Chapter-6.2
JSP Programming

Introduction

- There are basically two problems associated with Servlet technology.
    1. Servlet is difficult to produce a presentable a static HTML page via the println() method.
    2. Programmers, who wrote the servlet, may not be good graphics designer, while a graphics designer may not understand java programming, but to work with servlet needs knowledge of both.
- JSP is a complimentary technology to java servlet which facilitates the mixing of dynamic and static web content.
- Here the main page is written in html, while special tags are providing to insert the statements of java code.
- Using this technology, a programmer can totally focus on business logic while a web designer can focus on the presentation only because business programming logic and presentation are separated.
- JSP did not replace servlets as the technology for writing server-side applications. In fact, JSP was built on the servlet foundation and needs the servlet technology to work.
- **JSP Servlet is HTML inside java while JSP is java inside HTML.**

Advantage of JSP

1. Static HTML portion is separated from dynamic content.
2. Reuse of components and tag libraries.
3. Java power and portability.

How JSP works

- Inside the JSP container is a **special servlet** called the **page compiler.**
- The servlet container is configured to forward to this page compiler all HTTP requests with URLs that match the .jsp file extension.
- This page compiler turns a servlet container into a JSP container.
- When a .jsp page is first called, the page compiler parses and compiles the .jsp page into a servlet class.
- If the compilation is successful, the jsp servlet class is loaded into memory.
- On subsequent calls, the servlet class for that .jsp page is already in memory; however, it could have been updated. Therefore, the page compiler servlet will always compare the timestamp of the jsp servlet with the jsp page.
- If the .jsp page is more current, recompilation is necessary.
- With this process, once deployed, JSP pages only go through the time-consuming compilation process once.

Steps To create first JSP application

- Create a directory under %CATALINA_HOME%/webapps called myJSPApp. Hera catalina_home means tomcat's installed directory.
- Now write a JSP file

```
<HTML>
        <BODY>
        JSP is easy.
        </BODY>
</HTML>
```

- Save this file as SimplePage.jsp in the myJSPApp directory.
- Now, start your web browser, and type the following URL:

  http://localhost:8080/myJSPApp/SimplePage.jsp
- The browser is shown below.

Writing java code in JSP file

- If your page is purely static, we shouldn't put it in a JSP file because JSP files are slower to process than HTML files.
- If you think the code might include Java code in the future. This saves you the trouble of changing all the links to this page at the later stage.
- To write Java code in your JSP file, you embed the code in <% … %> tags.

  For example

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<%    out.println("JSP is easy");  %>
</BODY>
</HTML>
```

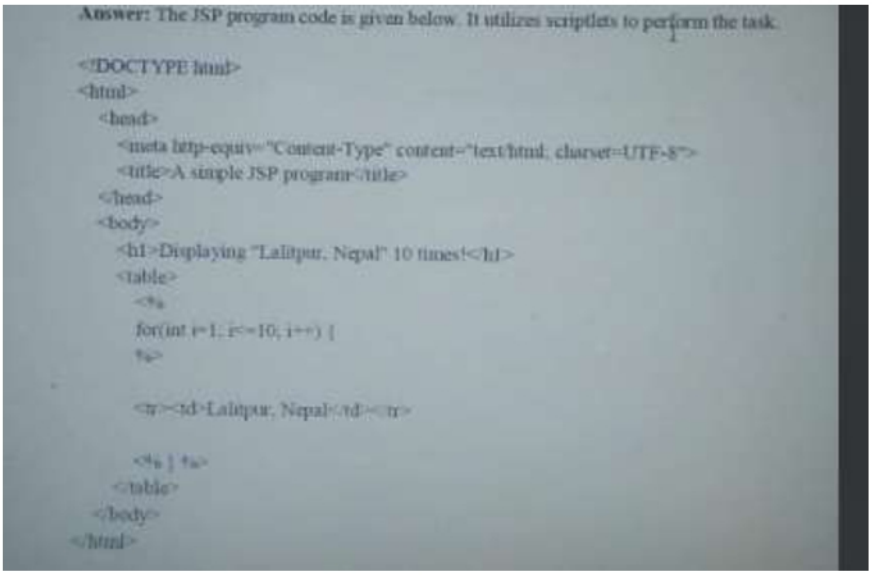
- Save the file as abc.jsp.
- Now, start your web browser, and type the following URL:

  http://localhost:8080/myJSPApp/abc.jsp
- The browser is shown below.

\

- Out is a writer used to write response message to the HTTP response message.

Question:



Write a JSP program to display "Tribhuvan University" 10 times.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<%
        for(int i=0;i<10;i++)
        {
                out.println("Tribhuvan University");
        }
%>
</BODY>
</HTML>
```

3

Write a JSP program to display the current server time.

Jsp program:

```
<HTML>
<BODY>
<%
        java.util.Calendar now =  java.util.Calendar.getInstance();
        int hour = now.get( java.util.Calendar.HOUR_OF_DAY);
        int minute = now.get( java.util.Calendar.MINUTE);
        int second=now.get( java.util.Calendar.SECOND);
%>
<h1>Current Server Time</h1>
<%
        out.println(hour +":"+minute+":"+second);
%>
</BODY>
</HTML>
```

Or

```
<HTML>
<BODY>
<%@ page import="java.util.*" %>    //import java.util.*;

<%
Calendar now =  Calendar.getInstance();
int hour = now.get(Calendar.HOUR_OF_DAY);
int minute = now.get(Calendar.MINUTE);
int second=now.get(Calendar.SECOND);
%>
<h1>Current Server Time</h1>
<%
out.println(hour +":"+minute+":"+second);
%>
</BODY>
</HTML>
```

Output:

Or

```
<HTML>
<BODY>
<%@ page import="java.util.*" %>
<h1>Current Server Time</h1>
<%
Date dt=new Date();
int hr=dt.getHours();
```

```
int min=dt.getMinutes();
int se=dt.getSeconds();
out.println(hr+"hr:"+min+"min:"+se+"sec");
%>
</BODY>
</HTML>
```

Output:

JSP Implicit Object

- There are several object references, such as pageContext, session, application, config, out, and so on.
- These object references are created whether they are used from inside the page or not.
- They are **automatically** available for the JSP page author to use.
- These objects are called implicit objects and are summarized as in the below table.

Table 8.1. JSP Implicit Objects

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

1. request and response object

- In servlets, both objects are passed in by the servlet container to the service method of the javax.servlet.http.HttpServlet class.

```
protected void service(HttpServletRequest request, HttpServletResponse response)    throws Exception
{

}
```

Crete a HTML form with two textfied for entering first name and last name, and a submit button. Create a JSP page that reads the data from html form and displays it.

Html file: index.html

```
<HTML>
    <BODY>
        <FORM  NAME=”myform” ACTION="display.jsp"  METHOD="post">
            First Name: <INPUT TYPE="TEXT" Name="fn">
            <BR/>
            Last Name: <INPUT TYPE="TEXT" Name="ln">
            <BR/>
            <INPUT TYPE="SUBMIT" VALUE="OK">
        </FORM>
```

```
        </BODY>
    </HTML>
```

JSP program

display.jsp

```
    <HTML>
    <BODY>
    <%
        String firstName = request.getParameter("fn");
        String lastName = request.getParameter("ln");
        out.println("First name: " + firstName);
    %>
    <br/>
    <%
        out.println("Last name: " + lastName);
    %>
    </BODY>
    </HTML>
```

Steps.

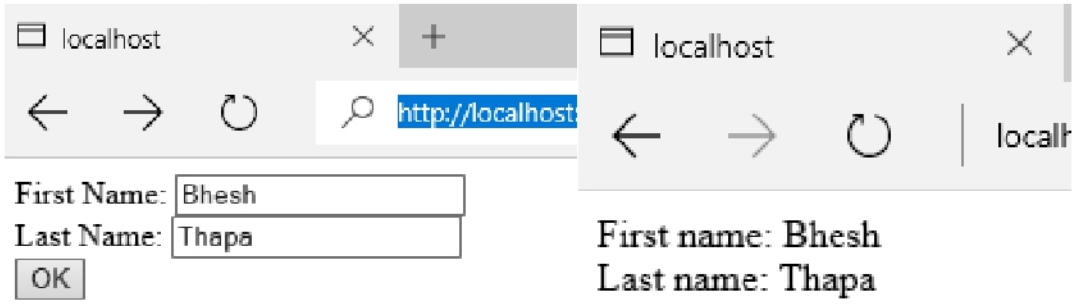
i. Save the html file with name index.html.

ii. Save the JSP file with name display.jsp.

iii. Open the browser and provide the following url

http://localhost:8080/myJSPApp/index.html

iv. Your will see the following outputs.



Write a JSP program to navigate to welcome page welcome.jsp with message "welcome to our site" on clicking submit button if the user input is "Bhesh" for username field and "javaiseasy" for passwordfield.

Welcome.jsp

```
<HTML>
    <BODY>
        <%
            String uname=request.getParameter("txt_uname");
            String pwd=request.getParameter("txt_pwd");
            if(uname.equals("Bhesh")&& pwd.equals("javaiseasy"))
            {
                out.print("Welcome to our site");
            }
            else
            {
                out.print("Invalid login");
            }


        %>
```

```
	</BODY>
</HTML>
```

Index.html

```
<html>
	<body>
		<form name="myfrm" method="post" action="welcome.jsp">
			Name:
			<input type="textbox" name="txt_uname" size=25 >
			Password:
			<input type="password" name="txt_pwd" size=25>
			<input type="submit" value="submit">
		</form>
	</body>
</html>
```

2. out Implicit Object
   - out is probably the most frequently used implicit object. You call either its print method or its println method to send text or other data to the client browser.
   - In a servlet, you always need to call the getWriter method of the javax.servlet.http.HttpServletResponse interface to obtain a PrintWriter before you can output anything to the browser, as follows:

     PrintWriter out = response.getWriter();
   - In JSP, you don't need to do this because you already have an **out** that represents a javax.servlet.jsp.JspWriter object.

3. session Implicit Object
   - The session implicit object represents the HttpSession object that you can retrieve in a servlet by calling the getSession method of the javax.servlet.http.HttpServletRequest interface, as in the following code:

     request.getSession();

     Some useful Methods Methods
     1. Object getAttribute(String attributeName)
     2. void setAttribute(String attributeName, Object object)
     3. void removeAttribute(String objectName)

Create a JSP page that implements session object to implement a counter.

```
	<HTML>
	<BODY>
	 <%
		String counterAttribute = (String) session.getAttribute("counter");
		int count = 0;
		try
		{
			count = Integer.parseInt(counterAttribute);
		}
		catch (Exception e) {    }
		count++;
		session.setAttribute("counter", Integer.toString(count));
		out.println("This is the " + count + "th time you visited this page in this  session.");
	%>
	</BODY>
```

```
</HTML>
```

Output:

This output was obtained when visited the save page for 16th time.

Create a login page that allows user to input username and password. When user presses the submit button, validate the user login and redirect to welcome.jsp page that shows the welcome message. If the user directly enters the URL of welcome.jsp page, it must automatically navigate to login page. [username:Ram Password: 1111]

Index.html

```
<html>
    <body>
    <form name="myfrm" method="post" action="login.jsp">
        Name:
        <input type=textbox name="txt_uname" size=25 >
        Password:
        <input type=password name="txt_pwd" size=25>
        <input type=submit value="submit">
    </form>
    </body>
</html>
```

Login.jsp

```
<%
    String uname=request.getParameter("txt_uname");
    String pwd=request.getParameter("txt_pwd");
    if(uname.equals("Ram")&& pwd.equals("1111"))
    {
        session.setAttribute("UserName", uname);
        response.sendRedirect("welcome.jsp");
    }
    else
    {
        out.println("Invalid Login");
    }
%>
```

Welcome.jsp

```
<HTML>
<BODY>
<%
    String uname = (String) session.getAttribute("UserName");
    if(uname==null)
    {
        response.sendRedirect("index.html");
    }
    //session.removeAttribute("UserName");
```

```
        //session.invalidate(); //for invalidate the current session
%>
 Welcome to our site
 </BODY>
</HTML>
```

4. application
   - An application on the Web may consists of several JSP files that work together to perform some purpose. The Application object is used to tie these files together.
   - The Application object is used to store and access variables from any page, just like the Session object. The difference is that ALL users share ONE Application object (with Sessions there is ONE Session object for EACH user).
   - The Application object holds information that will be used by many pages in the application (like database connection information). The information can be accessed from any page. The information can also be changed in one place, and the changes will automatically be reflected on all pages.
   - The application implicit object represents the javax.servlet.ServletContext object.
   - In an HttpServlet, you can retrieve the ServletContext method by using the getServletContext method.

   Some Methods are

   4. Object getAttribute(String attributeName)
   5. void setAttribute(String attributeName, Object object)
   6. void removeAttribute(String objectName)

Program to demonstrate user visit counter using application object
File: a.jsp

```
<HTML>
<BODY>
 <%
String counterAttribute = (String) application.getAttribute("counter");
int count = 0;
try
{
count = Integer.parseInt(counterAttribute);
}
catch (Exception e) {   }
count++;
application.setAttribute("counter", Integer.toString(count));
out.println("Total Visiter Counter= "+count);
%>
</BODY>
</HTML>
```

5. config
   - The config implicit object represents a **javax.servlet.ServletConfig** object that in a servlet can be retrieved by using the getServletConfig method.
   - This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.

Methods of config object

1. String getInitParameter(String paramname)
2. String getServletName() – It returns the name of the servlet which we define in the web.xml file inside <servlet-name> tag.

Index.html

```
<form name="MyForm" action="runjsp">
	<input type="text" name="uname">
	<input type="submit" value="Submit"><br/>
</form>
```

Web.xml

```
<?xml version="1.0" encoding ="ISO-8859-1"?>
<web-app>
	<servlet>
		<servlet-name>Example909</servlet-name>
		<jsp-file>/welcome.jsp</jsp-file>
		<init-param>
			<param-name>DriverName</param-name>
			<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
		</init-param>
		<init-param>
			<param-name>DriverType</param-name>
			<param-value>Type-1</param-value>
		</init-param>
	</servlet>
	<servlet-mapping>
		<servlet-name>Example909</servlet-name>
		<url-pattern>/runjsp</url-pattern>
	</servlet-mapping>
</web-app>
```

Welcome.jsp

```
<%
	String driver_name=config.getInitParameter("DriverName");
	String driver_type=config.getInitParameter("DriverType");
	String servlet_name=config.getServletName();
	out.print("Welcome "+request.getParameter("uname"));
	out.println("<br/>");
	out.print("Driver name is="+driver_name);
	out.println("<br/>");
	out.print("Driver type is="+driver_type);
	out.println("<br/>");
	out.print("Servlet Name is="+ servlet_name);
%>
```

Note: Save the web.xml file in WEB-INF folder. Similarly, save the html and jsp file in same director as of WEB-INF.

6. pageContext
   - The pageContext implicit object represents the javax.servlet.jsp.PageContext object .
   - The PageContext class provides methods that are used to create other objects.
   - For example, its getOut() method returns a JspWriter object that is used to send strings to the web browser.
   - Other methods that return servlet-related objects include the following:
     - getRequest, returns a ServletRequest object
     - getResponse, returns a ServletResponse object
     - getServletConfig, returns a ServletConfig object
     - getServletContext, returns a ServletContext object
     - getSession, returns an HttpSession object

Example:

Index.html

```
<form name="MyForm" action="welcome.jsp">
        <input type="text" name="uname">
        <input type="submit" value="Submit"><br/>
</form>
```

welcome.jsp

```
<%
        ServletRequest req=pageContext.getRequest();
        String uname=req.getParameter("uname");
        JspWriter o=pageContext.getOut();
        o.print("UserName="+uname);
%>
```

7. page
   - This is simply a synonym for this, and is used to call the methods defined by the translated servlet class. It can be thought of as an object that represents the entire JSP page.
   - This object is assigned to the reference of auto generated servlet class
   - It is written as:

     Object page=this;
   - Since page is a variable of type Object, it cannot be used to directly call the servlet methods like getServletInfo(), getServletConfig(), destroy() etc.
   - **To access any of the methods of the servlet through page it must be first cast to type Servlet.**

Example:

```
<%
        out.print(((Servlet)page).getServletInfo());
%>
```

8. exception
   - The exception object is available only on pages that have been defined as error pages.
   - It is typically used to generate an appropriate response to the error condition.

Java Directive

- The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- Types of directives:
  - page directive
  - include directive

Syntax:

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

Page Directive

- The page directive defines attributes that apply to an entire JSP page.
  Syntax:

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

- Some of the Attributes of JSP page directive
  1. Import : The import attribute is used to import class, interface or all the members of a package. It is similar to import keyword in java class or interface.

Example:

```
<html>
<body>
<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>
</body>
</html>
```

  2. contentType: The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type i.e. media type like document, file etc. of the HTTP response. Example: `audio/mpeg, application/zip. image/jpeg, text/html etc.`

Example:

```
<html>
<body>
<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %></body>
</html>
```

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

Example:

```
<html>
<body>
<%@ page info="Bhesh Thapa" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

  4. errorPage: The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.
     Example: Will be discussed shortly
  5. isErrorPage : The isErrorPage attribute is used to declare that the current page is the error page.

Example: will be discussed shortly

Exception Handling using exception object

- The exception object is an instance of a subclass of Throwable and is only available in error pages.
- In order to handle exception in JSP we need to first define an error page with necessary error response message. The error page is defined by using **page directive** by setting **isErrorPage** attribute value to **true** as below.
- The isErrorPage attribute can accept the value of "true" or "false", and the default value is "false". It indicates whether the current JSP page is an error page; that is, the page that will be displayed when an uncaught exception occurs in the other JSP page. If the current page is an error page, it has access to the **exception** implicit object.

ErrorPage.jsp File

```
<%@ page isErrorPage="true" %>
    <html>
        <body>
            OOPs error occured.
        </body>
    </html>
<%
    out.println(exception);
%>
```

- JSP gives us an option to specify Error Page for each JSP. Whenever the page throws an exception, the JSP container automatically invokes the error page.
- Following is an example to specifiy an error page for a main.jsp. To set up an error page, we declare page directive with errorPage attribute set to the ulr of specified error handle page as below

<%@ page errorPage = "xxx" %>

Example:

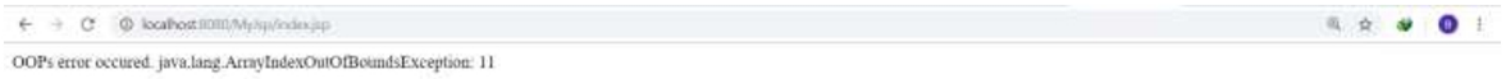
Main.jsp with error

```
<%@ page errorPage="ErrorPage.jsp" %>
<%
    int data[]=new int[10];
    data[11]=5;     //will generate exception
%>
```

ErrorPage.jsp file

```
<%@ page isErrorPage="true" %>
<html>
<body>
    OOPs error occured.
</body>
</html>
<%
    out.println(exception);
%>
```

Output:



Include Directive

- This directive enables JSP page authors to include the contents of other files in the current JSP page.
- The include directive is useful if you have a common source that will be used by more than one JSP page.
- Instead of repeating the same code in every JSP page, thus creating a maintenance problem, you can place the common code in a separate file and use an include directive from each JSP page.
- The included page itself can be static, such as an HTML file, or dynamic, such as another JSP page.

<%@ include file="relativeURL" %>

Example:

A Simple JSP Page that Includes Two Files

```
<%@ page import="java.util.Calendar" %>
<%@ include file="includes/Header.html" %>
 <%
        out.println("Current time: " + Calendar.getInstance().getTime());
%>
<%@ include file="includes/Footer.html" %>
```

The Header.html File

```
<HTML> <HEAD> <TITLE>Welcome</TITLE> <BODY>
```

The Footer.html File

```
</BODY> </HTML>
```

Scripting Element

- Scripting elements allow you to insert Java code in your JSP pages.
- There are three types of scripting elements:

1. Scriptlets : allows us to insert any number of valid java statements into the service() method.

Example:

```
<%
        out.println("java is easy");
%>
```

2. Declarations : Declarations allow you to declare methods and variables that can be used from any point in the JSP page.

Syntax:

<%! declare var/function %>

Example:

Index.jsp file

```
<%@ page import="java.util.*" %>
<%!
```

```
        String getSystemDateTime()
        {
                Date d=new Date();
                return(d.toString());
        }
%>
<%
        out.println("Current Date and Time: " + getSystemDateTime());
%>

<%!
        int i=5;
        //out.println(i); wrong
%>
<%
        int j=7;
        i++;
        out.println(i+j);
%>
```

Output:

- Write JSP page to define function **double getArea(double, double)** to calculate area of rectangle.

```
<%!
        public double getArea(double len, double bre)
        {
                double area=len*bre;
                return(area);
        }
%>

<%
        out.println("Area of Rectangle="+getArea(5.5,6.6));
%>
```

3. Expressions
   - The code placed within **JSP expression tag** is *written to the output stream of the response*. So we need not write out.print() to write data. It is mainly used to print the values of variable or method.
   - Expressions get evaluated when the JSP page is requested and their results are converted into a String and fed to the print method of the out implicit object.
   - If the result cannot be converted into a String, an error will be raised at translation time.
   - If this is not detected at translation time, at request-processing time, a ClassCastException will be raised.
   - An expression starts with a <%= and ends with a %>. You don't add a semicolon at the end of an expression.

Example:

Index.jsp

```
<%
        String name="Bhesh";
```

```
%>
<%=name %>
```

The expression is equivalent to the following scriptlet:

```
`
<%
        String name="Bhesh";
        out.print(name);
%>
```

- As you can see, an expression is shorter because the return value is automatically fed into the out.print.

Write JSP page that uses expression tag to print username entered by user.

Html file: index.html

```
<html>
    <body>
        <form action="welcome.jsp">
            <input type="text" name="uname"><br/>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

JSP file: welcome.jsp

```
<html>
    <body>
        <%= "Welcome "+request.getParameter("uname") %>
    </body>
</html>
```

A database db_student consists of table tbl_student having fields id,name address and phone number.

i). Create a suitable user interface that allow users to input the data for a student. When the user press submit button pass these information to the JSP for storing these data in the table tbl_student.[Assume DSN: NAG]

Html file

```
<html>
<body>
<form name=”myform” method="post" action="db.jsp">
ID:
<input type="text" name="id">
<br>Name:
<input type="text" name="name">
<br>Address:
<input type="text" name="add">
<br>Phone Number:
<input type="text" name="phone">
<br>
<input type="submit" value="Save">
</form>
</body>
</html>
```

db.jsp file

```
<%@ page import="java.sql.*" %>
<%
        String id=request.getParameter("id");
        String name=request.getParameter("name");
        String add=request.getParameter("add");
        String ph=request.getParameter("phone");

        try
        {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  //Type-1, Recommended to use Type-4
                String url = "jdbc:odbc:Nag";
                Connection conn = DriverManager.getConnection(url,"","");
                Statement st = conn.createStatement();
                st.executeUpdate("insert into tbl_student values ('"+id+"','"+name+"','"+add+"','"+ph+"')");
                out.println("Record insert successfull");
        }
        catch (Exception e)
```

```
        {
                out.println("Error Occured"+e);
        }
%>
```

ii.Create a JSP page that connects to the database db_student and displays all the records stored in the table tbl_student.

JSP file

```
<%@ page import="java.sql.*" %>
<html>
<body bgcolor=orange>
<TABLE border=1>
<TR>
 <TH>ID</TH>
 <TH>Name</TH>
 <TH>Address</TH>
 <TH>PhoneNumber</TH>
</TR>
<%
        try
        {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                String url = "jdbc:odbc:Nag";
                Connection conn = DriverManager.getConnection(url,"","");
                Statement st = conn.createStatement();
                Statement stmt = conn.createStatement();
                ResultSet rs;
                rs=stmt.executeQuery("Select * from tbl_student");
                 while (rs.next())
                {
                  out.println("<TR>");
                        out.println("<TD>" + rs.getString(1) + "</TD>");
                    out.println("<TD>" + rs.getString(2) + "</TD>");
                   out.println("<TD>" + rs.getString(3) + "</TD>");
                    out.println("<TD>" + rs.getString(4) + "</TD>");
                    out.println("</TR>");
                }
                rs.close();
                stmt.close();
                conn.close();
        }
        catch (Exception e)
        {
                out.println("Error Occured"+e);
```

```
            }
        %>
        </TABLE>
        </body>
        </html>
```

Output:

Assignment

1. Differentiate between JSP vs ASP
2. Differentiate between JSP and servlet.
3. Write JSP prage to define function **double getArea(double, double)** to calculate area of rectangle. The length and breadth values must be pass from html form.