UNIT-IV.

INTRODUCTION TO JSP (JAVA SERVER PAGES).

Java Server pages (JSP) is a technology that helps software developers to create dynamic web pages based on HTML, XML (or) other document types. Using JSP, one can easily seperate presentation logic 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.

JSP's are released in the year 1999 by Sun MicroSystems. JSP is similar to the java programming language. To deploy and run Java Server pages, a compatible web server with a servlet container, such as Apache Tomcat (or) Jetty is required.

JSP pages are easier to maintain then a servlet. JSP pages are opposite of servlets as servlet and 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.

Servlet Vs JSP.

Servlets:

1. Servlets are java programs which supports HTML tags too.
2. Generally used for developing business layer of an enterpri-se application.
3. Servlets are created and maintained by Java developers.

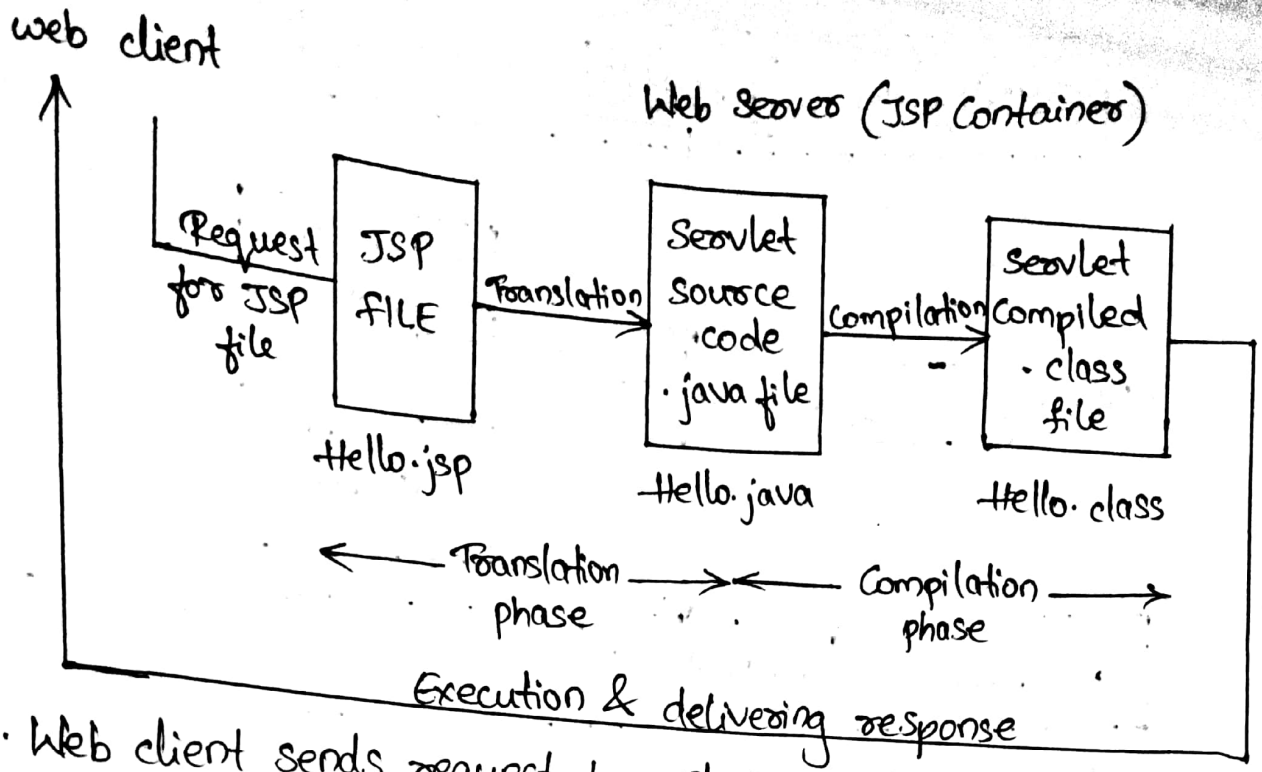
JSP:

1. JSP program is a HTML code which supports java statements too.
2. Used for developing presentation layer of an enterprise application.
3. frequently used for desiging websites and used for web designers.

Advantages of JSP.

1. JSP is useful for server side programming.
2. JSP has all the advantages that a servlet has, like:
   Better performance than (G) Built in session features, it also inherits the features of java technology like - multithreading, exception handling, Database Connectivity etc.
3. JSP enables the separation of content generation from content presentation, which makes it more flexible.
4. With the JSP, it is now easy for web designers to show case the information what is needed.
5. Web application programmers can concentrate on how to process / build the information.
6. High performance and scalability.
7. JSP is build on Java technology, so it is platform independent.

JSP Architecture:



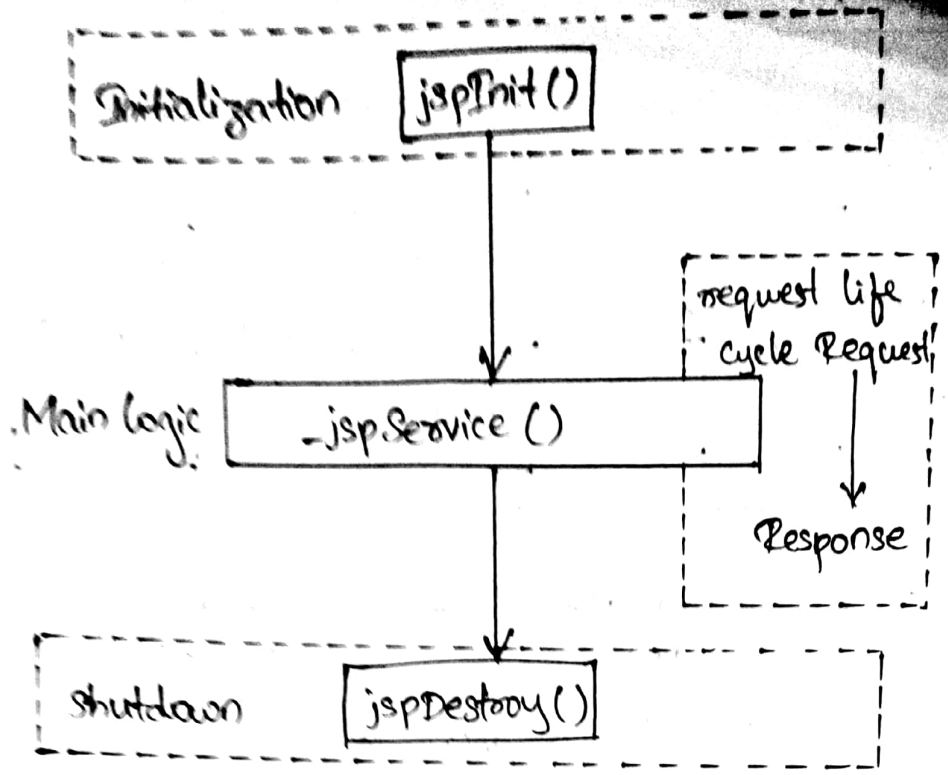
1. Web client sends request to web server for a JSP page (extension .jsp).
2. As per request, the web server (here after called as JSP Container) loads the page.
3. JSP container converts (translates) JSP file into servlet source code file (with .java extension). This is known as translation.
4. The translated .java file of servlet is compiled that result to servlet .class file. This is compilation.
5. The .class file of servlet is executed & output of execution is sent to client as response.

Now lets see the life cycle of a Java Server Page (JSP)

JSP life Cycle:

A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to servle life cycle with a additional step which is required to compile a JSP into servlet.

The major phases of JSP life cycle are very similar to servlet life cycle and they are as follows:



JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the jsp has been modified since it was last compiled, the JSP engine compiles the page.

JSP Initialization:

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method.

```
public void jspInit() {
    // Initialization code....
}
```

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, & create lookup tables in jspInit method.

JSP Execution:

This phase of JSP life cycle represents all interactions with requests until JSP is destroyed. Whenever a browser requests a JSP & page has been loaded & initialized, the JSP engine invokes the _jspService() method in JSP.

The _jspService() method takes HttpServletRequest and HttpServletResponse as its parameters.

```
void _jspService (HttpServletRequest request,
                  HttpServletResponse response)
{
  // service handling code...
}
```

The _jspService() method of JSP is invoked once per request and is responsible for generating once response for that request & this method is also responsible for generation responses to all HTTP methods i.e. GET, POST, DELETE etc.

JSP cleanUp:

The destruction phase of JSP life cycle represents when a JSP is being removed from use by a container.

The jspDestroy() method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connecting (or) closing open files.

The jspDestroy() method has following form

```
public void jspDestroy()
{
  // your cleanup code goes here.
}
```

SIMPLE JSP CODE WRITING AND EXECUTING

for executing JSP code we must have :

1. JDK installed.
2. Apache Tomcat must be installed.

#: 1. Open text editor and type following code

hello.jsp

```
<%@ page language="java" contentType="text/html"%>
<%@ page import="java.util.*"%>
<html>
<!... This is basic JSP page --->
<title> JSP Demo </title>
<body>
<%. --- Displaying message on browser---%>
<% out.println ("This is my first JSP page!"); %>
</body>
</html>
```

Create a separate directory at the path

c:\your-tomcat-directory\webapps\jsp-examples & store above code in newly created directory.

I have created a directory named HelloDemo in which I have stored above program by name hello.jsp. Note that while saving file using Notepad editor with extension .jsp, you must select all files option. If you donnot do that then file may get saved as hello.jsp.txt because that default extension for notepad files is .txt.

Step-2:

Start Tomcat web server by typing command startup at command poompt, or by clicking startup file.

Step-3:

Open some web browsers like firefox or IE. Type path for JSP page using prefix http://localhost:8080. Note localhost is default DNS for Tomcat webserver.

http://localhost:8080/foldername/your pgm. extension i.e

http://localhost:8080/examples/hello.jsp

Components of JSP / Anatomy of JSP.

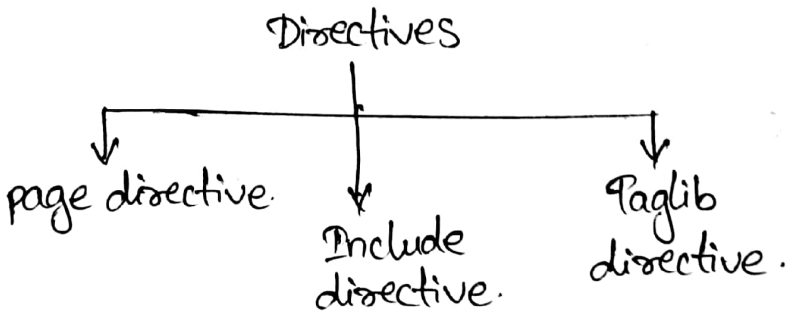
JSP page is built using components such as:

1. Directives.
2. Declarations. } Scripting Elements.
3. Expressions. } Scripting Elements.
4. Scriplets. } Scripting Elements.
5. Action tags.
6. Custom Tags.

1. Directives: The JSP directives are messages that tells the web container how to translate JSP page into the corresponding servlet

Syntax:

<%@ directive attribute = "value" %>



i) Page Directive: The page directive defines attributes that apply to an entire JSP page.

Syntax: <%@ page attributes = "value" %>

Ex: <%@ page language = "java" %>

The various attributes of page directive are:

a. import.
b. Content Type.

c. extends.
d. info
e. buffer
f. language.
g. isELIgnored.
h. isThreadSafe.
i. autoflush.
j. Session.
k. pageEncoding.
l. Errorpage.
m. isErrorPage.

a. import: The import attribute is used to import class, interface (or) all the members of package. It is similar to import keyword in java class (or) interface.

Ex: <%@ page import = "java.io.*" %>

b. Content Type: The Content Type attribute defines the MIME (Multi purpose Internet Mail Extension) type of HTTP response. The default value is "text/html; charset = ISO-8859-1".

Ex: <%@ page Content Type = "text/html" %>

c. Extends: The extends attribute defines the parent class that will be inherited by the generated servlet. It is rarely used.

Ex: <%@ page import = "java.sql.*, mypackage.myclass" %>

d. info: This attribute simply sets the information of JSP page which is retrieved later by using getServletInfo() method of servlet interface.

e. buffer: The buffer attribute sets the buffer size in kilobytes to handle output generated by JSP page. The default size of buffer is 8kb.

Ex: <%@ page buffer = "16 kb" %>

f. language: The language attribute specifies the scripting language used in JSP page. The default value is "java".

Ex: <%@ page language = "java" %>

g. isELIgnored: We can ignore the Expression language (EL) in jsp by isELIgnored attribute. By default its value is false, i.e Expression language is enabled by default.

Ex: <%@ page isELIgnored = "true" %>

h. Session: When the value in true, session data is available to JSP page otherwise not. By default, this value is true.

Ex: <%@ page language = "java" session = "true" %>

i. Error page: Error page is used to handle the un-handled exception in page. Which is not handled by exceptions in program.

Ex: <%@ page language = "java" session = "true"
error page = "error.jsp" %>

ii) include directive:

"include" is used to include a file in the JSP page. This directive is to include HTML, JSP or Servlet file into JSP file. This is a static inclusion of file i.e it will be included into JSP file at the time of compilation and once the JSP file is compiled any changes in included file, it will not be reflected.

Ex: <%@ include file = "/header.jsp" %>

iii) taglib directive:

taglib is used to include the custom tags in JSP pages.

Ex: <%@ taglib uri = "tlds/taglib.tld" prefix = "mytag" %>

2. JSP Declaratives:

The JSP declaration tag is used to declare field and methods.

The code written inside the JSP declaration tag is placed outside the Service() method of auto generated servlet. So, it does not get memory at each request.

Syntax:

```
<%!
//Declare all the variables here
%>
```

JSP declaration beings with <%! and ends with %>. We can embed any amount of Java code in JSP Declaration. Variables & functions defined in declaratives are class level & can be used anywhere in JSP page.

Ex:

```
<%@ page ContentType = "text/html" %>
<html>
<body>
<%!
  int cnt = 0;
  Private int getCount() {
//increment cnt & return the value
  cnt++;
  return cnt;
}
%>
<p> Values of cnt are : </p>
<p><%= getCount() %></p>
<p><%= getCount() %></p>
<p><%= getCount() %></p>
<p><%= getCount() %></p>
<p><%= getCount() %></p>
<p><%= getCount() %></p>
```

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

o/p:- Values of cnt are:

1
2
3
4
5
6.

3. Expressions:

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

Syntax:

```
<%= "Any thing" %>
```

JSP Expressions starts with <%= and ends with %>. Between, these, any no. of expressions can be embedded. The result of expression is converted to string to get displayed.

Ex:
```
<html>
<body>
<%= "Hello World" %>
</body>
</html>
```

CODE SNIPPETS.

#Templatetext.jsp

```
<%@ page language = "java" ContentType = "text/html" %>
<html>
<head>
<title>Demo</title>
</head>
<body bgcolor = "gray">
```

```
<h1>Hi</h1>
<h2>Hello</h2>
<p>
 <%. out. print ("JSP is equal to HTML & JAVA"); %>
</p>
</body>
</html>
```

Scriptlet demo.jsp.

```
<%@ page language = "java" content Type = "text/html" %>
<html>
 <head>
   <title> Introduction </title>
 </head>
 </body>
   <Stoong>
    <%. out.println ("value i is:"); %>
    <%. for (int i=1; i<5; i++) {
    out. print ("<br/>");
    if (i%.2==0)
    {
      out. print ("<i/>");
      out.println ("Even value:"+i);
    }
     else {
        out.print ("<i/>");
        out.println ("Odd value:"+i); }
     }%>
   </Stoong>
  </body>
  </html>
```

Implicit Objects:

The implicit objects are predefined variables used to acces request and application data. These objects are used by scripting elements.

Variable Name.	class/Interface Name.	Meaning	Sample Methods.
1. application	javax.servlet. ServletContext.	This object provides resources shared within a web application.	log() getServerInfo().
2. Config	javax.servlet. ServletConfig	It helps in passing information to servlet or JSP page during initialization.	getInitparameter getServletName(
3. request	javax.servlet.http. HttpServletRequest	It provides the methods for accessing information made by current request.	getContentLength getLocalAddr(getServerName
4. Out.	javax.servlet.jsp. JSPWriter.	It provids the methods related to Info.	clear() newline().
5. response.	javax.servlet.http. HttpServletResponse.	It provides methods related to adding Cookies, session or setting header.	addCookie() addHeader(). flushBuffer() getContentType() SetContentType()

6. page	java.lang.Object	This variable is assigned to instance of JSP implementation class. It is rarely used variable.	
7. pageContext	javax.servlet.jsp.PageContext.	It provides access to several JSP page attributes	getPage() getRequest() getResponse() getSession()
8. Session	javax.servlet.http.HttpSession.	This variable is used to access the current client's session.	getId() getCreationTime().
9. exception.	java.lang.Throwable.	This object is for handling error pages and contain information about runtime error.	

Using Beans in JSP Pages:

Java beans are reuseable components. We can use simple Java bean in JSP. This helps us in keeping the business logic separate from presentation logic. Beans are used in JSP pages as instance of class. We must specify the scope of the bean in JSP page. Here scope of the bean means the range time span of bean for its existance in JSP. When the bean is present in particular scope its id is also available in that scope.

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

Page Scope: The bean object gets disappeared as soon as current page gets discarded. The default scope for a bean in JSP page is a Page Scope.

2. Request Scope: It specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.

3. Session Scope: It specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.

4. Application Scope: It specifies that you can use this bean from any JSP page in the same application. During application scope the bean will get stored to ServletContext. It has wider scope than session.

The <jsp:useBean> action is used to specify the scope of the bean.

Syntax:

<jsp:useBean id="instanceName" scope="page/request/session/application"
class="packageName.className" type="packageName.className"
beanName="packageName.className/<%= expression >" >
</jsp:useBean>.

→ id: is used to identify the bean in specified scope.

→ class: Instantiates the specified bean class but it must has no-arg (or) no Constructor & must not be abstract.

→ type: provides the bean a data type if bean already exists in scope. It is mainly used with class (or) beanName attribute. If you use it without class (or) Name, no bean is instantiated.

→ beanName: Instantiates the bean using java.beans.Beans.instantiate() method.

following is an example code in which a bean textdemo.java used. In this file the class Textdemo is written. The purpose of this bean is simply to get (or) set some message. Initially the variable message is initialized to some string say "Asst proof".

Step-1: Write a simple java program for creating simple bean. This bean is used to set property message. Create a folder named textdemo at path-

c:\your-Tomcat-directory\webapps\jsp-examples\WEB-INF\classes, and save the file by name textdemo.java. It can be written as follows.

textdemo.java (Bean pgm):

```
package textdemo;
  public class Textdemo
  {
    public static String msg;
    public Textdemo()
    {
      msg = "Asst proof";
    }
    public String getMsg()
    {
      return msg;
    }
     public void SetMsg(String msg)
     {
       this.msg = msg;
     }
  }
```

Step-2: Now write a simple JSP in which the textdemo bean is invoked. This JSP file can be created in some folder. I have named this folder as MyBean. This folder can be saved at path-

\your-Tomcat-directory\webapps\jsp-examples.

And JSP file can be as follows.

beanJSP.jsp

```
<html>
 <title>use of JavaBean in JSP</title>
 <head>
 <h1>bean and JSP Demo:</h1>
  </head>
 <body>
 <jsp:useBean id="msgbean" class="textdemo.Textdemo"
            scope="session">
 <p>Getting value for msg property from bean</p>
 <strong>
  <jsp:get property name="msgbean" property="msg" ~~value=~~
                                          ~~"Hello path"~~/>
 </strong>
 <jsp:set property name="msgbean" property="msg" value=
                                          "Hello path"/>
 </jsp:useBean>
 <p>setting the value for msg property from JSP</p>
 <strong>
  <jsp:get property name="msgbean" property="msg"/>
 </strong>
 </body>
 </html>
```

Step-3: Open some web browser and type the path for JSP file.

http://localhost:8080/jsp-examples/MyBean/beanJSP.jsp.

and see the output.

USING COOKIES:

Cookies are short pieces of data sent by web servers to the client browser. The cookies are saved by the hard disk of the client in the form of small text file. Cookies help the web servers to identify web users and tracking them in the process. Cookies play very important role in the session tracking.

Various methods used in handling cookies are:

- Create Cookie.
- Read Cookie.
- Delete cookie.

Create Cookie:

Step-1: In JSP cookie is created using constructor named Cookie. It requires two parameters - name & value.

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

Step-2: Then we can set validity period for cookie using method SetMaxAge.

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

Step-3: Now our cookie is ready to send over. we can add cookie in HTTP response.

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

Read Cookie:

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

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

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

Delete cookie:

Step-1: Read the already created cookie & store it in cookie objec

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

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

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

Step-3: Add this cookie back to response header

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

Cookie objects have following methods.

- getComment()
- getMaxAge()
- getName()
- getPath()
- getValue()
- setComment(String)
- setMaxAge(int)
- setPath(String)
- setValue(String).

Ex: Implementation of cookie in JSP

Step-1: Create html file to read values from user.

Input.html:

```
<html>
<body>
<form method="post" action="Create cookie.jsp">
username <input type="text" name="name">
City <input type="text" name="city">
<input type="submit" name="submit" value="submit">
</form>
</body>
</html>
```

Step-2: Create cookie.

Createcookie.jsp:

```
<%@ page language="java" import="java.util.*" %>
<%
String name = request.getparameter("name");
String city = request.getparameter("city");
Cookie namecookie = new Cookie("name", name);
Cookie citycookie = new cookie("city", city);
response.addCookie(namecookie);
response.addCookie(citycookie);
name Cookie.SetMaxAge(60*60*24);
City Cookie.SetMaxAge(60*60*24);
%>
<h3>
<a href="Read Cookie.jsp"> click here to continue </a>
</h3>
```

Step-3: Read Created Cookie

ReadCookie.jsp:

```
<h3> Reading Cookie </h3>
<%>
Cookie[] cookies = request.getCookies();
for(int i=0; i< Cookies.length; i++)
{
    out.println("Cookie_Name"+ cookies[i].getName()+"<br>"
    out.println("Cookie_value"+ Cookies[i].getValue()+"<br>");
}
%>
<a href="Delete cookie.jsp"> click here to delete </a>
```

step-4: for deletion

Delete cookies.jsp:

```
<%
  Cookie name cookie = new Cookie("name", " ");
  name Cookie.setMaxAge(0);
  name Cookie.SetValue(" ");
  response.addCookie(name Cookie);

  Cookie cityCookie = new Cookie("city", " ");
  cityCookie.setMaxAge(0);
  cityCookie.setValue(" ");
  response.addCookie(cityCookie);
%>
<h3>Cookie is Deleted</h3>
<a href="ReadCookie.jsp"> Click here to check deletion </a>
```

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 page 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 same info again & again. The same set of info is used by various operations in hospital management system. In such case session scope is used.

HTTP is a request-response protocol. HTTP is also called as stateless protocol. That means when browser sends request to server, server processes it & sends response to browser & doesnot remember anything about request. So when browser

sends same request to server. Server take it as a new request, process, it once again & then sends it as response to browser. If there is a interactive webapp then it is required that server should keep track of user or request made by user. To solve this we have three methods.

- Use of cookies.
- Embedding hidden fields in HTML form.
- Sending URL string in response.

For sending all state info to & fro between browser & server, usually an ID is used. This ID is basically sessionID. This ID is passed between browser & server while processing info. This method of keeping track of all info between server & browser using Session-ID is called Session Tracking.

Passing Data between JSP Pages using Session Object.

These are some web app in which user moves from one page to another. Then it becomes necessary to keep track of user data. That means user data must be consistent throughout the web app. To bring this consistency of user data between web pages normally there is a use of an implicit object called Session. Using session we can save the data of particular page.

Ex: Step-1:

In this step we will create main page on which we will accept the user name & password.

first-Page.jsp:

```
<%@ page language="java" %>
<html>
<head>
    <title>Registration form</title>
```

```
</head>
<body>
  <form method="post" action="second-page.jsp">
  <strong> UserName </strong>
  <input type="text" name="userName">
  <br><br>
  <strong> password </strong>
  <input type="password" name="password">
  <input type="submit" value="Enter">
  </form>
 </body>
 </html>
```

Step-2:

Then using get parameter method of request object we are retrieving parameter UserName & Password. These retrieved values must be stored in the variable username & password. The using Session object we can set these value to attributes. Then using hyperlink for Third-page.jsp we can move to third page.

Second_Page.jsp:

```
<%@ page language="java" %>
<%
  String username = request.getparameter("UserName");
  String password = request.getparameter("Password");
 %>
<html>
  <head>
    <title> Session Continues </title>
  </head>
  <body>
```

```
<h3>
  <a href="Third-page.jsp"> click to view next session
</h3>                                    </a>
</body>
</html>
```

Third-page.jsp:

```
<%@ page language = "java" %>
<%
  String username = (String) session.getAttribute ("UserName");
  String Password = (String) session.getAttribute ("password");
%>
<html>
   <head>
     <title> End of session </title>
   </head>
   <body>
     <strong> username <%= username %> </strong>
     <br/>
     <strong> your password <%= password %>
   </body>                                    </strong>
</html>
```

Connecting To DATABASE in JSP:

While accessing JSP database from JSP page we should have some DB package installed. In this section we will consider the connectivity of JSP with MySQL DB.

Prerequisits:

- TOMCAT web server.

- MySQL server.
- JDK.

Step-1: Create a database named students in MySQL using following command.

```
mysql> CREATE DATABASE students;
```

Then create a table named students_table in the students database as follows:

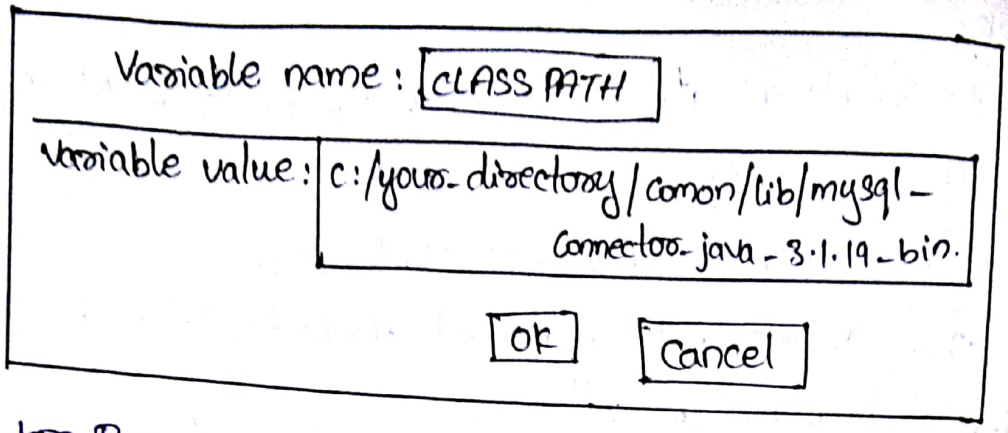
```
mysql> use students;
Database changed
mysql> CREATE TABLE students_table (roll_no INT(4) NOT
                                    NULL AUTO_INCREMENT)
-> name VARCHAR(50) NOT NULL,
-> address VARCHAR(50) NOT NULL,
-> phone VARCHAR(15) NOT NULL,
-> PRIMARY KEY (roll_no)
-> );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

Step-2:

For establishing the connectivity between JSP & MySQL using JDBC driver. What we need is to download MySQL JDBC Connector. Download this connector from http://www.mysql.com/products/connector/j/. Just pick up the mirror & download ZIP file. Then from the extracted folder just copy jar file named mysql-connector-java-xx-bin.jar to c:/your-tomcat-directory/common/lib.

Then just set CLASSPATH using environmental variables. for that purpose go to control panel → system → system

properties → environmental variables & set CLASSPATH.



Rules for Beans :-

1. Package should be first line of Java bean.
2. Bean should have an empty constructor.
3. All the variables in a bean should have "get", "set" methods.
4. The property name should start with Uppercase letter when used with "set", "get" methods.
5. for example the variable "name" the get, set methods will be getName(), set Name (String).
6. Set method should return a void value like "return void ()".

Cookie is a name-value pair, which is stored in client. If we want that men can also desable the cookie.

for security reasons, cookie based session management uses two types of cookies.

Non Secure session cookie → This can flow between the browsers and servers under SSL ∞ Socket Session Layer.

Secure authentication cookie → used to authenticate the data, This flow over SSL. This type of authentication cookie are used where the info security is very imp.

Using Beans in JSP:

Java Bean is a class that implements java.io.Serializabl interface and uses set/get methods to project its properties Since they are reusable instance of a class. Java Beans provides the flexibility to JSP pages. There are three basic actions or tags to embedd a java bean into a jsp page. They are:

1. <jsp:useBean>.
2. <jsp:setProperty>.
3. <jsp:getProperty>.

1. <jsp:UseBean>: This tag is used to associate a bean with the given "id" and "scope" attributes.

2. <jsp:setProperty>: This tag is used to set the value for a beans property mainly with the "name" attribute which defines a object already defined with the same scope. Other attributes are "property", "param", "value".

3. <jsp:getProperty>: This tag is used to get referenced beans instance property and stores in implict out object.