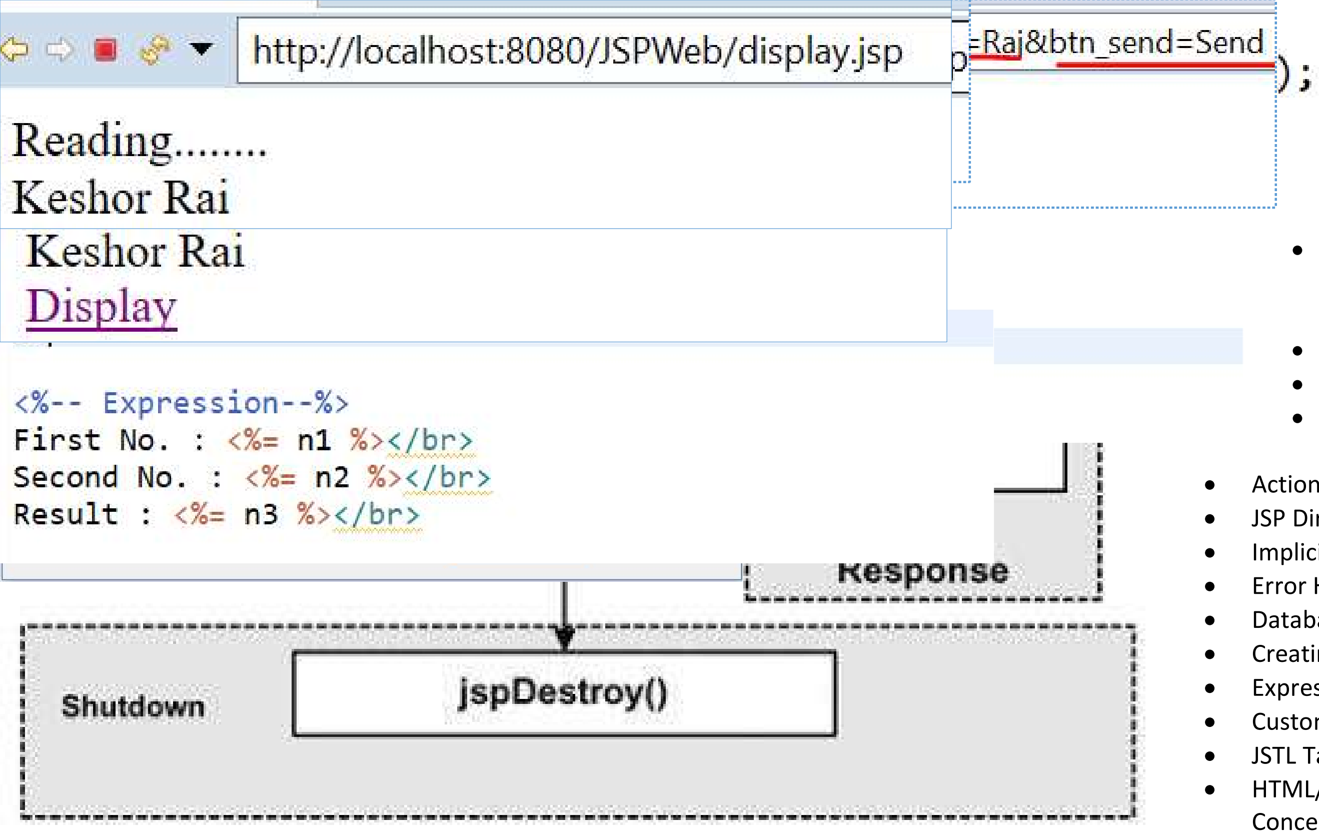


http://localhost:8080/JSPWeb/display.jsp

=Raj&btn_send=Send

Reading........
Keshor Rai
Keshor Rai
Display

```
<%-- Expression--%>
First No. : <%= n1 %></br>
Second No. : <%= n2 %></br>
Result : <%= n3 %></br>
```

JAVA SERVER PAGES (JSP)

INTRODUCTION

Java Server Pages (JSP) is 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. This tutorial will teach you how to use Java Server Pages to develop your web applications in simple and easy steps.

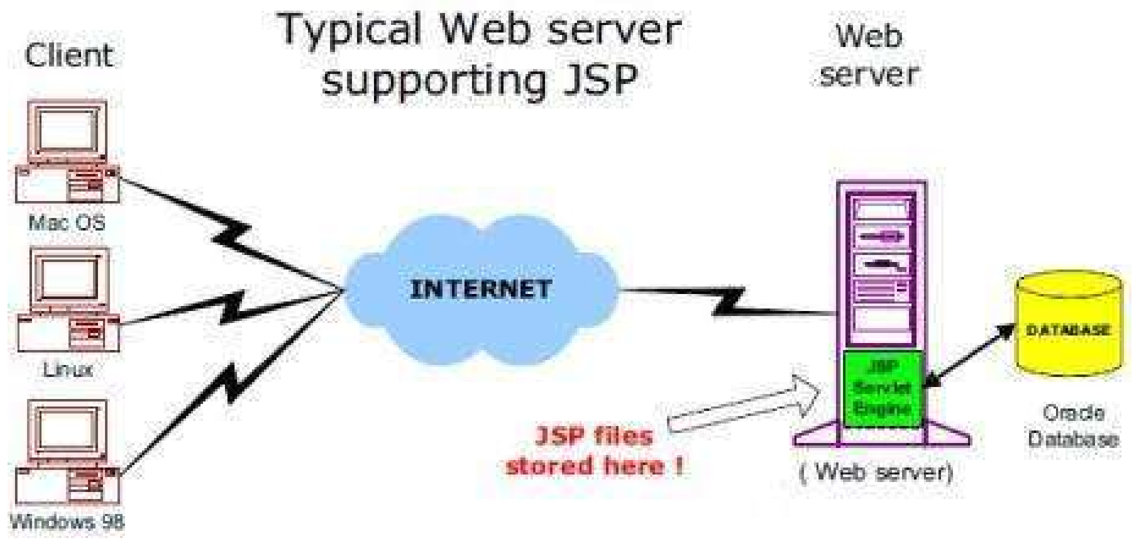
Environment Setting

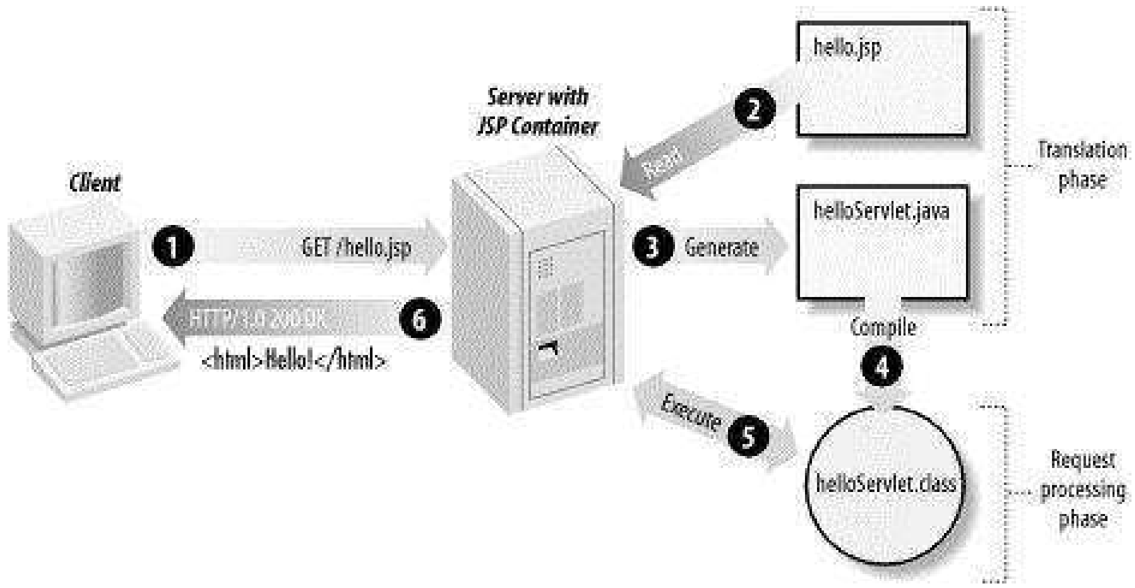
- Obtain and install JDK

- Set **PATH** and **JAVA_HOME, CLASSPATH** environment variables.
  PATH = C:\Program Files\Java\jdk1.8.0_161\bin
  JAVA_HOME = C:\Program Files\Java\jdk1.8.0_161
  CLASSPATH = .; c:\\apache-tomcat-8.5.43\\lib\\servlet-api.jar; c:\\apache-tomcat-8.5.43\\lib\\jsp-api.jar;

- Add Tomcat as Web Server

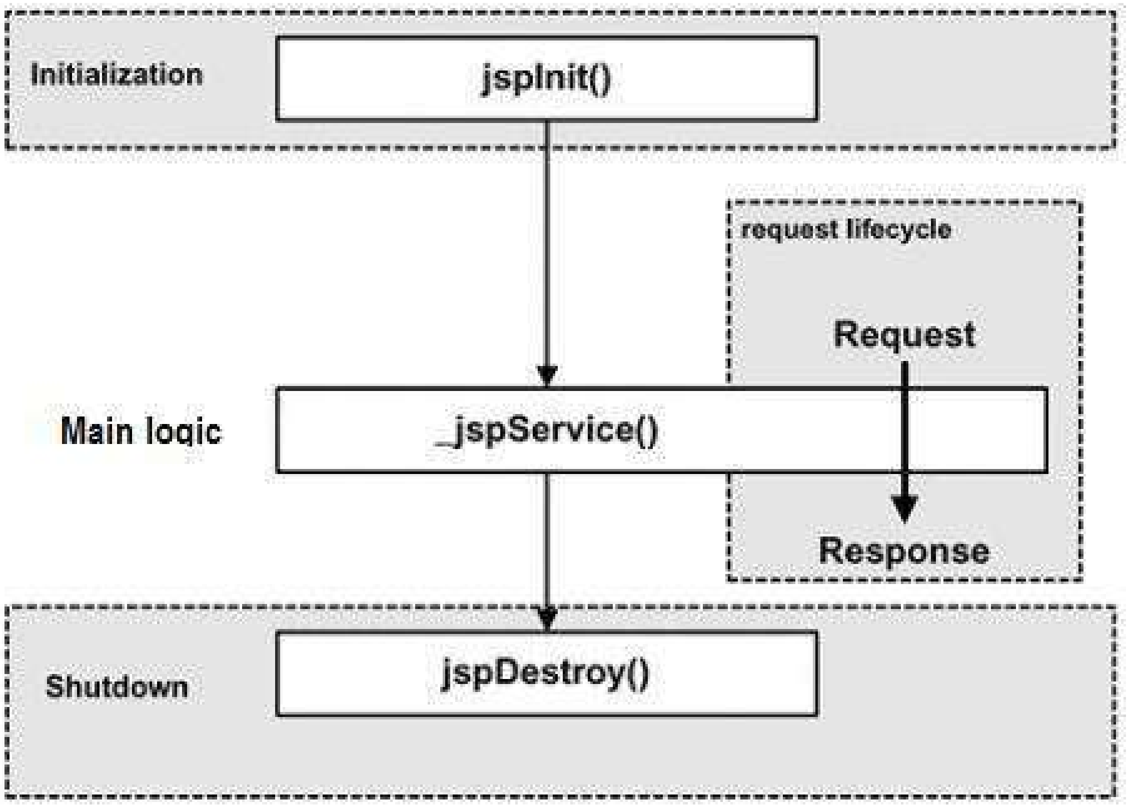
JSP Container





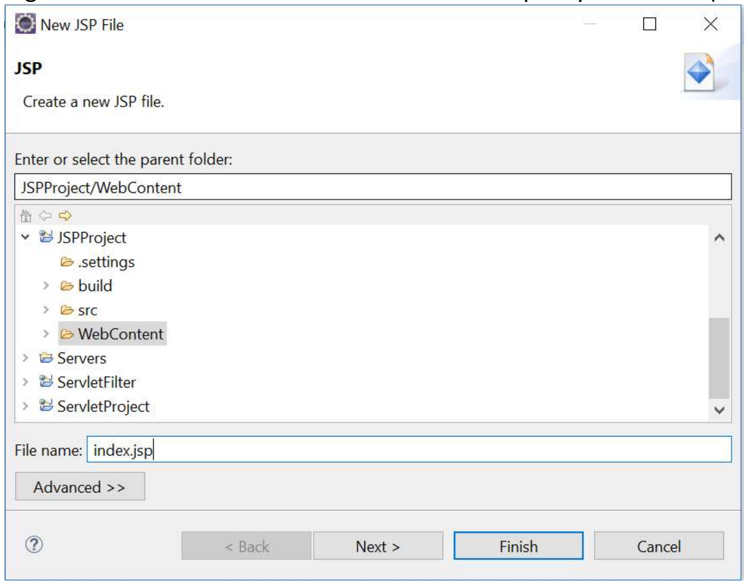
JSP Lifecycle Stages

- Compilation
- Initialization
- Execution
- Cleanup



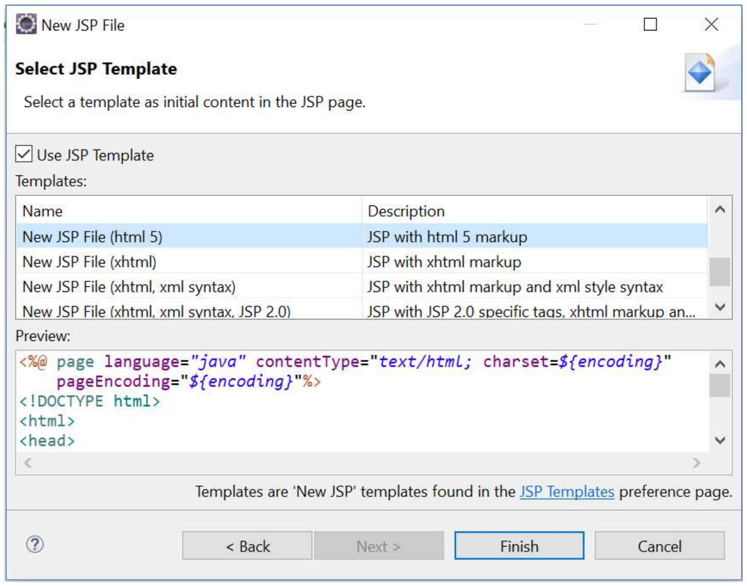
FIRST JSP PAGE
Creating first JSP Page

- Right Click on **Web Content** > **New** > **JSP File >** Specify File Name (**index.jsp**)



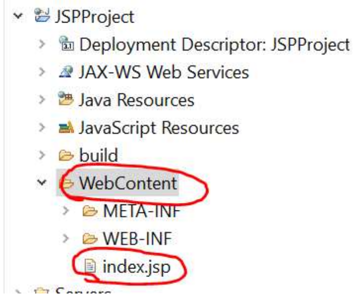
Enter or select the parent folder: JSPProject/WebContent

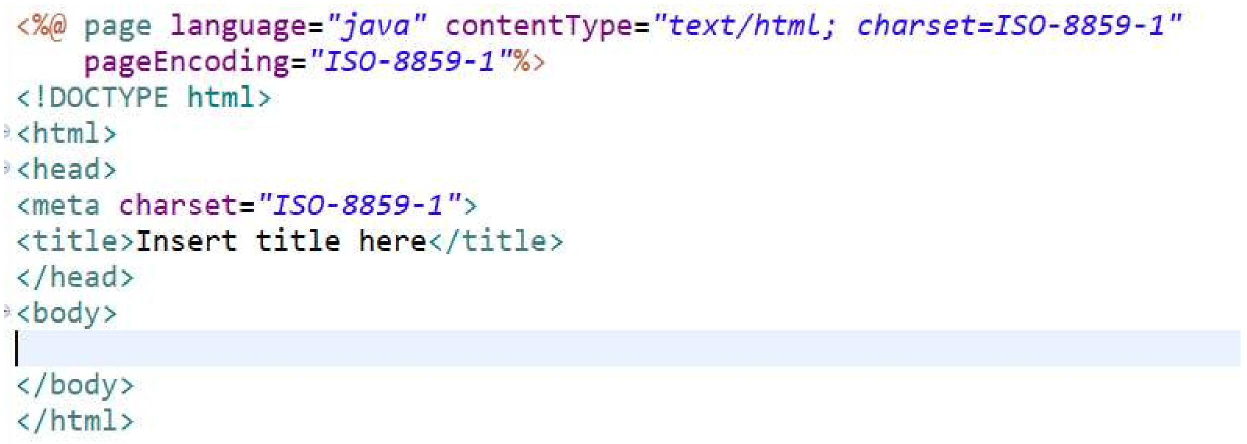
File name: index.jsp


New JSP File
Select JSP Template
Select a template as initial content in the JSP page.
Use JSP Template
Templates:
Preview:
Templates are 'New JSP' templates found in the JSP Templates preference page.
< Back
Next >
Finish
Cancel

Name	Description
New JSP File (html 5)	JSP with html 5 markup
New JSP File (xhtml)	JSP with xhtml markup
New JSP File (xhtml, xml syntax)	JSP with xhtml markup and xml style syntax
New JSP File (xhtml, xml syntax, JSP 2.0)	JSP with JSP 2.0 specific tags, xhtml markup an...

```
<%@ page language="java" contentType="text/html; charset=${encoding}"
    pageEncoding="${encoding}"%>
<!DOCTYPE html>
<html>
<head>
```

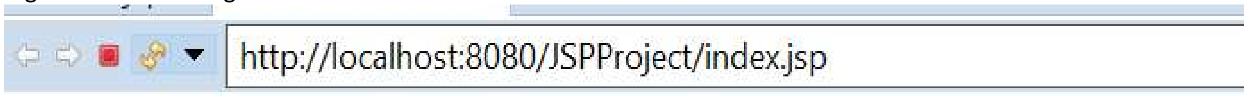




```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

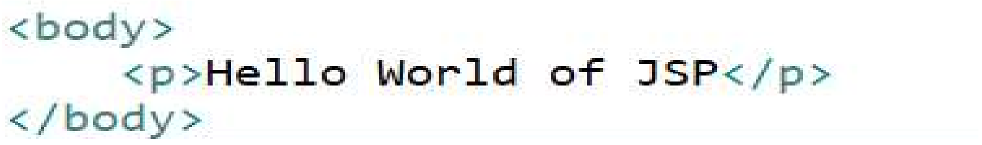
Browsing JSP Page File

- Right Click on JSP Page > **Run as** > **Run on Server**

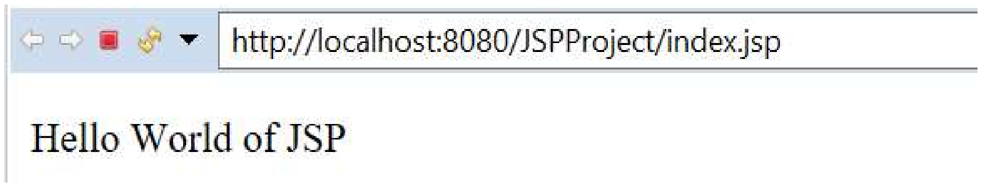


```
http://localhost:8080/JSPProject/index.jsp
```

Adding Content on JSP File



```
<body>
    <p>Hello World of JSP</p>
</body>
```



```
http://localhost:8080/JSPProject/index.jsp

Hello World of JSP
```

BASIC ELEMENTS/ SCRIPTLET ELEMENTS (JSP TAGS)

- JSP Scriptlet
- JSP Declarations
- JSP Expression
- JSP Comments

JSP Scriptlet
A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Syntax

```
<% source code %>
```

OR

```
<jsp:scriptlet>
      source code
</jsp:scriptlet>
```

JSP Declarations
A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Syntax

```
<%! declaration1; [ declaration2; ]+ ... %>

<jsp:declaration>
   declarations
</jsp:declaration>
```

JSP Expression
A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Syntax

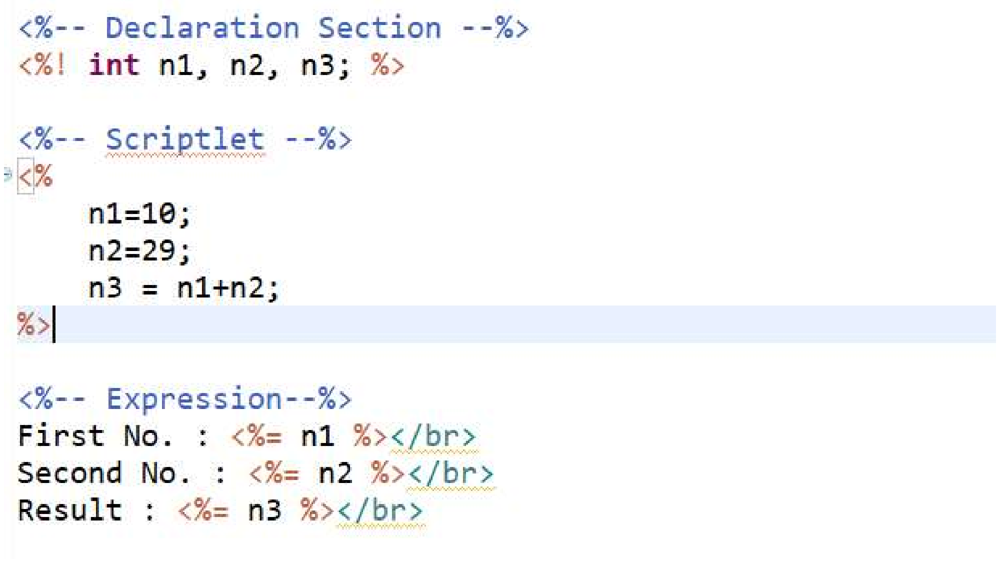
```
<%= expression %>

<jsp:expression>
      expression
</jsp:expression>
```

JSP Comments
JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

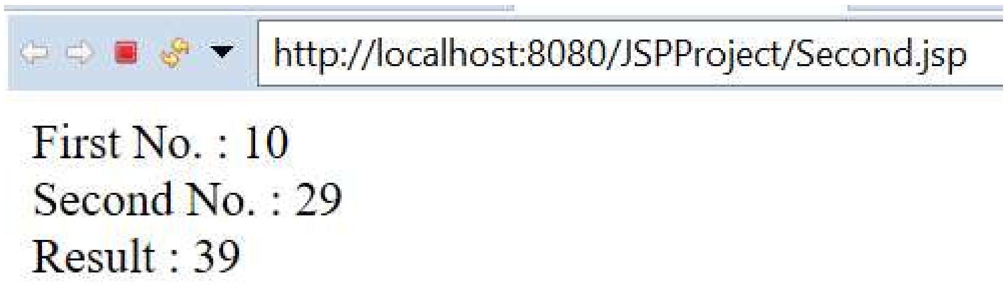
Syntax

```
<%-- This is JSP comment --%>
```

EXAMPLE -1 [Basic Elements]



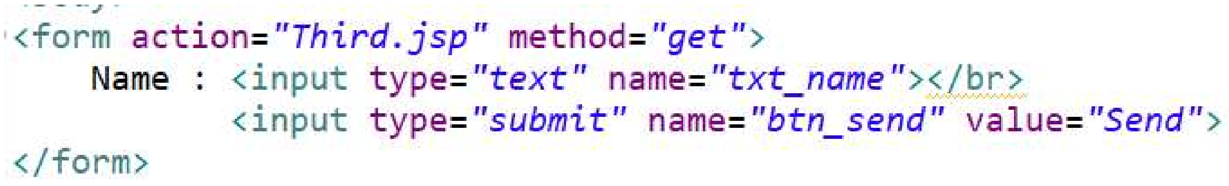
```
<%-- Declaration Section --%>
<%! int n1, n2, n3; %>

<%-- Scriptlet --%>
<%
    n1=10;
    n2=29;
    n3 = n1+n2;
%>

<%-- Expression--%>
First No. : <%= n1 %></br>
Second No. : <%= n2 %></br>
Result : <%= n3 %></br>
```



http://localhost:8080/JSPProject/Second.jsp

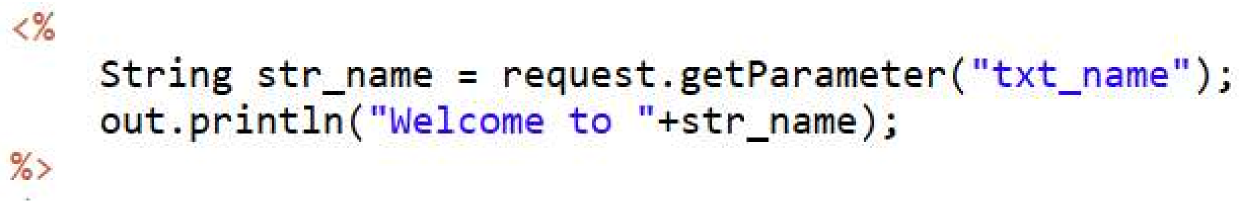
```
First No. : 10
Second No. : 29
Result : 39
```

Example-2
Web Form



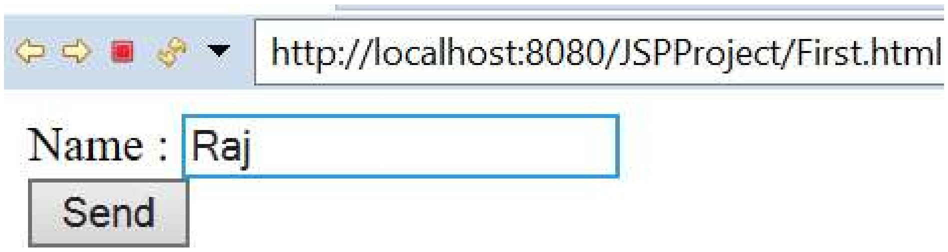
```
<form action="Third.jsp" method="get">
    Name : <input type="text" name="txt_name"></br>
           <input type="submit" name="btn_send" value="Send">
</form>
```

JSP File

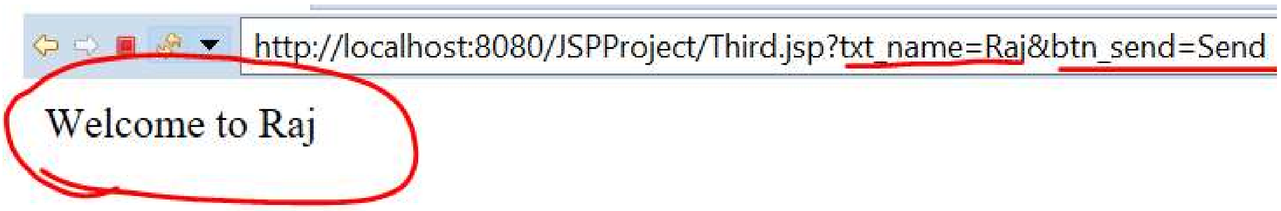


```
<%
    String str_name = request.getParameter("txt_name");
    out.println("Welcome to "+str_name);
%>
```

Output:


http://localhost:8080/JSPProject/First.html

Name : Raj


http://localhost:8080/JSPProject/Third.jsp?txt_name=Raj&btn_send=Send

Welcome to Raj

SERVER OBJECTS/ IMPLICIT JSP OBJECTS

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:

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

out

The out implicit object is an instance of a javax.servlet.jsp.JspWriter object and is used to send content in a response.

- **out.print(dataType dt)**
  Print a data type value

- **out.println(dataType dt)**
  Print a data type value then terminate the line with new line character.

- **out.flush()**
  Flush the stream.

request

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.

Example

webform

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

first.jsp

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

response

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.

form.jsp

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

welcome.jsp

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

config

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.

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

web.xml
```
	<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);
	%>
```

application

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 configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

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

web.xml
```
	<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);
%>
```

session

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.

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>
```

pageContext

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

- page
- request
- session
- application

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>
```

page

page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class.

Example

```
	Object page=this;
	For using this object it must be cast to Servlet type.For example:
	<% (HttpServlet)page.log("message"); %>
	Since, it is of type Object it is less used because you can use this object directly in jsp.For example:
	<% this.log("message"); %>
```

exception

exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive.

error.jsp

```
	<%@ page isErrorPage="true" %>
	<html>
		<body>
			Sorry following exception occured:<%= exception %>
		</body>
	</html>
```

ACTION ELEMENTS

- <jsp:include>
- <jsp:forward>
- <jsp:param>
- <jsp:useBean>
- <jsp:setProperty>
- <jsp:getProperty>
- <jsp:plugin>
- <jsp:body>
- <jsp:element>
- <jsp:text>
- <jsp:attribute>

<jsp:include>
Used to insert a JSP file in another file.

Syntax
```
<jsp:include page="page URL"  flush="Boolean Value" />
```

Example
```
<jsp:include page="images.jsp" flush="false" />
```

<jsp:forward>
Used for redirecting the request. When this action is encountered on a JSP page the control gets transferred to the page mentioned in this action.

Syntax
```
<jsp:forward page="URL of the another JSP OR Servlet page" />
```

Example
```
<jsp:forward page="display.jsp" />
```

<jsp:param>
This action is useful for passing the parameters to Other JSP action tags such as JSP include & JSP forward tag. This way new JSP pages can have access to those parameters using request object itself.

Syntax
```
<jsp: param name="param_name_here" value="value_of_parameter_here" />
```

Example
working.jsp
```
<jsp:forward page="display.jsp">
    <jsp:param name ="user" value="admin" />
    <jsp:param name ="password" value="admin" />
</jsp:forward>
```

display.jsp
```
USER :<%= request.getParameter("user") %>
PASSWORD :<%= request.getParameter("password") %>
```

<jsp:useBean>
Syntax
```
<jsp:useBean id="unique_name_to_bean"  class="package_name.class_name" />
```

Example
<jsp:setProperty>
<jsp:getProperty>

Example
form.jsp
```
<form action="process.jsp" method="post">
    Name:<input type="text" name="name"><br>
    Password:<input type="password" name="password"><br>
    Email:<input type="text" name="email"><br>
    <input type="submit" value="register">
</form>
```

process.jsp
```
<jsp:useBean id="u" class="User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>
Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

User.java
```
public class User {
    private String name, password, email;
    //setters and getters
}
```

<jsp:plugin>, <jsp:params>, <jsp:fallback>
<jsp:plugin> action is used to generate browser-dependent HTML code (OBJECTor EMBED) that displays and executes a Java Plugin software (Java applet or a JavaBean component) in the current JSP page. The **<jsp:params>** and **<jsp:fallback>** actions are optional sub elements.

Example [plugins.jsp]
```
<jsp:plugin
  type="applet" code="pkg1.MyApplet.class" codebase="html">

  <jsp:params>
    <jsp:param name="username" value="Tom" />
  </jsp:params>

  <jsp:fallback>
    <p>Could not load applet!</p>
  </jsp:fallback>
</jsp:plugin>
```

Example [<jsp:attribute>, <jsp:element>, <jsp:body>]
```
<jsp:element name="element1">
    <jsp:attribute name="attribute1">left</jsp:attribute>
    <jsp:body>Text to Display</jsp:body>
</jsp:element>

<p><jsp:text>Sample Text</jsp:text></p>
```

Example [<jsp:text>]
```
<p><jsp:text>Sample Text</jsp:text></p>
```

JSP DIRECTIVES

- Page Directive
- Include Directive
- TagLib Directive

Page Directive

- **import**
  Syntax
  ```
  <%@page import="value"%>
  ```
  Example
  ```
  <%@page import="java.io.*%>
  <%@page import="java.lang.*%>
  <%--Comment: OR Below Statement: Both are Same--%>
  <%@page import="java.io.*, java.lang.*"%>
  ```
- **session**
  Syntax
  ```
  <%@ page session="flag_value"%>
  ```
  Example
  ```
  <%@ page session="true"%>
  ```
  OR
  ```
  <%@ page session="false"%>
  ```

- **isErrorPage**
  Syntax
  ```
  <%@ page isErrorPage="value"%>
  ```
  Example
  ```
  <%@ page isErrorPage="true"%>
  ```

- **errorPage**
  Syntax
  ```
  <%@ page errorPage="value"%>
  ```
  Example
  ```
  <%@ page errorPage="DisplayError.jsp"%>
  ```

- **contentType**
  Syntax
  ```
  <%@ page contentType="value"%>
  ```
  Example
  ```
  <%@ page contentType="text/html"%>
  ```
  OR
  ```
  <%@ page contentType="text/xml"%>
  ```

- **isThreadSafe**
  Syntax
  ```
  <%@ page isThreadSafe="value"%>
  ```
  Example
  ```
  <%@ page isThreadSafe="false"%>
  ```

- **extends**
  Syntax
  ```
  <%@ page extends="value"%>
  ```
  Example
  ```
  <%@ page extends="mypackage.SampleClass"%>
  ```

- **info**
  Syntax
  ```
  <%@ page info="value"%>
  ```
  Example

```
<%@ page info="This code is given by Chaitanya Singh"%>
```

- **language**
  Syntax
  ```
  <%@ page language="value"%>
  ```
  Example
  ```
  <%@ page language="java"%>
  ```

- **autoflush**
  Syntax
  ```
  <%@ page autoFlush="value"%>
  ```
  Example
  ```
  <%@ page autoFlush="true"%>
  ```

- **buffer**
  Syntax
  ```
  <%@ page buffer="value"%>
  ```
  Example
  ```
  <%@ page buffer="none"%>
  <%@ page buffer="5kb"%>
  ```

- **isELIgnored**
  Syntax
  ```
  <%@ page isELIgnored="value"%>
  ```
  Example
  ```
  <%@ page isELIgnored="false"%>
  ```

Include Directive
Syntax
```
<%@include file ="value"%>
```
Example
```
<%@include file="myJSP.jsp"%>
```

TagLib Directive
Syntax
```
<%@taglib uri ="taglibURI" prefix="tag prefix"%>
```
Example
```
<%@ taglib uri="http://www.sample.com/mycustomlib" prefix="demotag" %>
<html>
<body>
<demotag:welcome/>
</body>
</html>
```

EXCEPTION HANDLING
The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

- By errorPage and isErrorPage attributes of page directive
- By <error-page> element in web.xml file

Example-1
index.jsp
```
<form action="process.jsp">
        No1:<input type="text" name="n1" /><br/><br/>
        No1:<input type="text" name="n2" /><br/><br/>
        <input type="submit" value="divide"/>
```

```
    </form>
```

process.jsp

```
    <%@ page errorPage="error.jsp" %>
    <%
        String num1=request.getParameter("n1");
        String num2=request.getParameter("n2");

        int a=Integer.parseInt(num1);
        int b=Integer.parseInt(num2);
        int c=a/b;
        out.print("division of numbers is: "+c);
    %>
```

error.jsp

```
    <%@ page isErrorPage="true" %>
    <h3>Sorry an exception occured!</h3>
    Exception is: <%= exception %>
```

Example-2
index.jsp

```
    <form action="process.jsp">
        No1:<input type="text" name="n1" /><br/><br/>
        No1:<input type="text" name="n2" /><br/><br/>
        <input type="submit" value="divide"/>
    </form>
```

process.jsp

```
    <%@ page errorPage="error.jsp" %>
    <%
        String num1=request.getParameter("n1");
        String num2=request.getParameter("n2");
        int a=Integer.parseInt(num1);
        int b=Integer.parseInt(num2);
        int c=a/b;
        out.print("division of numbers is: "+c);
    %>
```

error.jsp

```
    <%@ page isErrorPage="true" %>
    <h3>Sorry an exception occured!</h3>
    Exception is: <%= exception %>
```

web.xml [Error page for general erros]

```
    <web-app>
        <error-page>
            <exception-type>java.lang.Exception</exception-type>
            <location>/error.jsp</location>
        </error-page>
    </web-app>
```

web.xml [Error page for specific erro]

```
    <web-app>
        <error-page>
            <error-code>500</error-code>
            <location>/error.jsp</location>
        </error-page>
```

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

DATABASE CONNECTIVITY
The database is used for storing various types of data which are huge and has storing capacity in gigabytes. JSP can connect with such databases to create and manage the records.

Example-1 [Connect Database] [ConnectDB.jsp]

```
<%@page import="java.sql.Connection"%>
<%@page import="java.sql.DriverManager"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Connect Database</title>
</head>
<body>
	<%
		String driverName = "com.mysql.jdbc.Driver";
		String connectionUrl = "jdbc:mysql://localhost:3306/test";
		String userId = "admin";
		String password = "admin@123";
		Connection conn = null;
		String str_result="";
		try {
			Class.forName(driverName);
			conn = DriverManager.getConnection(connectionUrl, userId, password);

			conn.close();
			str_result="Connect database server sucessfully";
		}
		catch (ClassNotFoundException e){
			e.printStackTrace();
			str_result="Error : "+e.getMessage();
		}
	%>
	<div><%= str_result %></div>
</body>
</html>
```

Example-2 [Insert Record]

```
String driverName = "com.mysql.jdbc.Driver";
String connectionUrl = "jdbc:mysql://localhost:3306/test";
String userId = "root";
String password = "";
Connection conn = null;
PreparedStatement pstat = null;

String str_sql = "insert into tbl_person values(?, ?,?)";
String str_result="";

try {
	Class.forName(driverName);
	conn = DriverManager.getConnection(connectionUrl, userId, password);
	pstat = conn.prepareStatement(str_sql);
	pstat.setInt(1,13);
	pstat.setString(2,"Krishna");
	pstat.setString(3,"Ktm");
	pstat.executeUpdate();
```

```
                pstat.close();
                conn.close();
                str_result="Insert record sucessfully";
        }
        catch (ClassNotFoundException e){
                //e.printStackTrace();
                str_result="Error : "+e.getMessage();
        }
```

Example-3 [Update Record]

```
        String driverName = "com.mysql.jdbc.Driver";
        String connectionUrl = "jdbc:mysql://localhost:3306/test";
        String userId = "root";
        String password = "";
        Connection conn = null;
        PreparedStatement pstat = null;

        String str_sql = "update tbl_person set full_name =?, email=? where id =?";
        String str_result="";
        try {
                Class.forName(driverName);
                conn = DriverManager.getConnection(connectionUrl, userId, password);
                pstat = conn.prepareStatement(str_sql);
                pstat.setString(1,"Krishna Aryal");
                pstat.setString(2,"krishna@gmail.com");
                pstat.setInt(3,12);
                pstat.executeUpdate();
                pstat.close();
                conn.close();
                str_result="Update record sucessfully";
        }
        catch (ClassNotFoundException e){
                //e.printStackTrace();
                str_result="Error : "+e.getMessage();
        }
```

Example-4 [Delete Record]

```
        String driverName = "com.mysql.jdbc.Driver";
        String connectionUrl = "jdbc:mysql://localhost:3306/test";
        String userId = "root";
        String password = "";
        Connection conn = null;
        PreparedStatement pstat = null;

        String str_sql = "delete from tbl_person where id =?";
        String str_result="";
        try {
                Class.forName(driverName);
                conn = DriverManager.getConnection(connectionUrl, userId, password);
                pstat = conn.prepareStatement(str_sql);
                pstat.setInt(1,12);
                pstat.executeUpdate();
                pstat.close();
                conn.close();
                str_result="Delete record sucessfully";
        }
        catch (ClassNotFoundException e){
                //e.printStackTrace();
                str_result="Error : "+e.getMessage();
        }
```

Example-5 [Display Records]

```
        String driverName = "com.mysql.jdbc.Driver";
        String connectionUrl = "jdbc:mysql://localhost:3306/test";
        String userId = "root";
        String password = "";

        Connection conn = null;
        PreparedStatement pstat = null;
        ResultSet rs=null;

        String str_sql = "select * from tbl_person";
        String str_result="";
        try {
                Class.forName(driverName);
                conn = DriverManager.getConnection(connectionUrl, userId, password);
                pstat = conn.prepareStatement(str_sql);
                rs = pstat.executeQuery();
                %>
                <table border="1">
                        <tr><td>ID</td><td>NAME</td><td>EMAIL</td></tr>
                <%
                while(rs.next()){
                        out.println("<tr><td>"+rs.getInt(1)+"</td><td>"+ rs.getString(2)+
                                                "</td><td>"+ rs.getString(3) +"</td></tr>");
                }
                %>
                </table>
                <%
                pstat.close();
                conn.close();
                str_result="Display records sucessfully";
        }
        catch (ClassNotFoundException e){
                //e.printStackTrace();
                str_result="Error : "+e.getMessage();
        }
```

CREATING JAVA BEANS
Example-1 [Creating and Use BeanClass]
Calculator.java

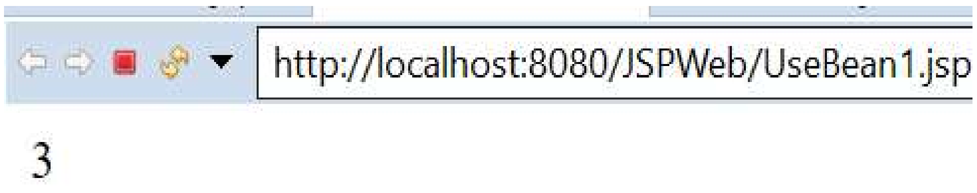
```
        package pkg1;
        public class Calculator {
                public int calc_sum(int n1, int n2) {
                        return n1+n2;
                }
        }
```

JSP File

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

```
<jsp:useBean id="obj" class="pkg1.Calculator"></jsp:useBean>
<title>Insert title here</title>
</head>
<body>
        <%= obj.calc_sum(1, 2) %>
</body>
</html>
```

Output:



http://localhost:8080/JSPWeb/UseBean1.jsp

3

Example-2 [Creating, Set/Get Property on JavaBean]

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<jsp:useBean id="person" class="pkg1.PersonBean"></jsp:useBean>
<jsp:setProperty property="*" name="person"/>
</head>
<body>
        <jsp:setProperty name="person" property="fullName" value="Krishna Aryal" />
        Person Details : <br>
        <jsp:getProperty name="person" property="fullName" />
</body>
</html>
```

Example-3 [Creating, Set/Get Property on JavaBean]
Web Form

```
<form action="process.jsp" method="post">
        Name:<input type="text" name="name"><br>
        <input type="submit" value="register">
</form>
```

Setter

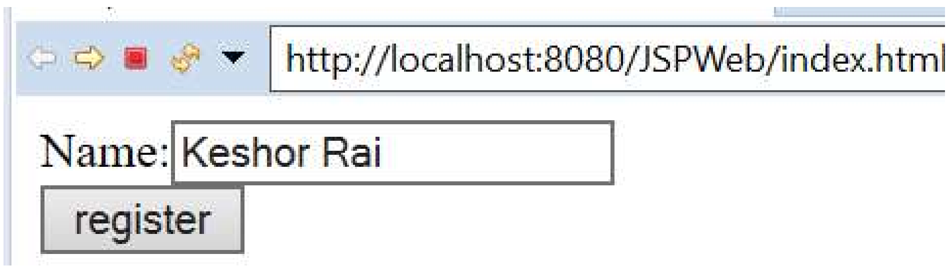
```
<jsp:useBean id="person" class="pkg1.PersonBean" scope="session"></jsp:useBean>
<jsp:setProperty property="*" name="person"/>
<%
        String name=request.getParameter("name");
%>
Setting.......<br>
<jsp:setProperty property="fullName" name="person" value="<%=name %>"/>
Reading........<br>
<jsp:getProperty property="fullName" name="person"/><br>
<a href="display.jsp">Display</a>
```

Getter

```
<jsp:useBean id="person" class="pkg1.PersonBean" scope="session"></jsp:useBean>
<jsp:setProperty property="*" name="person"/>
```

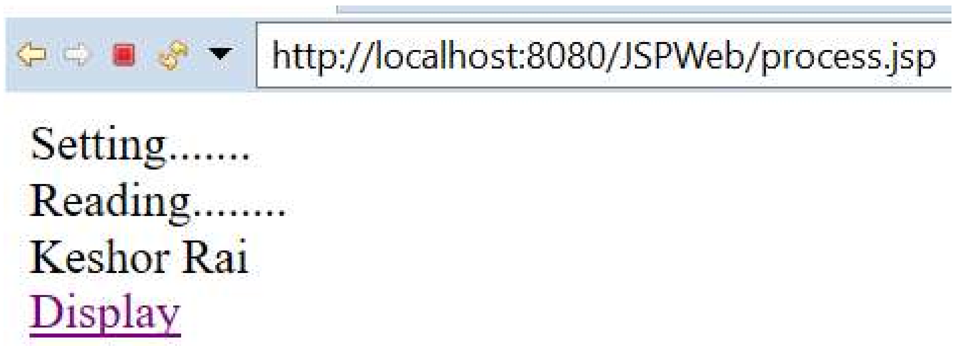
```
Reading........<br>
<jsp:getProperty property="fullName" name="person"/><br>
```

Output:



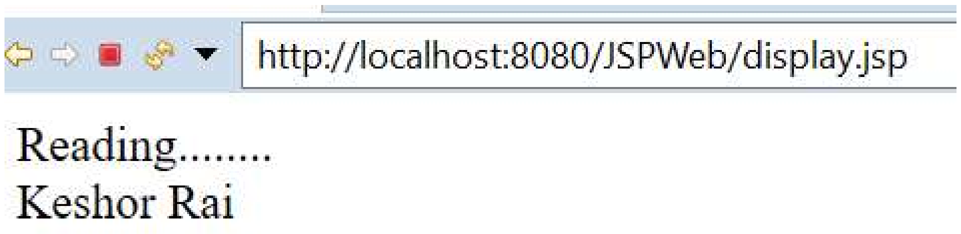
http://localhost:8080/JSPWeb/index.html

Name: Keshor Rai



http://localhost:8080/JSPWeb/process.jsp

```
Setting.......
Reading........
Keshor Rai
Display
```



http://localhost:8080/JSPWeb/display.jsp

```
Reading........
Keshor Rai
```

CUSTOM JSP ACTIONS
https://www.studytonight.com/jsp/
A custom tag is a user-defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The Web container then invokes those operations when the JSP page's servlet is executed.

EXPRESSION LANGUAGE
Expression Language(EL) was added to JSP 2.0 specification. The purpose of EL is to produce scriptless JSP pages. An expression can be mixed with static text/values and can also be combined with other expressions to form larger expression.

Implicit Object	Description
pageContext	It represents the PageContext object.
pageScope	It is used to access the value of any variable which is set in the Page scope
requestScope	It is used to access the value of any variable which is set in the Request scope.
sessionScope	It is used to access the value of any variable which is set in the Session scope
applicationScope	It is used to access the value of any variable which is set in the Application scope
param	Map a request parameter name to a single value
paramValues	Map a request parameter name to corresponding array of string values.

header	Map containing header names and single string values.
headerValues	Map containing header names to corresponding array of string values.
cookie	Map containing cookie names and single string values.

Example
index.jsp
```
<form method="POST" action="welcome.jsp">
   Name <input type="text" name="user" >
   <input type="submit" value="Submit">
</form>
```

welcome.jsp
```
<html>
   <head>
      <title>Welcome Page</title>
   </head>

         <body>
      <h1>Welcome ${param.name}</h1>
         </body>
</html>
```

EL Arithmetics

ARITHMETIC OPERATION	**OPERATOR**
Addition	+
Substraction	-
Multiplication	*
Division	/ and div
Remainder	% and mod

EL Relational and Logical operations

LOGICAL AND RELATIONAL OPERATOR	**OPERATOR**
Equals	== and eq
Not equals	!= and ne
Less Than	< and lt
Greater Than	> and gt
Greater Than or Equal	>= and ge
Less Than or Equal	<= and le
and	&& and and
or	|| and or
not	! and not

JSTL TAG LIBRARIES
JSP Standard Tag Library(JSTL) is a standard library of readymade tags. The JSTL contains several tags that can remove scriplet code from a JSP page by providing some ready to use, already implemented common functionalities.

JSTL is divided into 5 categories:
- JSTL Core
- JSTL Formatting
- STL sql
- JSTL XML
- JSTL functions

Using JSTL

- **Add taglib tag**
  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

- **Set value on variale**
  <c:set var="var1" value="Hello world of JSTL" />

- **Display value of variable**
  Current var1 : ${var1}</br>

JSTL CORE
The core group of tags are the most commonly used JSTL tags. Following is the syntax to include the JSTL Core library in JSP web project.

SN	TAG	DESCRIPTION
1	<c:set >	Sets the result of an expression
2	<c:out>	Displays the result of an expression
3	<c:remove >	Removes a scoped variable
4.	<c:catch>	Catches any Throwable that occurs in its body
5.	<c:if>	Simple conditional tag which evalutes its body if the supplied condition is true.
6	<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>.
7.	<c:when>	Subtag of <choose> that includes its body if its condition evalutes to 'true'.
8.	<c:otherwise >	Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluated to 'false'.
9.	<c:import>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
10.	<c:forEach >	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .
11.	<c:forTokens>	Iterates over tokens, separated by the supplied delimeters.
12.	<c:param>	Adds a parameter to a containing 'import' tag's URL.
13.	<c:redirect >	Redirects to a new URL.
14.	<c:url>	Creates a URL with optional query parameters

JSTL FORMATTING
The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites.

Include formatting library

```
<%@ taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>
```

Formatting Tags

SN	TAG	DESCRIPTION
1	<fmt:formatNumber>	To render numerical value with specific precision or format.
2	<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.
3	<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern.
4.	<fmt:parseDate>	Parses the string representation of a date and/or time
5.	<fmt:bundle>	Loads a resource bundle to be used by its tag body.
6	<fmt:setLocale>	Stores the given locale in the locale configuration variable.
7.	<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
8.	<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.

9.	<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable.
10.	<fmt:message>	Displays an internationalized message.
11.	<fmt:requestEncoding>	Sets the request character encoding.

STL SQL
The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.

Include SQL Tag Library
```
      <%@ taglib prefix = "sql" uri = "http://java.sun.com/jsp/jstl/sql" %>
```

SQL Tags

SN	TAG	DESCRIPTION
1	<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping.
2	<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
3	<sql:update>	Executes the SQL update defined in its body or through the sql attribute.
4.	<sql:param>	Sets a parameter in an SQL statement to the specified value.
5.	<sql:dateParam>	Sets a parameter in an SQL statement to the specified java.util.Date value.
6	<sql:transaction >	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

JSTL XML
The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents. Following is the syntax to include the JSTL XML library in your JSP.

Library
- XercesImpl.jar − Download it from https://www.apache.org/dist/xerces/j/
- xalan.jar − Download it from https://xml.apache.org/xalan-j/index.html

Note:
- tomcat installation path\lib

Include XML Library
```
      <%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>
```

XML Tags

SN	TAG	DESCRIPTION
1	<x:out>	Like <%= ... >, but for XPath expressions.
2	<x:parse>	Used to parse the XML data specified either via an attribute or in the tag body.
3	<x:set >	Sets a variable to the value of an XPath expression.
4.	<x:if >	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
5.	<x:forEach>	To loop over nodes in an XML document.
6	<x:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise> tags.
7	<x:when>	Subtag of <choose> that includes its body if its expression evalutes to 'true'.
8	<x:otherwise>	Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluates to 'false'.
9	<x:transform>	Applies an XSL transformation on a XML document.
10	<x:param >	Used along with the transform tag to set a parameter in the XSLT stylesheet.

JSTL FUNCTIONS

JSTL includes a number of standard functions, most of which are common string manipulation functions.

Include JSTL Functions Library

```
      <%@ taglib prefix = "fn" uri = "http://java.sun.com/jsp/jstl/functions" %>
```

JSTL Functions

SN	TAG	DESCRIPTION
1	fn:contains()	Tests if an input string contains the specified substring.
2	fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way.
3	fn:endsWith()	Tests if an input string ends with the specified suffix.
4.	fn:escapeXml()	Escapes characters that can be interpreted as XML markup.
5.	fn:indexOf()	Returns the index withing a string of the first occurrence of a specified substring.
6	fn:join()	Joins all elements of an array into a string.
7	fn:length()	Returns the number of items in a collection, or the number of characters in a string.
8	fn:replace()	Returns a string resulting from replacing in an input string all occurrences with a given string.
9	fn:split()	Splits a string into an array of substrings.
10	fn:startsWith()	Tests if an input string starts with the specified prefix.
11	fn:substring()	Returns a subset of a string.
12	fn:substringAfter()	Returns a subset of a string following a specific substring.
13	fn:substringBefore()	Returns a subset of a string before a specific substring.
14	fn:toLowerCase()	Converts all of the characters of a string to lower case.
15	fn:toUpperCase()	Converts all of the characters of a string to upper case.
16	fn:trim()	Removes white spaces from both ends of a string.

HTML/JSP WORKING CONCEPT