UNIT - IV

INTRODUCTION TO JSP

The Anatomy Of a JSP Page

A JSP page is simply a regular web page with JSP elements for generating the parts of the page that differ for each request, as shown in figure below.

```
<%@ page language = "java" contentType =
                        "text/html" %>          ] JSP element
<html>
 <body> bgcolor = "white">                      ] — template text

 <jsp:useBean
   id = "userInfo"
        class = "com.ora.jsp.beans.userinfo.
                        UserInfoBean">
 <jsp:setProperty name = "userInfo" property =
                        "*"/>
 </jsp:useBean>                                 ] JSP element

The following information was saved:            ] — template text
<ul>
  <li>User Name :

  <jsp:getProperty name = "userInfo"
                property = "userName"/>         ] — JSP element

  <li> Email Address :                          ] — template text

  <jsp:getProperty name = "userInfo"
                property = "emailAddr"/>        ] — JSP element
```

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

Everything in the page that is not a JSP element is called template text. Template text can really be any text : HTML, WML, XML or even plain text. Since HTML is by far the most common web page language in use today, most of the descriptions and examples in this book are HTML - based, but keep in mind that JSP has no dependency on HTML ; it can be used with any markup language. Template text is always passed straight through to the browser. When a JSP page request is processed, the template text and the dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

JSP Processing :

JSP pages can be processed using JSP container only. Following are the steps that need to be followed while processing the request for JSP page –

1) Client makes a request for required JSP page to the server. The server must have JSP container so that JSP request can be processed. For instance : Let the client makes a request for xyz.jsp page.

2) On receiving this request the JSP container searches and then reads the desired JSP page. Then this JSP page is straight away converted to corresponding servlet. Basically any JSP page is a combination of template text and JSP element. Every template text is translated into corresponding println statement.

For instance :

```
<html>
<head>
<title>Demo</title>
...
```

→

```
out.println("<html>");
out.println("<head>");
out.println("<title>Demo
                </title>");
...
```

Every JSP element is converted into corresponding Java code. This phase is called translation phase. The output of translation phase is a servlet.

For example : our xyz.jsp gets converted into xyzServlet.java

3) This servlet is compiled to generate the servlet class file. This phase is called request processing phase.

4) The JSP container thus executes the servlet class file.

5) A requested page is then returned to the client as a response.

Declarations

The JSP page that we write is turned into class definition. So when we declare a variable or method in JSP inside Declaration Tag. We can declare static member, instance variable and methods inside Declaration Tag.

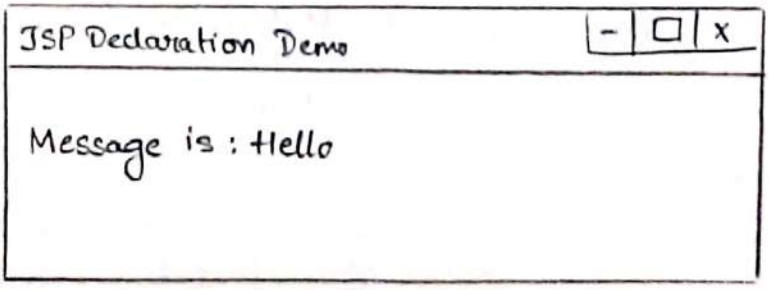
Syntax of Declaration Tag:

```
<%! Declaration code %>
```

Example:

```
<html>
  <head>
    <title> JSP Declaration Demo </title>
  </head>
  <%!
     String msg = "Hello";
  %>
  <body>
    Message is:
      <% out.println(msg); %>
  </body>
</html>
```

Output:



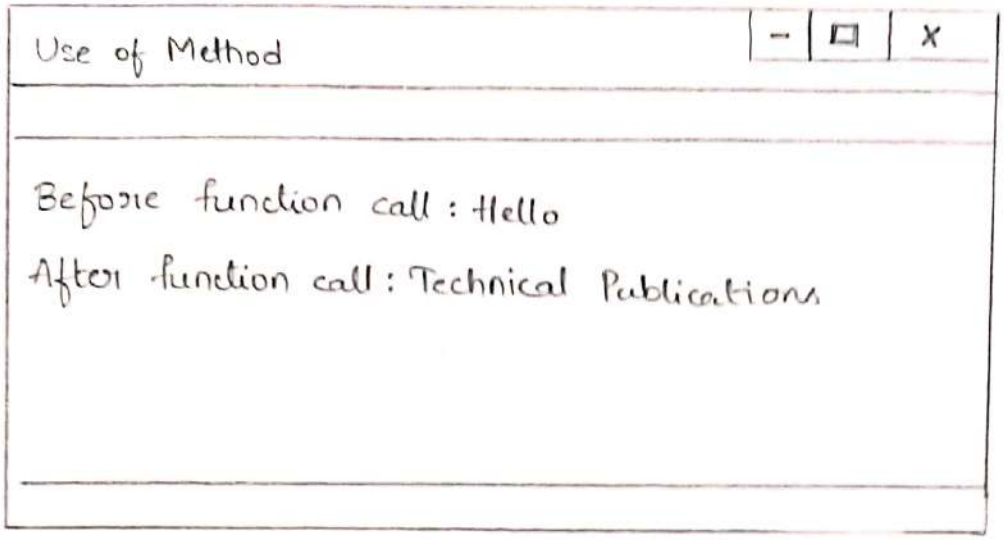
The above JSP code contains the declaration within <%! %> tag.

* We can declare a function of a method in JSP just similar to variable. Following JSP example illustrates the use of function declaration and definition.

MethodDemo.jsp

```
<%@ page language = "java" contentType =
                                "text/html" %>
    <%!
        String msg = "Hello";%>
    %>
    <%! public String MyFunction (String msg)
    {
        return msg;
    }
    %>
<html>
  <head>
    <title>Use of Method </title>
  </head>
  <body>
    <% out.println("Before function call : " +
                                msg ); %>
    <br/>
    After function call : <%= MyFunction ("Technical
                                Publications") %>
  </body>
</html>
```

Output :



JSP - Directives

Directives in JSP provide directions and instructions to the container, telling it how to handle certain aspects of the JSP processing.

* A JSP directive affects the overall structure of the servlet class. It usually has the following form -

```
<%@ directive attribute = "value" %>
```

* Directives can have a number of attributes which you can list down as key-value pairs and seperated by commas.

* The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

S.No.	Directive & Description
1.	<%@ page ... %> Defines page-dependent attributes, such as scripting language, error page and buffering requirements.
2.	<%@ include ... %> Includes a file during the translation phase.
3.	<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

JSP - The page Directive :

The page directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Syntax :

<%@ page attribute = "value" %>

Attributes :

S.No.	Attribute & purpose.
1.	buffer Specifies a buffering model for the output stream.
2.	autoFlush controls the behavior of the servlet output buffer.
3.	contentType Defines the character encoding scheme.
4.	extends Specifies a superclass that the generated servlet must extend
5.	language Defines the programming language used in the JSP page.
6.	session Specifies whether or not the JSP page participates in HTTP sessions

The Include Directive :

The 'include' directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the 'include' directives anywhere in your JSP page.

* The general usage form of this directive is as follows -

```
<%@ include file = "relative url">
```

* The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

* You can write the XML equivalent of the above syntax as follows -

```
<jsp:directive.include file = "relative url" />
```

The taglib Directive :

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

* The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides

means for identifying the custom tags in your JSP page.

* The taglib directive follows the syntax given below –

```
<%@ taglib uri="uri" prefix="prefixOfTag">
```

* Here, the uri attribute value resolves to a location the container understands and the prefix attribute informs a container what bits of markup are custom actions

```
<jsp:directive.taglib uri="uri" prefix =
                      "prefixOfTag" />
```

Expressions

The expression tag is used to represent the expression in JSP page.

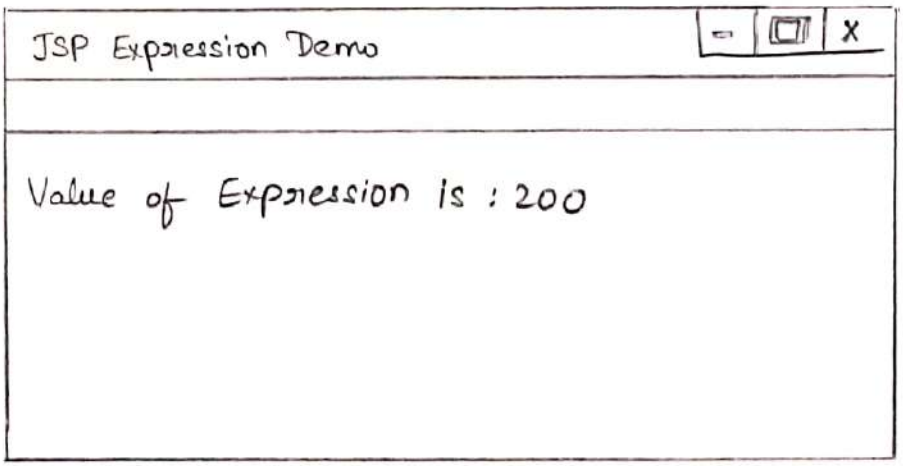
Syntax of writing expression:

```
<%= Java Expression %>
```

Example:

```
<html>
  <head>
    <title> JSP Expression Demo </title>
  </head>
  <body>
    Value of Expression is :
    <%= (10*20) %>
  </body>
</html>
```

Output:



* Code Snippets

The code that appears between the <%. and %> delimiters is called a scriptlet. Scriptlets are nothing but java code enclosed within <%. and %.> tags

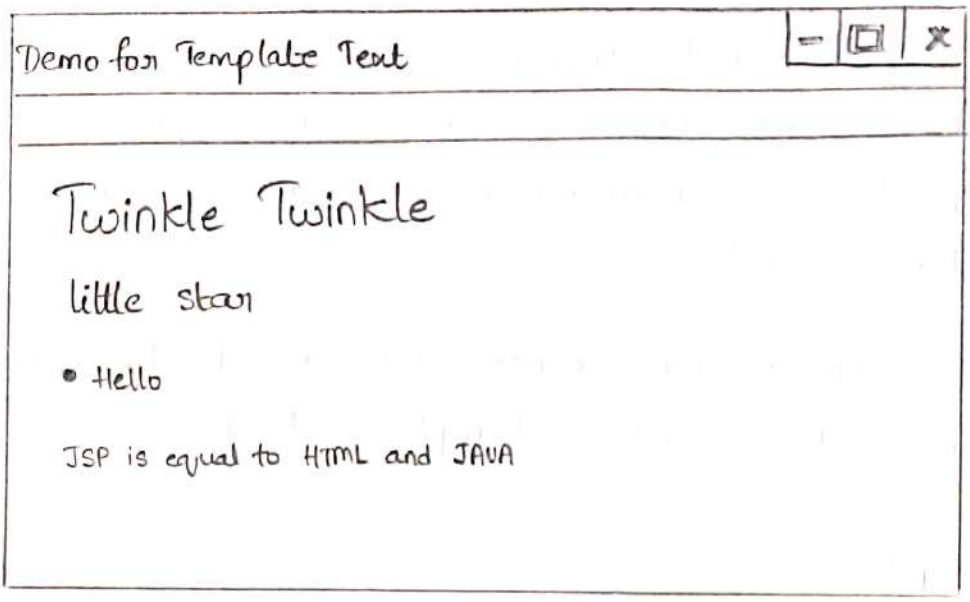
* Every Thing other than a JSP statement in the JSP is called template text.

Example :

TemplateText.jsp

```
<%@ page language="java" contentType =
                        "text/html" %>
<html>
  <head>
    <title>Demo for Template Text </title>
  </head>
  <body bgcolor="gray">
    <h1> Twinkle Twinkle </h1>
    <h2> little star </h2>
    <li> hello </li>
    <p>
      <% out.println("JSP is equal to HTML
                          and JAVA"); %>
    </p>
  </body>
</html>
```

Output :



JSP – Implicit Objects:

The Implicit Objects are the Java objects that the JSP container makes available to the developers in each page and the developer can call them directly without being explicitly declared.

* Following table lists out the nine Implicit objects that JSP supports –

• request	This is the HttpServletRequest object associated with the request
• response	This is the HttpServletResponse object associated with the response to the client
• out	This is the PrintWriter object used to send output to the client
• session	This is the HttpSession object associated with the request
• application	This is the ServletContext object associated with the application context
• config	This is the ServletConfig object associated with the page.
• page Context	This encapsulates use of server-specific features like higher performance JspWriters
• page	This is simply a synonym for this, & is used to call the methods defined by the translated servlet class

• Exception	The Exception object allows the exception data to be accessed by designated JSP.

Using Beans in JSP Pages:

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

* Following are the unique characteristics that distinguish a JavaBean from other Java classes –

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the Serializable interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

JavaBeans Properties:

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

* A JavaBean property may be read, write, read only or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class –

S.No.	Method & Description
1.	getPropertyName() For example, if property name is firstName, your method name would be getFirstName() to read that property. This method is called accessor.
2.	setPropertyName() For example, if property name is firstName, your method name would be setFirstName() to write that property. This method is called mutator.

* A read-only attribute will have only a getPropertyName() method, and a write-only attribute will have only a setPropertyName() method.

// Example

```
public class StudentBean implements Serializable
{
        String Rno;
        String Name;
        public void setRno(String rno)
        {
            this.Rno = rno;
        }
```

```
    public void getRno(String)
    {
        return Rno;
    }
    public void setName(String name)
    {
       this.Name = name;
    }
    public void getName( )
    {
       return Name;
    }
}
```

* There are various scopes using which the bean can be used in JSP page.

1) Page scope:

The bean object gets disappeared as soon the current page gets discarded. The default scope for a bean in jsp page is a page scope

2) Request scope:

The bean object remains in existence as long as the request object is present.

3) Session scope:

A session can be defined as a specific period of time, the user spends browsing the site.

4) Application scope:

During application scope the bean will get stored to ServletContext. Hence particular bean is available to all the servlets in the same web application.

* Application scope is the broadest scope provided by Jsp and it should be used only when it is necessary.

Example

xyz.html

```
<html>
  <body>
    <form action = "abc.jsp" method = "post">
    <input type = "text" name = "text1" id = "text1">
    <input type = "password" name = "password"
                                id = "text2">
    <input type = "button" name = "submit"
                          value = "submit">
    </form>
  </body>
</html>
```

abc.jsp

```
<html>
 <body>
  <jsp:useBean id = "login" class = "ValidateBean" />
  <jsp:setProperty name = "login" property = "user" />
  <jsp:setProperty name = "login" property = "pass" />
  You entered username as : <jsp:getProperty
                name = "login" property = "user" />
  You entered password as : <jsp:getProperty
                name = "login" property = "pass" />
  You are a <% login.validate ("naveen", "cse")
                %> user <br>
```

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

Validate Bean . java

```
class ValidateBean implements Serializable
{
    String Name;
    String Pass;
    public void setName(String name)
    {
      this.Name = name;
    }
    public void getName()
    {
      return Name;
    }
    public void setPass(String pass)
    {
      this.Pass = pass;
    }
    public void getPass()
    {
      return Pass;
    }
    public String Validate(String s1, String s2)
    {
      if (s1.equals(name)&& s2.equals(pass))
          return valid;
      else
          return invalid;
    }
}
```

Using Cookies

Cookies are the small text files that are stored in the client's computer.

* These are basically used to keep track of the users who browse the web. The information stored in the cookie is generally name, age, id, city and so on.

* The server script sends a set of cookies to the browser. The browser stores this information on the local machine and makes use of this information next time when the browser is browsing the web.

* Cookies are usually set in HTTP header.

* Various methods used in handling the cookies are -

1) Create Cookie
2) Read Cookie
3) Delete Cookie

1) Create cookie:

step 1 : In JSP the cookie is created using the constructor named Cookie. It requires two parameters — name and value.

Example —

```
Cookie cookie = new Cookie("name", "value");
```

Step 2 : Then we can set the validity period for the cookie using the method setMaxAge. For example to set the cookie alive for 24 hr we will write the code as

```
cookie.setMaxAge(60*60*24);
```

Step 3 : Now our cookie is ready to send over. We can add the cookie in HTTP response header as follows

```
response.addCookie(cookie);
```

2) Read Cookie :

Step 1 : First the cookie is retrieved using getCookies() method.

```
Cookie[] cookies = request.getCookies();
```

Step 2 : Then using getName() and getValue() methods the cookies are read.

3) Delete cookie :

Step 1 : Read the already created cookie and store it in cookie object

```
Cookie cookie = new Cookie("name", "");
```

Step 2: Then set its period of existence as 0 by setMaxAge method. This means that cookie is actually deleted.

```
cookie.setMaxAge(0);
cookie.setValue("");
```

Step 3: Add this cookie back to response header.

```
response.addCookie(cookie);
```

Cookie example

```
<body>
<%
   String  str1 = request.getParameter("item");
   String  str2 = request.getParameter("qty");
   String  str3 = request.getParameter("add");
   String  str4 = request.getParameter("list");
   if (str3 != null)
   {
      Cookie  c1 = new  Cookie (str1, str2);
      response.addCookie(c1);
      response.sendRedirect("index.html");

   }
   else if (str4 != null)
   {
      cookie  clientCookies[] = request.getCookies();
      for (int i=0; i<clientCookies.length; i++)
      {
         out.print("<B>" + clientCookies[i].
                  getName()+":"+ clientCookies[i].
                  getValue() + "</B><BR>");

      }
   }
%>
</body>
```

* Session Handling in JSP

If we use a request scope and try to access the data over multiple pages, then same data can be shared by multiple pages. But sometimes we need to use same data for multiple requests. For example in Hospital management system, the patient information is entered initially only. That patient may undergo through various tests or operations. It is then not necessary for him to enter the same information over again and again. The same set of information is used by various operations in the hospital management system. In such a case the session scope is used.

* HTTP is a request-response protocol. That means when user wants to access some web page, the web browser makes request to server and server returns that page as a response.

* But at the same time HTTP is also called as a stateless protocol. That means when browser sends a request to the server, server processes it and sends the response to the browser and does not remember anything about the request. So when browser sends the same request to the server, server takes it as a new request process. So, it is required that server should keep track of the user or request made by the user. To solve this problem there are three methods used -

1. Use of Cookies
2. Embedding hidden fields in an HTML form
3. Sending URL string in response body.

* For sending information to and fro between browser and server, usually an ID is used. This ID is basically a session-ID. Thus session-ID is passed between the browser and server while processing the information. This method of keeping track of all the information between server and browser using session-ID is called session tracking.

* # Connecting to database in JSP

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

a) Register the driver class
b) Creating connection
c) Creating statement
d) Executing queries
e) closing connection

a) Register the driver class:

The forName() method of class is used to register the driver class. This method is used to dynamically load the driver class

Syntax:

```
public static void forName(String className)
                    throws ClassNotFoundException
```

Example:

```
Class.forName("com.mysql.jdbc.Driver");
```

b) Create the connection object:

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax:

public static Connection getConnection(String url, String name, String password)

Example:

Connection con = DriverManager.getConnection (url, user, password);

c) Create a Statement Object:

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax:

public Statement createStatement() throws SQLException

Example:

Statement stmt = con.createStatement();

d) Execute the query:

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of the table.

Syntax:

```
public ResultSet executeQuery(String sql)
                    throws SQLException.
```

Example:

```
ResultSet rs = stmt.executeQuery("select * from emp");
```

e) close the connection object:

By closing connection, object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax:

```
public void close() throws SQLException.
```

Example:

```
con.close();
```