2(B)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
String url = "jdbc:mysql://localhost:3306/your_database";
String username = "your_username";
String password = "your_password";
Connection connection = DriverManager.getConnection(url, username, password);
String insertSQL = "INSERT INTO your_table (id, name, hourly_rate, part_time) VALUES (?, ?, ?, ?)";
PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

// Set values for the placeholders
preparedStatement.setInt(1, 1); // Set the id
preparedStatement.setString(2, "John Doe"); // Set the name
preparedStatement.setDouble(3, 20.5); // Set the hourly rate
preparedStatement.setBoolean(4, true); // Set the part-time status

// Execute the insert statement
int rowsInserted = preparedStatement.executeUpdate();
if (rowsInserted > 0) {
   System.out.println("Insertion successful.");
} else {
   System.out.println("Insertion failed.");
}

// Close the PreparedStatement
preparedStatement.close();
String deleteSQL = "DELETE FROM your_table WHERE id = ?";
PreparedStatement deleteStatement = connection.prepareStatement(deleteSQL);

int idToDelete = 1; // Set the id to delete
deleteStatement.setInt(1, idToDelete);

int rowsDeleted = deleteStatement.executeUpdate();
if (rowsDeleted > 0) {
   System.out.println("Deletion successful.");
} else {
   System.out.println("Deletion failed or no matching record found.");
}

// Close the PreparedStatement
deleteStatement.close();
```

Steps to connect JDBC;
Jdbc slide 42 to 53

3(b)

MyServer.java

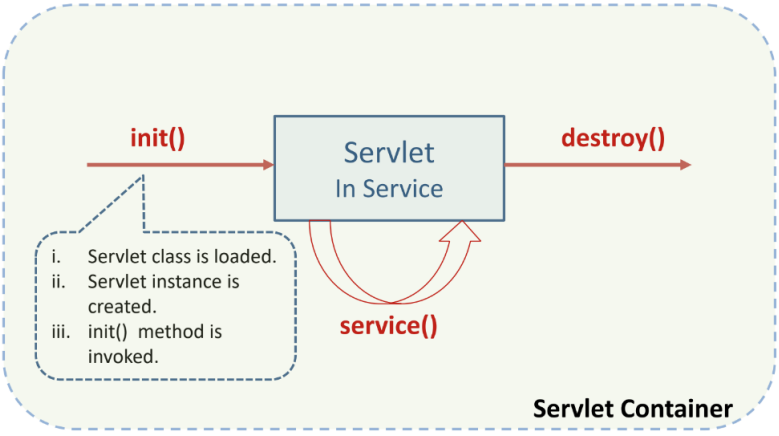
```
import java.io.*;
import java.net.*;

public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(1111);

Socket s=ss.accept();

DataInputStream dis=new DataInputStream
                  (s.getInputStream());
String  str=(String)dis.readUTF();

System.out.println("message= "+str);
ss.close();

}catch(Exception e)
{System.out.println(e);}
 }//psvm
}//class

Output
message= Hello Server
```

MyClient.java

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

public class MyClient {
public static void main(String[] args){
 try {
 Socket s=new Socket("localhost",1111);

 DataOutputStream dout=new
DataOutputStream(s.getOutputStream());

 dout.writeUTF("Hello Server");
 //Writes string to underlying o/p
                                    stream
  }//try
  catch(Exception e)
  {System.out.println(e);}
 } //psvm
}//class
```

4()a

Servlet Life Cycle



Servlet Life Cycle: init()

i. Servlet class is loaded

The classloader 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.**

ii. Servlet instance is created

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.

A Web application runs within a **Web container** of a Web server. Web container provides runtime environment.

iii. Init() method is invoked

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

Servlet Life Cycle: init()

Syntax:

```
public void init(ServletConfig config)
                                throws ServletException
{
    //initialization...
}
```

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

Servlet Life Cycle: 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.

Servlet Life Cycle: Service()

Syntax:

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

Servlet Life Cycle: Destroy()

- 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
  i. database connections,
  ii. halt background threads,
  iii. write cookie lists or hit counts to disk, and
  iv. perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.

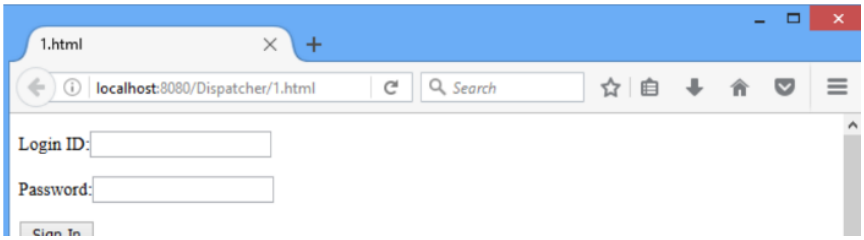
Servlet Life Cycle: Destroy()

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

4b

RequestDispatcher: 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>
```

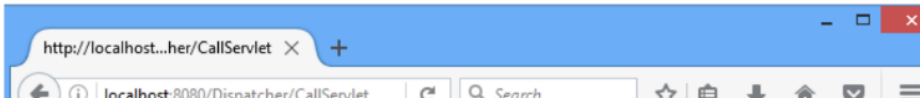


RequestDispatcher: Validate Servlet

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

RequestDispatcher: 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. }
```

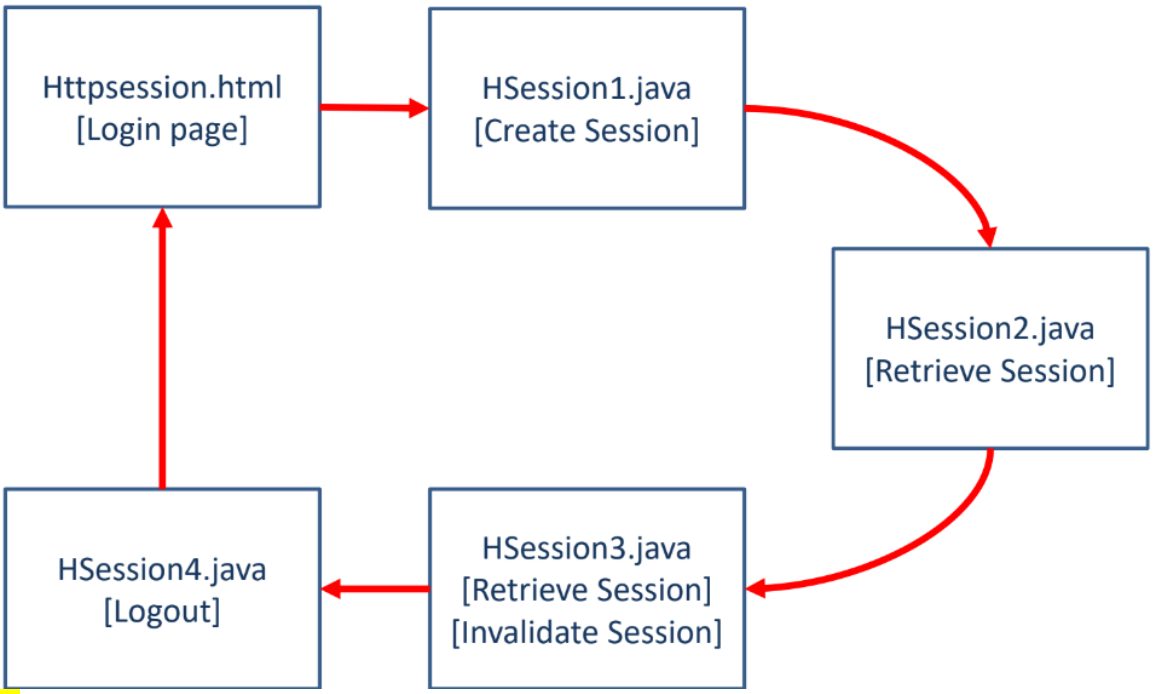


5a

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.
Servlet provides HttpSession Interface which 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.



5b

Jdbc slide 42 to 53

5c

To achieve one-way communication where only the client can send data to the server and the server is restricted from sending data to the client, you can use a simple socket-based communication approach in Java. This involves the client initiating a connection to the server and sending data, while the server listens for incoming connections and processes data from clients. Here's a step-by-step process with Java code examples for both the client and server:

Page 6

For steps and diagram 7 to 15

MyServer.java

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

public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(1111);

Socket s=ss.accept();

DataInputStream dis=new DataInputStream
                  (s.getInputStream());
String  str=(String)dis.readUTF();

System.out.println("message= "+str);
ss.close();

}catch(Exception e)
{System.out.println(e);}
 }//psvm
}//class

Output
message= Hello Server
```

MyClient.java

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

public class MyClient {
public static void main(String[] args){
 try {
 Socket s=new Socket("localhost",1111);

 DataOutputStream dout=new
DataOutputStream(s.getOutputStream());

 dout.writeUTF("Hello Server");
 //Writes string to underlying o/p
                                    stream
  }//try
  catch(Exception e)
  {System.out.println(e);}
 } //psvm
}//class
```