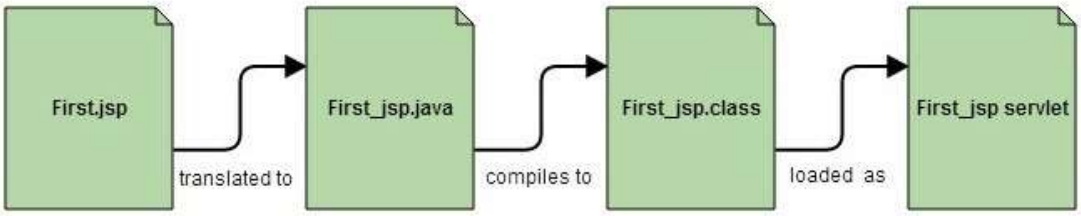
Introduction to JSP

JSP technology is used to create dynamic web applications. **JSP** pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it.

JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. **Using JSP, one can easily separate Presentation and Business logic** as a web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

In the end a JSP becomes a Servlet



Why JSP is preffered over servlets?

- JSP provides an easier way to code dynamic web pages.
- JSP does not require additional files like, java class files, web.xml etc
- Any change in the JSP code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.

- JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

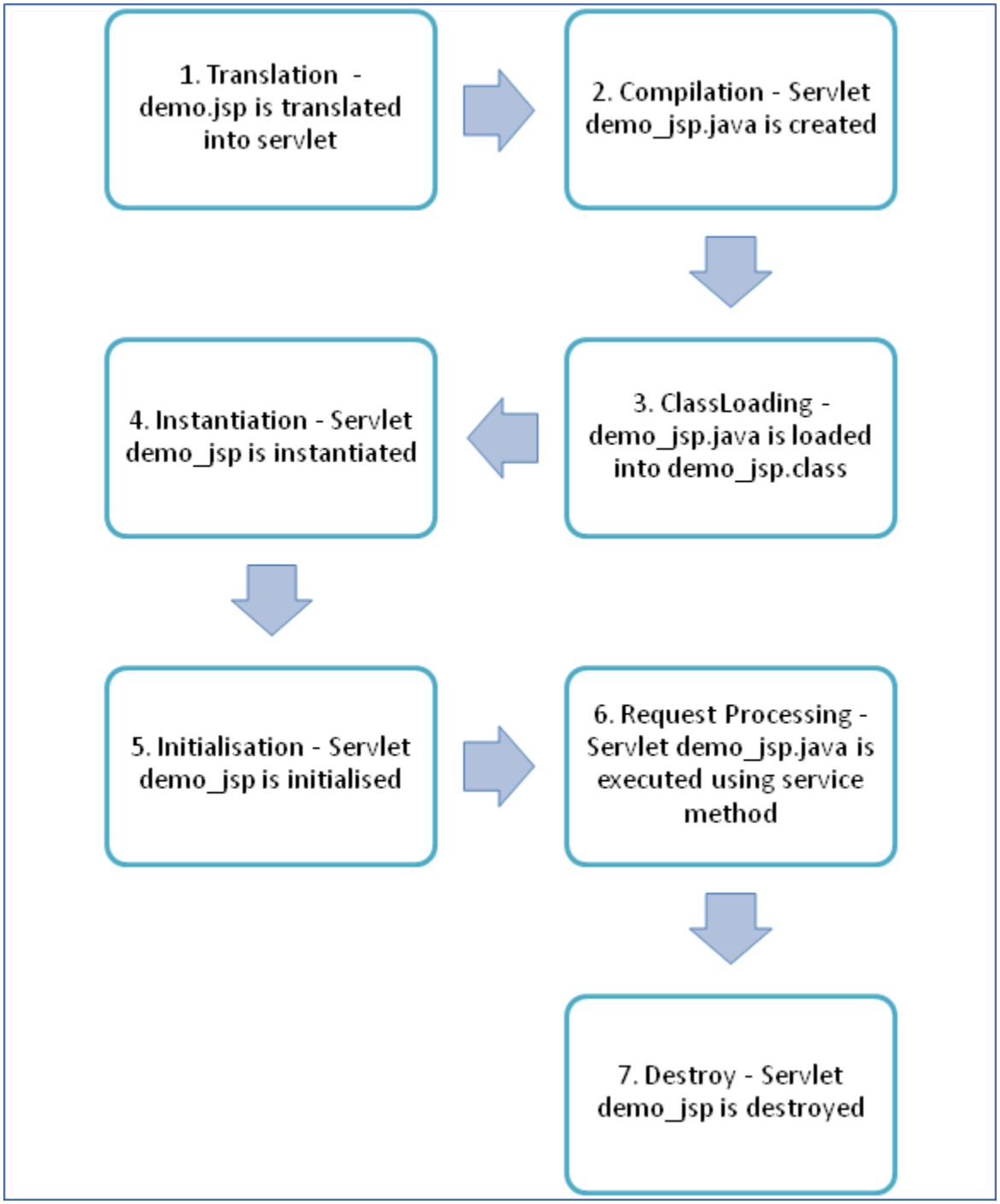
Advantage of JSP

- Easy to maintain and code.
- High Performance and Scalability.
- JSP is built on Java technology, so it is platform independent.

Lifecycle of JSP

A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code. Following are the JSP Lifecycle steps:

1. Translation of JSP to Servlet code.
2. Compilation of Servlet to bytecode.
3. Loading Servlet class.
4. Creating servlet instance.
5. Initialization by calling `jspInit()` method
6. Request Processing by calling `_jspService()` method
7. Destroying by calling `jspDestroy()` method



Web Container translates JSP code into a **servlet class source(.java) file**, then compiles that into a java servlet class. In the third step, the servlet class bytecode is loaded using classloader. The Container then creates an instance of that servlet class.

The initialized servlet can now service request. For each request the **Web Container** call the **_jspService()** method. When the

Container removes the servlet instance from service, it calls the **jspDestroy()** method to perform any required clean up.

1. **Translation of the JSP Page:**

A Java servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.

- The JSP container interprets the standard directives and actions, and the custom actions referencing tag libraries (they are all part of JSP page and will be discussed in the later section) used in this JSP page.

Demo.jsp

```
<html>
<head>
<title>Demo JSP</title>
</head>
<%
int demvar=0;%>
<body>
Count is:
<% Out.println(demovar++); %>
<body>
</html>
```

```
1  Public class demp_jsp extends HttpServlet{
2     Public void _jspservice(HttpServletRequest request, HttpServletResponse response)
3           Throws IOException, ServletException
4        {
5  PrintWriter out = response.getWriter();
6  response.setContentType("text/html");
7  out.write("<html><body>");
8  int demovar=0;
9  out.write("Count is:");
10 out.print(demovar++);
11 out.write("</body></html>");
12 }
13 }
14
```

2. **Compilation of the JSP Page**

- The generated java servlet file is compiled into java servlet class
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo_jsp.java is compiled to a class file demo_jsp.class

3. **Classloading**

- Servlet class that has been loaded from JSP source is now loaded into the container

4. **Instantiation**

- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
- A JSPPage interface which is provided by container provides init() and destroy () methods.
- There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.

5. **Initialization**

```
public void jspInit()
{
     //initializing the code
}
```

- _jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that
- It is only called once during a JSP life cycle, the method for initialization is declared as shown above

6. **Request processing**

```
void _jspservice(HttpServletRequest request HttpServletResponse
response)
{
     //handling all request and responses
}
```

- _jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.eGET, POST, etc.

7. **Destroy**

```
public void _jspdestroy()
{
            //all clean up code
}
```

- _jspdestroy() method is also invoked by the container
- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.
- We can override jspdestroy() method when we perform any cleanup such as releasing database connections or closing open files.

Creating a JSP Page

A JSP page looks similar to an HTML page, but a JSP page also has Java code in it. We can put any regular Java Code in a JSP file using a **scriplet tag** which start with <% and ends with %>. JSP pages are used to develop dynamic responses.

Example of creating a JSP Page in Eclipse

JSP Scripting Element

JSP Scripting element are written inside <% %> tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

Scripting Element	Example
Comment	<%-- comment --%>
Directive	<%@ directive %>
Declaration	<%! declarations %>
Scriptlet	<% scriplets %>
Expression	<%= expression %>

JSP Comment

JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing. JSP comments are only seen in the JSP page. These comments are not included in servlet source code during translation phase, nor they appear in the HTTP response. Syntax of JSP comment is as follows :

```
<%-- JSP  comment  --%>
```

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
   <%
      int count = 0;
   %>
   <body>
      <%-- Code to show page count  --%>
      Page Count is <% out.println(++count); %>
   </body>
</html>
```

JSP Scriptlet Tag

Scriptlet Tag allows you to write java code inside JSP page. Scriptlet tag implements the `_jspService` method functionality by writing script/java code. Syntax of Scriptlet Tag is as follows :

```
<% JAVA CODE %>
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
   <%
      int count = 0;
   %>
   <body>
      Page Count is <% out.println(++count); %>
   </body>
</html>
```

We have been using the above example since last few lessons and in this scriptlet tags are used. Everything written inside the scriptlet tag is compiled as java code. Like in the above example, we initialize `count` variable of type `int` with a value of 0. And then we print it while using `++` operator to perform addition on it.

JSP makes it so easy to perform calculations, database interactions etc directly from inside the HTML code. Just write your java code inside the scriptlet tags.

Example of JSP Scriptlet Tag

Index.html

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

welcome.jsp

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Welcome Page</title>
   </head>
   <%
    String user = request.getParameter("user");
   %>

   <body>
      Hello, <% out.println(user); %>
   </body>
</html>
```

Mixing scriptlet Tag and HTML

```
<table border = 1>
<%
   for ( int i = 0; i < n; i++ ) {
      %>
      <tr>
      <td>Number</td>
      <td><%= i+1 %></td>
      </tr>
```

```
      <%
   }
%>
</table>

<%
   if ( hello ) {
      %>
      <p>Hello, world</p>
      <%
   } else {
      %>
      <p>Goodbye, world</p>
      <%
   }
%>
```

JSP Declaration Tag

Declare a variable or method in JSP inside **Declaration Tag**, it means the declaration is made inside the Servlet class but outside the service(or any other) method. You can declare static member, instance variable and methods inside **Declaration Tag**.
Syntax of Declaration Tag :

```
<%! declaration %>
```

Example of Declaration Tag

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
   <%!
      int count = 0;
   %>
   <body>
      Page Count is:
      <% out.println(++count); %>
   </body>
</html>


public class hello_jsp extends HttpServlet
{
  int count=0;
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
                        throws IOException,ServletException
```

```
   {
     PrintWriter out = response.getWriter();
     response.setContenType("text/html");
     out.write("<html><body>");

     out.write("Page count is:");
     out.print(++count);
     out.write("</body></html>");
   }
}
```

When to use Declaration tag and not scriptlet tag

If you want to include any method in your JSP file, then you must use the declaration tag, because during translation phase of JSP, methods and variables inside the declaration tag, becomes instance methods and instance variables and are also assigned default values.

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
   <%!
      int count = 0;
      int getCount() {
         System.out.println( "In getCount() method" );
         return count;
      }
   %>
   <body>
      Page Count is:
      <% out.println(getCount()); %>
  </body>
</html>


public class hello_jsp extends HttpServlet
{
  int count = 0;
  int getCount() {
     System.out.println( "In getCount() method" );
     return count;
  }
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
                        throws IOException,ServletException
   {
     PrintWriter out = response.getWriter();
```

```
    response.setContenType("text/html");
    out.write("<html><body>");

    out.write("Page count is:");
    out.print(getCount());
    out.write("</body></html>");
  }
}
```

JSP Directive Tag

Directive Tag gives special instruction to Web Container at the time of page translation. Directive tags are of three types: **page**, **include** and **taglib**.

Directive	Description
<%@ page ... %>	defines page dependent properties such as language, session, errorPage etc.
<%@ include ... %>	defines file to be included.
<%@ taglib ... %>	declares tag library used in the page

The **Page directive** defines a number of page dependent properties which communicates with the Web Container at the time of translation.

- JSP directives are the messages to JSP container. They provide global information about an entire JSP page.
- JSP directives are used to give special instruction to a container for translation of JSP to servlet code.
- In JSP life cycle phase, JSP has to be converted to a servlet which is the translation phase.
- They give instructions to the container on how to handle certain aspects of JSP processing
- Directives can have many attributes by comma separated as key-value pairs.
- In JSP, directive is described in <%@ %> tags.

Basic syntax of using the page directive is <%@ page attribute="value" %> where attributes can be one of the following :

- import attribute
- language attribute
- extends attribute
- session attribute
- isThreadSafe attribute
- isErrorPage attribute
- errorPage attribute
- contentType attribute
- autoFlush attribute
- buffer attribute

import attribute

The import attribute defines the set of classes and packages that must be imported in servlet class definition. For example

```
<%@ page import="java.util.Date" %>
or
<%@ page import="java.util.Date,java.net.*" %>
```

language attribute

language attribute defines scripting language to be used in the page.

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

extends attribute

extends attribute defines the class name of the superclass of the servlet class that is generated from the JSP page.

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

<%@ page extends="demotest.DemoClass" %>
```

session attribute

session attribute defines whether the JSP page is participating in an HTTP session. The value is either true or false.

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

isThreadSafe attribute

isThreadSafe attribute declares whether the JSP is thread-safe. The value is either true or false

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

isErrorPage attribute

isErrorPage attribute declares whether the current JSP Page represents another JSP's error page.

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

errorPage attribute

errorPage attribute indicates another JSP page that will handle all the run time exceptions thrown by current JSP page. It specifies the URL path of another page to which a request is to be dispatched to handle run time exceptions thrown by current JSP page.

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

```
    errorPage="errorHandler.jsp"%>
```

contentType attribute

contentType attribute defines the MIME type for the JSP response.

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

autoFlush attribute

autoFlush attribute defines whether the buffered output is flushed automatically. The default value is "true".

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

buffer attribute

buffer attribute defines how buffering is handled by the implicit **out** object.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    buffer="16KB"%>

Example program
<%@ page language="java" contentType="text/html;" pageEncoding="ISO-8859-1"
    isELIgnored="false"%>
 <%@page import="java.util.Date" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Directive ABC JSP1</title>
</head>
<body>
<a>Date is:</a>
<%= new java.util.Date() %>
</body>
</html>
```

JSP Expression Tag

Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the **out.print()** method.

Syntax of Expression Tag

```
<%= Java Expression %>

<%= (2*5) %>

out.print((2*5));
```

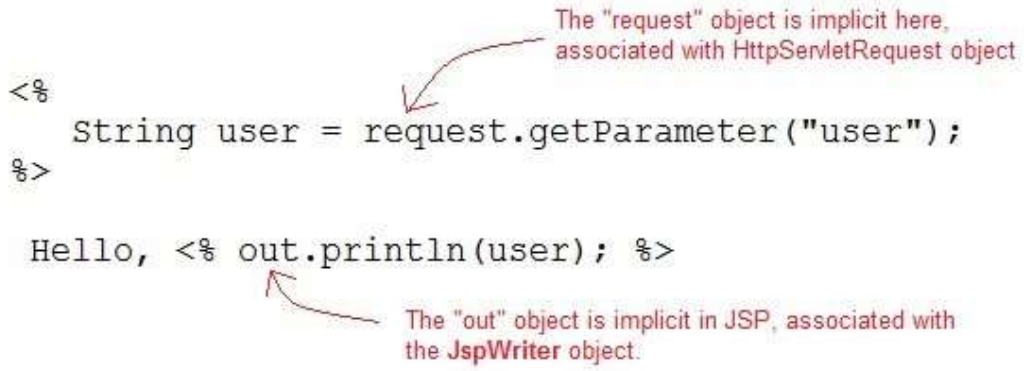
Example:

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
         <%
     int count = 0;
         %>
         <body>
      Page Count is <%= ++count %>
         </body>
</html>
```

Implicit Objects in JSP

JSP provide access to some implicit object which represent some commonly used objects for servlets that JSP page developers might need to use. For example you can retrieve HTML form parameter data by using **request** variable, which represent the **HttpServletRequest** object.

- JSP implicit objects are created during the translation phase of JSP to the servlet.
- These objects can be directly used in scriplets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.



Following are the JSP implicit object

Implicit Object	Description
request	The **HttpServletRequest** object associated with the request.
response	The **HttpServletRequest** object associated with the response that is sent back to the browser.
out	The **JspWriter** object associated with the output stream of the response.
session	The **HttpSession** object associated with the session for the given user of request.
application	The **ServletContext** object for the web application.
config	The **ServletConfig** object associated with the servlet for current JSP page.
pageContext	The **PageContext** object that encapsulates the enviroment of a single request for this current JSP page
page	The **page** variable is equivalent to **this** variable of Java programming language.
exception	The **exception** object represents the **Throwable** object that was thrown by some other JSP page.

Out

- Out is one of the implicit objects to write the data to the buffer and send output to the client in response
- Out object allows us to access the servlet's output stream
- Out is object of javax.servlet.jsp.jspWriter class
- While working with servlet, we need printwriter object

Example:

```
<%@ 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>Implicit ABC JSP1</title>
</head>
<body>
<% int num1=10;int num2=20;
out.println("num1 is " +num1);
out.println("num2 is "+num2);
%>
</body>
</html>
```

Request

- The request object is an instance of java.servlet.http.HttpServletRequest and it is one of the argument of service method
- It will be created by container for every request.
- It will be used to request the information like parameter, header information , server name, etc.
- It uses getParameter() to access the request parameter.

```
<%@ 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>Implicit ABC form JSP2</title>
</head>
<body>
<form action="ABC.jsp">
<input type="text" name="username">
<input type="submit" value="submit">
</form>
</body>
</html>
```

Response

- “Response” is an instance of class which implements HttpServletResponse interface
- Container generates this object and passes to _jspservice() method as parameter
- “Response object” will be created by the container for each request.
- It represents the response that can be given to the client
- The response implicit object is used to content type, add cookie and redirect to response page

```
<%@ 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>Implicit ABC JSP4</title>
</head>
<body>
<%response.setContentType("text/html"); %>
</body>
</html>
```

Config

- “Config” is of the type java.servlet.servletConfig
- It is created by the container for each jsp page
- It is used to get the initialization parameter in web.xml

```
<%@ 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>Implicit ABC JSP5</title>
</head>
<body>
<% String servletName = config.getServletName();
out.println("Servlet Name is " +servletName);%>
</body>
</html>
```

Application

- Application object (code line 10) is an instance of javax.servlet.ServletContext and it is used to get the context information and attributes in JSP.
- Application object is created by container one per application, when the application gets deployed.
- Servletcontext object contains a set of methods which are used to interact with the servlet container.We can find information about the servlet container

```
<%@ 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>ABC Implicit JSP6</title>
</head>
<body>
<% application.getContextPath(); %>
</body>
</html>
```

Session

- The session is holding “httpsession” object(code line 10).
- Session object is used to get, set and remove attributes to session scope and also used to get session information

```
<%@ 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>Implicit JSP</title>
</head>
<body>
<% session.setAttribute("user","ABCJSP"); %>
<a href="implicit_jsp8.jsp">Click here to get user name</a>
</body>
</html>
```

Another page

```
<%@ 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>implicit ABC JSP8</title>
</head>
<body>
<% String name = (String)session.getAttribute("user");
out.println("User Name is " +name);
%>
</body>
</html>
```

PageContext

- This object is of the type of pagecontext.
- It is used to get, set and remove the attributes from a particular scope

Scopes are of 4 types:

- Page
- Request
- Session
- Application

Page Scope

```
<%@ 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>Implicit ABC JSP9</title>
</head>
<body>
<%
pageContext.setAttribute("student","ABCstudent",pageContext.PAGE_SCOPE);
String name = (String)pageContext.getAttribute("student");
out.println("student name is " +name);
%>
</body>
</html>
```

Session Scope

Index.html

```
<html>
<head>
<title> User Login Page - Enter details</title>
</head>
```

```
<body>
<form action="validation.jsp">
Enter User-Id: <input type="text" name="uid"><br>
Enter Password: <input type="text" name="upass"><br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

validation.jsp

```
<html>
<head> <title> Validation JSP Page</title>
</head>
<body>
<%
String id=request.getParameter("uid");
String pass=request.getParameter("upass");
out.println("hello "+id);
pageContext.setAttribute("UName", id, PageContext.SESSION_SCOPE);
pageContext.setAttribute("UPassword", pass,
PageContext.SESSION_SCOPE);
%>
<a href="display.jsp">Click here to see what you have entered </a>
</body>
</html>
```

display.jsp

```
<html>
<head>
<title>Displaying User Details</title>
</head>
<body>
<%
String username= (String) pageContext.getAttribute("UName",
PageContext.SESSION_SCOPE);
String userpassword= (String) pageContext.getAttribute("UPassword",
PageContext.SESSION_SCOPE);
out.println("Hi "+username);
out.println("Your Password is: "+userpassword);
%>
</body>
</html>
```

Page

- Page implicit variable holds the currently executed servlet object for the corresponding jsp.
- Acts as this object for current jsp page.

```
<%@ 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>Implicit ABC JSP10</title>
</head>
<body>
<% String pageName = page.toString();
out.println("Page Name is " +pageName);%>
</body>
</html>
```

Exception

- Exception is the implicit object of the throwable class.
- It is used for exception handling in JSP.
- The exception object can be only used in error pages.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isErrorPage="true"%>
<!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>Implicit ABC JSP 11</title>
</head>
<body>
<%int[] num1={1,2,3,4};
out.println(num1[5]);%>
<%= exception %>
</body>
</html>
```

Display exception

JSP Action Tags: List of JSP Standard Action Elements

What is JSP Action?

JSP actions use the construct in XML syntax to control the behavior of the servlet engine. We can dynamically insert a file, reuse the beans components, forward user to another page, etc. through JSP Actions like include and forward. Unlike directives, actions are re-evaluated each time the page is accessed.

Standard Action Tags available in JSP

There are 12 types of Standard Action Tags in JSP. Here is the list of Standard Action tags in JSP:

- jsp:useBean
- jsp:include
- jsp:setProperty
- jsp:getProperty
- jsp:forward
- jsp:plugin
- jsp:attribute
- jsp:body
- jsp:text
- jsp:param
- jsp:attribute
- jsp:output

jsp:useBean

- This action name is used when we want to use beans in the JSP page.
- With this tag, we can easily invoke a bean.

Syntax of jsp: UseBean:

```
<jsp:useBean id="" class="" />
```

Here it specifies the identifier for this bean and class is full path of the bean class

```
<%@ 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>Action JSP1</title>
</head>
<body>
<jsp:useBean id="name" class="demotest.DemoClass">
</body>
</html>
```

jsp:include

- It also used to insert a jsp file into another file, just like including Directives.
- It is added during request processing phase
- **Syntax of jsp:include**
- `<jsp:include page="page URL" flush="true/false">`

Example:

Action_jsp2 (Code Line 10) we are including a date.jsp file

```
<%@ 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>Date ABC JSP</title>
</head>
<body>
<jsp:include page="date.jsp" flush="true" />
</body>
</html>
```

Date.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>
<p>
Today's date: <%= {new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

jsp:setProperty

- This property of standard actions in JSP is used to set the property of the bean.
- We need to define a bean before setting the property
- `<jsp:setproperty name="" property="" >`

jsp:getProperty

- This property is used to get the property of the bean.

- It converts into a string and finally inserts into the output.
- `<jsp:getAttribute name="" property="" >`

Example of setProperty and getProperty:

TestBean.java:

```
package demotest;

import java.iO.Serializable;

public class TestBean implements Serializable{

        private String msg = "null";

        public String getMsg() {
                return msg;
        }

        public void setMsg(String msg) {
                this.msg = msg;
        }
}
```

Action_jsp3.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>ABC Action 3</title>
</head>
<body>
<jsp:useBean id="ABCTest" class="demotest.TestBean" />
<jsp:setProperty name="ABCTest" property="msg" value="ABCTutorial" />
<jsp:getProperty name="ABCTest" property="msg" />
</body>
</html>
```

jsp:forward

It is used to forward the request to another jsp or any static page.

Here the request can be forwarded with no parameters or with parameters.

Syntax:

```
<jsp:forward page="value">
```

Example:

Action_jsp41.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>ABC Action JSP1</title>
</head>
<body>
<jsp:forward page="jsp_action_42.jsp" />
</body>
</html>
```

Jsp_action_42.jsp

```
ABC Action JSP2</title>
</head>
<body>
<a>This is after forward page</a>
</body>
</html>
```

jsp:plugin

- It is used to introduce Java components into jsp, i.e., the java components can be either an applet or bean.
- It detects the browser and adds <object> or <embed> JSP tags into the file
- **Syntax:**
- `<jsp:plugin type="applet/bean" code="objectcode" codebase="objectcodebase">`

jsp:param

- This is child object of the plugin object described above
- It must contain one or more actions to provide additional parameters.

Syntax:

```
<jsp:params>
<jsp:param name="val" value="val"/ >
</jsp:params>
```

Example of plugin and param

Action_jsp5.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>Action ABC jsp5</title>
</head>
<body>
```

```
<jsp:plugin type="bean" code="Student.class" codebase="demotest.Student">
  <jsp:params>
     <jsp:param name="id" value="5" />
     <jsp:param name="name" value="ABC" />
  </jsp:params>
</jsp:plugin>
</body>
</html>
```

Student.java

```
package demotest;

import java.io.Serializable;

public class Student implements Serializable {

        public String getName () {
                return name;
        }
        public void setName (String name) {
                this.name = name;
        }
        public int getId() {
                return id;
        }
        public void setId (int id) {
                this.id = id;
        }
        private String name = "null";
        private int id = 0;

}
```

jsp:body

- This tag is used to define the XML dynamically i.e., the Elements can generate during request time than compilation time.
- It actually defines the XML, which is generated dynamically element body.

Syntax:

```
<jsp:body></jsp:body>
```

jsp:attribute

- This tag is used to define the XML dynamically i.e. the elements can be generated during request time than compilation time
- It actually defines the attribute of XML which will be generated dynamically.

Syntax:

```
<jsp:attribute></jsp:attribute>
```

Example of body and attribute:

Action_jsp6.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>Action ABC JSP6</title>
</head>
<body>
<jsp:element name="ABCXMLElement">
<jsp:attribute name="ABCXMLattribute">
Value
</jsp:attribute>
<jsp:body>ABC XML</jsp:body>
</jsp:element>
</body>
</html>
```

jsp:text

- It is used to template text in JSP pages.
- Its body does not contain any other elements, and it contains only text and EL expressions.

Syntax:

```
<jsp:text>template text</jsp:text>
```

Example:

Action_jsp7.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>ABC Action JSP7</title>
</head>
<body>
<jsp:text>ABC Template Text</jsp:text>
</body>
</html>
```

jsp:output

- It specifies the XML declaration or the DOCTYPE declaration of jsp
- The XML declaration and DOCTYPE are declared by the output

Syntax:

```
<jsp:output doctype-root-element="" doctype-system="">
```

Example:

```
<%@ 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>Action ABC JSP8</title>
</head>
<body>
<jsp:output doctype-root-element="html PUBLIC" doctype-
system="http://www.w3.org/TR/html4/loose.dtd"/>
</body>
</html>
```

What is Expression Language (EL)?

Expression Language (EL) is mechanism that simplifies the accessibility of the data stored in Java bean component and other object like request, session and application, etc. There are many operators in JSP that are used in EL like arithmetic and logical operators to perform an expression. It was introduced in JSP 2.0

JSP Syntax of Expression Language (EL)

Syntax of EL :$(expression)

- In JSP, whatever present in the braces gets evaluated at runtime sent to the output stream.
- The expression is a valid EL expression and it can be mixed with a static text and can be combined with other expression to form larger expression.

To get a better idea, on how expression works in JSP, we will see below example.

In this example, we will see how EL is used as an operator to add two numbers (1+2) and get the output respectively.

```
<%@ 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>ABC JSP1</title>
</head>
<body>
<a>Expression is:</a>
${1+2};
</body>
</html>
```

Flow Control Statements:

JSP provides the power of Java to be embedded in the application. We can use all the APIs and building blocks of Java in JSP programming including control flow statements which include decision making and the loop statements.

There are two types of flow control statements described below;

1. Decision-Making statements
2. Loop Statements

Decision-Making Statements:

Decision-making statement in JSP is based on whether the condition set is true or false. The statement will behave accordingly.

There are two types of decision-making statements described below:

1. If – else
2. switch

JSP If-else

"If else" statement is basic of all control flow statements, and it tells the program to execute the certain section of code only if the particular test evaluates to true.

This condition is used to test for more than one condition whether they are true or false.

- If the first condition is true then "if block" is executed and
- if it is false then "else block" is executed

Syntax for if – else statement:

```
If(test condition)
{
        //Block of statements
}
else
{
```

```
    //Block of statements
}
<%@ 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>ABC JSP2</title>
</head>
<body>
<%! int month=5; %>
<% if(month==2){ %>
<a>Its February</a>
<% }else{ %>
<a>Any month other than February</a>
<%} %>
</body>
</html>
```

JSP Switch

The body of the switch statement is called as a “switch block”.

- The switch case is used to check the number of possible execution paths.
- A switch can be used with all data types
- The switch statements contain more than one cases and one default case
- It evaluates the expression then executes all the statements following the matching case

Syntax for switch statement:

```
switch (operator)
{
   Case 1:
       Block of statements
break;
   Case 2:
      Block of statements
break;


case n:
    Block of statements
break;
default:
  Block of statements
break;
}
```

- Switch block begins with one parameter, which is the operator that needs to be passed and

- Then there are different cases which provides condition and whichever matches with the operator that case is executed.

```
<%@ 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>ABC JSP3</title>
</head>
<body>
<%! int week=2; %>
<% switch(week){
case 0:
    out.println("Sunday");
    break;
case 1:
    out.println("Monday");
    break;
case 2:
    out.println("Tuesday");
    break;
case 3:
    out.println("wednesday");
    break;
case 4:
    out.println("Thursday");
    break;
case 5:
    out.println("Friday");
    break;
    default:
        out.println("Saturday");
    }
%>
</body>
</html>
```

Loop Statements

JSP For loop

It is used for iterating the elements for a certain condition, and it has three parameters.

- Variable counter is initialized
- Condition till the loop has to be executed
- Counter has to be incremented

For loop Syntax:

```
For(inti=0;i<n;i++)
{
           //block of statements
```

```
}
<%@ 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>ABC JSP4</title>
</head>
<body>
<%! int num=5; %>
<% out.println("Numbers are:");
for(int i=0;i<num;i++){
    out.println(i);
}%>
</body>
</html>
```

JSP While loop

It is used to iterate the elements wherein it has one parameter of the condition.

Syntax:

```
While(i<n)
{
      //Block of statements
}
<%@ 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>ABC JSP5</title>
</head>
<body>
<%! int day=2; int i=1; %>
<% while(day>=i){
    if(day==i){
        out.println("Its Monday");
        break;}
    i++;}
%>

</body>
</html>
```

JSP Operators

JSP Operators supports most of its arithmetic and logical operators which are supported by java within expression language (EL) tags.

Frequently used operators are mentioned below:

Following are the operators:

.	Access a bean property or Map entry
[]	Access an array or List element
( )	Group a subexpression to change the evaluation order
+	Addition
–	Subtraction or negation of a value
*	Multiplication
/ or div	Division
% or mod	Modulo (remainder)
== or eq	Test for equality
!= or ne	Test for inequality
< or lt	Test for less than
> or gt	Test for greater than
<= or le	Test for less than or equal
>= or ge	Test for greater than or equal
&& or and	Test for logical AND
|| or or	Test for logical OR
! or not	Unary Boolean complement
Empty	Test for empty variable values

```
<%@ 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>ABC JSP6</title>
</head>
<body>
<% int num1=10; int num2 = 50;
   int num3 = num1+num2;
   if(num3 != 0 || num3 > 0){
       int num4= num1*num2;
       out.println("Number 4 is " +num4);
```

```
        out.println("Number 3 is " +num3);
    }%>
</body>
</html>
```

What is Exception in JSP?

Exceptions in JSP occur when there is an error in the code either by the developer or internal error from the system. Exception handling in JSP is the same as in Java where we manage exceptions using Try Catch blocks. Unlike Java, there are exceptions in JSP also when there is no error in the code.

Types of Exceptions in JSP

Exceptions in JSP are of three types:

1. Checked Exception
2. Runtime Exception
3. Error Exception

Checked Exceptions

It is normally a user error or problems which are not seen by the developer are termed as checked exceptions.

Some Checked exception examples are:

1. **FileNotFoundException**: This is a checked exception (where it tries to find a file when the file is not found on the disk).
2. **IO Exception**: This is also checked exception if there is any exception occurred during reading or writing of a file then the IO exception is raised.
3. **SQLException**: This is also a checked exception when the file is connected with SQL database, and there is issue with the connectivity of the SQL database then SQLException is raised

Runtime Exceptions

Runtime exceptions are the one which could have avoided by the programmer. They are ignored at the time of compilation.

Some Runtime exception examples are:

1. **ArrayIndexOutOfBoundsException**: This is a runtime exception when array size exceeds the elements.
2. **ArithmeticException**: This is also a runtime exception when there are any mathematical operations, which are not permitted under normal conditions, for example, dividing a number by 0 will give an exception.

3. **NullPointer Exception**: This is also a runtime exception which is raised when a variable or an object is null when we try to access the same. This is a very common exception.

Errors:

The problem arises due to the control of the user or programmer. If stack overflows, then error can occur.

Some examples of the error are listed below:

1. **Error**: This error is a subclass of throwable which indicates serious problems that an application cannot catch.
2. **Instantiation Error**: This error occurs when we try to instantiate an object, and it fails to do that.
3. **Internal Error**: This error occurs when there is an error occurred from JVM i.e. Java Virtual Machine.

Error Exceptions

It is an instance of the throwable class, and it is used in error pages.

Some methods of throwable class are:

- **Public String getMessage()** – returns the message of the exception.
- **Public throwablegetCause()** – returns cause of the exception
- **Public printStackTrace()**– returns the stacktrace of the exception.

How to Handle Exception in JSP

Exception_example.jsp

```
<%@ page errorPage="ABC_error.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>Exception ABC JSP1</title>
</head>
<body>
<%
 int num = 10;
   if (num == 10)
   {
      throw new RuntimeException("Error condition!!!");
   }
 %>
```

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

ABC_error.jsp

```
<%@ page isErrorPage="true" %>
<%@ 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>ABC Exception Page</title>
</head>
<body>
<p>ABC Exception has occurred</p>
<% exception.printStackTrace(response.getWriter()); %>
</body>
</html>
```

JSP Include Directive

The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive is:

```
<%@ include file="filename.jsp" %>
```

welcome.jsp

```
<html>
  <head>
    <title>Welcome Page</title>
  </head>

 <body>
    <%@ include file="header.jsp" %>
    Welcome, User
 </body>
</html>
```

header.jsp

```
<html>
 <body>
   <img src="header.jpg" alt="This is Header image" / >
 </body>
</html>
```

Taglib Directive

This directive basically allows user to use Custom tags in JSP.

Syntax of Taglib Directive:

```
<%@taglib uri ="taglibURI" prefix="tag prefix"%>
```

JSP Custom tags

User-defined tags are known as **custom tags**.
To create a custom tag we need three things:
1) Tag handler class: In this class we specify what our custom tag will do when it is used in a JSP page.
2) TLD file: Tag descriptor file where we will specify our tag name, tag handler class and tag attributes.
3) JSP page: A JSP page where we will be using our custom tag.

Tag handler class:
A tag handler class should implement `Tag`/`IterationTag`/ `BodyTag` interface or it can also extend `TagSupport`/`BodyTagSupport`/`SimpleTagSupport` class. All the classes that support custom tags are present inside `javax.servlet.jsp.tagext`.
`Details.java`

```
Package book.com
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class Details extends SimpleTagSupport {
   //StringWriter object
   StringWriter sw = new StringWriter();

   public void doTag() throws JspException, IOException
   {
       getJspBody().invoke(sw);
       JspWriter out = getJspContext().getOut();
       out.println(sw.toString()+"Appended Custom Tag Message");
   }
}
```

TLD File
This file should present at the location: `Project Name/WebContent/WEB-INF/` and it should have a **.tld** extension.

Note:
`<name>` tag: custom tag name. In this example we have given it as `MyMsg`

`<tag-class>` tag: Fully qualified class name. Our tag handler class `Details.java` is in package `book.com` so we have given the value as `book.com.Details`.

`message.tld`

```
<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>My Custom Tag: MyMsg</short-name>
<tag>
<name>MyMsg</name>
<tag-class>book.com.Details</tag-class>
<body-content>scriptless</body-content>
</tag>
</taglib>
```

Index.Jsp page

```
<%@ taglib prefix="myprefix" uri="WEB-INF/message.tld"%>
<html>
<head>
  <title>Accessing Custom Tag Body Example</title>
</head>
<body>
  <myprefix:MyMsg>
    Test String
  </myprefix:MyMsg>
</body>
</html>
```

Cookies in JSP

What are Cookies?

- Cookies are the text files which are stored on the client machine.
- They are used to track the information for various purposes.
- It supports HTTP cookies using servlet technology
- The cookies are set in the HTTP Header.
- If the browser is configured to store cookies, it will keep information until expiry date.

JSP cookies methods

Following are the cookies methods:

- *Public void setDomain(String domain)*

  This JSP set cookie is used to set the domain to which the cookie applies
- *Public String getDomain()*

  This JSP get cookie is used to get the domain to which cookie applies
- *Public void setMaxAge(int expiry)*

  It sets the maximum time which should apply till the cookie expires
- *Public intgetMaxAge()*

  It returns the maximum age of cookie in JSP
- *Public String getName()*

  It returns the name of the cookie
- *Public void setValue(String value)*

  Sets the value associated with the cookie
- *Public String getValue()*

  Get the value associated with the cookie
- *Public void setPath(String path)*

  This set cookie in JSP sets the path to which cookie applies
- *Public String getPath()*

  It gets the path to which the cookie applies
- *Public void setSecure(Boolean flag)*

  It should be sent over encrypted connections or not.
- *Public void setComment(String cmt)*

  It describes the cookie purpose
- *Public String getComment()*

  It the returns the cookie comments which has been described.

How to Handle Cookies in JSP

1. Creating the cookie object
2. Setting the maximum age
3. Sending the cookie in HTTP response headers

Action_cookie.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> Cookie</title>
</head>
<body>
<form action="action_cookie_main.jsp" method="GET">
Username: <input type="text" name="username">
<br />
Email: <input type="text" name="email" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Action_cookie_main.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">
<%

   Cookie username = new Cookie("username",
                         request.getParameter("username"));
   Cookie email = new Cookie("email",
                         request.getParameter("email"));


   username.setMaxAge(60*60*10);
   email.setMaxAge(60*60*10);

   // Add both the cookies in the response header.
   response.addCookie( username );
   response.addCookie( email );
%>


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

<b>Username:</b>
   <%= request.getParameter("username")%>
<b>Email:</b>
   <%= request.getParameter("email")%>

</body>
</html>
```

JSP Form Processing

Forms are the common method in web processing. We need to send information to the web server and that information.

There are two commonly used methods to send and get back information to the web server.

1. GET Method:

- This is the default method to pass information from browser to web server.
- It sends the encoded information separated by ?character appended to URL page.
- It also has a size limitation, and we can only send 1024 characters in the request.
- We should avoid sending password and sensitive information through GET method.

2. POST Method:

- Post method is a most reliable method of sending information to the server.
- It sends information as separate message.
- It sends as text string after ?in the URL.
- It is commonly used to send information which are sensitive.

JSP handles form data processing by using following methods:

1. getParameter():It is used to get the value of the form parameter.
2. getParameterValues():It is used to return the multiple values of the parameters.
3. getParameterNames()It is used to get the names of parameters.
4. getInputStream()It is used to read the binary data sent by the client

Action_form.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> Form</title>
</head>
<body>
<form action="action_form_process.jsp" method="GET">
UserName: <input type="text" name="username">
<br />
Password: <input type="text" name="password" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Action_form_process.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>

<h1>Form Processing</h1>

<p><b>Welcome User:</b>
   <%= request.getParameter("username")%>
</p>

</body>
</html>
```

JSP Database Connection: Select, Insert, Update & Delete

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.

1. Create the database
2. Create the table
3. Insert the record

JSP Operations: Insert, Update, Delete, Select

Select

The Select operation is used to select the records from the table.

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ 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>Abc Database JSP1</title>
</head>
```

```
<body>

 <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/AbcTest"
     user="Abcroot"  password="Abc"/>

<sql:query dataSource="${snapshot}" var="result">
SELECT * from Abc_test;
</sql:query>

<table>
<tr>
   <th>Abc ID</th>
   <th>Name</th>

</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
   <td><c:out value="${row.emp_id}"/></td>
   <td><c:out value="${row.emp_name}"/></td>

</tr>
</c:forEach>
</table>

</body>
</html>
```

Insert

Insert operator is used to insert the records into the database.

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="Abccore"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="Abcsql"%>
<%@ 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>Abc Database JSP1</title>
</head>
<body>

 <Abcsql:setDataSource var="Abc" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/AbcTest"
     user="Abcroot"  password="Abc"/>

     <Abcsql:update dataSource="${Abc}" var="Abcvar">
INSERT INTO Abc_test VALUES (3, 'emp emp3');
</Abcsql:update>



</body>
</html>
```

Delete

This is delete operation where we delete the records from the table Abc_test.

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="Abccore"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="Abcsql"%>
<%@ 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>Abc Database JSP1</title>
</head>
<body>

 <Abcsql:setDataSource var="Abc" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/AbcTest"
     user="Abcroot"  password="Abc"/>
     <Abccore:set var="Abcid" value="3"/>
     <Abcsql:update dataSource="${Abc}" var="Abcvar">
DELETE FROM Abc_test WHERE emp_id = ?
  <Abcsql:param value="${Abcid}" />
</Abcsql:update>



</body>
</html>
```

Update

The update is used to edit the records in the table.

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="Abccore"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="Abcsql"%>
<%@ 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>Abc Database JSP1</title>
</head>
<body>

<Abcsql:setDataSource var="Abc" driver="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost/AbcTest"
     user="Abcroot"  password="Abc"/>
     <Abccore:set var="Abcid" value="2"/>
     <Abcsql:update dataSource="${Abc}" var="Abcvar">
```

```
UPDATE Abc_test SET emp_name='emp Abc99'
  <Abcsql:param value="${Abcid}" />
</Abcsql:update>



</body>
</html>
```

JSP Program: Registration & Login Form

JSP File Upload & File Download

Example: Using Action

Action_file.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>Abc File</title>
</head>
<body>
<a>Abc File Upload:</a>
Select file: <br />
<form action="action_file_upload.jsp" method="post"
                        enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>
```

Action_file_upload.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@ page import="java.io.*,java.util.*, javax.servlet.*" %>
<%@ page import="javax.servlet.http.*" %>
<%@ page import="org.apache.commons.fileupload.*" %>
<%@ page import="org.apache.commons.fileupload.disk.*" %>
<%@ page import="org.apache.commons.fileupload.servlet.*" %>
<%@ page import="org.apache.commons.io.output.*" %>
<!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>Abc File Upload</title>
</head>
<body>
<%
```

```
   File file ;
   int maxFileSize = 5000 * 1024;
   int maxMemSize = 5000 * 1024;
   String filePath = "E:/Abc99/data";

   String contentType = request.getContentType();
   if ((contentType.indexOf("multipart/form-data") >= 0)) {

      DiskFileItemFactory factory = new DiskFileItemFactory();
      factory.setSizeThreshold(maxMemSize);
      factory.setRepository(new File("c:\\temp"));
      ServletFileUpload upload = new ServletFileUpload(factory);
      upload.setSizeMax( maxFileSize );
      try{
         List fileItems = upload.parseRequest(request);
         Iterator i = fileItems.iterator();
         out.println("<html>");
         out.println("<body>");
         while ( i.hasNext () )
         {
            FileItem fi = (FileItem)i.next();
            if ( !fi.isFormField () )  {
                String fieldName = fi.getFieldName();
                String fileName = fi.getName();
                boolean isInMemory = fi.isInMemory();
                long sizeInBytes = fi.getSize();
                file = new File( filePath + "yourFileName") ;
                fi.write( file ) ;
                out.println("Uploaded Filename: " + filePath + fileName +
"<br>");
            }
         }
         out.println("</body>");
         out.println("</html>");
      }catch(Exception ex) {
         System.out.println(ex);
      }
   }else{
      out.println("<html>");
      out.println("<body>");
      out.println("<p>No file uploaded</p>");
      out.println("</body>");
      out.println("</html>");
   }
%>
</body>
</html>
```

Example: Using JSP operations

Uploading_1.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>Abc Uploading File</title>
```

```
</head>
<body>
File: <br />
<form action="Abc_upload" method="post"
                        enctype="multipart/form-data">
<input type="file" name="Abc_file" size="50" />
<br />
<input type="submit" value="Upload" />
</form>
</body>
</html>
```

Abc_upload.java

```
package demotest;

import java.io.File;
import java.io.IOException;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

public class Abc_upload extends HttpServlet {
        private static final long serialVersionUID = 1L;

    public Abc_upload() {
        super();
        // TODO Auto-generated constructor stub
    }
        protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                        if(ServletFileUpload.isMultipartContent(request)){
                            try {
                                List <FileItem> multiparts = new
ServletFileUpload(new DiskFileItemFactory()).parseRequest(request);
                                for(FileItem item : multiparts){
                                    if(!item.isFormField()){
                                        String name = new
File(item.getName()).getName();
                                        item.write( new
File("c:/Abc/upload" + File.separator + name));
                                    }
                                }
                               //File uploaded successfully
                               request.setAttribute("Abcmessage", "File
Uploaded Successfully");
                            } catch (Exception ex) {
                               request.setAttribute("Abcmessage", "File
Upload Failed due to " + ex);
                            }
                        }else{
```

```
                                request.setAttribute("Abcmessage","No File
found");
 }

request.getRequestDispatcher("/result.jsp").forward(request, response);

                    }

}
```

Result.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>Abc Result</title>
</head>
<body>
<% String msg = (String)request.getAttribute("message");
   out.println(msg);
%>
</body>
</html>
```

Downloading File:

Downloading_1.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>Downloading Abc Example</title>
</head>
<body>
Abc Downloading File<a href="Abc_download">Download here!!!</a>
</body>
</html>
```

Abc_download.java

```
package demotest;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Abc_download
 */
public class Abc_download extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
		String Abcfile = "test.txt";
		String Abcpath = "c:/Abc/upload/";
		response.setContentType("APPLICATION/OCTET-STREAM");
		response.setHeader("Content-Disposition", "attachment;
filename=\""
				+ Abcfile + "\"");

		FileInputStream fileInputStream = new
FileInputStream(Abcpath
				+ Abcfile);

		int i;
		while ((i = fileInputStream.read()) != -1) {
			out.write(i);
		}
		fileInputStream.close();
		out.close();
	}


	/**
	 * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

1. **Build a Todo web application using Servlet + JSP + JDBC + MySQL database.**

2. **Create a web application on Managing books in library Technology used: Servlet + JSP + JDBC + MySQL + Tomcat**
3. **Create Login and Registration page**
4. **Apply Validation on Login and Registration page**
5. **Forgot Password Functionality on login page**