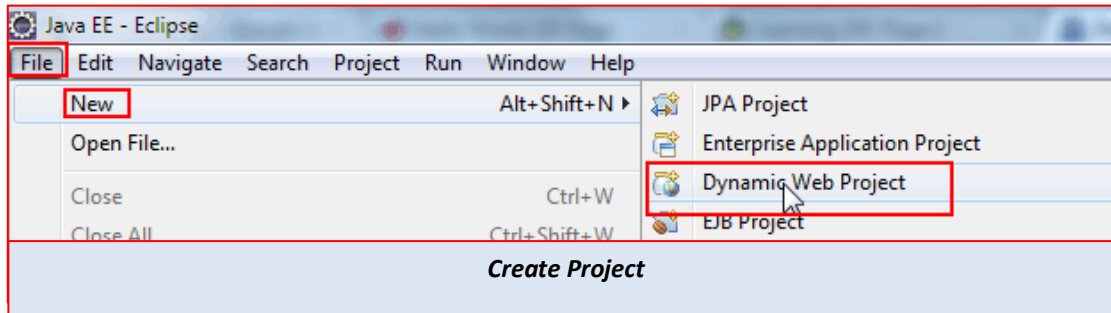


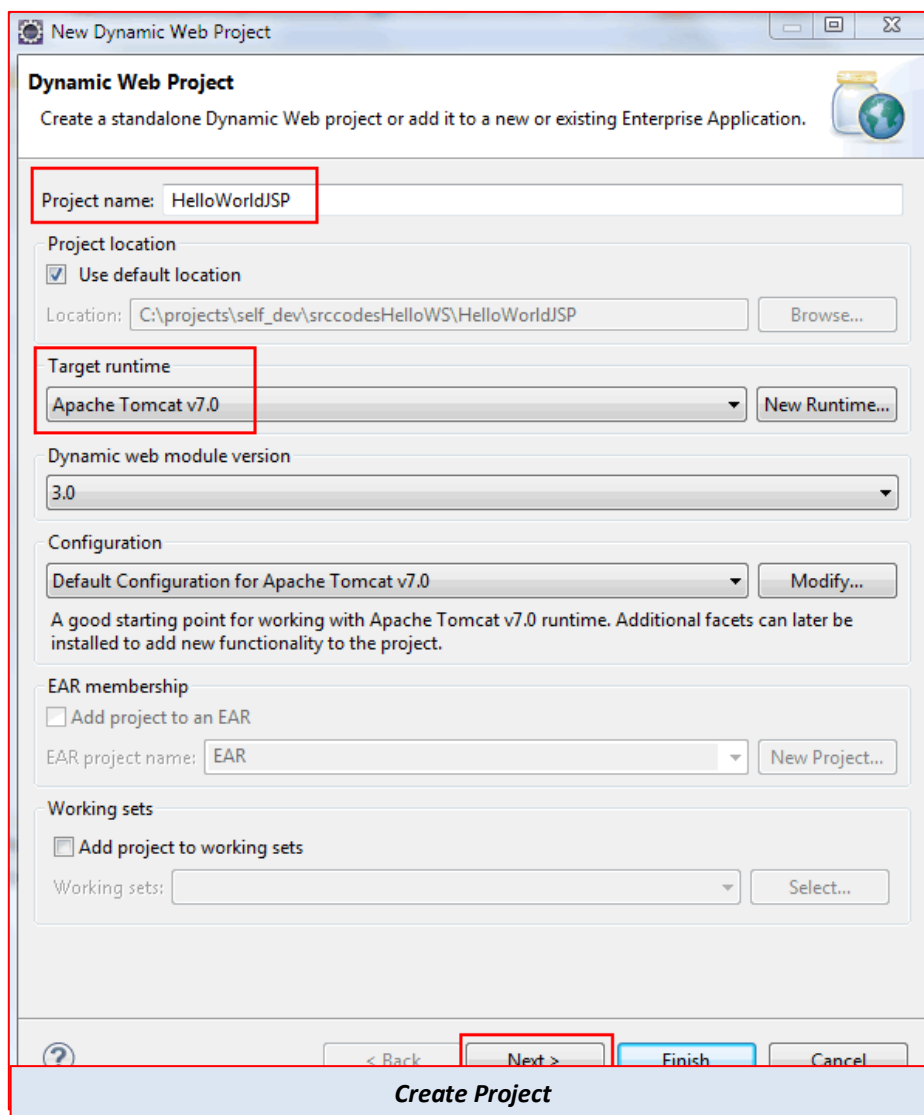
## Lesson 3 Practical A – JSP

### Objectives: Understand the concept of JSP

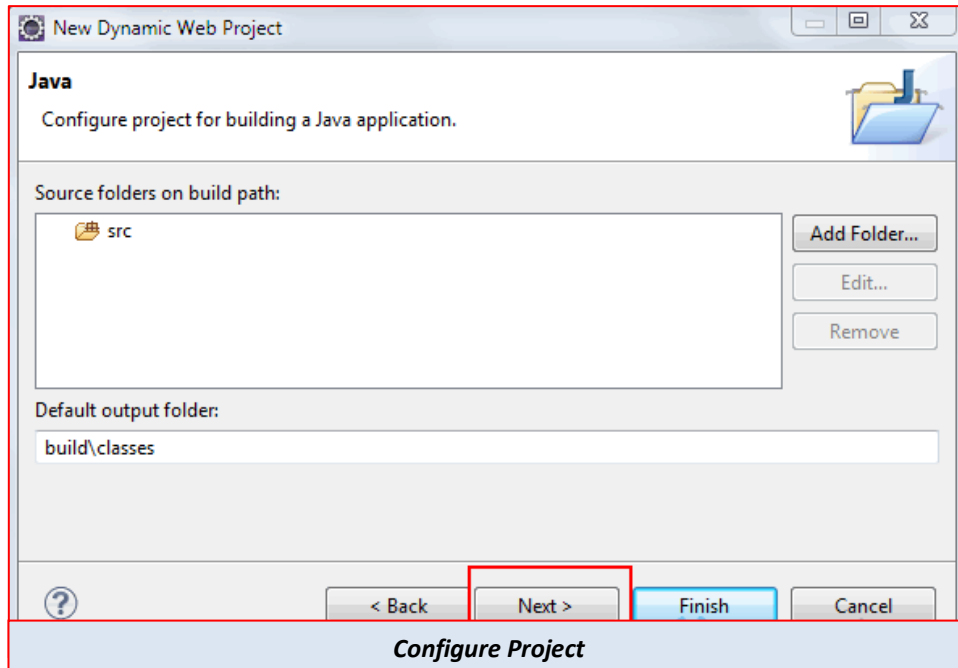
1. Run Eclipse IDE
2. Create Dynamic Web Project
  - Select from the menu File --> New --> Dynamic Web Project.



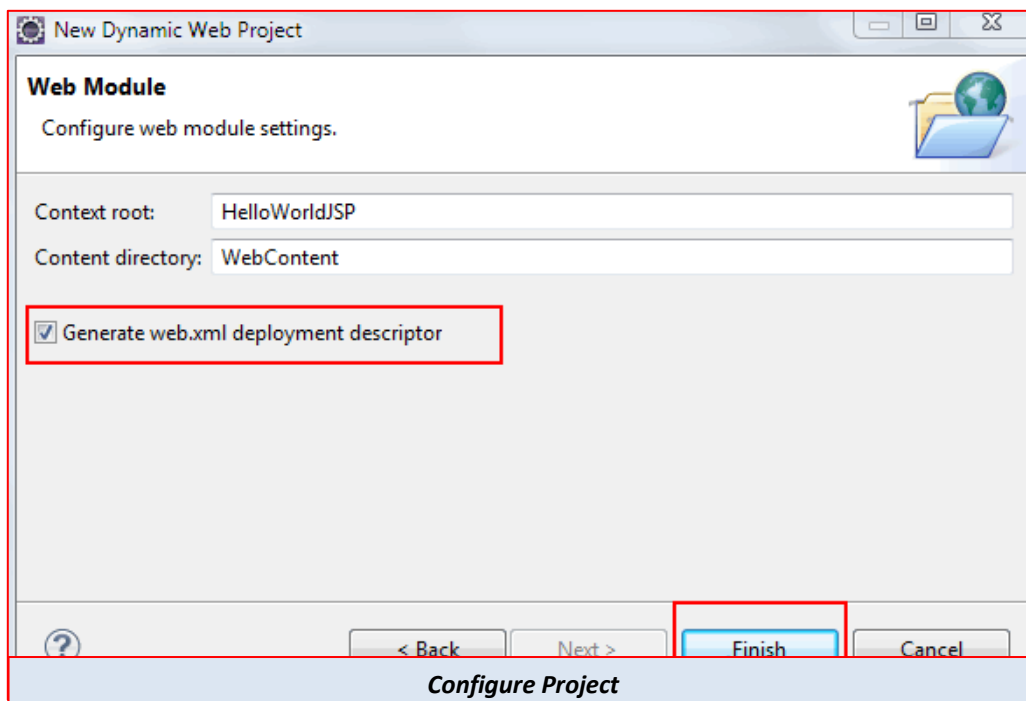
3. Enter "HelloWorldJSP" as the project name. Keep rest of the settings as it is as shown in the following screenshot.



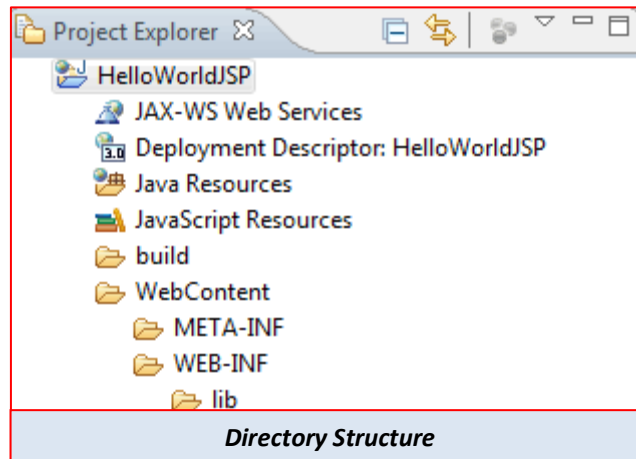
- Click "Next" button.



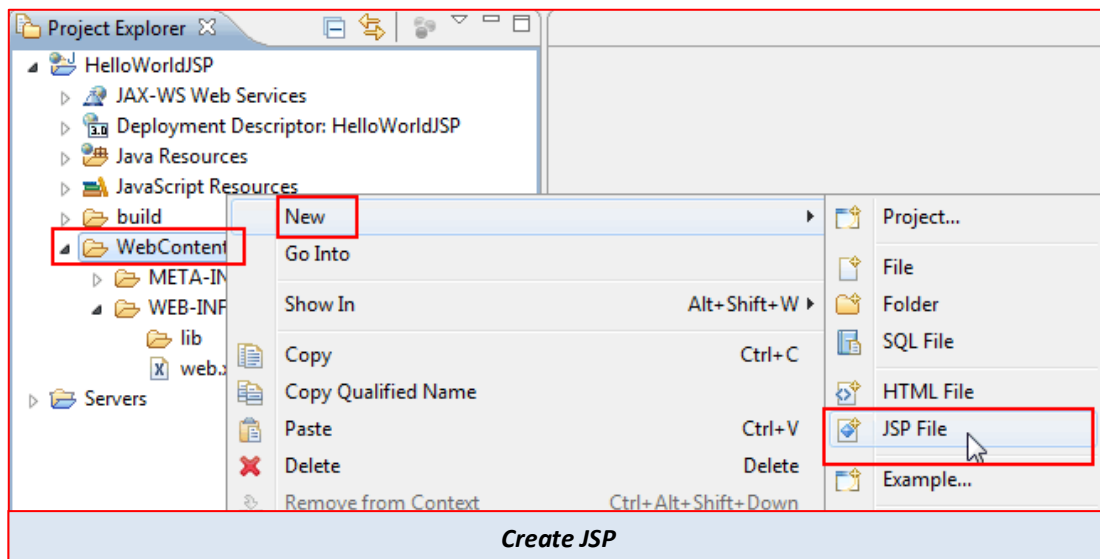
- Click "Next" button.



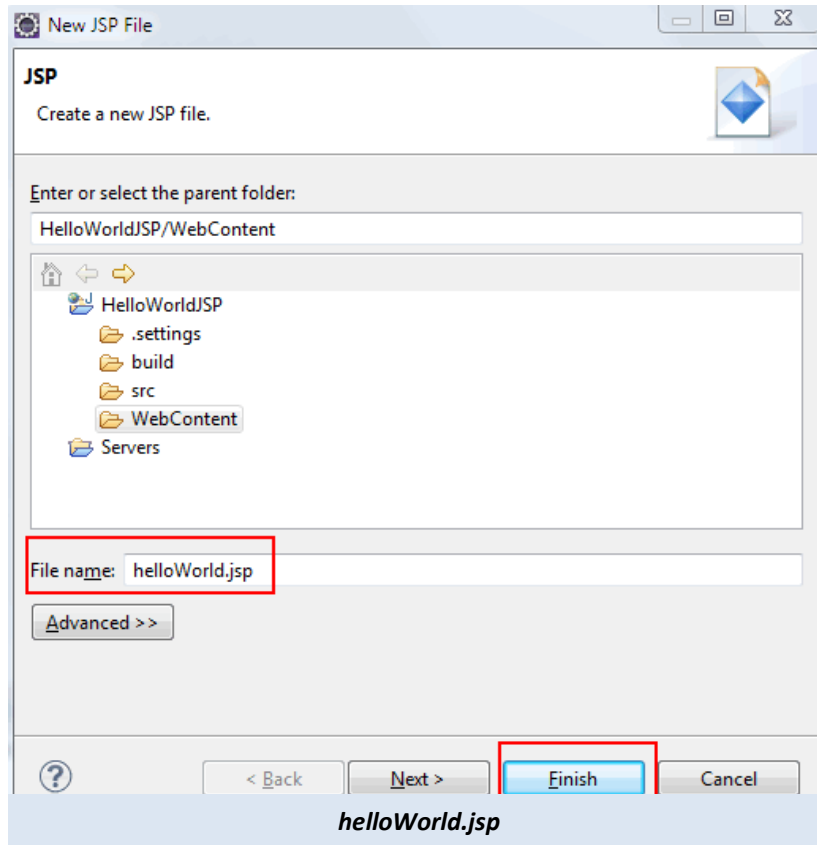
- Check 'Generate web.xml deployment descriptor' checkbox and click "Finish" button and Eclipse IDE will generate the web project automatically as shown below



7. Create Jsp page
8. Right click on 'WebContent' folder and select from context menu New → Jsp File.



9. Write "helloWorld.jsp" in the 'File Name' field and Click "Finish" button.



10. Eclipse will generate a jsp page and open the same in the JSP editor as shown below

File: helloWorld.jsp

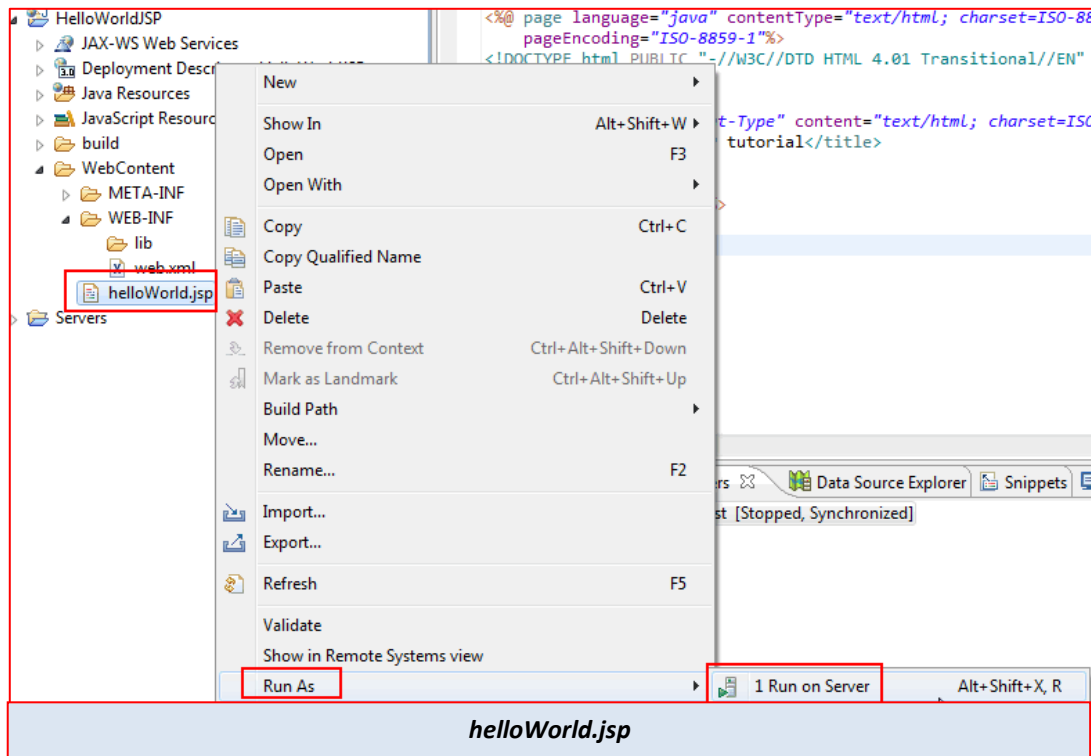
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert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

11. Write JSP Code
12. Edit the generated 'helloWorld.jsp' as per the following code.

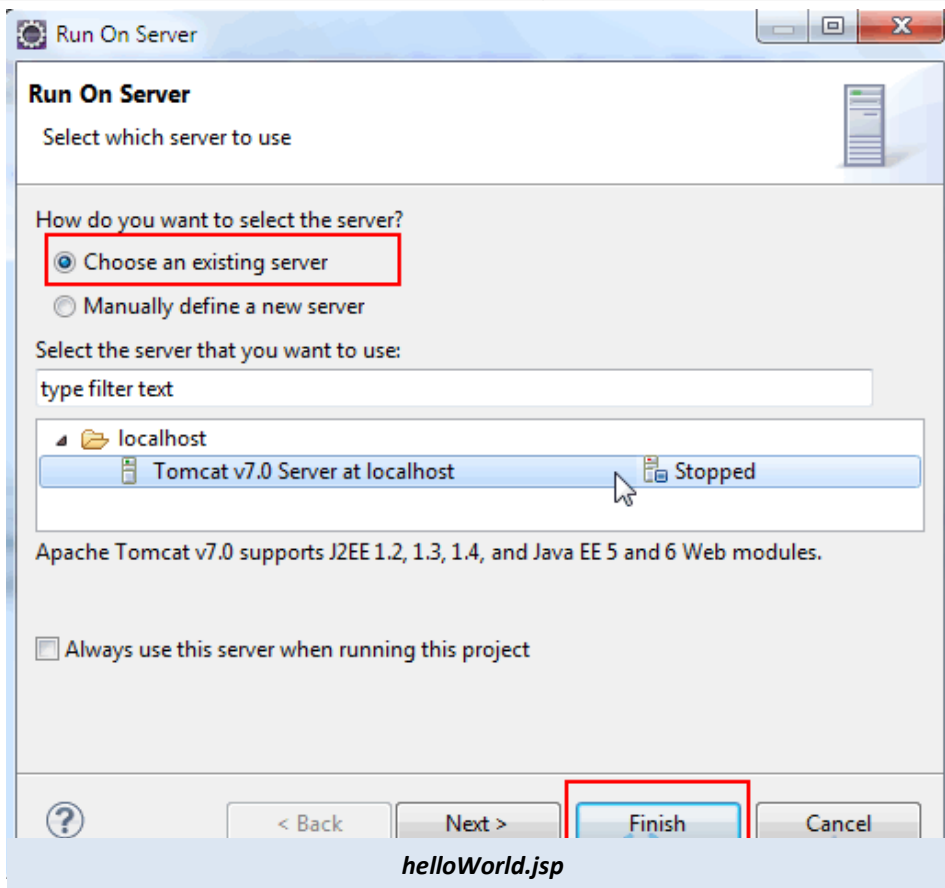
```

<%@ 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>Hello World - JSP tutorial</title>
</head>
<body>
    <%= "Hello World!" %>
</body>
</html>
  
```

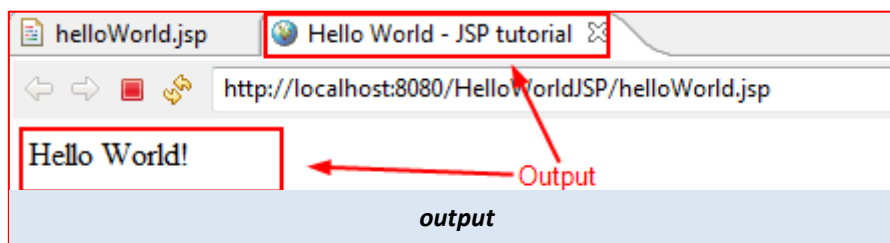
13. Run Your Code
14. Right click on 'helloWorld.jsp' and select from context menu 'Run As' → 'Run on Server'.



15. Select the existing tomcat server. If not available then manually define a new web server.



16. Click "Finish" button. HelloWorldJSP web application will be deployed in the tomcat web server.  
17. Eclipse will launch a browser and your server side jsp code should now print a 'Hello World!' message in the browser.



## 18. More examples ...

(Optional)

## 19. JSP to generate dynamic HTML (Expression)

what makes JSP useful is the ability to embed Java. Java expressions enclosed within the character sequences **<%= and %>** , are evaluated at run time.

Try out this example

*Each time the page is reloaded, the code will return the current time.*

```
<HTML>
<BODY>
Current Date and Time : <%= new java.util.Date() %>
</BODY>
</HTML>
```

*Try out other properties*

System.getProperty for various system properties such as java.version, java.home, os.name, user.name, user.home, user.dir etc.

## 20. Embedding Java Code (Scriptlets)

Scriptlets allows programmers to write blocks of Java code inside a JSP. You do this by placing your Java code between **<% and %>** characters. The scriptlet contains Java code that is executed every time the JSP is invoked.

*Try this code (from previous section) by adding in a scriptlet.*

```
<HTML><BODY>

<%
    System.out.println( "Getting the system date and time" );
    java.util.Date date = new java.util.Date();
%>

Current Date and Time : <%= date %>

</BODY></HTML>
```

"System.out.println" display messages on the server log – a very much convenient way to do simple debugging if you really need to.

A scriptlet does not generate HTML. The following example demonstrate how a scriptlet can generate HTML output by using the “out” variable (servlet example).

*Try this code :*

```
<HTML><BODY>

<%
    System.out.println( "Getting the system date and time" );
    java.util.Date date = new java.util.Date();
%>

Current Date and Time :

<%
    out.println( String.valueOf( date ));
%>

</BODY></HTML>
```

Another very useful pre-defined variable is "request". It is of type ***javax.servlet.http.HttpServletRequest***

The code below demonstrate a sample JSP "request" to return the IP address of the remote host connecting to the server.

*Try this code (Request) :*

```
<HTML><BODY>

<%
    System.out.println( "Getting the system date and time" );
    java.util.Date date = new java.util.Date();
%>

Current Date and Time :

<%
    out.println( date );
    out.println("<br>");
    out.println( "Remote host IP address: " );
    out.println( request.getRemoteHost());
%>

</BODY></HTML>
```



## 21. Scriptlets with HTML

Write a scriptlets to display a HTML table.

*Try this code :*

```
HTML<BODY>

<%
    System.out.println( "Getting the system date and time" );
    java.util.Date date = new java.util.Date();
%>

<TABLE BORDER=2>

    <%
        for ( int i = 0; i < n; i++ ) {
    %>
        <TR>
            <TD>Number</TD>
            <TD> <%= i+1 %> </TD>
        </TR>
    <%
        }
    %>

</TABLE>

</BODY></HTML>
```

## 22. JSP directives (Action Elements)

JSP directives allows programmers to use "import" or "include" statements in JSPs,.

*Try the example :*

```
<%@ page import="java.util.* , java.text.*" %>    (A JSP "directive" starts with <%@ characters.)
<%@ include file="hello.jsp" %>                    (Include an external jsp file)
<HTML><BODY>

<%
    System.out.println( "Evaluating date now" );
    Date date = new Date();
%>

Current Date and Time :<%= date %>

</BODY></HTML>
```

## 23. Scriptlets

Scriptlets allows programmers to write blocks of Java code inside a JSP. You do this by placing your Java code between `<%` and `%>` characters. The scriptlet contains Java code that is executed every time the JSP is invoked.

## Forms and Implicit objects

Create the index.html file and form objects. There are 2 textbox input in the index.html page.

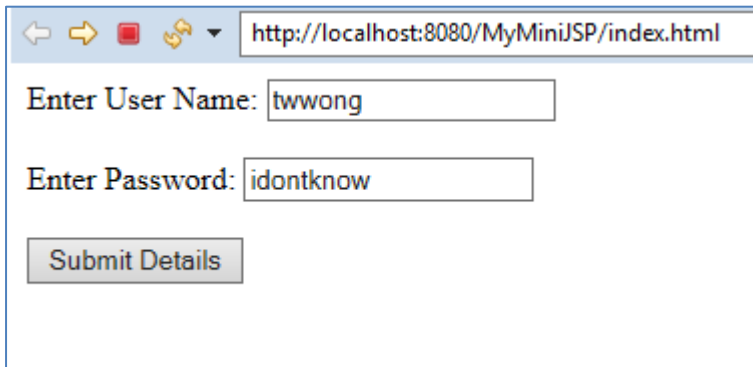
```
<html>
<head>
<title>Enter UserName and Password</title>
</head>
<body>
<form action="userinfo.jsp">
Enter User Name: <input type="text" name="uname" /> <br><br>
Enter Password: <input type="text" name="pass" /> <br><br>
<input type="submit" value="Submit Details"/>
</form>
</body>
</html>
```

Create userinfo.jsp file to handle the form objects

```
<%@ page import = " java.util.* " %>
<html>
<body>
<%
String username=request.getParameter("uname");
String password=request.getParameter("pass");
out.print("Name: "+username+" Password: "+password);
%>
</body>
</html>
```

Run the example in your Eclipse IDE.

Output :

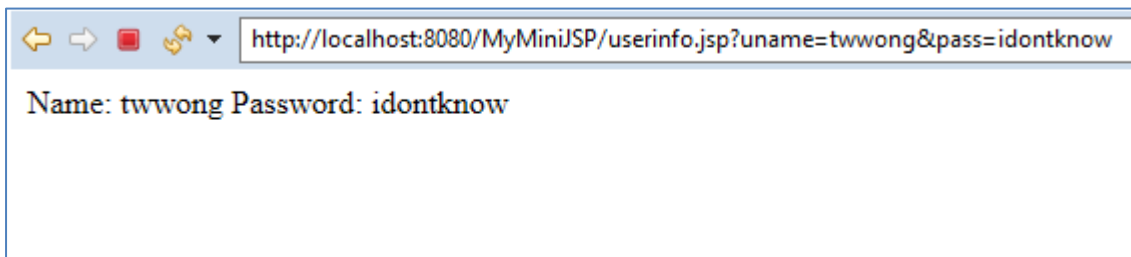


http://localhost:8080/MyMiniJSP/index.html

Enter User Name: tw Wong

Enter Password: idontknow

Submit Details



http://localhost:8080/MyMiniJSP/userinfo.jsp?uname=tw Wong&pass=idontknow

Name: tw Wong Password: idontknow

## Session Implicit object handling

The below html page would display a text box along with a submit button. The submit action would transfer the control to session.jsp page.

### Index.html

```
<html>
<head>
<title>Welcome Page: Enter your name</title>
</head>
<body>
<form action="session.jsp">
<input type="text" name="inputname">
<input type="submit" value="click here!!"><br/>
</form>
</body>
</html>
```

The session.jsp page displays the name which user has entered in the index page and it stores the the same variable in the session object so that it can be fetched on any page until the session becomes inactive.

### session.jsp

```
<html>
<head>
<title>Passing the input value to a session variable</title>
</head>
<body>
<%
String uname=request.getParameter("inputname");
out.print("Welcome "+ uname);
session.setAttribute("sessname",uname);
%>
<a href="output.jsp">Check Output Page Here </a>
</body>
</html>
```

In this page we are fetching the variable's value from session object and displaying it.

### Output.jsp

```
<html>
<head>
<title>Output page: Fetching the value from session</title>
</head>
<body>
<%
String name=(String)session.getAttribute("sessname");
out.print("Hello User: You have entered the name: "+name);
%>
</body>
</html>
```

## Output

