Unit II

Q) JSP

It stands for Java Server Pages.
It is a server side technology
It is used for creating web application.
It is used to create dynamic web content
In this, JSP tags are used to insert JAVA code into HTML pages.
It is an advanced version of Servlet Technology.
It is a web based technology helps us to create dynamic and platform independent web pages.
In this, Java code can be inserted in HTML/XML pages or both.
JSP is first converted into servlet by JSP container before processing the clients request.
JSP pages are more advantageous than

PAGE No.
DATE / /

Servlet:
- They are easy to maintain
- No recompilation or redeployment is required.
- JSP has access to entire API of JAVA.
- JSP are extended version of Servlet.

Features of JSP
- Coding in JSP is easy :- As it is just adding JAVA code to HTML/XML.
- Reduction in the length of code - In JSP we use action tags, custom tags etc.
- Connection to Database is easier - It is easier to connect website to database and allows to read or write data easily to the database.
- Make Interactive websites - In this we can create dynamic web pages which helps user to interact in real time environment.
- Portable, Powerful, flexible and easy to maintain :- as these are browser and server independent
- No Redeployment and No Re-Compilation :- Its dynamic, secure and platform independent so no need to re-compilation.
- Extension to servlet - as it has all features of servlets, implicit objects and custom tags

JSP Syntax

PAGE No.
DATE / /

Syntax available in JSP are following

1) Declaration Tag - It is used to declare variables.

Syntax :-

<%! Dec var %>

Example:-

<%! int var = 10; %>

2) Java Scriplets :- It allows us to add any number of JAVA code, variables and expressions.

Syntax :-

<% java code %>

3) JSP Expression :- It evaluates and convert the expression to a string.

Syntax :-

<% = expression %>

Example:-

<% num1 = num1 + num2 %>

4) JAVA Comments :- It contains the text that is added for information which has to be ignored.

Syntax :-

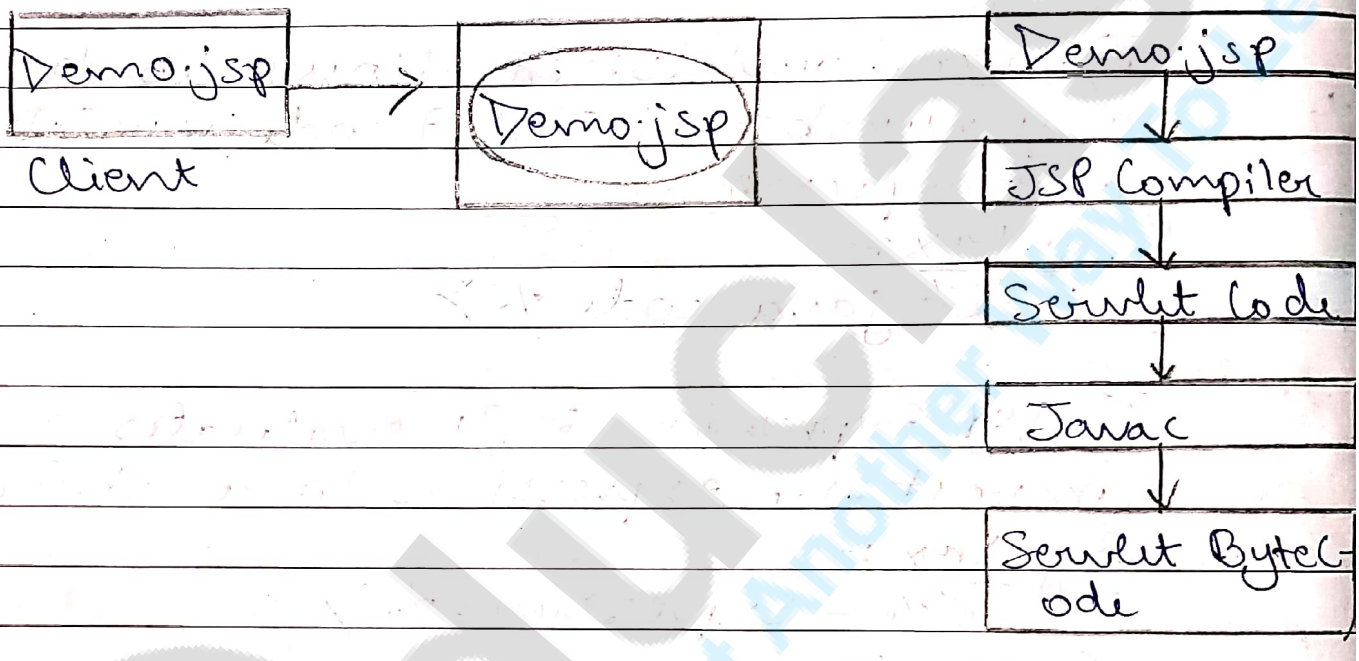
<% -- JSP Comments %>

Process of Execution

Steps for Execution of JSP are following :-

- Create html page from where request

will be sent to server eg try.html
- To handle to request of user next is to create .jsp file eg:- new.jsp
- Create project folder structure.
- Create XML file eg my.xml.
- Create WAR file.
- Start Tomcat
- Run Application



Example of Hello World

demo.jsp

```
<html>
 <head>
<meta http-equiv="Content-Type" content=
"text/html; charset=ISO-8859-1">
<title> Hello World </title>
 </head>
<body>
  <%= "Hello World!" %>
</body>
</html>
```

PAGE No.
DATE / /

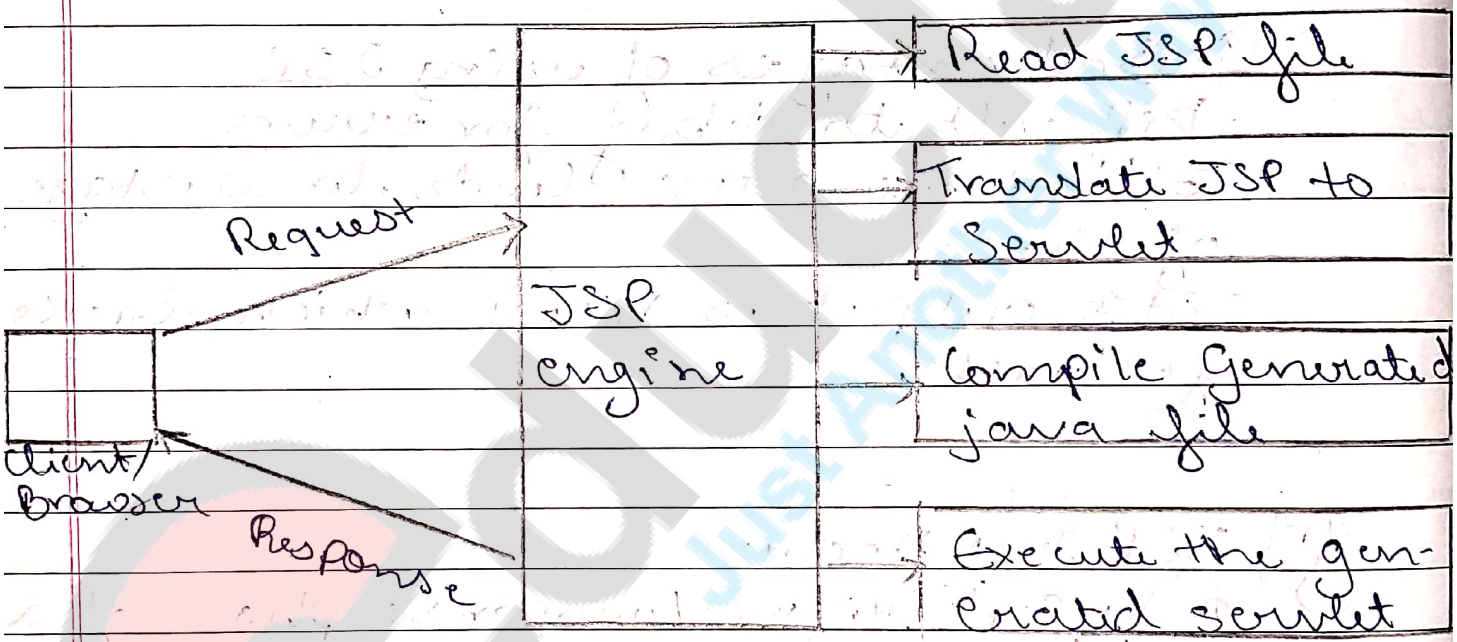
Advantages of using JSP
- It does not require advanced knowledge of JAVA
- It is capable of handling exceptions
- Easy to use and learn
- It uses tags which are easy to use and understand
- Implicit objects are there which reduces the length of code
- It is suitable for both JAVA and non JAVA programmer.

Disadvantages of using JSP
- Difficult to debug for errors
- First time access leads to wastage of time.
- It's output is HTML which lacks features.

Q) JSP Architecture
A typical request/response phase of a JSP is defined as follows
a) Request is initiated for a jsp file by client using browser.
b) Webserver (JSP engine) loads the JSP file and translate the JSP file to generate a java code. Generated java code will be a Servlet.
c) Once Servlet (Java code) is generated, JSP engine compiles the servlet. Any compilation errors will

be detected in this phase.
d) Now servlet class is loaded by the container and executes it.
e) Engine sends the response back to the client.

Translation and compilation phase is done only when
a) first request came for the jsp file
b) the generated servlet is older than the JSP file. This is the case when JSP file is modified.



JSP Directives

JSP Directives are used to give special instruction to container for translation of JSP to servlet code. JSP directives are placed between <%@ %>

JSP provides three directives for us to use:-
1) JSP page directive
2) JSP include directive

3) JSP taglib directive
Every jsp directive has a set of attributes to provide specific type of instructions. So usually a JSP directive will look like <%@ directive attribute = "value" %>.

JSP page directive
page directive provide attributes that gets applied to the entire JSP page. page directive has a lot of attributes. We can define multiple attributes in a single page directive or we can have multiple page directives in a single JSP page.

1) import attribute - This is one of the most used page directive attribute. It is used to instruct container to import other java classes, interfaces, enums etc, while generating servlet code. This is similar to import statements in java classes, interfaces. An example of import page directive usage is :-
<%@ page import = "java.util.Date, java.util.List, java.io.*" %>

2) contentType attribute :- This attribute is used to set the content type and character set of the response. The default value of contentType attribute is "text/html; charset = ISO-8859-1". We can use it like below.
<%@ page contentType = "text/html;

charset=US-ASCII "%>

3) pageEncoding attribute - We can set response encoding type with this page directive attribute, its default value is "ISO-8859-1".
<%@ page pageEncoding="US-ASCII" %>

4) extends attribute - This attribute is used to define the super class of the generated servlet code. This is very rarely used and we can use it if we have extended HttpServlet and overridden some of its implementations. for example;
<%@ page extends="org.apache.jasper.runtime.HttpJspBase" %>

5) info attribute - We can use this attribute to set the servlet description and we can retrieve it using Servlet interface getServletInfo() method. for example;
<%@ page info="Home Page JSP" %>

6) buffer attribute - We know that JSP writer has buffering capabilities, we can use this attribute to set the buffer size in KB to handle output generated by JSP page. Default value of buffer attribute is

8Kb. We can define 16 KB buffer size as:
<%@ page buffer = "16 Kb" %>

7) language attribute - language attribute is added to specify the scripting language used in JSP page. It's default value is "java" and this is the only value it can have.
<%@ page language = "java" %>

8) isELIgnored attribute - We can ignore the Expression Language (EL) in JSP using this page directive attribute. Its datatype is Java Enum and default value is false, so EL is enabled by default. We can instruct container to ignore EL using below directive:
<%@ page isELIgnored = "true" %>

9) isThreadSafe attribute - We can use this attribute to implement SingleThread Modal interface in generated Servlet. Its an Enum with default value as true. If we set its value to false, the generated servlet will implement SingleThread Modal and eventually we will loose all the benefits of servlet multi-threading features. You should never set its value to false
<%@ page isThreadSafe = "false" %>

10) errorPage attribute - This attribute is used to set the error page for the JSP, if the JSP throws exception, the request is redirected to the error handler defined in this attribute. Its datatype is URL. For example;
<%-@ page errorPage = "errorHandler.jsp" %>

11) isErrorPage attribute - This attribute is used to declare that current JSP page is an error page. It's of type Enum and default value is false. If we are creating an error handler JSP page for our application, we have to use this attribute to let container know that it's an error page. JSP implicit attribute exception is available only to the error page. For example;
<%-@ page isErrorPage = "true" %>

12) autoFlush attribute - autoFlush attribute is to control the buffer output. Its default value is true and output is flushed automatically when buffer is full. If we set it to false, the buffer will not be flushed automatically and if its full, we will get exception for buffer overflow. We can

use this attribute when we want to make sure that JSP response is sent in full or none. for example,
<%@ page autoFlush="false" %>

13) session attribute - By default JSP page creates a session but sometimes we don't need session in JSP page. We can use this attribute to indicate compiler to not create session by default. It's default value is true and session is created. To disable the session creation, we can use it like below
<%@ page session="false" %>

14) trimDirectiveWhitespaces attribute - This attribute was added in JSP 2.1 and used to strip out extra white spaces from JSP page output. Its default value is false. It helps in reducing the generated code size, notice the generated servlet code keeping this attribute value as true and false. You will notice out.write("\n") when its true.
<%@ page trimDirectiveWhitespaces="true" %>

JSP include directive
JSP include directive is used to include the contents of another file to the current JSP page. The included file

can be HTML, JSP, text files etc. Include directive is very helpful in creating templates for user views and break the pages into header, footer, sidebar sections.
We can include any resource in the JSP page like below
<%@ include file = "test.html" %>
The file attribute value should be the relative URI of the resource from the current JSP page.

JSP taglib directive
JSP taglib directive is used to define a tag library with prefix that we can use in JSP. We can define JSP tag libraries like below:
<%@ taglib uri = "/WEB-INF/c.tld" prefix = "c" %>
JSP taglib directive is used in JSP standard tag libraries.

(3) JSP scripting elements
The scriptlet tag is used to write the Java code inside the JSP page. JSP scripting elements are written inside <% %> tags.
JSP Scripting Elements
There are three types of scripting elements:
- Scriptlet tag

- declaration tag
- expression tag

JSP scriplet tag
JSP scriplet tag is used to write JAVA code inside JSP page. It includes source code.
Syntax
<% java code %>

Example : implements JSP scriplet tag
```
<html>
<head>
<title> My first JSP page </title>
</head>
<body>
<% out.print("Welcome to JSP world");
%>
</body>
</html>
```

Output : Welcome to JSP world

JSP Declaration Tag
JSP declaration tags are used to declare variable fields and methods. The declaration tag is placed outside the service method and inside the Servlet class to make them class level variable and methods.
Syntax :
<%! Declaration %>

Example:

```
<html>
<head>
<title> JSP Declaration Tag </title>
</head>
<body>
<%! int square (int a)
{
  return a * a;
}
%>
<%= "Square of 5 is :" + square(5)
%>
</body>
</html>
```

Output:

Square of 5 is : 25

JSP Expression Tag

JSP expression tag is used to write the client response to the output stream.

Expression tag keeps the expression that can be used as an argument to the output stream method.

Syntax:

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

Example: Displays current date using Expression tag

```
<%@ page contentType = "text/html"
```

```
import = "java.util.*" %>
<html>
 <head>
  <title> JSP Expression Tag </title>
 </head>
 <body>
   <h2> Current Data </h2>
  <p> Today's Data : <%= (new java.util.
Data()).toLocalString() %> </p>
 </body>
</html>
```

Output:

Today's Data : 03-Aug-2019 10:52:21

JSP Comments

JSP comment is text or statement that is not considered by JSP container. JSP comments are hidden in JSP page source and ignored by JSP container.

Syntax:

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

Example

```
<html>
 <head>
  <title> JSP Comments </title>
 </head>
 <body>
  <%-- this will hide on JSP page --%>
 </body>
</html>
```

Q] Default objects in JSP / Implicit Objects

These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are being created inside service method so we can directly use them within Scriptlet without initializing and declaring them. There are total 9 implicit objects available in JSP.

Implicit Object and their corresponding classes:

out	java.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
exception	javax.servlet.jsp.JspException
page	java.lang.Object
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

1) Out - This is used for writing content to the client (browser). It has several methods which can be used for properly formatting output message to the browser and for dealing with the buffer.

2) Request - The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP we often prompts user to fill in those details, this object is then used to get those entered details on an another JSP page (action page) for validation and other purpose.

3) Response - It is basically used for modifying or dealing with the response which is being sent to the client (browser) after processing the request.

4) Session - It is most frequently used implicit object, which is used for storing the user's data to make it available on other JSP pages till the user session is active.

5) Application :- This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application.

6) Exception :- Exception implicit object is used in exception handling for displaying the error messages. This object is only available to the JSP pages, which has isErrorPage set to true.

7) Page: Page implicit object is reference to the current Servlet instance (Converted Servlet, generated during translation phase from a JSP page). We can simple use this in place of it.

8) pageContext - It is used for accessing page, request, application and session attributes.

9) Config - This is a servlet configuration object and mainly used for accessing getting configuration information such as servlet context, servlet name, configuration parameters etc.

Q) JSP Actions

JSP Actions lets you perform some action

Directives vs Actions

1) Directives are used during translation phase while actions are used request processing phase.

2) Unlike Directives Actions are re-evaluated each time the page is accessed.

The following are the action elements used in JSP:-

1) <jsp:include> Action:
Like include page directive this action is also used to insert a JSP file in another file.
<jsp:include> vs include directive:-
In <jsp:include> the file is being included during request processing while in case of include directive it has been included at translation phase.
Syntax of <jsp:include>:
<jsp:include page="page URL" flush="Boolean Value" />
Here page URL is:- the location of the page needs to be included & flush value can be either true or false (Boolean value).

Example:

```
<html>
<head>
<title> Demo of JSP include action Tag </title>
</head>
<body>
<h3> JSP page: Demo Include </h3>
<jsp:include page="sample.jsp" flush="false" />
</body>
</html>
```

page: Page value is sample.jsp which means this is the page needs to be included in the current file. Just the file name

mentioned which shows that the sample.jsp is in the same directory.
flush: Its value is false, which means resource buffer has not been flushed out before including to the current page.

2) <jsp:forward> Action
<jsp:forward> is used for redirecting the request. When this action is encountered on a JSP page the control gets transferred to the page mentioned in this action.
Syntax of <jsp:forward>:

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

Example:
first.jsp
```
<html>
<head>
<title>Demo of JSP forward Action Tag</title>
</head>
<body>
<h3>JSP page: Demo forward</h3>
<jsp:forward page="second.jsp" />
</body>
</html>
```

Now when JSP engine would execute

first.jsp (the above code) then after action tag, the request would be transferred to another JSP page (second.jsp).
Note: first.jsp and second.jsp should be in the same directory otherwise you have to specify the complete path of second.jsp.

3) <jsp:param> Action
This action is useful for passing the parameters to other JSP action tags such as JSP include & JSP forward tag. This way new JSP pages can have access to those parameters using request object itself.
Syntax of <jsp:param>:
<jsp:param name="param_name_here" value="value_of_parameter_here" />
Example
first.jsp
<html>
<head>
<title> Demo of JSP Param Action Tag </title>
</head>
<body>
<h3> JSP page: Demo Param along with forward </h3>
<jsp:forward page="second.jsp">
<jsp:param name="date" value="20-05-2012" />
<jsp:param name="time" value="10:15

AM" />
<jsp:param name="data" value="ABC"
/>
</jsp:forward>
</body>
</html>

In the above example first.jsp is passing parameters (data, time & data) to second.jsp and second.jsp can access these parameters using the below code-

Date: <%= request.getParameter("data") %>
Time: <%= request.getParameter("time") %>
My Data: <%= request.getParameter("data") %>

4) <jsp:useBean> Action
This action is useful when you want to use Beans in a JSP page, through this tag you can easily invoke a bean.
Syntax of <jsp:useBean>:

<jsp:useBean id="unique_name_to_identify_bean" class="package_name.classname" />
Example of <jsp:useBean>, <jsp:setProperty> & <jsp:getProperty>:
Once Bean class is instantiated using

above statement, you have to use jsp:setProperty and jsp:getProperty actions to use the beans parameters.

EmployeeBeanTest.jsp

```
<html>
<head>
<title>JSP Page to Show use of useBean action</title>
</head>
<body>
<h1>Demo: Action</h1>
<jsp:useBean id="student" class="javabeansample.StuBean" />
<jsp:setProperty name="student" property="*" />
<h1>
name: <jsp:getProperty name="student" property="name" /><br>
empno: <jsp:getProperty name="student" property="rollno" /><br>
</h1>
</body>
</html>
```

StudentBean.java

```
package javabeansample;
public class StuBean{
    public StuBean(){
    }
    private String name;
    private int rollno;
    public void setName(String name)
```

```
{
this.name=name;
}
public String getName()
{
return name;
}
public void setRollno(int rollno)
{
this.rollno=rollno;
}
public int getRollno()
{
return rollno;
}
}
```

5) <jsp:setProperty> Action

This action tag is used to set the property of a Bean, while using this action tag, you may need to specify the Bean's unique name (it is nothing but the id value of useBean action tag)

Syntax of <jsp:setProperty>

```
<jsp:usebean id="unique_name_to_identify_bean" class="package_name.class_name"/>
...
<jsp:setProperty name="unique_name_to_identify_bean" property="property_name"
/>
```

or

<jsp:useBean id="unique_name_to_identify_bean" class="package_name.class_name">
...
...
<jsp:setProperty name="unique_name_to_identify_bean" property="property_name" />
</jsp:useBean>

In property_name, you can also use '*', which means any request parameter which matches to the Bean's property will be passed to the corresponding setter method.

6) <jsp:getProperty> Action

It is used to retrieve or fetch the value of Bean's property.

Syntax of <jsp:getProperty>

<jsp:useBean id="unique_name_to_identify_bean" class="package_name.class_name">
...
...
<jsp:getProperty name="unique_name_to_identify_bean" property="property_name" />

OR

<jsp:useBean id="unique_name_to_identify_bean" class="package_name.class_name">
...
...
<jsp:getProperty name="unique_name_to_identify_bean" property="property_name" />
</jsp:useBean>

PAGE No.
DATE / /

7) <jsp:plugin> Action
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> tags into the file

Syntax
<jsp:plugin type="applet/bean" code="objectcode" codebase="objectcodebase">

Here the type specifies either an object or a bean
Code specifies class name of applet or bean
Code base contains the base URL that contains files of classes

8) <jsp:body> Action
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>
Here we write XML body tag within this tags.

9) <jsp:text> Action
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>.

Here template text refers to only template text (which can be any generic text which needs to be printed on jsp) or any EL expression.

10) <jsp:attribute> Action

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>

Q] Exception handling in JSP

We can handle exceptions using the below two methods :-

1) Exception handling using exception implicit object.

2) Exception handling using try catch blocks within scriplets

Exception handling using exception implicit object

In the below example - we have specified

the exception handling page using errorpage attribute of page directive. If any exception occurs in the main JSP page the control will be transferred to the page mentioned in errorPage attribute.
The handler page should have isErrorPage set to true in order to use exception implicit object. Thats the reason it is set isErrorPage true for errorpage.jsp.

index.jsp

```
<%@page errorPage="errorpage.jsp" %>
<html>
<head>
<title>JSP exception handling example</title>
</head>
<body>
<%
int num1=122;
int num2=0;
int div= num1/num2;
%>
</body>
</html>
```

errorpage.jsp

```
<%@ page isErrorPage="true" %>,
<html>
<head>
<title>Display the Exception Message here</title>
</head>
```

```
<body>
<h2>errorpage.jsp</h2>
<i>An exception has occured in the index.jsp page. Please fix the errors. Below is the error message:</i>
<b><%= exception %></b>
</body>
</html>
```

Exception handling using try catch blocks within scriplets

Since try catch blocks are java code so it must be placed inside scriplet. On the below example the array of length 5 is declared and tried to access the 7th element which doesn't exist. It causes Array Index out of bounds exception.

error.jsp

```
<html>
<head>
<title>Exception handling using try catch blocks</title>
</head>
<body>
<%
try{
int arr[] = {1,2,3,4,5};
int num = arr[6];
out.println("7th element of arr"+num);
}
catch(Exception exp){
```

```
out.println("There is something wrong:"
+exp);
}
%>
</body>
</html>
```

Q) Session tracking techniques in JSP

HTTP is a "stateless" protocol which means each time a client retrieves a web page, the client opens a new connection to the web server and the server does not keep any record of previous client request. Session tracking is a mechanism that is used to maintain state about a series of requests from the same user (requests originating from the same browser) across some period of time. A session id is a unique token number assigned to a specific user for the duration of that users session.

Need of Session Tracking:

HTTP is a stateless protocol so when there is a series of continuous request and response from a same client to a server, the server cannot identify which client is sending request. If we want to maintain the conversational state, session tracking is needed. For example, in a shopping cart application a client keeps on adding items into

his cart using multiple requests. When every request is made, the server should identify in which client's cart the item is to be added. So in this scenario, there is a certain need for session tracking.
Solution is, when a client makes a request it should introduce itself by providing unique identifier every time. There are four ways to maintain session between web client and web server.

Methods to track session:

1) Cookies
2) URL Rewriting
3) Hidden fields
4) Session API

Cookies:

Cookies mostly used for session tracking. Cookie is a key value pair of information, sent by the server to the browser. This should be saved by the browser in its space in the client computer. Whenever the browser sends a request to that server it sends the cookie along with it. Then the server can identify the clients using the cookie.
This is not an effective way because many time browser does not support a cookie or users can opt to disable cookies using their browser preferences. In such case, the browser will not save the cookie at client computer and session

Tracking Tools:

URL Rewriting:-
Here is a simple URL which will pass two values using GET method. You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. For example Original URL: http://javawebtutor.com/jsp/name Rewritten URL: http://javawebtutor.com/jsp/name?sessionid=12345. When a request is made is made an additional parameter is appended with the url. sessionid=12345, the session identifier is attached as sessionid=12345 which can be accessed at the web server to identify the client.

Hidden form fields:
HTML forms have an entry that looks like <INPUT TYPE="HIDDEN" NAME="session" VALUE="...">. This means that, when the form is submitted, the specified name and value are included in the GET or POST data. This can be used to store information about the session. However, it has the major disadvantage that it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.

PAGE No.
DATE / /

The session Object:
Servlets provided HttpSession Interface will provide a way to identify a user across multiple request to a Website and to store information about that user.
For JSPs, by default a session is automatically created for the request if one does not exist. So if your starting page is a JSP, a session would have already been created when you get the first request.

1) Setting session:
Before we validate or check the existing session it is important to know that how we can set session in JSP. We need to use session.setAttribute("ATTRIBUTE NAME", "ATTRIBUTE VALUE") method for setting value in session. You can set as many attribute as you want.

2) Retrieving values from session attribute:
To retrieve value from session attribute we need to use following method session.getAttribute("attribute name");
The information can be stored in the session for the current session by setAttribute() and then retrieve by getAttribute() method, whenever needed.
Setting Session: session.setAttribute("username", name);
Getting Session: session.getAttribute("

username")

To understand a session a very simple example of getting the name from the user and saving it in the session is created. We will retrieve this name from session on next page and display on the page.

index.jsp

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>Session Management Example</title>
</head>
<body>
<form method="post" action="firstpage.jsp">
<font size=5>Enter your name<input type="text" name="name"></font>
<br><br>
<font size=5>Enter your password<input type="password" name="password"></font><br><br>
<input type="submit" name="submit" value="Submit">
</form>
</body>
</html>
```

firstpage.jsp

```
<%
String name = request.getParameter("name");
String password = request.getParameter
("password");
if(name.equals("Atul") && password.
equals("Nair")){ session.setAttribute
("username",name);
response.sendRedirect("secondpage.jsp");
}
else{
  response.sendRedirect("index.jsp");
  }
%>
```

Secondpage.jsp

```
<html>
<head>
<title> Welcome in the program of sessi-
on </title>
</head>
<body>
<font size=5> Hello <%= session.getA-
ttribute("username") %> </font>
</body>
</html>
```

Output in Browser :-

Enter your name: Atul

Enter your password: ....

Submit

Hello Atul

When entered value is wrong you will be redirected to index.jsp page

Enter your name [ ]
Enter your password [ ]
[Submit]

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.

Example:

In the coming example we are creating a custom tag MyMsg which will display the message "This is my own custom tag" when used in a JSP page.

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. In the below we are extending the class SimpleTagSupport

Details.java

```
package beginnersbook.com;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class Details extends SimpleTagSupport{
public void doTag() throws JspException, IOException
  JspWriter out = getJspContext().getOut();
  out.println("This is my own custom tag");
  }
}
```

TLD file

This file should present at the location: Project Name/WebContent/WEB-INF/ and it should have .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 beginnersbook.com so we have given the value as beginnersbook.com.Details.

message.tld

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

Using custom tag in JSP:-
Above we have created a custom tag named MyMsg. Here we will be using it.
Note: taglib directive should have the TLD file path in uri field. Above we have created the message.tld file so we have given the path of that file. Choose any prefix and specify it in taglib directive's prefix field. Here we have specified it as myprefix. Custom tag is called like

this: <prefix:tagName/>. Our prefix is myprefix and tag name is MyMsg so we have called it as <myprefix:MyMsg /> in the below JSP page:

```
<%@ taglib prefix="myprefix" uri="WEB-INF/message.tld" %>
<html>
<head>
<title>Custom Tags in JSP Example</title>
</head>
<body>
<myprefix:MyMsg />
</body>
</html>
```

Output:

This is my own custom tag

9) JSTL

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

JSTL is divided into 5 groups:

1) JSTL Core: JSTL Core provides several

core tags such as if, forEach, import, out etc to support some basic scripting task. Url to include JSTL CoreTag & inside JSP page is →
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

2) JSTL formatting : JSTL formatting library provides tags to format text, date, number for Internationalised web sites. Url to include JSTL formatting Tags inside JSP page is →
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

3) JSTL SQL : JSTL SQL library provides support for Relational Database connection and tags to perform operations like insert, dlete, update, select etc on SQL database. Url to include JSTL SQL Tag inside JSP page is →
<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>

4) JSTL XML : JSTL XML library provides support for XML processing. It provides flow control, transformation features etc. Url to include JSTL XML Tag inside JSP page is →
<%@taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

5) JSTL functions: JSTL functions is library provides support for string manipulation. Url to include JSTL function tag inside JSP page is →

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

JSP JSTL Core library

The JSTL core library contains several tags that can be used to eliminate the basic scripting overhead such as for loop, if..else conditions etc from a JSP Page. Some important tags of JSTL Core library:-

• JSTL if tag: The if tag is a conditional tag used to evaluate conditional expressions. When a body is supplied with if tag, the body is evaluated only when the expression is true. For example:

```
<%-@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title> Tag Example</title>
</head>
<body>
<c:if test="${param.name == 'study tonight'}">
<p> Welcome to ${param.name}</p>
</c:if>
</body>
```

</html>

• JSTL out tag: The out tag is used to evaluate an expression and write the result to JspWriter. For example-

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:out value="${param.name}" default="StudyTonight" />
</body>
</html>
```

The value attribute specifies the expression to be written to the JspWriter. The default attribute specifies the value to be written if the expression evaluates null.

• JSTL forEach tag: This tag provides a mechanism for iteration within a JSP page. JSTL forEach tag works similarly to enhanced for loop of Java Technology. You can use this tag to iterate over an existing collection of items. For example:

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

```
<html>
<head>
<title> Tag Example </title>
</head>
<c:forEach var="message" items="${errorMsgs}">
 <li> ${message} </li>
</c:forEach>
</body>
</html>
```

Here the attribute items has its value as an EL expression which is a collection of error messages. Each item in the iteration will be stored in a variable called message which will be available in the body of the forEach tag.

• JSTL choose, when, otherwise tag:-
These are conditional tags used to implement conditional operations. If the test condition of the when tag evaluates to true, then the content within when tag is evaluated, otherwise the content within the otherwise tag is evaluated. We can also implement if-else-if construct by using multiple when tag. The when tags are mutually exclusive, that means the first when tag which evaluates to true is evaluated and then, the control exits the choose block.

If none of the when condition evaluates to true, then otherwise condition is evaluated for example.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag example</title>
</head>
<body>
<c:forEach var="tutorial" items="${MyTutorialMap}" begin="0" end="5"
varStatus="status">
<c:choose>
<c:when test="${status.count % 2 == 0}">
<p>Divisible by 2: ${tutorial.key}</p>
<br/>
</c:when>
<c:when test="${status.count % 5 == 0}">
<p>Divisible by 5: ${tutorial.key}</p>
<br/>
</c:when>
<c:otherwise>
<p>Neither divisible by 2 nor 5: ${
tutorial.key}</p><br/>
</c:otherwise>
</c:choose>
</c:forEach>
</body>
</html>
```

PAGE No.
DATE / /

• JSTL import tag: <c:import> tag is used to dynamically add the contents from the provided URL to the current page, at request time. The URL resource used in the <c:import> url attribute can be from outside the web container for example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:import url="http://www.example.com/hello.html">>
<c:param name="showproducts" value="true" />
</c:import>
</body>
</html>
```

• JSTL url tag: The JSTL url tag is used to store a url in a variable and also perform url rewriting when necessary. for example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
```

</head>
<body>
<a href='<c:url value="/home.jsp" />'>
Go Home
</body>
</html>

• JSTL set tag :- The JSTL set tag is used to store a variable in specified scope or update the property of JavaBean instance. Following is the example of setting the name property of a student bean:

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:set target="student" property="name" value="${param.name}" />
</body>
</html>

• JSTL catch tag :- The JSTL catch tag is used to handle exception and doesn't forward the page to the error page. For example:

<%@taglib uri="http://java.sun.com/

```
jsp|jstl|core" prefix="c" %>
<html>
<head>
<title> Tag Example </title>
</head>
<body>
  <c:catch>
    <% int a=0;
       int b=10;
       int c=b/a;
    %>
  </c:catch>
  </body>
  </html>
```