CMR
INSTITUTE OF
TECHNOLOGY

USN

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 4 – Feb. 2022

Sub:	**Advances in Java**							**Sub Code:**	**20MCA33**
Date:	**2/2/2022**	**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	
			CO	RBT
	PART I			
1	Define Servlet. Explain the basic servlet structure and its life cycle methods.	10	CO1	L2
	OR			
2	Explain the various steps of JDBC with code snippet.	10	CO4	L2
	PART II			
3	Explain the different type of JDBC drivers	10	CO4	L2
	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	Create a JSP application which uses jsp:include and jsp:forward action to display a Webpage.	05	CO5	L6
	PART IV			
7	Discuss the types of JDBC statements with an example	10	CO4	L2
	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 JAVA Program to insert data into Student DATA BASE and retrieve info based on particular queries	10	CO4	L6
	OR			
10	Explain the different attributes of page directive with an example.	10	CO2	L2

CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 4– Feb 2022

Sub:	Advances in Java					Sub Code:	20MCA33	Branch:	MCA
Date:	2/2/2022	Duration:	90 min’s	Max Marks:	50	Sem	III		

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

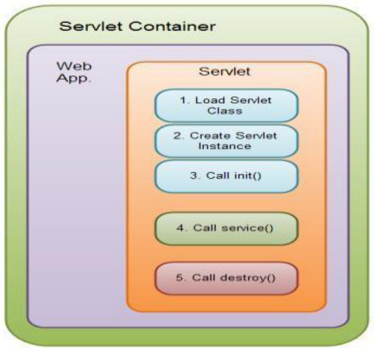
Java Servlets are programs that run on a Web or Application server

 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 :

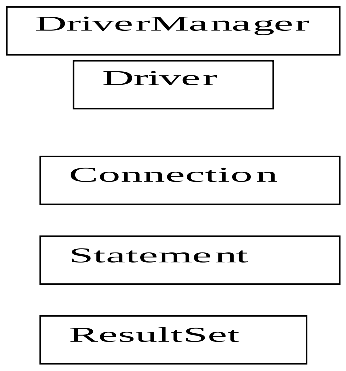
 The destroy() method is called only once at the end of the life cycle of a servlet.
 This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
 After the destroy() method is called, the servlet object is marked for garbage collection.

The destroy method definition looks like this:

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

2. Explain the various steps of JDBC with code snippet.

Seven Basic Steps in Using JDBC



1. Load the Driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement Object
5. Execute a query
6. Process the results
7. Close the Connection

1. <u>Load the JDBC driver</u>

When a driver class is first loaded, it registers itself with the driver Manager Therefore, to register a driver, just load it!
Example:

String driver = “sun.jdbc.odbc.JdbcOdbcDriver”; Class.forName(driver);
Or Class.forName(sun.jdbc.odbc.JdbcOdbcDriver);

2. <u>Define the Connection URL</u>

The purpose of loading and registering the JDBC driver is to bring the JDBC driver into the Java Virtual Machine (JVM).

jdbc : subprotocol : source

- each driver has its own subprotocol
each subprotocol has its own syntax for the source

jdbc:msql://host[:port]/database

Ex: jdbc:msql://foo.nowhere.com:4333/accounting

3. Establish the Connection

- DriverManager Connects to given JDBC URL with given **user name** and **password**
- **Throws** java.sql.SQLException
- **returns** a Connection object
- A Connection represents a session with a specific database.
- The connection to the database is established by **getConnection()**, which requests access to the database from the DBMS.
- A Connection object is returned by the getConnection() if access is granted; else getConnection() throws a SQLException.
- If username & password is required then those information need to be supplied to access the database.

  String url = jdbc : odbc : Employee;

  Connection c = DriverManager.getConnection(url,userID,password);
- Sometimes a DBMS requires extra information besides userID & password to grant access to the database.
- This additional information is referred as properties and must be associated with Properties or Sometimes DBMS grants access to a database to anyone without using username or password.

  Ex: Connection c = DriverManager.getConnection(url) ;

4. Create a Statement Object

A Statement object is used for executing a static SQL statement and obtaining the results produced by it.

Statement stmt = con.createStatement();

This statement creates a Statement object, *stmt* that can pass SQL statements to the DBMS using connection, *con*.

5. Execute a query

Execute a SQL query such as SELECT, INSERT, DELETE, UPDATE Example

```
String SelectStudent= "select * from STUDENT";
```

6. **Process the results**

- A ResultSet provides access to a table of data generated by executing a Statement.
- Only one ResultSet per Statement can be open at once.
- The table rows are retrieved in sequence.
- A ResultSet maintains a cursor pointing to its current row of data.
- The 'next' method moves the cursor to the next row.

7. **Close the Connection**

connection.close();

- Since opening a connection is expensive, postpone this step if additional database operations are expected

3. Explain the different type of JDBC drivers

i. JDBC driver specification classifies JDBC drivers into four groups. They are…

Type 1: JDBC-to-ODBC Driver

ii. Microsoft created ODBC (Open Database Connection), which is the basis from which Sun created JDBC. Both have similar driver specifications and an API.
iii. The JDBC-to-ODBC driver, also called the JDBC/ODBC Bridge, is used to translate DBMS calls between the JDBC specification and the ODBC specification.
iv. MS Access and SQL Server contains ODBC driver written in C language using pointers, but java does not support the mechanism to handle pointers.
v. So JDBC-ODBC Driver is created as a bridge between the two so that JDBC-ODBC bridge driver translates the JDBC API to the ODBC API.

 Type-1 ODBC Driver for MS Access and SQL Server Drawbacks of Type-I

Driver:

1. ODBC binary code must be loaded on each client.
2. Transaction overhead between JDBC and ODBC.
3. It doesn"t support all features of Java.
4. It works only under Microsoft, SUN operating systems.

Type 2: Java/Native Code Driver or Native-API Partly Java Driver

vi. It converts JDBC calls into calls on client API for DBMS.

vii. The driver directly communicates with database servers and therefore some database client software must be loaded on each client machine and limiting its usefulness for internet

viii. The Java/Native Code driver uses Java classes to generate platform- specific code that is code only understood by a specific DBMS.

Ex: Driver for DB2, Informix, Intersoly, Oracle Driver, WebLogic drivers **Drawbacks**

of Type-I Driver:

1. Some database client software must be loaded on each client machine
2. Loss of some portability of code.
3. Limited functionality
4. The API classes for the Java/Native Code driver probably won"t work with another manufacturer"s DBMS.

<u>Type 3: Net-Protocol All-Java Driver</u>

ix. It is completely implemented in java, hence it is called pure java driver. It translates the JDBC calls into vendor‟s specific protocol which is translated into DBMS protocol by amiddleware server
x. Also referred to as the Java Protocol, most commonly used JDBC driver.
xi. The Type 3 JDBC driver converts SQL queries into JDBC- formatted statements, in-turn they are translated into the format required by the DBMS.

Ex: Symantec DB

<u>Drawbacks:</u>

xii. It does not support all network protocols.
xiii. Every time the net driver is based on other network protocols.

<u>Type 4: Native-Protocol All-Java Driver or Pure Java Driver</u>

xiv. Type 4 JDBC driver is also known as the Type 4 database protocol.
xv. The driver is similar to Type 3 JDBC driver except SQL queries are translated into the format required by the DBMS.
xvi. SQL queries do not need to be converted to JDBC-formatted systems.
xvii. This is the fastest way to communicated SQL queries to the DBMS.
xviii. Here the driver uses network protocol this protocol is already built-into the database engine; here the driver talks directly to the database using java sockets. This driver is better than all other drivers, because this driver supports all network protocols.
xix. Use Java networking libraries to talk directly to database engines

Ex: Oracle, MYSQL

<u>Only disadvantage:</u> need to download a new driver for each database engine

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

When a Web server responds to a request, the response typically consists of a status line, some response headers, a blank line, and the document.

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
```

(Blank Line)
The status codes fall into five general categories:
• **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.

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.

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.

5. 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.

6. Create a JSP application which uses jsp:include and jsp:forward action to display a Webpage.

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"))
{
 %>


<%
}
```

```
        else
        {
<jsp:forward page="main.jsp"></jsp:forward>

        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>
```

7. Discuss the types of JDBC statements with an example

- The Statement object is used whenever J2EE component needs to immediately execute a query without first having the query compiled.

Statement Object contains 3 methods:

1. **Execute() (**used for **DDL** commands like, **Create, Alter, Drop)**
2. **executeUpdate() (**Used for **DML** commands like, **Insert, Update, Delete)**
3. **exceuteQuery() (**Used for **Select** command**)**

- The **execute()** method is used during execution of DDL commands and also used when there may be multiple results returned.
- The **executeUpdate()** executes INSERT, UPDATE, DELETE, and returns an int value specifying the number of rows affected or 0 if zero rows selected
- The **executeQuery()** method, which passes the query as an argument. The query is then

transmitted to the DBMS for processing.

- The executeQuery() method executes a simple select query and returns a ResultSet object.
- The ResultSet object contains rows, columns, and metadata that represent data requested by query.

Example-1:

Statement stmt = con.createStatement();
ResultSet res = stmt.**executeQuery**("select * from Employee");
OR

Example-2:

Statement stmt = con.createStatement();
stmt.**executeUpdate**("Insert into employee values(„12345‟,‟sk‟,98453));
stmt.**executeUpdate**("update employee set Mobile=89706 where Mobile=12345);
OR

Example-3:

Statement stmt = con.createStatement();
stmt.**execute**("Drop table Employee");
stmt.**execute**("Create table Employee (name varhcar(10), age Number(3))");

```
import java.sql.*;

public class StatementDemo {
   public static void main(String args[]){
     try{
       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
       Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
       Statement stmt;
       stmt= con.prepareStatement("select * from employee where Name='abc'");
       ResultSet rs=stmt.executeQuery();
       while(rs.next()){
          System.out.println(rs.getString(1));
       }
     } // end of try
     catch(Exception e){ System.out.println("exception" + e); }
   } //end of main
   } // end of class
```

The Bold line can be replace
by any of the

1.6.1 PreparedStatement Object

- The preparedStatement object allows you to execute parameterized queries.
- A SQL query can be precompiled and executed by using the PreparedStatement object.

  Ex: Select * from publishers where pub_id=?
- Here a query is created as usual, but a question mark is used as a placeholder for a value that is inserted into the query after the query is compiled.
- The preparedStatement() method of Connection object is called to return the PreparedStatement object.

  Ex:

  ```
  PreparedStatement stat;
  stat= con.prepareStatement("select * from publisher where pub_id=?")
  ```

```
import java.sql.*;

public class JdbcDemo {
    public static void main(String args[]){
      try{
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
         PreparedStatement pstmt;
         pstmt= con.prepareStatement("select * from employee whereUserName=?");
         pstmt.setString(1,"khutub");
         ResultSet rs1=pstmt.executeQuery();
         while(rs1.next()){
             System.out.println(rs1.getString(2));
         }
       } // end of try
       catch(Exception e){System.out.println("exception"); }
    } //end of main
    } // end of class
```

1.6.2 CallableStatement

- The CallableStatement object is used to call a stored procedure from within a J2EE object.
- A Stored procedure is a block of code and is identified by a unique name.
- The type and style of code depends on the DBMS vendor and can be written in PL/SQL, Transact-SQL, C, or other programming languages.
- IN, OUT and INOUT are the three parameters used by the CallableStatement object to call a stored procedure.
- The IN parameter contains any data that needs to be passed to the stored procedure and whose value is assigned using the setxxx() method.
- The OUT parameter contains the value returned by the stored procedures. The OUT parameters must be registered using the registerOutParameter() method, later retrieved by using the getxxx()
- The INOUT parameter is a single parameter that is used to pass information to the stored procedure and retrieve information from the stored procedure.

```
Connection
con; try{
String query = "{CALL
LastOrderNumber(?))}"; CallableStatement
stat = con.prepareCall(query);
stat.registerOutParameter( 1
,Types.VARCHAR); stat.execute();
String lastOrderNumber =
stat.getString(1); stat.close();
}
catch (Exception e){}
```

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

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.

• **Typical advantages of Cookies**

– 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

Create a Cookie

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

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

• **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

• **Place the Cookie into the HTTP response**

– Use response.addCookie.

– If you forget this step, no cookie is sent to the browser!

response.addCookie(c);

Accessing the cookies:

Call request.getCookies

– This yields an array of Cookie objects.

• **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());       }
}
}
```

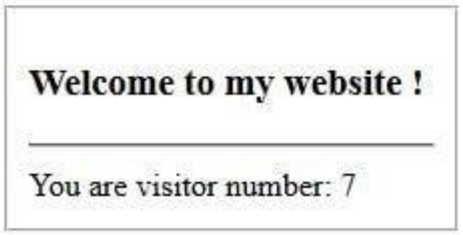
9. Write a Servlet application to count the total number of visits on your website.

```
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("<form><fieldset style='width:15%'>");
        out.println("<h3>Welcome to my website !</h3><hr>");
        out.println("You are visitor number: "+ (++iHitCounter));
        out.println("</fieldset></form>");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        doGet(request, response);
    }
}
```

Output:


Welcome to my website !

You are visitor number: 7

9. Write a JAVA Program to insert data into Student DATA BASE and retrieve info based on particular queries.

```
package j2ee.p9;
import java.sql.*;
import java.io.*;
public class Studentdata {
public static void main(String[] args) {
Connection con;
PreparedStatement pstmt;
Statement stmt;
ResultSet rs;
String uname, pword;
Integer marks,count;
try
{
Class.forName("com.mysql.jdbc.Driver"); // type1 driver
try{
con=DriverManager.getConnection("jdbc:mysql://127.0.0.1/mca","root","system"); //  type1 access
connection
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
   do
   {
System.out.println("\n1. Insert.\n2. Select.\n3. Update.\n4. Delete.\n5. Exit.\nEnter your choice:");
int choice=Integer.parseInt(br.readLine());
switch(choice)
{
case 1: System.out.print("Enter UserName :");
uname=br.readLine();
System.out.print("Enter Password :");
pword=br.readLine();
pstmt=con.prepareStatement("insert into student values(?,?)");
pstmt.setString(1,uname);
pstmt.setString(2,pword);
pstmt.execute();
System.out.println("\nRecord Inserted successfully.");
break;
case 2:
```

```
stmt=con.createStatement();
rs=stmt.executeQuery("select *from student");
if(rs.next())
{
System.out.println("User Name\tPassword\n--------------------------------");
do
{
uname=rs.getString(1);
pword=rs.getString(2);

System.out.println(uname+"\t"+pword);
}while(rs.next());
}
else
System.out.println("Record(s) are not available in database.");
break;
case 3:
System.out.println("Enter User Name to update :");
uname=br.readLine();
System.out.println("Enter new password :");
pword=br.readLine();
stmt=con.createStatement();
count=stmt.executeUpdate("update student set password='"+pword+"'where username='"+uname+"'");
System.out.println("\n"+count+" Record Updated.");
break;
case 4: System.out.println("Enter User Name to delete record:");
uname=br.readLine();
stmt=con.createStatement();
count=stmt.executeUpdate("delete from student where username='"+uname+"'");


if(count!=0)
System.out.println("\nRecord "+uname+" has deleted.");
else
System.out.println("\nInvalid USN, Try again.");
break;

case 5: con.close(); System.exit(0);
default: System.out.println("Invalid choice, Try again.");
}//close of switch
}while(true);
}//close of nested try
catch(SQLException e2)
{
System.out.println(e2);
}
catch(IOException e3)
```

```
{
System.out.println(e3);
}
}//close of outer try
catch(ClassNotFoundException e1)
{
System.out.println(e1);
}
}
}
```

10. Explain the different attributes of page directive with an example.

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

Syntax of JSP page directive

1. <%@ page attribute="value" %>

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

Example of import attribute

1. <html>

```
<body>

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

</body>
</html>
```

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

```
<html>
<body>

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

</body>
</html>
```

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)info

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

Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
```

```
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example of buffer attribute

```
<html>
<body>

<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8)isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
```

```
  implements SingleThreadModel{
.......
}
```

9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

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

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

10)isErrorPage

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

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

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

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```