CMR INSTITUTE OF TECHNOLOGY

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

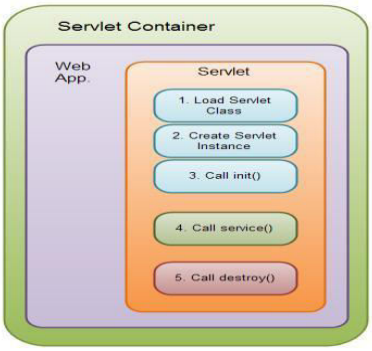
Internal Assessment Test 1 – March 2017

Sub:	Advanced Java Programming							Code:	13MCA 42
Date:	27.03.2017	Duration:	90 mins	Max Marks:	50	Sem:	4	Branch:	MCA

Answer any five of the following. 5x10=50M

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.a. Briefly explain the different HTTP 1.1 request header
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.

b. 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.

3.a With an example, explain the import attribute and the session attribute

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.

Syn:

<%@ page import="name of class" %>

Ex:

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

Session Attribute:

The session attribute controls whether the page participates in HTTP sessions. Use of this attribute takes one of the following two forms. A value of true (the default) signifies that the predefined variable session (of type HttpSession) should be bound to the existing session if one exists; otherwise, a new session should be created and bound to session. A value of false means that no sessions will be automatically created and that attempts to access the variable session will result in errors at the time the JSP page is translated into a servlet. Using session="false" may save significant amounts of server memory on high-traffic sites. However, note that using session="false" does not disable session tracking—it merely prevents the JSP page from creating new sessions for users who don't have them already. So, since sessions are user specific, not page specific, it doesn't do any good to turn off session tracking for one page unless you also turn it off for related pages that are likely to be visited in the same client session.

Ex:

```
<% @ page session=”false”%>
```

b. Write a jsp program to implement verification of a particular user login and display a welcome page

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body bgcolor="pink">
<form action="prg5.jsp”>
<!-- username : <input type="text"  -->
User name :  <input type="text" name="uname"><br>
<!-- password : <input type="password"  -->
password: <input type="password" name="pass"><br>
<input type="submit" value="submit"></form></body>
</html>
```

prg5.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body bgcolor="pink">
<%String user="MCA";
String pass="CMRIT";
//Getting the input uname from the html form and storing in String ‘uname’
String uname=request.getParameter("uname");
//Getting the input pass from the html form and storing in String ‘password’
String password=request.getParameter("pass");
// checking for the valid username and password
```

```
if(user.equals(uname) && pass.equals(password))
{
        //Printing the welcome message with username and password
        out.print("<b>Welcome</b>"+uname+"<br>");
        out.print("Username is:"+uname+"<br>");
        out.print("Password is:"+pass);
}
else
{
        //Printing the message  "Invalid User name and Password"  in the web Browser
        out.println("invalid UserName AND PASSWORD");
}
%>
</body>
</html>
```

4. 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:

1.Creating a cookie object

- 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.

5. 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>
```

6.a.Explain the following action tags with a code snippet.
i)<jsp:forward>
ii)<jsp:parameter>

i)<jsp:forward>

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syn:

<jsp:forward page=”name of resource” />

Ex:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to login.jsp and the get method is used  -->
<form method="get" action="login.jsp">
UserName :  <input type="text" name ="name"><br>
Password :  <input type="password" name ="pass"><br>
<input type="Submit" value ="Submit"/><br>
</form>
</body>
</html>
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
//Getting the input name from the html form and storing in String ‘uname’
String uname = request.getParameter("name");
//Getting the input pass from the html form and storing in String ‘upass’
String upass = request.getParameter("pass");
if(uname.equals("admin") && upass.equals("admin"))
{
 %>
            <jsp:forward page="main.jsp"></jsp:forward>

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

main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
```

```
</head>
<body>
<%
// Getting the input name from the html form and storing in String ‘un’-->
String un=request.getParameter("name");
// Getting the input pass from the html form and storing in String ‘pw’-->
String pw=request.getParameter("pass");
%>
<h1>welcome:<%=un%></h1>
<h1>your user name is:<%=un%></h1>
<h1>your password is:<%=pw%></h1>
</body>
</html>
```

b. Write the differences between <jsp:include> action tag and include directive.

Table 13.1 Differences Between `jsp:include` and the `include` Directive

	`jsp:include` Action	**`include` Directive**
What does basic syntax look like?	`<jsp:include page="..." />`	`<%@ include file="..." %>`
When does inclusion occur?	Request time	Page translation time
What is included?	Output of page	Actual content of file
How many servlets result?	Two (main page and included page each become a separate servlet)	One (included file is inserted into main page, then that page is translated into a servlet)

Table 13.1 Differences Between `jsp:include` and the `include` Directive *(continued)*

	`jsp:include` Action	**`include` Directive**
Can included page set response headers that affect the main page?	No	Yes
Can included page define fields or methods that main page uses?	No	Yes
Does main page need to be updated when included page changes?	No	Yes
What is the equivalent servlet code?	`include` method of `RequestDispatcher`	None

7.What is the use of get() and post() methods.

Write a servlet program to implement and demonstrate get () and post() methods.

```
package prg3.j2ee;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class prg3
 */
@WebServlet("/prg3")
public class prg3 extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public prg3() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(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");
		// 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.getParameter("color");
		out.println("<html><body bgcolor="+col+">");
		out.println("You have selected "+col);
		out.println("</body></html>");
		out.close();
	}
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</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 -->
<select name="color" size="1">
<Option value="red">RED</Option>
<Option value="green">GREEN</Option>
<Option value="blue">BLUE</Option>
</select>
<input type="Submit" value="Enter">
</form>
</body>
</html>
```