UNIT 4 JAVA SERVER PAGES

Structure

4.0 INTRODUCTION

Servlet is a server-side programming language. In unit 2 and 3 of this course, you have already gone through the concept of Servlets in detail. . In this Unit, you will learn another server-side language i.e. Java Server Pages (JSP). Both the JSP and Servlets are correlated. JSP uses a component-based approach that allows web developers to easily combine static HTML for look-and-feel with Java components for dynamic features.

JSP is a specification of Sun Microsystems which first appeared in 1999. The current specification of JSP is 2.3. The updates between JSP 2.2 and 2.3 are relatively minor. JSP is a technology for developing dynamic web pages. It follows the characteristics of Java ‘write once and run anywhere’. A JSP document contains HTML tags as well as JSP elements. JSP page comprises different components such as directives, scripting elements, standard actions and implicit objects.

This unit covers how to create a JSP page. It also provides a basic understanding of JSP components that make an entire JSP page, Java Bean, Custom tag, and life cycle of Java Server Pages. This unit will also introduce you to the exception handling in

JSP as well as database connectivity which are a necessary feature for any web application.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- use JSP to create dynamic web pages,
- use directives, scripting and action elements in a JSP page,
- write java code within the JSP page,
- create a custom tag,
- forward request from JSP pages to other resources,
- write program for handling exceptions at page and application level in JSP, and
- develop database applications using JDBC and JSP.

4.2 JSP OVERVIEW

Java Server Pages (JSP) is powerful web technology. Using this technology, you can create dynamic content based web pages. Dynamic web pages are different from static web pages in which web server creates a web page when a web client or user requests it. For example, your online results on IGNOU website, the page for every student instead IGNOU web server dynamically creates a page depending on your enrolment number.

As you know very well, an HTML page is a static page; it contains static content that always remains the same. When you insert some dynamic content or java code inside the HTML page, it becomes JSP page. A JSP page encompasses a very simple structure that makes it easy for developers to write JSP code as well as for servlet engine to translate the page into a corresponding servlet. You can change content dynamically with the help of Java Bean and JSP elements. The JSP elements are the basic building blocks of the page. JSP page consists of directives, scripting elements and action elements. Each of these elements can use either JSP syntax or be expressed in XML syntax, but you cannot use both syntaxes simultaneously. For this problem, you can use the include mechanism to insert files that may use a different syntax.

Both JSP and Servlets are very closely related to each other. You have already studied the Servlets in Unit 2 and Unit 3 of this course. The Java Server Pages have more advantages over the servlets, which are as follows:

- JSP allows programmers to insert the Java code directly into the HTML file, making the JSP document and development process easier, while Servlets use plenty of 'println' statements for printing an HTML document that is usually very difficult to use. The JSP has no such tedious task to perform.
- JSP supports element based dynamic content that allows programmers to develop custom tags libraries to satisfy application needs.
- In a JSP page, visual content and logic are separated, which is not possible in the servlets.
- JSP pages can be used in conjunction with servlets that handle business logic.
- Major advantage of JSP page is that in case JSP page is modified, there is no need to recompile and redeploy the project. But the Servlet code needs to be

updated and recompiled if we have to change the look and feel of the application.

To create the first JSP page, write some HTML code and java code, similar to code given below, and save it with .jsp extension. This is a simple example of JSP where you are using the scriptlets tag to put Java code in the JSP page. You will learn scriptlets tag in section 4.5 of this unit.

```
<html><body>

<% out.println("Welcome to the IGNOU family"); %>

</body></html>
```

For running the JSP program, you need Apache's Tomcat. You have already installed this software for servlets using the procedure defined in section 2.7 of Block 1 Unit 2 of this course. You should only create a 'JSP' folder under the 'ex' folder, and this folder exists in the 'webapps' directory in Apache Tomcat. For this, you can see the following figure-1. Place your JSP document in the 'JSP' folder to run the JSP page.

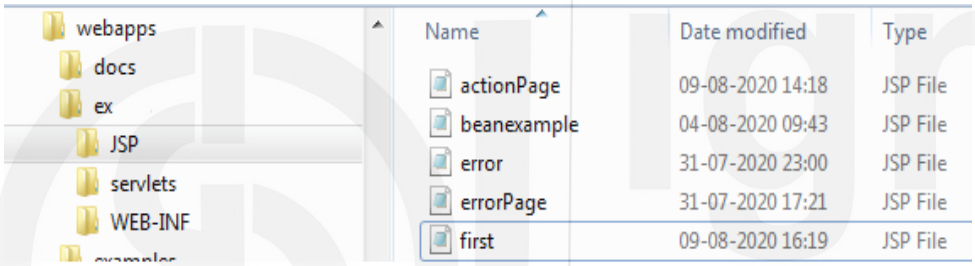

Figure 1: Directory structure for your program in Tomcat

Start your Tomcat Server, open a browser, and type the following URL to execute your first JSP program.

http://localhost:8080/ex/JSP/first.jsp

The output of the program is displayed in figure-2:

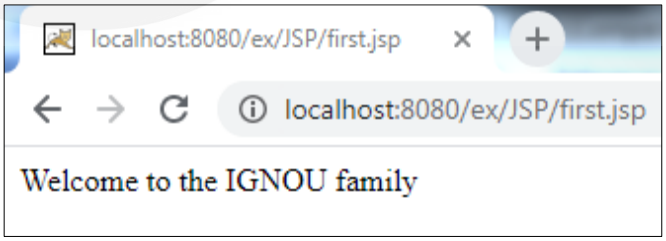

Figure 2: First Program in JSP

You have seen the above program, which looks like an HTML document with some added tags containing java code that allows the server to insert dynamic content in the page. When a client sends a request for a JSP page, the server executes a JSP page elements merges with static contents and sends the dynamically page back to the client browser.

Steps for Creating and running procedure of JSP in NetBeans

Creating and running procedure of servlet in NetBeans is defined in Unit-2 of this block. Installation process is given in Lab manual. This Unit is devoted to JSP. The creating and running procedure of servlet as well as JSP are similar. To create a JSP, open 'Source Packages' in NetBeans, right click on default package -> New -> JSP.

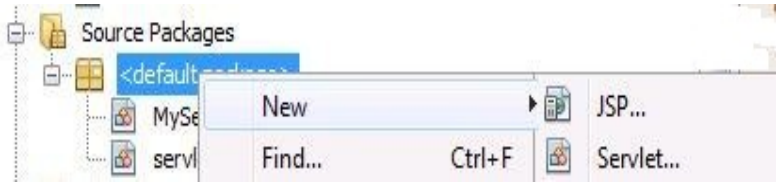

Figure 3: Creation window of JSP

Give a name and select location to JSP file as you have given a name MyJSP.jsp under MyProject in Figure-4. Also select JSP File (Standard Syntax) option.

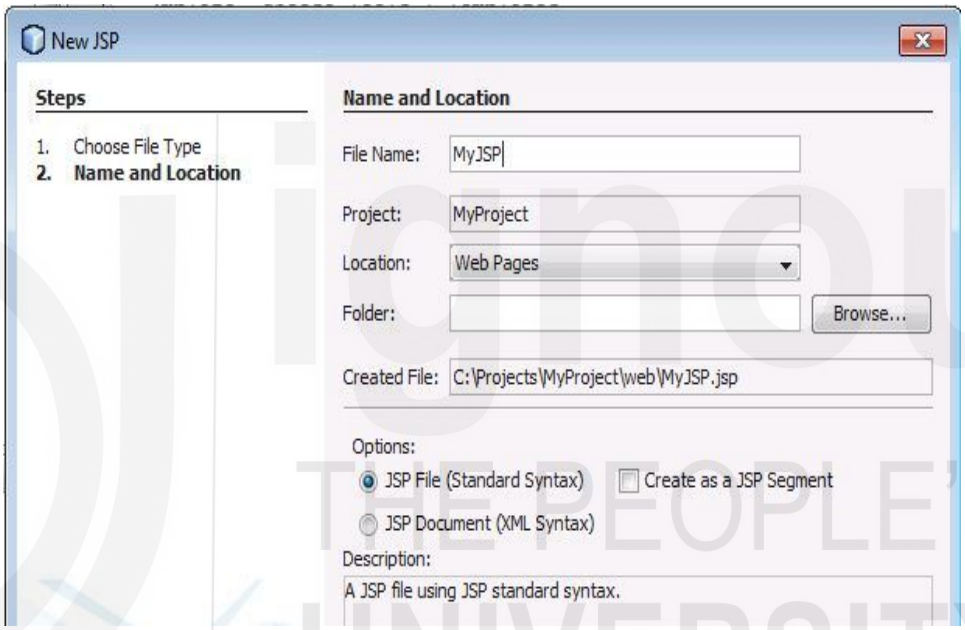

Figure 4: Name and Location window for JSP file

Finally select Finish button from button Panel.

Figure 5: Button Panel

Figure 6:Source code of MyJSP file

Now JSP file is ready, you can change the code as per your reqiurement .

Once this is completed. For running the particular JSP file, right-click on the file and select 'Run File' (Shift+F6). The output of the MyJSP file is displayed as shown in figure-7:

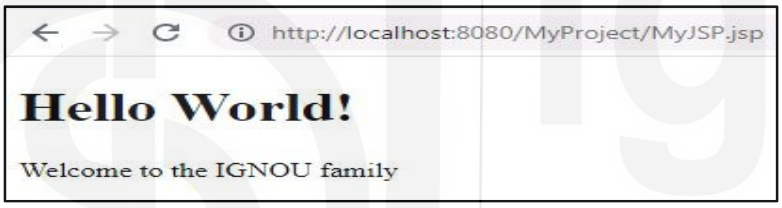

Figure 7: Output Screen of MyJSP file

JSP uses server-side scripting that is actually translated into servlets and compiled before they are run. You can study more about the life cycle of JSP in subsequent section 4.3.

4.3 JSP LIFE CYCLE

In this section, you will go through the life cycle of JSP and see how a JSP page is displayed. When the JSP is first accessed, it is translated into a corresponding servlet (i.e. java class) and compiled, then JSP page services request as a servlet. The

Figure 8: The steps of a JSP page processing

translation of JSP page is done by the JSP engine of the underlying web container/servlet container (e.g. Tomcat). Figure-8 shows how a JSP page is processed.

The life cycle of JSP page is controlled by three methods i.e. jspInit(), _jspService() and jspDestroy().

jspInit() - The jspInit() method is called only once during life cycle of a JSP. Similarly, servlet also has an init() method whose purpose is the same as that of jspInit(). jspInit() method is used to initialize objects and variables that are used throughout the life cycle of JSP. This method is defined in JspPage interface. This method is invoked when the JSP page is initialized. It has no parameters, returns no value and thrown no exceptions.

The signature of the method is as follows:

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

_jspService() – _jspService() is the method that is called every time the JSP page is requested to serve a request. This method is defined in the javax.servlet.jsp.HttpJspPage interface. This method takes HttpServletRequest and HttpServletResponse objects as an argument. The _jspService() method corresponds to the body of the JSP page. It is defined automatically by the processor and should never be redefined by you. It returns no value. The underscore ('_') signifies that you cannot override this method. The signature of the method is as follows:

```
public void _jspService
(
   javax.servlet.http.HttpServletRequest request,
   javax.servlet.http.HttpServletResponse response)
   throws javax.servlet.ServletException, java.io.IOException
{
  // services handling code
}
```

jspDestroy()- The jspDestroy() is invoked only once when the JSP page is about to be terminated. It is synonymous to the destroy() method of a servlet. You have to override jspDestroy() if you need to perform any cleanup, such as releasing database connections or closing open files. The signature of the method is as follows:

```
public void jspDestroy(){    //  cleanup code    }
```

Following figure-9 shows the life cycle of JSP page:

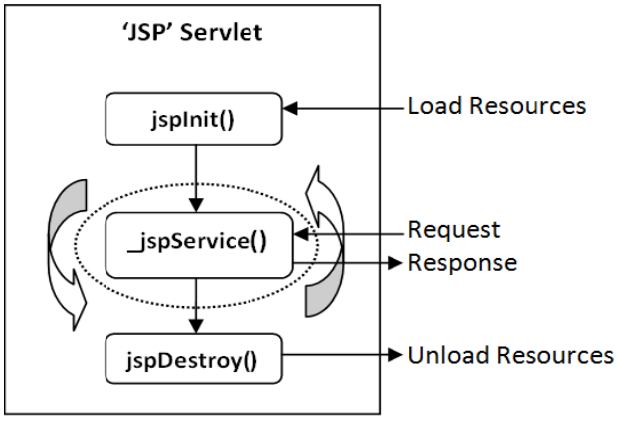

Figure 9: life cycle of JSP Page

4.4 JSP API

The JSP technology is based on JSP API (Application Programming Interface) that contains two packages such as **javax.servlet.jsp** and **javax.servlet.jsp.tagext.** In addition to these two packages, JSP also needs two packages of servlets such as ***javax.servlet*** and ***javax.servlet.http***. Both packages provide interfaces and classes for writing servlets. You have already studied this in Block 1 Unit 2 of this course.

Package javax.servlet.jsp

The javax.servlet.jsp package has two interfaces such as HttpJspPage and JspPage and four classes such as JspEngineInfo, JspFactory, JspWriter and PageContext. These are as follows:

Interface	**Description**
JspPage	The JspPage is the interface that all JSP servlet classes must implement. The JspPage interface has two methods, JspInit and JspDestroy. The jspInit method is similar to the 'init' method in the javax.servlet.Servlet interface. The jspDestroy method is analogous with destroy method of the javax.servlet.Servlet interface. You have studied both the methods in section 4.3 of this Unit. JSP developers rarely make full use of these two methods.
HttpJspPage	The HttpJspPage interface directly extends the JspPage interface. Each JSP page implements this interface. It supports only one method _jspService(). The _jspService() method is already described in life cycle of JSP in section 4.3 of this Unit.
Class	**Description**
JspEngineInfo	The JspEngineInfo is an abstract class that provides information of the current JSP engine.
JspFactory	The JspFactory is an abstract class that defines a number of factory methods available to a JSP page at runtime for the purposes of creating instances of various interfaces and classes used to support the JSP implementation.
JspWriter	The jspWriter class is derived from the java.io.Writer class. This is the normal mechanism for JSP pages to produce output to the response. No. of methods are defined in this class such as print( ) and println( ).
PageContext	The PageContext class is an abstract class. It extends JspContext to provide useful context information when JSP technology is used in a servlet environment. This class provides methods that are used to create other objects. For example, getServletConfig() returns a ServletConfig object and getServletContext() returns a ServletContext object.

Apart from these interfaces and classes, the two exception classes: JspException and JspError are also defined in javax.servlet.jsp package of JSP API. JspException is the base class for all JSP exceptions. For details, please refer to the following link:
https://docs.oracle.com/cd/E17802_01/products/products/jsp/2.1/docs/jsp-2_1-pfd2/javax/servlet/jsp/package-summary.html

Package javax.servlet.jsp.tagext

The javax.servlet.jsp.tagext encompasses classes and interfaces for the definition of JSP Tag Libraries. The use of this package is defined in the creation of custom tag under the section 4.5.1(taglib directives) of this unit. The interfaces in the

javax.servlet.jsp.tagext package are BodyTag, IterationTag, Tag, and TryCatchFinally. The classes are defined in this package such as BodyContent, BodyTagSupport, PageData, TagAttributeInfo, TagData, TagExtraInfo, TagInfo, TagLibraryInfo, TagLibraryValidator, TagSupport, TagvariableInfo and VariableInfo. You know more about the package details; please refer to the following link:
https://docs.oracle.com/javaee/5/api/javax/servlet/jsp/tagext/package-summary.html

4.5 COMPONENTS OF JSP

In this section, we are going to learn the components of JSP that make up a JSP page. There are three types of JSP components such as **Directives, Scripting and Action**. All the components are to be discussed in detail in the following sections:

4.5.1 Directives

Directives have great use as it guides the JSP container for translating and compilation of JSP page. Directives control the processing of an entire JSP page. It appears at the top of the page. Using directives, the container translates a JSP page into the corresponding servlet. They do not directly produce any output. A directive component comprises one or more attributes name/value pairs. Directives are defined by using <%@ and %> tags. The syntax of Directive is as under:

<%@ directive attribute = "value" %>

There are three types of directives used in JSP documents**: page, include** and **taglib**. Each one of these directives and their attributes is defined in the following sections:

The *page* Directive

The page directive defines attributes that apply to an entire JSP page, such as the size of the allocated buffer, imported packages and classes/interfaces, and the name of the page that should be used to report run time errors. The page Directive is a JSP element that provides global information about an entire JSP page. This information will directly affect the compilation of the JSP document. The syntax of JSP page directive is as under:

<%@ page attribute = "value" %>

The following are the different properties that can be defined by using page directive:

Attribute	Description
Language= "scripting language"	This attribute outlines the language that will be used to compile the JSP document. By default, it is java language.
import= "import list"	This attribute defines the names of packages.
session= "true|false"	It specifies whether or not the JSP document participates in HTTP session. The default is *true*.
extends= "classname"	This attribute states the name of the parent class that will be inherited by the generated servlet. It is rarely used.
buffer= "none|size in kb"	The default value is 8kb. It specifies the size of '*out'* buffer.
autoFlush= "true|false"	The default is *true*. It means that the *out* buffer will be automatically flushed when full. If it is *false*, it will be raised as an exception when the buffer is full.
Info= "text"	If this attribute is used, the servlets will override the getServletInfo() method.
errorPage= "error_page_url"	This attribute defines the relative URL to JSP document that will handle the exception.
isErrorPage= "true|false"	This attribute indicates that the current page can act as an error page for another JSP page. The default value is *false*.
isThreadSafe= "true|false"	The default value is *true*. It indicates that the page can service more than a request at a time. When it is *false*, the SingleThreadModel is used.

An example of the use of page directive is as follows:

```
<%@ page import="java.io.*, java.util.Date" buffer="16k" autoFlush="false" % >
<%@ page errorPage="error.jsp" %>
```

You can specify one or more page directives in your JSP document. In the above example, first page directive statement instructs the web container to import java.io package and java.util.Date class. It also instructs the web container to set buffer size to 16k and turn off autoflushing. The second page directive statement defines a name of error page which is used for handling errors.

The include Directive

JSP include directive is used to include files such as HTML, JSP into the current JSP document at the translation time. It means that it enables you to import the content of another static file into a current JSP page. The advantage of using an include directive is to take advantage of code re-usability. This directive can appear anywhere in a JSP document. The syntax of include directive is as follows:

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

Note: *Relative URL only specifies the filename or resource name, while absolute URL specifies the protocol, host, path, and name of the resource name.*

Consider the following example for include Directive. In this example, there are two files: an HTML file, and another is a JSP file. You can create both files but run only JSP file and the result is displayed like the figure-5.

Source code for header.html:

```
<html> <body>
<h2>Indira Gandhi National Open University</h2>
<h4>The text is from Header File</h4>
</body></html>
```

Source code for index.jsp:

```
<html><body>
<h3>Example of include directive</h3>
<%@ include file= "header.html" %>
</body></html>
```

The output of the above program is shown in the following figure-10:

Figure 10: Example of Include Directive

The taglib Directive

This directive allows users to use Custom tags in JSP. A custom tag is user-defined tag. The custom tag eliminates the need for scriptlet tag. In this section, you will learn

about the creation of custom tag libraries in Java Server Pages. It is a reusable code in a JSP page and tag library is a collection of custom tags. The taglib directive has the following syntax:

<%@ taglib uri=”tagLibraryURI” prefix=”tagPrefix” %>

The uri (Uniform Resource Identifier) attribute defines an absolute or relative uri of tag library descriptor (TLD) file, and the prefix attribute defines the string that will identify a custom tag instance.

There are four components which you need to use customs tags in a JSP page:

1) The **Java file** that does the processing or Tag handler class

2) **Tag Library Descriptor** (TLD) file that points from JSP to Java file

3) **JSP** file that contains the tag being used

4) **Deployment descriptor** file (web.xml) that states the server where to find the TLD file

Tag Handler Class

It is a java class that defines the behaviour of the tags. This class must implement the javax.servlet.jsp.tagext package.

Tag Library Descriptor (TLD) file

A tag library descriptor file is an xml document. It defines a tag library and its tags. This file must be saved with a .tld extension to the file name. It contains <taglib> root element and <tlib-version>, <jsp-version>, <short-name>, <tag> which all are sub elements of the taglib element. The <tag> is the most important element in the TLD file because it specifies the tag's name and class name. You can define more than one <tag> element in the same TLD file.

Deployment Descriptor file (web.xml)

The deployment descriptor is an xml file that specifies the configuration details of the tag. The most essential element for custom tag in web.xml file is <taglib-location>. Using the web.xml, the JSP container can find the name and location of the TLD file.

JSP file

Once you have created tag handler java class, tag descriptor file, and defined configuration details in deployment descriptor file, you have to write a JSP file that uses the custom tag.

The following are five easy steps for building a JSP application that uses custom tags:

Step-1: Write, compile and deploy a java class called MyCustomTag.java, which is given in the following source program. The class file must be placed in the directory, say ‘customTag’ under the WEB-INF/classes directory like the figure 11. The directory ‘customTag’ is user defined directory.

Source Code for simple CustomTag

```
package customTag;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class MyCustomTag extends TagSupport
{
  public int doEndTag() throws JspException
```

```
  {
    JspWriter out = pageContext.getOut();
    try
    { out.println("Hello!! Creating a First Custom tag"); }
    catch(Exception e) {}
    return super.doEndTag();
  }
  //doEndTag()
}
//main class
```

Figure 11: classes directory structure for Custom Tag in Tomcat

Step-2: Create a TLD file named taglib.tld as shown in following source program and save it in WEB-INF directory.

```
<!DOCTYPE taglib  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN"  "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
        <tlib-version>1.0</tlib-version>
        <jsp-version></jsp-version>
        <short-name></short-name>
        <tag>
                <name>myTag</name>
                <tagclass>customTag.MyCustomTag</tagclass>
        </tag>
        </taglib>
```

Step-3: Create a JSP file named SimpleCustomTag.jsp from the following source code and save it in the directory say 'JSP' like the figure 12 under your 'webapps' folder.

```
<html><body>
<%@ taglib uri="/myTLD" prefix="easy"  %>
<easy:myTag />
</body></html>
```

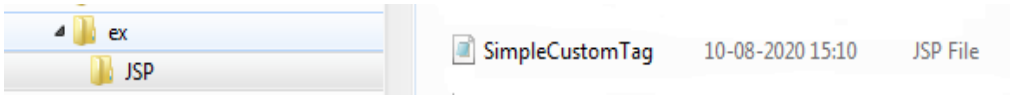

Figure 12: File structure for Custom tag in JSP

Step-4: Edit your deployment descriptor (web.xml) file. To use custom tags, you must specify a <taglib> element in your deployment descriptor file. It is a large file; you can place the following code under the <web-app> root element.

```
<jsp-config>
<taglib> <taglib-uri> /myTLD </taglib-uri>
        <taglib-location>/WEB-INF/taglib.tld </taglib-location>
</taglib>
 </jsp-config>
```

Step-5: Start server let us say Tomcat (if you are using this). Open web browser and run the JSP page using the URL as http://localhost:8080/ex/JSP/SimpleCustomTag.jsp. The following screen comes as an output for a simple custom tag.

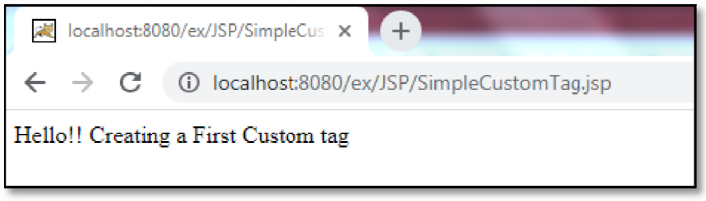

Figure 13: A simple JSP Custom Tag page

When a user requests to a JSP page, the JSP container first checks the taglib directive and gets the taglib 'uri' attribute. On the basis of this, JSP container looks into the web.xml file to find taglib location and continues the processing and gets the name of tag. After getting the name and location of TLD file, it obtains the java class. Now, the JSP container loads the class for custom tag and continues the processing.

4.5.2 Scripting elements

JSP scripting elements is a mechanism for embedding java code fragments directly into a HTML page. There are three scripting language elements such as ***declarations, expressions, scriptlets*** involved in JSP scripting. Each of these scripting elements has an appropriate location in the generated servlet. In this section, you will look at these elements and how together they will result in a complete servlet.

JSP Scripting Element	Description
Declarations	To declare the variables and methods for the page.
Expressions	To return a value from the scripting code as a *String* to the page.
Scriptlets	To execute java source code

Declarations

Declarations are used to declare the variables and methods that you can use in the JSP document. The declaration part is initialized when the JSP document is initialized. After the declarations have been initialized, they are available to other expressions, declarations and scriptlets. A declaration starts with <%! and ends with %>. It has the following syntax:

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

Following is an example for JSP Declarations:

```
<%!  x=0; %> // x is an integer type variable
<%! int x, y, z; %> // x, y, z is an integer type variable
<%!  String name = new String("JSP");  %>  // string type variable declaration
<%! public String getName() { return name; }  %> //method declaration
```

Expressions

The code placed within JSP expression element is written to the output stream of the response. You need not write out.print() to write data. Expressions are evaluated at request time, and the result is inserted where the expression appears in the JSP file. The syntax of the expression is as follows:

```
<%= expression %>
```

Note: Do not end your statement with a semicolon in case of expression tag.
Here is an example of expressions:

```
<%! int x, y, z; %> // x, y, z is an integer type variable
<%
   x=5, y=5;
   z = x/y;
%>
<%=z %>
```

For example, to show the today date and time:

```
<%= new java.util.Date() %>
```

Scriptlets

A scriptlet element is used to execute java source code in JSP. It is executed at the request time and makes use of declarations, expressions and Java Beans. You can write scriptlets anywhere in a page. It contains a valid java statement within the <% and %> tag and gets inserted into the _jspService() method of generated servlet. The syntax of the scriptlet is as follows:

```
<% // java code %>
```

Note: Semicolon at the end of scriptlet is necessary.

The following example will demonstrates how to add two numbers and print this result on output screen using Scriptlets.

```
<html><body>
<%  int num1 = 5,  num2 = 15, sum;
      sum= num1 + num2;
      out.println("sum= "+sum);
%>
<footer><hr>&copy;2020 SOCIS, IGNOU</footer>
</body></html>
```

Output of the above program is as follows:

Figure 14: Output screen for Scriptlets example

☞ Check Your Progress 1

1. Write the basic steps for processing JSP request.

2. Write a JSP program for Fibonacci Series.

3. Write a JSP program using Scriptlets that add numbers from 1 to 10 and prints this result.

4.5.3 Action Elements

In the previous section, you have already learnt about two JSP components i.e. Directives and Scripting elements. Now, you will learn about the third JSP component i.e. Action elements or Standard actions tags that can be embedded in a JSP program. JSP provides many standard action tags that you can use for specific tasks such as add java bean objects in JSP document, forward the request to another resource or include other resources in current resource, etc. At the compile time, they are replaced by java code. The list of some important standard JSP action elements are given below:

JSP Action	Description
<jsp:useBean>	To get the java bean object from given scope or to create a new object of java bean.
<jsp:getProperty>	To get the property of a java bean, used with jsp:useBean action.
<jsp:setProperty>	To set the property of a java bean object, used with jsp:useBean action.
<jsp:param>	To define the name/value pair of parameter to be passed to <jsp:useBean>, <jsp:forward>, <jsp:include> and <jsp:plugin> tag.
<jsp:forward>	To forward the request to another resource.
<jsp:include>	To include a resource at runtime, the recourse may be HTML, JSP or any other file
<jsp:plugin>	To embed another component such as applet in JSP

Before you start learning about how you can add Java Bean in the JSP page, you must look at what a bean is. A Java Bean is a java class that maintains some data called properties and has a no-arguments constructor. It is reusable component that work on any Java Virtual Machine. To create Java Bean, you must create a java class that implements java.io.Serializable interface and uses public get/set methods to show its properties.

<jsp:useBean> Action

This action is used to include an instance of java bean within JSP document. The syntax of first JSP standard action is as follows:

```
<jsp:useBean    id=”name”
                  scope=”page | request | session | application”
                 class=”className” >
                  body
 </jsp:useBean>
```

In the above syntax, ‘id’ represents the name of the object and this attribute is necessary. The ***‘class’*** attribute represents the fully qualified class name of the object. The class name is case sensitive. The ‘***scope***’ attribute represents the life of the object. It may be page, request, session or application. It means how long the object is available for a single user or all application users. JSP provides different scope for sharing data between web pages. These are:

- **Page** - ‘page’ scope means the JSP object can be accessed only from within the same page where it is created. By default, it is page. JSP implicit objects out, exception, response, pageContext, config and page have ‘page’ scope.
- **Request** – Beans with request scope are accessible only within pages that are processing the same request for that the object was created in. Objects that have request scope are most often used when you need to share information between resources that is pertinent for the current request only. Only Implicit object request has the ‘request’ scope.
- **Session** – This object is accessible from any JSP within the same session. Implicit object session has the ‘session’ scope.
- **Application** - This object is accessible from any JSP within the same web application. Implicit object application has the ‘application’ scope.

<jsp:setProperty> Action

The second JSP standard action is used to set the Java Beans property value. The syntax for this action is as follows:

```
<jsp:setProperty  name=”beanName”  property=”propertyName” />
```

In the above syntax, ‘**name**’ attribute represents the name of the bean instance defined by <jsp:useBean> action and ‘**property**’ attribute represents the bean property for which you want to set a value.

<jsp:getProperty> Action

This final bean handling action is *<jsp:getProperty>* which gets the named property and outputs its value for inclusion in the page as a *String*. The syntax for this action is as follows:

```
<jsp:getProperty  name=”name”  property=”propertyName” />
```

In the above syntax, ‘**name**’ attribute represents the name of the bean instance defined by <jsp:useBean> action and ‘**property**’ attribute represents the bean property for which you want to get a value.

The following is a simple Java Bean example that stores University details:

Let’s create a Java Bean named university.java and place class file under WEB-INF/classes/bean1 directory.

Source code for university.java

```
package bean1;
public class Iguniversity
{
  public IGuniversity()
  {
  }
     private String uname;              //define university name
     private int year;                  //define year variable
     private String school;
     /* ------ getMethod   and setmethod for   university name--- */
          public String getUname()
          {
             return uname;
          }
          public void setUname(String uname)
          {
            this.uname = uname;
          }

     /* ------ getMethod   and setmethod for Year--- */

          public int getYear()
          {
            return year;
          }
          public void setYear(int year)
          {
            this.year= year;
          }

          /* ------ getMethod  and setmethod for School--- */
          public String getSchool()
          {
            return school;
          }
          public void setSchool(String school)
          {
            this.school = school;
          }
}
```

You can access the properties of the Java Bean from the following JSP file named as universitydetails.jsp.

```
<jsp:useBean id="universityinfo" class="bean1.IGuniversity" scope="session" />
<jsp:setProperty property="*" name="universityinfo"/>

You have entered following university details:<br>
<jsp:getProperty property="uname" name="universityinfo"/><br>
<jsp:getProperty property="year" name="universityinfo" /><br>
<jsp:getProperty property="school" name="universityinfo"/><br>
```

Now, you can create a JSP file named ‘beanexample.jsp’ for putting values to the bean.

```
<html><head><title>useBean, getProperty and setProperty example </title></head>
<form action="universitydetails.jsp" method="post">
University Name: <input type="text" name="uname"><br>
 Year <input type="text" name="year"><br>
School: <input type="text" name="school"><br>
<input type="submit" value="Go"> </form> </html>
```

Now, you can run 'beanexample.jsp' file in your browser using the URL as http://localhost:8080/ex/JSP/beanexample.jsp and get the following screen for the above programs.

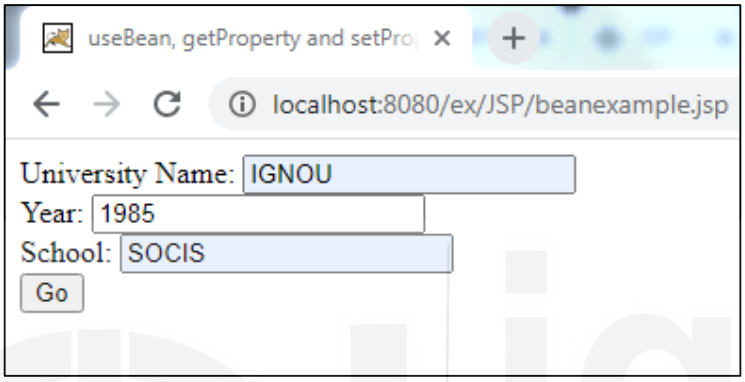

Figure 15: Input Screen for Bean Example

The following output screen is as follows for the University details program:

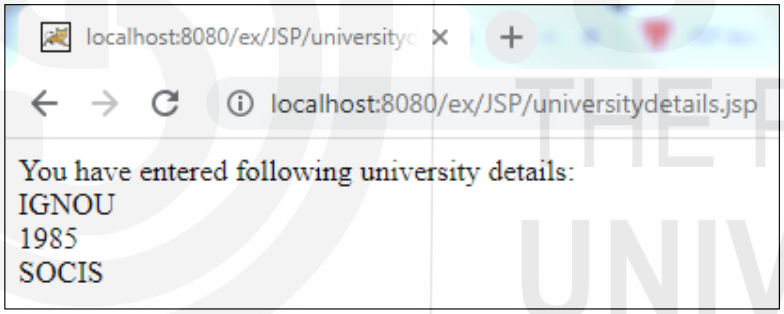

Figure 16: Output Screen for a Java Bean Example

<jsp:param>

The <jsp:param> action is used within the body of <jsp:include> , <jsp:forward> and <jsp:plugin> tag to supply extra name/value pair of parameter. Following is the syntax of the <jsp:param> action :

```
<jsp:param  name="paramName"  value="paramValue" />
```

In the above syntax, name attribute defines the name of parameter being referenced and value attribute represents the value of named parameter.

<jsp:include>

The <jsp:include> action is used to include additional static or dynamic resources in current JSP document at run-time. The static resource is inserted at the translation time and the dynamic resource at the request time. In the previous section, you have already studied the static include i.e. include directive.

The syntax for this action is as follows:

```
<jsp:include page="relativeURL" flush="true" />
                                        OR
<jsp:include page="relativeURL"  flush="true" >
        <jsp:param ……/>
</jsp:include>
```

In the above syntax, ***page*** attribute defines the path of the resource to be included and ***flush*** attribute indicates whether the buffer is flushed. It is an optional parameter. The first syntax is used when <jsp:include> does not have a parameter name/value pair. If you want to pass the parameter to the included resource, use the second syntax. The <jsp:param> tag allows parameter to be appended to the original request.

<jsp:forward>

The <jsp:forward> action is used to terminate the current execution of JSP page and forwarding the client request to another URL, whether it can be an HTML file, JSP or Servlet within the same application at the run time. The <jsp:param> tag is used with <jsp:forward> action element to passing parameter. The syntax for <jsp:forward> action tag is as follows:

```
<jsp: forward page="relative_url" flush="true" />
                                        OR
<jsp: forward page="relative_url"  flush="true" >
        <jsp:param ……/>
</jsp: forward >
```

<jsp:plugin>

The <jsp:plugin> action element enables the JSP to include a bean or a applet in the client page. The <jsp:param> is also used with <jsp:plugin> action element to send parameters to Applet or Bean. It has the following syntax:

```
<jsp:plugin type="pluginType" code="classFile"codebase="relativeURLpath>
<jsp:param ……/>
</jsp: plugin >
```

In the above syntax, ***type*** attribute indicates the type of plugin to include in JSP page, ***code*** attribute represents the name of class file and ***codebase*** attribute is the path where the code attribute can be found.

4.6 JSP IMPLICIT OBJECT

JSP Implicit objects or predefined variables are the java objects that you can use directly without being declared first in scriptlets of JSP document. JSP implicit objects are created during the translation phase of JSP to the servlet. These variables are automatically available for the JSP page developer to use. These nine implicit objects are summarized in the following table:

Table: JSP Implicit Objects

S.No.	Implicit Object	Type	Scope
1	out	javax.servlet.jsp.JspWriter	Page
2	request	javax.servlet.HttpServletRequest	Request
3	response	javax.servlet.HttpServletResponse	Page
4	session	javax.servlet.http.HttpSession	Session
5	application	javax.servlet.ServletContext	Application
6	page	javax.servlet.jsp.HttpJspPage	Page
7	pageContext	javax.servlet.jsp.pageContext	Page
8	config	javax.servlet.http.servletConfig	Page
9	exception	java.lang.throwable	Page

4.6.1 out Object

This implicit object is a simple and frequently used in scriptlet of JSP page. You call either its print() or println() method to send output to the client browser. For example,

<% out.println(“ JSP is an easy language”); %>

In the above code, you use the implicit out object that represents javax.servlet.jsp.JspWriter class.

4.6.2 request Object

The request object is an instance of HttpServletRequest interface. This object is used to retrieve the values that the client browser passed to the server during HTTP request, such as cookies, session, headers information, or parameters associated with the request. The most common use of request object is to access queries by calling the getParameter() and getQueryString() method.

For example: <% String name=request.getQueryString(); %>

You can find another example of request.getParameter() method in Database Connectivity section of this Unit.

4.6.3 response Object

The response object is an instance of HttpServletResponse interface. It is used to add or manipulate response such as redirect response to another resource, send error etc. Using response object, a reply is sent back to the client browser.

<% response.sendRedirect(“http://www.ignou.ac.in”);%>

For example, when the above code is executed, the sendRedirect() method of the javax.servlet.HttpServletResponse to redirect the user to IGNOU website.

4.6.4 session Object

The session object is represented by the javax.servlet.http.HttpSession interface. This object is most often used when there is a need to share information between requests for a single user. It is used to store the user’s data to make it available on other JSP pages until the user session is active. This object is used to track the user information in the same session.

4.6.5 application Object

The application object is an instance of javax.servlet.ServletContext. The ServletContext object is created only once by the web container when application or project is deployed on the server. This object is used to get initialization parameter from configuration file. There is one ServletContext object for each web application per Java Virtual Machine.

4.6.6 page Object

The page object is an instance of javax.servlet.jsp.HttpJspPage interface. The page object is just as it sounds, a reference to the current instance of the JSP page. The page object is a synonym for ‘this’ reference and is not useful for programming language.

4.6.7 pageContext Object

The pageContext object is an instance of javax.servlet.jsp.PageContext. It is used to represent the entire JSP page. The pageContext object is used to set, get and remove attribute of the JSP page.

It provides a single point of access to many page attributes such as directives information, buffering information, errorPageURL and page scope. It is also provide a convenient place to store shared data. This object stores references to the request and response objects for each request. The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, for identifying the four scopes of attributes.

4.6.8 config Object

The config object is an instance of javax.servlet.http.servletConfig. The container for each jsp page creates it. This object is used to get the configuration information of the particular JSP page by using the getServletConfig method. This can be useful in retrieving standard global information such as the paths or file locations.

4.6.9 exception Object

This implicit object only exists in a defined errorPage. It contains reference to uncaught exception that caused the error page to be invoked. This object is assigned to the Throwable class, which is the super class of all errors and exceptions in the java language. You can find a complete description of errorPage mechanism including the use of this exception object in section 4.8 (exception handling in JSP) of this unit.

☞ Check Your Progress 2

1. What is the purpose of action elements in JSP?
   _____
   _____
   _____
   _____
2. Write and explain the various bean scopes.
   _____
   _____
   _____
   _____
3. Explain the application implicit object with example.
   _____
   _____
   _____
   _____

4.7 AN INTRODUCTION TO JSP STANDARD TAG LIBRARY

In the section 4.5.2 (Scripting elements), you have learnt about the scriptlets in JSP pages to generate dynamic content. Sometimes it creates readability issues and made it difficult to maintain the JSP pages. To overcome this problem, Custom tags have been introduced. Although custom tags are better choice than scriptlets but web

developers need to be consumed a lot of time in coding and testing such tags. A new feature named JSP Standard Tag Library (JSTL) has been introduced for web developers to develop web pages in a better manner.

The Java Server Pages Standard Tag Library is a collection of useful JSP tags to simplify the JSP development. The JSTL tags can be classified as per their features such as Core Tags, Formatting tags, SQL tags, XML tags and JSTL Functions. For JSTL 1.1 Tag details, refer to the link: https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/. To begin working with JSP JSTL tags, you need to first install the JSTL library. If you are using the Apache Tomcat, then use this link:

https://tomcat.apache.org/taglibs/index.html to download the library.

There are different types of JSTL tags to include in JSP document as per your requirement.

JSTL Core Tags

The core group of tags are the most commonly used JSTL tags**.** To use these tags in JSP, you should have used the following taglib:

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

JSTL Formatting Tags

The Formatting tags provide support for message formatting, number and date formatting. To use these tags in JSP, you should have used the following taglib:

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

JSTL XML tags

These tags provide support for creating and manipulating the XML documents. The JSTL XML tag library has custom tags used for interacting with the XML data. This includes parsing the XML, transforming the XML data and the flow control based on the XPath expressions. To use these tags in JSP, you should have used the following taglib:

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

JSTL Functions

JSTL includes a number of standard functions; most of them are common string manipulation functions. To use these tags in JSP, you should have used the following taglib:

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

JSTL SQL Tags

The JSTL SQL tag library provides tags to access the database such as Oracle, mySQL, or Microsoft SQL Server. To use these tags in JSP, you should be used the following taglib:

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

There are two important JSTL SQL tag such as <sql:setDataSource> and <sql:query> which are described below:

JSTL <sql:setDataSource> tag

This tag is used to create a DataSource configuration that may be stored into a variable that can be used as an input to another JSTL database-action. You can use this tag like the following:

```
<sql:setDataSource  var="name_of_variable" driver="name_of_driver"
        url="jdbc_url"   user=" "  password=" "/>
```

In the <sql:setDataSource> tag, **var** attribute is optional and it specifies the variable's name, which stores a value of the specified data source. The **dataSource** attribute is optional that specifies the data source. The **driver** attribute defines the JDBC driver class name and **url** attribute specifies the JDBC URL associated with the Database. Both the **user** and **password** are the optional attributes that specify the JDBC database user and password name.

JSTL <sql:query> tag

This tag is used for executing the SQL query written into its body or through the sql attribute. For example:

```
<sql:query dataSource="${db}" var="name_of_variable">
  SELECT ProgCode, Prname from programme;
</sql:query>
```

In the <sql:query> tag, **var** is a required attribute that specifies the name of the variable to stores a value of the query result, **sql** attribute specifies the statement of SQL query to execute and **dataSource** is an optional attribute specifies the data source.

4.8 EXCEPTION HANDLING IN JSP

When you are writing a JSP code, there is a chance that you might make coding errors that can occur in any part of the code. In Java Server Pages (JSP), there are two types of errors. The first type of error comes at translation or compilation time when JSP page is translated from JSP source file to Java Servlet class file. The second type of JSP error which is called a request time error occurs during run time or request time. These run time errors result in the form of exception. Exception Handling is the process to handle runtime errors.

There are three ways to perform exception handling in JSP. They are:

- errorPage and isErrorPage attributes of page directive
- <error-page> tag in Deployment Descriptor file
- try and catch block in Scriptlets

4.8.1 Using page directive attributes

You have already learned about the page directive in section 4.5.1 of this Unit. The page directive defines attributes that apply to an entire JSP page. The two attributes of the page directive, such as ‘errorPage’ and ‘isErrorPage’ and one implicit object ‘exception’ are useful in JSP exception handling.

The errorPage attribute

The errorPage attribute of page directive is used to specify the name of error page that handles the exception. The error page contains the exception handling code description for a particular page. The syntax of this attribute is as follows:

<%@ page errorPage=”relative URL” %>

The isErrorPage attribute

The isErrorPage attribute of page directive indicates whether or not the JSP page is an errorPage. The default value of this attribute is false.

<%@ page isErrorPage=”true” %>

For exception handling, you need to create a simple JSP page and write only one additional ***errorPage*** attribute of page directive at the top of the JSP page to make an error page and set its value equal to the location of your JSP error page. In a similar way, you can create another JSP page where an exception may occur. Here, you can put first line of *isErrorPage* attribute of page directive at the top of the JSP page and set its value equal to true and write the second line of code designates where the thrown exception is being used.

For example, you have to create an ‘error.jsp’ named file which contains source code for handling exception and the other page named ‘processPage.jsp’ file, where exception may occur and define the errorPage attribute of page directive. The third file inputPage.jsp is used for input values. This example is for dividing two values and displaying the result.

Source code for inputPage.jsp

```
<html>
<head> <title>InputPage.jsp</title> </head>
 <body>
<form action="processPage.jsp">
Enter Number 1<input type="text" name="num1"><br>
Enter Number 2 <input type="text" name="num2"><br>
<input type="submit" value="submit">
</form>
 </body>
</html>
```

Source code for processPage.jsp

```
<%@ page errorPage="error.jsp" %>
<html>
<head> <title> procesPage.jsp </title> </head>
 <body>
 <%
String n1 = request.getParameter("num1");
String n2 = request.getParameter("num2");

int Var1 = Integer.parseInt(n1);
int Var2 = Integer.parseInt(n2);
int Var3 = Var1 / Var2;

out.println("First number = "+ Var1);
out.println("Second number = "+ Var2);
out.println("Division of two numbers are  "+ Var3);

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

Source code for error.jsp.

```
<%@ page isErrorPage="true" %>
<html>
<head>
 <title>error.jsp</title>
</head>
 <body>
Your page generate an Exception : <br>
<%= exception.getMessage() %>
</body>
</html>
```

Output of the above programs

When you run the above program code, it shows the following two screens one after another. In the first screen (figure-17), there are two input box where you will input two integer values and click on submit button. In the second screen (figure-18), the browser displays the result after the division of two numbers.

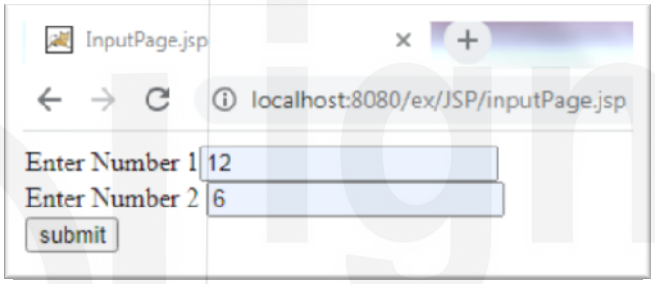

Figure 17: Input Screen for entering values

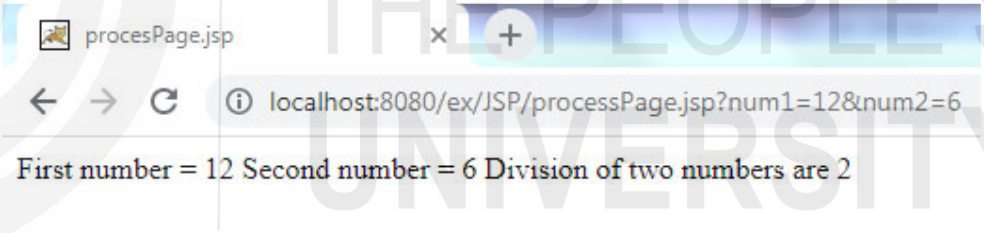

Figure 18: Output Screen for division of two numbers

Now, you will run the above same program again and input any integer value in first text box and zero in second text box and click on submit button. The program will generate an Arithmetic Exception: division by zero. When you will input any float value in any text box, then it will also generate exception.

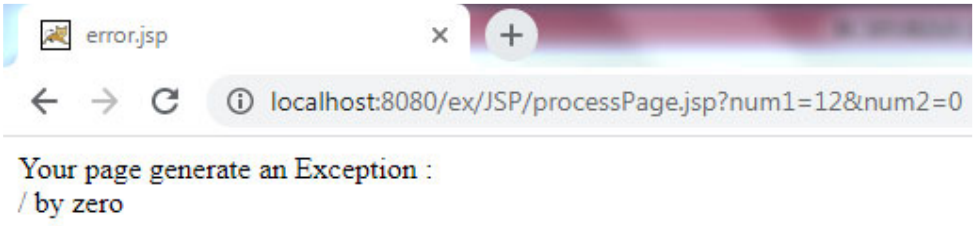

Figure 19: Exception handling Screen

4.8.2 Using try and catch block in Scriptlets

If you want to handle exception within the current JSP page, you can also use the standard java exception handling mechanism in the Scriptlets component of the JSP page. For this, you can use try and catch clause in your source code. In try block, you can write the normal code that might throw an exception. The catch block is used to handle the exception. Both the try and catch clauses are written inside the scriptlets. Unlike page directive, there is no need to write an error page for this mechanism.

You can use the try and catch block like the following manner:

```
<%
        try   {   // code that thrown an exception   }
        catch (exception e) {  // exception handler for exception occur in try block}

%>
```

The following program code is for handling exception using try and catch clause of Java. In this program, normal coding is defined in try block and exception handling code is in catch block. In try block, there are three integer variable x, y and z defined. The variable x contains a value zero and y contains value 10. When the program tries to divide the value of y by x then exception occurs. The program control flow is preceded and displays only those states which are included in catch block.

Source Code for Exception handling through try and catch clause

```
<html>
<head><title> Exception handling through try and catch clause </title></head>
<body>
 <%
  try
        {
                int x = 10;
                x = x/ 0;
                out.println("Output = " + x);
        }
  catch (Exception e)
        {    out.println("Error caught by Catch block : " + e.getMessage());       }
 %>
</body></html>
```

Output of the program:

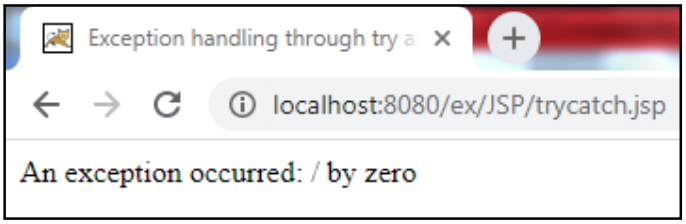

Figure 20: Output Screen for Exception handling through try and catch clause

4.8.3 Using <error-page> tag in Deployment Descriptor file

In the above section, you have learned about the page level exception handling in JSP through page directive and try-catch mechanism in scriptlets. If you want to describe the exception handling in the JSP page at the application level, you can use the <error-

page> tag element in the Deployment Descriptor file. The deployment descriptor is a file named web.xml. It resides in the web applications under the WEB-INF/ directory.

This approach is better as you don't need to use the page directive to declare the error page in each JSP file. Making a single entry in the web.xml file serves the purpose. You can specify either an <*exception-type*> element with the expected exception's class name, such as java.lang.ArithmeticException or <*error-code*> element with an HTTP error code value such as 500 with <location> element. The <location> element states JSP container for path of the resource to show when the error occurs. If you want to handle all the exceptions you need to specify the java.lang.Exception in the exception type element.

To include a generic error page for all exceptions at application level in the following way in web.xml file:

```
<error-page>
  <exception-type>Exception</exception-type>
  <location>/error.jsp</location>
</error-page>
```

If you want to handle exception using any specific error status code such as File not found error 404, Server error 500, then you can specify <error-code> element instead of <exception-type>. This is the best way to declare error page by using error-code element in web.xml file. It is used as under:

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

Here is the example for exception handling at the application level using <error-page> element. The example code is very similar to the above-used example in page directive option. For this example, four files are needed to run the program:

- **inputPage.jsp** file for input values (The code is same as described in the example of page directive section 4.8.1)
- **processPage.jsp** for dividing two numbers and displaying the result. (You can use the same code which is described in the example of page directive section 4.8.1 except the line which contains errorPage attribute of page directive).
- **error.jsp** file for displaying the exception (which is also the same as the example defined in page directive option).
- **web.xml** file for specifying the <error-page> element.

Source code for web.xml: You can include the following code in your web application under the WEB-INF/web.xml file. In this example, error.jsp file is placed under the jsp/error.jsp folder in web application.

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

Now, you can run the program as similar to the example of page directive section 4.8.1. In this way, you can handle the exceptions at page as well as application level.

Note: The page directive declaration overrides any matching error page configurations in the deployment descriptor. Suppose the JSP page throws java.lang.ArithmeticException and deployment descriptor has an exception-type attribute for that exception, the web container invokes the page directive instead of any URI specified in deployment descriptor (web.xml) configuration.

Check Your Progress 3

1. What is the use of JSTL tags in JSP?

   _____

   _____

   _____

   _____

2. What is JSP error page?

   _____

   _____

   _____

   _____

3. Explain the use of deployment descriptor in JSP.

   _____

   _____

   _____

   _____

4.9 DATABASE CONNECTIVITY

Database access is one of the important features of Web application. Java has its own technology for database connectivity called JDBC (Java Database Connectivity). JDBC provides a standard library for accessing a wide range of relational databases like MS-Access, Oracle and Microsoft SQL Server. Using JDBC API, you can access a wide variety of different SQL databases. The core functionality of JDBC is found in java.sql package.

In the previous sections, you have seen examples on Scriptlets and Java Bean where data is fetched from HTML form and displayed on JSP document in a static manner. This section will provide you in-depth knowledge of data access, specifically insertion and retrieval of data to/from a database.

Consider a table named as Student is created in Microsoft SQL Server database with Roll No, Name and Course name. This table is used in both the following sections. This section defines example for storing/retrieving data into/from a Microsoft SQL Server using type-4 JDBC driver. You can run these programs on any web server such as Tomcat. For running these programs, you need a JDBC driver in .jar form and place them in lib directory under the web application.

4.9.1 Insert data in Database using JSP

Let us take an example: a JSP form named “Form.jsp” contains student’s details like student enrolment no., student name and Programme name. When you submit these details it access ‘actionPage.jsp’ document to store data into the Database.

Source Code for Form.jsp

```
<html>
<body>
<form name="form1" method="post" action="actionPage.jsp" >
<h3>Indira Gandhi Open University </h3>
<fieldset>
<legend><b>Student Details </b></legend>
Enrolment No.:<input name="srno" type="text" value="" size=9><br>
Student Name : <input name="sname" type="text" value="" size=15><br>
Programme     : <input type="text" name="prg" value="" size=15> <br>
<input type="submit" value="Submit" />
</fieldset>
</form>
</body>
</html>
```

In the following source code, the first step is to get the data from 'Form.jsp' program using request.getParameter() method. After connecting to Database, an insert query is executed using executeUpdate() method. The program is written by using try and catch clause of standard Java mechanism within JSP scriptlets and data connectivity through the Type-4 driver of SQL Server.

Source Code for actionPage.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="java.sql.*;" %>
<html>
<head><title>Insert data into database</title></head>
<body>
<h3>Insert data in Database using JSP</h3>
<table border=1>
<%
String rollNo = request.getParameter("srno");
String StuName = request.getParameter("sname");
String prgName = request.getParameter("prg");

Connection con = null;   //create connection object
Statement stmt = null;   // create statement object

// connection string using Type-4 Microsoft SQL Server driver
// you can also change the next line with your own environment
String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;DatabaseName=SOCI
S";
try
{
  // load JDBC type-4 SQL Server driver
   Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
  con = DriverManager.getConnection(url);
  if (con != null)
  {
    stmt = con.createStatement();
    //insert query
    String rsql ="insert into student values("+rollNo+","'+StuName+"','"+prgName+"'"+")";
    //execute query
```

```
      stmt.executeUpdate(rsql);
      out.println("Your data is successfully stored in database");
    }
    if(con == null)
    {
      con.close();  // release connection
    }
}
// end of try clause
catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
 %>
```

When you run the 'Form.jsp' program, you will see the following screen (figure-21):

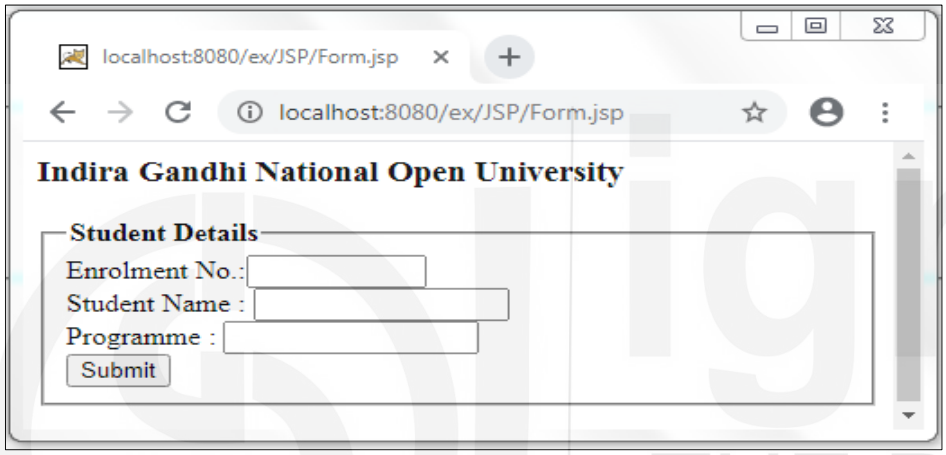

Figure 21: Input Form for storing data into database

In the above screen, when you will enter values then the following screen (figure-22) will show you a message for data storage

Figure-22 : Data stored in persistent storage

4.9.2 Retrieve Data from Database using JSP

The following example gives you an illustration of how to retrieve data from the Database. After execution of the above program, you have stored sufficient data into Database. Now, you will execute the following code for retrieving the data from Database. In this program, one additional ResultSet object is used for retrieving data from select query. The data is retrieved from ResultSet object by using getXXX() method such as getInt() and getString(). Note that if the column name is an integer type, you should use the getInt() method instead of the getString() method.

Source Code for getData.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="java.sql.*;" %>
<html>
<head><title>Retrieved data from database</title></head>
<body>
<h3>  Retrieve Data from Database using JSP</h3>
<table border=1>
<%
Connection con = null;   //create connection object
Statement stmt = null;   // create statement object
ResultSet rs = null;       // create ResultSet object

// connection string using Type-4 Microsoft SQL Server driver
// you can change the next line with your own environment
String url=
"jdbc:microsoft:sqlserver://127.0.0.1:1433;user=sa;password=poonam;DatabaseName
=
SOCIS";
Try
{
  // load sql server JDBC type-4 driver
   Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");

  con = DriverManager.getConnection(url);

  if (con != null)
  {
    stmt = con.createStatement();

    // select SQL statement
    String rsql ="select * from Student";

    //execute query
    rs = stmt.executeQuery(rsql);
     %>
    <tr><td>Roll Number</td><td>Student Name</td><td>Programme</td></tr>
    <%
    while( rs.next() )
    {
      %><tr>
      <td><%= rs.getInt("RollNo") %></td>
      <td><%= rs.getString("Student_Name") %></td>
      <td><%= rs.getString("Programme") %></td>
      </tr>
      <%
    }
  }
if(con == null) {con.close();}
}
catch(SQLException se){ out.print("SQL:"+se.getMessage());}
catch(Exception e){ out.print("Exception:"+e.getMessage());}
 %>
```

After running the above program, the following output screen (figure-23) is displayed:

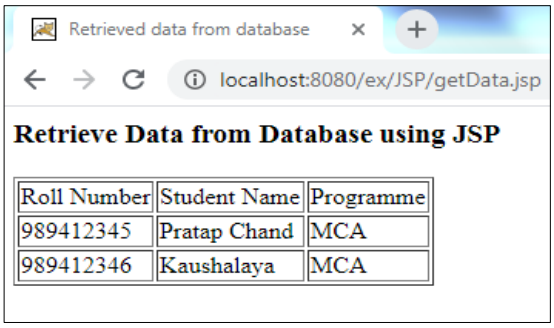

Figure 23: Display data from database

If you want to build your project using Oracle Database as back end, for this you can change only port no and Oracle JDBC driver name in the above source code.

4.10 SUMMARY

In this unit you have gone through the components of 'Java Server Pages' which makes the entire JSP document as directives, scripting and action elements. A 'Java Server Pages' component is a type of Java servlet that is designed to fulfil the role of a user interface for a Java web application. This Unit also introduced you the JSP API on which the entire JSP relies. JSP API is a set of classes and interfaces that is used for making JSP pages. JSP supports nine automatically defined variables which are called implicit objects. When you write a JSP program, you might face a run time error(s). For this JSP provides exception handling through the page directive element, core java mechanism and deployment descriptor file.
Now, you are able to create a JSP document that can be used for a variety of purposes such as retrieving information from a database, passing control between pages and accessing Java Beans components.

4.11 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☞ Check Your Progress 1

1) When a client requests a JSP page, the browser sends a request to web server which is forwarded to the JSP engine. If it is a new request from the client then it is translated and compiled by the JSP engine into a servlet. Now, servlet is ready for servicing the client request and generates response which returns to the browser via web server. For the second time the same request from the client (including browser and web server request) to JSP engine, the JSP engine determines the request that the JSP-Servlet is up-to-date, if yes then it immediately passes control to the respective JSP-Servlet for fulfilling the request

2)
```
<html><body>
<%! int  n = 10, num1 = 0, num2 = 1; %>
<%  for (int  i = 1; i <= n; i++)
        {
          out.print(num1 + " , ");
```

```
            int sum = num1 + num2;
            num1 =num2;
            num2 = sum;
        }
    %>
</body></html>
```

3)
```
<html><body>
<%  int sum=0;
for (int  i = 1; i <=10; i++)
            {  sum = sum+i ;   }
            out.print("sum ="+  sum);
 %>
</body></html>
```

☞ Check your Progress 2

1) Standard actions are tags that affect the runtime behaviour of the JSP. These JSP action tag is used to perform some specific tasks such as insert a file dynamically, reuse external JavaBean components, forward the request to the other page and generate HTML for Java Applet Plugin.

2) JSP provides different scope for sharing data between web pages. These are:

- Page - 'page' scope means the JSP object can be accessed only from within the same page where it is created. By default, it is page. JSP implicit objects out, exception, response, pageContext, config and page have 'page' scope.
- Request – Beans with request scope are accessible only within pages processing the same request that the object was created in. Objects that have request scope are most often used when you need to share information between resources that is pertinent for the current request only. Only Implicit object request has the 'request' scope.
- Session – This object is accessible from any JSP within the same session. Implicit object session has the 'session' scope.
- Application - This object is accessible from any JSP within the same web application. Implicit object application has the 'application' scope.

3) It is used for getting initialization parameters and for sharing the attributes and their values across the entire JSP application. For example, <%=application.getServerInfo()%>, it returns the name and version of the servlet container on which the servlet is running.

☞ Check your Progress 3

1) The 'Java Server Pages' Standard Tag Library (JSTL) contains a set of tags to simplify the JSP development. JSTL provides tags to control the JSP page behaviour and common tasks such as xml data processing, conditionals execution, iteration and SQL tags.

2) A JSP error page is designed to handled runtime errors and display a customized view of the exception. You can include an error page in your application at page or application level. At page level, you can use page directive or standard java mechanism options. At the application level, you can only use an <error page> element of the deployment descriptor.

3) This file is an xml file whose root element is <web-app>. It is reside in the web applications under the WEB-INF/ directory. You can configure JSP tag libraries, welcome files, customizing HTTP error code or exception type. You can use the <error-page> element in the deployment descriptor to specify exception type or HTTP error code and location of the error page. The JSP tag libraries can be defined using the <tag-lib> element of the deployment descriptor.

4.12 REFERENCES/FURTHER READINGS

- Kathy Sierra, Bryan Basham, anddd Bert Bates ,"Head First Servlets and JSP", O'Reilly Media, Inc., 2008.
- Brown-Burdick et al., "Professional JSP", Apress,2001.
- Budi Kurniawan , "Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions: A Developer's Guide to Scalable Solutions" *Techmedia* , 2002.
- James Goodwill , "Pure JSP", Techmedia, 2017.
- https://docs.oracle.com/cd/E17802_01/products/products/jsp/2.1/docs/jsp-2_1-pfd2/javax/servlet/jsp/package-summary.html
- https://docs.oracle.com/javaee/5/api/javax/servlet/jsp/tagext/package-summary.html
- https://docs.oracle.com/javaee/5/jstl/1.1/docs/tlddocs/
- https://tomcat.apache.org/taglibs/index.html
- list of drivers is available at : http://www.devx.com/tips/Tip/28818