CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 1 – Dec. 2022

Sub:	**Advances in Java**							**Sub Code:**	**20MCA33**
Date:	**27/12/22**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**III**	**Branch:**	**MCA**

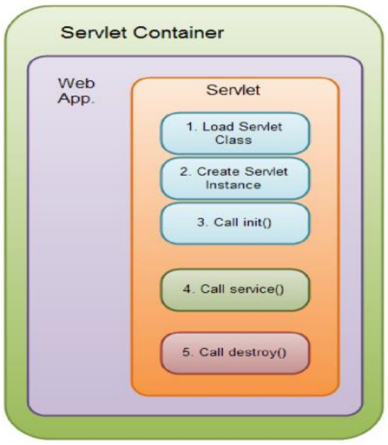
Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
	PART I		CO	RBT
1	Define Servlet. Explain the basic servlet structure and its life cycle methods.	10	CO1	L2
	OR			
2	Write a servlet application to establish communication between html and servlet page using hyperlink to wish Good Morning or afternoon or evening or night based on the time.	10	CO1	L4
	PART II			
3	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	CO1	L4
	OR			
4.a.	Narrate the major range of http status codes along with their purpose	05	CO1	L1
b.	List and explain the different HTTP 1.1 request headers and its methods	05	CO1	L2
	PART III			
5	Explain different types of session tracking techniques with example	10	CO1	L2
	OR			
6.a.	Write the differences between JSP and servlets.	05	CO2	L2
b.	What are the need, benefit and advantages of JSP	05	CO2	L1
	PART IV			
7	Write a servlet program using cookies to remember user preferences.	10	CO1	L4
	OR			
8a	What is cookie and Explain the advantages of cookies.	5	CO1	L3
b	Explain the working of cookie in java with code snippets	5	CO1	L2
	PART V			
9	Write a Servlet application to count the total number of visits on your website.	10	CO1	L2
	OR			
10	Explain the different types of JSP tags with an example.	10	CO2	L1

1. **Define Servlet. Explain the basic servlet structure and its life cycle methods.**

- Java 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.

Now let us discuss the life cycle methods in details.

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 servlet application to establish communication between html and servlet page using hyperlink to wish Good Morning or afternoon or evening or night based on the time.
index.html

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Wishing message</title>
    </head>
    <body>
        <a href = "http://localhost:2020/WishSrvApp/test">Get wishing</a>
    </body>
</html>
```

WishApp.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class WishApp extends HttpServlet
{
    public void service(HttpServletRequest req, HttpServletResponse res) throws IOException,
ServletException
    {
        //set response content type
        res.setContentType("text/html");
        //get printWrite obj
        PrintWriter pw = res.getWriter();
        //write request processing logic to generate wish message
        Calendar cal = Calendar.getInstance();
        //get current hours of the day
        int hour = cal.get(Calendar.HOUR_OF_DAY);//24 hrs format
        //generate wish message
        if(hour<12)
            pw.println("Good Morning!!");
        else if (hour < 16)
            pw.println("Good afternoon");
        else if(hour<20)
            pw.println("Good evening");
        else
            pw.println("Good night");

        pw.println("<br><br><a href= '../WishSrvApp/index.html'>Home</a>");
        //close stream object
        pw.close();
    }
}
```

web.xml

```
<web-app>
    <servlet>
```

```
        <servlet-name>abc</servlet-name>
        <servlet-class>WishApp</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/test</url-pattern>
    </servlet-mapping>
</web-app>
```

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

```
<html>

   <head>

      <title>TODO supply a title</title>

      <meta charset="UTF-8">

      <meta name="viewport" content="width=device-width, initial-scale=1.0">

   </head>

   <body>

      <!-- send the form data to url mapping “prg3“ and the get method is used  -->

<form method ="post" action="prg3">

<!--Display 3 Colors RED, BLUE, GREEN in the dropdown Box -->

<h1> Select your colors</h1>

<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>

<h1> Select your Course</h1>

UG:<input type="radio" name="course" value="ug"/><br>

PG:<input type="radio" name="course" value="pg"/><br>

       <input type="submit" value="Submit"/>

     </form>

   </body>

</html>
```

Prg3.java

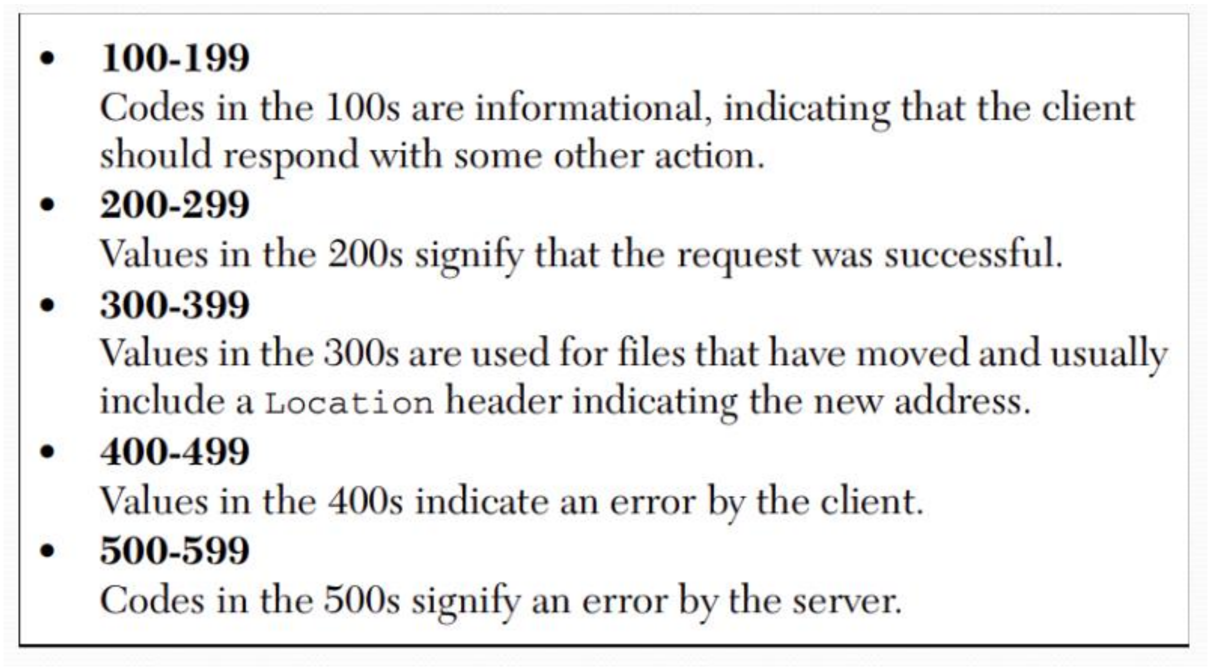
```
import java.io.IOException;

import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class prg3 extends HttpServlet {
@Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

                  // Setting the HTTP Content-Type response header to text/html
                  response.setContentType("text/html");
                  // Returns a PrintWriter object out that can send character text to the client.
                  PrintWriter out=response.getWriter();
                  // To retrieve the optional values (color) from HTML page and store in the string color
            String[] col = request.getParameterValues("color");
                  String cor = request.getParameter("course");
                  out.println("<html><body>");
            out.println("Selected Colours");
            for(String c:col)
               out.println(c);
                  out.println("<br/>You have selected course "+cor);
                  out.println("</body></html>");
                  out.close();

   }
}
```

4.a. Narrate the major range of http status codes along with their purpose.



- **100-199**
  Codes in the 100s are informational, indicating that the client should respond with some other action.
- **200-299**
  Values in the 200s signify that the request was successful.
- **300-399**
  Values in the 300s are used for files that have moved and usually include a `Location` header indicating the new address.
- **400-499**
  Values in the 400s indicate an error by the client.
- **500-599**
  Codes in the 500s signify an error by the server.

The status line consists of the
HTTP version (HTTP/1.1 in the example above),
a status code (an integer; 200 in the above example),
a very short message corresponding to the status code (OK in the example).

200 (OK)
– Everything is fine; document follows.
– Default for servlets.
204 (No Content)
– Browser should keep displaying previous document.
301 (Moved Permanently)
– Requested document permanently moved elsewhere (indicated in Location header).
– Browsers go to new location automatically.
– Browsers are technically supposed to follow 301 and 302 (next page) requests only when the incoming request is GET, but do it for POST with 303. Either way, the Location URL is retrieved with GET.
– Servlets should use sendRedirect, not setStatus, when setting this header. See example.
• **401 (Unauthorized)**
– Browser tried to access password-protected page without proper Authorization header.
• **404 (Not Found)**
– No such page. Servlets should use sendError to set this.
– Problem: Internet Explorer and small (< 512 bytes) error pages. IE ignores small error page by default.

4.b. List and explain the different HTTP 1.1 request headers and its methods
When a browser requests for a web page, it sends lot of information to the web server which can not be read directly because this information travel as a part of header of HTTP request. You can check HTTP Protocol for more information on this.

Header	**Description**
Accept	This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or**image/jpeg** are the two most common possibilities.
Accept-Charset	This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.
Accept-Encoding	This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or**compress** are the two most common possibilities.
Accept-Language	This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc.
Authorization	This header is used by clients to identify themselves when accessing password-protected Web pages.
Connection	This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used
Content-Length	This header is applicable only to POST requests and gives the size of the POST data in bytes.
Cookie	This header returns cookies to servers that previously sent them to the

	browser.
Host	This header specifies the host and port as given in the original URL.
If-Modified-Since	This header indicates that the client wants the page only if it has been
	changed after the specified date. The server sends a code, 304 which
	means Not Modifiedheader if no newer result is available.
If-Unmodified-	This header is the reverse of If-Modified-Since; it specifies that the
Since	operation should succeed only if the document is older than the
	specified date.
Referer	This header indicates the URL of the referring Web page. For example,
	if you are at Web page 1 and click on a link to Web page 2, the URL of
	Web page 1 is included in the Referer header when the browser
	requests Web page 2.
User-Agent	This header identifies the browser or other client making the request
	and can be used to return different content to different types of
	browsers.

Methods to read HTTP Header:

There are following methods which can be used to read HTTP header in your servlet

program. These methods are available with *HttpServletRequest* object.

· **getCookies**

The getCookies method returns the contents of the Cookie header, parsed and stored in an array of Cookie objects.

• **getAuthType and getRemoteUser**

The getAuthType and getRemoteUser methods break the Authorization header into its component pieces.

• **getContentLength**

The getContentLength method returns the value of the Content-Length header (as an int). **getContentType**

The getContentType method returns the value of the Content-Type header (as a String). • **getDateHeader and getIntHeader**

The getDateHeader and getIntHeader methods read the specified header and then convert them to Date and int values, respectively.

• **getHeaderNames**

Rather than looking up one particular header, you can use the getHeaderNames method to get an Enumeration of all header names received on this particular request. • **getHeaders**

In most cases, each header name appears only once in the request. Occasionally, however, a header can appear multiple times, with each occurrence listing a separate value. • **getMethod**

The getMethod method returns the main request method (normally GET or POST, but things like HEAD, PUT, and DELETE are possible).

• **getRequestURI**

The getRequestURI method returns the part of the URL that comes after the host and port but before the form data. For example, for a URL of

http://randomhost.com/servlet/search.BookSearch,

getRequestURI would return /servlet/search.BookSearch.

• **getProtocol**

Lastly, the getProtocol method returns the third part of the request line, which is generally HTTP/1.0 or HTTP/1.1.

5.a. Explain different types of session tracking techniques with example.

Hidden Form:

<INPUT TYPE="HIDDEN" NAME="session" VALUE="a1234">

This entry means that, when the form is submitted, the specified name and valare automatically included in the GET or POST data. This hidden field can be used tostore information about the session but has the major disadvantage that it only works if every page is dynamically generated by a form submission. Clicking on a regular hypertext link does not result in a form submission, so hidden form fields cannot support general session tracking, only tracking within a specific series ooperations such as checking out at a store.

Cookies

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:

<u>1.Creating a cookie object</u>

- 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());
}}}
```

Session Tracking:

1. Accessing the session object associated with the currentrequest.

Call request.getSession to get an HttpSessionobject, which is a simple hash table for storing user-specific data.

2. Looking up information associated with a session.

Call getAttribute on the HttpSession object, cast the return value to the appropriate type, and check whether the result is null.

3.Storing information in a session.

Use setAttribute with a key and a value.

4.Discarding session data.

Call removeAttribute to discard a specific value. Call invalidate to discard an entire session. Call logout to log the client out of the Web server and invalidate all sessions associated with that user.

6.a. Write the differences between JSP and servlets.

JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
It’s easier to code in JSP than in Java Servlets.	Its little much code to write here.
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.

6.b. What are the need, benefit and advantages of JSP

Need of JSP:

- JSP provides an easier way to code dynamic web pages.
- JSP does not require additional files like, java class files, web.xml etc

- Any change in the JSP code is handled by Web Container (Application server like tomcat), and doesn't require re-compilation.
- JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

Advantages of JSP:

Extension to Servlet
JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
Easy to maintain
JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
Fast Development: No need to recompile and redeploy
If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
Less code than Servlet
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects
Benefits of JSP:
JSP pages are translated into servlets. So, fundamentally, any task JSP pages can perform could also be accomplished by servlets. However, this underlying equivalence does not mean that servlets and JSP pages are equally appropriate in all scenarios. The issue is not the power of the technology, it is the convenience, productivity, and maintainability of one or the other. After all, anything you can do on a particular computer platform in the Java programming language you could also do in assembly language. But it still matters which you choose. JSP provides the following benefits over servlets alone:
• It is easier to write and maintain the HTML. Your static code is ordinary HTML: no extra backslashes, no double quotes, and no lurking Java syntax.
• You can use standard Web-site development tools. For example, we use Macromedia Dreamweaver for most of the JSP pages in the book. Even HTML tools that know nothing about JSP can be used because they simply ignore the JSP tags.
• You can divide up your development team. The Java programmers can work on the dynamic code. The Web developers can concentrate on the presentation layer. On

large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

7.a. Write a servlet program using cookies to remember user preferences.

Servlet1.java

```
package j2ee.prg4;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * Servlet implementation class store
 */
@WebServlet("/store")
public class store extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public store() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

	// Setting the HTTP Content-Type response header to text/html
	response.setContentType("text/html;charset=UTF-8");
	// Returns a PrintWriter object that can send character text to the client.
	PrintWriter out=response.getWriter();
	try
	{
		//Requesting input color from html page and storing in String variable s1
		String s1=request.getParameter("color");
		//Checking the color either RED or Green or Blue
		if (s1.equals("RED")||s1.equals("BLUE")||s1.equals("GREEN"))
		{
```

```
                        // Creating cookie object ck1 and storing the selected color
                        Cookie ck1=new Cookie("color",s1);
                        //adding the cookie to the response
                        response.addCookie(ck1);
                        //writing the output in the html format
                        out.println("<html>");
                        out.println("<body>");
                        out.println("You selected: "+s1);
                        out.println("<form action='retrieve' method='post'>");
                        out.println("<input type='Submit' value='submit'/>");
                        out.println("</form>");
                        out.println("</body>");
                        out.println("</html>");
                            }
            }
            finally
            {
                    //Closing the output object
                out.close();
            }
  }
}
```

retrieve.java

```
package j2ee.prg4;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class retrieve
 */
@WebServlet("/retrieve")
public class retrieve extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public retrieve() {
        super();
        // TODO Auto-generated constructor stub
    }
          /**
           * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
           */
          protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

                    // Setting the HTTP Content-Type response header to text/html
                    response.setContentType("text/html;charset=UTF-8");
                    // Returns a PrintWriter object that can send character text to the client.
                    PrintWriter out=response.getWriter();
                    try
                    {
                                //Requesting all the cookies and stored in cookie array ck[]
                                      Cookie ck[]=request.getCookies();
                                      out.println("<html>");
                                      out.println("<head>");
                                      out.println("<title>servlet</title>");
                                      out.println("</head>");
                                      // Getting the value from cookie and setting the HTML form
background color
                                      out.println("<body bgcolor="+ck[0].getValue()+">");
                                      //Getting the value from cookie and displaying the color name in
HTML form
                                      out.println("You selected color is: "+ck[0].getValue()+"</h1>");
                                      out.println("</body>");
                                      out.println("</html>");
                    }
                    finally
                    {
                              //closing the printwriter object out
                              out.close();
                    }
          }

}
```

Index.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to the url store and the post method is used  -->
<form action="store" method="post">
<!-- Display the Radio button with three option  -->
```

```
RED:<input type="radio" name="color" value="RED"/><br>
GREEN:<input type="radio" name="color" value="GREEN"/><br>
BLUE:<input type="radio" name="color" value="BLUE"/><br>
<input type="submit" value="submit"/>
</form>
</body>
</html>
```

8.a. What is cookie and Explain the advantages of cookies.

A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Advantages:

- Identifying a user during an e-commerce session
- Servlets have a higher-level API for this task.
- Avoiding username and password
- Customizing a site
- Focusing advertising

8.b. Explain the working of cookie in java with code snippets

Creating Cookie:

1.Create a Cookie object.

Call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie c = new Cookie("userID", "a1234");
```

2. Set the maximum age.

To tell browser to store cookie on disk instead of just In memory, use setMaxAge (argument is in seconds)

```
c.setMaxAge(60*60*24*7); // One week
```

3. Place the Cookie into the HTTP response

Use response.addCookie.

```
response.addCookie(c);
```

Reading Cookies from the Servlet:

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

Use the value (getValue) in application-specific way.

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null)
{
for(Cookie cookie: cookies)
{
if (cookieName.equals(cookie.getName()))
{           doSomethingWith(cookie.getValue());      }
}
}
```

9. Write a Servlet application to count the total number of visits on your website.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class CounterServlet extends HttpServlet
{
    //Instance variable used for counting hits on this servlet
    private int iHitCounter;
    //init method just initializes the hitCounter to zero
    public void init() throws ServletException
    {
        iHitCounter = 0;
    }
    public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
    {
        PrintWriter out =  response.getWriter();
        out.println(&quot;&lt;form&gt;&lt;fieldset style=&#39;width:15%&#39;&gt;&quot;);
        out.println(&quot;&lt;h3&gt;Welcome to my website !&lt;/h3&gt;&lt;hr&gt;&quot;);
        out.println(&quot;You are visitor number: &quot;+ (++iHitCounter));
        out.println(&quot;&lt;/fieldset&gt;&lt;/form&gt;&quot;);
    }
    public void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
    {
        doGet(request, response);
    }
}
```

10. Explain the different types of JSP tags with an example

1) Expression Tag: (<%= … %>)

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Syntax two forms:

<%= expr %>

<jsp:expression> expr </jsp:expression> (XML form)

2) Scriptlet Tag (<% … %>)

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Embeds Java code in the JSP document that will be executed each time the JSP page is processed.

Code is inserted in the service() method of the generated Servlet

Syntax two forms:

<% any java code %>

<jsp:scriptlet> ... </jsp:scriptlet>. (XML form)

Example

– <% if (Math.random() < 0.5) { %>

Have a <B>nice</B> day! <% } else { %>
Have a <B>lousy</B> day! <% } %>

• Representative result

– if (Math.random() < 0.5) { out.println("Have a <B>nice</B> day!"); } else {
out.println("Have a <B>lousy</B> day!");
}

3) Declaration Tag (<%! … %>)

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Code is inserted in the body of the servlet class, outside the service method.

o May declare instance variables.

o May declare (private) member functions.

Syntax two forms:

<%! declaration %>

<jsp:declaration> declaration(s)</jsp:declaration>

Example for declaration of Instance Variable:

```
<html>
<body>
<%! private int accessCount = 0; %>
<p> Accesses to page since server reboot:
<%= ++accessCount %> </p>
</body></html>
```

4) Directive Tag (<%@ … %>)

Directives are used to convey special processing information about the page to the JSP container.

The Directive tag commands the JSP virtual engine to perform a specific task, such as importing a Java package required by objects and methods.

Directive Description

<%@ page ... %> Defines page-dependent attributes, such as
scripting language, error page , and buffering
requirements.

<%@ include ... %> Includes a file during the translation phase.

<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

The page directive is used to provide instructions to the container that pertain to the current

JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of page directive:

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

You can write XML equivalent of the above syntax as follows:

```
<jsp:directive.page attribute="value" />
```