USN :

CMR Institute of Technology, Bangalore
I - INTERNAL ASSESSMENT

Semester: 4-CBCS 2018
Subject: ADVANCED JAVA PROGRAMMING (18MCA41)
Faculty: Dr Helen Josephine

Date: 6 Apr 2021
Time: 09:00 AM - 10:30 AM
Max Marks: 50

PART A						
Answer any1 question(s)						
Q.No			Marks	CO	PO	BT/CL
1		Define Servlet. Explain life cycle of Servlet with neat diagram and appropriate method implementations	10	CO1	PO1	L1
2		Write a Java servlet program using cookies to remember user preferences.	10	CO1	PO1	L3
PART B						
Answer any1 question(s)						
Q.No			Marks	CO	PO	BT/CL
3		Explain the list of the request headers and methods to read request headers	10	CO1	PO1	L1
4	a	What are the need, benefit and advantages of JSP	5	CO2	PO1	L2
	b	Write a short note on getAttribute() and setAttribute()	5	CO1	PO1	L1
PART C						
Answer any1 question(s)						
Q.No			Marks	CO	PO	BT/CL
5	a	Write the differences between JSP and servlets.	5	CO2	PO1	L3
	b	Compare get and post method in java servlet	5	CO1	PO1	L3
6		What is session tracking techniques? List and explian the different session tracking techniques in Java	10	CO1	PO1	L2
PART D						
Answer any1 question(s)						
Q.No			Marks	CO	PO	BT/CL
7		Explain the different types of JSP tags with an example	10	CO1	PO1	L1
8	a	Narrate the major range of http status codes along with their purpose. Give atleast 2 status code and their meaning for each range.	10	CO1	PO1	L1
PART E						
Answer any1 question(s)						
Q.No			Marks	CO	PO	BT/CL
9		Write a Java JSP program which uses jsp:include and jsp:forward action to display a web page.	10	CO2	PO1	L3
10		Explain any five attributes of page directive with an example.	10	CO2	PO1	L1

CMR
INSTITUTE OF
TECHNOLOGY

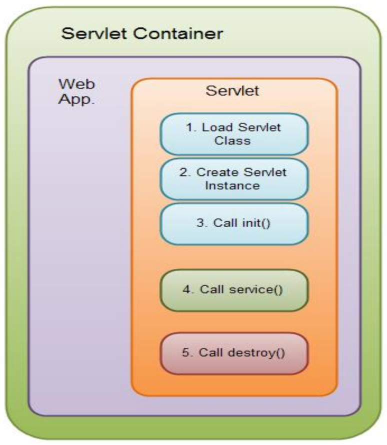
Internal Assessment Test 1 – Answer key

Sub:	Advanced Java Programming							**Sub Code:**	**18MCA41**
Date:	6/04/2021	Duration:	90 min's	Max Marks:	50	**Sem**	**4**	**Branch:**	**MCA**

1. Define Servlet. Explain life cycle of Servlet with neat diagram and appropriate method implementations

- 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.

Servlet Life Cycle



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 Java 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>
```

3. Explain the list of the request headers and methods to read request headers

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.

Following is the important header information which comes from browser side and you would use very frequently in web programming:

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 Modified**header if no newer result is available.
If-Unmodified-Since	This header is the reverse of If-Modified-Since; it specifies that the 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.

4a. 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.

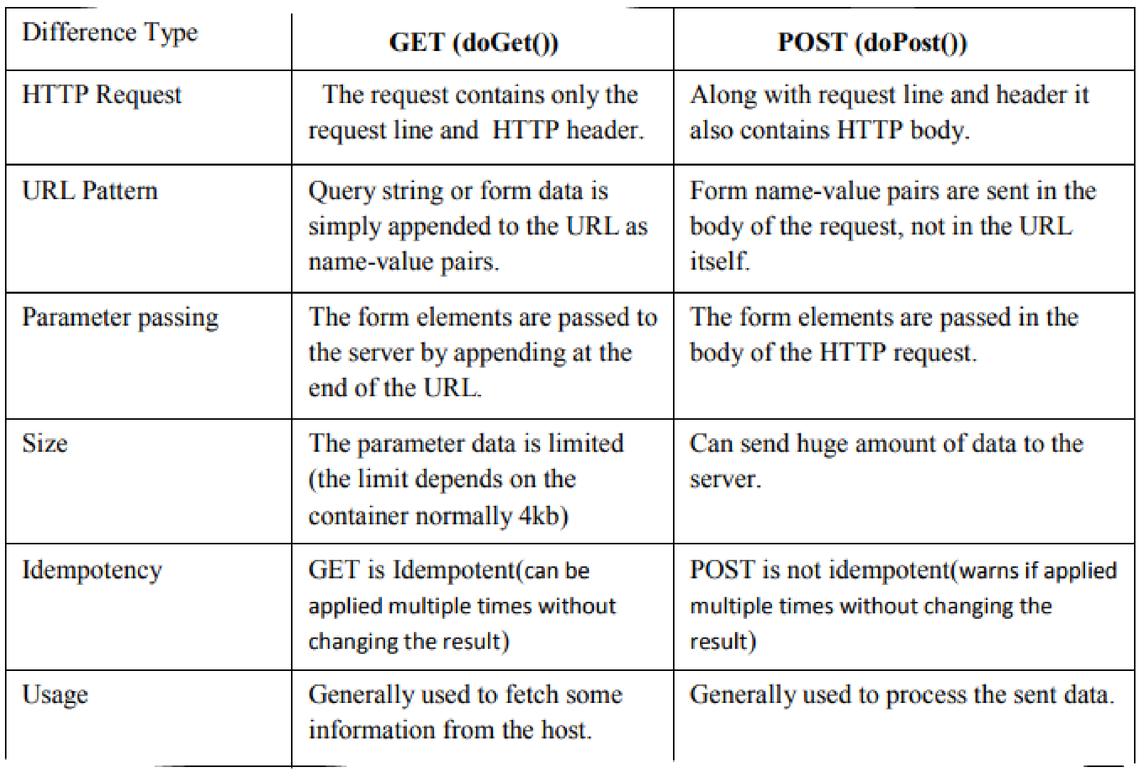
- *Less code than Servlet*
    - In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc

4b. Write a short note on getAttribute() and setAttribute()

5a. Write the differences between JSP and servlets.

C	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.

5b. Compare get and post method in java servlet



Difference Type	GET (doGet())	POST (doPost())
HTTP Request	The request contains only the request line and HTTP header.	Along with request line and header it also contains HTTP body.
URL Pattern	Query string or form data is simply appended to the URL as name-value pairs.	Form name-value pairs are sent in the body of the request, not in the URL itself.
Parameter passing	The form elements are passed to the server by appending at the end of the URL.	The form elements are passed in the body of the HTTP request.
Size	The parameter data is limited (the limit depends on the container normally 4kb)	Can send huge amount of data to the server.
Idempotency	GET is Idempotent(can be applied multiple times without changing the result)	POST is not idempotent(warns if applied multiple times without changing the result)
Usage	Generally used to fetch some information from the host.	Generally used to process the sent data.

6. What is session tracking techniques? List and explian the different session tracking techniques in Java

Session Tracking Techniques

There are four techniques used in Session tracking:

- Cookies
- HttpSession
- Hidden Form Field
- URL Rewriting

Cookies – Refer previous answer

Example prg for cookies / Lab Exercise No. 4:

Index.html:
```
<form action="FirstServlet" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

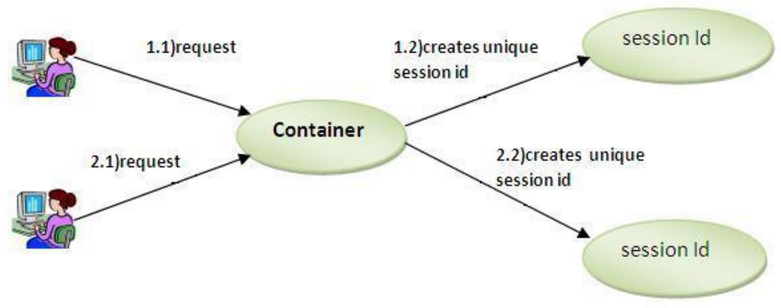
FirstServlet.java:
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response){
try{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String n=request.getParameter("userName");
out.print("Welcome "+n);
Cookie ck=new Cookie("uname",n);//creating cookie object
response.addCookie(ck);//adding cookie in the response
//creating submit button
out.print("<form action='SecondServlet' method='post'>");
out.print("<input type='submit' value='go'>");
out.print("</form>");
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 doPost(HttpServletRequest request, HttpServletResponse response){
try{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
Cookie ck[]=request.getCookies();
out.print("Hello "+ck[0].getValue());
out.close();
}catch(Exception e){System.out.println(e);}}
```

2.HttpSession Interface

In such case, container creates a session id for each user.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);}
  }


}
```

Hidden Form fields

- Hidden Form fields are added to an HTML form that are not displayed in the client's browser.
- They are sent back to the server when the form that contains them is submitted.

Advantages :

- The advantages of hidden form fields are their ubiquity and support for anonymith
- Hidden fields are supported in all the popular browsers,
- They demand no special server requirements
- They can be used with clients that have not registered or loggin

Disadvantage

- It works only for the sequence of dynamically generated forms
- This technique breaks down immediately with static documents. E-mailed documents, bookmarked documents, and browser shutdowns

Example prg for hidden form field

Index.html:

```
<form action="FirstServlet" method="post">
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);

//creating submit button
out.print("<form action='SecondServlet’>");
out.print("<input type='hidden’ name=’userName’ value=’”+n+”’>”);
out.print("<input type='submit' value='go'>");
out.print("</form>");
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 doPost(HttpServletRequest request, HttpServletResponse response){
try{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
 String n=request.getParameter("userName");
out.print("Welcome "+n);
out.close();
}catch(Exception e){System.out.println(e);}}
```

URL Rewriting

- With URL rewriting, every local URL the user might client on is dynamically modified or rewritten, to include extra information.
- The extra info. can be in the form of extra path information, added parameters or some custom, server-specific URL change.

URL?name1=value1&name2=value2

Advantages

- IT will always work whether cookie is disable or not, so it is broser independent
- Extra form submission is of required on each page

Disadvantage:

- It will work only with links
- It can send only textual information

Example prg for URL rewritting

Index.html:
```
<form action="FirstServlet" method="post">
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);
out.print(<a href=”servlet2?uname=”+n+”>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 doPost(HttpServletRequest request, HttpServletResponse response){
```

```
try{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
 String n=request.getParameter("userName");
out.print("Welcome "+n);
out.close();
}catch(Exception e){System.out.println(e);}}
```

7. 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" />
```

Attributes:

Attribute	Purpose
buffer	Specifies a buffering model for the output stream.
autoFlush	Controls the behavior of the servlet output buffer.
contentType	Defines the character encoding scheme.

errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.

8a. Narrate the major range of http status codes along with their purpose. Give atleast 2 status code and their meaning for each range.

When a Web server responds to a request from a browser or other Web client, the response typically consists of a status line, some response headers, a blank line, and the document. Here is a minimal example:

HTTP/1.1 200 OK
Content-Type: text/plain
Hello World

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.
302 (Found)
– Requested document temporarily moved elsewhere (indicated in Location header).
– Browsers go to new location automatically.
– 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.
– Fun and games: http://www.plinko.net/404/

Setting the status Code
response.setStatus(int statusCode)
– Use a constant for the code, not an explicit int. Constants are in HttpServletResponse
– Names derived from standard message. E.g., SC_OK, SC_NOT_FOUND, etc.

response.sendError(int code, String message)

– Wraps message inside small HTML document

response.sendRedirect(String url)
– Sets status code to 302
– Sets Location response header also

9. Write a Java JSP program which uses jsp:include and jsp:forward action to display a web page.

index.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>
<!-- 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>

<%
}
else
{
            out.println("Wrong Credentials Username and Password"+"<br>");
            out.println("Enter Corrects Username and Password.. Try again"+"<br><br>");%>
    <jsp:include page="index.jsp"></jsp:include>
<%
}
%>
</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>
```

10. Explain any five attributes of page directive with an example.

i) import

The import attribute of the page directive lets us specify the packages that should be imported by the servlet into which the JSP page gets translated.

By default, the servlet imports java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*, and possibly some number of server-specific entries. Never write JSP code that relies on any server-specific classes being imported automatically; doing so makes your code nonportable. Use of the import attribute takes one of the following form

<% @ page import =”package.classname” %>

Ex: <% @ page import=”java.util.*” %>

ii) errorPage and isErrorPage

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.

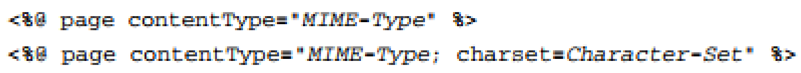
```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>
<%= 100/0 %>
</body>
</html>
```

The isErrorPage attribute is used to declare that the current page is the error page.

```
<html>
<body>
<%@ page isErrorPage="true" %>
             Sorry an exception occured!<br/>
            The exception is: <%= exception %>
 </body>
</html>
```

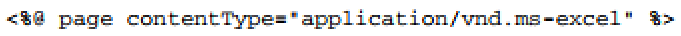
iii) Content Type

The contentType attribute sets the Content-Type response header, indicating the MIME type of the document being sent to the client.



```
<%@ page contentType="MIME-Type" %>
<%@ page contentType="MIME-Type; charset=Character-Set" %>
```

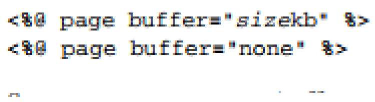
Ex:



```
<%@ page contentType="application/vnd.ms-excel" %>
```

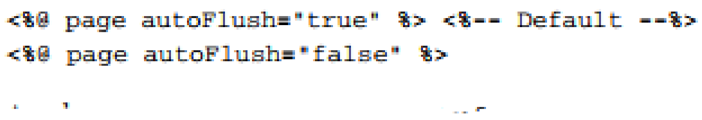
iv) buffer and autoflush

The buffer attribute specifies the size of the buffer used by the out variable, which is of type JspWriter. Use of this attribute takes one of two forms:



```
<%@ page buffer="sizekb" %>
<%@ page buffer="none" %>
```

The autoFlush attribute controls whether the output buffer should be automatically flushed when it is full (the default) or whether an exception should be raised when the buffer overflows (autoFlush="false"). Use of this attribute takes one of the following two forms.



```
<%@ page autoFlush="true" %> <%-- Default --%>
<%@ page autoFlush="false" %>
```