



Silver Oak College of Engineering & Technology

GUJARAT TECHNOLOGICAL UNIVERSITY

BACHELOR OF ENGINEERING

ADVANCED JAVA PROGRAMMING

(3160707)

6th SEMESTER

COMPUTER ENGINEERING

Laboratory Manual



VISION

To create competent professionals in the field of Computer Engineering and promote research with a motive to serve as a valuable resource for the IT industry and society.

MISSION

1. To produce technically competent and ethically sound Computer Engineering professionals by imparting quality education, training, hands on experience and value based education.
2. To inculcate ethical attitude, sense of responsibility towards society and leadership ability required for a responsible professional computer engineer.
3. To pursue creative research, adapt to rapidly changing technologies and promote self-learning approach in Computer Engineering and across disciplines to serve the dynamic needs of industry, government and society.

Program Educational Objectives (PEO):

- PEO1: To provide the fundamentals of science, mathematics, electronics and computer science and engineering and skills necessary for a successful IT professional.
- PEO2: To provide scope to learn, apply skills, techniques and competency to use modern engineering tools to solve computational problems.
- PEO3: To enable young graduates to adapt to the challenges of evolving career opportunities in their chosen fields of career including higher studies, research avenues, entrepreneurial activities etc.

PEO4: To inculcate life-long learning aptitude, leadership qualities and teamwork ability with sense of ethics for a successful professional career in their chosen field.



PROGRAM OUTCOMES (POs) Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change



ADVANCED JAVA PROGRAMMING PRACTICAL BOOK

DEPARTMENT OF COMPUTER ENGINEERING PREFACE

It gives us immense pleasure to present the first edition of Advanced Java Practical Book for the B.E. 3rd year students of Silver Oak Group of Institutes.

The theory and laboratory course of Advanced Java, at Silver Oak Group of Institutes, Ahmedabad, is designed in such a manner that students can develop the basic understanding of the subject during theory classes and gain the hands-on practical experience during their laboratory sessions. Java being one of the most prominent languages opens the door to many application developments involving naïves of Input/output, Networking, the standard utilities, Applets, Frameworks, GUI-based controls and much more. Many of these concepts were detailed in our first part in Object Oriented Programming using Java. In this second part of java, Advanced Java, we will look into networking concepts using TCP/UDP, Java Database Connectivity, web application development with Servlet, JSP and JSF, frameworks like Hibernate and Spring.

The Laboratory Manual presented here takes you onto the learning journey of advanced concepts of Java. In this you will be exploring the wide range of topics like creating a network application with help of TCP/UDP connection, linking & communication a database with java application using JDBC, creating a dynamic server side web application using Servlet, Java Server Pages and Java Server Faces. We will also look into the hibernate Object Relational Mapping (ORM) framework and application frameworkSpring.



INSTRUCTIONS TO STUDENTS

1. Be prompt in arriving to the laboratory and always come well prepared for the experiment.
2. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
3. Students are instructed to come to lab in formal dresses only.
4. Students are supposed to occupy the systems allotted to them and are not supposed to talk or make noise in the lab.
5. Students are required to carry their observation book and lab records with completed exercises while entering the lab.
6. Lab records need to be submitted every week.
7. Students are not supposed to use pen drives in the lab.
8. The grades for the Advanced Java Practical course work will be awarded based on your performance in the laboratory, regularity, recording of experiments in the Advanced Java practical Final Notebook, lab quiz, regular viva-voce and end-term examination.
9. Find the answers of all the questions mentioned under the section, Post Practical Questions" at the end of each experiment in the Advanced Java Practical Book.



CERTIFICATE

This is to certify that

Mr./Ms.....**Savan Rajeshbhai Patel**..... with
enrolment no.**180770107153**.....from
Semester ...**6th**...Div....**B**.. has successfully completed his/her
laboratory experiments in the **ADVANCED JAVA
PROGRAMMING (3160707)**from the department of
.....Computer Engineering.....during the
academic year2020..... -...20 21...

Date of Submission:

Staff In charge:

Head of Department:

TABLE OF CONTENT

Sr. No	Experiment Title	Page No		Date of Start	Date of Completion	Sign	Marks (out of 10)
		To	From				
1	Implement TCP Server for transferring files using Socket and ServerSocket.						
2	Implement TCP Server programming in which more than one client can connect and communicate with Server for sending the string and server returns the reverse of string to each of client.						
3	Implement cookies to store firstname and lastname using Java server pages.						
4	Implement the shopping cart for users for the online shopping. Apply the concept of session.						
5	Implement student registration form with enrollment number, first name, last name, semester, contact number. Store the details in database. Also implement search, delete and modify facility for student records.						
6	Write a Servlet program to print system date and time.						

6	Write a Servlet program to print system date and time.						
7	Write a Servlet to create a Login form with help of request-dispatcher. If the login fails, it should redirect back to loginpage, else to the next page.						
8	Design a web page that takes the Username from user and if it is a valid username prints "Welcome Username". Use JSF to implement.						
9	Write Hibernate application to store customer records and retrieve the customer record including name, contact number, address.						
10	Write an application to keep record and retrieve record of student. The record includes student id, enrollment number, semester, SPI. Use MVC architecture.						

-

PRACTICAL SET – 1

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

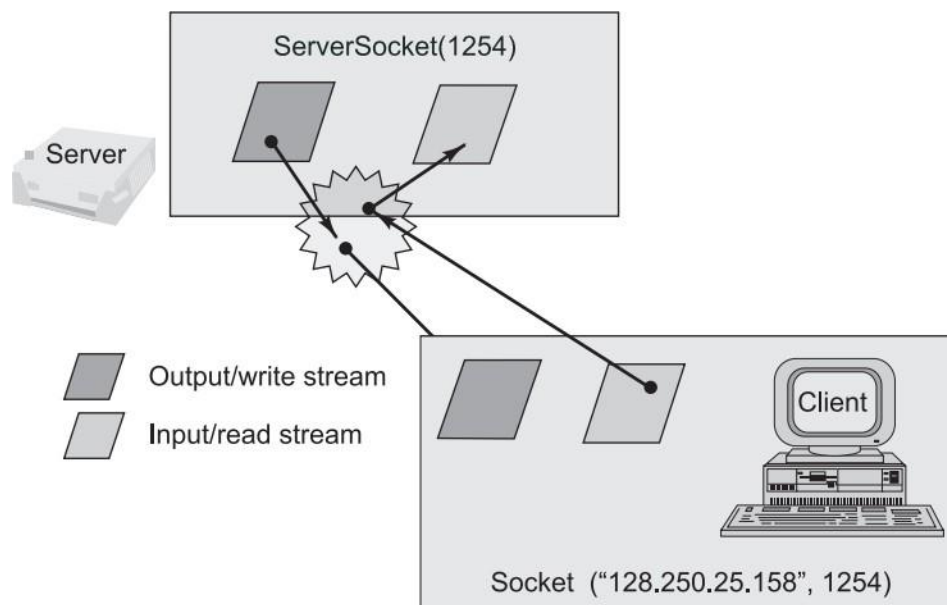
The java.net package provides support for the two common network protocols –

- **TCP** – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** – UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

The two key classes from the java.net package used in creation of server and client programs are:

- ServerSocket
- Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket). In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very like the file concept: developers must open a socket, perform I/O, and close it. Figure illustrates key steps involved in creating socket-based server and client programs.



Socket-based client and server programming

TCP/IP SOCKET PROGRAMMING

A simple Server Program in Java The steps for creating a simple server program are:

1. Open the ServerSocket:
`ServerSocket server = new ServerSocket (PORT);`
2. Wait for the Client Request:
`Socket client =server.accept();`
3. Create I/O streams for communicating to the client
`DataInputStream is = new DataInputStream(client.getInputStream());`
`DataOutputStream os = new DataOutputStream(client.getOutputStream());`
4. Perform communication with client
Receive from client: `String line = is.readLine();`
Send to client: `os.writeBytes("Hello\n");`
5. Close socket:
`client.close();`

A simple Client Program in Java The steps for creating a simple client program are:

1. Create a SocketObject:
`Socket client = new Socket (server, port_id);`
2. Create I/O streams for communicating with the server.
`is = new DataInputStream(client.getInputStream());`
`os = new DataOutputStream(client.getOutputStream());`
3. Perform I/O or communication with the server:
Receive data from the server: `String line = is.readLine();`
Send data to the server: `os.writeBytes("Hello\n");`
4. Close the socket when done:
`client.close();`

References:

1. <http://www.buyya.com/java/Chapter13.pdf>
2. Complete Reference J2EE Publisher :McGrawpublication

Aim : Implement TCP Server for transferring files using Socket and ServerSocket.

Code:

FileServer.java =>

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class FileServer {
    public static void main(String[] args) throws Exception {
        // Initialize Sockets
        ServerSocket ssock = new ServerSocket(5000);
        Socket socket = ssock.accept();
        // The InetAddress specification
        InetAddress IA = InetAddress.getByName("localhost");

        // Specify the file
        File file = new File("/Users/savanpatel/figma.svg");
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);
        // Get socket's output stream
        OutputStream os = socket.getOutputStream();
        // Read File Contents into contents array
        byte[] contents;
        long fileLength = file.length();
        long current = 0;
        long start = System.nanoTime();
        while (current != fileLength) {
            int size = 10000;
            if (fileLength - current >= size)
                current += size;
```

```

        else {
            size = (int) (fileLength - current);
            current = fileLength;
        }
        contents = new byte[size];
        bis.read(contents, 0, size);
        os.write(contents);
        System.out.print("Sending file ... " + (current * 100) /
fileLength + "% complete!");
    }
    os.flush();
    // File transfer done. Close the socket connection!
    socket.close();
    ssock.close();
    System.out.println("File sent succesfully!");
}
}

```

FileClient.java =>

```

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

public class FileClient {
    public static void main(String[] args) throws Exception {
        // Initialize socket
        Socket socket = new Socket(InetAddress.getByName("localhost"),
5000);
        byte[] contents = new byte[10000];
        // Initialize the FileOutputStream to the output file's full path.
        FileOutputStream fos = new FileOutputStream("/Users/savanpatel/
figmaChanged.svg");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        InputStream is = socket.getInputStream();
        // No of bytes read in one read() call
        int bytesRead = 0;
    }
}

```

```

        while ((bytesRead = is.read(contents)) != -1)
            bos.write(contents, 0, bytesRead);
        bos.flush();
        socket.close();
        System.out.println("File saved successfully!");
    }
}

```

Output:

The screenshot shows an IDE with two tabs: FileClient.java 1 and FileServer.java 3. The FileClient.java tab is active, displaying the following code:

```

1  import java.io.BufferedOutputStream;
2  import java.io.FileOutputStream;
3  import java.io.InputStream;
4  import java.net.InetAddress;
5  import java.net.Socket;
6
7  public class FileClient {
8      public static void main(String[] args) throws Exception {
9          // Initialize socket
10         Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
11         byte[] contents = new byte[10000];
12         // Initialize the FileOutputStream to the output file's full path.
13         FileOutputStream fos = new FileOutputStream("/Users/savanpatel/figmaChanged.svg");
14         BufferedOutputStream bos = new BufferedOutputStream(fos);
15         InputStream is = socket.getInputStream();
16         // No of bytes read in one read() call
17         int bytesRead = 0;
18         while ((bytesRead = is.read(contents)) != -1)
19             bos.write(contents, 0, bytesRead);
20         bos.flush();
21         socket.close();
22         System.out.println("File saved successfully!");
23     }
24 }

```

The bottom of the IDE shows the terminal output for two separate terminal windows. The left window shows the output of running FileServer.java, and the right window shows the output of running FileClient.java.

```

savanpatel@Savans-MacBook-Pro java % java FileServer
Sending file ... 83% complete!Sending file ... 100% complete!File sent successfully!
savanpatel@Savans-MacBook-Pro java %

savanpatel@Savans-MacBook-Pro java % java FileClient
File saved successfully!
savanpatel@Savans-MacBook-Pro java %

```

Conclusion:

PRACTICAL SET – 2

java.io.DataInputStream.readUTF() Method

Description : The java.io.DataInputStream.readUTF() method reads in a string that has been encoded using a modified UTF-8 format. The string of character is decoded from the UTF and returned as String.

Declaration : public final String readUTF()

Parameters: NA

Return Value: This method returns a unicode string.

Exception:

IOException – If the stream is closed or the or any I/O error occurs. EOFException – If the input stream has reached the ends.

UTFDataFormatException – If the bytes do not represent a valid modified UTF-8 encoding.

java.io.DataOutputStream.writeUTF() Method

Description: The java.io.DataOutputStream.writeUTF(String str) method writes a string to the underlying output stream using modified UTF-8 encoding.

Declaration: public final void writeUTF(String str)

Parameters: str – a string to be written to the output

stream. **Return Value :** This method does not return any value.

Exception:

IOException – If an I/O error occurs.

Datagrams:

TCP guarantees the delivery of packets and preserves their order on destination. Sometimes these features are not required and since they do not come without performance costs, it would be better to use a lighter transport protocol. This kind of service is accomplished by the UDP protocol which conveys *datagram packets*.

Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

The format of datagram packet is:

| Msg | length | Host | serverPort |

Java supports datagram communication through the following classes:

- DatagramPacket
- DatagramSocket

The class **DatagramPacket** contains several constructors that can be used for creating packet object.

One of them is:

DatagramPacket(byte[] buf, int length, InetAddress address, int port);

This constructor is used for creating a datagram packet for sending packets of length length to the specified port number on the specified host. The message to be transmitted is indicated in the first argument.

The key methods of DatagramPacket class are:

byte[] getData(): Returns the data buffer.

int getLength(): Returns the length of the data to be sent or the length of the data received.

void setData(byte[] buf): Sets the data buffer for this packet.

void setLength(int length) : Sets the length for this packet.

The class **DatagramSocket** supports various methods that can be used for transmitting or receiving data a datagram over the network.

References:

1. https://www.tutorialspoint.com/java/io/dataoutputstream_writeutf.htm
2. Complete Reference J2EE Publisher :McGraw publication

Aim - Implement TCP Server programming in which more than one client can connect and communicate with Server for sending the string and server returns the reverse of string to each of client.

Code:

multiThreadServer.java =>

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class multiThreadServer {
    public static void main(String args[]) {

        Socket s = null;
        ServerSocket ss2 = null;
        System.out.println("Server Listening.....");
        try {
            ss2 = new ServerSocket(4445); // can also use static final
PORT_NUM , when defined

        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Server error");
        }

        while (true) {
            try {
                s = ss2.accept();
                System.out.println("connection Established");
                ServerThread st = new ServerThread(s);
                st.start();
            }
        }
    }
}
```

```
}
```

```
        catch (Exception e) {  
            e.printStackTrace();  
            System.out.println("Connection Error");  
        }
```

```
}
```

```
}
```

```
}
```

```
}
```

```
class ServerThread extends Thread {
```

```
    String line = null;  
    BufferedReader is = null;  
    PrintWriter os = null;  
    Socket s = null;  
    StringBuilder reversedString = new StringBuilder();
```

```
    public ServerThread(Socket s) {  
        this.s = s;  
    }
```

```
    public void run() {  
        try {  
            is = new BufferedReader(new  
InputStreamReader(s.getInputStream()));  
            os = new PrintWriter(s.getOutputStream());
```

```
        } catch (IOException e) {  
            System.out.println("IO error in server thread");  
        }
```

```
        try {  
            line = is.readLine();  
            while (line.compareTo("QUIT") != 0) {
```

```
reversedString.append(line);
```

```
        // reverse StringBuilder input1
        reversedString.reverse();
        os.println(reversedString);
        os.flush();
        System.out.println("Response to Client : " + line);
        line = is.readLine();
    }
} catch (IOException e) {
```

```
        line = this.getName(); // reused String line for getting thread
name
        System.out.println("IO Error/ Client " + line + " terminated
abruptly");
    } catch (NullPointerException e) {
        line = this.getName(); // reused String line for getting thread
name
        System.out.println("Client " + line + " Closed");
    }
}
```

```
finally {
    try {
        System.out.println("Connection Closing..");
        if (is != null) {
            is.close();
            System.out.println(" Socket Input Stream Closed");
        }
    }
```

```
        if (os != null) {
            os.close();
            System.out.println("Socket Out Closed");
        }
        if (s != null) {
            s.close();
            System.out.println("Socket Closed");
        }
    }
```

```

        } catch (IOException ie) {
            System.out.println("Socket Close Error");
        }
    } // end finally
}
}

```

multiThreadClient.java =>

```
// A simple Client Server Protocol .. Client for Echo Server
```

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;

```

```
public class multiThreadClient {
```

```
    public static void main(String args[]) throws IOException {
```

```

        InetAddress address = InetAddress.getLocalHost();
        Socket s1 = null;
        String line = null;
        BufferedReader br = null;
        BufferedReader is = null;
        PrintWriter os = null;

```

```

        try {
            s1 = new Socket(address, 4445); // You can use static final
constant PORT_NUM
            br = new BufferedReader(new InputStreamReader(System.in));
            is = new BufferedReader(new
InputStreamReader(s1.getInputStream()));
            os = new PrintWriter(s1.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
            System.err.print("IO Exception");

```

```
}
```

```
System.out.println("Client Address : " + address);  
System.out.println("Enter Data to echo Server ( Enter QUIT to  
end):");
```

```
String response = null;  
try {  
    line = br.readLine();  
    while (line.compareTo("QUIT") != 0) {  
        os.println(line);  
        os.flush();  
        response = is.readLine();  
        System.out.println("Server Response : " + response);  
        line = br.readLine();  
    }  
}
```

```
}
```

```
    } catch (IOException e) {  
        e.printStackTrace();  
        System.out.println("Socket read Error");  
    } finally {
```

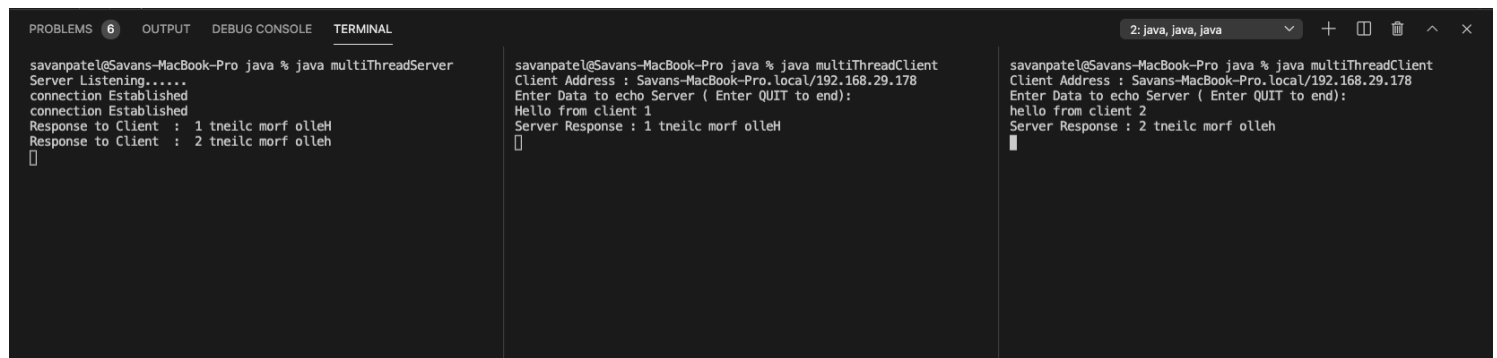
```
        is.close();  
        os.close();  
        br.close();  
        s1.close();  
        System.out.println("Connection Closed");
```

```
}
```

```
}
```

```
}
```

Output:



The screenshot shows an IDE terminal window with three panes. The left pane shows the output of the server program, the middle pane shows the output of the client program, and the right pane shows the output of the client program again. The terminal output is as follows:

```
savanpatel@Savans-MacBook-Pro java % java multiThreadServer
Server Listening.....
connection Established
connection Established
Response to Client : 1 tneilc morf olleH
Response to Client : 2 tneilc morf olleh
█
```

```
savanpatel@Savans-MacBook-Pro java % java multiThreadClient
Client Address : Savans-MacBook-Pro.local/192.168.29.178
Enter Data to echo Server ( Enter QUIT to end):
Hello from client 1
Server Response : 1 tneilc morf olleH
█
```

```
savanpatel@Savans-MacBook-Pro java % java multiThreadClient
Client Address : Savans-MacBook-Pro.local/192.168.29.178
Enter Data to echo Server ( Enter QUIT to end):
Hello from client 2
Server Response : 2 tneilc morf olleh
█
```

Conclusion:

PRACTICAL SET – 3

In a web application, server may be responding to several clients at a time so session tracking is a way by which a server can identify the client. As we know HTTP protocol is stateless which means client needs to open a separate connection every time it interacts with server and server treats each request as a new request.

Now to identify the client, server needs to maintain the state and to do so, there are several session tracking techniques.

- a) Cookies
- b) Hidden Fields
- c) URL Rewriting
- d) Session Object

Cookie

Cookie is a key value pair of information, sent by the server to the browser and then browser sends back this identifier to the server with every request there on.

There are two types of cookies:

Session cookies - are temporary cookies and are deleted as soon as user closes the browser. The next time user visits the same website, server will treat it as a new client as cookies are already deleted.

Persistent cookies - remains on hard drive until we delete them or they expire.

URL Rewriting

URL Rewriting is the approach in which a session (unique) identifier gets appended with each request URL so server can identify the user session.

For example if we apply URL rewriting on `http://localhost:8080/jsp-tutorial/home.jsp`, it will become something like `?jSessionId=XYZ` where `jSessionId=XYZ` is the attached session identifier and value `XYZ` will be used by server to identify the user session.

References:

1. <https://www.javatpoint.com/jsp-implicit-objects>
2. Professional Java Server Prog by Subrahmanyam Allamaraju, Cedric Buest WileyPub.

Aim -Implement cookies to store firstname and lastname using Java server pages.

Code:

index.html =>

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form action="servlet1" method="post">
        First Name:<input type="text" name="firstName"/><br/>
        Last Name:<input type="text" name="lastName"/><br/>

        <input type="submit" value="go"/>
    </form>
</html>
```

FirstServlet.java =>

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
        try{

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
```



```

String fn=request.getParameter("firstName");

String ln=request.getParameter("lastName");
out.print("Welcome "+ fn + " "+ ln + " to my gorgeous website. :)");


Cookie ck1=new Cookie("firstName",fn);
response.addCookie(ck1);

Cookie ck2=new Cookie("lastName",ln);
response.addCookie(ck2);

//creating submit button
out.print("<form action='servlet2' method='post'>");
out.print("<input type='submit' value='go'>");
out.print("</form>");

out.close();

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

```

SecondServlet.java =>

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
        try{

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            Cookie ck1[]=request.getCookies();
            out.print("Hello "+ ck1[0].getValue() + " " + ck1[1].getValue());

```

```

        out.close();

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

```

web.xml =>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://www.springframework.org/
schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd" id="WebApp_ID"
version="3.1">

```

```

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

```

```

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

```

```

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

```

```

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

```

</web-app>

Output:

index.html FirstServlet.java SecondServlet.java web.xml Insert title here

http://localhost:8080/javaLabManual/index.html

First Name: Savan

Last Name: Patel

go

index.html FirstServlet.java SecondServlet.java web.xml http://localhost:8080/javaLabManual/servlet1

http://localhost:8080/javaLabManual/servlet1

Welcome Savan Patel to my gorgeous website. :)

go

index.html FirstServlet.java SecondServlet.java web.xml http://localhost:8080/javaLabManual/servlet2

http://localhost:8080/javaLabManual/servlet2

Hello Savan Patel

storage

Local Storage

http://localhost:8080

Session Storage

IndexedDB

Web SQL

Cookies

http://localhost:8080

Cache

Filter

Only show cookies with an issue

Name	Value	Domain	Path	Expires / Max-...	Size	HttpOnly	Secure	SameSite	Priority
PHPSESSID	4sa7hk7elcpc2rd6r9q2cjd	localhost	/	Session	35				Medium
lastName		localhost	/javaLabManual	Session	13				Medium
connect.sid	lastName	localhost	/	Session	91	✓			Medium
firstName	Savan	localhost	/javaLabManual	Session	14				Medium

Conclusion:

PRACTICAL SET – 4

Action Tags: JSP provides various action tags or elements. Each JSP action tag is used to perform some specific tasks. The action tags are used to control the flow between pages and to use Java Bean. A Java Bean is a java class that should follow following conventions: It should have a no-arg constructor. It should be Serializable. It should provide methods to set and get the values of the properties, known as getter and setter methods. A bean encapsulates many objects into one object, so we can access this object from multiple places. Here, are mentioned action tags used with Bean Class:

The `<jsp:useBean>` action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

```
<jsp:useBean id = "students" class = "Bean.StudentBean">
```

The `setProperty` and `getProperty` action tags are used for developing web application with Java Bean. In web development, bean class is mostly used because it is a reusable software component that represents data.

The `jsp:setProperty` action tag sets a property value or values in a bean using the setter method.

```
<jsp:setProperty name="bean" property="username" value="abc" />
```

The `jsp:getProperty` action tag returns the value of the property.

```
<jsp:getProperty name="obj" property="name" />
```

JSP implicit objects are created by servlet container while translating JSP page to Servlet source to help developers. We can use implicit objects in JSP directly in scriptlets that goes in service - method. However we can't use

jsp implicit objects in JSP declaration because that code will go at class level.

We have 9 jsp implicit objects that we can directly use in JSP page. Seven of them are declared as local variable at the start of `_jspService()` method whereas two of them are part of

_jspService() method argument
that we can use.

Request Object : It is also an implicit object of class HttpServletRequest class and using this object request parameters can be accessed.

Syntax:

request.getParameters("Name"); (where "Name" is the formelement)

Session Object : Session object is of class HttpSession and used to maintain the session information. It stores the information in Name-Value pair.

Syntax:

session.setAttribute("Name","Id"); | session.getAttribute("Name");

References:

1. <https://www.javatpoint.com/jsp-implicit-objects>
2. https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm

Aim: Implement the shopping cart for users for the online shopping. Apply the concept of session.

Code :

index.html =>

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/
bootstrap.bundle.min.js" integrity="sha384-
JEW9xMcG8R+phH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
crossorigin="anonymous"></script>
</head>
<body>
    <form action="servlet1" method="post">
        <div class="form-check">
            <input class="form-check-input" name="product" type="checkbox"
value="mobile" id="flexCheckDefault">
            <label class="form-check-label" for="flexCheckDefault">Mobile</label>
        </div>
        <div class="form-check">
            <input class="form-check-input" name="product" type="checkbox"
value="laptop" id="flexCheckDefault">
            <label class="form-check-label" for="flexCheckDefault">Laptop</label>
        </div>
        <div class="form-check">
            <input class="form-check-input" name="product" type="checkbox"
value="headphones" id="flexCheckDefault">
            <label class="form-check-label" for="flexCheckDefault">Headphones</
label>
        </div>
        <input type="submit" value="go"/>
    </form>
```

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

FirstServlet.java =>

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Arrays;
import java.util.List;
```

```
public class FirstServlet extends HttpServlet {
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
```

```
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
```

```
            String[] products = request.getParameterValues("product");
            List list = Arrays.asList(products);
```

```
            HttpSession session=request.getSession();
            session.setAttribute("product",list);
            out.print("Your products => "+ list);
            out.print("<form action='servlet2' method='post'>");
            out.print("<input type='submit' value='CheckOut'>");
            out.print("</form>");
```

```
            out.close();
```

```
        }catch(Exception e){System.out.println(e);}
    }
```

```
}
```

SecondServlet.java =>

```
import java.io.*;
import java.util.Arrays;
import java.util.List;
import javax.servlet.*;
```



```
import javax.servlet.http.*;
```

```
public class SecondServlet extends HttpServlet {
```

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException{
```

```
try{
```

```
    response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
```

```
    HttpSession session=request.getSession(false);
```

```
    Object n= session.getAttribute("product");
```

```
    out.print("Hello " + n);
```

```
    out.close();
```

```
    }catch(Exception e){System.out.println(e);}
}
```

```
}
```

Web.xml =>

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

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://  
xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://www.springframework.org/  
schema/beans
```

```
    http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
    http://www.springframework.org/schema/context
```

```
    http://www.springframework.org/schema/context/spring-context.xsd
```

```
    http://www.springframework.org/schema/mvc
```

```
    http://www.springframework.org/schema/mvc/spring-mvc.xsd" id="WebApp_ID"
```

```
version="3.1">
```

```
<servlet>
```

```
<servlet-name>s1</servlet-name>
```

```
<servlet-class>FirstServlet</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

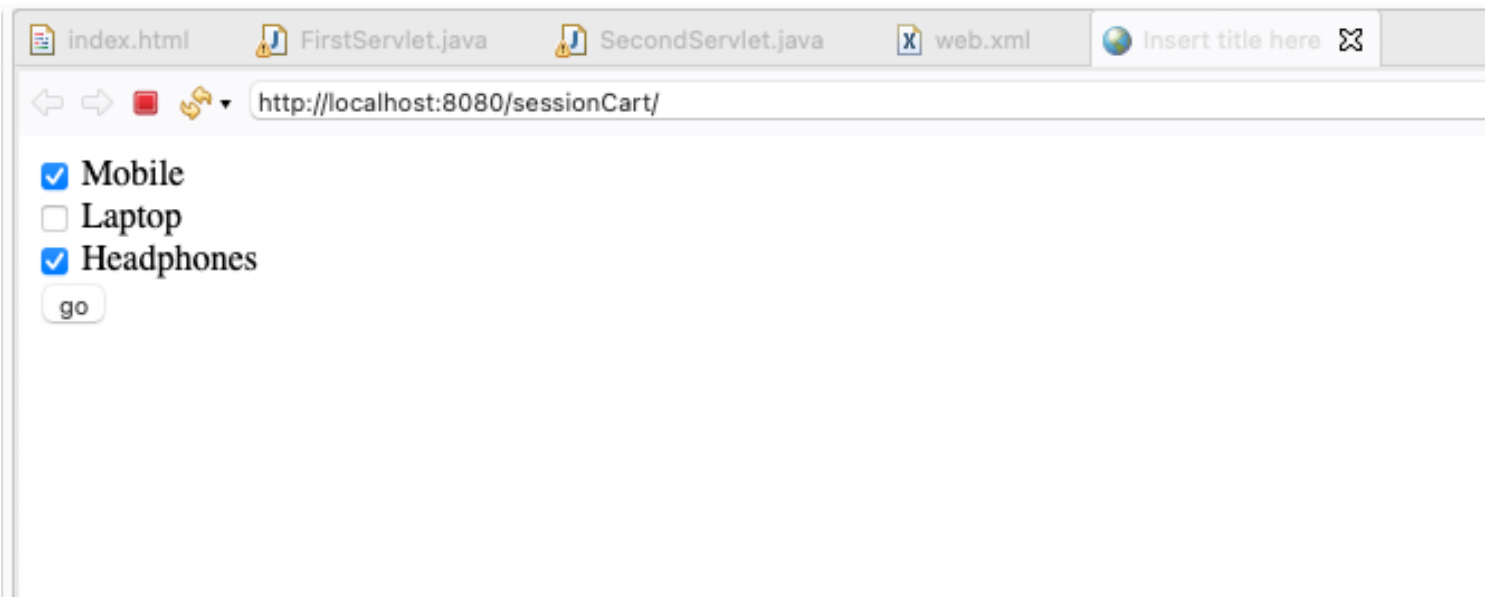
```
<servlet-name>s1</servlet-name>
```

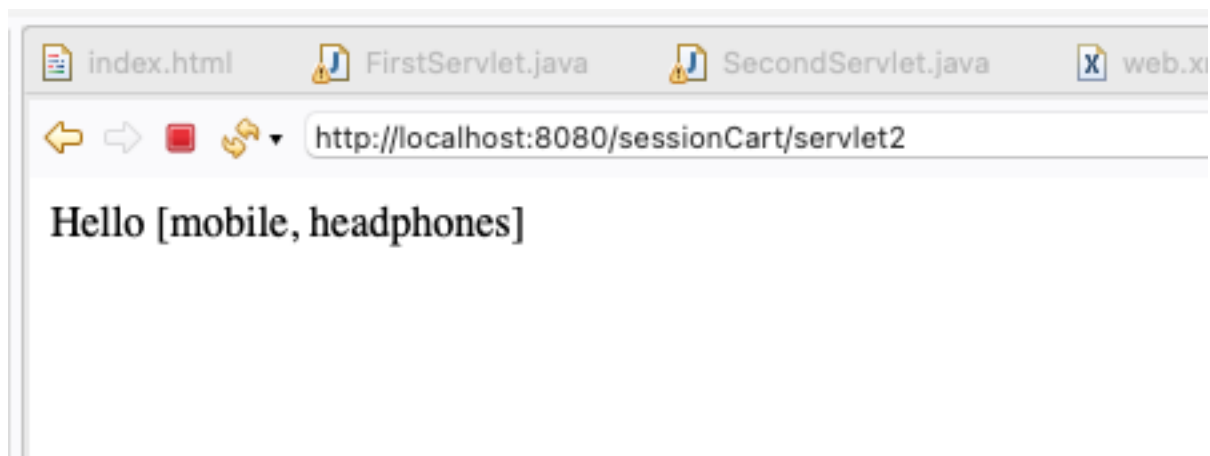
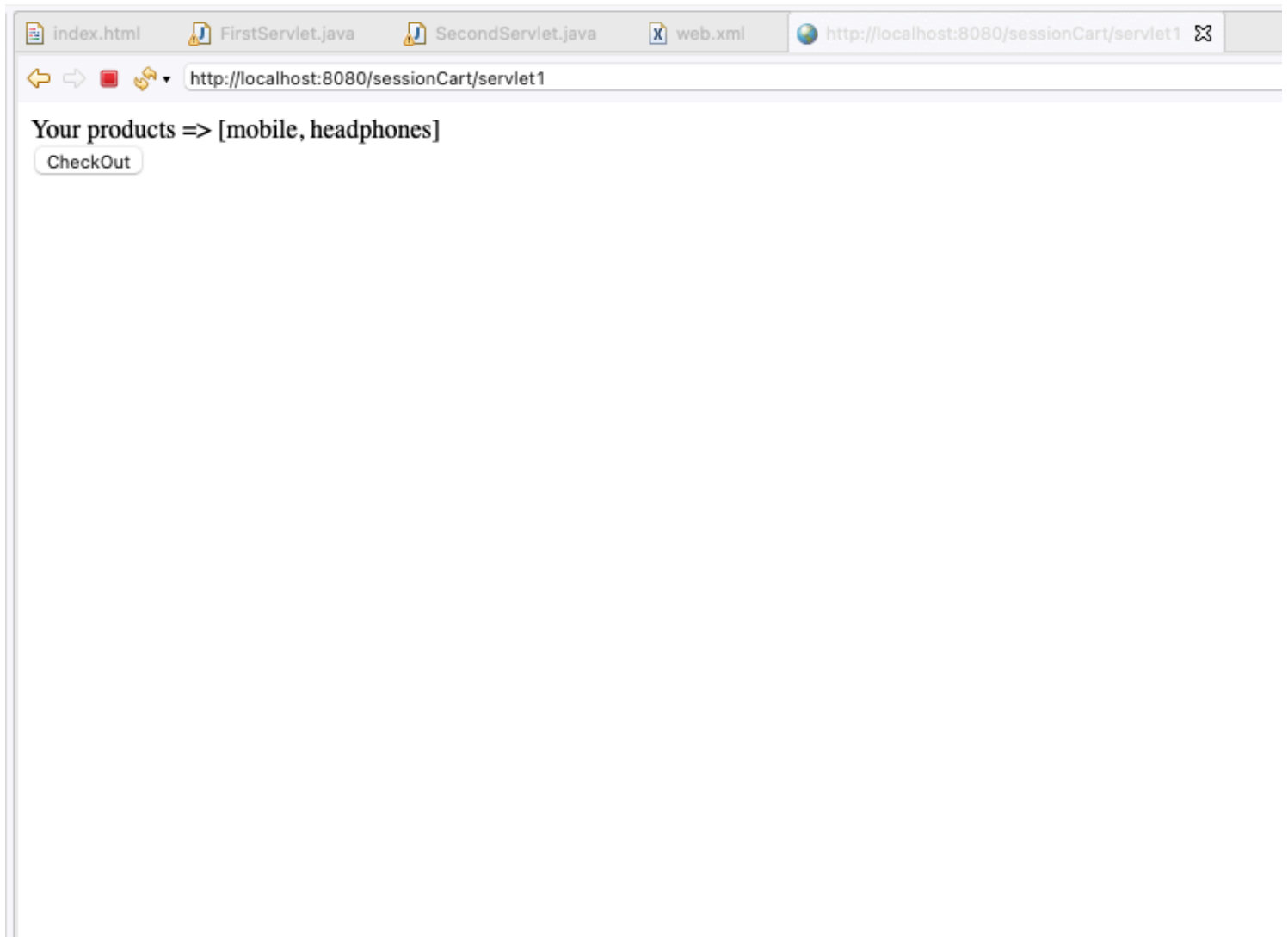
```
<url-pattern>/servlet1</url-pattern>  
</servlet-mapping>
```

```
<servlet>  
<servlet-name>s2</servlet-name>  
<servlet-class>SecondServlet</servlet-class>  
</servlet>
```

```
<servlet-mapping>  
<servlet-name>s2</servlet-name>  
<url-pattern>/servlet2</url-pattern>  
</servlet-mapping>  
</web-app>
```

Output:





Filter

Only show cookies with an issue

Name	Value	Domain	Path	Expires / Max-...	Size	HttpOnly	Secure	SameSite	Priority
connect.sid	s%3AplURNi6TRSOK&axsMP6Vy319co70oJTE.Nj4GSd8HuxK8U3F88zJ9wYyaCMMT1w...	localhost	/	Session	91	✓			Medium
PHPSESSID	4aa7hkj7elcpc2rd9r9q2ccjd	localhost	/	Session	35				Medium
JSESSIONID	676CEA609A8163DE2A3C812546F7024F	localhost	/sessionCart	Session	42	✓			Medium

Conclusion:

PRACTICAL SET – 5

JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as –

- Java Applications
- Java Applets
- Java Servlets
- Java Server Pages (JSPs)
- Enterprise Java Beans (EJBs).

These different executables can use a JDBC driver to access a database, and take advantage of the stored data.

The Statement Interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of **ResultSet** i.e. it provides factory method to get the object of **ResultSet**.

Commonly used methods of Statement interface:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of **ResultSet**.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create,

drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

PreparedStatement

The PreparedStatement interface is a sub interface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance:

The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax:

```
public PreparedStatement prepareStatement(String query) throws SQLException{}
```

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameterindex.
public void setString(int paramIndex, String	sets the String value to the given parameter

value)	index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameterindex.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

CallableStatement

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database using stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

What is the difference between stored procedures and functions?

The differences between stored procedures and functions are given below:

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.

We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

The `prepareCall()` method of `Connection` interface returns the instance of `CallableStatement`.
Syntax is given below:

1. `public CallableStatement prepareCall("{ call procedurename(?,?...?)}");`

The example to get the instance of `CallableStatement` is given below:

1. `CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");`

It calls the procedure my procedure that receives 2 arguments.

References:

[1] <https://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm>

Aim : Implement student registration form with enrollment number, first name, last name, semester, contact number. Store the details in database. Also implement search, delete and modify facility for student records.

Code:

Register.java =>

```
import java.sql.*;
```

```
public class Register {
```

```
    public static void main(String[] args) throws Exception {
```

```
        String url="jdbc:mysql://127.0.0.1:3306/savan";
```

```
        String uname = "root";
```

```
        String pass = "rootdemo";
```

```
        System.out.println("Savan Patel\n180770107153\n6th Sem");
```

```
        Class.forName("com.mysql.cj.jdbc.Driver"); //Do add throws Exception because forName check for  
exception.
```

```
        Connection con = DriverManager.getConnection(url, uname, pass);
```

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

```
        // Adding data to DB
```

```
        String query = "insert into student values ('180770107002','test','demo','6','9898989898') ";
```

```
        int count = st.executeUpdate(query);
```

```
        System.out.println(count + " row's affected");
```

```
        // Displaying table after adding data
```

```
        String query1 = "select * from student";
```

```
        ResultSet rs = st.executeQuery(query1);
```

```
        String userData = "";
```

```
        while(rs.next()){
```

```

        + rs.getString(5);
        userData = rs.getString(1) + " : " + rs.getString(2) + " : " + rs.getString(3) + " : " + rs.getInt(4) + " : "
        System.out.println(userData);
    }
    System.out.println();

    // Updating the values
    String query2 = "UPDATE student SET firstname = 'demo' WHERE enrollment = '180770107002'";
    st.executeUpdate(query2);

    //Displaying again to check updated values
    ResultSet rs1 = st.executeQuery(query1);
    while(rs1.next()){
        userData = rs1.getString(1) + " : " + rs1.getString(2) + " : " + rs1.getString(3) + " : " + rs1.getInt(4) +
        " : " + rs1.getString(5);
        System.out.println(userData);
    }
    System.out.println();

    // Delete Query
    String query3 = "DELETE FROM student WHERE enrollment = '180770107002'";
    st.executeUpdate(query3);

    ResultSet rs11 = st.executeQuery(query1);
    while(rs11.next()){
        userData = rs11.getString(1) + " : " + rs11.getString(2) + " : " + rs11.getString(3) + " : " +
        rs11.getInt(4) + " : " + rs11.getString(5);
        System.out.println(userData);
    }

```

```
System.out.println();
```

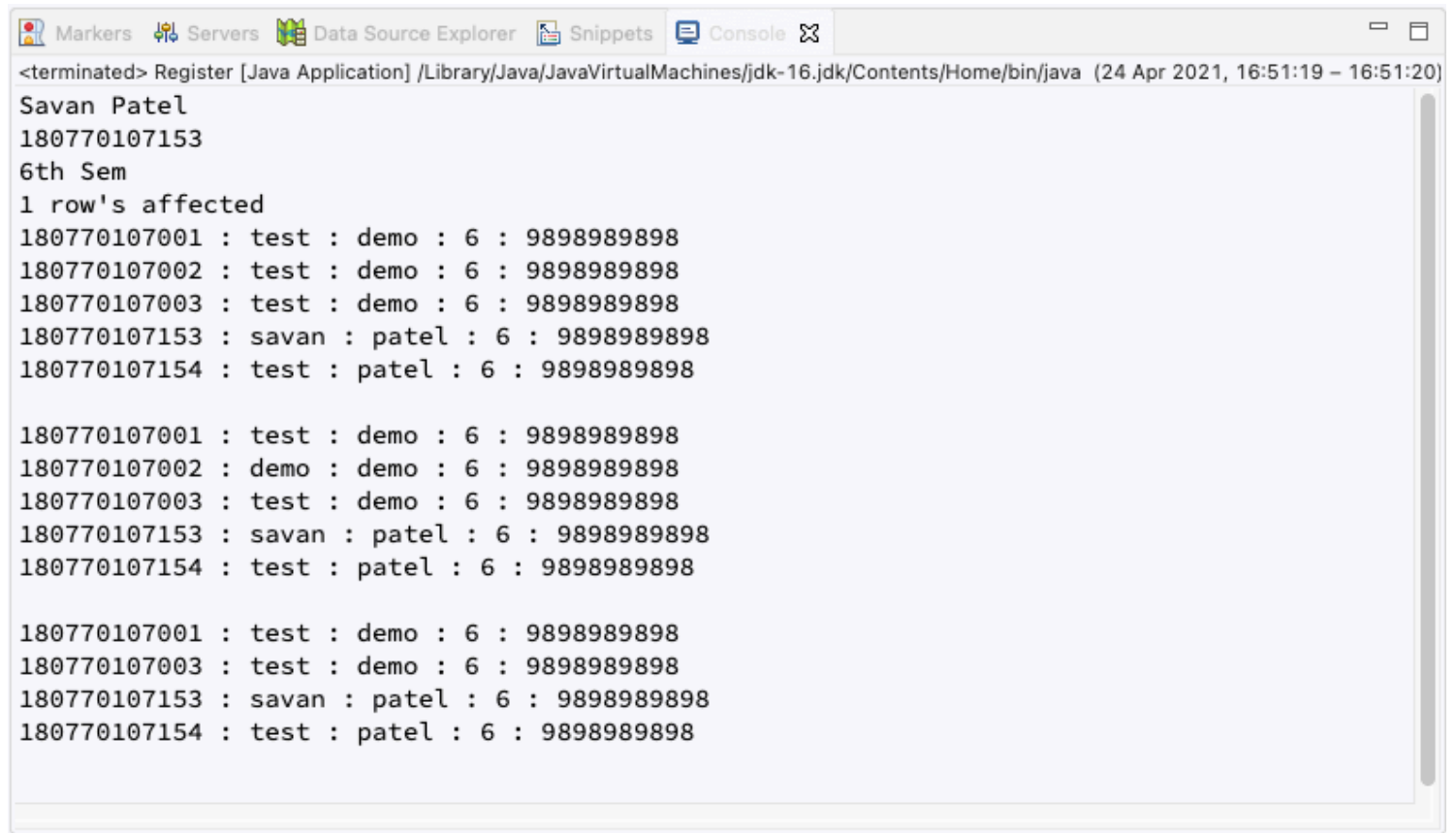
```
st.close();
```

```
con.close();
```

```
}
```

```
}
```

Output



```
<terminated> Register [Java Application] /Library/Java/JavaVirtualMachines/jdk-16.jdk/Contents/Home/bin/java (24 Apr 2021, 16:51:19 - 16:51:20)
Savan Patel
180770107153
6th Sem
1 row's affected
180770107001 : test : demo : 6 : 9898989898
180770107002 : test : demo : 6 : 9898989898
180770107003 : test : demo : 6 : 9898989898
180770107153 : savan : patel : 6 : 9898989898
180770107154 : test : patel : 6 : 9898989898

180770107001 : test : demo : 6 : 9898989898
180770107002 : demo : demo : 6 : 9898989898
180770107003 : test : demo : 6 : 9898989898
180770107153 : savan : patel : 6 : 9898989898
180770107154 : test : patel : 6 : 9898989898

180770107001 : test : demo : 6 : 9898989898
180770107003 : test : demo : 6 : 9898989898
180770107153 : savan : patel : 6 : 9898989898
180770107154 : test : patel : 6 : 9898989898
```

Conclusion:

PRACTICAL SET – 6

Servlets: Servlets are small programs that execute on the server side of a web connection, it dynamically extends the functionality of a web server. It is used to create a web application that reside on the server side and generate a dynamic web page.

The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing our own servlets. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only. All servlets must implement the **javax.servlet.Servlet** interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the **GenericServlet** class provided with the Java Servlet API. The **HttpServlet** class provides methods, such as **doGet()** and **doPost()**, for handling HTTP-specific services.

The servlet life-cycle includes three major functions of a Servlet interface :

1. Initializing a Servlet (**Servlet.init(ServletConfig)**) : **init()** method is used to initialize a servlet object. **ServletConfig** object allows to access name-value configured initialization parameters & container details.
2. Handling a request (**Servlet.service(ServletRequest, ServletResponse)**) : This stage is heart of servlet lifecycle, actual request processing is carried out here. **ServletRequest** object allows to access client's request. **ServletResponse** object assist servlet to send response to client. The actual program is handled in this section.
3. Destroying a Servlet (**Servlet.destroy()**) : It is used to release to all servlet instances and resources. It allows all thread to complete their work and release itself.

The major interfaces of Servlet are:

ServletRequest Interface: It is used to encapsulate client request. It provide important information about client to Servlet.

ServletResponse Interface: It is used to encapsulate client's response. It always listens to a Servlet object.

ServletConfig Interface: It allows a servlet container to pass information to a servlet. It is used to initialize a parameter.

ServletContext Interface: **ServletContext** defines a set of method that a servlet use to communicate with Servlet Container. It allows parameter to be accessed by all servlets in an application.

References:

1. Java-2, The Complete Reference, by Herbert Schildt

2. <https://www.journaldev.com/1877/servlet-tutorial-java>

Aim : Write a Servlet program to print system date and time.

Code:

```
ServletDemo.java =>

// Import required java libraries
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class ServletDemo extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("Enrollment number => 180770107153");
        out.println("Name => Savan Patel");
        out.println("Roll No.=> B-74");
        String title = "Display Current Date & Time";
        Date date = new Date();
        String docType = "<!doctype html public \"-//w3c//dtd html 4.0 \" + \"transitional//
en\">\n";

        out.println(docType +
            "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
            "<h1 align = \"center\">" + title + "</h1>\n" +
            "<h2 align = \"center\">" + date.toString() + "</h2>\n" +
            "</body> </html>"
        );
    }
}
```

web.xml =>

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

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://  
xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://www.springframework.org/  
schema/beans
```

```
    http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
    http://www.springframework.org/schema/context
```

```
    http://www.springframework.org/schema/context/spring-context.xsd
```

```
    http://www.springframework.org/schema/mvc
```

```
    http://www.springframework.org/schema/mvc/spring-mvc.xsd" id="WebApp_ID"
```

```
version="3.1">
```

```
<servlet>
```

```
<servlet-name>date</servlet-name>
```

```
<servlet-class>servletDemo</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

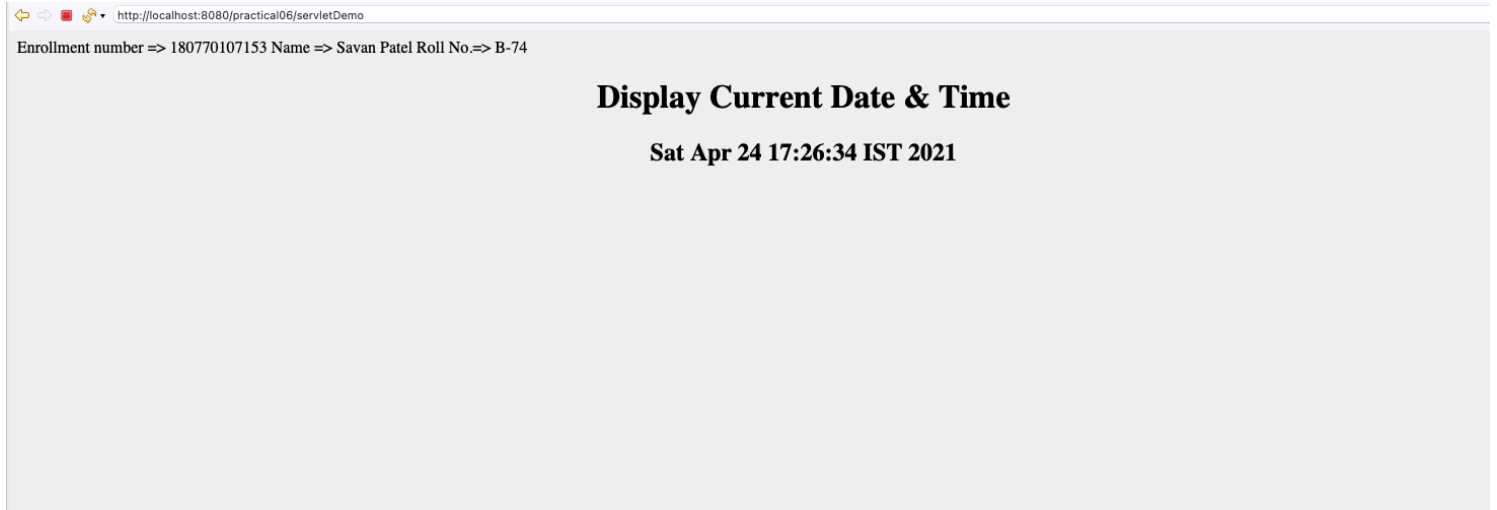
```
<servlet-name>date</servlet-name>
```

```
<url-pattern>/servletDemo</url-pattern>
```

```
</servlet-mapping>
```

```
</web-app>
```


Output:



Conclusion:

PRACTICAL SET – 7

Web.xml : The most important file of a Servlet application is web.xml file. The *web.xml* file is the standard deployment descriptor for the Web application that the Web service is a part of. It declares the filters and servlets used by the service. The general format of the web.xml is as given below. It includes the servlet which are part of your application and its mapping to the appropriate url-pattern, through which it can be accessed. All the Servlet used in application are mentioned in the web.xmlfile.

```
<web-app ...>
<?xml version="1.0" encoding="UTF-8"?>
<servlet>
    <servlet-name>Demo1</servlet-name>
    <servlet-class>Demo1</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Demo1</servlet-name>
    <url-pattern>Demo1</servlet-class>
</servlet-mapping>
</web-app>
```

Apart from the Servlets, filters, listeners, configuration/initialization parameters, error pages, welcome files, security and session management concepts can be included in web.xml file. The initialization parameters can be included either for a specific servlet or for all the servlet in a application, with help of **init-param** and **context-param** tags respectively. They can be accessed with help of methods of **ServletConfig** and **ServletContext** interface, alongwith **getinitparameter(name)** method. To add a initialization parameter to a particular servlet, it can written as below, within the servlet tag:

```
<servlet>....

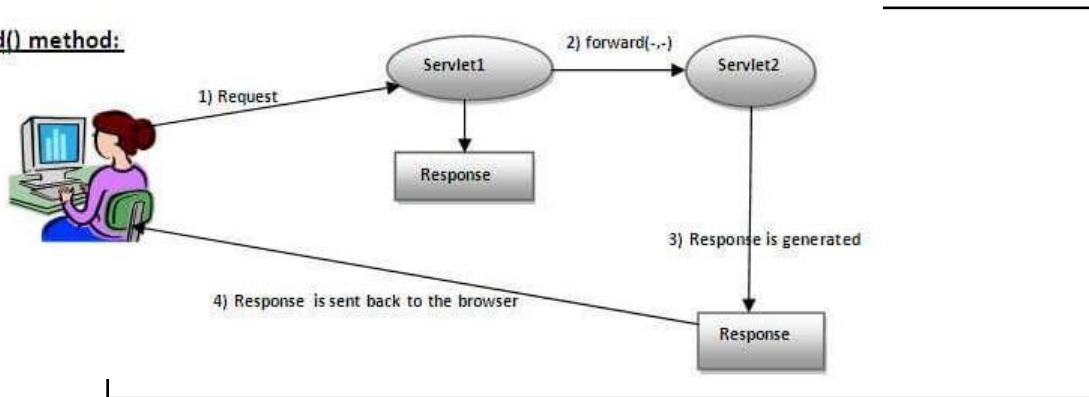
<init-param>
    <param-name> semester </param-name>
    <param-value> 6 </param-value>
</init-param>
</servlet>
```

RequestDispatcher : Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server. The servlet container creates the RequestDispatcher object, which is used as a wrapper around a server resource located at a particular path or given by a particular name. This interface is intended to wrap

servlets, but a servlet container can create RequestDispatcher objects to wrap any type of resource. The RequestDispatcher interface provides two important methods. They are:

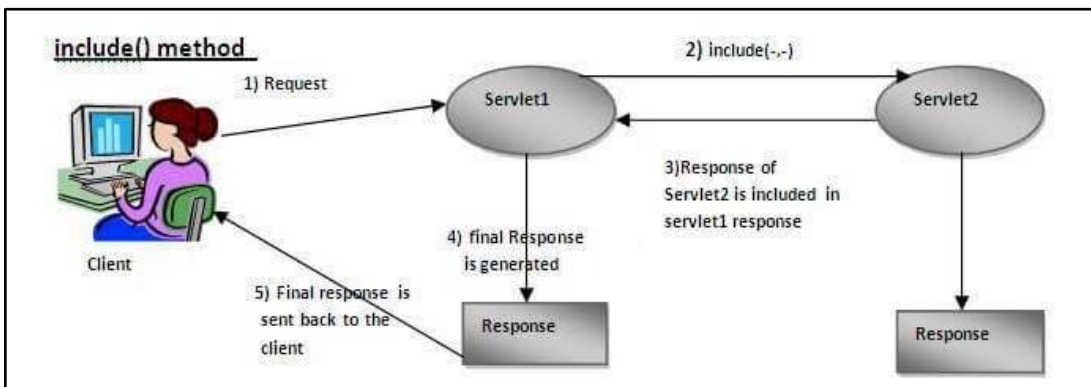
1. **public void forward(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

forward() method:



2. **public void include(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:** Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

include() method



References:

1. Java-2, The Complete Reference, by Herbert Schildt
2. <https://docs.oracle.com/>

Aim : Write a Servlet to create a Login form with help of request dispatcher. If the login fails, it should redirect back to loginpage, else to the next page.

Code:

Index.html =>

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form action="servlet1" method="post">
        Name: <input type="text" name="userName"/><br/>
        Password: <input type="password" name="userPass"/><br/>
        <input type="submit" value="login"/>
    </form>
</body>
</html>
```

Login.java =>

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Login extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name=request.getParameter("userName");
        String pass=request.getParameter("userPass");

        if(name.equals("savan") &&
```

```

        pass.equals("test")){
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
rd.forward(request, response);
}
else{
out.print("Sorry UserName or Password Error!");
RequestDispatcher rd=request.getRequestDispatcher("/index.html");
rd.include(request, response);

}
}
}

```

WelcomeServlet.java =>

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

```

```

public class WelcomeServlet extends HttpServlet {

```

```

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

```

```

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

```

```

        String n=request.getParameter("userName");
        out.print("Welcome "+n);
    }

```

```

}

```

Web.xml =>

```

<?xml version="1.0" encoding="UTF-8"?>

```

```

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://www.springframework.org/
schema/beans

```

```

    http://www.springframework.org/schema/beans/spring-beans.xsd

```

```

    http://www.springframework.org/schema/context

```

```
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd" id="WebApp_ID"
version="3.1"> <display-name>practical07</display-name>
<servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>Login</servlet-class>
</servlet>
<servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>WelcomeServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/servlet2</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

Output:

Name:

Password:

Sorry UserName or Password Error!

Name:

Password:

Name:

Password:

Welcome savan

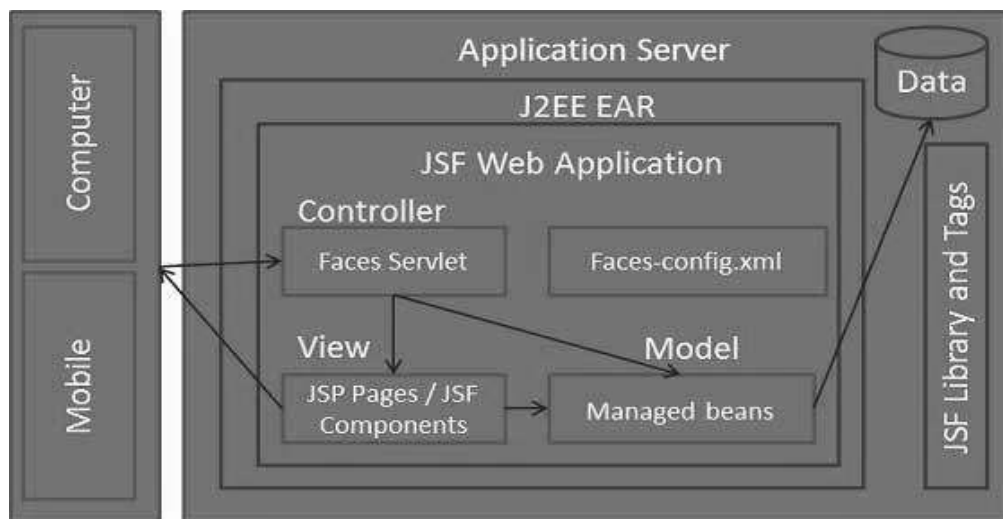
Conclusion:

PRACTICAL SET – 8

JavaServer Faces (JSF) is a MVC web framework that simplifies the construction of User Interfaces (UI) for server-based applications using reusable UI components in a page. JSF provides a facility to connect UI widgets with data sources and to server-side event handlers. The JSF specification defines a set of standard UI components and provides an Application Programming Interface (API) for developing components. JSF enables the reuse and extension of the existing standard UI components.

MVC Design Pattern

- Model - Carries Data and logic
- View - Shows User Interface
- Controller - Handles processing of an application
- The purpose of MVC design pattern is to separate model and presentation enabling developers to focus on their core skills and collaborate more clearly.
- Web designers have to concentrate only on view layer rather than model and controller layer. Developers can change the code for model and typically need not change view layer. Controllers are used to process user actions. In this process, layer model and views may be changed.



Object	Class	Scope
Out	javax.servlet.jsp.JspWriter	Page
Request	javax.servlet.HttpServletRequest	Request
Response	javax.servlet.ServletResponse	Page
Session	javax.servlet.http.HttpSession	Session
Application	javax.servlet.ServletContext	Application

exception	java.lang.Throwable	Page
Config	javax.servlet.ServletConfig	Page
Page	java.lang.Object	Page
PageContext	javax.servlet.jsp.PageContext	Page

JSF application life cycle consists of 6 phases which are as follows

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, 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 contain all the information required to process a request.

Phase 2: Apply request values

- After the component tree is created/restored, each component in the component tree uses the decode method to extract its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors.
- If any decode methods event listeners called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.

Phase 3: Process validation

- During this phase, JSF processes all validators registered on the component tree. It examines the component attribute rules for the validation and compares these rules to the local value stored for the component.
- If the local value is invalid, JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and displays the same page again with the error message.

Phase 4: Update model values

- After the JSF checks that the data is valid, it walks over the component tree and sets the corresponding server-side object properties to the components' local values. JSF will update the bean properties corresponding to the input component's value attribute.
- If any updateModels methods called renderResponse on the current FacesContext instance, JSF moves to the render response phase.

Phase 5: Invoke application

- During this phase, JSF handles any application-level events, such as submitting a form/linking to another page.

Phase 6: Render response

- During this phase, JSF asks container/application server to render the page if the application is using JSP pages. For initial request, the components represented on the page will be added to the component tree as JSP container executes the page. If this is not an initial request, the component tree is already built so components need not be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags in the page.
- After the content of the view is rendered, the response state is saved so that subsequent requests can access it and it is available to the restore view phase.

References:

1. Java-2, The Complete Reference, by Herbert Schildt
2. <https://www.oracle.com/java/technologies/javaserverfaces.html>

Aim :Design a web page that takes the Username from user and if it is a valid username prints “Welcome Username”. Use JSF to implement.

Code:

index.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">
<!-- Dont forget to add space b/w PUBLC and system id -->
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
<title>User Form</title>
</h:head>
<h:body>
<h:form>
<h:outputLabel for="username">User Name</h:outputLabel>
<h:inputText id="username" value="#{user.name}" required="true"
requiredMessage="User Name is required" /><br/>
<h:commandButton id="submit-button" value="Submit" action="response.xhtml" />
</h:form>
</h:body>
</html>
```

Response.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://xmlns.jcp.org/jsf/html">
<h:head>
<title>Welcome Page</title>
</h:head>
<h:body>
<h2>Hello, <h:outputText value="#{user.name}"></h:outputText></h2>
</h:body>
</html>
```

User.java =>

```
package managedbean;
```

```
import javax.faces.bean.ManagedBean;
```

```
import javax.faces.bean.RequestScoped;
```

```
@ManagedBean
```

```
@RequestScoped
```

```
public class User {
```

```
String name;
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
}
```

web.xml =>

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

```
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
```

```
xmlns.jcp.org/xml/ns/javaee
```

```
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
```

```
<context-param>
```

```
<param-name>javax.faces.PROJECT_STAGE</param-name>
```

```
<param-value>Development</param-value>
```

```
</context-param>
```

```
<servlet>
```

```
<servlet-name>Faces Servlet</servlet-name>
```

```
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
```

```
<load-on-startup>1</load-on-startup>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
<servlet-name>Faces Servlet</servlet-name>
```

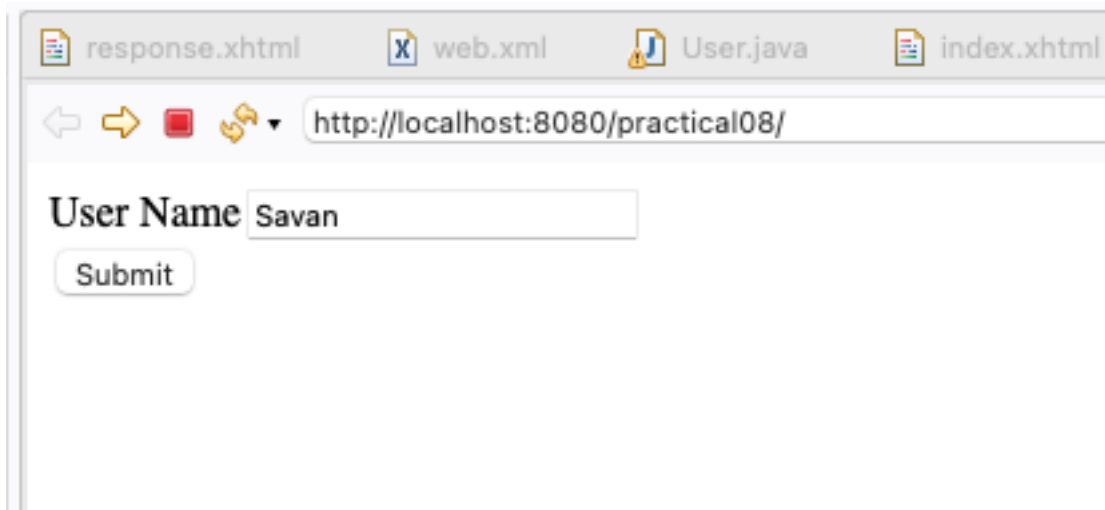
```
<url-pattern>/faces/*</url-pattern>
```

```
</servlet-mapping>
```

```
<session-config>
```

```
<session-timeout>  
30  
</session-timeout>  
</session-config>  
<welcome-file-list>  
<welcome-file>faces/index.xhtml</welcome-file>  
</welcome-file-list>  
</web-app>
```

Output:



Conclusion:

PRACTICAL SET – 9

It was started in 2001 by Gavin King as an alternative to EJB2 style entity bean. Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables. The main goal of hibernate is to relieve the developer from the common data persistence related tasks. It maps the objects in the java with the tables in the database very efficiently and also you can get maximum using its data query and retrieval facilities. Mainly by using Hibernate in your projects you can save incredible time and effort. Hibernate framework simplifies the development of java application to interact with the database. Hibernate is an open source, lightweight, ORM (Object Relational Mapping) tool.

Advantages of Hibernate

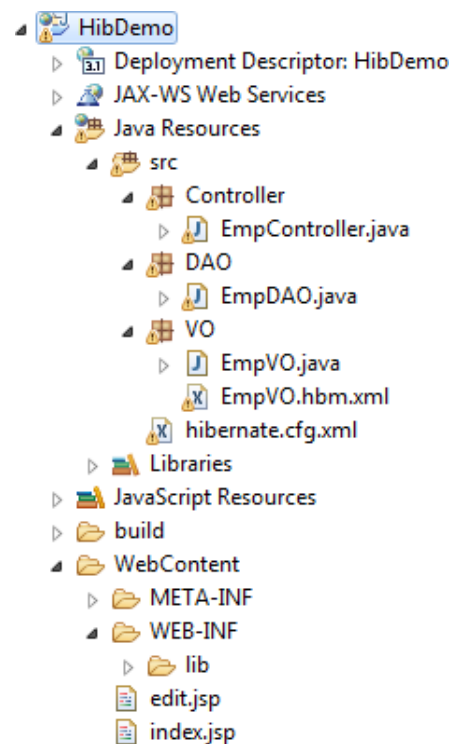
Opensource and Lightweight: Hibernate framework is opensource under the LGPL license and lightweight.

Fast performance: The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

Database Independent query: HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries.

Automatic table creation: Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

Simplifies complex join: To fetch data from multiple tables is easy in hibernate framework.



Directory Structure

References:

1. <https://www.javatpoint.com/jsp-implicit-objects>

2. Hibernate 2nd edition, Jeff Linwood and Dave Minter, Beginning Aprèspublication

Aim: Write Hibernate application to store customer records and retrieve the customer record including name, contact number, address.

Code:

Output:

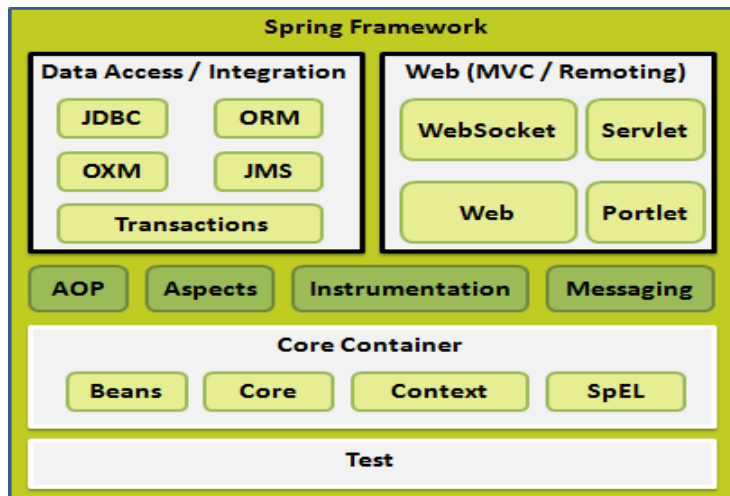
Conclusion:

PRACTICAL SET – 10

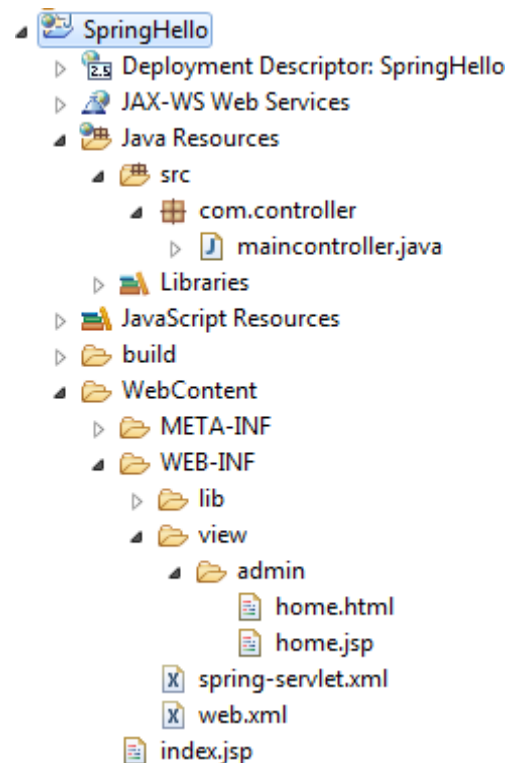
It was developed by Rod Johnson in 2003. Spring framework makes the easy development of JavaEE application. It is helpful for beginners and experienced persons. Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, EJB, JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems. Spring is lightweight when it comes to size and transparency. Spring framework targets to make J2EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

Advantages of Spring

- Predefined Templates, Loose Coupling, Easy to test, Lightweight, Fast Development, Powerful abstraction, Declarative support.



Spring Framework



Directory Structure

References:

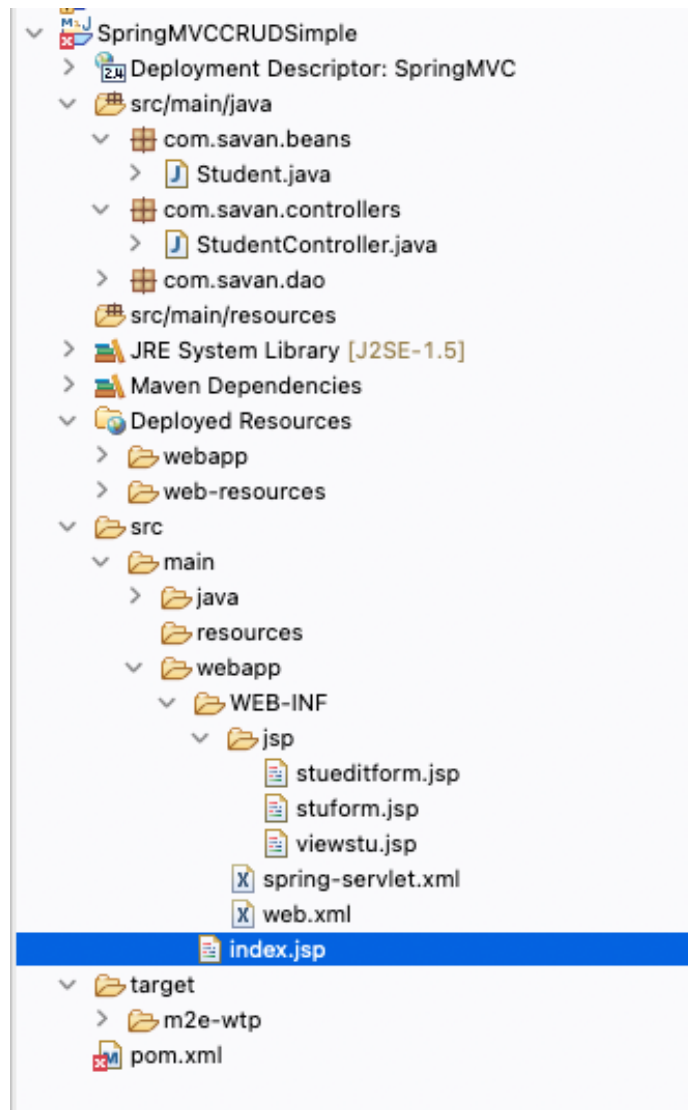
1. <https://www.javatpoint.com/jsp-implicit-objects>

2. Spring in Action 3rd edition , Craig walls, ManningPublication

Aim: Write an application to keep record and retrieve record of student. The record includes student id, enrollment number, semester, SPI. Use MVC architecture.

Code:

Folder structure =>



index.jsp =>

```
<a href="stuform">Add Student Data</a>
<a href="viewstu">View Student Data</a>
```

stuform.jsp =>

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

```
    <h1>Add New Student Data</h1>
    <form:form method="post" action="save">
        <table >
            <tr>
                <td>Enrollment : </td>
                <td><form:input path="enrollment" /></td>
            </tr>
            <tr>
                <td>Semester :</td>
                <td><form:input path="sem" /></td>
            </tr>
            <tr>
                <td>SPI :</td>
                <td><form:input path="spi" /></td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="Save" /></td>
            </tr>
        </table>
    </form:form>
```

Stueditform.jsp =>

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

```
    <h1>Edit Student details</h1>
```



```

<form:form method="POST" action="/SpringMVCCRUDSimple/editsave">
  <table >
    <tr>
      <td></td>
      <td><form:hidden path="id" /></td>
    </tr>
    <tr>
      <td><u>Enrollment</u> : </td>
      <td><form:input path="enrollment" /></td>
    </tr>
    <tr>
      <td><u>Semester</u> : </td>
      <td><form:input path="sem" /></td>
    </tr>
    <tr>
      <td><u>SPI</u> : </td>
      <td><form:input path="spi" /></td>
    </tr>
    <tr>
      <td></td>
      <td><input type="submit" value="Edit Save" /></td>
    </tr>
  </table>
</form:form>

```

view.jsp =>

```

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

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

<h1>Students List</h1>
<table border="2" width="70%" cellpadding="2">
  <tr><th>Id</th><th><u>Enrollment</u></th><th><u>Semester</u></th><th><u>SPI</u></th><th>Edit</th><th>Delete</th></tr>
  <c:forEach var="student" items="${list}">
    <tr>
      <td>${student.id}</td>
      <td>${student.enrollment}</td>
      <td>${student.sem}</td>

```

```

<td>${student.spi}</td>
<td><a href="editstu/${student.id}">Edit</a></td>
<td><a href="deletestu/${student.id}">Delete</a></td>
</tr>
</c:forEach>
</table>
<br/>
<a href="stuform">Add New Student</a>

```

spring-servlet.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.savan.controllers"></context:component-
scan>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/jsp/"></property>
<property name="suffix" value=".jsp"></property>
</bean>

<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
<property name="url" value="jdbc:mysql://localhost:3306/savan"></property>
<property name="username" value="root"></property>
<property name="password" value="<enter your password here>"></property>
</bean>

<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="ds"></property>
</bean>

```

```

<bean id="dao" class="com.savan.dao.StudentDao">
<property name="template" ref="jt"></property>
</bean>
</beans>

```

Web.xml =>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
JAVA.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>SpringMVC</display-name>
  <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>/</url-pattern>
  </servlet-mapping>
</web-app>

```

(com.savan.beans) > Student.java =>

```

package com.savan.beans;

public class Student {

private int id;

private String enrollment;

private int sem;

private float spi;


public int getId() {
    return id;
}

public void setId(int id) {

```

```

        this.id = id;
    }

    public String getEnrollment() {
        return enrollment;
    }

    public void setEnrollment(String enrollment) {
        this.enrollment = enrollment;
    }

    public int getSem() {
        return sem;
    }

    public void setSem(int sem) {
        this.sem = sem;
    }

    public float getSpi() {
        return spi;
    }

    public void setSpi(float spi) {
        this.spi = spi;
    }

}

```

(com.savan.controllers) > StudentControllers.java =>

package com.savan.controllers;

```

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

```

```

import com.savan.beans.Student;
import com.savan.dao.StudentDao;
@Controller
public class StudentController {
    @Autowired
    StudentDao dao;//will inject dao from xml file

    @RequestMapping("/stuform")
    public String showform(Model m){
        m.addAttribute("command", new Student());
        return "stuform";
    }

    @RequestMapping(value="/save",method = RequestMethod.POST)
    public String save(@ModelAttribute("student") Student student){
        dao.save(student);
        return "redirect:/viewstu";
    }

    /* It provides list of employees in model object */
    @RequestMapping("/viewstu")
    public String viewemp(Model m){
        List<Student> list=dao.getStudents();
        m.addAttribute("list",list);
        return "viewstu";
    }

    /* It displays object data into form for the given id.
    * The @PathVariable puts URL data into variable.*/
    @RequestMapping(value="/editstu/{id}")
    public String edit(@PathVariable int id, Model m){
        Student stu=dao.getStudentById(id);
        m.addAttribute("command",stu);
        return "stueditform";
    }

    /* It updates model object. */
    @RequestMapping(value="/editsave",method = RequestMethod.POST)
    public String editsave(@ModelAttribute("student") Student student){
        dao.update(student);
        return "redirect:/viewstu";
    }
}

```

```

    @RequestMapping(value="/deletestu/{id}",method = RequestMethod.GET)
    public String delete(@PathVariable int id){
        dao.delete(id);
        return "redirect:/viewstu";
    }
}

```

(com.savan.dao) > StudentDao.java =>

```

package com.savan.dao;

```

```

import java.sql.ResultSet;

```

```

import java.sql.SQLException;

```

```

import java.util.List;

```

```

import org.springframework.jdbc.core.BeanPropertyRowMapper;

```

```

import org.springframework.jdbc.core.JdbcTemplate;

```

```

import org.springframework.jdbc.core.RowMapper;

```

```

import com.savan.beans.Student;

```

```

public class StudentDao {

```

```

    JdbcTemplate template;

```

```

    public void setTemplate(JdbcTemplate template) {

```

```

        this.template = template;

```

```

    }

```

```

    public int save(Student p){

```

```

        String sql="insert into stu(enrollment,sem,spi) values('"+p.getEnrollment()+"','"+p.getSem()+"','"+p.getSpi()+"')";

```

```

        return template.update(sql);

```

```

    }

```

```

    public int update(Student p){

```

```

        String sql="update stu set enrollment='"+p.getEnrollment()+"', sem='"+p.getSem()+"',spi='"+p.getSpi()+"' where id='"+p.getId()+"'";

```

```

        return template.update(sql);

```

```

    }

```

```

    public int delete(int id){

```

```

String sql="delete from stu where id="+id+"";
return template.update(sql);
}

public Student getStudentById(int id){
    String sql="select * from stu where id=?";
    return template.queryForObject(sql, new Object[]{id},new BeanPropertyRowMapper<Student>(Student.class));
}

public List<Student> getStudents(){
    return template.query("select * from stu",new RowMapper<Student>(){
        public Student mapRow(ResultSet rs, int row) throws SQLException {
            Student e=new Student();
            e.setId(rs.getInt(1));
            e.setEnrollment(rs.getString(2));
            e.setSem(rs.getInt(3));
            e.setSpi(rs.getFloat(4));
            return e;
        }
    });
}
}

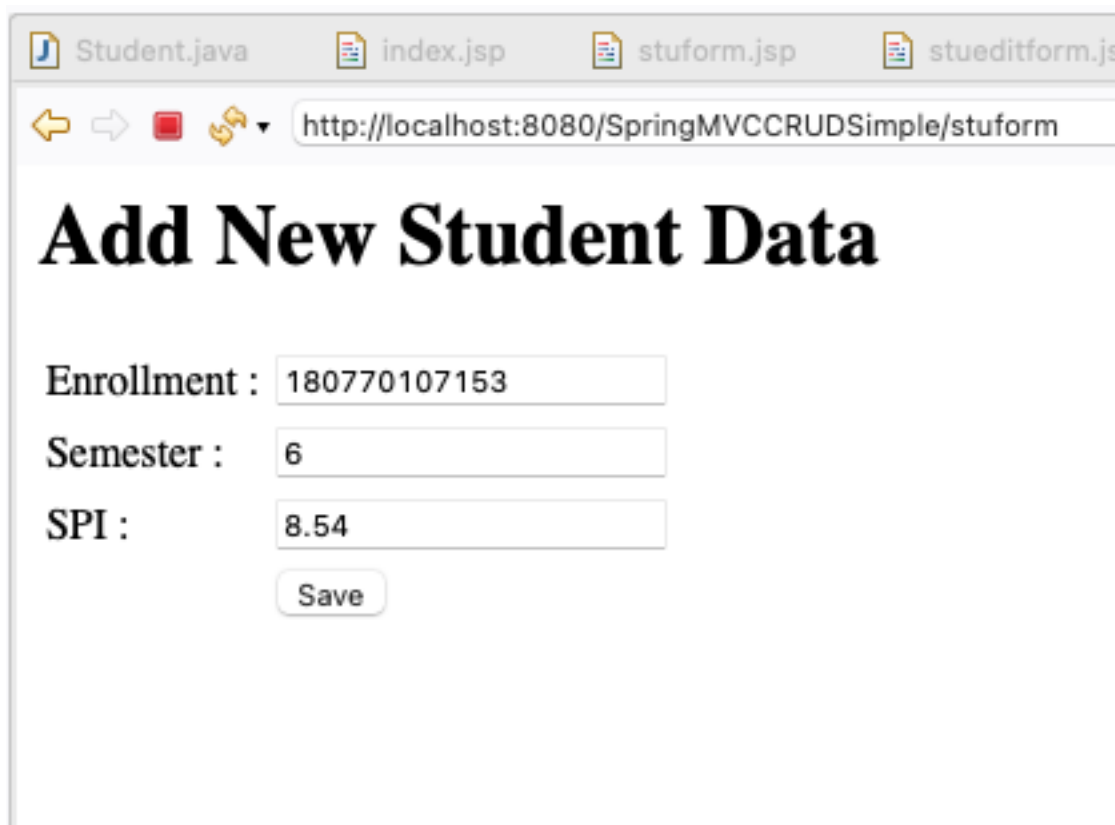
```

Output:

Home page =>

[Add Student Data](#) [View Student Data](#)

Adding new Student Data =>



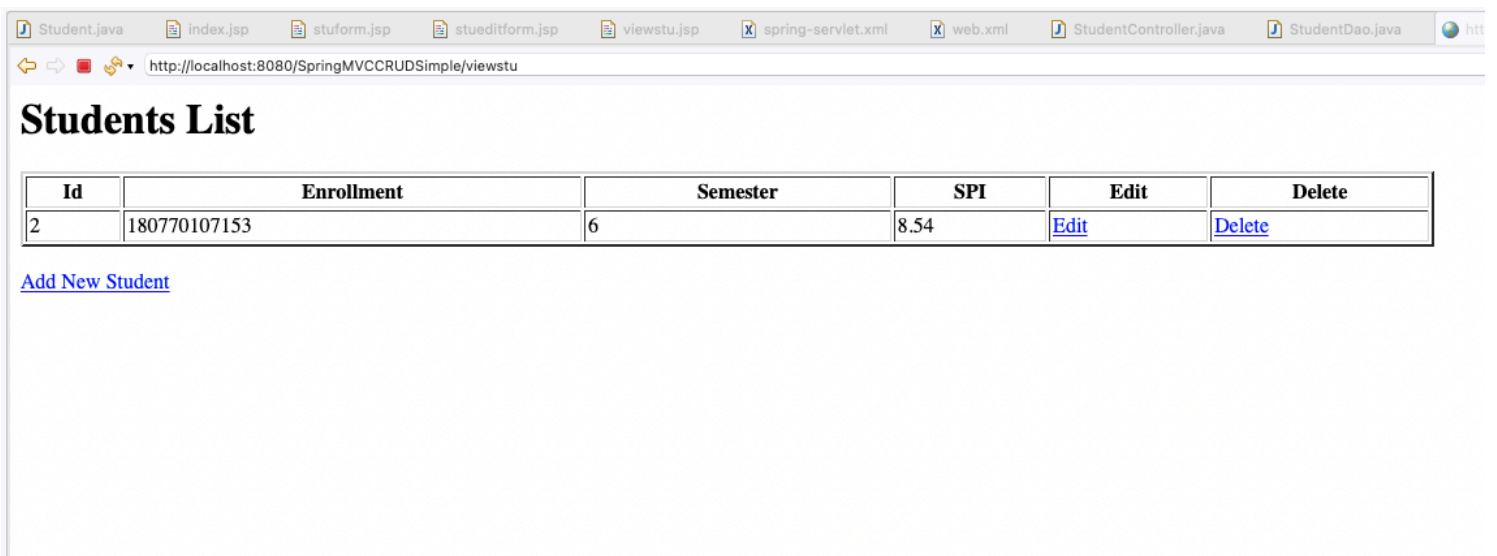
Add New Student Data

Enrollment :

Semester :

SPI :

Displaying all the data =>



Students List

Id	Enrollment	Semester	SPI	Edit	Delete
2	180770107153	6	8.54	Edit	Delete

[Add New Student](#)

Student data edit (semester from 6 to 8) =>

Edit Student details

Enrollment : 180770107153

Semester : 8

SPI : 8.54

Edit Save

Updated student data (see 6 -> 8) =>

Students List

Id	Enrollment	Semester	SPI	Edit	Delete
2	180770107153	8	8.54	Edit	Delete

[Add New Student](#)

StudentData deleted =>

Students List

Id	Enrollment	Semester	SPI	Edit	Delete
----	------------	----------	-----	------	--------

[Add New Student](#)

Conclusion: