Q1. What is Server Socket? Discuss the difference between the Socket and ServerSocket class.

Ans.
- The ServerSocket class (java.net) can be used to create a server socket.
- This object is used to establish communication with the clients.
- A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.
- The actual work of the server socket is performed by an instance of the SocketImpl class.
- An application can change the socket factory that creates the socket implementation to configure itself to create sockets appropriate to the local firewall.

Constructor

ServerSocket(int port)	Creates a server socket, bound to the specified port.

Method

public Socket **accept**()	Returns the socket and establish a connection between server and client.

Syntax
```
ServerSocket ss=new ServerSocket(Port_no);
```

Difference between the Socket and ServerSocket class.

ServerSocket	Socket
It is placed in server side, which sends request to client side socket (Socket) and wait for the response from client.	It is placed in client side, which sends request to server side socket (ServerSocket) and wait for the response from serve.
`ServerSocket ss=new ServerSocket (1111);`	`Socket s = new Socket("localhost",1111);`

Q2. What is Datagram Socket and Datagram Packet? Explain in detail with example.

Ans. Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Datagram Socket
- DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.
- A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Constructor

DatagramSocket()	It creates a datagram socket and binds it with the available Port Number on the localhost machine.
DatagramSocket(int port)	It creates a datagram socket and binds it with the given Port Number.
DatagramSocket(int port, InetAddress address)	It creates a datagram socket and binds it with the specified port number and host address.

Datagram Packet

- Java DatagramPacket is a message that can be sent or received.
- Additionally, packet delivery is not guaranteed.
- Datagram packets are used to implement a connectionless packet delivery service.
- Each message is routed from one machine to another based solely on information contained within that packet.
- Multiple packets sent from one machine to another might be routed differently, and might arrive in any order.

Constructor

DatagramPacket(byte[] barr, int length)	It creates a datagram packet. This constructor is used to receive the packets.
DatagramPacket(byte[] barr, int length, InetAddress address, int port)	It creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

DSender.java
```
import java.net.*; //required for Datagram Class
public class DSender{
  public static void main(String[] args)
                      throws Exception
 {
    DatagramSocket ds = new DatagramSocket();
    String str = "Message sent by Datagram socket";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
    DatagramPacket dp = new DatagramPacket
                          (str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

DReceiver.java
```
import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception
{
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0,dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

Q3. Explain InetAddress methods with appropriate example.
Ans.
- This class represents an Internet Protocol (IP) address.
- The java.net.InetAddress class provides methods to get an IP of host name.
- It is the superclass of Inet6Address and Inet4Address classes.
- There are no constructors for this class but static methods which returns instances of InetAddress class for general use.

Methods

Method	Description

public static InetAddress **getByName**(String host) throws UnknownHostException	Determines the IP address of a given host's name.
	 `InetAddress ip =InetAddress.getByName("www.darshan.ac.in");`
	 `System.out.println("ip:"+ ip);`

public static InetAddress **getLocalHost**() throws UnknownHostException	Returns the address of the local host.
	 `InetAddress ip=InetAddress.getLocalHost();`
	 `System.out.println("LocalHost:"+ip);`

public String **getHostName**()	It returns the host name of the IP address.
	 `InetAddress ip =InetAddress.getByName("10.254.3.34");`
	 `System.out.println("Hostname:"+ip.getHostName());`

public String **getHostAddress**()	It returns the IP address in string format.
	 `InetAddress ip =InetAddress.getByName("www.darshan.ac.in");`
	 `System.out.println("HostAddress:"+ip.getHostAddress());`

Q4. Explain URL and URLConnection class with example
Ans. **URL Class**
- The Java URL class represents an URL.
- This class is pointer to "resource" on the World Wide Web.

Example
`URL url=new URL("http://www.darshan.ac.in");`

Method

public URLConnection **openConnection**() throws IOException	This method of URL class returns the object of URLConnection class `URLConnection urlcon=url.openConnection();`

URLConnection class
- URLConnection is the superclass of all classes that represent a communications link between the application and a URL.
- Instances of this class can be used both to read from and to write to the resource referenced by the URL.

Method

public InputStream **getInputStream**() throws IOException	Returns an input stream that reads from this open connection.
public OutputStream **getOutputStream**() throws IOException	Returns an output stream that writes to this connection.

Example
```
import java.io.*; //required for input stream
import java.net.*; //required for URL & URLConnection
public class URLConnectionDemo {
    public static void main(String[] args){
    try{
        URL url=new URL("http://www.darshan.ac.in");
        URLConnection urlcon=url.openConnection();
        InputStream stream=urlcon.getInputStream();
        int i;
        while((i=stream.read())!=-1){
            System.out.print((char)i);
        }
    }catch(Exception e){System.out.println(e);}
  }
}
```

Q5. **How to display IP address and host name for local machine.**

Ans.
```
import java.net.InetAddress;
public class Main {
 public static void main(String[] args)throws Exception {
     InetAddress addr = InetAddress.getLocalHost();
     System.out.println("Local HostAddress:  "+addr.getHostAddress());
     String hostname = addr.getHostName();
     System.out.println("Local host name: "+hostname);
     }
}
```

Q6. **Write TCP and UDP program for two way communication.**

Ans. **TCP**
-Server-

```java
import java.io.*;
import java.net.*;
class Server1
{
        public static void main(String ar[])throws Exception
        {
                ServerSocket ss=new ServerSocket(777);
                Socket s=ss.accept();
                System.out.println("Connection Established");
                OutputStream obj=s.getOutputStream();
                PrintStream ps=new PrintStream(obj);

                String str="Hello Client";
                ps.println(str);
                ps.println("Bye");

                ps.close();
                ss.close();
                s.close();
        }
}
```

-TCPClient-

```java
import java.io.*;
import java.net.*;
class Client1
{
        public static void main(String ar[])throws Exception
        {
                Socket s=new Socket("localhost",777);
                InputStream obj=s.getInputStream();
        BufferedReader br=new BufferedReader(new InputStreamReader(obj));
                String str;
                while((str=br.readLine())!=null)
                {
                        System.out.println("From Server: "+str);
                }
                br.close();
                s.close();
        }
}
```

UDP
-Server-
```
import java.io.*;
import java.net.*;
class UDPServer
{
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true)
        {
            DatagramPacket receivePacket =
                 new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String( receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket =
          new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

-UDPClient-
```
import java.io.*;
import java.net.*;

class UDPClient
{
    public static void main(String args[]) throws Exception
    {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();

        InetAddress IPAddress = InetAddress.getByName("localhost");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

```
        clientSocket.send(sendPacket);

        DatagramPacket receivePacket =
                    new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());
        System.out.println("FROM SERVER:" + modifiedSentence);
        clientSocket.close();
    }
}
```

Q7. **Write a client-server program using UDP socket. Client send the list of N strings and server responds the concatenation of those strings.**

Ans. **Dsender.java**
```
import java.net.*;
import java.util.Scanner;
public class Dsender{
    public static void main(String[] args) throws Exception{
        DatagramSocket ds=new DatagramSocket();
        DatagramPacket dp;
        InetAddress ip=InetAddress.getByName("localhost");
        String str;
        Scanner sc=new Scanner(System.in);
        while(true){
        System.out.print("Enter Msg:");
        str=sc.nextLine();
        dp=new DatagramPacket(str.getBytes(),str.length(),ip,3000);
        ds.send(dp);
        }
    }}
```
Dreceiver.java
```
import java.net.*;
public class Dreceiver{
    public static void main(String[] args)throws Exception{
        String str="",concat="";
        DatagramSocket ds=new DatagramSocket(3000);
        byte[] buf;
        DatagramPacket dp;
      while(true){
        buf=new byte[1024];
        dp=new DatagramPacket(buf, 1024);
        ds.receive(dp);
        str=new String(dp.getData(),0,dp.getLength());
        if( !str.equals("exit"))
        { concat+=str;         }
        else{break;}
        }
        System.out.println(concat);
        ds.close();}}
```

Q8. **Write a client server program using TCP where client sends a string and server checks whether that string is palindrome or not and responds with appropriate message.**

Ans.   **-Server-**

```
import java.net.*;
import java.io.*;
public class Server {

     public static void main(String args[]) throws Exception
   {
        ServerSocket ss=new ServerSocket(777);
        Socket s=ss.accept();
        System.out.println("Connection Established");
        OutputStream obj=s.getOutputStream();
        PrintStream ps=new PrintStream(obj);
        InputStream obj1=s.getInputStream();
        BufferedReader br=new BufferedReader(new outStreamReader(obj1));
        String str = br.readLine();
        String newstr = "";
        for(int i = str.length()- 1;i>=0;i-- )
        {
                  char c = str.charAt(i);
                  newstr = newstr + c;

        }
        if(str.equalsIgnoreCase(newstr))
        {
                  ps.println("string is palindrome ");
        }
        else
        {
                  ps.println("string is not palindrome ");
        }
        ps.close();
        ss.close();
        s.close();
     }
   }
```

-Client-

```
import java.net.*;
import java.io.*;
public class Client {
 public static void main(String args[]) throws Exception
 {
    Socket s=new Socket("localhost",777);
 BufferedReader kbr=new BufferedReader(new InputStreamReader(System.in));
    InputStream obj=s.getInputStream();
BufferedReader br=new BufferedReader(new InputStreamReader(obj));
                OutputStream os = s.getOutputStream();
                PrintStream ps = new PrintStream(os);
                System.out.println("Enter text");
                String str = kbr.readLine();
                ps.println(str);
```

```
                    String newStr = br.readLine();
                    System.out.println("Response from server=" + newStr);
                    br.close();
                    s.close();
                }
        }
```

Q9. Write a client-server program using TCP or UDP where the client sends 10 numbers and server responds with the numbers in sorted order.

Ans. **Server-**

```
import java.net.*;
import java.io.*;
import java.util.*;
public class Server {
    public static void main(String args[]) throws Exception
    {
            ServerSocket ss=new ServerSocket(7777);
            Socket s=ss.accept();
            System.out.println("connected..........");
            DataInputStream din=new DataInputStream(s.getInputStream());
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
            int r,i=0;
            int n=din.readInt();
            int a[]=new int[n];
            System.out.println("data:");
            int count=0;
            System.out.println("Receiving Data....");
            for(i=0;i<n;i++)
            {
                    a[i]=din.readInt();
             }
                        System.out.println("Data Received");
                        System.out.println("Sorting Data........");
                        Arrays.sort(a);
                        System.out.println("Data Sorted");
                        System.out.println("Sending Data........");
                        for(i=0;i<n;i++)
                        {
                                dout.writeInt(a[i]);
                        }
                        System.out.println("\nData Sent Successfully");
                        s.close();
                        ss.close();
            }
    }
```

-Client-

```
import java.net.*;
import java.io.*;
import java.util.*;
public class Client {
    public static void main(String[] args) throws Exception
    {   Socket s=new Socket("127.0.0.1",7777);
        if(s.isConnected())
        {   System.out.println("Connected to server");
        }
        System.out.println("Enter size of array:");
        Scanner scanner=new Scanner(System.in);
        int n=scanner.nextInt();
        int a[]=new int[n];
        System.out.println("Enter element to array:");
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());
        dout.writeInt(n);
        for(int i=0;i<n;i++)
        {
                int r=scanner.nextInt();;
                dout.writeInt(r);
        }
         System.out.println("Data Sent");
         DataInputStream din=new DataInputStream(s.getInputStream());
         int r;
         System.out.println("Receiving Sorted Data....");
         for(int i=0;i<n;i++)
         {      r=din.readInt();
                System.out.print(r+" ");
         }
         s.close();
         }
    }
```

Q10. Write a TCP Client-Server program to get the Date & Time details from Server on the Client request.

Ans.   **-Server-**

```
import java.net.*;
import java.io.*;
import java.util.Date;
public class Server {
    public static void main(String args[]) throws Exception
    {   ServerSocket ss=new ServerSocket(777);
        while(true)
        {   System.out.println("Waiting For Connection ...");
            Socket soc=ss.accept();
            DataOutputStream out=new DataOutputStream(soc.getOutputStream());
            out.writeBytes("Server Date  " + (new Date()).toString() + "\n");
            out.close();
            soc.close();
         }}}
```

-Client-

```
import java.net.*;
import java.io.*;
public class Client {
    public static void main(String args[]) throws Exception
    {
        Socket s=new Socket("localhost",777);
        BufferedReader in=
            new BufferedReader(new InputStreamReader(s.getInputStream()));
        System.out.println(in.readLine());
    }
}
```

Q1. What is JDBC? Explain the types of JDBC drivers?
Ans.
What is JDBC?
- JDBC is an API, which is used in java programming for interacting with database.
- JDBC (Java Data Base Connection) is the standard method of accessing **databases** from **Java application**.
- JDBC is a specification from **Sun Microsystem** that provides a **standard API** for java application to communicate with different database.
- JDBC is a **platform independent** interface between relational database and java applications.

JDBC Drivers
1. **Type1 (JDBC-ODBC Driver)**
   - Depends on support for ODBC
   - Type1 is not portable driver
   - Translate JDBC calls into ODBC calls and use Windows ODBC built in drivers
   - ODBC must be set up on every client
   - For server side servlets ODBC must be set up on web server
   - Driver sun.jdbc.odbc.JdbcOdbc provided by JavaSoft with JDK
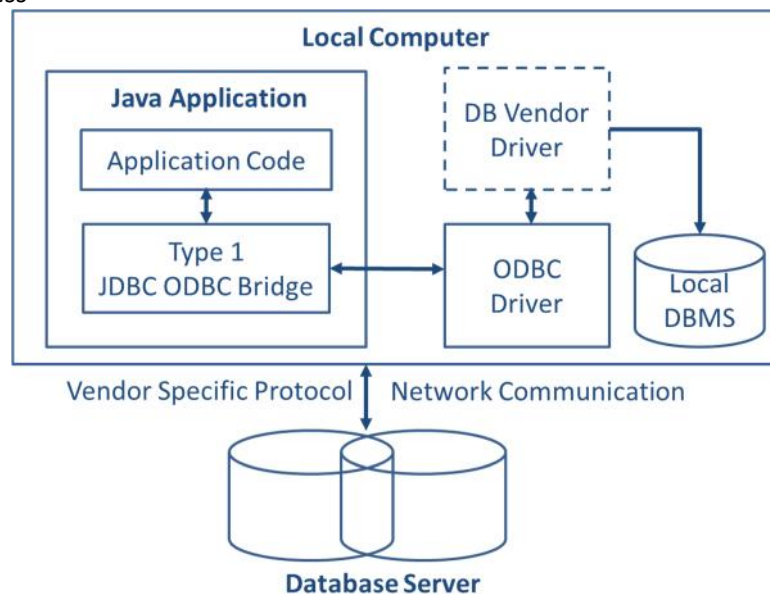   - No support from JDK 1.8 (Java 8) onwards.
     E.g. MS Access

Figure: Type1 (JDBC-ODBC Driver)

Advantages :
   - Allow to communicate with all database supported by ODBC driver
   - It is vendor independent driver
Disadvantages:
   - Due to large number of translations, execution speed is decreased
   - Dependent on the ODBC driver
   - ODBC binary code or ODBC client library to be installed in every client machine
   - Uses java native interface to make ODBC call

- Because of listed disadvantage, type1 driver is not used in production environment. It can only be used, when database doesn't have any other JDBC driver implementation.

2. **Type 2 (Native Code Driver)**
   - JDBC API calls are converted into native API calls, which are unique to the database.
   - These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge.
   - Native code Driver are usually written in C, C++.
   - The vendor-specific driver must be installed on each client machine.
   - Type 2 Driver is suitable to use with server side applications.
   - E.g. Oracle OCI driver, Weblogic OCI driver, Type2 for Sybase
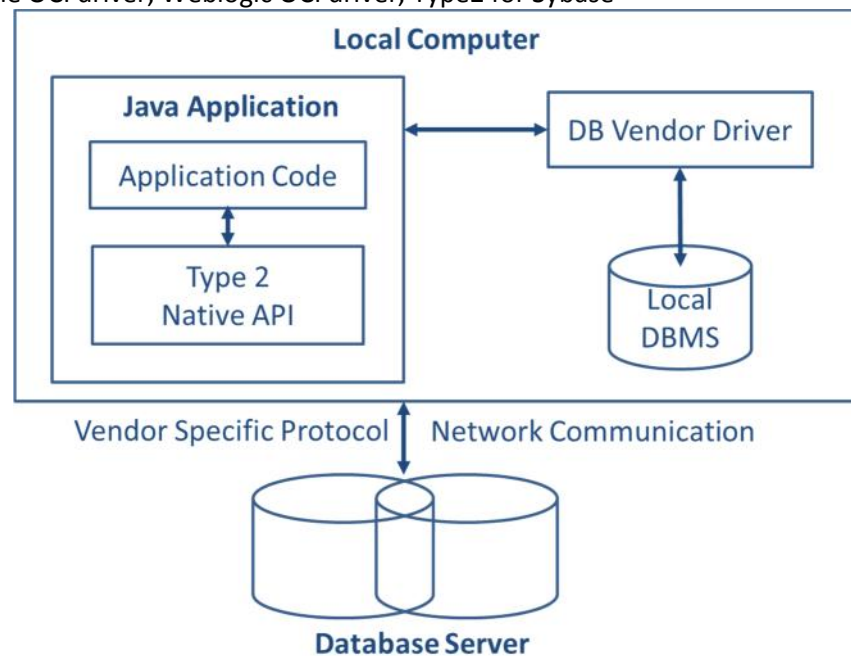
Figure: Type 2 (Native Code Driver)

Advantages
- As there is no implementation of JDBC-ODBC bridge, it may be considerably faster than a Type 1 driver.

Disadvantages
- The vendor client library needs to be installed on the client machine hence type 2 drivers cannot be used for the Internet.
- This driver is platform dependent.
- This driver supports all java applications except applets.
- It may increase cost of application, if it needs to run on different platform (since we may require buying the native libraries for all of the platform).
- Mostly obsolete now
- Usually not thread safe

3. **Type 3 (Java Protocol)**
   - This driver translate the jdbc calls into a database server independent and middleware server specific calls.
   - With the help of the middleware server, the translated jdbc calls further translated into database server specific calls.
   - This type of driver also known as net-protocol fully java technology-enabled driver.
   - Type-3 driver is recommended to be used with applets. its auto-downloadable.
   - Can interface to multiple databases – Not vendor specific.
   - Follows a three-tier communication approach.
   - The JDBC clients use standard network sockets to communicate with a middleware application server.
   - The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
   - This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.
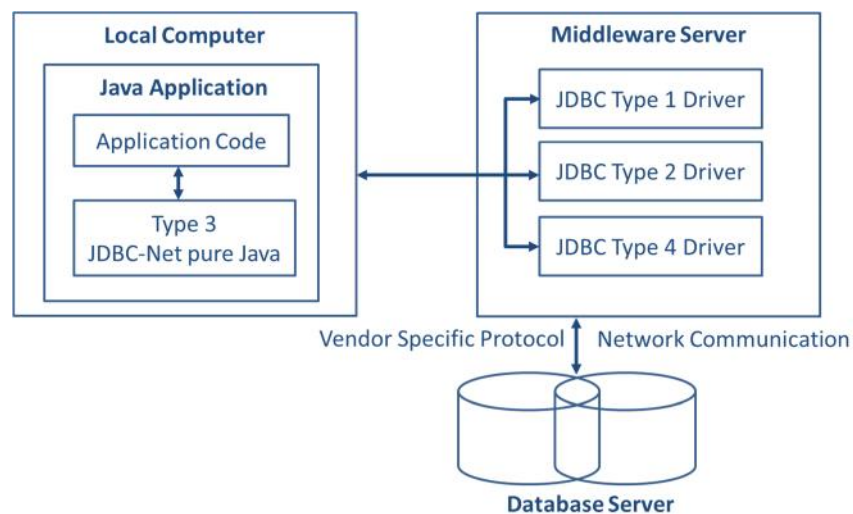

Figure: Type 3 (Java Protocol)

Advantages
   - Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client.
   - A single driver can handle any database, provided the middleware supports it.
   - We can switch from one database to other without changing the client-side driver class, by just changing configurations of middleware server.
     E.g.: IDS Driver, Weblogic RMI Driver

Disadvantages
   - Compared to Type 2 drivers, Type 3 drivers are slow due to increased number of network calls.
   - Requires database-specific coding to be done in the middle tier.
   - The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.

4. **Type 4 (Database Protocol)**
   - It is known as the Direct to Database **Pure Java Driver**
   - Need to download a new driver for each database engine
   - Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection.
   - This kind of driver is extremely flexible, you don't need to install special software on the client or server.
   - This type of driver is lightweight and generally known as thin driver.
   - You can use this driver when you want an auto downloadable option the client side application
   - i.e. thin driver for oracle from oracle corporation, weblogic and ms sqlserver4 for ms sql server from BEA system
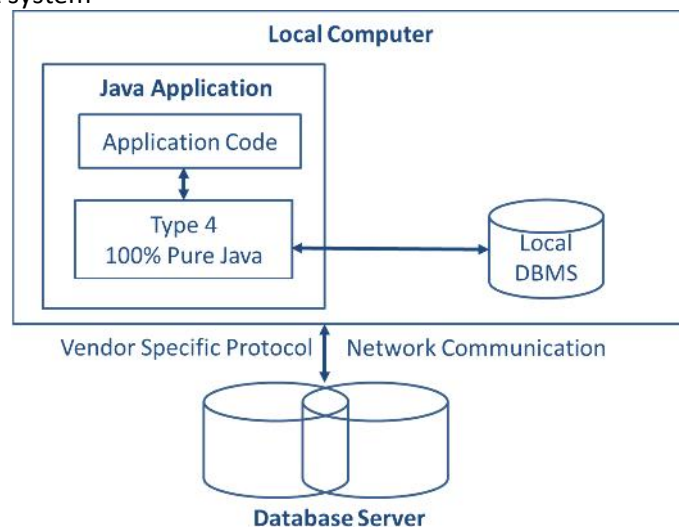

Figure: Type 4 (Database Protocol)

Advantages
   - Completely implemented in Java to achieve platform independence.
   - No native libraries are required to be installed in client machine.
   - These drivers don't translate the requests into an intermediary format (such as ODBC).
   - Secure to use since, it uses database server specific protocol.
   - The client application connects directly to the database server.
   - No translation or middleware layers are used, improving performance.
   - The JVM manages all the aspects of the application-to-database connection.

Disadvantage
   - This Driver uses database specific protocol and it is DBMS vendor dependent.

Comparison between JDBC Drivers

Type	Type 1	Type 2	Type 3	Type 4
Name	JDBC-ODBC Bridge	Native Code Driver/ JNI	Java Protocol/ Middleware	Database Protocol
Vendor Specific	No	Yes	No	Yes
Portable	No	No	Yes	Yes
Pure Java Driver	No	No	Yes	Yes
Working	JDBC-> ODBC call ODBC -> native call	JDBC call -> native specific call	JDBC call -> middleware specific. Middleware -> native call	JDBC call ->DB specific call
Multiple DB	Yes [only ODBC supported DB]	NO	Yes [DB Driver should be in middleware]	No
Example	MS Access	Oracle OCI driver	IDA Server	MySQL
Execution Speed	Slowest among all	Faster Compared to Type1	Slower Compared to Type2	Fastest among all
Driver	Thick Driver	Thick Driver	Thin Driver	Thin Driver

Q2. **Explain Thick and Thin driver. Comment on selection of driver. Write code snippet for each type of JDBC connection.**

Ans. **Thick driver**
- Thick client would need the client installation.
  E.g. Type 1 and Type 2.

Thin driver
- The thin client driver, which mean you can connect to a database without the client installed on your machine.
  E.g. Type 4

Comment on selection of driver
- If you are accessing one type of database such as MySQL, Oracle, Sybase or IBM etc., the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Write code snippet for each type of JDBC connection
1. **MySQL**
```
Class.forName("com.mysql.jdbc.Driver");

Connection conn=
DriverManager.getConnection("jdbc:mysql://localhost:PortNo/database
Name","uid", "pwd");
```

2. **Oracle**
```
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn=
DriverManager.getConnection("jdbc:oracle:thin:@hostname:port
Number:databaseName","root", "pwd");
```

3. **DB2**
```
Class.forName("com.ibm.db2.jdbc.net.DB2Driver");

Connection conn=
DriverManager.getConnection("jdbc:db2:hostname:port Number
/databaseName")
```

Q3.   Explain Statement Interface with appropriate example.

Ans.   **Java.sql.Statement**

- Used for general-purpose access to your database.
- Useful for **static** SQL statements, e.g. SELECT specific row from table etc.
- The Statement interface defines a standard abstraction to execute the SQL statements requested by a user and return the results by using the ResultSet object.
- The Statement interface is created after the connection to the specified database is made.
- The object is created using the createStatement() method of the Connection interface, as shown in following code snippet:

```
Statement stmt = con.createStatement();
```

```
1.  import java.sql.*;
2.  public class ConnDemo {
3.  public static void main(String[] args) {
4.    try {
5.         // Load and register the driver
6.          Class.forName("com.mysql.jdbc.Driver");

7.         // Establish the connection to the database server
8.          Connection conn= DriverManager.getConnection
9.          ("jdbc:mysql://localhost:3306/database_name","root","pwd");
10.         // Create a statement
11.     Statement stmt = conn.createStatement();
12. // Execute the statement
13.          ResultSet rs = stmt.executeQuery("SELECT * from Table");

14.     // Retrieve the results
15.      while(rs.next()){
16.         System.out.print(rs.getInt(1)+"\t");
17.         System.out.print(rs.getString("Name")+"\t");
18.                System.out.println(rs.getString(3));
19.      }//while

20. // Close the statement and connection
21.          stmt.close();
22.          conn.close();
23.  }catch(Exception e){System.out.println(e.toString());
24. }//PSVM
25. }//class
```

Q4.   Explain Prepared Statement with example.

Ans.
- The PreparedStatement interface is subclass of the Statement interface, can be used to represent a precompiled query, which can be executed multiple times.
- Prepared Statement is used when you plan to execute same SQL statements many times.
- PreparedStatement interface accepts input parameters at runtime.
- A SQL statement is precompiled and stored in a PreparedStatement object.
- This object can then be used to efficiently execute this statement multiple times.
- The object is created using the prepareStatement() method of Connection interface, as shown in following snippet:

```
String query = "insert into emp values(? ,?)";
PreparedStatement ps = con.prepareStatement(query);
ps.setInt(1,5);
ps.setString(2,"New Employee");
int n = ps.executeUpdate();
```

Advantages:
- The performance of the application will be faster, if you use PreparedStatement interface because query is compiled only once.
- This is because creating a PreparedStatement object by explicitly giving the SQL statement causes the statement to be precompiled within the database immediately.
- Thus, when the PreparedStatement is later executed, the DBMS does not have to recompile the SQL statement.
- Late binding and compilation is done by DBMS.
- Provides the programmatic approach to set the values.

Disadvantage:
The main disadvantage of PreparedStatement is that it can represent only one SQL statement at a time.

Example of PreparedStatement
Write a program to insert student records to database using prepared statement
```
1. import java.sql.*;
2. public class PreparedInsert {
3. public static void main(String[] args) {
4. try {
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection conn= DriverManager.getConnection
7. ("jdbc:mysql://localhost:3306/DIET", "root","pwd");

8. String query="insert into dietstudent values(?,?,?,?)";
9. PreparedStatement ps=conn.prepareStatement(query);
10.     ps.setString(1, "14092"); //Enr_no
11.     ps.setString(2, "abc_comp"); //Name
12.     ps.setString(3, "computer"); //Branch
13.     ps.setString(4, "cx"); //Division
```

```
14.        int i=ps.executeUpdate();
15.        System.out.println("no. of rows updated ="+i);
16.        ps.close();
17.        conn.close();
18.        }catch(Exception e){System.out.println(e.toString());} }//PSVM
   }//class
```

Q5. Explain Callable Statement with example.

Ans.
- CallableStatement interface is used to call the stored procedures.
- Therefore, the stored procedure can be called by using an object of the CallableStatement interface.
- The object is created using the prepareCall() method of Connection interface.

```
CallableStatement cs=conn.prepareCall("{call Proc_Name(?,?)}");
cs.setInt(1,2222);
cs.registerOutParameter(2,Types.VARCHAR);
cs.execute();
```

- Three types of parameters exist: IN, OUT, and INOUT.
- PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from the OUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

Example of CallableStatement
Writa a Callable Statement program to retrieve branch of the student using {getBranch() procedure} from given enrollment number. Also write code for Stored Procedure
Stored Procedure: getbranch()

```
1.  DELIMITER @@
2.  DROP PROCEDURE getbranch @@
3.  CREATE PROCEDURE databaseName.getbranch
4.  (IN enr_no INT, OUT my_branch VARCHAR(10))
5.  BEGIN
6.  SELECT branch INTO my_branch
7.  FROM dietStudent
8.  WHERE enr_no=enrno;
9.  END @@
10. DELIMITER ;
```

Callable Statement program

```
1.  import java.sql.*;
2.  public class CallableDemo {
3.  public static void main(String[] args) {
4.  try {
5.    Class.forName("com.mysql.jdbc.Driver");
6.    Connection conn= DriverManager.getConnection
7.         ("jdbc:mysql://localhost:3306/Diet", "root","pwd");
8.
9.    CallableStatement cs=conn.prepareCall("{call getbranch(?,?)}");
10.            cs.setInt(1,2222);
11.            cs.registerOutParameter(2,Types.VARCHAR);
12.            cs.execute();
13.            System.out.println("branch="+cs.getString(2));
14.            cs.close();
15.            conn.close();
16.         }catch(Exceptione){System.out.println(e.toString());}
17.         }//PSVM
18. }//class
```

Q6.   Differentiate Statement, Prepared Statement and Callable Statement.

Ans.

Statement	Prepared Statement	Callable Statement
Super interface for Prepared and Callable Statement	extends Statement (sub-interface)	extends PreparedStatement (sub-interface)
Used for executing simple SQL statements like CRUD (create, retrieve, update and delete	Used for executing dynamic and pre-compiled SQL statements	Used for executing stored procedures

The Statement interface cannot accept parameters.	The PreparedStatement interface accepts input parameters at runtime.	The CallableStatement interface can also accept runtime input parameters.
stmt = conn.createStatement();	PreparedStatement ps=con.prepareStatement ("insert into studentDiet values(?,?,?)");	CallableStatement cs=conn.prepareCall("{call getbranch(?,?)}");
java.sql.Statement is slower as compared to Prepared Statement in java JDBC.	PreparedStatement is faster because it is used for executing precompiled SQL statement in java JDBC.	None
java.sql.Statement is suitable for executing DDL commands - CREATE, drop, alter and truncate in java JDBC.	java.sql.PreparedStatement is suitable for executing DML commands - SELECT, INSERT, UPDATE and DELETE in java JDBC.	java.sql.CallableStatement is suitable for executing stored procedure.

Q7. Explain JDBC Architecture.

Ans. **JDBC API**
- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- JDBC API provides **classes** and **interfaces** to connect or communicate Java application with database.
- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –
  1. JDBC API: This provides the application-to-JDBC Manager connection.
  2. JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

JDBC Driver Manager (*Class*)
- This class manages a list of database drivers.
- It ensures that the correct driver is used to access each data source.
- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.
- Matches connection requests from the java application with the proper database driver using communication sub protocol.
- The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
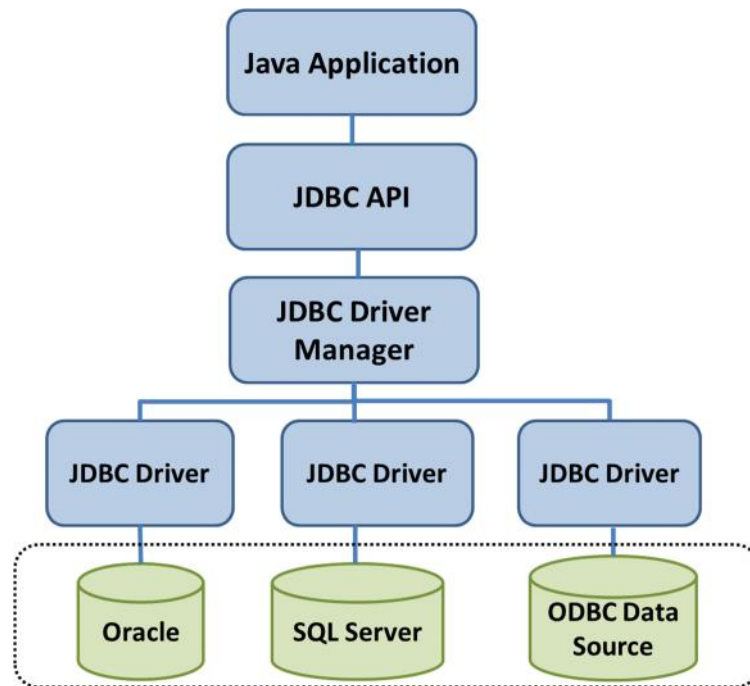
Figure: JDBC Architecture

Driver (*Interface*)
- This interface handles the communications with the database server.
- You will interact directly with Driver objects very rarely.
- Instead, you use DriverManager objects, which manages objects of this type.
- It also abstracts the details associated with working with Driver objects.

Connection (*Interface*)
- This interface with all methods for contacting a database.
- The connection object represents communication context, i.e., all communication with database is through connection object only.

Statement (*Interface*)
- You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

ResultSet (*Interface*)
- These objects hold data retrieved from a database after you execute an SQL query using Statement objects.
- It acts as an iterator to allow you to move through its data.

SQLException (*Class*)
This class handles any errors that occur in a database application.

Q8. Explain methods of ResultSet Interface.
Ans. **Categories**

1.	Navigational methods	Used to move the cursor around.
2.	Get methods	Used to view the data in the columns of the current row being pointed by the cursor.
3.	Update methods	Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

ResultSet: Navigational methods

boolean first() throws SQLException	Moves the cursor to the first row.
boolean last() throws SQLException	Moves the cursor to the last row.
boolean next() throws SQL Exception	Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
boolean previous() throws SQLException	Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
boolean absolute(int row) throws SQLException	Moves the cursor to the specified row.
boolean relative(int row) throws SQLException	Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
int getRow() throws SQLException	Returns the row number that the cursor is pointing to.

ResultSet: Get methods

int getInt (String columnName) throws SQLException	Returns the integer value to the current row in the column named columnName.
int getInt (int columnIndex) throws SQLException	Returns the integer value to the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

String getString (String columnLabel) throws SQLException	Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
String getString (int columnIndex) throws SQLException	Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.

ResultSet: Update methods

void updateString (int col_Index, String s) throws SQLException	Changes the String in the specified column to the value of s.
void updateInt (int col_Index, int x) throws SQLException	Updates the designated column with an integer value.
void updateFloat (int col_Index, float x) throws SQLException	Updates the designated column with a float value.
void updateDouble (int col_Index,double x) throws SQLException	Updates the designated column with a double value.

Q9. Differentiate executeQuery(), executeUpdate() and execute() with appropriate example.

Ans.

executeQuery()	executeUpdate()	execute()
ResultSet executeQuery (String sql) throws SQLException	int executeUpdate(String sql) throws SQLException	Boolean execute(String sql) throws SQLException
This is used generally for reading the content of the database. The output will be in the form of ResultSet. Generally SELECT statement is used.	This is generally used for altering the databases. Generally DROP, INSERT, UPDATE, DELETE statements will be used in this. The output will be in the form of int. This int value denotes the number of rows affected by the query.	If you dont know which method to be used for executing SQL statements, this method can be used. This will return a boolean. TRUE indicates the result is a ResultSet and FALSE indicates it has the int value which denotes number of rows affected by the query.
E.g.:ResultSet rs= stmt.executeQuery(query);	E.g.: int i= stmt.executeUpdate(query);	E.g.: Boolean b= stmt.execute(query);

Q10. Explain Resultset Type and Concurrency

Ans. Resultset Type

ResultSet.*TYPE_FORWARD_ONLY*	The cursor can only move forward in the result set. (Default Type)
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Concurrency of ResultSet

ResultSet.*CONCUR_READ_ONLY*	Creates a read-only result set. (Default Type)
ResultSet.*CONCUR_UPDATABLE*	Creates an updateable result set.

Example
```
Statement stmt = conn.createStatement(
                        ResultSet.TYPE_FORWARD_ONLY,
                        ResultSet.CONCUR_READ_ONLY);
```

Q11. Explain ResultsetMetaData Interface with Example.

Ans.
- Metadata means data about data.
- If you have to get metadata of a table like
  1. total number of column
  2. column name
  3. column type etc.
- ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Example: ResultSetMetaData
```
1. import java.sql.*;
2. public class MetadataDemo {
3. public static void main(String[] args) {
4.     try {Class.forName("com.mysql.jdbc.Driver");
5.         Connection conn= DriverManager.getConnection
6.         ("jdbc:mysql://localhost:3306/gtu", "root","pwd");
7. Statement stmt = conn.createStatement
   (ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_ONLY);
8.  ResultSet rs = stmt.executeQuery("SELECT * from gtu");

9.      ResultSetMetaData rsmd=rs.getMetaData();
10.     System.out.println("Total columns:
                            "+rsmd.getColumnCount());
11.     System.out.println("Column Name of 1st column:
                            "+rsmd.getColumnName(1));
12.     System.out.println("Column Type Name of 1st column:"
                    +rsmd.getColumnTypeName(1));
13.         stmt.close();
14.         conn.close();
15.       }catch(Exception e)
16.     {System.out.println(e.toString());}
17.      }//PSVM
18.     }//class
```
OUTPUT:
```
Total columns: 3
Column Name of 1st column:Enr_no
Column Type Name of 1st column:INT
```

Q12. Explain DatabaseMetaData Interface with Example

Ans. DatabaseMetaData interface provides methods to get meta data of a database such as

1. Database product name
2. Database product version
3. Driver name
4. Name of total number of tables etc.

Example: DatabaseMetaData

```java
import java.sql.*;
public class DatabaseMetaDataDemo {
 public static void main(String[] args) {
  try {
       Class.forName("com.mysql.jdbc.Driver");
       Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/temp6",
"root","root");
       DatabaseMetaData dbmd=con.getMetaData();
       System.out.println
      ("getDatabaseProductName:"+dbmd.getDatabaseProductName());
       System.out.println("getDatabaseProductVersion():
                         "+dbmd.getDatabaseProductVersion());

    System.out.println("getDriverName():"+dbmd.getDriverName())
                                                    ;
      System.out.println("getDriverVersion():
                             "+dbmd.getDriverVersion());
     System.out.println("getURL():"+dbmd.getURL());
System.out.println("getUserName():"+dbmd.getUserName());

          } catch (Exception ex) {
            System.out.println("Exception:"+ex.toString());
        }}}
```

OUTPUT:
```
getDatabaseProductName:MySQL
getDatabaseProductVersion():5.6.16
getDriverName():MySQL-AB JDBC Driver
getDriverVersion():mysql-connector-java-5.1.23 ( Revision:
                             ${bzr.revision-id} )
getURL():jdbc:mysql://localhost:3306/temp6
getUserName():root@localhost
```

Q13. Explain Transaction Management in JDBC with appropriate example.

Ans.
- Transaction Management in java is required when we are dealing with relational databases.
- By default when we create a database connection, it runs in **auto-commit** mode.
- It means that whenever we execute a query and it's completed, the commit is fired automatically.
- So every SQL query we fire is a transaction and if we are running some DML or DDL queries, the changes are getting saved into database after every SQL statement finishes.
- Sometimes we want a group of SQL queries to be part of a transaction so that we can commit them when all the queries runs fine. If we get any exception, we have a choice of rollback all the queries executed as part of the transaction.
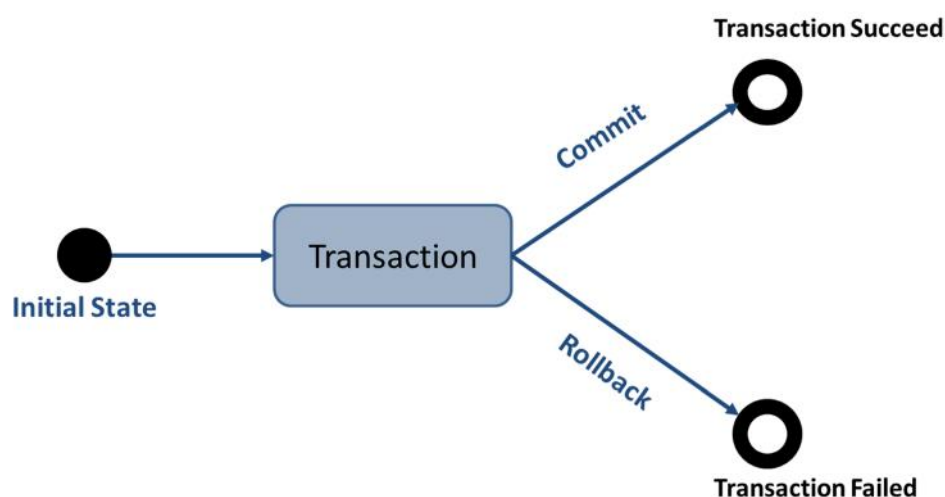
Figure: Transaction Management

Advantage of Transaction Management
- **Fast performance:** It makes the performance fast because database is hit at the time of commit.

In JDBC, **Connection** interface provides methods to manage transaction.

void setAutoCommit(boolean status)	It is true by default means each transaction is committed by default.
void commit()	Commits the transaction.
void rollback()	Cancels the transaction.

Example

```
1. import java.sql.*;
2. class RollbackDemo{
3. public static void main(String args[]){
4. try{ Class.forName("com.mysql.jdbc.Driver");
5.    Connection con=DriverManager.getConnection(
6.             "jdbc:mysql://localhost:3306/GTU","root","root");
7.    con.setAutoCommit(false);//bydeafault it is true
8.    Statement stmt=con.createStatement();
9.    int i=stmt.executeUpdate("insert into diet
                                    values(606,'ghi','ee')");
10.   con.commit(); //Commit Transaction
11.   i+=stmt.executeUpdate("insert into diet
                                    values(607,'mno','ch')");
12.      System.out.println("no. of rows inserted="+i);
13.      con.rollback(); //Rollback Transaction
14.      con.close();
15.   }catch(Exception e){ System.out.println(e);}
16. }}
```

Q14. Explain Transaction Isolation Level in JDBC

Ans.
- JDBC isolation level represents that, how a database maintains its interiority against the problem such as
  1. dirty reads
  2. non-repeatable reads
  3. phantom reads

  that occurs during concurrent transactions.

What is Phantom read?
- At the time of execution of a transaction, if two queries that are identical are executed, and the rows returned are different from one another.
- If you execute a query at time T1 and re-execute it at time T2, additional rows may have been added to the database, which may affect your results. It is stated that a phantom read occurred.

What is Dirty read?
- Dirty read occurs when one transaction is changing the record, and the other transaction can read this record before the first transaction has been committed or rolled back.
- This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid data.

E.g.

Transaction A begins	Transaction B begins
UPDATE EMPLOYEE SET SALARY = 10000 WHERE EMP_ID= '123';	**SELECT * FROM EMPLOYEE;** (Transaction B sees data which is updated by transaction A. But, those updates have not yet been committed.)

What is Non-Repeatable Read?
- Non Repeatable Reads happen when in a **same transaction** same query yields to a different result.
- This occurs when one transaction repeatedly retrieves the data, while a difference transactions alters the underlying data.
- This causes the different or non-repeatable results to be read by the first transaction.

Transaction Isolation Level

Initial Val.	Isolation Level	Description
1	TRANSACTION_READ_UNCOMMITTED	It allows non-repeatable reads, dirty reads and phantom reads to occur.
2	TRANSACTION_READ_COMMITTED	It ensures only those data can be read which is committed.
4	TRANSACTION_REPEATABLE_READ	It is closer to serializable, but phantom reads are also possible.
8	TRANSACTION_SERIALIZABLE	In this level of isolation dirty reads, non-repeatable reads, and phantom reads are prevented.

You can get/set the current isolation level by using method
1. getTransactionIsolation()
2. setTransactionIsolation(int isolationlevelconstant)

Example
```
con.setTransactionIsolation(8);
System.out.println("con.getTransactionIsolation():"
                              +con.getTransactionIsolation());
```

Q15. Explain Batch Processing in JDBC

Ans.
- Instead of executing a single query, we can execute a batch (group) of queries.
- It makes the performance fast.
- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Methods of Statement Interface

void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

```
1. Class.forName("com.mysql.jdbc.Driver");
2. Connection con=DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/GTU","root","root");
3. con.setAutoCommit(false);
4. Statement stmt=con.createStatement();
5. String query1,query2,query3,query4,query5;
6. query1="create table DietStudent(enr INT PRIMARY
   KEY, name VARCHAR(20),sem INT,branch
              VARCHAR(10))";
7. query2="insert into DietStudent
              values(6001,'java',6,'ce')";
8. query3="insert into DietStudent
              values(6002,'php',6,'ce')";
9. query4="update DietStudent set name='cg' where
                              enr=6002";
10.     query5="delete from DietStudent where
                              name='java'";
11.     stmt.addBatch(query1);
12.     stmt.addBatch(query2);
13.     stmt.addBatch(query3);
14.     stmt.addBatch(query4);
15.     stmt.addBatch(query5);
16.     int[] i=stmt.executeBatch();
17.     con.commit();
```

GTU Questions

1. What is JDBC?   [Win -14]
   List out different types of JDBC driver and explain role of each.   [Sum -15]
   Write code snippet for each type of JDBC connection.   [Win -15]
   Explain Thick and Thin driver.   [Sum -16]
   Comment on selection of driver.   [Win -16]

2. Explain Prepared statements with suitable example   [Win -15]
                                                                                                          [Sum -16]
                                                                                                          [Win -16]
                                                                                                          [Win -17]

3. Give the use of Statement, PreparedStatement and CallableStatement object.   [Win -14]
   Write code to insert three records into student table using PreparedStatement
   (assume student table with Name, RollNo, and Branch field).

4. What is phantom read in JDBC? Which isolation level prevents it?   [Sum -16]

5. Discuss CallableStatement with example.   [Win -17]

Q1. **What is Servlet? List and Explain various stages of Servlet life cycle. Explain role of web container.**

Ans. **What is Servlet?**

Servlet is java class which extends the functionality of web server by dynamically generating web pages. Servlet technology is used to create Dynamic web application.

List and Explain various stages of Servlet life cycle

In the life cycle of servlet there are three important methods. These methods are

1. init()
2. service()
3. destroy()
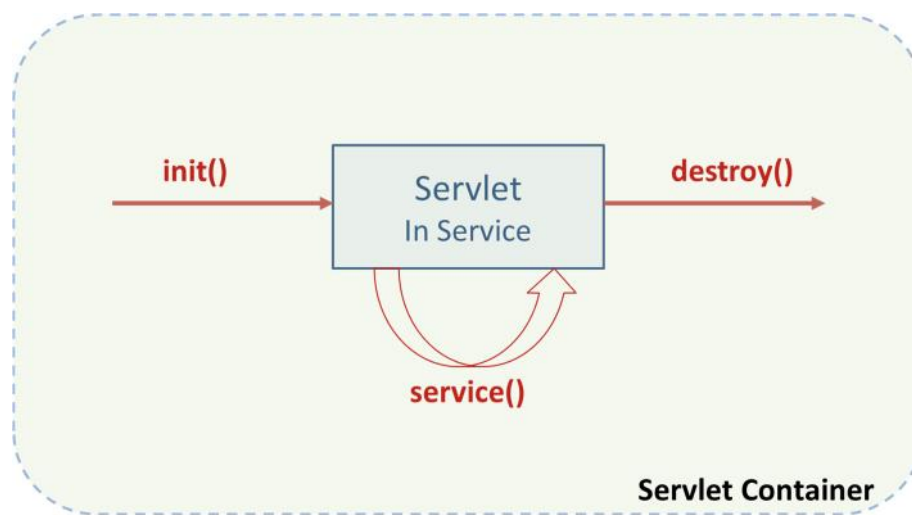
Figure: Servlet Life Cycle

- The client enters the URL in the web browser and makes a request. The browser then generates the HTTP request and sends it to the Web server.
- Web server maps this request to the corresponding servlet.

init()

- The server basically invokes the **init()** method of servlet. This method is called only when the servlet is loaded in the memory for the first time.
- The class loader is responsible to load the servlet class.
- The servlet class is loaded when the first request for the servlet is received by the web container.
- The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

- The web container calls the init method only once after creating the servlet instance. The init() method is used to initialize the servlet.

```
public void init(ServletConfig config)throws ServletException
{
        //Servlet Initialization…
}
```

- A servlet configuration object used by a servlet container to pass information to a servlet during initialization.

service()

- 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 response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.

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

destroy()

- Finally server unloads the servlet from the memory using 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
  1. database connections,
  2. halt background threads,
  3. write cookie lists or hit counts to disk, and
  4. perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.

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

Example

```
1. import java.io.*;
2. import javax.servlet.*;
3. public class MyServlet1 extends GenericServlet
4. {
5. public void init() throws ServletException
6. {//Initailization Code
7. }
8. public void service(ServletRequest request,
                       ServletResponse response)
                              throws ServletException,IOException
9. {    //Servlet code
10.     }

11.     public void destroy()
12.     {//Finalization Code
13. }}
```

Q2. Differentiate Servlets and CGI
Ans.

CGI(Common Gateway Interface)	Servlet
CGI is not portable (as CGI programs written inside the native language).	Servlets are portable (written in java).
In CGI each request is handled by heavy weight OS process.	In Servlets each request is handled by lightweight Java Thread.
CGI is more expensive than Servlets, because For each request CGI Server receives, It creates new Operating System Process.	Servlets is inexpensive than CGI because In Servlet, All the requests coming from the Client are processed with the threads instead of the OS process.
Session tracking and caching of previous computations cannot be performed.	Session tracking and caching of previous computations can be performed
CGI cannot handle cookies	Servlets can handle cookies
CGI does not provide sharing property.	Servlets can share data among each other.

Q3. Differentiate GenericServlet and HttpServlet
Ans.

GenericServlet	HttpServlet
javax.servlet.GenericServlet (abstract class)	javax.servlet.http.HttpServlet (abstract class)
It is the immediate subclass of Servlet interface.	The immediate super class of HttpServlet is GenericServlet.
It defines a generic, protocol-independent servlet. it can be used with any protocol, say, SMTP, FTP, CGI including HTTP etc.	It defines a HTTP protocol specific servlet.
GenericServlet is a super class of HttpServlet class.	HttpServlet is a sub class of GenericServlet class.
All methods are concrete except service() method. service() method is abstract method.	All methods are concrete (non-abstract). service() is non-abstract method. service() can be replaced by doGet() or doPost() methods.

Q4. Differentiate doGet() vs doPost()
Ans.
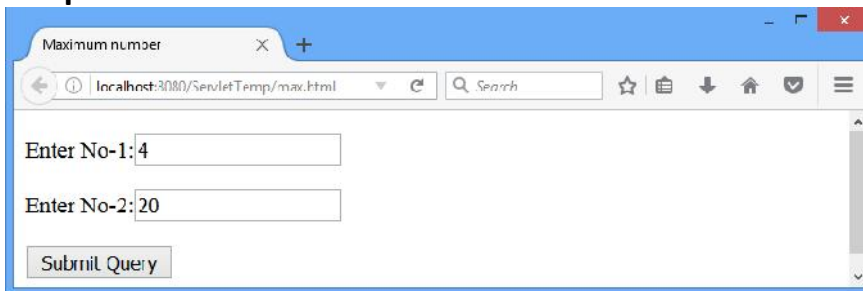
doGet()	doPost()
In doGet(), parameters are appended to the URL and sent along with header information	In doPost(), parameters are sent in separate line in the body
Maximum size of data that can be sent using doGet() is 240 bytes	There is no maximum size for data
Parameters are not encrypted	Parameters are encrypted here
Application: Used when small amount of data and insensitive data like a query has to be sent as a request. It is default method.	**Application**: Used when comparatively large amount of sensitive data has to be sent. E.g. submitting sign_in or login form.
doGet() is faster comparatively	doPost() is slower compared to doGet() since doPost() does not write the content length
doGet() generally is used to query or to get some information from the server	Dopost() is generally used to update or post some information to the server
This is default method of http	Not the case

Q5. **Write a Servlet program using doPost() to enter two numbers and find maximum among them.**

Ans. **max.html**

```
1. <html>
2.     <head>
3.         <title> Maximum number </title>
4.     </head>
5.     <body>
6.         <form action="/ServletTemp/Max" method="POST" >
7.             <p>Enter No-1:<input type="text" name="no1"></p>
8.             <p>Enter No-2:<input type="text" name="no2"></p>
9.             <p><input type="submit"></p>
10.        </form>
11.     </body>
12. </html>
```

Output: max.html

Max.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4. public class Max extends HttpServlet
5. {    public void doPost(HttpServletRequest request,
                           HttpServletResponse response)
                               throws ServletException,IOException
6.     {    int n1=0,n2=0;
7.          response.setContentType("text/html");
8.          PrintWriter out=response.getWriter();

9.          n1=Integer.parseInt(request.getParameter("no1"));
10.         n2=Integer.parseInt(request.getParameter("no2"));
11.
12.         if(n1>n2)
13.             out.println("n1="+n1+"is max number");
14.         else if(n2>n1)
```

```
15.             out.println("n2="+n2+"is max number");
16.        else if(n1==n2)
17.     out.println("n1= "+n1+"and n2="+n2+"are equal numbers");
18.          }
19.      }
```

Output:Max.java

n2=20is max number
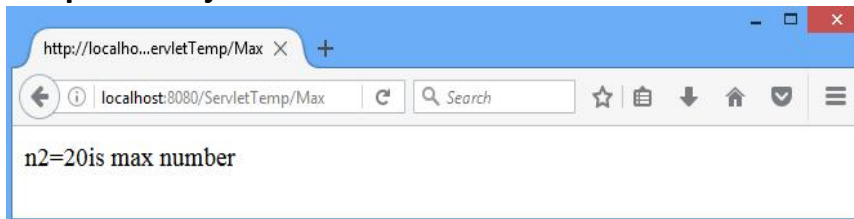
Q6.   Explain ServletConfig with example.

Ans.
- It is used to get configuration information from web.xml file.
- If the configuration information is modified from the web.xml file, we don't need to change the servlet.

  E.g. `String str = config.getInitParameter("name")`

Advantage of ServletConfig
- The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

How to get the object of ServletConfig
- **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

Usage of ServletConfig
If any specific content is modified from time to time. you can manage the Web application easily without modifying servlet through editing the value in web.xml
E.g. `ServletConfig config=getServletConfig();`

web.xml

```
<web-app>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>MyServlet</servlet-class>
        <init-param>
            <param-name>name</param-name>
            <param-value>cxcy</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

MyServlet.java
```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class MyServlet extends HttpServlet
5. {    String msg;
6.      PrintWriter out;
7. public void init(ServletConfig config)throws
                                        ServletException
8. {         msg = config.getInitParameter("name"); }
9. public void doGet(HttpServletRequest request ,
                  HttpServletResponse response) throws
                  i. ServletException,IOException
10.      {    response.setContentType("text/html");
11.      out = response.getWriter();
12.      out.println("<h1>"+ msg +"</h1>");
13.      }
14.      public void destroy()
15.      {         out.close();      }}
```

Output: MyServlet.java
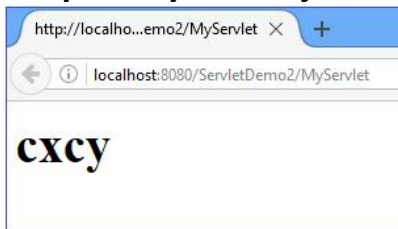
Methods of ServletConfig interface
1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

Q7.   Explain ServletContext with example
Ans.
- ServletContext is created by the web container at time of deploying the project.
- It can be used to get configuration information from web.xml file.
- There is only one ServletContext object per web application.
- If any information is shared to many servlet, it is better to provide it from the web.xml file using the <context-param> element.

Advantage of ServletContext

- **Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet.
- We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.

```
//We can get the ServletContext object from ServletConfig object
    ServletContext context=getServletConfig().getServletContext();
```

2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

```
    //Another way to get the ServletContext object
    ServletContext application=getServletContext();
```

Example of ServletContext

Web.xml

```
<web-app>
    <servlet>
        <servlet-name>ServletContextDemo</servlet-name>
        <servlet-class>ServletContextDemo</servlet-class>
    </servlet>


    <servlet-mapping>
        <servlet-name>ServletContextDemo</servlet-name>
        <url-pattern>/ServletContextDemo</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>name</param-name>
        <param-value>DIET</param-value>
    </context-param>
</web-app>
```

ServletContextDemo.java

```
1.   import java.io.*;
2.   import javax.servlet.*;
3.   import javax.servlet.http.*;
4.   public class ServletContextDemo extends HttpServlet{
5.       public void doGet(HttpServletRequest
     req,HttpServletResponse              res)  throws
     ServletException,IOException
6.   {   res.setContentType("text/html");
7.       PrintWriter out=res.getWriter();
8.       //creating ServletContext object
9.       ServletContext context=getServletContext();
10.    //Getting value of initialization parameter and printing it
11.       String college=context.getInitParameter("name");
12.       out.println("College name is="+college);
13.       out.close();
14.  }}
```

Output: ServletContextDemo.java
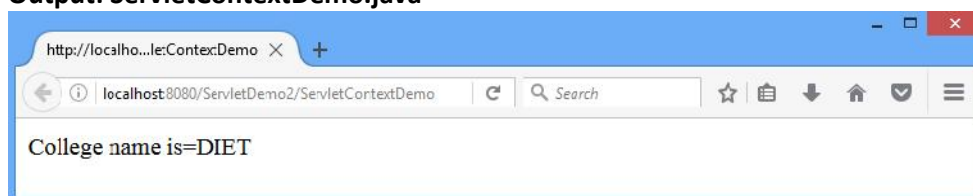
Methods of ServletContext interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

Q8. **Differentiate ServletConfig and ServletContext Interface.**

Ans.

Servlet Config	Servlet Context
ServletConfig object is one per servlet class	ServletContext object is global to entire web application
Object of ServletConfig will be created during initialization process of the servlet	Object of ServletContext will be created at the time of web application deployment
Scope: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.	Scope: As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
We should give request explicitly, in order to create ServletConfig object for the first time	ServletContext object will be available even before giving the first request
In web.xml – <init-param> tag will be appear under <servlet-class> tag	In web.xml – <context-param> tag will be appear under <web-app> tag

Q9. **Explain Methods of HttpServletRequest with appropriate example.**

Ans.

1.String **getContextPath**()	Returns the portion of the request URI that indicates the context of the request.

E.g.
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
{out.println("<p>request.getContextPath():"
                        +request.getContextPath()+"</p>");
}
```
Output:request.getContextPath():/ServletTemp

2.Enumeration **getHeaderNames()**	Returns an enumeration of all the header names this request contains.

E.g.
```
      Enumeration h=request.getHeaderNames();
      while(h.hasMoreElements())
      {
        String paramName = (String)h.nextElement();
        out.print("<p>" + paramName + "\t");
        String paramValue = request.getHeader(paramName);
        out.println( paramValue + "</p>\n");
      }
```

Output:
host localhost:8080
user-agent Mozilla/5.0 (Windows NT 6.2; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0
accept text/html,application/xhtml+xml,
       application/xml;q=0.9,*/*;q=0.8
accept-**language** en-US,en;q=0.5
accept-encoding gzip, deflate
connection keep-alive
upgrade-insecure-requests 1

3.String **getHeader**(String name)	Returns the value of the specified request header as a String.

E.g. `out.println("<p>request.getHeader():"`
                       `+request.getHeader("host")+"</p>");`
`out.println("<p>request.getHeader():"+request.getHeader("referer"`
                                       `)+"</p>");`

Output:
request.getHeader():host=localhost:8080
request.getHeader():referer=http://localhost:8080/ServletTemp/servletmeth.html

4. String **getQueryString**()	Returns the query string that is contained in the request URL after the path.

```
public void doGet(HttpServletRequest request,
           HttpServletResponse response)
{out.println("<p>request.getQueryString():"
       +request.getQueryString()+"</p>");}
```
Output:
request.getQueryString(): no1=1&no2=2
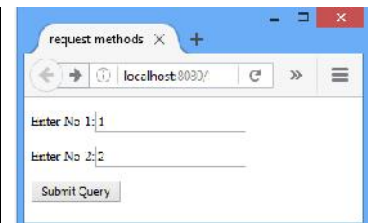
5.String **getServletPath**()	Returns the part of this request's URL that calls the servlet. This path starts with a "/" character and includes either the servlet name or a path to the servlet

E.g. `out.println("<p>request.getServletPath():"`
                       `+request.getServletPath()+"</p>");`
Output: request.getServletPath(): /ServletMeth

6. String **getMethod**()	Returns the name of the HTTP method with which this request was made, for example GET or POST

E.g. `out.println("<p>request.getMethod():"`
                       `+request.getMethod()+"</p>");`

Output: request.getMethod(): GET

Q.10 **Write servlet which displayed following information of client.**
I. Client Browser
II. Client IP address
III. Client Port No
IV. Server Port No
V. Local Port No
VI. Method used by client for form submission
VII. Query String name and values

Ans.

```
1.   import java.io.*;
2.   import javax.servlet.http.*;
3.   public class ServletInfo extends HttpServlet{
4.   PrintWriter out;
5.   public void doGet(HttpServletRequest req,HttpServletResponse
     res)                                         throws
     IOException
6.   {
7.   res.setContentType("text/html");
8.   out=res.getWriter();
9.   // I. Client Browser: we use String getHeader(user-agent)
10.  out.println("<p> Client Browser=" +req.getHeader
.    ("user-agent")+"</p>");
11.  //II. Client IP address
12.  out.println("<p> Client IP address= "+req.getRemoteAddr());
13.  //III. Client Port No
14.  out.println("<p> Client Port No= "+req.getRemotePort());
15.  //IV. Server Port No
16.  out.println("<p> Server Port No= "+req.getServerPort());
17.  //V. Local Port No
18.  out.println("<p> Local Port No= "+req.getLocalPort());
19.  //VI. Method used by client for form submission
20.  out.println("<p> Method used by client= "+req.getMethod());
21.  //VII. Query String name and values
22.  out.println("<p> Query String name & values=
                                    "+req.getQueryString());
23.  }}
```

Output:
Client Browser=Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:58.0) Gecko/20100101 Firefox/58.0
Client IP address= 0:0:0:0:0:0:0:1
Client Port No= 64779
Server Port No= 8080
Local Port No= 8080
Method used by client= GET
Query String name & values= null

Q11. What is Request Dispatcher? What is the difference between Request dispatcher's forward () and include () method?

Ans. **javax.servlet.RequestDispatcher Interface**

- The RequestDispatcher interface provides the facility of dispatching the request to another resource.
- Resource can be HTML, Servlet or JSP.
- This interface can also be used to include the content of another resource.
- It is one of the way of servlet collaboration.
- The **getRequestDispatcher**() method of ServletRequest interface returns the object of RequestDispatcher.

Syntax
```
RequestDispatcher getRequestDispatcher(String resource)
```
Example
```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
rd.forward(request, response);//method may be include/forward
```

There are two methods defined in the **RequestDispatcher interface**

void **forward**(ServletRequest request, ServletResponse response) throws ServletException, IOException	Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
void **include**(ServletRequest request, ServletResponse response) throws ServletException, IOException	Includes the content of a resource (Servlet, JSP page, or HTML file) in the response.

RequestDispatcher: forward()
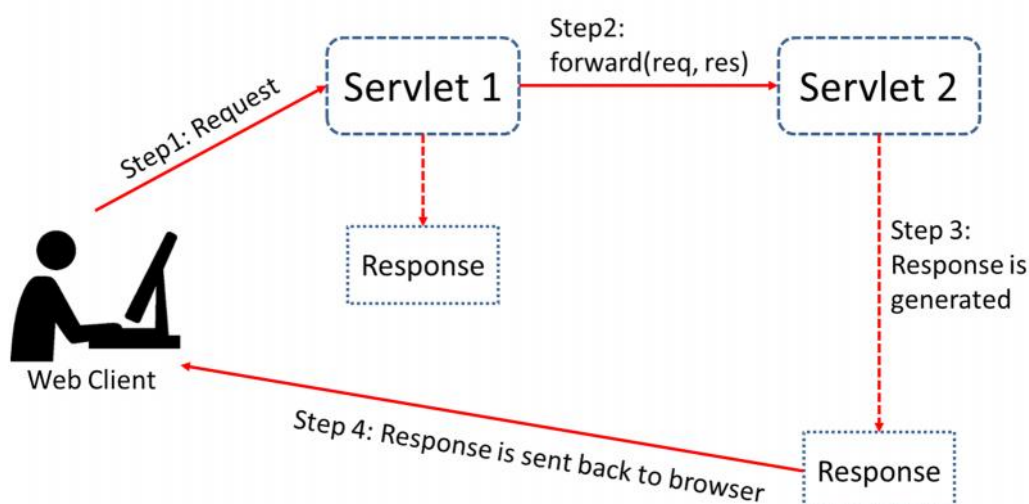
Figure: Working of RequestDispatcher.forward()

Example: forward()
//for java servlet
```
RequestDispatcher rd =  request.getRequestDispatcher("servlet2");
rd.forward(request, response);
```
//for html page
```
RequestDispatcher rd= request.getRequestDispatcher("/1.html");
rd.forward(request, response);
```
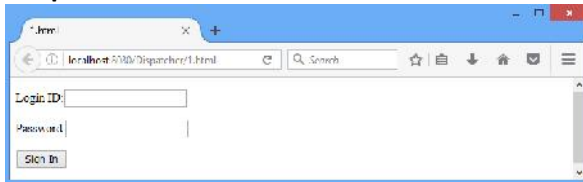
RequestDispatcher: include()
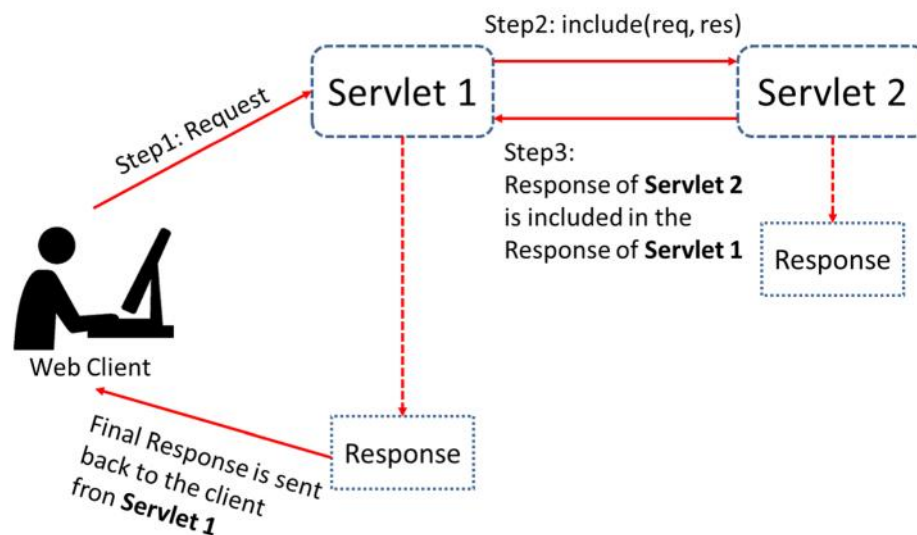
Figure: Working of RequestDispatcher.include()

Example: include()
//for java servlet
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
rd.include(request, response);
//for html page
RequestDispatcher rd=request.getRequestDispatcher("/1.html");
rd.include(request, response);

Q12. **Write a Servlet program to authenticate user with user_id and password, if user is authenticated then forward to welcome page else include message with invalid user_id/password.**

Ans. **1.html**

```
1. <html>
2.     <head>
3.         <title>1.html</title>
4.     </head>
5.     <body>
6.         <form action="/Dispatcher/CallServlet" method="POST">
7.             <p>Login ID:<input type="text" name="login"></p>
8.             <p>Password:<input type="text" name="pwd"></p>
9.             <p><input type="submit" value="Sign In"></p>
10.          </form>
11.       </body>
12.    </html>
```

Output:

CallServlet.java

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class CallServlet extends HttpServlet
5. {   public void doPost(HttpServletRequest request,
                          HttpServletResponse response)
                             throws ServletException,IOException
6.     {   response.setContentType("text/html");
7.         PrintWriter out=response.getWriter();
8.         RequestDispatcher rd;
9.         String login=request.getParameter("login");
10.        String pwd=request.getParameter("pwd");
11.        if(login.equals("java") && pwd.equals("servlet"))
12.            {   rd=request.getRequestDispatcher("FwdDemo");
13.                rd.forward(request, response);
14.        }//if
15.        else
16.        {   out.println("<p><h1>Incorrect Login Id/Password
                                        </h1></p>");
17.                rd=request.getRequestDispatcher("/1.html");
18.                rd.include(request, response);   }//else
19.    }//dopost }
```

FwdDemo.java

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class FwdDemo extends HttpServlet{
5. public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
6.                       throws ServletException,IOException
7.     {    response.setContentType("text/html");
8.          PrintWriter out=response.getWriter();
9.          String username=request.getParameter("login");
10.          out.println("<h1>"+"Welcome "+username+"</h1>");
11.          }
12.     }
```

Output:forward()
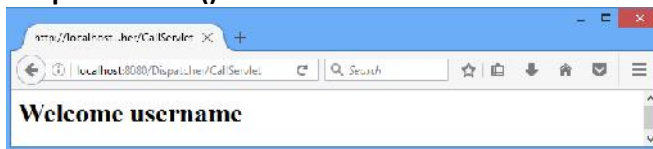
Output:include()

Q13. Explain response.sendRedirect() with appropriate example.

Ans.   The **sendRedirect**() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

Syntax
```
void sendRedirect(String location) throws IOException
```
Example
```
response.sendRedirect("http://www.darshan.ac.in");
response.sendRedirect("/1.html");//relative path
response.sendRedirect("http://localhost:8080/1.html");
```

```
                                        //absolute path
```

Program: sendRedirect()

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class Redirect extends HttpServlet
5. {    public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
6.                       throws ServletException,IOException
7.       {    response.setContentType("text/html");
8.            PrintWriter out=response.getWriter();
9.            String login=request.getParameter("login");
10.              String pwd=request.getParameter("pwd");
11.              if(login.equals("java") && pwd.equals("servlet"))
12.              {   response.sendRedirect("/Dispatcher/Welcome");
13.              }
14.            else
15.            response.sendRedirect("/Dispatcher/redirect.html");
16.          } //doGet
17.      }
```

Q14. Differentiate forward() and sendRedirect()

Ans.

forward()	sendRedirect()
forward() is method of RequestDispatcher interface	sendRedirect() is the method of HttpServletResponse
In forward(), redirect happens at server end and not visible to client.	In sendRedirect() ,redirection happens at client end and it's visible to client.
It is faster than the redirect.	It is slower than a forward, since it requires two browser requests(one for actual request and another for redirected request).
In case of forward() original URL remains unaffected.	While in case of sendRedirect() browser knows that it's making a new request, so original URL changes.
The request is transfer to other resource within same server. Thus, it can be used within server.	The request may transfer to other resource to different server. Thus, it can be used within and outside the server.
When forward is called on RequestDispather object we pass request and response object so our old request object is present on new resource which is going to process our request.	In case of SendRedirect call old request and response object is lost because it's treated as new request by the browser.
Syntax: forward(ServletRequest request, ServletResponse response)	**Syntax:** void sendRedirect(String url)

Q15. What is Session? Why we require Session? List the different ways to manage the session.

Ans. Define Session

"A session refers to the entire interaction between a client and a server from the time of the client's first request, which generally begins the session, to the time of last request/response."

Why we require Session?

- HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
- Session is required to keep track of users and their information.

List the different ways to manage the session.

- Session Management is a mechanism used by the Web container to store session information for a particular user.
- There are four different techniques for session management.
    1. Hidden form field
    2. URL Rewriting
    3. Cookies
    4. HttpSession

Q16. Explain Hidden form field with appropriate example.

Ans.
- Hidden Form Field, a hidden (invisible) textfield is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet.
  E.g. `<input type="hidden" name="session_id"  value="054">`

Real application of hidden form field

- It is widely used in comment form of a website.
- In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

- Easy to implement
- It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.
- It does not support hyperlink submission.
- Security

- Hidden field will be visible with GET method
- User might view page source and can view hidden field

Program: Hidden form field
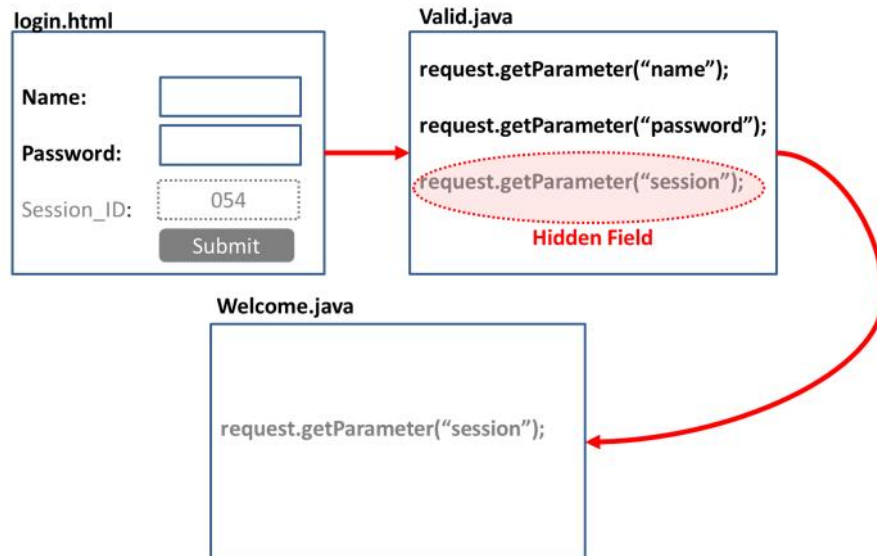
login.html

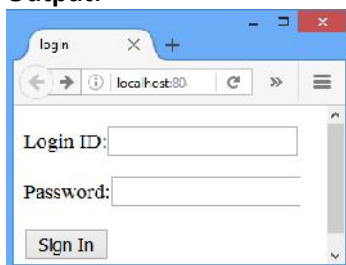```
1. <html>
2.   <head>
3.   <title>login</title>
4.   </head>
5. <body>
6. <form action="/Session/Valid" method="POST">
7.   <p>Login ID:<input type="text" name="login"></p>
8.   <p>Password:<input type="text" name="pwd"></p>
9.   <p><input type="hidden" name="session_id"
                  value="054"></p>
10.   <p><input type="submit" value="Sign In"></p>
11.   </form>
12.   </body>
13.   </html>
```

**Output:**

**Valid.java**

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class Valid extends HttpServlet
5. {    public void doPost(HttpServletRequest request,
                         HttpServletResponse response)
6.                      throws ServletException,IOException
7.    {
8.       response.setContentType("text/html");
9.       PrintWriter out=response.getWriter();
10.          RequestDispatcher rd;
11.          String login=request.getParameter("login");
12.          String pwd=request.getParameter("pwd");
13.       String session=request.getParameter("session_id");
14.       if(login.equals("java") && pwd.equals("servlet"))
15.       {
16.          rd=request.getRequestDispatcher("Welcome");
17.          rd.forward(request, response);
18.       }//if
19.       else
20.       {
21.          out.println("<p><h1>Incorrect LoginId/Password
                                     </h1></p>");
22.          rd=request.getRequestDispatcher("/login.html");
23.          rd.include(request, response);
24.       }//else
25.    } }
```

Welcome.java

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class Welcome extends HttpServlet
5. {    public void doPost(HttpServletRequest request,
                         HttpServletResponse response)
6.                    throws ServletException,IOException
7.    {   response.setContentType("text/html");
8.        PrintWriter out=response.getWriter();
9.        String session=request.getParameter("**session_id**");
10.          String username=request.getParameter("login");
11.          out.println("<h1>"+"id:"+session+"</h1>");
12.          out.println("<h3>"+"Welcome "+username+"</h3>");
13.       }
14.    }
```

Q17. Explain URL Rewriting with appropriate example.

Ans.
- In URL rewriting, a token or identifier is appended to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:

    URL ? Name1 = value1 & name2 = value2 &…

- Here, A name and a value is separated using an equal (=) sign and name/value pair is separated from another parameter using the ampersand(&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a Servlet, we can use **getParameter()** method to obtain a parameter value.

Advantage of URL Rewriting
- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting
- It will work only with links.
- It can send only textual information.
- URL header size constraint.
- Security
    - name/value field will be visible with URL followed by '?'.

Program: URL Rewriting
Url1.java
```
1.    import javax.servlet.*;
2.    import javax.servlet.http.*;
3.    import java.io.*;
4.    public class Url1 extends HttpServlet
5.    {   public void doGet(HttpServletRequest request,
                            HttpServletResponse response)
6.                          throws ServletException,IOException
7.    {  String url;
8.       response.setContentType("text/html");
9.       PrintWriter out=response.getWriter();
10.     //for URL rewriting
11.     url= "http://localhost:8080/Session
                    /Url2?s_id1=054&s_id2=055";
12.     out.println("<a href="+url+">next page</a>");
13.   } }
```

Output:
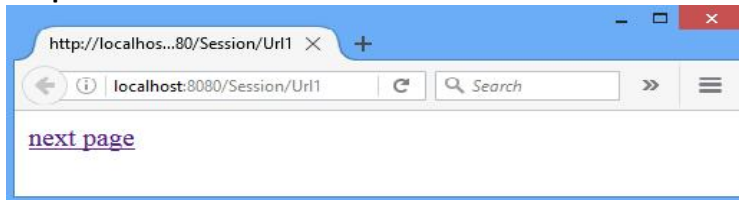
Url2.java

```
1.    import javax.servlet.*;
2.    import javax.servlet.http.*;
3.    import java.io.*;
4.    public class Url2 extends HttpServlet
5.    {   public void doGet(HttpServletRequest request,
                            HttpServletResponse response)
6.                            throws ServletException,IOException
7.        {   response.setContentType("text/html");
8.            PrintWriter out=response.getWriter();
9.            String session1=request.getParameter("s_id1");
10.           String session2=request.getParameter("s_id2");
11.           out.println("<h3>"+"id:"+session1+"</h3>");
12.           out.println("<h3>"+"id:"+session2+"</h3>");
13.       }
14.   }
```

Output:
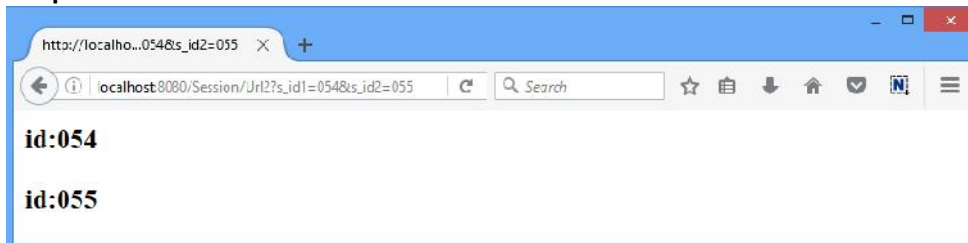
Q18. **What is Cookie? What does the cookie contains? Explain methods of Cookie class.**

Ans. **What is Cookie? What does the cookie contains?**
- A cookie is a small piece of information that is persisted between the multiple client requests.
- A cookie has a
    1. Name
    2. Single value
    3. Optional attributes such as
        i. comment
        ii. path
        iii. domain qualifiers
        iv. a maximum age
        v. Version number etc.

Types of Cookie
There are 2 types of cookies in servlets.
1. **Non-persistent cookie/Session cookie:** It is **valid for single session** only. It is removed each time when user closes the browser.
2. **Persistent cookie:** It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Cookie class(javax.servlet.http.Cookie)
- This class provides the functionality of using cookies.
- It provides a lots of useful methods for cookies.

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

Example
```
Cookie c = new Cookie("session_id","054");
                                    //creating cookie object
```

Methods of Cookie class

void setMaxAge(int expiry)	Sets the maximum age in seconds for this Cookie
int getMaxAge()	Gets the maximum age in seconds of this Cookie. By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.
String getName()	Returns the name of the cookie. The name cannot be changed after creation.
void setValue(String newValue)	Assigns a new value to this Cookie.
String getValue()	Gets the current value of this Cookie.

Other Methods of HttpServletRequest & HttpServletResponse	
void addCookie (Cookie cookieObj)	Method of HttpServletResponse interface is used to add cookie in response object.
Cookie[] getCookies()	Returns an array containing all of the Cookie objects the client sent with this request. This method returns null if no cookies were sent.

How to create Cookie?
```
//creating cookie object
     Cookie c= new Cookie("session_id","054");

//adding cookie in the response from server to client
     response.addCookie(c);
```

How to retrieve Cookies?
```
Cookie c[]=request.getCookies();
for(int i=0;i<c.length;i++)
{
     out.print(c[i].getName()+""+c[i].getValue());
          //printing name&value of cookie
}
```

How to delete Cookie?
1. Read an already existing cookie and store it in Cookie object.
2. Set cookie age as zero using setMaxAge() method to delete an existing cookie
3. Add this cookie back into response header.
```
//deleting value of cookie
   Cookie c = new Cookie("user","");
//changing the maximum age to 0 seconds
   c.setMaxAge(0);
//adding cookie in the response
   response.addCookie(c);
```
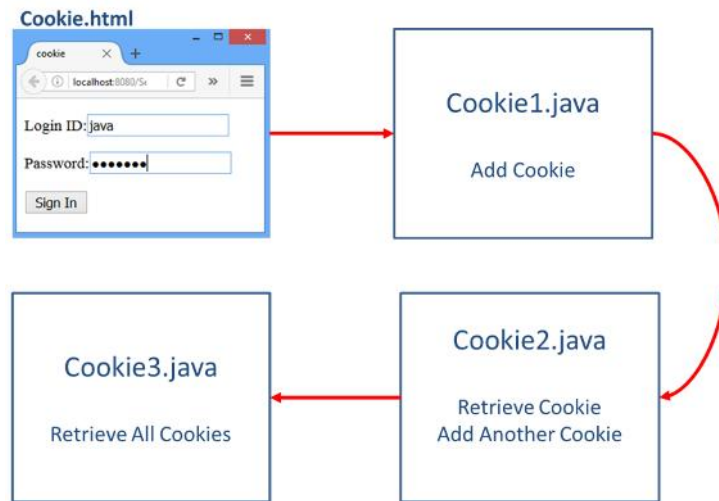
Advantage of Cookies
- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantage of Cookies
- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

Q19. Write a Servlet program for Session Management using Cookie.

Ans.

cookie.html

```
1. <html>
2. <head>
3. <title>cookie</title>
4. </head>
5. <body>
6. <form action="/Session/Cookie1" >
        <p>Login ID:<input type="text" name="login"></p>
7. <p>Password:<input type="password" name="pwd"></p>
8. <p><input type="submit" value="Sign In"></p>
9. </form>
10.     </body>
11.     </html>
```
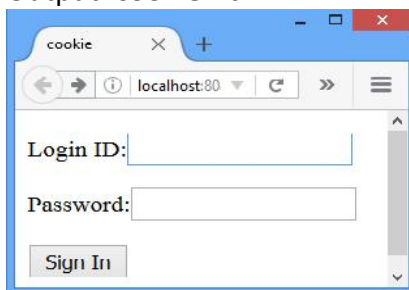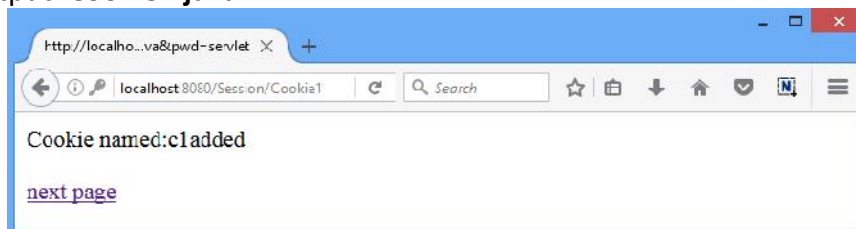
Output: **cookie.html**

Cookie1.java

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class Cookie1 extends HttpServlet
5. {   public void doGet(HttpServletRequest request,
                HttpServletResponse response)
6.                            throws ServletException,IOException
7.     {   response.setContentType("text/html");
8.         PrintWriter out=response.getWriter();
9.         String login=request.getParameter("login");
10.        String pwd=request.getParameter("pwd");
11.
12.        if(login.equals("java") && pwd.equals("servlet"))
13.        {
14.           Cookie c = new Cookie("c1",login);//create cookie
15.           response.addCookie(c);//adds cookie with response
16.
17.           out.println("Cookie named:"+c.getName()+" added");
18.           String path="/Session/Cookie2";
19.           out.println("<p><a href="+path+">next page</a></p>");
20.        }
21.      else
22.     {
23.     out.println("<p><h1>Incorrect Login Id/Password </h1></p>");
24.               rd=request.getRequestDispatcher("/cookie.html");
25.               rd.include(request, response);}
26.     } }
```
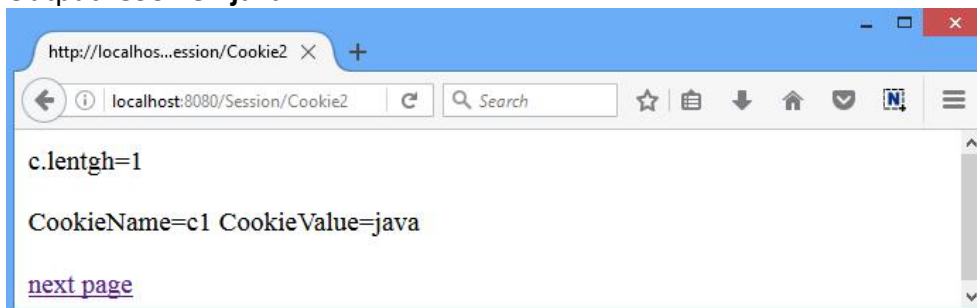
Output: **Cookie1.java**

Cookie2.java

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4.
5. public class Cookie2 extends HttpServlet
6. {   public void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws
                         ServletException,IOException
7.      {
8.        response.setContentType("text/html");
9.         PrintWriter out=response.getWriter();
10.        Cookie c[]=request.getCookies();
11.        out.println("c.length="+c.length);
12.        for(int i=0;i<c.length;i++)
13.        {    out.println("CookieName="+c[i].getName()+
14.                    "CookieValue="+c[i].getValue());
15.        }
16.         //to add another cookie
17.            Cookie c1 = new Cookie("c2","054");
18.            response.addCookie(c1);
19.            String path="/Session/Cookie3";
20.            out.println("<a href="+path+">next page</a>");
21.  }}
```

Output: **Cookie2.java**

Cookie3.java

```
1. import javax.servlet.*;
2. import javax.servlet.http.*;
3. import java.io.*;
4. public class Cookie3 extends HttpServlet
5. {   public void doGet(HttpServletRequest request,
                HttpServletResponse response)
6.                      throws ServletException,IOException
7.      {  response.setContentType("text/html");
8.         PrintWriter out=response.getWriter();
9.         Cookie c[]=request.getCookies();
10.            for(int i=0;i<c.length;i++)
```

```
11.                    {    out.println("<p>");
12.                         out.println("CookieName="+c[i].getName()+
13.                                  "CookieValue="+c[i].getValue());
14.                         out.println("</p>");
15.          }
16.        }
17.     }
```

Output: **Cookie3.java**

CookieName=c1 CookieValue=java

CookieName=c2 CookieValue=054
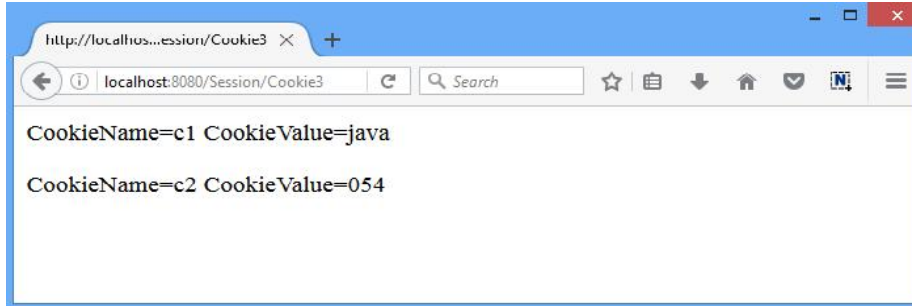
Q20. Explain Session Management using HttpSession.

Ans.
- HttpSession Interface(javax.servlet.http.**HttpSession**) provides a way to identify a user across more than one page request
- The 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.
- **HttpSession** object is used to store entire session with a specific client.
- We can store, retrieve and remove attribute from **HttpSession** object.
- Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- In this technique create a session object at server side for each client.
- Session is available until the session time out, until the client log out.
- The default session time is 30 minutes and can configure explicit session time in web.xml file.

Working of HttpSession
1. On client's first request, the Web Container generates a unique **session ID** and gives it back to the client with response.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.
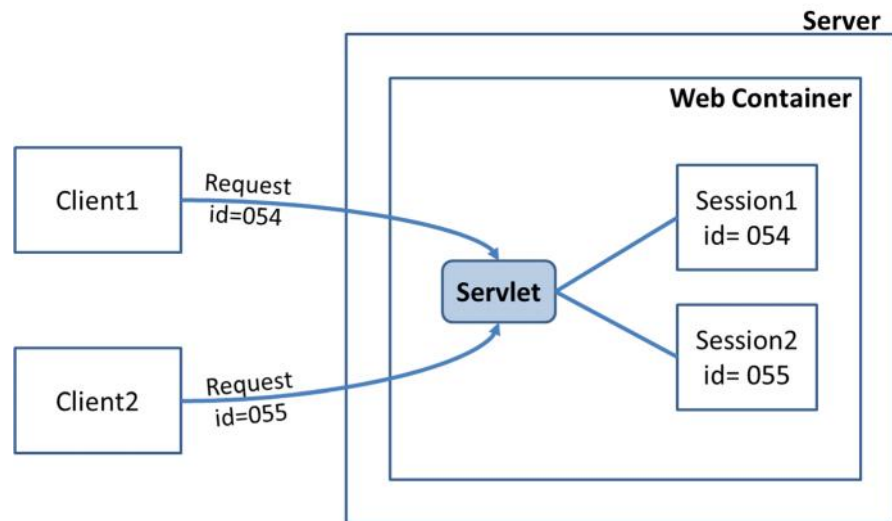
Figure: Working of HttpSession

Methods of HttpServletRequest interface:

HttpSession getSession()	Returns the current session associated with this request, or if the request does not have a session, creates one.
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.
String getId()	Returns a string containing the unique identifier value.
long getCreationTime()	Returns the time when this session was created, measured in milliseconds.
long getLastAccessedTime()	Returns the last time the client sent a request associated with this session, as the number of milliseconds.
void invalidate()	Invalidates this session then unbinds any objects bound to it.

How to create the Session?
```
HttpSession hs=request.getSession();
hs.setAttribute("s_id", "diet054");
```

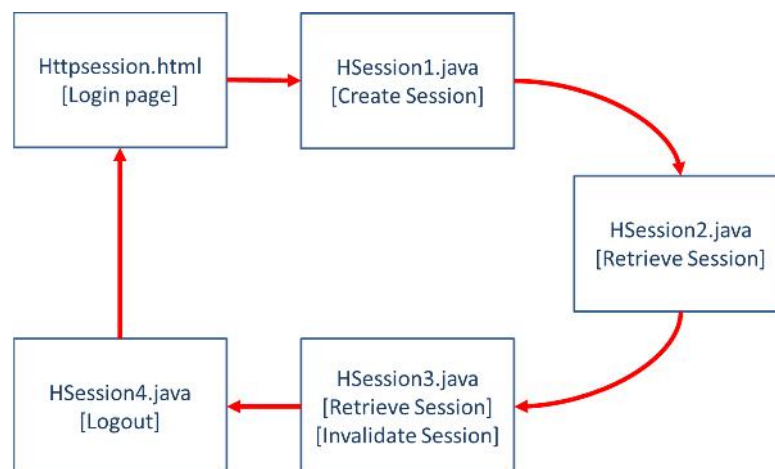**How to retrieve a Session?**
```
HttpSession hs=request.getSession(false);
String n=(String)hs.getAttribute("s_id");
```

How to invalidate a Session?
```
hs.invalidate();
```

Q21. Write a servlet program for Session Management using HttpSession.
Ans.

Httpsession.html
```
1. <html>
2. <head>
3. <title>HttpSession</title>
4. </head>
5. <body>
6. <form action="/Session/HSession1" method="Get">
7.    <p>Login ID:<input type="text" name="login"></p>
8.    <p><input type="submit" value="Sign In"></p>
9. </form>
10.      </body>
11.      </html>
```
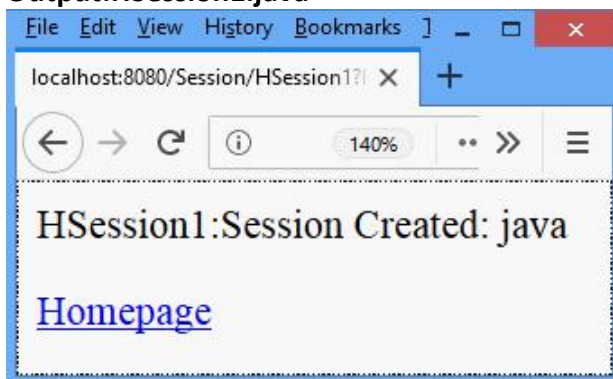**Output:Httpsession.html**

HSession1.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class HSession1 extends HttpServlet
5. {    public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                                throws ServletException,IOException
6. {      response.setContentType("text/html");
7.        PrintWriter out=response.getWriter();
8.        RequestDispatcher rd;
9.        String login=request.getParameter("login");
10.       if(login.equals("java") )
11.       {      HttpSession hs=request.getSession();
12.              hs.setAttribute("s_id",login);
13.              out.println("<p> HSession1:Session Created:
                             "+hs.getAttribute("s_id")+"</p>");
14.              out.print("<a href='HSession2'>Homepage</a>");
15.       }
16.       else
17.       {    out.println("<p><h1>Incorrect Login Id/Password
                                           </h1></p>");
18.            rd=request.getRequestDispatcher("/httpsession.html");
19.            rd.include(request, response);
20.       }
21.    } }
```
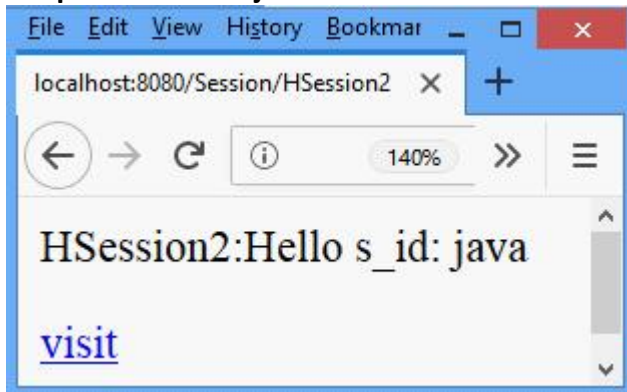
Output:HSession1.java

HSession2.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class HSession2 extends HttpServlet
5. {    public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                                 throws ServletException,IOException
6. {      response.setContentType("text/html");
7.        PrintWriter out=response.getWriter();
8.        HttpSession hs=request.getSession(false);
9.        String n=(String)hs.getAttribute("s_id");
10.       out.print("HSession2:Hello s_id: "+n);
11.       out.print("<p><a href='HSession3'>visit</a></p>");
12.       } }
```
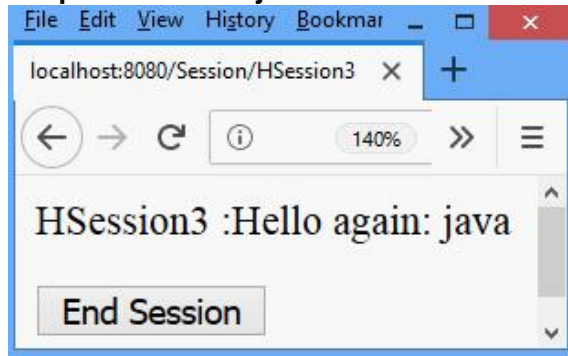
Output:HSession2.java

HSession3.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class HSession3 extends HttpServlet
5. {    public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                                 throws ServletException,IOException
6. {      response.setContentType("text/html");
7.        PrintWriter out=response.getWriter();
8.        HttpSession hs=request.getSession(false);
9.        String n=(String)hs.getAttribute("s_id");
10.       out.print("HSession3 :Hello again: "+n);
11.       out.println("<p><form action='/Session/HSession4'></p>");
12.       out.println("<p><input type='submit' value='End
                                      Session'></p></form>");
13.       hs.invalidate();//Session Invalidated
14.       }}
```

Output:HSession3.java

HSession4.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class HSession4 extends HttpServlet
5. {    public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                                throws ServletException,IOException
6. {      response.setContentType("text/html");
7.        PrintWriter out=response.getWriter();
8.        HttpSession hs=request.getSession(false);
9.        try{
10.            String n=(String)hs.getAttribute("s_id");
11.       }
12.       catch(NullPointerException ne)
13.       {out.println("<p>HSession4:Session Invalidated
              </p>"+ne.toString());}
14.       out.println("<form action='/Session/httpsession.html'>");
15.       out.println("<p><input type='submit'
                                    value='logout'></p></form>");
16.       }
17. }
```

Output:HSession4.java
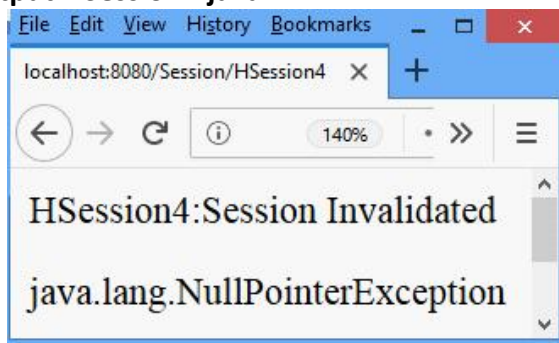
Q22. **What is filter? What is its use? List different filter interfaces with their important methods. List the applications of filter.**

Ans. **What is filter?**

A filter is an object that is invoked at the preprocessing and post processing of a request. Java Servlet Filter is used to intercept request and do some pre-processing and can be used to intercept response and do post-processing before sending to client in web application.

What is its use?

- Recording all incoming requests
- Logs the IP addresses of the computers from which the requests originate
- Conversion
- Data compression
- Encryption and Decryption
- Input validation etc.

Filter interfaces

The javax.servlet package contains the three interfaces of Filter API.

1. **Filter**
   - For creating any filter, you must implement the Filter interface.
   - Filter interface provides the life cycle methods for a filter.

   Method of Filter Interface

void **init**(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
void **doFilter** (HttpServletRequest reuest, HttpServletResponse response, FilterChain chain)	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks.
void **destroy**()	This is invoked only once when filter is taken out of the service.

2. **FilterChain**
   - The object of FilterChain is responsible to invoke the next filter or resource in the chain.
   - This object is passed in the doFilter method of Filter interface.
   - The FilterChain interface contains only one method:

void **doFilter** (HttpServletRequest request, HttpServletResponse response)	It passes the control to the next filter or resource.

3. **FilterConfig**
- FilterConfig is created by the web container.
- This object can be used to get the configuration information from the web.xml file.
  Method

void init(FilterConfig config)	init() method is invoked only once it is used to initialize the filter.
String getInitParameter (String parameterName)	Returns the parameter value for the specified parameter name.

List the applications of filter
1. Authentication-Blocking requests based on user identity.
2. Logging and auditing-Tracking users of a web application.
3. Image conversion-Scaling maps, and so on.
4. Data compression-Making downloads smaller.
5. Localization-Targeting the request and response to a particular locale.

Advantage of Filter
- Filter is pluggable.
- One filter don't have dependency onto another resource.
- Less Maintenance Cost

The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.
So maintenance cost will be less.

Q23. Write a Servlet Program that uses two filters
     i)   Authentication Filter: Checks authentication value (userid/password)
     ii)   Config Param Filter: Checks value of config param in web.xml

Ans.
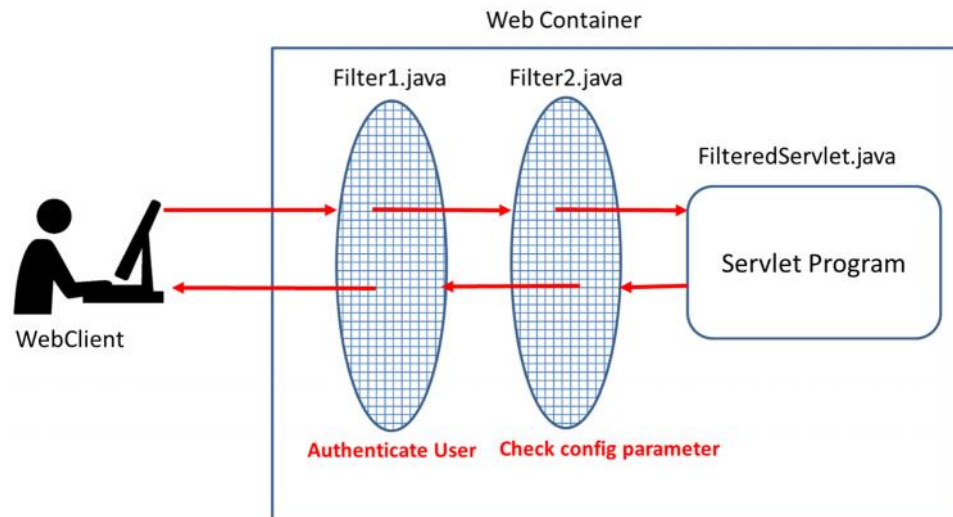
Web.xml

```
1. <web-app>
2. <servlet>
3.    <servlet-name>FilteredServlet</servlet-name>
4.    <servlet-class>FilteredServlet</servlet-class>
5. </servlet>
6. <servlet-mapping>
7.    <servlet-name>FilteredServlet</servlet-name>
8.    <url-pattern>/FilteredServlet</url-pattern>
9. </servlet-mapping>

10.     <filter>
11.             <filter-name>f1</filter-name>
12.             <filter-class>Filter1</filter-class>
13.     </filter>
14.     <filter-mapping>
15.             <filter-name>f1</filter-name>
16.             <url-pattern>/FilteredServlet</url-pattern>
17.     </filter-mapping>
```

```
18.      <filter>
19.              <filter-name>f2</filter-name>
20.              <filter-class>Filter2</filter-class>
21.              <init-param>
22.                  <param-name>permit</param-name>
23.                  <param-value>yes</param-value>
24.              </init-param>
25.      </filter>
26.      <filter-mapping>
27.              <filter-name>f2</filter-name>
28.              <url-pattern>/FilteredServlet</url-pattern>
29.        </filter-mapping>
30.      </web-app>
```

index.html

```
1. <html>
2.     <head>
3.         <title>filter</title>
4.     </head>
5.   <body>
6.    <form action="/Filter/FilteredServlet" >
7.        <p>Login ID:<input type="text"      name="login"></p>
8.        <p>Password:<input type="password" name="pwd"></p>
9.        <p><input type="submit" value="Sign In"></p>
10.  </form>
11.  </body>
12. </html>
```

Filter1.java

```
31.      import java.io.IOException;
32.      import java.io.PrintWriter;
33.      import javax.servlet.*;
34.      public class Filter1 implements Filter{
35.      public void init(FilterConfig config) {}
36.      public void doFilter(ServletRequest req,
37.                          ServletResponse resp,
38.                          FilterChain chain)
39.                          throws IOException, ServletException
40.      {
41.         PrintWriter out=resp.getWriter();
42.         out.print("<p>filter1 is invoked before</p>");
43.         if(req.getParameter("login").equals("java") &&
                       req.getParameter("pwd").equals("servlet"))
     {
44.      chain.doFilter(req, resp);//send request to next resource
     }//if
```

```
45.      else
46.        {out.print("<p>invalid login/password</p>");}//else
47.
48.        out.print("<p>filter1 is invoked after</p>");
49.      }
50.    public void destroy() {}}
```

Filter2.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3. import javax.servlet.*;
4. public class Filter2 implements Filter{
5. String permission;
6. public void init(FilterConfig config) throws ServletException
7. {           permission=config.getInitParameter("permit");
   }
8. public void doFilter(ServletRequest req, ServletResponse resp,
                                          FilterChain chain)
                        throws IOException, ServletException
9. {         PrintWriter out=resp.getWriter();
10.             out.print("<p>filter2 is invoked before</p>");
11.             if(permission.equals("yes"))
12.           { chain.doFilter(req, resp);}//if
13.         else
14.         { out.println("Permission Denied"); }//else
15.             out.print("<p>filter2 is invoked after</p>");
            }
16.         public void destroy() {}}
```

FilteredServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3. import javax.servlet.*;
4. public class FilteredServlet extends HttpServlet
5. {
6.  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
7.             throws ServletException, IOException
8. {
9.        response.setContentType("text/html");
10.       PrintWriter out = response.getWriter();
11.       out.println("<p><h3>welcome to servlet</h3></p>");
12.         }
13.     }
```

Output:
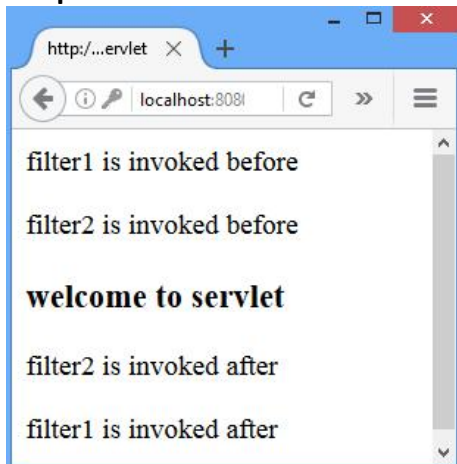
Q23. Explain Session Timeout

Ans. The session timeout in a web application can be configured in two ways
1. Timeout in the deployment descriptor (web.xml)
2. Timeout with setMaxInactiveInterval()

Timeout in the deployment descriptor (web.xml)
```
<web-app>
    <session-config>
        <session-timeout> 10 </session-timeout>
    </session-config>
</web-app>
```
Note that the value of the timeout is set in minutes, not in seconds.

Timeout with setMaxInactiveInterval()
The timeout of **the current session only** can be specified programmatically via the API of
the *javax.servlet.http.HttpSession*
```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(10*60);//time specified in seconds
```

Q24. State and explain Types of Servlet Events

Ans.
- Events are basically occurrence of something.
- Changing the state of an object is known as an event.
- There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.
- In web application world an event can be
    1. Initialization of application
    2. Destroying an application
    3. Request from client
    4. Creating/destroying a session
    5. Attribute modification in session etc.

Event classes

ServletRequestEvent	Events of this kind indicate lifecycle events for a ServletRequest. The source of the event is the ServletContext of this web application.
ServletContextEvent	This is the event class for notifications about changes to the servlet context of a web application.
ServletRequestAttributeEvent	This is the event class for notifications of changes to the attributes of the servlet request in an application.
ServletContextAttributeEvent	Event class for notifications about changes to the attributes of the ServletContext of a web application.
HttpSessionEvent	This is the class representing event notifications for changes to sessions within a web application.
HttpSessionBindingEvent	Send to an Object that implements HttpSessionBindingListener when bound into a session or unbound from a session.

Q25. Write a Servlet program which retrieves record from database

Ans.

```
1. import java.io.*;
2. import java.sql.*;
3. import javax.servlet.*;
4. import javax.servlet.http.*;
5. public class JDBCServlet extends HttpServlet
6. {
7. public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
8.                     throws ServletException,IOException
9.    {   response.setContentType("text/html");
10.       PrintWriter out=response.getWriter();
        try{
11.       Class.forName("com.mysql.jdbc.Driver");
12.       Connection con=DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/ajava","root","DIET");
13.       Statement st=con.createStatement();
14.       ResultSet rs=st.executeQuery("select * from cxcy");
15.       while(rs.next())
16.       { out.println("<p>"+rs.getInt(1));
17.         out.println(rs.getString(2));
18.         out.println(rs.getString(3)+"</p>");
19.        }
20.      }catch(Exception e)
21.      {out.println("<p>inside exception"+e.toString()+"</p>");}
```

```
22.       }//doGet()
23.       }//Class
```

Servlet Interview Questions

Q1. Servlet is Java class. Then why there is no constructor in Servlet? Justify your answer.

Ans.
- Servlet object has a well-defined life cycle where creation and initialization steps are distinct.
- Servlets are not directly instantiated by Java code, instead container create there instance and keep them in pool.
- Servlet implementation classes can have constructor but they should be using init() method to initialize Servlet because of two reasons, first you cannot declare constructors on interface in Java, which means you cannot enforce this requirement to any class which implements Servlet interface and second, Servlet require ServletConfig object for initialization which is created by container

Q2. Can we write the constructor in Servlet?

Ans.
- Servlet is a java class, whose object will be created by servlet container, and then container will call 3 life cycle method on that object.
- Container creates our servlet class object using 0-args constructor, If u want to place your own constructor you can, but make sure 0-args constructor is always present.
- However, our parameterized constructor will not be used by servlet container, and will be of no use.
- Also If u declare your own 0-args constructor, make sure it is public, so that it could be easily accessible to container

Q3. What happens if you add a main method to servlet?

Ans.
- We can place but it never executes automatically, because it is not a life cycle method.
- We can see main() method acting as helper method(normal method) to life cycle methods. We can call this main() from any lifecycle method explicitly as a helper method.
- Main method is for standalone apps,not for servlets which are executed by life cycle methods.

Q4. Consider a scenario in which 4 users are accessing a servlet instance. Among which one user called destroy() method. What happens to the rest 3 users?

Ans. We know that by default servlet is multithreaded, for every client request a new thread will be created and assigned to that to perform the service. so if one thread initiates destroy() only itself will be terminated but other threads not terminates.

Q5.   How does the JVM execute a servlet compared with a regular Java class?

Ans.
- Servlets are standard Java classes and are executed by the Java Virtual Machine in exactly the same way as any other. However, the environment or context in which Servlets are executed is different. A Servlet is not invoked directly through a main method, the class is loaded and run by a Servlet Container.
- A servlet is not invoked directly using a main() method like any other class.
- The servlet class is invoked and executed by a web container (Like Apache Tomcat).
- The container reads the configuration (like web.xml), identifies the servlet class, and uses java class loader system to load and run the servlets.

Q6.   How to get the fully qualified name of the client that sent the request in servlet?

Ans.   A servlet can use getRemoteAddr() and getRemoteHost() to retrieve the client's IP Address and client's host name from a http requisition.

Q7.   Why GenericServlet is an abstract class ?

Ans.
- GenericServlet class is abstract because there is a method called service() which is public abstract void. service() must be override, service() method defines what type of protocol is used for request.
- Another thing is that according to Java specification those classes have abstract methods must declared abstract.
- Abstract methods have no body only prototype.

Q8.   Who is responsible to create the object of servlet?

Ans.   The web container or servlet container.

Q9.   What is difference between Cookies and HttpSession?

Ans.   Cookie works at client side whereas HttpSession works at server side.

Q10.   Can filtering be done in an ordered way? If so then how to achieve it?

Ans.   Yes. The order of filter-mapping elements in web.xml determines the order in which the web container applies the filter to the servlet. To reverse the order of the filter, you just need to reverse the filter-mapping elements in the web.xml file.

GTU Questions

1.	Write a Login servlet. Take input username and password from html file login.html and authenticate the user. Write the web.xml.[7]	Win'17
2.	List and Explain various stages of Servlet life cycle. Explain role of web container.[7]	Win'17 Sum'16
3.	What is Session? Explain various Session tracking mechanisms in servlet with example.[7]	Win'17 Sum'16 Win'16
4.	What is filter? What is its use? List different filter interfaces with their important methods[7]	Win'17 Win'16
5.	Explain use of ServletConfig and ServletContext object with example.[4]	Win'17 Sum'17
6.	Discuss the use of GET and POST with example.[4]	Win'17 Sum'16
7.	Write a servlet RegistrationServlet to get the values from registration.html html page and display the contents. Write the web.xml file.[7]	Win'17
8.	Design a form to input details of an employee and submit the data to a servlet. Write code for servlet that will save the entered details as a new record in database table Employee with fields (EmpId, EName, Email, Age).[7]	Sum'17
9.	Explain Request and Response object in Servlet.[7]	Win'16
10.	Write servlet which displayed following information of client.[7] I. Client Browser II. Client IP address III. Client Port No IV. Server Port No V. Local Port No VI. Method used by client for form submission VII. Query String name and values	Sum'16
11.	Write small web application which takes marks of three subject and pass to servlet. Servlet forward to model class having method getClass() and getPercentage(). Display class and percentage in .jsp page.[7]	Sum'16
12.	i. GenericServlet vs HttpServlet ii. doGet() vs doPost() iii. Servlets vs CGI iv. Session and Cookie [7]	Sum'16

Q1. **List and Explain various stages of JSP life cycle. Briefly give the function of each phase. cleanliness**

Ans.
1. A JSP life cycle can be defined as the entire process from its creation till the destruction.
2. It is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.
3. A JSP page is converted into Servlet in order to service requests.
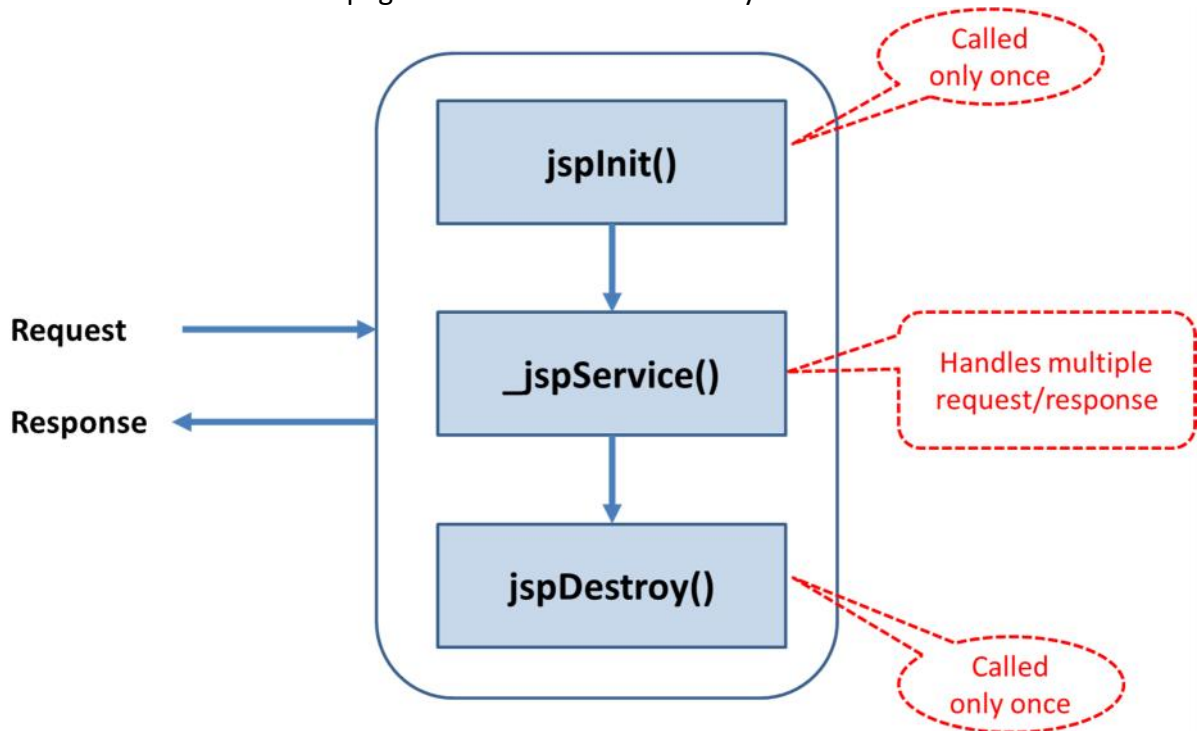4. The translation of a JSP page to a Servlet is called Lifecycle of JSP.

Figure: JSP Life Cycle

JSP Lifecycle Stages
1. **Translation of JSP to Servlet code**
   - Web Container translates JSP code into a servlet source(.java) file.
   - This is the first step in its tedious multiple phase life cycle.
   - In the translation phase, the container validates the syntactic correctness of the JSP pages and tag files.
   - The container interprets the standard directives and actions, and the custom actions referencing tag libraries used in the page.

2. **Compilation of Servlet to bytecode**
   - The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
   - The translation of a JSP source page into its implementation class can happen at any time between initial deployment of the JSP page into the JSP container and the receipt and processing of a client request for the target JSP page.

3. **Loading Servlet class**
   The java servlet class that was compiled from the JSP source is loaded into the container.

4. **Creating servlet instance**
   In this execution phase the container manages one or more instances of this class in response to requests and other events.

5. **Initialization by calling jspInit() method**
   - When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit().
   - Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

   ```
   public void jspInit()
   {
   //initializing the code
   }
   ```

6. **Request Processing by calling _jspService() method**
   - This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
   - Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.
   - The _jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters .

   ```
   void _jspService(HttpServletRequest request,
                    HttpServletResponse response) {
      // Service handling code...
   }
   ```

   - The _jspService() method of a JSP is invoked on request basis. This is responsible for generating the response for that request.

7. **Destroying by calling jspDestroy() method**
   - The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
   - The jspDestroy() method is the JSP equivalent of the destroy method for servlets.
   - Override jspDestroy(), when you need to perform any cleanup, such as releasing database connections or closing open files.

   ```
   public void jspDestroy() {
      // Your cleanup code goes here.
   }
   ```

   - When the call to destroy method is made then, the servlet is ready for a garbage collection
   - This is the end of the JSP life cycle.

Q2. **Compare JSP with Servlet. Also state advantages of JSP over Servlets.**
Ans.

JSP	Servlet
JSP is a webpage scripting language that generates dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
A JSP technically gets converted to a servlet We embed the java code into HTML. E.g. <html> <% java code %> </html>	A servlet is a java class. We can put HTML into print statements. E.g. out.println("<html code>");
JSPs are extension of servlets which minimizes the effort of developers to write User Interfaces using Java programming.	A servlet is a server-side program and written purely on Java.
JSP runs slower than servlet. As, it has the transition phase for converting from JSP to a Servlet. Once it is converted to a Servlet then it will start the compilation	Servlets run faster than JSP
In MVC architecture JSP acts as view.	In MVC architecture Servlet acts as controller.
We can build custom tags using JSP API	We cannot build any custom tags in servlet.

Advantages of JSP over Servlets
1. JSP needs no compilation. There is automatic deployment of a JSP, recompilation is done automatically when changes are made to JSP pages.
2. In a JSP page visual content and logic are separated, which is not possible in a servlet.
   i.e. JSP separates business logic from the presentation logic.
3. Servlets use *println* statements for printing an HTML document which is usually very difficult to use. JSP has no such tedious task to maintain.

Q3. **Explain JSP Scripting Elements with appropriate example.**
Ans. The scripting elements provides the ability to insert java code inside the jsp. There are three types of traditional scripting elements:
- scriptlet tag
- expression tag
- declaration tag

Scriptlet tag

- A scriptlet tag is used to execute java source code in JSP.
- A scriptlet can contain
    - Any number of JAVA language statements
    - Variable
    - Method declarations
    - Expressions that are valid in the page scripting language

Syntax

```
<%  // java source code   %>
```

Example

```
<% out.print("welcome to jsp"); %>
<% int a=10;     %>
```

- Everything written inside the scriptlet tag is compiled as java code.
- JSP code is translated to Servlet code, in which **_jspService()** method is executed which has HttpServletRequest and HttpServletResponse as argument.
- JSP page can have any number of scriptlets, and each scriptlets are appended in _jspService ().

Program: First.jsp

```
1. <html>
2. <body>
3. <% out.println("Hello World! My First JSP Page"); %>
4. </body>
5. </html>
```


Expression tag

- The code placed within **JSP expression tag** is *written to the output stream of the response*.
- So you need not write out.print() to write data.
- It is mainly used to print the values of variable or method.
- Do not end your statement with semicolon in case of expression tag.

Syntax

```
<%=statement %>
```

Example

```
<%=(2*5) %>
```

Declaration

- The **JSP declaration tag** is used *to declare variables and methods*
- The declaration of jsp declaration tag is placed outside the _jspService() method.

Syntax

```
<%!  variable or method declaration %>
```

Example

```
<%! int a = 10; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

Comments

- The comments can be used for documentation.
- This JSP comment tag tells the JSP container to ignore the comment part from compilation.

Syntax

```
<%-- comments --%>
```

JSP comment	<%-- jsp comment --%>
Java comment	/* java comment */ or // for single line
Html comment	<!-- html comment -->

Write a JSP program to demonstrate use of all three scripting elements

```
1. <html>
2. <body>
3. <%-- comment:JSP Scipting elements --%>
4. <%! int i=0; %> <%--declaration--%>
5. <% i++; %> <%--Sciptlet--%>
6. Welcome to world of JSP!
7. <%= "This page has been accessed " + i + " times" %><%--expression--%>
8. </body>
9. </html>
```

Output

Q4. Explain JSP Page Directives with appropriate example.

Ans.
- JSP directives provide directions and instructions to the container, telling it how to translate a JSP page into the corresponding servlet.
- A JSP directive affects the overall structure of the servlet class.
- JSP engine handles directives at Translation time.
- There are two types of directives:
  1. page directive
  2. include directive

Syntax
```
<%@ directive attribute="value" %>
```

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

Syntax
```
<%@page attribute="value" %>
```
Example
```
<%@page import="java.util.Date,java.util.List,java.io.*" %>
<%@page contentType="text/html; charset=US-ASCII" %>
```

Q5. Explain all the Attributes of Page Directive.

Ans. **Attributes of JSP page directive**

import	Used to import class, interface or all the members of a package
	<%@ page **import**="java.util.Date" %>
	Today is: <%= **new** Date() %>
contentType	The contentType attribute defines the MIME type of the HTTP response. The default value is "text/html;charset=ISO-8859-1".
	<%@ page **contentType**=application/msword %>
extends	The extends attribute defines the parent class that will be inherited by the generated servlet
	<%@ page **extends**="javax.servlet.HttpServlet" %>
info	This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() .
	<%@ page **info**="Authored by : AuthorName" %>
buffer	The buffer attribute sets the buffer size in kb to handle output generated by the JSP page.
	The default size of the buffer is 8Kb.
	<%@ page **buffer**="16kb" %>
language	The language attribute specifies the scripting language used in the JSP page. The default value is "java".
	<%@ page **language**="java" %>

isELIgnored	We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. EL is enabled by default. <%@ page **isELIgnored**="true" %>//Now EL will be ignored
autoFlush	The **autoFlush** attribute specifies whether buffered output should be flushed automatically when the buffer is filled.Bydefault it is true. <%@ page **autoFlush**="true" %>
isThreadSafe	This option marks a page as being thread-safe. By default, all JSPs are considered thread-safe(true). If you set the isThreadSafe = false, the JSP engine makes sure that only one thread at a time is executing your JSP. <%@ page **isThreadSafe** ="false" %>
session	The session attribute indicates whether or not the JSP page uses HTTP sessions. <%@ page **session**="true" %>//Bydefault it is true
pageEncoding	We can set response encoding type with this page directive attribute, its default value is "ISO-8859-1". <%@ page **pageEncoding** ="US-ASCII" %>
errorPage	It is used to define the error page, if exception occurs in the current page, it will be redirected to the error page. <%@ page **errorPage**="myerrorpage.jsp" %>
isErrorPage	The isErrorPage attribute is used to declare that the current page is the error page. <%@ page **isErrorPage**="true" %>

Q6. Explain JSP Include Directives with appropriate example.
Ans.
- JSP include directive is used to include the contents of another file to the current JSP page during translation time.
- Include directive is used for merging external files to the current JSP page during translation phase
- The included file can be HTML, JSP, text files etc.

Advantage of Include directive
Code Reusability
Syntax
```
<%@ include attribute= "value" %>
```
Example
```
<%@ include file="1.jsp" %>
```

Q7. Explain JSP implicit objects with appropriate example.

Ans.
- There are **9 jsp implicit objects**.
- These objects are *created by the web container* that are available to all the jsp pages.

Sr.No.	Implicit Object	Example

1.	out	• For writing any data to the buffer, JSP provides an implicit object named *out*.
		• It is an object of JspWriter
		`<html>`
		`<body>`
		`    <% out.print(“DIET”); %>`
		`</body>`
		`</html>`

2.	request	• Instance of *javax.servlet.http.HttpServletRequest* object associated with the request.
		• Each time a client requests a page the JSP engine creates a new object to represent that request.
		• The request object provides methods to get HTTP header information including from data, cookies, HTTP methods etc.
		`<%`
		`out.println(request.getParameter("login"));%>`

3.	response	• The response object is an instance of a javax.servlet.http.HttpServletResponse object.
		• Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, redirect response to another resource, send error etc.
		
		`<%response.sendRedirect("www.darshan.ac.in");`
		`%>`

4.	config	• Config is an implicit object of type *javax.servlet.ServletConfig.*
		• This object can be used to get initialization parameter for a particular JSP page.
		
		`<% out.print("Welcome "+`
		`request.getParameter("login"));`
		
		`String c_name=`
		`config.getInitParameter("College");`
		`out.print("<p>College name`
		`                is="+c_name+"</p>");%>`

5.	session	- In JSP, session is an implicit object of type **javax.servlet.http.HttpSession**. - The Java developer can use this object to set, get or remove attribute or to get session information.

6.	pageContext	- Instance of **javax.servlet.jsp.PageContext** - The pageContext object can be used to set, get or remove attribute. - The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, which identify the four scopes.
		`<% String name=`
		`(String)pageContext.getAttribute`
		`   ("user",PageContext.APPLICATION_SCOPE);`
		`out.print("Hello "+name);  %>`

7.	page	- This object is an actual reference to the instance of the page. - It is an instance of **java.lang.Object** - Direct synonym for the **this** object. ***Example: returns the name of generated servlet file*** `<%= page.getClass().getName() %>`

8.	application	- Instance of **javax.servlet.ServletContext** - The instance of ServletContext is created only once by the web container when application or project is deployed on the server. - This object can be used to get initialization parameter from configuration file (web.xml). - This initialization parameter can be used by all jsp pages.
		`<%//refers to context parameter of web.xml`
		`String driver=application.getInitParameter("`
		`name");`
		`out.print("name is="+name); %>`

9.	exception	- Exception is an implicit object of type **java.lang.Throwable** class. This object can be used to print the exception. - But it can only be used in error pages.
		`<%@ page isErrorPage="true" %>`
		
		`<html> <body>`
		
		`exception occured:  <%=exception %>`

| | | `</body>   </html>` |

Q8. Explain JSP Action elements with appropriate example.

Ans.
- JSP actions use constructs in XML syntax to control the behavior of the servlet engine.
- We can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

Syntax
```
<jsp:action_name attribute="value" />
```

There are four types of JSP Action elements

1. <jsp:param>
- This action is useful for passing the parameters to other JSP action tags such as JSP include & JSP forward tag.
- This way new JSP pages can have access to those parameters using request object itself.

Syntax
```
<jsp:param name ="name" value="value"  />
```
Example
```
<jsp:param name ="date" value="10-03-2017" />
<jsp:param name ="time" value="10:15AM" />
<jsp:param name ="data" value="ABC" />
```

2. <jsp:include>
- The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
- The jsp:include tag can be used to include static as well as dynamic pages

Attribute	Description
page	The relative URL of the page to be included.
flush	The boolean attribute determines whether the included resource has its buffer flushed before it is included. By default value is *false*.

Syntax
```
<jsp:include page="relative URL"  flush="true" />
```
Example
```
<jsp:include page="2.jsp" />
```

3. <jsp:forward>

Forwards the request and response to another resource.

Syntax

```
<jsp:forward page="Relative URL" />
```

Example

```
<jsp:forward page="2.jsp" />
```

4. <jsp:plugin>

- This tag is used when there is a need of a plugin to run a Bean class or an Applet.
- The <jsp:plugin> action tag is used to embed applet in the jsp file.
- The <jsp:plugin> action tag downloads plugin at client side to execute an applet or bean.

Syntax

```
<jsp:plugin type="applet|bean"
            code="nameOfClassFile"
            codebase= "URL"
/>
```

Example

MyApplet.java

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Welcome in Java Applet.",40,20);
}}
```

MyPlugin.jsp

```
<html> <body>
        <jsp:plugin
           type="applet"
           code="MyApplet.class"
           codebase="/JSPClass/MyApplet"/>
</body></html>
```

Q9. **What is EL Scripting? Explain EL implicit object and EL operator with appropriate example.**

Ans. **What is EL Scripting?**

- Expression Language(EL) Scripting.
- The Java **Expression Language** is a special purpose programming language mostly used in Java web applications for embedding expressions into web pages.
- It is the newly added feature in JSP technology version 2.0.
- **The purpose of EL is to produce script less JSP pages.**

Syntax `${expr}`

Example

EL	Output
${a=10}	10
${10+20}	30
${20*2}	40
${10==20}	false
${'a'<'b'}	true

EL Implicit Object

pageScope	It is used to access the value of any variable which is set in the Page scope
requestScope	It is used to access the value of any variable which is set in the Request scope.
sessionScope	It is used to access the value of any variable which is set in the Session scope
applicationScope	It is used to access the value of any variable which is set in the Application scope
pageContext	It represents the PageContext object.
param	Map a request parameter name to a single value
paramValues	Map a request parameter name to corresponding array of string values.
header	Map containing header names and single string values.
headerValues	Map containing header names to corresponding array of string values.
cookie	Map containing cookie names and single string values.

- An expression can be mixed with static text/values and can also be combined with other expressions

Example1

```
${param.name}
${sessionScope.user}
```

Example2

EL1.jsp

```
1. <form action="EL1.jsp">
2.  Enter Name:<input type="text" name="name" >
3.  <input type="submit" value="go">
4. </form>
```
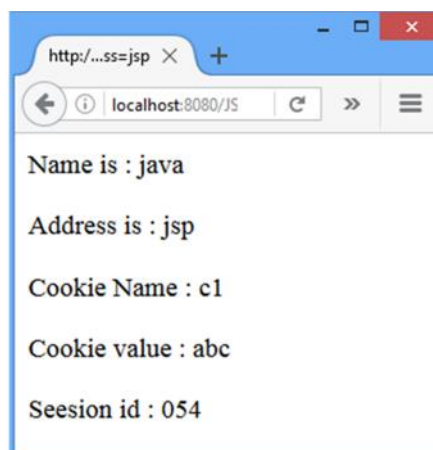
EL2.jsp

```
1. Welcome, ${ param.name }
```
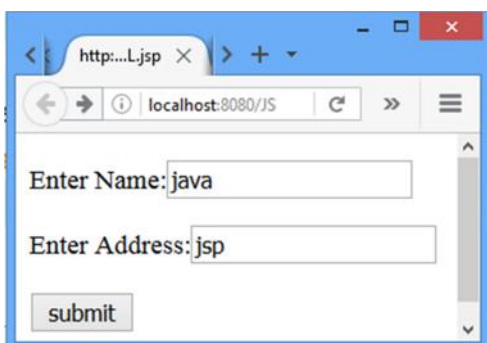
Example3:

Cookie_Session1.jsp

```
1. <form action="EL2.jsp">
2. <% Cookie ck=new Cookie("c1","abc");
3.    response.addCookie(ck);
4.    session.setAttribute("sid","054");  //for session
5. %>
6. Enter Name:<input type="text" name="name" >
7. Enter Address:<input type="text" name="address" >
8. <input type="submit" value="submit">
9. </form>
```

Cookie_Session2.jsp

```
1. <p>Name is :        ${param.name}</p>
2. <p>Address is :     ${param.address}</p>
3. <p>Cookie Name :    ${cookie.c1.name}</p>
4. <p>Cookie value :   ${cookie.c1.value}</p>
5. <p>Session id :     ${sessionScope.sid}</p>
```

JSP EL Operator

JSP EL Arithmetic Operators

Arithmetic operators are provided for simple calculations in EL expressions.

They are +, -, *, / or div, % or mod.

JSP EL Logical Operators

They are && (and), || (or) and ! (not).

JSP EL Relational Operators

They are == (eq), != (ne), < (lt), > (gt), <= (le) and >= (ge).

JSP EL Important Points

- EL expressions are always within curly braces prefixed with $ sign, for example ${expr}
- We can disable EL expression in JSP by setting JSP page directive isELIgnored attribute value to TRUE.

  ```
  <%@ page isELIgnored="true" %>
  ```
- JSP EL can be used to get attributes, header, cookies, init params etc, but we can't set the values.
- JSP EL implicit objects are different from JSP implicit objects except pageContext
- JSP EL is NULL friendly, if given attribute is not found or expression returns null, it doesn't throw any exception.

Q10. Explain Exception Handling in JSP.

Ans. **JSP provide 3 different ways to perform exception handling:**

1. Using simple **try...catch** block.
2. Using **isErrorPage** and **errorPage** attribute of **page directive**.
3. Using <**error-page**> tag in **Deployment Descriptor.**

1. **Using try...catch block** is just like how it is used in Core Java.

   Example
   ```
   <html>
   <body>
     <%
      try{
         int i = 100;
         i = i / 0;
         out.println("The answer is " + i);
      }
      catch (Exception e){
         out.println("An exception occurred: " + e.getMessage());
      }
     %>
   </body>
   </html>
   ```

2. **Using isErrorPage and errorPage attribute of page directive**
   Example
   1.jsp
   ```
   <%@page  errorPage= "2.jsp" %>
   <%    int i=10;
         i=i/0; %>
   ```
   2.jsp
   ```
   <%@page isErrorPage="true" %>
   <html> <body>
        An Exception had occured
                   <%  out.println(exception.toString());%>
   </body> </html>
   ```

3. **Using <error-page> tag in Deployment Descriptor**
   - Declaring error page in Deployment Descriptor for entire web application.
   - Specify Exception inside
   - **<error-page>** tag in the Deployment Descriptor.
   - We can even configure different error pages for different exception types, or HTTP error code type(503, 500 etc).

Example1:web.xml
```
     <error-page>
         <exception-type>
             java.lang.Throwable
         </exception-type>
         <location>/error.jsp</location>
     </error-page>
```

Example2:web.xml
```
     <error-page>
         <exception-type>
             java.lang.ArithmeticException
         </exception-type>
     <location>/error.jsp</location>
     </error-page>
```

Example3:web.xml
```
     <error-page>
         <error-code>404</error-code>
         <location>/error.jsp</location>
     </error-page>
     <error-page>
         <error-code>500</error-code>
         <location>/error.jsp</location>
     </error-page>
```

Q11. Write a JSP program to retrieve record from database.

Ans.
```
1. <%@page import="java.sql.*" %>
2. <%
3. Class.forName("com.mysql.jdbc.Driver");
4. Connection con=DriverManager.getConnection(
5.          "jdbc:mysql://localhost:3306/GTU","root","pwd");
6. Statement stmt=con.createStatement();
7. ResultSet rs=stmt.executeQuery("select * from diet");
8. while(rs.next())  {
9.          out.println("<p>"+rs.getString(1));
10.         out.println(rs.getString(2));
11.         out.println(rs.getString(3)+"</p>");
12.     }
13.     con.close();
14.     %>
```

Q12. Explain JSTL core tags library

Ans.
- The core group of tags are the most frequently used JSTL tags.
- The JSTL core tag provides variable support, URL management, flow control etc.

Syntax to include JSTL Core library in your JSP:
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

1	c:out	It display the result of an expression, similar to the way <%=...%> tag work. E.g. `<c:out value="${'Welcome to JSTL'}"/>`
2	c:import	It is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server. E.g.`<c:import var="data" url="http://www.darshan.ac.in"/>`
3	c:set	It is used to set the result of an expression evaluated in a 'scope'. This tag is similar to jsp:setProperty action tag. `<c:set var="Income" scope="session" value="${4000*4}"/>`

4	c:remove	It is used for removing the specified scoped variable from a particular scope
		 `<c:set var="income" scope="session" value="${4000*4}"/>`
		 `<c:remove var="income"/>`

5	c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
		E.g.
		`<c:if test="${income > 8000}">`
		`    <p>My        income        is:        <c:out`
		`value="${income}"/><p>`
		`</c:if>`

6	c:catch	It is used for catching any Throwable exceptions that occurs in the body and optionally exposes it.
		E.g.
		`<c:catch var ="MyException">`
		`    <% int x = 2/0;%>`
		`</c:catch>`

7	c:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java switch statement in which we choose between a numbers of alternatives.
	c:when	It is subtag of <choose > that will include its body if the condition evaluated be 'true'.
	c:otherwise	It is also subtag of < choose > it follows <when> tags and runs only if all the prior condition evaluated is 'false'.

		The <c:when> and <c:otherwise> works like if-else statement. But it must be placed inside <c:choose tag>.
		
		E.g.
		`<c:choose>`
		`    <c:when test="${marks > 75}">`
		`        Congratulations! you hold Distinction`
		`    </c:when>`
		`    <c:otherwise>`
		`        Sorry! Result is unavailable.`
		`    </c:otherwise>`
		`</c:choose>`

8	c:forEach	It is an iteration tag used for repeating the nested body content for fixed number of times. The < c: for each > tag is most commonly used tag because it iterates over a collection of object.
		E.g.`<c:forEach var="i" begin="0" end="5">`
		`    count <c:out value="${i}"/><p>`
		`    </c:forEach>`

9	c:forTokens	It iterates over tokens which is separated by the supplied delimiters.
		`E.g. <c:forTokens items="DIET-CE-Department"`
		`                delims="-" var="name">`

10	c:url	This tag creates a URL with optional query parameter. It is used for url encoding or url formatting. This tag automatically performs the URL rewriting operation. `<c:url value="/MyJspFile.jsp"/>`

11	c:param	It allow the proper URL request parameter to be specified within URL and it automatically perform any necessary URL encoding.
		E.g.
		`<c:param name="CollegeCode" value="054"/>`
		`<c:param name="Name" value="DIET"/>`

12	c:redirect	This tag redirects the browser to a new URL. It is used for redirecting the browser to an alternate URL by using automatic URL rewriting. E.g. `<c:redirect url="http://darshan.ac.in"/>`

Q13. Explain JSTL Function tags library

Ans. The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions.

Syntax:
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"
 %>
```

fn:contains	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith	It is used to test if an input string ends with the specified suffix.
fn:startsWith	It is used for checking whether the given string is started with a particular string value.
fn:toLowerCase	It converts all the characters of a string to lower case.
fn:toUpperCase	It converts all the characters of a string to upper case.
fn:length	It returns the number of characters inside a string, or the number of items in a collection.

fn:indexOf	It returns an index within a string of first occurrence of a specified substring.
fn:substring	It returns the subset of a string according to the given start and end position.
fn:replace	It replaces all the occurrence of a string with another string sequence.
fn:trim	It removes the blank spaces from both the ends of a string.

Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"
                                      prefix="fn" %>
<c:set var="String1" value="  Welcome to diet CE Department  "/>
<c:if test="${fn:contains(String1, 'diet')}">
    <p>Found diet string<p>
    <p>Index of DIET : ${fn:indexOf(String1, "diet")}</p>
    <c:set var="str2" value="${fn:trim(String1)}" />
    <p>trim : ${str2}</p>
    The string starts with "Welcome":
      ${fn:startsWith(String1, 'Welcome')}
    <p>To UPPER CASE: ${fn:toUpperCase(String1)}</p>
    <p>To lower case: ${fn:toLowerCase(String1)}</p>
</c:if>
```

Output:

Q14. Explain JSTL Formatting tags library.

Ans.
1. The formatting tags provide support for message formatting, number and date formatting etc.
2. *Syntax*

   <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number. E.g. `<fmt:parseNumber var="j" type="number"` `value="${Amount}" />`

fmt:formatNumber	It is used to format the numerical value with specific format or precision.
	E.g.
	`<fmt:formatNumber value="${Amount}"`
	`type="currency"    />`
	`<fmt:formatNumber type="number"`
	`maxFractionDigits="6"`
	`value="${Amount}" />`

fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.
	`<fmt:formatDate type="date" value="${Date}" />`
	`<fmt:formatDate type="time" value="${Date}" />`

fmt:parseDate	It parses the string representation of a time and date.
	E.g.
	`<fmt:parseDate value="${date}" var="parsedDate"`
	`pattern="dd-MM-yyyy" />`

fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
	E.g.
	`<fmt:setTimeZone value="IST" />`
	`<c:set var="date" value="<%=new java.util.Date()%>" />`

fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
	E.g.
	`<c:set var="timeZone" value="GMT-8"/>`
	`<fmt:timeZone value="${timeZone}">`

fmt:message	It display an internationalized message. E.g. `<fmt:message key="String"/>`

15. Explain JSTL SQL tags library.

Ans.
- The JSTL sql tags provide SQL support.
- Syntax:

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

sql:query	It is used for executing the SQL query defined in its sql attribute or the body.
sql:setDataSource	It is used for creating a simple data source suitable only for prototyping.
sql:update	It is used for executing the SQL update defined in its sql attribute or in the tag body.
sql:param	It is used to set the parameter in an SQL statement to the specified value.
sql:dateParam	It is used to set the parameter in an SQL statement to a specified java.util.Date value.
sql:transaction	It is used to provide the nested database action with a common connection.
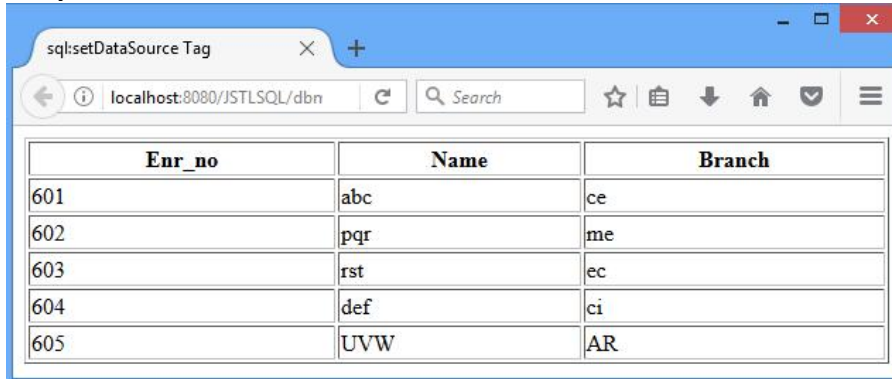
Q16. Write a JSP program using JSTL SQL taglib to display student details in tabular form by iterating through the database table student.

Ans.
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<sql:setDataSource var="db"  driver="com.mysql.jdbc.Driver"
                        url="jdbc:mysql://localhost:3306/gtu"
                        user="root"  password="root"/>
<sql:query dataSource="${db}" var="rs">
    SELECT * from diet;
</sql:query>
<table border="1" width="100%">
<tr>
    <td>Enr_no</td>
    <td>Name</td>
    <td>Branch</td>
</tr>
<c:forEach var="table" items="${rs.rows}">
    <tr>
        <td><c:out value="${table.Enr_no}"/></td>
      <td><c:out value="${table.Name}"/></td>
      <td><c:out value="${table.Branch}"/></td>
    </tr>
```

```
</c:forEach>
</table>
```
Output:

Q17. Explain JSTL XML tags library

Ans.
- The JSTL XML tags are used for providing a JSP-centric way of manipulating and creating XML documents.
- The xml tags provide flow control, transformation etc.
- Syntax:
  <%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>

x:out	Similar to <%= ... > tag, but for XPath expressions.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:set	It is used to sets a variable to the value of an XPath expression.
x:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations.
x:when	It is a subtag of that will include its body if the condition evaluated be 'true'.
x:otherwise	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.
x:if	It is used for evaluating the test XPath expression and if it is true, it will processes its body content.
x:transform	It is used in a XML document for providing the XSL (Extensible Stylesheet Language) transformation.
x:param	It is used along with the transform tag for setting the parameter in the XSLT style sheet.

Example:
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<c:set var="myBook">
<books>
    <myBook>
        <title>TheSecret</title>
        <author>RhondaByrne</author>
    </myBook>
    <myBook>
        <title>Meluha</title>
        <author>Amish</author>
    </myBook>
</books>
 </c:set>
<x:parse xml="${myBook}" var="output"/>
<b>Name of the Book is</b>:
<x:out select="$output/books/myBook[1]/title" />
<p><b>Author of the Meluha is</b>:
<x:out select="$output/books/myBook[2]/author" />  </p>
<x:set var="myTitle"
select="$output/books/myBook[2]/title"/>
<p>x:set:<x:out select="$myTitle" /></p>
```

Output:

Q18. Explain steps to create JSP Custom Tag?

Ans.
- A custom tag is a user-defined JSP language element.
- When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler.
- The Web container then invokes those operations when the JSP page's servlet is executed.
- JSP tag extensions let you create new tags that you can insert directly into a Java Server Page just as you would the built-in tags.

To create a custom tag we need three things:
- **Tag handler class:** In this class we specify what our custom tag will do, when it is used in a JSP page.
- **TLD file:** Tag descriptor file where we will specify our tag name, tag handler class and tag attributes.
- **JSP page:** A JSP page where we will be using our custom tag.
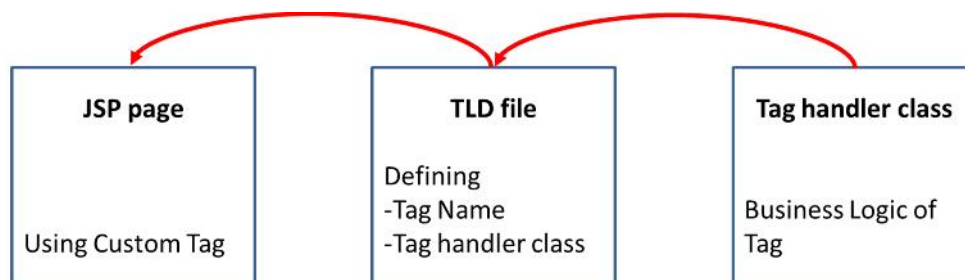
Step-1:Create the Tag handler class
- Define a custom tag named <ex:Hello>
- To create a custom JSP tag, you must first create a Java class that acts as a tag handler.

HelloTag.java
```java
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class HelloTag extends SimpleTagSupport
{
  public void doTag() throws JspException,
                    IOException
 {
    JspWriter out = getJspContext().getOut();
    out.println("Hello Custom Tag!");
  }}
```
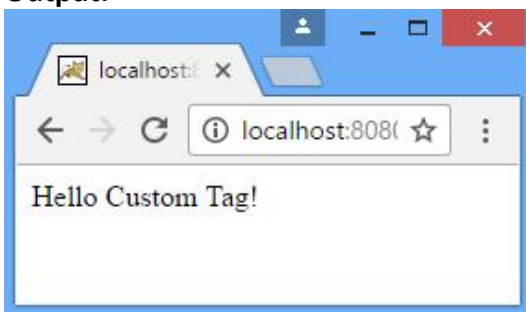
Step-2: Create TLD file

- Tag Library Descriptor (TLD) file contains information of tag and Tag Hander classes.
- It must be contained inside the WEB-INF directory.

```
<taglib>
<tlib-version>1.0</tlib-version>
   <jsp-version>2.0</jsp-version>
   <uri>WEB-INF/tlds/mytags.tld</uri>
   <tag>
     <name>Hello</name>
     <tag-class>MyPackage.HelloTag</tag-class>
     <body-content>empty</body-content>
   </tag>
</taglib>
```

Step-3:Create JSP Page

```
<%@ taglib prefix="ex" uri="WEB-INF/tlds/mytags.tld"%>
<html>
   <body>
     <ex:Hello/>
   </body>
</html>
```

Output:

Q1. Draw the JSF request processing life cycle and briefly give the function of each phase.

Ans. **JSF application lifecycle consist of six phases which are as follows:**

Phase-I: Restore View (RV)
Phase-II: Apply Request Values (ARV)
Phase-III: Process Validations (PV)
Phase-IV: Update Model Values (UMV)
Phase-V: Invoke Application (IA)
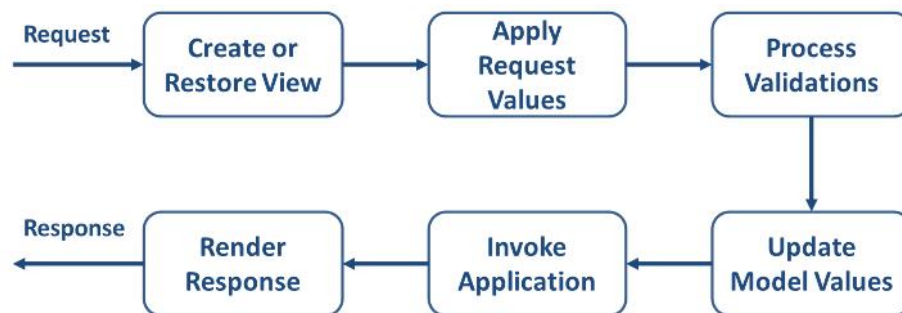**Phase-IV**: Render Response (RR)

Figure: JSF Request Processing Lifecycle

Phase 1: Restore view
- JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request.
- During this phase, the JSF builds the view, wires event handlers and validators to UI components and saves the view in the FacesContext instance.
- The FacesContext instance will now contains all the information required to process a request.

Phase 2: Apply request values
- In this phase, the values that are entered by the user will be updated on each and every individual component defined in the View graph.
- Component stores this value.
- If any of the **Conversions** or the **Validations** fail, then the current processing is terminated and the control directly goes to the **Render Response** for rendering the conversion or the validation errors to the Client.

Phase 3: Process validation
- This Phase will process any Validations that are configured for **UI Components**.
- These validations will only happen for the UI Components only if the property 'rendered' property is set to 'true'.

Phase 4: Update model values
- After the JSF checks that the data is valid, it walks over the component tree and set the corresponding server-side object properties to the component's local values.
- The JSF will update the bean properties corresponding to input component's value attribute.

Phase 5: Invoke application
- During this phase, the JSF handles any application-level events, such as submitting a form / linking to another page.
- Instances MB(Managed Bean), adds value of component to properties of MB, Method of MB is executed.

Phase 6: Render response
- And finally, we have reached the Render Response whose job is to render the response back the Client Application.

Q2.   Explain JSF Standard Component in detail.
Ans.

h:inputText	HTML input of type="text" `<h:inputText id="username" value=""/>`
h:inputSecret	HTML input of type="password" `<h:inputSecret id="password" value="" />`

h:inputHidden	HTML input of type="hidden"
	`<h:inputHidden value = "Hello World"`
	`                id = "hiddenField" />`

h:selectMany Checkbox	A group of HTML check boxes
	`<h:selectManyCheckbox value="">`
	` <f:selectItem  itemValue="1"`
	`               itemLabel="Diet J2SE" />`
	` <f:selectItem itemValue="2"`
	`               itemLabel="Diet J2EE" />`
	`</h:selectManyCheckbox>`

h:selectOneRadio	`<h:selectOneRadio value="Semester">`
	`   <f:selectItem  itemValue="1"`
	`               itemLabel="Sem 4" />`
	`   <f:selectItem  itemValue="2"`
	`               itemLabel="Sem 6" />`
	`</h:selectOneRadio>`

h:outputText	HTML text `<h:outputText value="Username:" />`

h:commandButton	HTML input of type="submit" button.
h:Link	HTML anchor `<h:link value ="Page 1" outcome = "page1" />`

Q3.　Write a short note on JSF Facelets.

Ans.
- JSF provides special tags to create common layout for a web application called facelets tags.
- These tags provide flexibility to manage common parts of multiple pages at one place.
- Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces views using HTML style templates and to build component trees.
- For these tags, you need to use the following namespaces of URI in html node.

```
<html
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:ui = "http://java.sun.com/jsf/facelets"  >
```

The following tags are provided by the Facelet tags

ui:insert	Inserts content into a template. That content is define with the ui:define tag
ui:define	The define tag defines content that is inserted into a page by a template.
ui:composition	The <ui:composition> tag provides a template encapsulating the content to be included in the other facelet.
ui:include	This tag includes the component in the src attribute as a part of the current JSF page.

Facelets features include the following:
- Use of XHTML for creating web pages
- Support for Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries
- Support for the Expression Language (EL)
- Templating for components and pages
- .xhtml instead of .jsp
- No tld files and no tag classes to defined a UIComponent.
- Faster than using JSP.

Advantages of Facelets :
- Support for code reuse through templating and composite components
- Functional extensibility of components and other server-side objects through customization
- Faster compilation time
- Compile-time EL validation
- High-performance rendering

Example
```
<h:body>
        <div id="top" class="top">
            <ui:insert name="top">Top Section</ui:insert>
        </div>
        <ui:define name="top">
                Welcome to Template Client Page
            </ui:define>

    </h:body>
</html>
```

Q4. List the JSF validation tags and explain any two.

Ans.
- JSF provides inbuilt validators to validate its UI components. These tags can validate the length of the field, the type of input which can be a custom object.
- For these tags you need to use the following namespaces of URI in html node.
```
<html
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:f = "http://java.sun.com/jsf/core">
```

f:validateLength	Validates length of a string
	`<f:validateLength  minimum = "10"`
	`                    maximum = "15"/>`

f:validateLongRange	Validates range of numeric value
	`<f:validateLongRange minimum = "10"`
	`                     maximum = "15"/>`

f:validateDoubleRange	Validates range of float value
	`<f:validateDoubleRange`
	`             minimum = "1000.00"`
	`             maximum = "100000.00"/>`

f:validateRegex	Validate JSF component with a given regular expression.
	`<f:validateRegex pattern`
	`             ="((?=.*[a-z]).{6,})" />`

Custom Validator	Creating a custom validator • Create a validator class by implementing javax.faces.validator.Validator interface. • Implement validate() method of above interface. • Use Annotation @FacesValidator to assign a unique id to the custom validator.

Q5. Write a short note on JSF Expression Language.

Ans.
- JSF provides a rich expression language. We can write normal operations using #{operation-expression} notation.
- Some of the advantages of JSF Expression languages are following.
  1. Provides easy access to elements of a collection which can be a list, map or an array.
  2. Provides easy access to predefined objects such as request.
  3. Arithmetic, logical, relational operations can be done using expression language.
  4. Automatic type conversion.
  5. Shows missing values as empty strings instead of NullPointerException.

Notation:

    #{operation-expression}

Example:

    #{i=10} <!-- output: 10-->
    #{10 > 9} <!-- output: true-->
    #{userCar.add} <!– calls add() of UserCar bean   -->

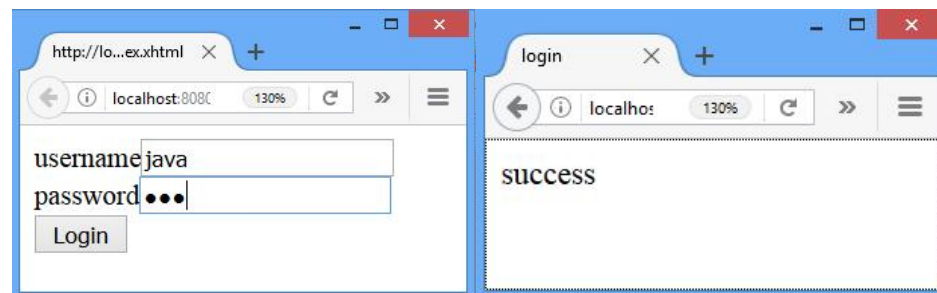## Q6. Write a JSF program to authenticate user with given UID and Password

Ans. **LoginBean.java**

```java
import javax.faces.bean.ManagedBean;
@ManagedBean
public class LoginBean {
    String username;
    String password;
    public String getUsername() {return username;}
    public void setUsername(String username) {
                    this.username = username;}
    public String getPassword() {return password;}
    public void setPassword(String password) {
                    this.password = password;}
   public String login() {
    if(username.equals("java") && password.equals("jsf"))
         return "success";
    else{return "failure";} }}
```

Index.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:c="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
<h:body>
   <h:form id="loginForm">
    <h:outputLabel value="username" />
    <h:inputText  value="#{loginBean.username}" />
    <h:outputLabel value="password" />
   <h:inputSecret value="#{loginBean.password}"></h:inputSecret>
   <h:commandButton value="Login" action="#{loginBean.login}">
   </h:commandButton>
   </h:form>
</h:body></html>
```

Output :

Q7.   Explain JSF Database ACCESS

Ans.

6. We can easily integrate JDBC with JSF for Database Access, let's understand with an example.

7. Files required for JSF DB access are as follows:

   1. AuthenticationBean.java
   2. index.xhtml
   3. success.xhtml
   4. fail.xhtml
   5. Faces-config.xml [Navigational file]

AuthenticationBean.java

```java
import java.sql.*;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
@ManagedBean
@RequestScoped
public class AuthenticationBean
{
    String uname;
    String password;
    public String getUname()
        {return uname;}
    public String getPassword()
        {return password;}
    public void setUname(String uname)
        {this.uname = uname;}
    public void setPassword(String password)
        {this.password = password;}
```

// This method will perform authentication from database

```java
    public String validateFromDB()throws Exception
    {
        int i=0;
    if(uname.equals("diet") && password.equals("diet") ){
        Class.forName("com.mysql.jdbc.Driver");
        Connection con= DriverManager.getConnection (
            "jdbc:mysql://localhost:3306/ajava", "root", "root");
        Statement st=con.createStatement();
        i=st.executeUpdate("insert into cxcy
                values(2011,'dfg','r1')");
    }
    if(i!=0)
        {return "success";}
    else {return "failure";}
}}
```

index.xhtml

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html   xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html">
<h:body>
    <h:form><center>
```

```
    Enter Name :<h:inputText id="name"
                  value="#{authenticationBean.uname}" >
                  </h:inputText>
    Enter Password <h:inputSecret id="pwd"
                  value="#{authenticationBean.password}"/>
    <h:commandButton value="Submit"
      action="#{authenticationBean.validateFromDB}"/>
  </center></h:form>
</h:body></html>
```

success.xhtml
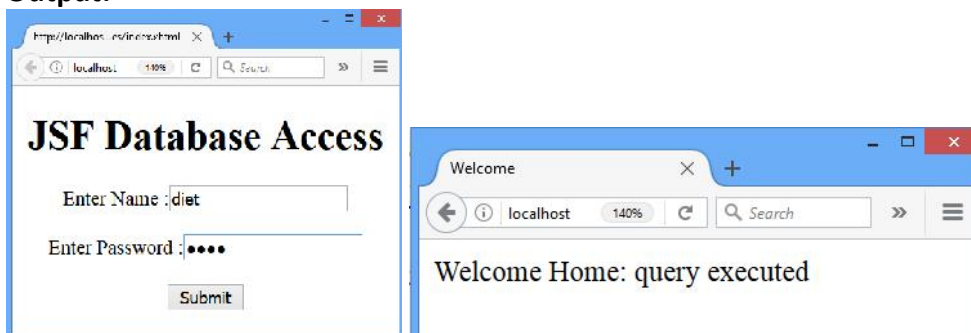```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
              transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:h="http://java.sun.com/jsf/html">
<h:body>
     Welcome Home: query executed
</h:body>
</html>
```

fail.xhtml
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
              Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
              transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html">
<h:body>
   Login Failed
</h:body>
</html>
```

Faces-config.xml

```xml
<?xml version='1.0' encoding='UTF-8'?>

<faces-config version="2.1"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-
                    facesconfig_2_1.xsd">
<navigation-rule>
   <from-view-id>/index.xhtml</from-view-id>
<navigation-case>
 <from-action> #{authenticationBean.validateFromDB}</from-
action>
 <from-outcome>success</from-outcome>
    <to-view-id>/success.xhtml</to-view-id>
</navigation-case>
<navigation-case>
 <from-action> #{authenticationBean.validateFromDB}</from-
action>
 <from-outcome>failure</from-outcome>
   <to-view-id>/fail.xhtml</to-view-id>
</navigation-case>
</navigation-rule>
</faces-config>
```

Output:

enr_no	sname	branch
101	abc	ce
102	pqr	it
103	qwe	me
104	xyz	ce
2011	dfg	r1

Q8. Write a short note on PrimeFaces.

Ans.
- PrimeFaces is a lightweight library with one jar, zero-configuration and no required dependencies
- To use JSF prime faces library we need to use following code:
  ```
  <html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:p="http://primefaces.org/ui ">
  ```

Characteristics of PrimeFaces

1. **Simplicity and Performance**

   PrimeFaces is a lightweight library, all decisions made are based on keeping PrimeFaces as lightweight as possible.

   Usually adding a third-party solution could bring a overhead however this is not the case with PrimeFaces.

   It is just one single jar with no dependencies and nothing to configure.

2. **Ease of Use**

   Components in PrimeFaces are developed with a design principle which states that "A good UI component should hide complexity but keep the flexibility"

3. **Strong Community Feedback**

   PrimeFaces community continuously helps the development of PrimeFaces by providing feedback, new ideas, bug reports and patches.

JSF prime faces provides following collection of tags:

1. <p:inputText>
2. <p:inputSecret>
3. <p:commandButton>
4. <p:commandLink>
5. <p:ajax>
6. <p:barChart>
7. <p:calendar>
8. <p:colorPicker>
9. <p:dialog>
10. <p:fileUpload>
11. <p:fileDownload>
12. <p:inputText>
13. <p:inputSecret>
14. <p:commandButton>
15. <p:commandLink>
16. <p:ajax>
17. <p:barChart>
18. <p:calendar>
19. <p:colorPicker>
20. <p:dialog>
21. <p:fileUpload>
22. <p:fileDownload>

Q1. What is Hibernate? List the advantages of hibernate over JDBC.
Ans.

- Hibernate is used convert object data in JAVA to relational database tables.
- It is an open source Object-Relational Mapping (ORM) for Java.
- Hibernate is responsible for making data persistent by storing it in a database.

JDBC	Hibernate
JDBC maps Java classes to database tables (and from Java data types to SQL data types)	Hibernate automatically generates the queries.
With JDBC, developer has to write code to map an object model's data to a relational data model.	Hibernate is flexible and powerful ORM to map Java classes to database tables.
With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java. So with JDBC, mapping between Java objects and database tables is done manually.	Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
Require JDBC Driver for different types of database.	Makes an application portable to all SQL databases.
Handles all create-read-update-delete (CRUD) operations using SQL Queries.	Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
Working with both Object-Oriented software and Relational Database is complicated task with JDBC.	Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.
JDBC supports only native Structured Query Language (SQL)	Hibernate provides a powerful query language Hibernate Query Language-HQL (independent from type of database)

List the advantages of hibernate over JDBC
- Hibernate is flexible and powerful ORM to map Java classes to database tables.
- Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
- Hibernate automatically generates the queries.
- It makes an application portable to all SQL databases.
- Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
- Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

- Hibernate provides a powerful query language Hibernate Query Language-HQL (independent from type of database).
- Hibernate supports Inheritance, Associations, Collections.
- Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One.
- Hibernate provided Dialect classes, so we no need to write SQL queries in hibernate, instead we use the methods provided by that API.

Q2.   Draw and explain the architecture of Hibernate.
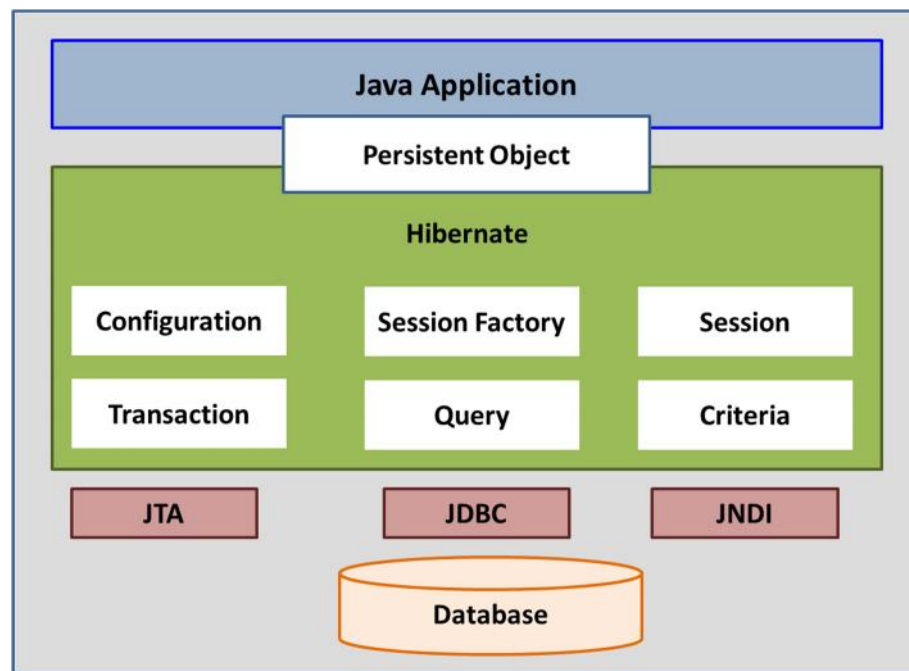**Ans.**

Figure: Hibernate Architecture

- For creating the first hibernate application, we must know the objects/elements of Hibernate architecture.
- They are as follows:
  i. Configuration
  ii. Session factory
  iii. Session
  iv. Transaction factory
  v. Query
  vi. Criteria

1. **Configuration Object**
   - The Configuration object is the first Hibernate object you create in any Hibernate application.
   - It is usually created only once during application initialization.

- The Configuration object provides two keys components:
- **Database Connection:**
    - This is handled through one or more configuration files supported by Hibernate.
    - These files are **hibernate.properties** and **hibernate.cfg.xml**.

- **Class Mapping Setup**:
    - This component creates the connection between the Java classes and database tables.

2. SessionFactory Object
- The SessionFactory is a thread safe object and used by all the threads of an application.
- Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application.
- You would need one SessionFactory object per database using a separate configuration file.
- So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.

3. Session Object
- A Session is used to get a physical connection with a database.
- The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.
- The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed as needed.

4. Transaction Object
- A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality.
- Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

5. Query Object
- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

6. Criteria Object
- Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Q3. **What is HQL? How does it different from SQL? List its advantages.**
Ans. *What is HQL?*
- The Hibernate ORM framework provides its own query language called Hibernate Query Language.
- Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL.
- Therefore, it is database independent query language.

How does it different from SQL?

SQL	HQL
SQL is based on a relational database model	HQL is a combination of object-oriented programming with relational database concepts.
SQL manipulates data stored in tables and modifies its rows and columns.	HQL is concerned about objects and its properties.
SQL is concerned about the relationship that exists between two tables.	HQL considers the relation between two objects.

Advantages of HQL:
- Provides full support for relation operations
- Returns results as objects
- Support polymorphic queries
- Easy to learn and use
- Supports for advanced features
- Provides database independency

Q4. **What is O/R Mapping? How it is implemented using Hibernate. Give an example of Hibernate XML mapping file.**
Ans. Three most important mapping are as follows:
1. Collections Mappings
2. Association Mappings
3. Component Mappings

Collections Mappings
- If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java.
- Hibernate can persist instances of **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List**, and any **array** of persistent entities or values.

Association Mappings:
- The mapping of associations between entity classes and the relationships between tables is the soul of ORM.
- There are the four ways in which the cardinality of the relationship between the objects can be expressed.
- An association mapping can be unidirectional as well as bidirectional.

Mapping type	Description
Many-to-One	Mapping many-to-one relationship using Hibernate
One-to-One	Mapping one-to-one relationship using Hibernate
One-to-Many	Mapping one-to-many relationship using Hibernate
Many-to-Many	Mapping many-to-many relationship using Hibernate

Component Mappings:
If the referred class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the Component class.
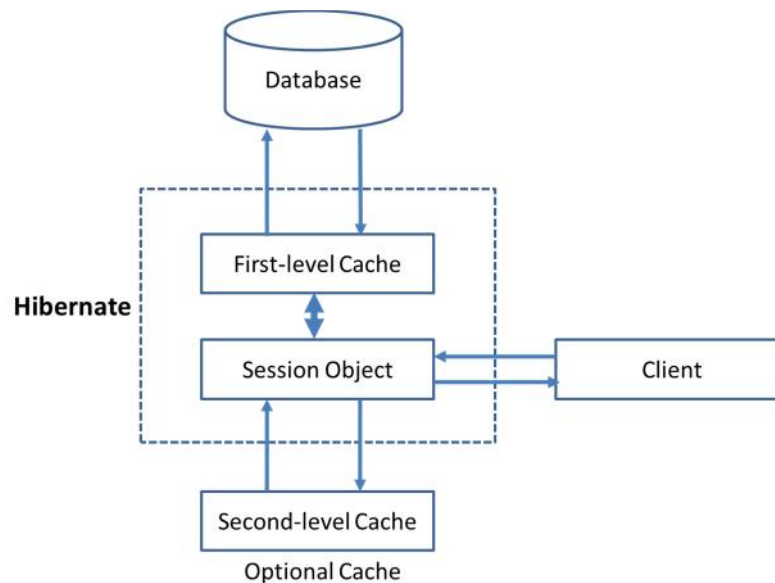The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences.

Give an example of Hibernate XML mapping file.
```
<hibernate-mapping>
  <class name="hibernatetest.Customer" table="customers">
  <id column="C_ID" name="customerID" type="int">
  <generator class="native">
  </generator></id>
  <property name="customerName">
  <column name="name">
  </column></property>
  <property name="customerAddress">
  <column name="address">
  </column></property>
  <property name="customerEmail">
  <column name="email">
  </column></property>
  </class>
</hibernate-mapping>
```

Q5. Explain the Hibernate cache architecture.
Ans.

Figure: Hibernate Cache Architecture

- Caching is all about application performance optimization.
- It is situated between your application and the database to avoid the number of database hits as many as possible.
- To give a better performance for critical applications.

First-level cache:
- The first-level cache is the Session cache.
- The Session object keeps an object under its own control before committing it to the database.
- If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued.
- If you close the session, all the objects being cached are lost.

Second-level cache:
- It is responsible for caching objects across sessions.
- Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache.
- Any third-party cache can be used with Hibernate.
- An org.hibernate.cache.CacheProvider interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.
- While preparing a Hibernate mapping document, we map the Java data types into RDBMS data types.
- The types declared and used in the mapping files are not Java data types; they are not SQL database types either.
- These types are called *Hibernate mapping types*, which can translate from Java to SQL data types and vice versa.

Q6. **Write a program to insert record in to the database using hibernate.**

Ans. **Steps to run first hibernate example with MySQL in Netbeans IDE 8.2**

Step-1: Create the database

```
CREATE DATABASE retailer;
```

Step-2: Create table result

```
CREATE TABLE customers(
    name varchar(20),
    C_ID int NOT NULL AUTO_INCREMENT,
    address  varchar(20),
    email varchar(50),
    PRIMARY KEY(C_ID)
);
```

Step-3: Create new java application.

```
File > New project > Java > Java Application > Next
Name it as HibernateTest.
Then click Finish to create the project.
```

Step-4: Create a POJO(Plain Old Java Objects) class
- We create this class to use variables to map with the database columns.
- Right click the package (hibernatetest) & select New > Java Class Name it as Customer.
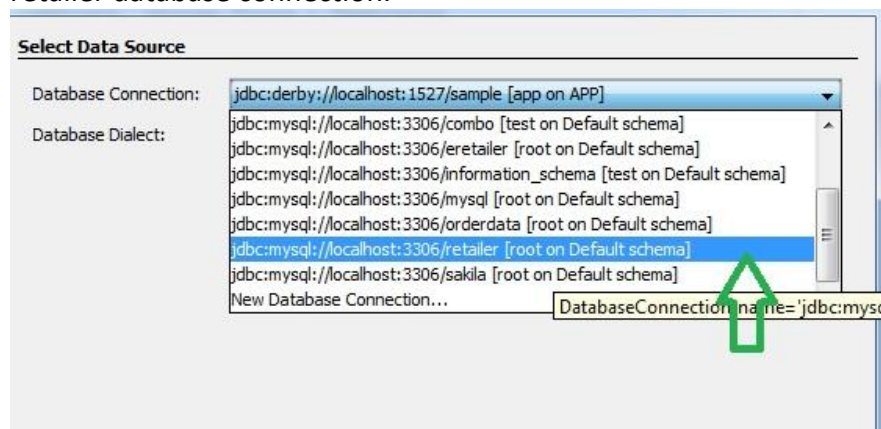- Click Finish to create the class.

```
package hibernatetest;
public class Customer {
    private String customerName;
    private int customerID;
    private String customerAddress;
private String customerEmail;
public void setCustomerAddress(String customerAddress) {
        this.customerAddress = customerAddress;}
public void setCustomerEmail(String customerEmail) {
    this.customerEmail = customerEmail;}
public void setCustomerID(int customerID) {
    this.customerID = customerID; }
public void setCustomerName(String customerName) {
        this.customerName = customerName;     }
public String getCustomerAddress() {
        return customerAddress;     }
public String getCustomerEmail() {
        return customerEmail;     }
public int getCustomerID() {
        return customerID;     }
public String getCustomerName() {
        return customerName;     }}
```

Step-5: Connect to the database we have already created. [retailer]
- Select Services tab lying next to the Projects tab.
- Expand Databases.
- Expand MySQL Server. There we can see the all databases on MySQL sever
- Right click the database retailer. Select Connect.

Step-6: Creating the configuration XML
- Hibernate need a configuration file to create the connection.
- Right click package hibernatetest select New > Other > Hibernate > Hibernate Configuration Wizard.
- Click Next > In next window click the drop down menu of Database Connection and select retailer database connection.

hibernate.cfg.xml

```xml
<hibernate-configuration>
      <session-factory>
        <property name="hibernate.connection.driver_class">
                        com.mysql.jdbc.Driver </property>
        <property name="hibernate.connection.url">

         jdbc:mysql://localhost:3306/retailer</property>
        <property name="hibernate.connection.username">
                              root</property>
        <property name="hibernate.connection.password">
                              root</property>
        <property name="hibernate.connection.pool_size">
                              10</property>
        <property name="hibernate.dialect">
                  org.hibernate.dialect.MySQLDialect</property>
      <property name="current_session_context_class">
                              thread</property>
```

```
            <property name="cache.provider_class">

            org.hibernate.cache.NoCacheProvider</property>
            <property name="show_sql">true</property>
            <property name="hibernate.hbm2ddl.auto">
                             update</property>

            <mapping resource="hibernate.hbm.xml"></mapping>
         </session-factory>
</hibernate-configuration>
```

Step-7: Creating the mapping file [hibernate.hbm]
- Mapping file will map relevant java object with relevant database table column.
- Right click project select New > Other > Hibernate > Hibernate Mapping Wizard
- click Next name it as hibernate.hbm
- click Next> In next window we have to select Class to Map and Database Table.
- After selecting correct class click OK
  Select Database Table
  Click drop down list and select the table you want to map.
  Code for mapping file.

hibernate.hbm.xml
```
  <hibernate-mapping>
    <class name="hibernatetest.Customer" table="customers">
         <id column="C_ID" name="customerID" type="int">
         <generator class="native">    </generator></id>
         <property name="customerName">
             <column name="name">  </column>
         </property>
         <property name="customerAddress">
             <column name="address">  </column>
          </property>
         <property name="customerEmail">
             <column name="email">  </column>
         </property>
    </class>
  </hibernate-mapping>
```

Step-8: Now java program to insert record into the database

```java
package hibernatetest;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class HibernateTest {
  public static void main(String[] args) {
  Session session = null;
  try
  {
     SessionFactory sessionFactory = new
org.hibernate.cfg.Configuration().configure().buildSessionFactory(
);
     session =sessionFactory.openSession();
        session.beginTransaction();
        System.out.println("Populating the database !");
        Customer customer = new Customer();
        customer.setCustomerName("DietCX");
        customer.setCustomerAddress("DIET,Hadala");
        customer.setCustomerEmail("dietcx@darshan.ac.in");
        session.save(customer);
        session.getTransaction().commit();
        System.out.println("Done!");
        session.flush();
        session.close();
     }catch(Exception ){System.out.println(e.getMessage());    } }
}
```

Q7. **Develop program to get all students data from database using hibernate. Write necessary xml files.**

Ans.
```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class HibSelect {
 public static void main(String[] args) {
   Session session = null;
      try
      {
      SessionFactory sessionFactory = new
org.hibernate.cfg.Configuration().configure().buildSessionFactory(
);
     session =sessionFactory.openSession();
     session.beginTransaction();
     System.out.println("Populating the database !");

     hibernatetest.Customer customer= new hibernatetest.Customer();
     Query query=session.createQuery("from hibernatetest.Customer");
                         //here persistent class name is Emp
     List list=query.list();
     while(list.isEmpty())
     {
        System.out.println(list.get(0).toString());
        System.out.println(list.get(1).toString());
        System.out.println(list.get(2).toString());
     }

   session.save(customer);

        session.getTransaction().commit();
        System.out.println("Done!");
        session.flush();
        session.close();
      }catch(Exception e){System.out.println(e.toString());
      } } }
```

Write necessary xml files.

hibernate.cfg.xml

```xml
<hibernate-configuration>
        <session-factory>
          <property name="hibernate.connection.driver_class">
                             com.mysql.jdbc.Driver </property>
          <property name="hibernate.connection.url">

           jdbc:mysql://localhost:3306/retailer</property>
          <property name="hibernate.connection.username">
                                    root</property>
          <property name="hibernate.connection.password">
                                    root</property>
          <property name="hibernate.connection.pool_size">
                                       10</property>
          <property name="hibernate.dialect">
                      org.hibernate.dialect.MySQLDialect</property>
        <property name="current_session_context_class">
                                  thread</property>
        <property name="cache.provider_class">

        org.hibernate.cache.NoCacheProvider</property>
        <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">
                           update</property>

        <mapping resource="hibernate.hbm.xml"></mapping>
      </session-factory>
</hibernate-configuration>
```

hibernate.hbm.xml

```xml
   <hibernate-mapping>
     <class name="hibernatetest.Customer" table="customers">
         <id column="C_ID" name="customerID" type="int">
         <generator class="native">    </generator></id>
         <property name="customerName">
             <column name="name">  </column>
         </property>
         <property name="customerAddress">
             <column name="address">  </column>
          </property>
         <property name="customerEmail">
             <column name="email">  </column>
         </property>
     </class>
   </hibernate-mapping>
```

Q8. Explain Hibernate Annotation.

Ans.
- Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping.
- Consider we are going to use following EMPLOYEE table to store our objects:

```
create table EMPLOYEE (
id INT NOT NULL auto_increment,
first_name VARCHAR(20) default NULL,
 last_name VARCHAR(20) default NULL,
salary INT default NULL,
PRIMARY KEY (id) );
```

- Following is the mapping of Employee class with annotations to map objects with the defined EMPLOYEE table:

```java
import javax.persistence.*;
@Entity
@Table(name = "EMPLOYEE")
public class Employee {
@Id @GeneratedValue
@Column(name = "id")
private int id;
@Column(name = "first_name")
private String firstName;
@Column(name = "last_name")
private String lastName;
@Column(name = "salary")
private int salary;
public Employee() {}
public int getId() {
return id;
}
public void setId( int id ) {
this.id = id;
}
public String getFirstName() {
return firstName;
}
public void setFirstName( String first_name ) {
this.firstName = first_name;
}
public String getLastName() {
return lastName;
}
public void setLastName( String last_name ) {
this.lastName = last_name;
}
```

```
public int getSalary() {
return salary;
}
public void setSalary( int salary ) {
this.salary = salary;
}
}
```

- @Entity Annotation:
  - Employee class which marks this class as an entity bean
- @Table Annotation:
  - The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.
- @Id and @GeneratedValue Annotations:
  - Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation

  @GeneratedValue is same as Auto Increment.
- @Column Annotation:
  - The @Column annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes:
  - **name** attribute permits the name of the column to be explicitly specified.
  - **length** attribute permits the size of the column used to map a value particularly for a String value.
  - **nullable** attribute permits the column to be marked NOT NULL when the schema is generated.
  - **unique** attribute permits the column to be marked as containing only unique values.

Q1. Explain architecture of Spring MVC Framework. Explain all modules in brief.

Ans.
- Spring's web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.
- Spring's DispatcherServlet is completely integrated with Spring IoC container and allows us to use every other feature of Spring.

Following is the Request process lifecycle of Spring 3.0 MVC:
1. The client sends a request to web container in the form of http request.
2. This incoming request is intercepted by Front controller (DispatcherServlet) and it will then tries to find out appropriate Handler Mappings.
3. With the help of Handler Mappings, the DispatcherServlet will dispatch the request to appropriate Controller.
4. The Controller tries to process the request and returns the Model and View object in form of ModelAndView instance to the Front Controller.
5. The Front Controller then tries to resolve the View (which can be JSP, Freemarker, Velocity etc) by consulting the View Resolver object. The selected view is then rendered back to client.

Q2. What is Spring Web MVC framework? List its key features.

Ans.
- Spring links objects together instead of the objects linking themselves together.
- Spring object linking is defined in XML files, allowing easy changes for different application configurations thus working as a plug in architecture.
- In an MVC architecture your controllers handle all requests.
- Spring uses a "DispatcherServlet" defined in the web.xml file to analyze a request URL pattern and then pass control to the correct Controller by using a URL mapping defined in a "spring bean" XML file. All frameworks integrate well with spring.
- Consistent Configuration, open plug-in architecture
- Integrates well with different O/R Mapping frameworks like Hibernate
- Easier to test applications with.
- Less complicated than other frameworks.
- Active user community.
- Spring is well organized and seems easier to learn comparatively
- Spring also supports JDBC Framework that makes it easier to create JDBC Apps.

Advantage of Spring MVC Framework

1. Predefined Templates
2. Loose Coupling
3. Easy to test
4. Lightweight
5. Fast Development
6. Declarative Support
7. Hibernate and JDBC Support
8. MVC Architecture and JavaBean Support

Features of Spring MVC Framework

1. Inversion of Control (IoC) Container
   It is used to provide object reference to class during runtime.
2. Data Access Framework
   It enables developers to easily write code to access the persistent data throughout the application.
3. Transaction Management
   It enables developers to model a wide range of transaction by providing Java Transaction API (JTA).
4. Spring Web Services
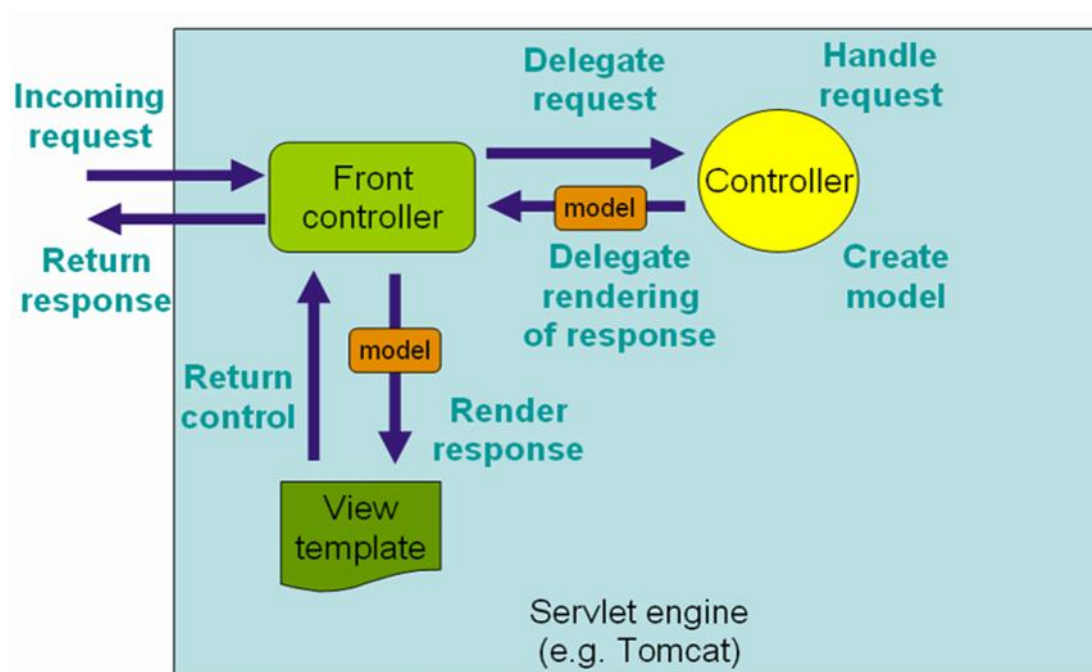   It provides powerful mapping for transmitting incoming XML request to any object.

Figure: Spring MVC Architecture

Features of Spring 3.0:

- Spring 3.0 framework supports Java 5. It provides annotation based configuration support. Java 5 features such as generics, annotations, varargs etc can be used in Spring.
- A new expression language Spring Expression Language SpEL is being introduced. The Spring Expression Language can be used while defining the XML and Annotation based bean definition.
- Spring 3.0 framework supports REST web services.
- Data formatting can never be so easy. Spring 3.0 supports annotation based formatting. We can now use the @DateFimeFormat(iso=ISO.DATE) and @NumberFormat(style=Style.CURRENCY) annotations to convert the date and currency formats.
- Spring 3.0 has started support to JPA 2.0.

Q3. What is Dependency Injection?

Ans.
- Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.
- Dependency Injection makes our programming code loosely coupled.
- DI is a concept of injecting an object into a class rather than explicitly creating object in a class, since IoC container injects object into class during runtime.

Example:Standard code without Dependency Injection
```
public class TextEditor {
    private SpellChecker spellChecker;

    public TextEditor() {
        spellChecker = new SpellChecker();
    }
}
```
Example: Code with Dependency Injection
```
public class TextEditor {
    private SpellChecker spellChecker;

    public TextEditor(SpellChecker spellChecker) {
        this.spellChecker = spellChecker;
    }
}
```
Here, the TextEditor should not worry about SpellChecker implementation. The SpellChecker will be implemented independently and will be provided to the TextEditor at the time of TextEditor instantiation. This entire procedure is controlled by the Spring Framework.

Q4. What is Spring IoC container?

Ans.
- The Spring container is at the core of the Spring Framework.
- The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.
- The Spring container uses DI to manage the components that make up an application.
- The main tasks performed by IoC container are:
  1. to instantiate the application class
  2. to configure the object
  3. to assemble the dependencies between the objects
- There are two types of IoC containers. They are:
  1. BeanFactory
  2. ApplicationContext

Q5. Briefly explain spring bean life cycle.

Ans.
- The life cycle of a spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.
- Though, there is lists of the activities that take place behind the scenes between the time of bean Instantiation and its destruction, but this chapter will discuss only two important bean lifecycle callback methods which are required at the time of bean initialization and its destruction.
- To define setup and teardown for a bean, we simply declare the <bean> with init-method and/or destroy-method parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroy-method specifies a method that is called just before a bean is removed from the container.

HelloWorld.java
```
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }
   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
   public void init(){
      System.out.println("Bean is going through init.");
   }
   public void destroy(){
      System.out.println("Bean will destroy now.");
   }
}
```

Q6. Develop small application using Spring MVC framework.

Ans. **Spring MVC Hello World Example**

There are given 7 steps for creating the spring MVC application. The steps are as follows:

1. Create the request page (optional)
2. Create the controller class
3. Provide the entry of controller in the web.xml file
4. Define the bean in the xml file
5. Display the message in the JSP page
6. Load the spring core and mvc jar files
7. Start server and deploy the project

Step-1: Create the request page (optional)

This is the simple jsp page containing a link. It is optional page. You may direct invoke the action class instead.

index.jsp

```
<a href="hello.html">click</a>
```

Step-2: Create the controller class

- To create the controller class, we are using two annotations @Controller and @RequestMapping.
- The **@Controller** annotation marks this class as Controller.
- The **@Requestmapping** annotation is used to map the class with the specified name.
- This class returns the instance of ModelAndView controller with the mapped name, message name and message value. The message value will be displayed in the jsp page.

HelloWorldController.java

```java
package com.javatpoint;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class HelloWorldController {
    @RequestMapping("/hello")
    public ModelAndView helloWorld() {
        String message = "HELLO SPRING MVC HOW R U";
        return new ModelAndView("hellopage", "message", message);

    }
}
```

Step-3: Provide the entry of controller in the web.xml file

In this xml file, we are specifying the servlet class DispatcherServlet that acts as the front controller in Spring Web MVC. All the incoming request for the html file will be forwarded to the DispatcherServlet.

web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
 <servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
</web-app>
```

Step-4: Define the bean in the xml file

- This is the important configuration file where we need to specify the ViewResolver and View components.
- The context:component-scan element defines the base-package where DispatcherServlet will search the controller class.
- Here, the InternalResourceViewResolver class is used for the ViewResolver.
- The prefix+string returned by controller+suffix page will be invoked for the view component.
- This xml file should be located inside the WEB-INF directory.

spring-servlet.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
```

```
 xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.0.xsd">
    <context:component-scan  base-package="com.javatpoint" />
    <bean class="org.springframework.web.servlet.view.InternalRe
sourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

Step-5: Display the message in the JSP page

This is the simple JSP page, displaying the message returned by the Controller.

It must be located inside the WEB-INF/jsp directory for this example only.

hellopage.jsp

```
Message is: ${message}
```