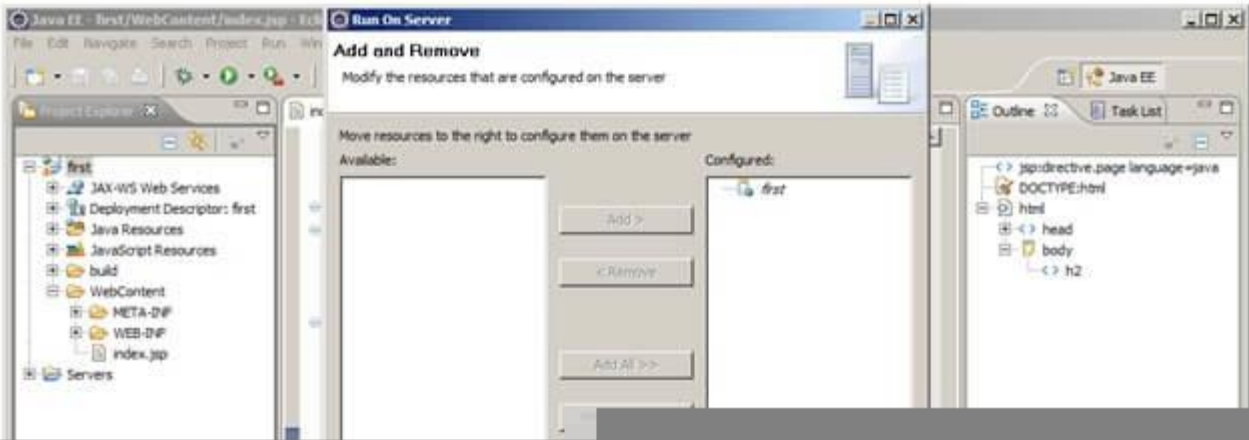
For starting the server and deploying the project in one step Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.



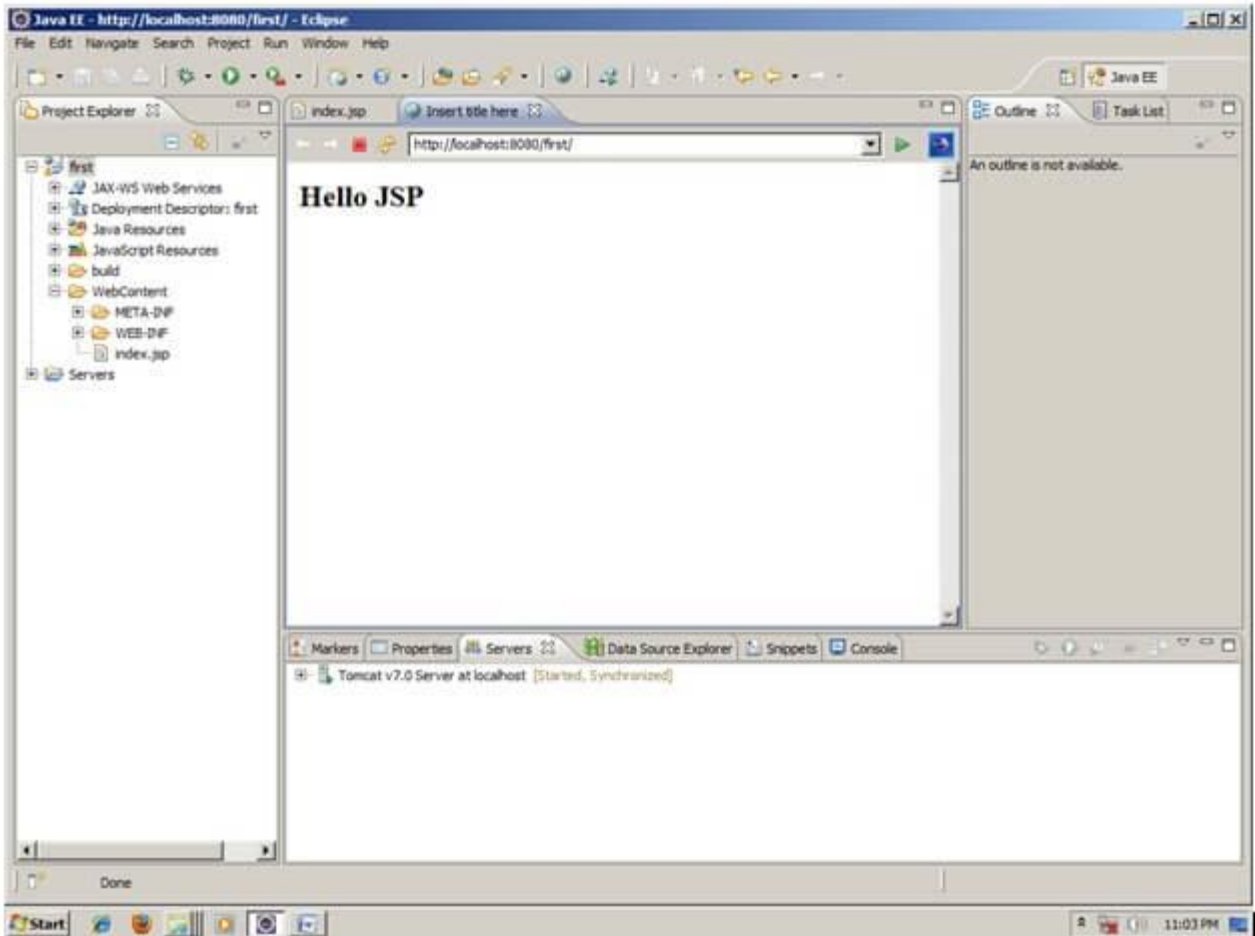








Yes, Let's see JSP is successfully running now.



## JSP Scriptlet tag (Scripting elements)

1. Scripting elements
2. JSP scriptlet tag
3. Simple Example of JSP scriptlet tag
4. Example of JSP scriptlet tag that prints the user name

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

## JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

### JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

1. `<% java source code %>`

#### Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>  
<body>  
<% out.print("welcome to jsp"); %>  
</body>  
</html>
```

---

#### Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

*File: index.html*

```
<html>  
<body>  
<form action="welcome.jsp">  
<input type="text" name="uname">  
<input type="submit" value="go"><br/>
```

**</form>**

**</body>**

**</html>**

*File: welcome.jsp*

**<html>**

**<body>**

**<%**

**String name=request.getParameter("uname");**

**out.print("welcome "+name);**

**%>**

**</form>**

**</body>**

**</html>**

### **JSP expression tag**

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

1. **<%= statement %>**

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

**<html>**

**<body>**

**<%= "welcome to jsp" %>**

**</body>**

**</html>**

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the `getTime()` method of `Calendar` class. The `getTime()` is an instance method of `Calendar` class, so we have called it after getting the instance of `Calendar` class by the `getInstance()` method.

*index.jsp*

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The `index.html` file gets the username and sends the request to the `welcome.jsp` file, which displays the username.

*File: index.jsp*

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

*File: welcome.jsp*

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

### JSP Declaration Tag

1. JSP declaration tag
2. Difference between JSP scriptlet tag and JSP declaration tag
3. Example of JSP declaration tag that declares field
4. Example of JSP declaration tag that declares method

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

1. <%! field or method declaration %>

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.



The declaration of scriptlet tag is placed inside the `_jspService()` method.

The declaration of jsp declaration tag is placed outside the `_jspService()` method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
```

```
<%= "Cube of 3 is:"+cube(3) %>
```

```
</body>
```

```
</html>
```

### JSP Implicit Objects

1. JSP Implicit Objects
2. out implicit object
3. Example of out implicit object

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext

page	Object
exception	Throwable

### 1) JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

```
PrintWriter out=response.getWriter();
```

But in JSP, you don't need to write this code.

Example of out implicit object

In this example we are simply displaying date and time.

index.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Output

### **JSP request implicit object**

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example of JSP request implicit object

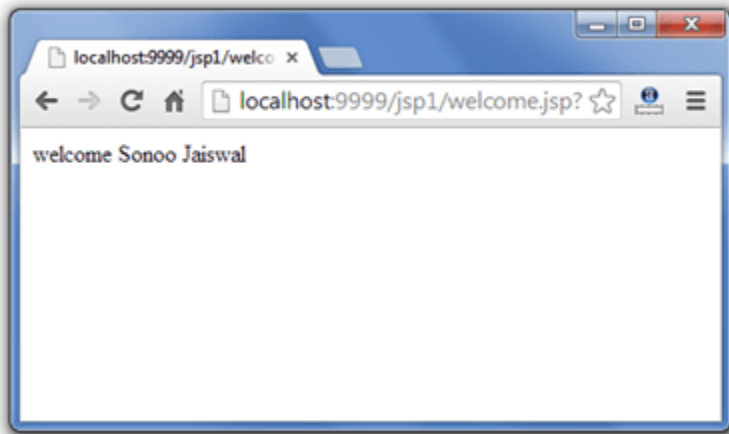
### **index.html**

```
<form action="welcome.jsp">  
<input type="text" name="uname">  
<input type="submit" value="go"><br/>  
</form>
```

### **welcome.jsp**

```
<%  
String name=request.getParameter("uname");  
out.print("welcome "+name);  
%>
```

Output



### 3) JSP response implicit object

In JSP, response is an implicit object of type `HttpServletResponse`. The instance of `HttpServletResponse` is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example of response implicit object

### index.html

```
<form action="welcome.jsp">  
<input type="text" name="uname">  
<input type="submit" value="go"><br/>  
</form>
```

### welcome.jsp

```
<%  
response.sendRedirect("http://www.google.com");  
%>
```

Output



### 4) JSP config implicit object

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

Example of config implicit object:

### **index.html**

```
<form action="welcome">  
<input type="text" name="uname">  
<input type="submit" value="go"><br/>  
</form>
```

### **web.xml file**

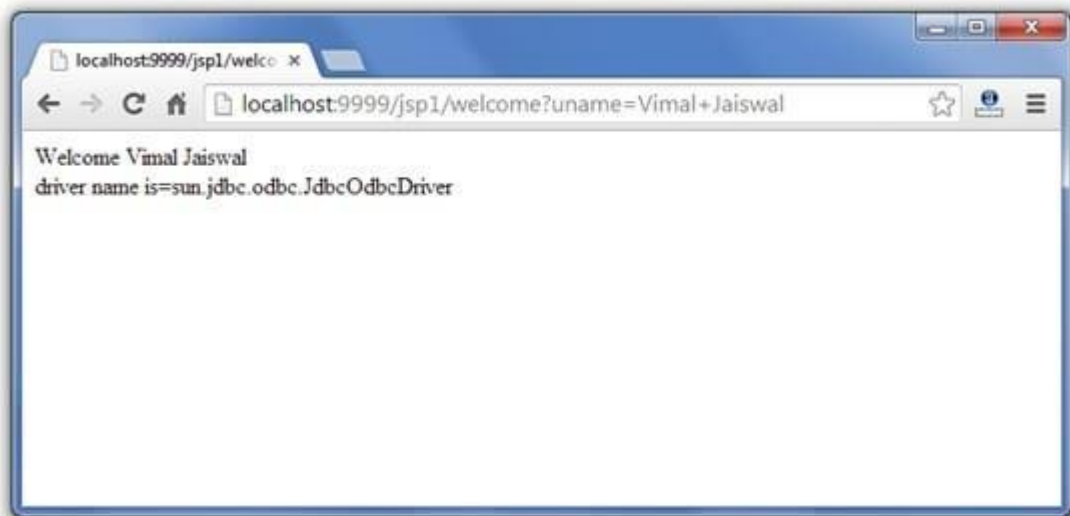
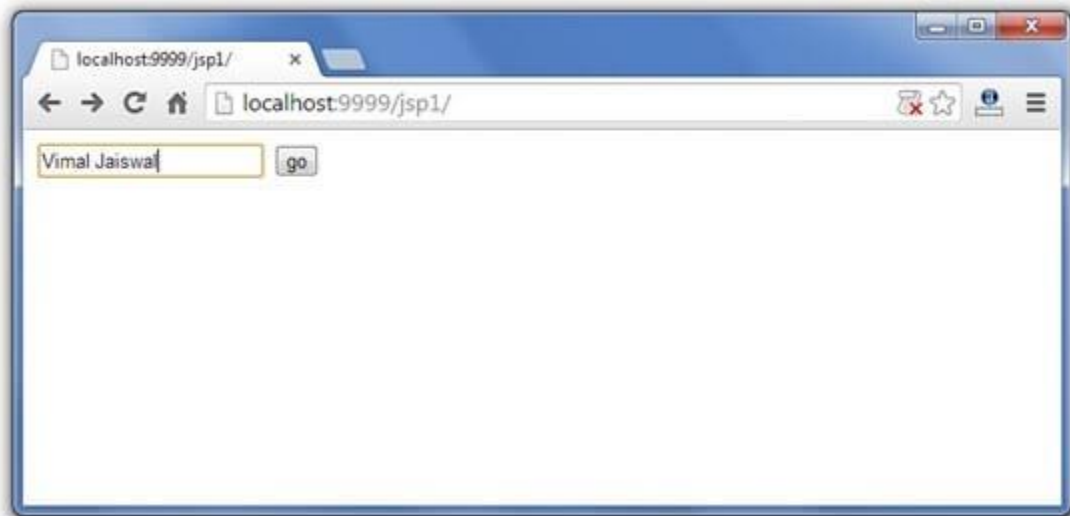
```
<web-app>  
  
<servlet>  
<servlet-name>sonoojaiswal</servlet-name>  
<jsp-file>/welcome.jsp</jsp-file>  
  
<init-param>  
<param-name>dname</param-name>  
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>  
</init-param>  
  
</servlet>  
  
<servlet-mapping>  
<servlet-name>sonoojaiswal</servlet-name>  
<url-pattern>/welcome</url-pattern>  
</servlet-mapping>  
  
</web-app>
```

### **welcome.jsp**

```
<%
```

```
out.print("Welcome "+request.getParameter("uname"));  
String driver=config.getInitParameter("dname");  
out.print("driver name is="+driver);  
%>
```

Output





## 5) JSP application implicit object

In JSP, application is an implicit object of type *ServletContext*.

The instance of *ServletContext* is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

Example of application implicit object:

### index.html

```
<form action="welcome">  
<input type="text" name="uname">  
<input type="submit" value="go"><br/>  
</form>
```

### web.xml file

```
<web-app>  
  
<servlet>  
<servlet-name>sonoojaiswal</servlet-name>  
<jsp-file>/welcome.jsp</jsp-file>  
</servlet>  
  
<servlet-mapping>  
<servlet-name>sonoojaiswal</servlet-name>  
<url-pattern>/welcome</url-pattern>  
</servlet-mapping>
```

```
<context-param>  
<param-name>dname</param-name>  
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>  
</context-param>
```

```
</web-app>
```

```
welcome.jsp
```

```
<%
```

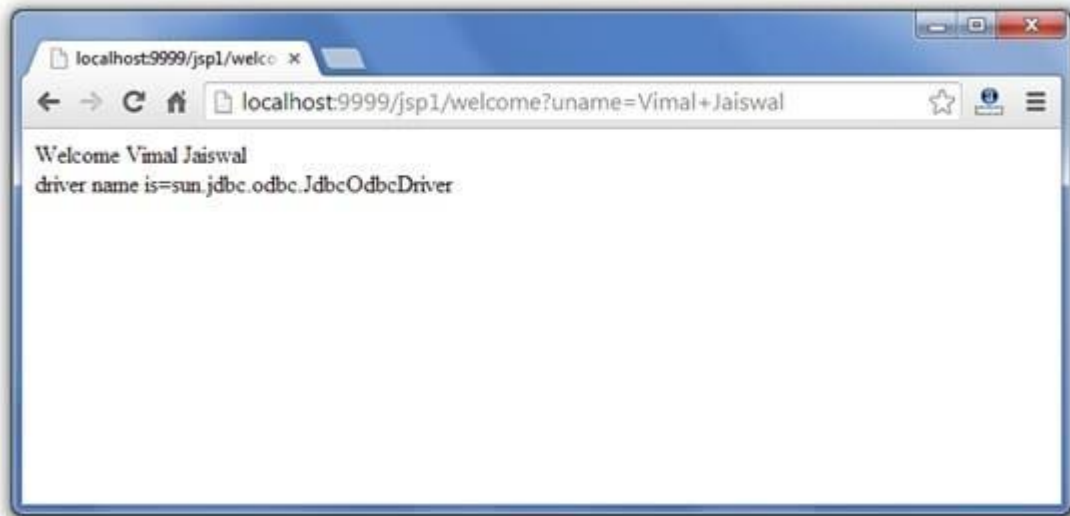
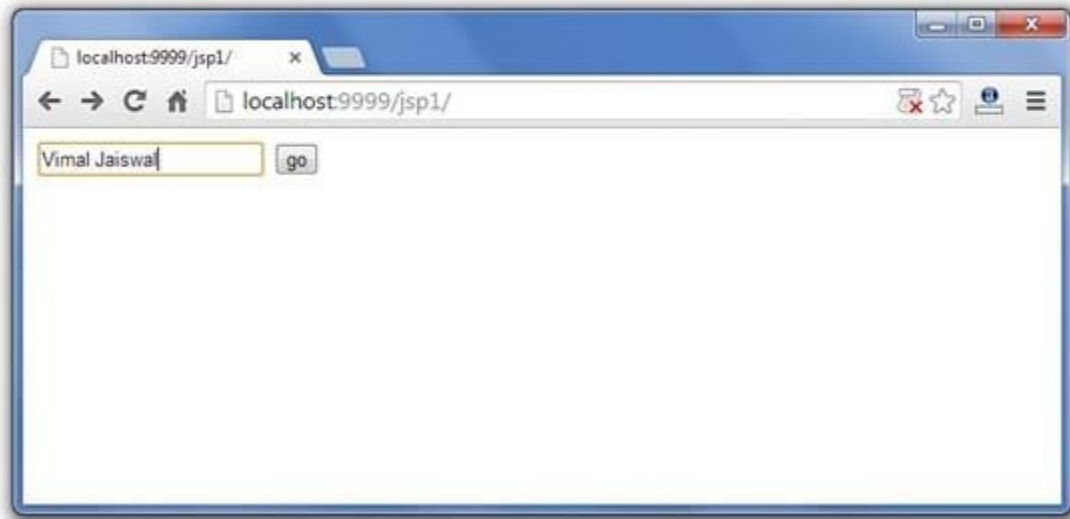
```
out.print("Welcome "+request.getParameter("uname"));
```

```
String driver=application.getInitParameter("dname");
```

```
out.print("driver name is="+driver);
```

```
%>
```

Output



## 6) session implicit object

In JSP, session is an implicit object of type HttpSession.

The Java developer can use this object to set, get or remove attribute or to get session information.

Example of session implicit object

## **index.html**

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

## **welcome.jsp**

```
<html>
<body>
<%

String name=request.getParameter("uname");
out.print("Welcome "+name);

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

## **second.jsp**

```
<html>
<body>
<%
```

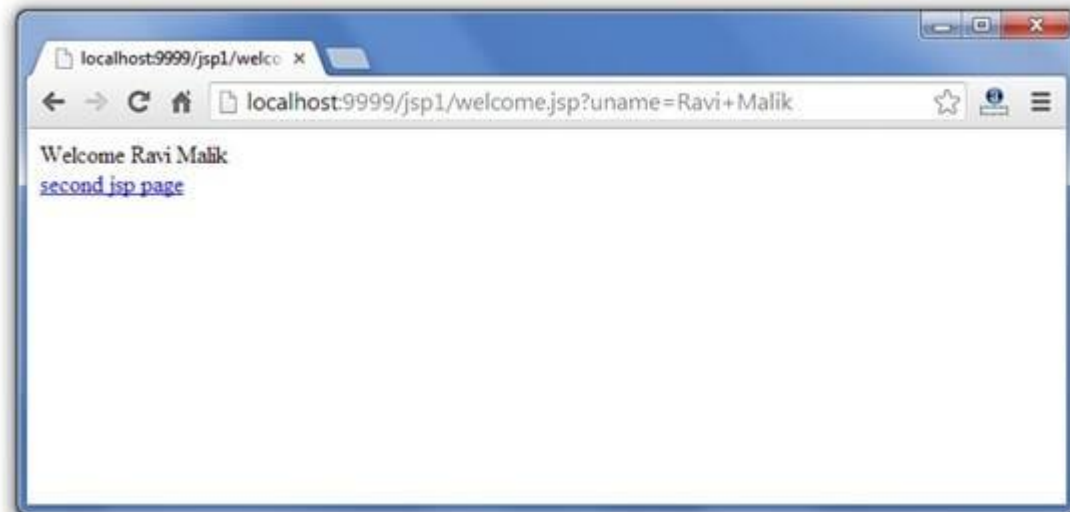
```
String name=(String)session.getAttribute("user");  
out.print("Hello "+name);
```

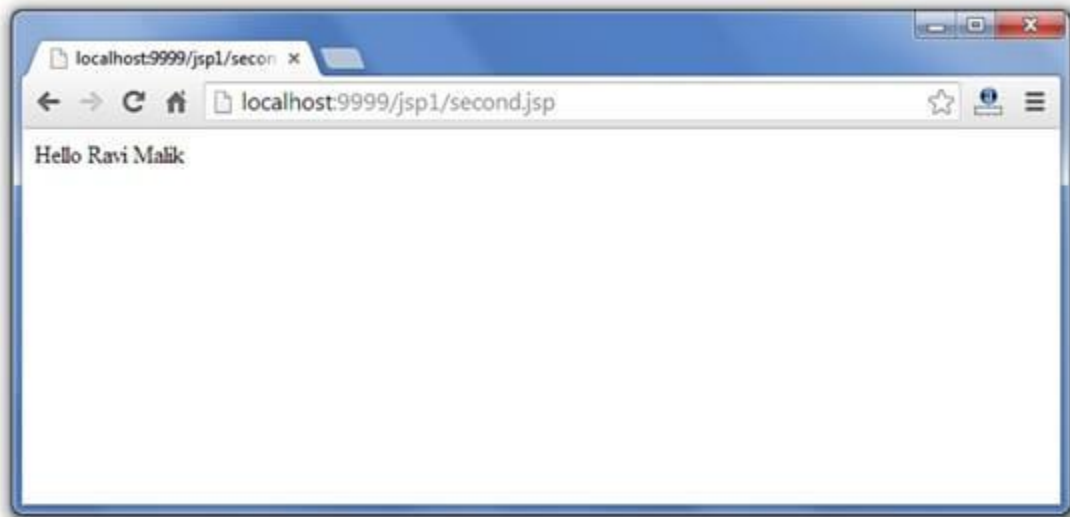
```
%>
```

```
</body>
```

```
</html>
```

Output





### 7) pageContext implicit object

In JSP, pageContext is an implicit object of type PageContext class. The pageContext object can be used to set, get, and remove attribute from one of the following scopes:

- page
- request
- session
- application

In JSP, page scope is the default scope.

Example of pageContext implicit object

#### index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
```

```
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

### **welcome.jsp**

```
<html>
<body>
<%
```

```
String name=request.getParameter("uname");
out.print("Welcome "+name);
```

```
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
```

```
<a href="second.jsp">second jsp page</a>
```

```
%>
</body>
</html>
```

### **second.jsp**

```
<html>
<body>
<%
```

```
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
```

```
%>
```



</body>

</html>

Output

