USN | | | | | | | | | | |

CMR INSTITUTEOF TECHNOLOGY

Internal Assessment Test 1–Nov.2025

Sub:	Web Programming using Java							Sub Code:	MMCB311F
Date:	7/11/2025	Duration:	90min's	Max Marks:	50	Sem & Sec:	III B	Branch:	MCA

Note: Answer FIVE FULL Questions, choosing ONE full question from each Module

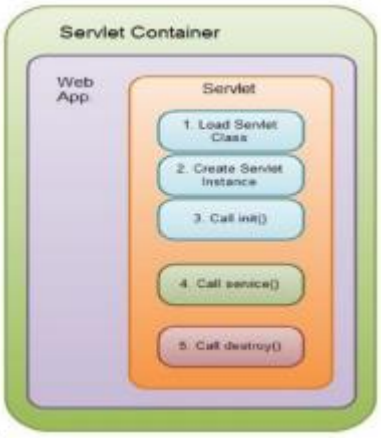
		MARKS	OBE	
			CO	RBT
	PART I			
1	Explain the life cycle of a servlet with an example.	10	CO2	L2
	OR			
2.	Write a JAVA Servlet Program to create a cookie with user preferences and to display.	10	CO2	L4
	PART II			
3	What do you mean by session handling? Explain with a neat diagram.	10	CO2	L2
	OR			
4.	Write a Servlet program to read data from a HTML form (gender data from radio buttons and colours data from check boxes) and display	10	CO2	L4
	PART III			
5.	Explain the steps for creating cookies with an example.	10	CO2	L2
	OR			
6.	Explain Jsp architecture with a neat diagram.	10	CO3	L2
	PART IV			
7.	What are different types of tags in JSP demonstrate with an example.	10	CO3	L2
	OR			
8.	What is JSP? Explain the advantages of JSP over Servlet.	10	CO3	L2
	PART V			
9.	Explain JSTL library and its usage in detail.	10	CO3	L2
	OR			
10.	What are the different implicit objects in JSP? Explain with an example.	10	CO3	L2

1. **Explain the life cycle of a Servlet with an example.**

A. Servlets are programs that run on a Web or Application server act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
Servlets are server side components that provide a powerful mechanism for developing web applications.

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet
The servlet is initialized by calling the init () method.
The servlet calls service() method to process a client's request.
The servlet is terminated by calling the destroy() method.
Finally, servlet is garbage collected by the garbage collector of the JVM.

S.



The init() method :

- The init method is designed to be called only once.
- It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException {
// Initialization code...
}
```

The service() method :

- The service() method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Signature of service method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
{
}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc.methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
// Servlet code
}
```

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet.

This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection.

The destroy method definition looks like this:

```
public void destroy() {
// Finalization code...
}
```

2. **Write a java Servlet program to create a Cookie with user preferences and to display.**

```
<!DOCTYPE html>
<html>
<head>
   <title>User Preferences</title>
</head>
<body>
   <h2>Set Your Preferences</h2>
   <form action="PreferenceServlet" method="post">
      <label><b>User Name:</b></label><br>
      <input type="text" name="username" required><br><br>

      <label><b>Preferred Theme:</b></label><br>
      <select name="theme">
         <option value="Light">Light</option>
         <option value="Dark">Dark</option>
         <option value="Blue">Blue</option>
      </select><br><br>

      <input type="submit" value="Save Preferences">
   </form>
</body>
</html>
```

Preference Servlet:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PreferenceServlet extends HttpServlet {

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String username = request.getParameter("username");
      String theme = request.getParameter("theme");
```

```
        // Create cookies
        Cookie userCookie = new Cookie("username", username);
        Cookie themeCookie = new Cookie("theme", theme);

        // Set expiry time (1 day)
        userCookie.setMaxAge(24 * 60 * 60);
        themeCookie.setMaxAge(24 * 60 * 60);

        // Add cookies to response
        response.addCookie(userCookie);
        response.addCookie(themeCookie);

        out.println("<html><body>");
        out.println("<h2>Preferences Saved!</h2>");
        out.println("<p><b>User Name:</b> " + username + "</p>");
        out.println("<p><b>Preferred Theme:</b> " + theme + "</p>");
        out.println("<p><a href='DisplayPreferences'>View Stored Preferences</a></p>");
        out.println("</body></html>");
    }
}
```

Display Preferences:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DisplayPreferences extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Cookie[] cookies = request.getCookies();

        String username = "Not Found";
        String theme = "Not Found";

        if (cookies != null) {
            for (Cookie c : cookies) {
```

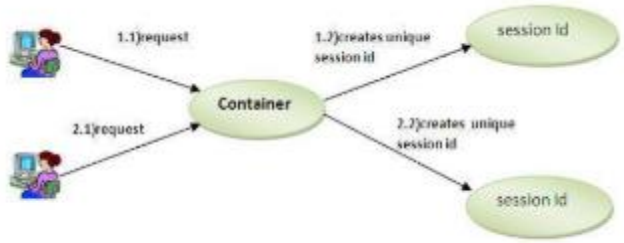
```
            if (c.getName().equals("username")) {
                username = c.getValue();
            }
            if (c.getName().equals("theme")) {
                theme = c.getValue();
            }
        }
    }

    out.println("<html><body>");
    out.println("<h2>Stored User Preferences</h2>");
    out.println("<p><b>User Name:</b> " + username + "</p>");
    out.println("<p><b>Preferred Theme:</b> " + theme + "</p>");
    out.println("</body></html>");
  }
}
```

3. **What do you mean by Session Handling? Explain with a neat diagram.**

container creates a session id for each user when communicated to the server.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?
The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession
In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.
index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    HttpSession session=request.getSession();
```

```
        session.setAttribute("uname",n);

        out.print("<a href='servlet2'>visit</a>");

        out.close();

            }catch(Exception e){System.out.println(e);}
    }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session=request.getSession(false);
        String n=(String)session.getAttribute("uname");
        out.print("Hello "+n);

        out.close();

            }catch(Exception e){System.out.println(e);}
    }


}
```

4. **Write a Servlet program to read data from a HTML form (gender data from radio buttons and colours from check box) and display.**

```
<!DOCTYPE html>
<html>
<head>
    <title>Gender and Color Selection</title>
</head>
<body>
    <h2>Choose Your Gender and Favorite Colors</h2>
    <form action="DisplayDataServlet" method="post">
```

```
        <label><b>Gender:</b></label><br>
        <input type="radio" name="gender" value="Male"> Male<br>
        <input type="radio" name="gender" value="Female"> Female<br><br>

        <label><b>Favorite Colors:</b></label><br>
        <input type="checkbox" name="color" value="Red"> Red<br>
        <input type="checkbox" name="color" value="Green"> Green<br>
        <input type="checkbox" name="color" value="Blue"> Blue<br>
        <input type="checkbox" name="color" value="Yellow"> Yellow<br><br>

        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Servlet Code:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DisplayDataServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String gender = request.getParameter("gender");
        String[] colors = request.getParameterValues("color");

        out.println("<html><body>");
        out.println("<h2>Submitted Information</h2>");

        if (gender != null)
            out.println("<p><b>Gender:</b> " + gender + "</p>");
        else
            out.println("<p><b>Gender:</b> Not Selected</p>");

        if (colors != null) {
            out.println("<p><b>Selected Colors:</b></p>");
            out.println("<ul>");
            for (String color : colors) {
```

```
                out.println("<li>" + color + "</li>");
            }
            out.println("</ul>");
        } else {
            out.println("<p><b>Selected Colors:</b> None</p>");
        }

        out.println("</body></html>");
        out.close();
    }
}
```

5. **Explain the steps for creating Cookies with an example.**

cookies are small bits of textual information that a web server sends to a browser and that the browser later returns unchanged when visiting the same web site or domain

Sending cookies to the client:

1.Creating a cookie object: Create an object for the cookie class

• Cookie():constructs a cookie.

• Cookie(String name, String value)constructs a cookie with a specified name and value.

EX:

```
Cookie ck=new Cookie("user",”mca");
```

2.Setting the maximum age

setMaxAge() is used to specify how long (in seconds) the cookie should be valid.

Ex:cookie.setMaxAge(60*60*24);

3.Placing the cookie into the HTTP response headers.

We use response.addCookie to add cookies in the HTTP response header as follows:

response.addCookie(cookie);

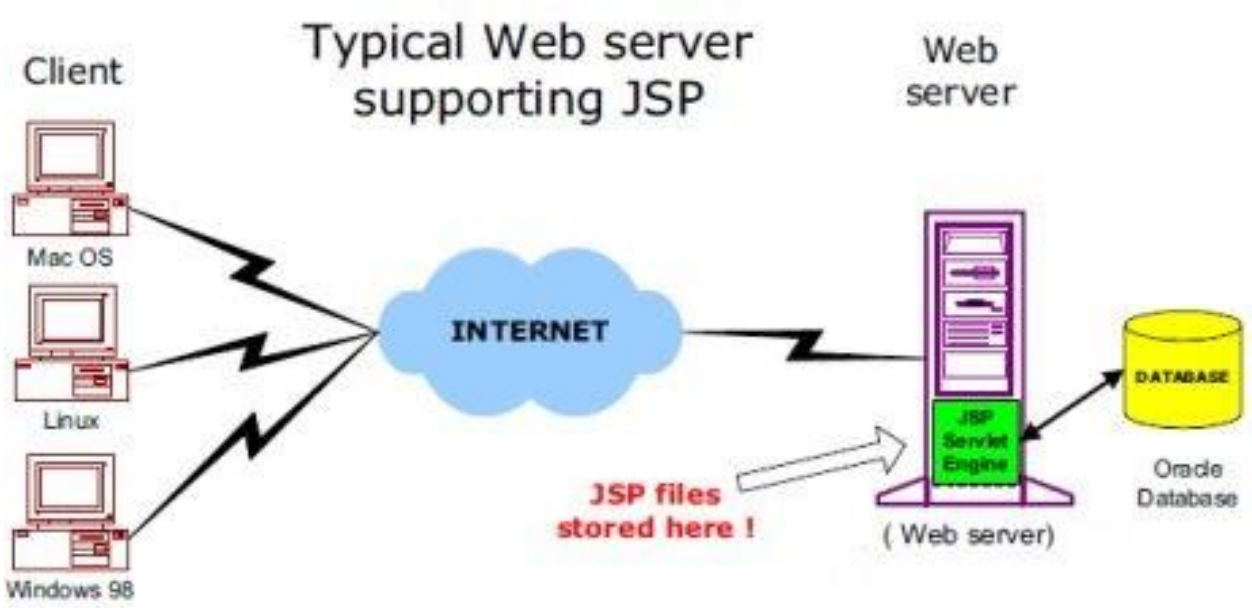
Reading cookies from the client:

1. Call request.getCookies(). This yields an array of cookie objects.
2. Loop down the array, calling getName on each one until you find the cookie of interest.

Ex:

```
String cookieName=“userID”;
Cookie[] cookies=request.getCookies();
If(cookies!=null)
{
for(int i=0;i<cookies.length;i++){
Cookie cookie=cookies[i];
if(cookieName.equals(cookie.getName())){
doSomethingwith(cookie.getValue());
}}}
```

6. **Explain Jsp architecture with a neat diagram.**



JSP Processing:

1. Browser sends an HTTP request to the web server.
2. The web server recognizes that the HTTP request is for a JSP page **(.jsp page**) and forwards it to a JSP engine.
3. The JSP engine loads the JSP page from disk and converts it into a servlet content. All template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
4. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
5. Servlet engine loads the Servlet class and executes it. & produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
6. The web server forwards the HTTP response to your browser in terms of static HTML content.

7. **What are different types of tags in JSP demonstrate with an example.**

JSP scriptlet tag:A scriptlet tag java source code in JSP.

<% java source code %>

In this example, we are displaying a welcome message.

<html>

<body>

<% out.print(“Welcome to jsp” %>

</body>

</html>

JSP Declaration Tag: The JSP declaration tag is used to declare variables, objects and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

<%! Field or method declaration %>

Ex:

<html>

<body>

<%! Int data=50; %>

<%= “value “ +data %>

</body>

</html>

JSP Expression Tag:

Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the out.print() method.

Syntax :<%= java expression %>

<%= (2*5) %>

JSP Directives:

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

Syntax

<%@ directive attribute=”value” %>

There are three types of directives: 1. import 2. include directive directive 3. taglib directive

8. **What is JSP? Explain the advantages of JSP over Servlet.**

- It is Extension to Servlet Technology.
- It has all the features of servlet and additionally has implicit objects, predefined tags, custom tags
- It is easily managed. It separates business logic with presentation logic
- It can be easily deployed
- If JSP page is modified, no need to redeploy but if changes are required in servlet , the entire code needs to be updated and recompile

JSP	**Servlet**
JSP is a web page scripting language that can generate dynamic content	**Servlets are Java programs that already compiled which also creates dynamic web content**
JSP runs slower compared to servlet as it takes compilation time to convert into servlets	**Servlets run faster compared to JSP.**
It’s easier to code in JSP than in Java	**It is not easier when compared to JSP**
Servlets.	
In MVC, jsp act as a view.	**In MVC, servlet act as a controller.**
JSP are generally preferred when there is not much processing of data required.	**servlets are best for use when there is more processing and manipulation involved.**
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	**There is no such custom tag facility in servlets.**
We can achieve functionality of JSP at client side by running JavaScript at client side.	**There are no such methods for servlets.**

9. **Explain JSTL library and its usage in detail.**

- Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

Advantage of JSTL

- Fast Developement JSTL provides many tags that simplifies the JSP.
- Code Reusability We can use the JSTL tags in various pages.

No need to use scriptlet tag It avoids the use of scriptlet tagThe JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

- Core Tags
- Formatting tags
- SQL tags
- XML tags
- JSTL Functions

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is **http://java.sun.com/jsp/jstl/core** . The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting etc. The url for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml**and prefix is **x**.
SQL tags	The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

10. What are the different implicit objects in JSP? Explain with an example.

JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out.

It is the object of JspWriter.

- In case of servlet you need to write:

  PrintWriter out=response.getWriter();

- But in JSP, you don't need to write this code.

E**xample of out implicit object**

- <html>
- <body>
- <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
- </body>
- </html>

Request :

- The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.
- It can also be used to set, get and remove attributes from the jsp request scope.

index.html

- <form action="welcome.jsp">
- <input type="text" name="uname">
- <input type="submit" value="go">
- </form>
- welcome.jsp
- <%
- String name=request.getParameter("uname");
- out.print("welcome "+name); %>

Response Object :

- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

index.html

- <**form** action="welcome.jsp">
- <**input** type="text" name="uname">
- <**input** type="submit" value="go"><**br/**>
- </**form**>
- **welcome.jsp**
- <%
- response.sendRedirect("http://www.google.com");
- %>

Config:

- In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.
- **index.html**

<**form** action="welcome">

<**input** type="text" name="uname">

<**input** type="submit" value="go"><**br/**>

</**form**>

- **web.xml file**

<**web-app**>

<**servlet**>

<**servlet-name**>sonoojaiswal</**servlet-name**>

<**jsp-file**>/welcome.jsp</**jsp-file**>

<**init-param**>

<**param-name**>dname</**param-name**>

<**param-value**>sun.jdbc.odbc.JdbcOdbcDriver</**param-value**>

</**init-param**>

</**servlet**>

<servlet-mapping>

<servlet-name>sonoojaiswal**</servlet-name>**

<url-pattern>/welcome**</url-pattern>**

</servlet-mapping>

</web-app>

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

Application:

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

Example

- **index.html**

  <form action="welcome"**>**

  input type="text" name="uname"**>**

  <input type="submit" value="go"**>**

  </form>

- **welcome.jsp**

  <%

  out.print("Welcome "+request.getParameter("uname"));

  String driver=application.getInitParameter("dname");

  out.print("driver name is="+driver);

  %>

- **<web-app>**
- **<servlet>**
- **<servlet-name>**sonoojaiswal**</servlet-name>**
- **<jsp-file>**/welcome.jsp**</jsp-file>**
- **</servlet>**
- **<servlet-mapping>**
- **<servlet-name>**sonoojaiswal**</servlet-name>**
- **<url-pattern>**/welcome**</url-pattern>**
- **</servlet-mapping>**
- **<context-param>**
- **<param-name>**dname**</param-name>**
- **<param-value>**sun.jdbc.odbc.JdbcOdbcDriver**</param-value>**
- **</context-param>**
- **</web-app>**

Session:

- In JSP, session is an implicit object of type HttpSession. The Java developer can use this object to set,get or remove attribute or to get session information.
- **index.html**
- <html>
- <body>
- <form action="welcome.jsp">
- <input type="text" name="uname">
- <input type="submit" value="go">
- </form>
- </body>

</html>

Example

- **welcome.jsp**
- <html>
- <body>
- <%
- String name=request.getParameter("uname");
- out.print("Welcome "+name);
- session.setAttribute("user",name);
- second jsp page
- %>
- </body>
- </html>

- **second.jsp**
- <html>
- <body>
- <%
-
- String name=(String)session.getAttribute("user");
- out.print("Hello "+name);
-
- %>
- </body>
- </html>

Pagecontext:

- In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:page
- request
- session
- application
- In JSP, page scope is the default scope.
- **index.html**
- <html>
- <body>
- <form action="welcome.jsp">
- <input type="text" name="uname">
- <input type="submit" value="go">
- </form>
- </body>
- </html>

- **welcome.jsp**
- <html>
- <body>
- <%
-
- String name=request.getParameter("uname");
- out.print("Welcome "+name);
-
- pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
-
- second jsp page
-
- %>
- </body>
- </html>

- **second.jsp**
- <html>
- <body>
- <%
-
- String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
- out.print("Hello "+name);
-
- %>
- </body>
- </html>

Page:

- In JSP, page is an implicit object of type Object class.
- This object is assigned to the reference of auto generated servlet class.
- It is written as:Object page=this;
- For using this object it must be cast to Servlet type.
- For example:<% (HttpServlet)page.log("message"); %>
- Since, it is of type Object it is less used because you can use this object directly in jsp.For example:<% this.log("message"); %>

Exception:

- In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example:
- **error.jsp**
- <%@ page isErrorPage="true" %>
- <html>
- <body>
-
- Sorry following exception occured:<%= exception %>
-
- </body>
- </html>