

Practical 1 (a)

Aim: Create chat application using either TCP or UDP protocol.

USING TCP

****Server Side****

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

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

```
import java.util.Scanner;
```

```
public class Server
```

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

```
{        ServerSocketss=new ServerSocket(7888);
```

```
        Socket s=ss.accept();
```

```
        DataInputStream din=new DataInputStream(s.getInputStream());
```

```
        String str;
```

```
        str=din.readUTF();
```

```
        System.out.println("Client:\t"+str);
```

```
        DataOutputStreamdout=new DataOutputStream(s.getOutputStream());
```

```
        DataInputStreammsg=new DataInputStream(System.in);
```

```
        while(true)
```

```
{            str=din.readUTF();
```

```
            System.out.print("Client:\t"+str);
```

```
            System.out.print("Server:\t");
```

```
            str=msg.readLine();
```

```
            dout.writeUTF(str);
```

```
        }
```

```
    }
```

```
}
```

****Client Side ****

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

```
import java.net.Socket;
```

```
import java.util.Scanner;
```

```
public class Client
```

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

```
{        Socket s=new Socket("127.0.0.1",7888);
```

```
        if(s.isConnected())
```

```
        {            System.out.println("Connected to server");        }
```

```
        DataInputStreammsg=new DataInputStream(System.in);
```

```
        String str="Start Chat.....";
```

```
        DataOutputStreamdout=new DataOutputStream(s.getOutputStream());
```

```
        dout.writeUTF(str);
```

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

```
        DataInputStream din=new DataInputStream(s.getInputStream());
```

```
        while(true)
```

```
        {            System.out.print("Client:\t");
```

```
                str=msg.readLine();
```

```
                dout.writeUTF(str+"\n");
```

```
                str=din.readUTF();
```

```
                System.out.println("Server:\t"+str);
```

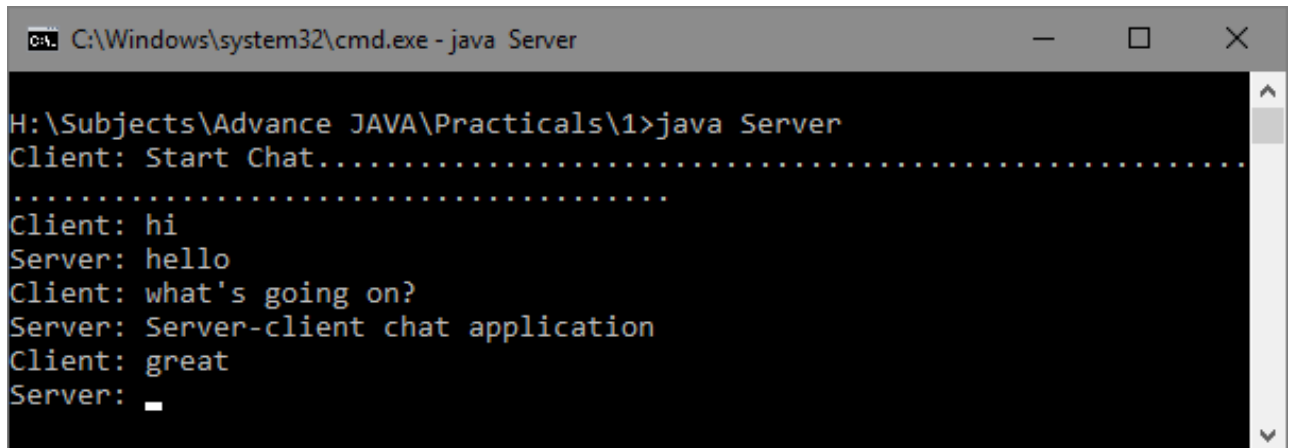
```
        }
```

```
    }
```

```
}
```

Output:

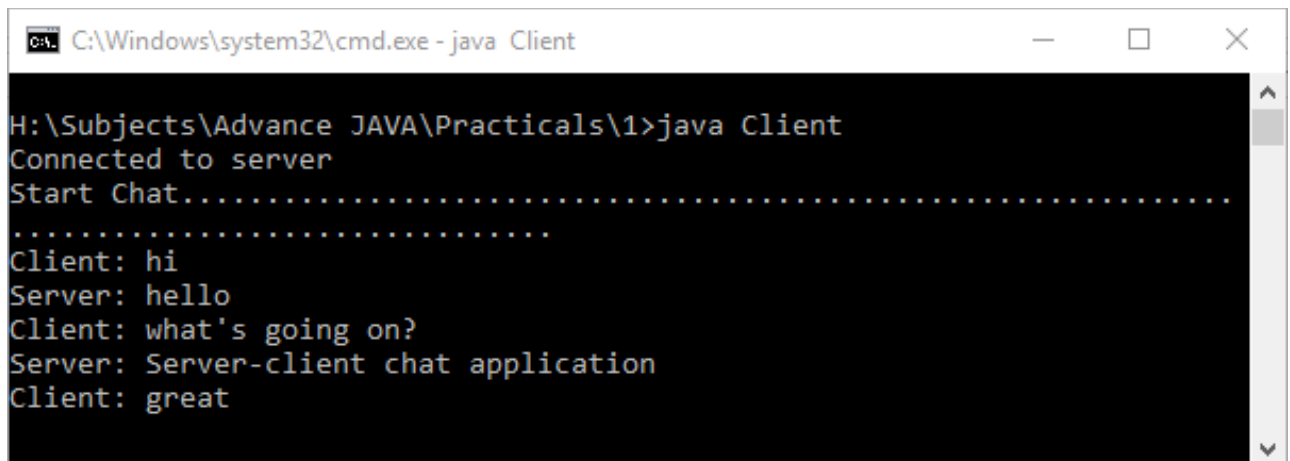
Server



```
C:\Windows\system32\cmd.exe - java Server

H:\Subjects\Advance JAVA\Practicals\1>java Server
Client: Start Chat.....
.....
Client: hi
Server: hello
Client: what's going on?
Server: Server-client chat application
Client: great
Server: _
```

Client



```
C:\Windows\system32\cmd.exe - java Client

H:\Subjects\Advance JAVA\Practicals\1>java Client
Connected to server
Start Chat.....
.....
Client: hi
Server: hello
Client: what's going on?
Server: Server-client chat application
Client: great
```

Practical 1 (b)

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

****Server Code****

```
import java.io.*;

import java.net.*;

class Server

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

    {
        ServerSocketss=new ServerSocket(7777);

        Socket s=ss.accept();

        System.out.println("connected.....");

        FileInputStream fin=new FileInputStream("Send.txt");

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

        int r;

        while((r=fin.read())!=-1)

        {
            dout.write(r);
        }

        System.out.println("\nFiletransfer Completed");

        s.close();

        ss.close();

    }
}
```

****Client Code****

```
import java.io.*;

import java.net.*;

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");
        }

        FileOutputStream fout= new FileOutputStream("received.txt");

        DataInputStream din=new DataInputStream(s.getInputStream());

        int r;

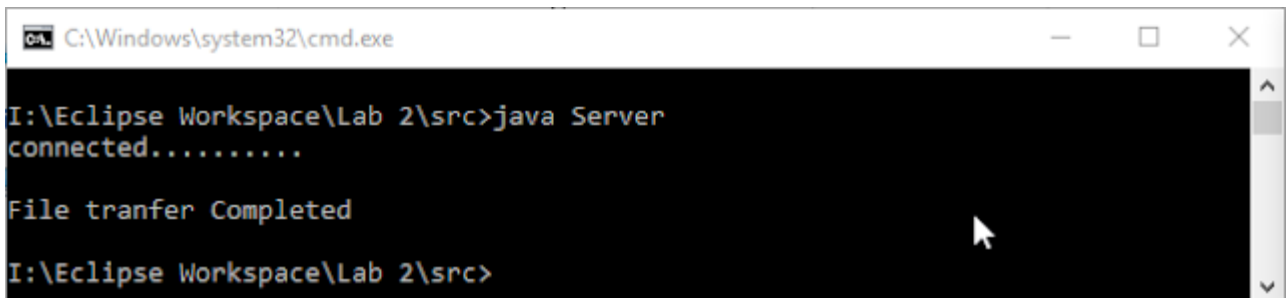
        while((r=din.read())!=-1)

        {
            fout.write((char)r);
        }

        s.close();
    }
}
```

Output:

Server



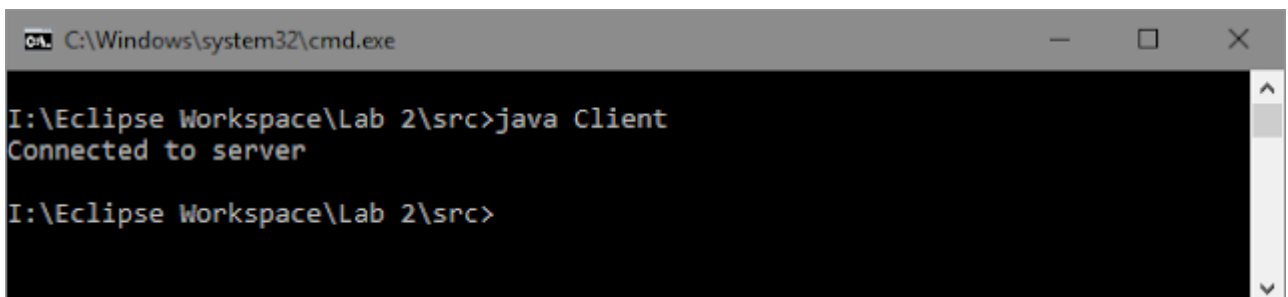
```
C:\Windows\system32\cmd.exe

I:\Eclipse Workspace\Lab 2\src>java Server
connected.....

File tranfer Completed

I:\Eclipse Workspace\Lab 2\src>
```

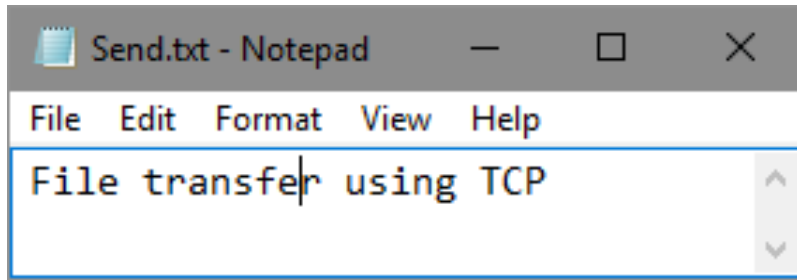
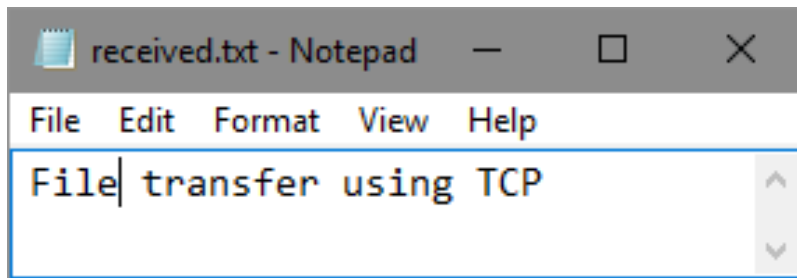
Client



```
C:\Windows\system32\cmd.exe

I:\Eclipse Workspace\Lab 2\src>java Client
Connected to server

I:\Eclipse Workspace\Lab 2\src>
```

Sent File*Received File*

Practical 2(a)

Aim: Implement any one sorting algorithm using UDP on Server application and Give Input On Client side and client should sorted output from server and display sorted on input side.

****Server Code****

```
import java.io.*;

import java.net.*;

import java.util.*;

class Server

{
    public static void main(String args[]) throws Exception
    {
        ServerSocketss=new ServerSocket(7777);

        Socket s=ss.accept();

        System.out.println("connected.....");

        DataInputStream din=new DataInputStream(s.getInputStream());

        DataOutputStreamdout=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 Code****

```
import java.io.*;

import java.net.*;

import java.util.Scanner;

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:");

        DataOutputStreamdout=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+" ");

        }

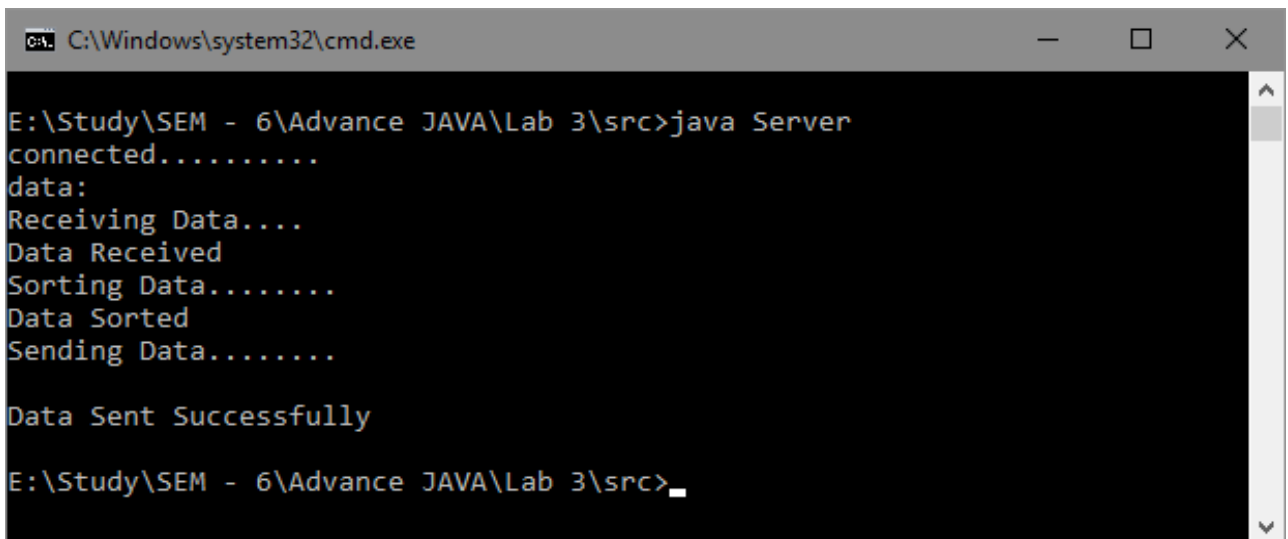
        s.close();

    }

}
```

Output:

Server

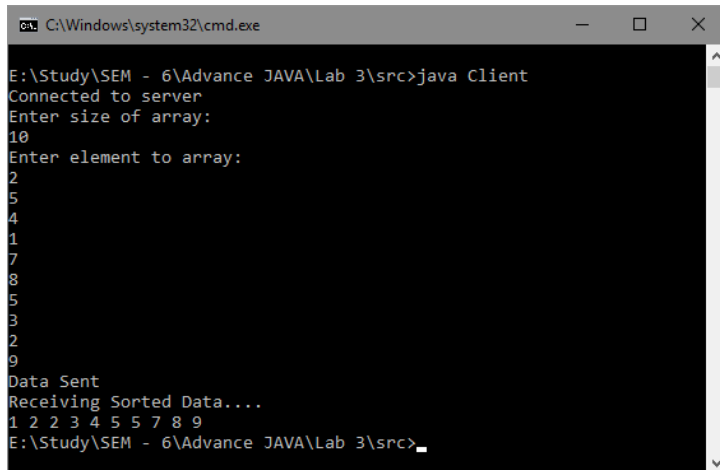


```
C:\Windows\system32\cmd.exe

E:\Study\SEM - 6\Advance JAVA\Lab 3\src>java Server
connected.....
data:
Receiving Data....
Data Received
Sorting Data.....
Data Sorted
Sending Data.....

Data Sent Successfully

E:\Study\SEM - 6\Advance JAVA\Lab 3\src>
```

Client

```
C:\Windows\system32\cmd.exe
E:\Study\SEM - 6\Advance JAVA\Lab 3\src>java Client
Connected to server
Enter size of array:
10
Enter element to array:
2
5
4
1
7
8
5
3
2
9
Data Sent
Receiving Sorted Data....
1 2 2 3 4 5 5 7 8 9
E:\Study\SEM - 6\Advance JAVA\Lab 3\src>
```

Practical 2(b)

Aim: Implement Concurrent 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.

****Server Code****

```
import java.net.*;

import java.io.*;

public class Server

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

    {
        int count=1;

        ServerSocket ss=new ServerSocket(7878);

        while(true)

        {
            new RevThread(ss.accept(),count).start();

            System.out.println(count+" client connected");

            count++;

        }

    }

}

class RevThread extends Thread

{
    Socket s=null;

    int n;

    public RevThread(Socket socket,int count)

    {
        s=socket;

        n=count;

    }

    public void run()

    {
        try

        {
            while(true)
```

```
        {
            System.out.println("receiving from client "+n);

            DataInputStream din=new DataInputStream(s.getInputStream());

            String str=din.readUTF();

            System.out.println("processing data of Client "+n);

            StringBuffer rev=new StringBuffer();

            rev=rev.append(str);

            rev=rev.reverse();

            String revStr=new String(rev);

            System.out.println("sending to client "+n);

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

            dout.writeUTF(revStr);

        }

    }

    catch(IOException e)

    {
        System.out.println(e);
    }

}
```

****Client Code****

```
import java.io.*;

import java.net.*;

public class Client

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

    {
        Socket s=new Socket("127.0.0.1",7878);

        if(s.isConnected())

        {
            System.out.println("Connected to Server....");
        }

        while(true)
```

```
        {
            System.out.println("Enter String to reverse:");

            DataInputStream in=new DataInputStream(System.in);

            String str=in.readLine();

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

            DataInputStream din=new DataInputStream(s.getInputStream());

            String rev=din.readUTF();

            System.out.println("Reversed String:\t"+rev);

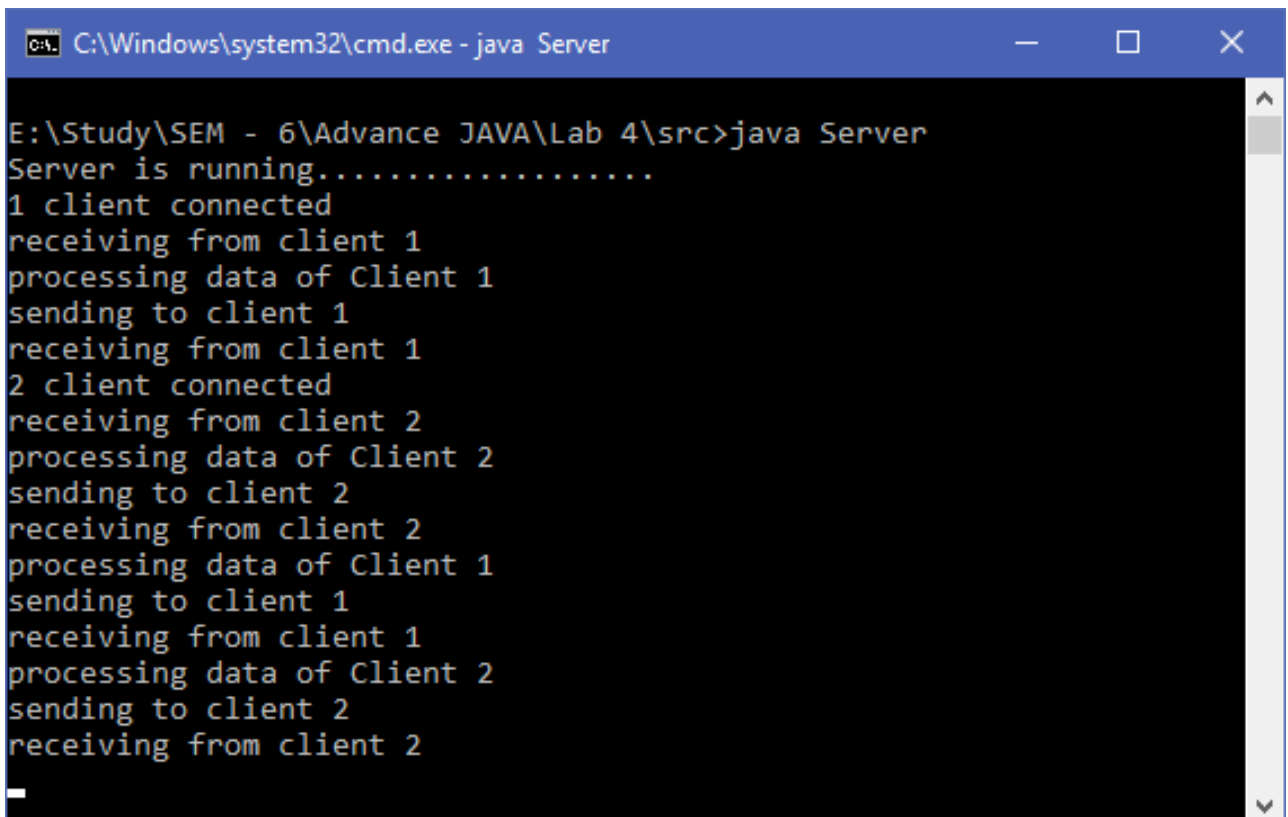
        }

    }

}
```

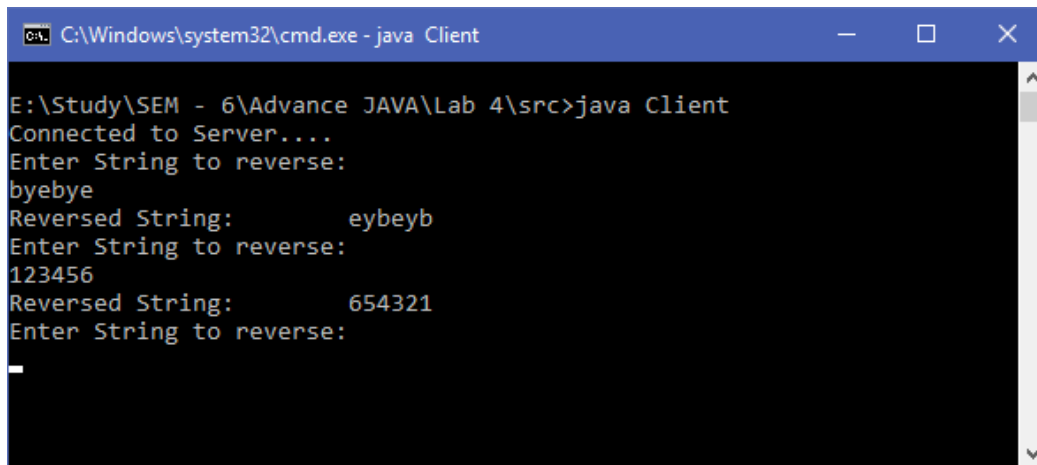
Output:

Server



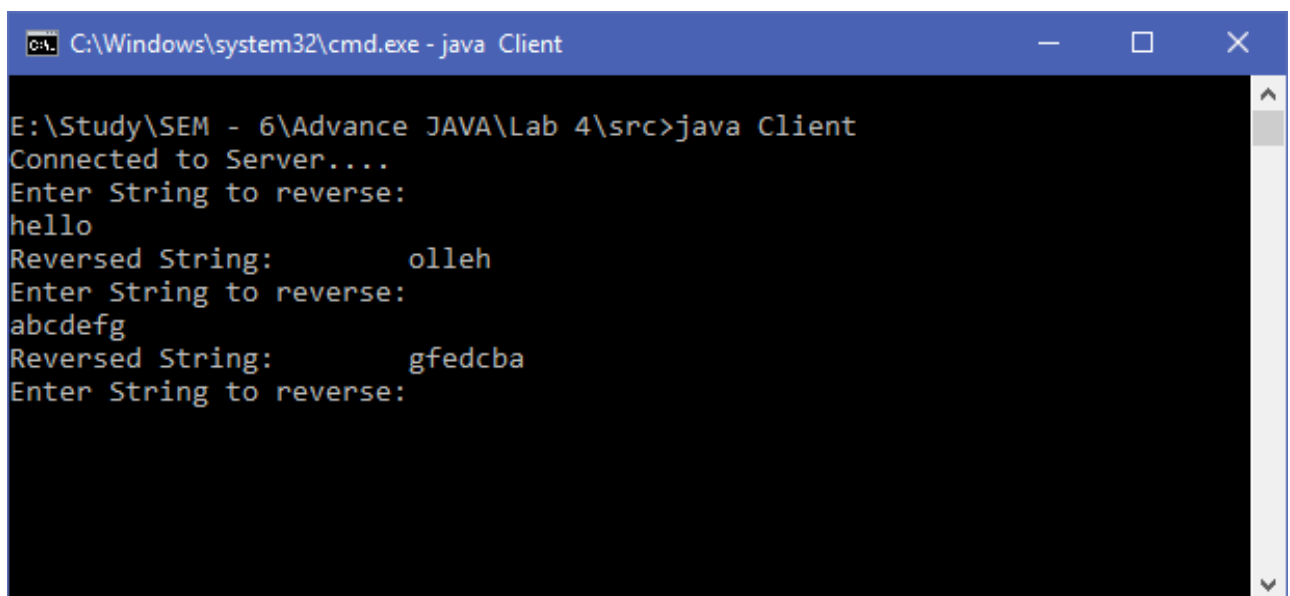
```
C:\Windows\system32\cmd.exe - java Server

E:\Study\SEM - 6\Advance JAVA\Lab 4\src>java Server
Server is running.....
1 client connected
receiving from client 1
processing data of Client 1
sending to client 1
receiving from client 1
2 client connected
receiving from client 2
processing data of Client 2
sending to client 2
receiving from client 2
processing data of Client 1
sending to client 1
receiving from client 1
processing data of Client 2
sending to client 2
receiving from client 2
```

Client 1

```
C:\Windows\system32\cmd.exe - java Client

E:\Study\SEM - 6\Advance JAVA\Lab 4\src>java Client
Connected to Server....
Enter String to reverse:
byebye
Reversed String:      eybeyb
Enter String to reverse:
123456
Reversed String:      654321
Enter String to reverse:
_
```

Client 2

```
C:\Windows\system32\cmd.exe - java Client

E:\Study\SEM - 6\Advance JAVA\Lab 4\src>java Client
Connected to Server....
Enter String to reverse:
hello
Reversed String:      olleh
Enter String to reverse:
abcdefg
Reversed String:      gfedcba
Enter String to reverse:
```

Practical 3(a)

Aim: Implement Student information system using JDBC

****StudentI.java Code****

```
import java.rmi.Remote;

import java.sql.*;

import java.util.*;

public interface StudentI extends Remote

{
    public abstract ArrayList insert(int id,String name,String branch,int atd) throws Exception;
}
```

****StudentC.java Code****

```
import java.sql.*;

import java.rmi.*;

import java.rmi.server.*;

import java.util.*;

public class StudentC extends UnicastRemoteObject implements StudentI

{
    public StudentC() throws Exception
    {
        super();
    }

    public ArrayList insert(int id,String name,String branch,int atd) throws Exception
    {
        ArrayList ar=new ArrayList();

        Class.forName("com.mysql.jdbc.Driver");

        Connection con =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql","root","birju");

        Statement stmt=con.createStatement();

        stmt.executeUpdate("insert into student values("+id+", '"+name+"', '"+branch+"', '"+atd+"')");
        ResultSet rs=stmt.executeQuery("select * from student where id="+id);

        rs.next();

        int Id=rs.getInt(1);
```

```
        name=rs.getString(2);

        branch=rs.getString(3);

        atd=rs.getInt(4);

        ar.add(new Integer(Id));

        ar.add(name);

        ar.add(branch);

        ar.add(new Integer(atd));

        con.close();

        return ar;

    }

}
```

****Server.java Code****

```
import java.rmi.*;

public class Server

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

    {
        StudentI obj=new StudentC();

        Naming.rebind("stinfo",obj);

        System.out.println("Server Started");

    }

}
```

****Client.java Code****

```
import java.rmi.*;

import java.sql.*;

import java.util.*;

public class Client

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



```
{    Scanner scan= new Scanner(System.in);

    System.out.println("Enter id,name,branch,attendance for student:");

    int id=scan.nextInt();

    String name=scan.next();

    String branch=scan.next();

    int atd=scan.nextInt();

    StudentI obj=(StudentI)Naming.lookup("stinfo");

    ArrayList ar=obj.insert(id,name,branch,atd);

    Iterator it=ar.iterator();

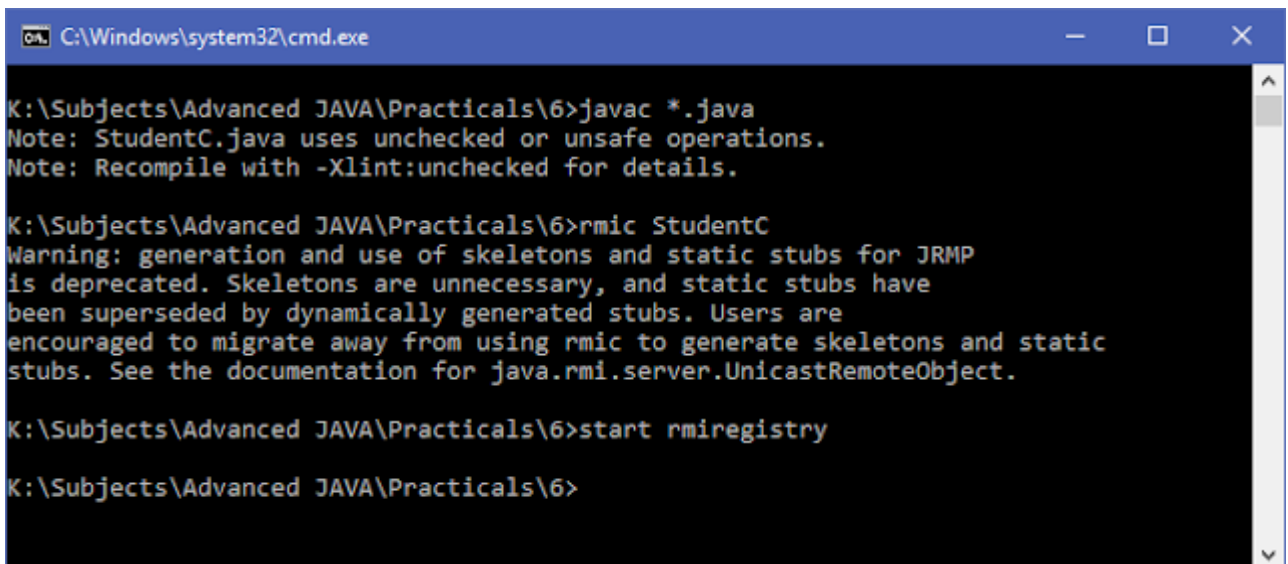
    System.out.println("Id\tName\tBranch\tAttendance");

    System.out.println(it.next()+"\t"+it.next()+"\t"+it.next()+"\t"+it.next()+"\t");

}
}
```

Output:

Commands



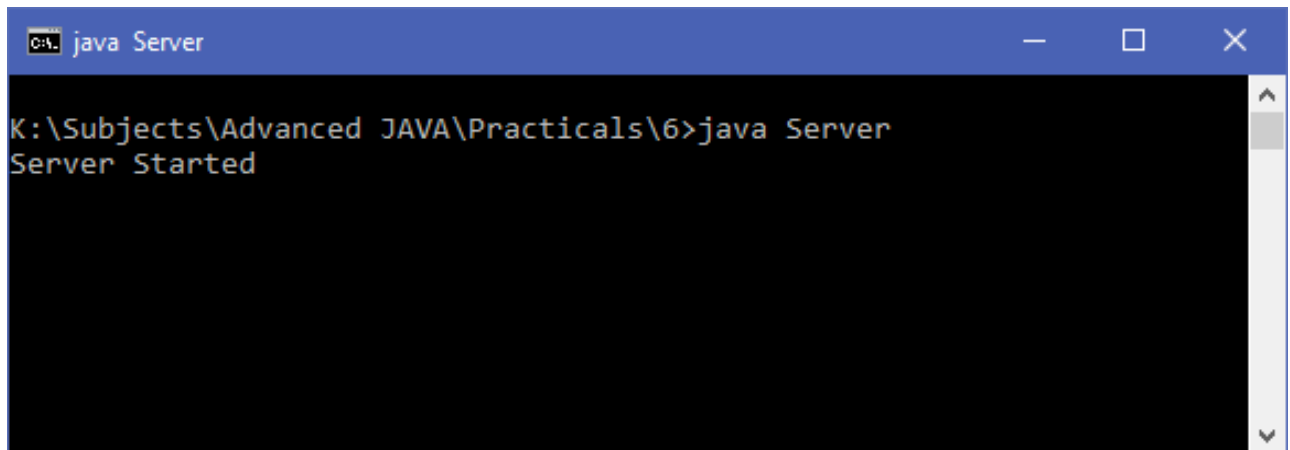
```
C:\Windows\system32\cmd.exe

K:\Subjects\Advanced JAVA\Practicals\6>javac *.java
Note: StudentC.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

K:\Subjects\Advanced JAVA\Practicals\6>rmic StudentC
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

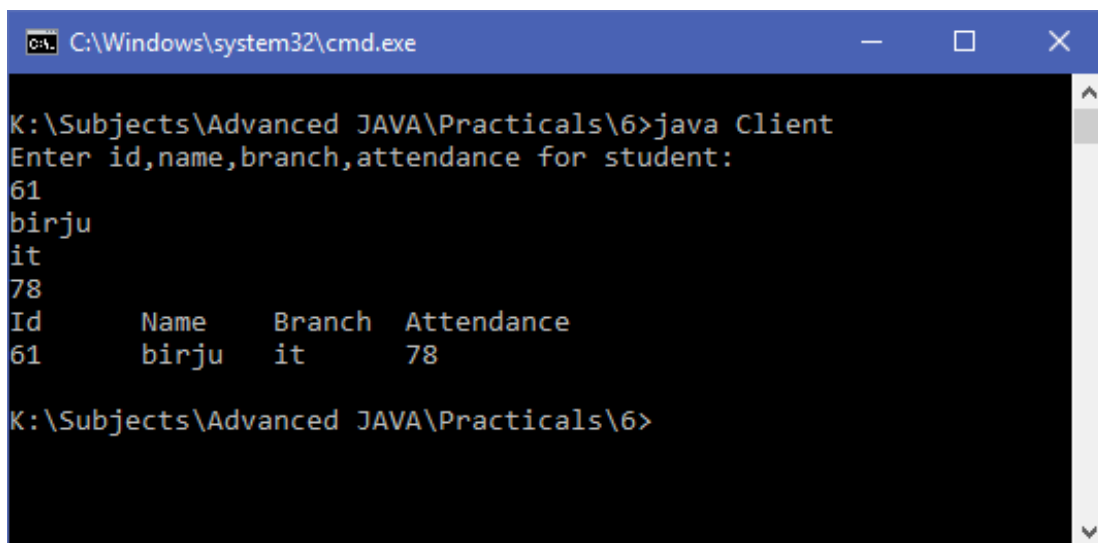
K:\Subjects\Advanced JAVA\Practicals\6>start rmiregistry

K:\Subjects\Advanced JAVA\Practicals\6>
```

ServerA screenshot of a Windows command prompt window titled "java Server". The window has a blue title bar with standard minimize, maximize, and close buttons. The command prompt shows the directory "K:\Subjects\Advanced JAVA\Practicals\6" and the command "java Server" being executed. The output is "Server Started".

```
C:\> java Server

K:\Subjects\Advanced JAVA\Practicals\6>java Server
Server Started
```

ClientA screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a blue title bar with standard minimize, maximize, and close buttons. The command prompt shows the directory "K:\Subjects\Advanced JAVA\Practicals\6" and the command "java Client" being executed. The output prompts for student information, which is entered as "61", "birju", "it", and "78". The output then displays a table with the entered data.

```
C:\Windows\system32\cmd.exe

K:\Subjects\Advanced JAVA\Practicals\6>java Client
Enter id,name,branch,attendance for student:
61
birju
it
78
Id      Name    Branch  Attendance
61      birju   it      78

K:\Subjects\Advanced JAVA\Practicals\6>
```

Practical 3 (b)

Aim: User can create a new database and also create new table under that database. Once database has been created then user can perform database operation by calling above functions. Use following Java Statement interface to implement program: 1. Statement 2. Prepared statement 3. Callable statement

****SqlCon.java Code****

```
import java.sql.*;

import java.util.Scanner;

public class SqlCon
{
    public static void main(String[] args) throws Exception
    {
        System.out.println("Enter database name to create new database:");

        Scanner scan=new Scanner(System.in);

        String dbname=scan.next();

        Class.forName("com.mysql.jdbc.Driver");

        Connection con =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/", "root", "birju");

        Statement stmt=con.createStatement();

        stmt.executeUpdate("create database "+dbname);

        System.out.println("Database Created");

        stmt.execute("use "+dbname);

        System.out.println("Enter table name to create table:");

        String tbname=scan.next();

        stmt.executeUpdate("create table "+tbname+" (id int,name char(20),branch
        char(10),address char(20))");

        System.out.println("Table Created with (id,nam,branch,address) fields");

        while(true)
        {
            System.out.println("Enter your
            choice:\n1.Insert\t2.Delete\t3.select\t4.Exit");

            int choice=scan.nextInt();
```

```
switch(choice)
{
    case 1:
        {
            System.out.println("Enter (id,name,branch,address) of
student:");

            int id=scan.nextInt();

            String name=scan.next();

            String branch=scan.next();

            String add=scan.next();

            PreparedStatement sr;

            String str = "insert into "+tbname+" values(?,?,?,?)";

            sr = con.prepareStatement(str);

            sr.setInt(1,id);

            sr.setString(2,name);

            sr.setString(3,branch);

            sr.setString(4,add);

            sr.executeUpdate();

            System.out.println("Data inserted");

            break;
        }

    case 2:
        {
            System.out.println("Enter id of student:");

            int id=scan.nextInt();

            ResultSet rs=stmt.executeQuery("select * from "+tbname+"
where id="+id);

            if(rs.next())
            {
                stmt.executeUpdate("delete from "+tbname+"
where id="+id);

                System.out.println("Data deleted");
            }
        }
}
```

```
        else

            {          System.out.println("Data with id "+id+" doesn't
exist");}

            break;

    }

case 3:

{          System.out.println("Enter id of student:");

            int id=scan.nextInt();

            stmt.execute("CREATE PROCEDURE "+dbname+".selector
(IN idr INT) " + "BEGIN " + "SELECT * " + "FROM "+tbname+" "+
"WHERE id = idr; "+ "END");

            CallableStatement cs = con.prepareCall("{call " +
dbname+".selector(?)}");

            cs.setInt(1, id);

            System.out.println("procedure call sucess");

            cs.execute();

            ResultSet rs=cs.getResultSet();

            if(rs.next())

            {          System.out.println("id\tName\tBranch\tAddress");

                        System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+
"\t"+rs.getString(3)+"\t"+rs.getString(4));

            }

            break;

    }

case 4:

{          break;  }

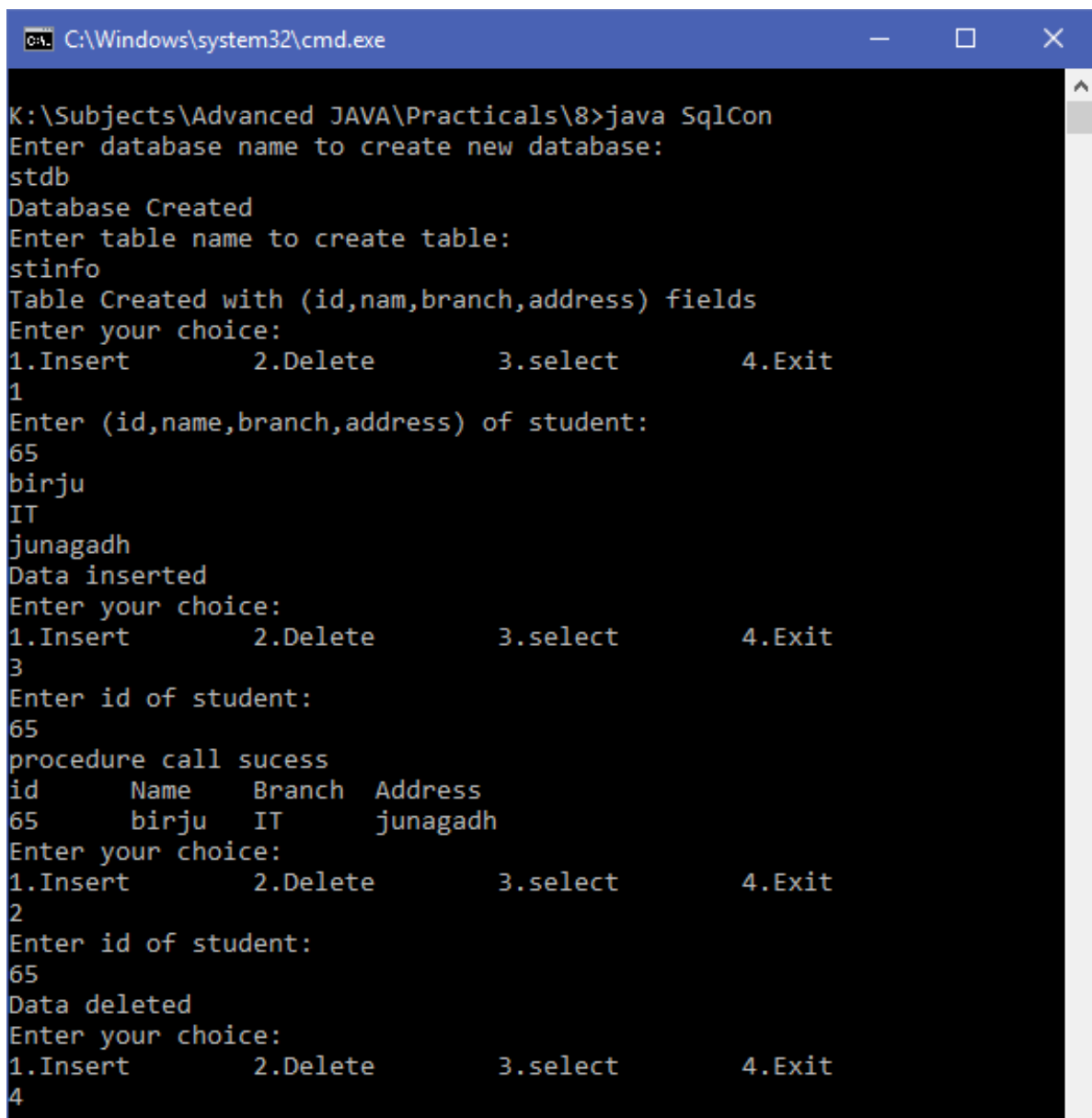
default:

{          System.out.println("Invalid option");    }

}
```

```
        if(choice==4)
        {
            break; }
    }
    con.close();
}
```

Output:



```
C:\Windows\system32\cmd.exe

K:\Subjects\Advanced JAVA\Practicals\8>java SqlCon
Enter database name to create new database:
stdb
Database Created
Enter table name to create table:
stinfo
Table Created with (id,nam,branch,address) fields
Enter your choice:
1.Insert      2.Delete      3.select      4.Exit
1
Enter (id,name,branch,address) of student:
65
birju
IT
junagadh
Data inserted
Enter your choice:
1.Insert      2.Delete      3.select      4.Exit
3
Enter id of student:
65
procedure call sucess
id      Name      Branch  Address
65      birju    IT      junagadh
Enter your choice:
1.Insert      2.Delete      3.select      4.Exit
2
Enter id of student:
65
Data deleted
Enter your choice:
1.Insert      2.Delete      3.select      4.Exit
4
```

Practical 4

Aim: Create Servlet file and study web descriptor file.

Java web applications use a deployment descriptor file to determine how URLs map to servlets, which URLs require authentication, and other information. This file is named web.xml and resides in the app's WAR under the WEB-INF/ directory. web.xml is part of the servlet standard for web applications.

A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

The deployment descriptor is a file named web.xml. It resides in the app's WAR under the WEB-INF/ directory. The file is an XML file whose root element is <web-app>.

web.xml File :-

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

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

  <servlet>

    <servlet-name>WebServlet</servlet-name>

    <servlet-class>WebServlet</servlet-class>

  </servlet>

  <welcome-file-list>

    <welcome-file>MainPage.html</welcome-file>

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

  </welcome-file-list>

  <servlet-mapping>

    <servlet-name>WebServlet</servlet-name>

    <url-pattern>/WebServlet</url-pattern>

  </servlet-mapping>

</web-app>
```

Here, above example of practical 5 it show from where the server url request redirect. here the Servlet class and name is.WebServlet and <url-pattern> tag map the /Servlet URL in Servlet and fetch

result about URL. <url-pattern> parent tag is <servlet-mapping> and always start with <web-app> and end with</web-app>.

Practical 5

Aim: Create Servlet file which contains following functions: 1. Connect 2. Create Database 3. Create Table 4. Insert Records into respective table 5. Update records of table of database 6. Delete Records from table. 7. Delete table and database.

```
**index.html**
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Java</title>
```

```
<link rel="stylesheet" type="text/css" href="main.css">
```

```
</head>
```

```
<body bgcolor="#222222">
```

```
<h1 align="center">Welcome To Student Portal</h1>
```

```
<div class="main">
```

```
<input type="button" class="button" name="insert_user"
```

```
onclick="window.location.href='page.html'" value="Insert Student"><br>
```

```
<input type="button" class="button green" name="delete_user"
```

```
onclick="window.location.href='delete.html'" value="Delete Student"> <br>
```

```
<input type="button" class="button grey" name="update_user"
```

```
onclick="window.location.href='update.html'" value="Update Student"> <br>
```

```
<input type="button" class="button blue"
```

```
name="select_user"onclick="window.location.href='select.html'" value="Select Student">
```

```
</div>
```

```
</body>
```

```
</html>
```

```
**delete.html Code**
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>

<title>Servlet</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" type="text/css" href="main.css">

</head>

<body bgcolor="#222222">

<form action="WebServlet" method="post" style=" text-align:center;">

<h2 style="color:white;font-family: Times; ">Student Information</h2>

<input class="side" type="text" pattern="[0-9]*" required="" name="s_enrollno" minlength="12"
maxlength="12" placeholder="Enrollment No" style=" width: 370px;"> <br>

<input class="b1" type="submit" value="Delete Data" name="s1">

</form>

</body>

</html>
```

****page.html Code:****

```
<!DOCTYPE html>

<html>

<head>

<title>Servlet</title>

<meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="main.css">

</head>

<body bgcolor="#222222">

<form action="WebServlet" method="post" style="text-align:center;">

<h1 style="color:white;font-family: Times; ">Student Information</h1>
```

```
<input class="side" type="text" pattern="[0-9]*" required="" name="s_enrollno"
placeholder="Enrollment No" minlength="12" maxlength="12" maxlength="12"> <br>

<input class="side a" type="text" pattern="[A-Za-z]+" required="" name="s_firstname"
placeholder="First Name">

<input class="side b" type="text" pattern="[A-Za-z]+" name="s_lastname" placeholder="Last Name"
required> <br>

<input class="side" type="text" pattern="[A-Za-z]+" name="s_branch" placeholder="Branch"
required> <br>

<input class="side" type="text" pattern="[0-9]*" required="" name="s_mobilenno"
placeholder="Mobile No" minlength="10" maxlength="10"> <br>

<input class="b1" type="submit" value="Insert Data" name="s1" >

</form>

</body>

</html>
```

****select.html Code****

```
<!DOCTYPE html>

<html>

<head>

<title>Servlet</title>

<meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" type="text/css" href="main.css">

</head>

<body bgcolor="#222222">

<form action="WebServlet" method="post" style="text-align:center;">

    <h1 style="color:white;font-family: Times; ">Student Information</h1>

    <input class="side" type="text" pattern="[0-9]*" required="" name="s_enrollno"
placeholder="Enrollment No" minlength="12" maxlength="12" style=" width: 370px;"> <br>

    <input class="b1" type="submit" value="View Data" name="s1">
```

```
</form>
```

```
</body>
```

```
</html>
```

```
**update.html Code**
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Servlet</title>
```

```
<meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <link rel="stylesheet" type="text/css" href="main.css">
```

```
</head>
```

```
<body bgcolor="#222222">
```

```
<form action="WebServlet" action="newupdate.html" method="post" style="text-align:center;">
```

```
    <h1 style="color:white;font-family: Times; ">Student Information</h1>
```

```
    <h2 style="color:white;">Insert Enrollment No To Change Details Of Student</h2>
```

```
    <input class="side" type="text" pattern="[0-9]*" required="" name="s_enrollno"
placeholder="Enrollment No" minlength="12" maxlength="12"><br>
```

```
    <h2 style="color:white;">Insert New Data</h2>
```

```
    <input class="side a" type="text" pattern="[A-Za-z]+" required="" name="s_firstname"
placeholder="First Name">
```

```
    <input class="side b" type="text" pattern="[A-Za-z]+" name="s_lastname" placeholder="Last
Name" required> <br>
```

```
    <input class="side" type="text" pattern="[A-Za-z]+" name="s_branch" placeholder="Branch"
required> <br>
```

```
    <input class="side" type="text" pattern="[0-9]*" required="" name="s_mobilenno"
placeholder="Mobile No" minlength="10" maxlength="10" > <br>
```

```
    <input class="b1" type="submit" value="Update Data" name="s1" >
```

```
</form>
```

```
</body>
```

```
</html>
```

****main.css Code****

```
body
```

```
{  
    margin:0px;  
    padding:0px;  
}
```

```
.main
```

```
{  
    text-align: center;  
}
```

```
.button
```

```
{  
    background-color: #f44336; /* Green */  
    color: white;  
    padding: 15px 32px;  
    text-align: center;  
    text-decoration: none;  
    font-size: 30px;  
    margin: 8px 2px;  
    cursor: pointer;  
    width: 400px;  
    border: 0px solid black;  
    transition: width 0.5s,background-color 0.5s;  
}
```

```
.button:hover
```

```
{  
    width: 99%;  
    background-color: white;  
    color: black;  
}
```

```
input
{
    margin: 50px; }

h1
{
    color:white; }

.green
{
    background-color: #4CAF50; }

.grey
{
    background-color: #555555; }

.blue
{
    background-color: #008CBA; }

.b1
{
    width: 395px;
    height: 40px;
    margin:10px;
    color: white;
    background-color: #008CBA;
    border: 0px solid grey;
    cursor: pointer;
}

.side
{
    padding: 10px;
    margin: 10px;
    width: 370px;
}

.a
{
    width: 160px; }

.b
```

```
{      width: 160px  }
```

****WebServlet.java Code****

```
import java.io.*;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import javax.servlet.*;

import javax.servlet.http.*;

public class WebServlet extends HttpServlet

{      @Override

        protected void doPost(HttpServletRequest request, HttpServletResponse response)throws

ServletException, IOException

        {      response.setContentType("text/html;charset=UTF-8");

                PrintWriter out = response.getWriter();

                try

                {      Class.forName("com.mysql.jdbc.Driver");

                        System.out.println("loaded driver succesfull");

                        Connection cn =

DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql","root","birju");

                        Statement st=cn.createStatement();

                        st.executeUpdate("create database student");

                        st.executeUpdate("use student");

                        st.execute("create table student2 (s_enrollno BIGINT,s_firstname

varchar(20),s_lastname varchar(20),s_branch varchar(10),s_mobileno BIGINT)");

                        ResultSet rs ;
```

```
String s2 = request.getParameter("s1");

Long eno = Long.valueOf(request.getParameter("s_enrollno"));

String fname = request.getParameter("s_firstname");

String lname = request.getParameter("s_lastname");

String branch = request.getParameter("s_branch");

String mobile = request.getParameter("s_mobilenno");

PreparedStatement sr;

Statement stmt = cn.createStatement();

ResultSet res;

// For Insert Data.....

switch (s2)

{
    //For Delete Data.....

    case "Insert Data":

        String str = "insert into student2 values(?,?,?,?)";

        sr = cn.prepareStatement(str);

        sr.setLong(1, eno);

        sr.setString(2, fname);

        sr.setString(3, lname);

        sr.setString(4, branch);

        sr.setString(5, mobile);

        int i = sr.executeUpdate();

        out.println("Student Data inserted");

        cn.close();

        break;

    case "Delete Data":

        String str1 = "select s_enrollno from student2 where s_enrollno = "+eno+" ";

        rs = stmt.executeQuery(str1);
```



```
        if(rs.next())

        {
            str1 = "delete from student2 where s_enrollno = ?";

            sr = cn.prepareStatement(str1);

            sr.setLong(1,eno);

            sr.executeUpdate();

            out.println("Student Data Deleted");

        }

        else

        {
            out.println("Student Data Not Valid");
            cn.close();

        }

        break;

        case "Update Data":

            String str3 = "select * from student2 where s_enrollno = "+eno+"";

            rs =stmt.executeQuery(str3);

            if(rs.next())

            {
                str3= "update student2 set s_enrollno=?,
s_firstname=?,s_lastname=?,s_branch=?,s_mobilen=? where
s_enrollno = "+eno+"";

                sr=cn.prepareStatement(str3);

                sr.setLong(1,eno);

                sr.setString(2,fname);

                sr.setString(3,lname);

                sr.setString(4,branch);

                sr.setString(5,mobile);

                sr.executeUpdate();

                out.println("Update Student2 Data Succesfull");

                cn.close();

            }

        }
```

```
        else

            {        out.println("Student Data Not Valid");    }

break;

case "View Data":

    res = stmt.executeQuery("select * from student2 where
s_enrollno="+eno+"");

    if(res.next())

    {        Long enu= res.getLong("s_enrollno");

            String fname1= res.getString("s_firstname");

            String lname1= res.getString("s_lastname");

            String branch1= res.getString("s_branch");

            String mb1= res.getString("s_mobileno");

            out.println("<html><body>");

            out.println("Enrollment No:- ");

            out.println(enu);out.println("<br>");

            out.println("First Name:- ");

            out.println(fname1);out.println("<br>");

            out.println("Last Name:- ");

            out.println(lname1);out.println("<br>");

            out.println("Branch:- ");

            out.println(branch1); out.println("<br>");

            out.println("Mobile No: ");

            out.println(mb1);out.println("<br>");

            out.println("</html></body>");

        }

    else

        {        out.println("Student Data is Not Valid");    }

break;
```

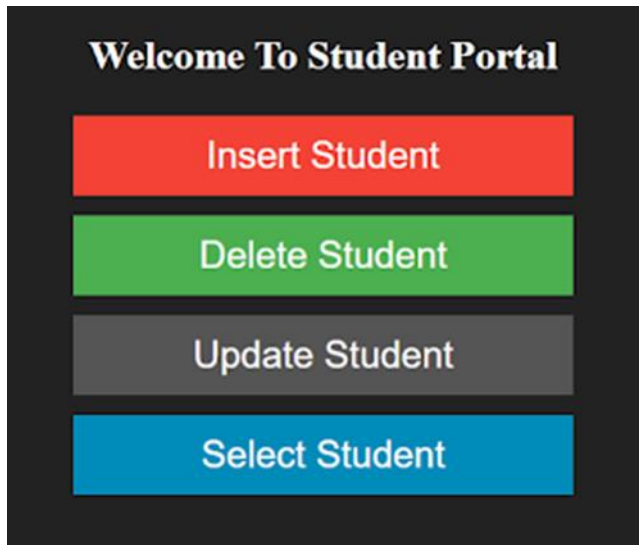
```
        }  
    }  
    catch(ClassNotFoundException | SQLException e  
    {  
        System.out.println(e);  
    }  
}
```

****web.xml Code****

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee">  
  
    <servlet>  
        <servlet-name>WebServlet</servlet-name>  
        <servlet-class>WebServlet</servlet-class>  
    </servlet>  
  
    <welcome-file-list>  
        <welcome-file>MainPage.html</welcome-file>  
        <welcome-file>index.html</welcome-file>  
    </welcome-file-list>  
  
    <servlet-mapping>  
        <servlet-name>WebServlet</servlet-name>  
        <url-pattern>/WebServlet</url-pattern>  
    </servlet-mapping>  
  
</web-app>
```

Output:

index.html



Welcome To Student Portal

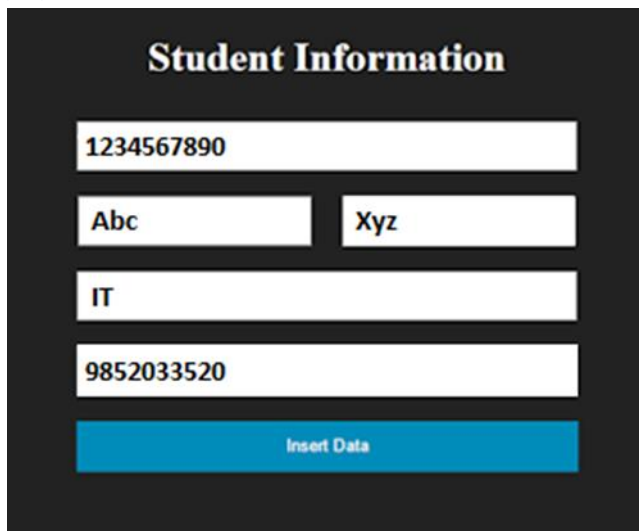
Insert Student

Delete Student

Update Student

Select Student

page.html



Student Information

1234567890

Abc Xyz

IT

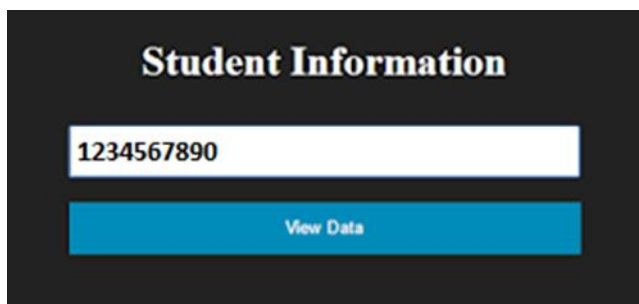
9852033520

Insert Data

Result

Student Data inserted

select.html



Student Information

1234567890

View Data

Result

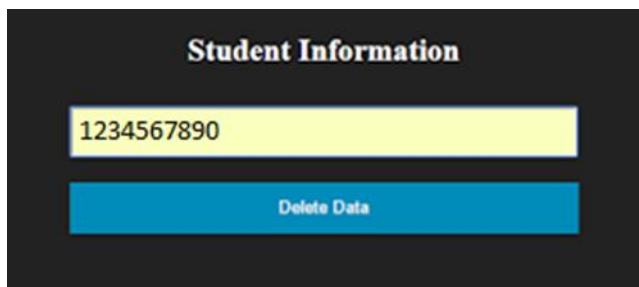
Enrollment No:- 1234567890

First Name:- Abc

Last Name:- Xyz

Branch:- IT

Mobile No:- 9852033520

delete.htmlA screenshot of a web form titled "Student Information" on a black background. It features a yellow input field containing the enrollment number "1234567890" and a blue button labeled "Delete Data" below it.

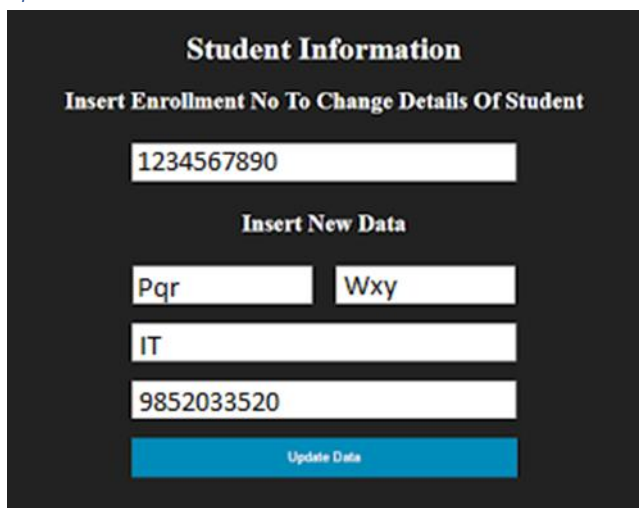
Student Information

1234567890

Delete Data

Result

Student Data Deleted

update.htmlA screenshot of a web form titled "Student Information" on a black background. It includes a subtitle "Insert Enrollment No To Change Details Of Student" and a white input field with "1234567890". Below this is a section "Insert New Data" with four input fields: "Pqr", "Wxy", "IT", and "9852033520". A blue button labeled "Update Data" is at the bottom.

Student Information

Insert Enrollment No To Change Details Of Student

1234567890

Insert New Data

Pqr Wxy

IT

9852033520

Update Data

Result

Update Student2 Data Succesfull

Practical 6

Aim: Create login form and perform state management using Cookies, HttpSession and URL Rewriting.

****index.html Code****

```
<html>
```

```
<head>
```

```
<title>Login</title>
```

```
<meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <style>
```

```
        .b1
```

```
        {      width: 370px;
```

```
                height: 40px;
```

```
                margin:10px;
```

```
                color: white;
```

```
                background-color: #008CBA;
```

```
                border: 0px solid grey;
```

```
                cursor: pointer;
```

```
        }
```

```
        .side
```

```
        {      padding: 10px;
```

```
                margin: 10px;
```

```
                width: 370px;
```

```
        }
```

```
        h1
```

```
        {      color:white;    }
```

```
        .box
```

```
        {            margin-top:200px;        }

    </style>

</head>

<body bgcolor="#222222" align="center">

<div class="box">

<h1> Welcome to Login Portal</h1>

<form action="FirstServlet" method="post">

    <input class="side" type="text" name="userName" placeholder="Username"/> <br/>

    <input class="side" type="password" name="password" placeholder="Password"/> <br/>

    <input class="b1" type="submit" value="Login"/>

</form>

</div>

</body>

</html>
```

****FirstServlet.java Code****

```
import java.io.*;

import javax.servlet.ServletException;

import javax.servlet.http.*;

public class FirstServlet extends HttpServlet

{
    @Override

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

    {
        try

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

            PrintWriter out = response.getWriter();

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

            String pwd=request.getParameter("password");
```

```
        if(pwd.equals("birju") && name.equals("admin"))

        {
            out.print("<h1>Welcome "+name+"</h1>");

            Cookie ck=new Cookie("uname",name);//creating cookie object
            response.addCookie(ck);          //adding cookie in the response

            out.print("<b>Cookie has been generated for this
            session<br></b>");

            out.print("<b>Click on button to view Cookie...</b>"); //creating
            submit button

            out.print("<form action='SecondServlet' method='post'>");

            out.print("<br><input type='submit' value='go'>");

            out.print("</form>");

        }

        else

        {
            out.println("Incorrect Username or Password!!!");
        }

        out.close();

    }

    catch(Exception e)

    {
        System.out.println(e);
    }

}
```

****SecondServlet.java Code:****

```
import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;
```



```
public class SecondServlet extends HttpServlet

{
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException

    {

        @Override

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

        {

            @Override

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

            {
                try

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

                    PrintWriter out = response.getWriter();

                    Cookie ck[]=request.getCookies();

                    out.print("Value stored in Cookie : "+ck[0].getValue());

                    out.close();

                }

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

            }

        }

    }
}
```

****web.xml Code****

```
<?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">

<servlet>
```

```
<servlet-name>FirstServlet</servlet-name>

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

</servlet>

<servlet>

    <servlet-name>SecondServlet</servlet-name>

    <servlet-class>SecondServlet</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>FirstServlet</servlet-name>

    <url-pattern>/FirstServlet</url-pattern>

</servlet-mapping>

<servlet-mapping>

    <servlet-name>SecondServlet</servlet-name>

    <url-pattern>/SecondServlet</url-pattern>

</servlet-mapping>

<session-config>

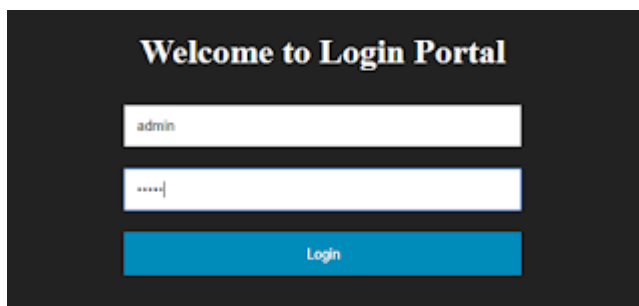
    <session-timeout> 30 </session-timeout>

</session-config>

</web-app>
```

Output:

[index.html](#)



FirstServlet

Welcome admin

Cookie has been generated for this session
Click on button to view Cookie...

SecondServlet

Value stored in Cookie : admin

Practical 7

Aim: Implement Authentication filter using filter API.

****index.html Code****

```
<html>

<head> <title>Filter API</title>

<style>

.b1
{
    width: 370px;

    height: 42px;

    margin:10px;

    color: white;

    background-color: #008CBA;

    border: 0px solid grey;

    cursor: pointer;
}

.side
{
    padding: 10px;

    margin: 20px;

    width: 370px;
}

.b1:hover
{
    background-color: #4CAF50;
}

</style>

</head>

<body bgcolor="#222222">

<div class="box">

<form action="filtering" style="text-align:center;">
```

```
<h1 style="color:white;font-family: Times;margin-top:160px;">Login Portal</h1>

<input class="side" type="text" required="" name="name" placeholder="UserName"> <br>

<input class="side" type="password" required="" name="password"
placeholder="Password" style="margin-top:-5px;"> <br>

<input class="b1" type="submit" value="Login">

</form>

</div>

</body>

</html>
```

****MyFilter.java Code****

```
import java.io.*;

import javax.servlet.*;

public class MyFilter implements Filter

{
    @Override

    public void init(FilterConfig arg0) throws ServletException{}

    @Override

    public void doFilter(ServletRequest req, ServletResponse resp,FilterChain chain) throws
IOException, ServletException

    {
        PrintWriter pw=resp.getWriter();

        String pwd=req.getParameter("password");

        if(pwd.equals("birju"))

        {
            chain.doFilter(req, resp);
        }

        else

        {
            pw.print("Invalid Password");

            RequestDispatcher redi=req.getRequestDispatcher("index.html");

            redi.include(req, resp);

        }

    }

}
```

```
        @Override

        public void destroy() { }

    }
```

****ServletFilter Code****

```
import java.io.*;

import javax.servlet.http.HttpServlet;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ServletFilter extends HttpServlet

{

    @Override

    public void doGet(HttpServletRequest res,HttpServletResponse resp) throws

IOException,ServletException

    {

        resp.setContentType("text/html;charset=UTF-8");

        PrintWriter out = resp.getWriter();

        String str = res.getParameter("name");

        out.print("<h1>Hello "+str+"</h1><h2>You have sucessfully logged in....</h2>");

    }

}
```

****web.xml Code****

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

<web-app>

<servlet>

    <servlet-name>ServletFilter</servlet-name>

    <servlet-class>ServletFilter</servlet-class>

</servlet>
```

```
<welcome-file-list>

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

</welcome-file-list>

<servlet-mapping>

    <servlet-name>ServletFilter</servlet-name>

    <url-pattern>/filtering</url-pattern>

</servlet-mapping>

<filter>

    <filter-name>f1</filter-name>

    <filter-class>MyFilter</filter-class>

</filter>

<filter-mapping>

    <filter-name>f1</filter-name>

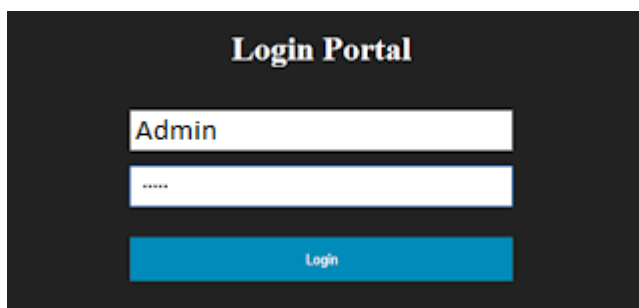
    <url-pattern>/filtering</url-pattern>

</filter-mapping>

</web-app>
```

Output:

[index.html](#)



Result

Hello Admin

You have successfully logged in...

Practical 8

Aim: Create database of student subject-wise data and retrieve all data using JSP and generate xml structure along with DTD and XML Schema definition

index.jsp

```
<%--  
  
    Document: index  
  
    Created on: Mar 29, 2018, 9:51:41 AM  
  
    Author: admin  
  
--%>  
  
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
  
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>  
  
<html>  
  
<head>  
  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
  
    <title>JSP Page</title>  
  
</head>  
  
<body>  
  
    <h3>Students Info:</h3>  
  
    <c:import var="studentInfo" url="student.xml"/>  
  
    <x:parse xml="${studentInfo}" var="output"/>  
  
    <table border="1">  
  
    <tr>  
  
        <th>Student id</th>  
  
        <th>Student name</th>  
  
        <th>Subject1 name</th>  
  
        <th>Subject1 marks</th>
```

```
        <th>Subject2 name</th>

        <th>Subject2 marks</th>

    </tr>

    <x:forEach select="$output/students/student" var="item">

    <tr>

        <td> <x:out select="$item/id" /> </td>

        <td> <x:out select="$item/name" /> </td>

        <td> <x:out select="$item/subject[1]/subject_name"/> </td>

        <td> <x:out select="$item/subject[1]/subject_marks"/> </td>

        <td> <x:out select="$item/subject[2]/subject_name"/> </td>

        <td> <x:out select="$item/subject[2]/subject_marks"/> </td>

    </tr>

    </x:forEach>

</table>

</body>

</html>
```

student.xml

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

<!DOCTYPE students [

    <!ELEMENT students (student+)>

    <!ELEMENT student (id,name,subject+)>

    <!ELEMENT subject (subject_name,subject_marks)>

    <!ELEMENT id (#PCDATA)>

    <!ELEMENT name (#PCDATA)>

    <!ELEMENT subject_name (#PCDATA)>

    <!ELEMENT subject_marks (#PCDATA)>

]>
```

```
<students>
<student>
  <id>1</id>
  <name>Rishi</name>
  <subject>
    <subject_name>AJ</subject_name>
    <subject_marks>80</subject_marks>
  </subject>
  <subject>
    <subject_name>WT</subject_name>
    <subject_marks>90</subject_marks>
  </subject>
</student>
<student>
  <id>2</id>
  <name>Varun</name>
  <subject>
    <subject_name>AJ</subject_name>
    <subject_marks>85</subject_marks>
  </subject>
  <subject>
    <subject_name>WT</subject_name>
    <subject_marks>85</subject_marks>
  </subject>
</student>
<student>
  <id>3</id>
```

```
<name>Hetul</name>

<subject>

    <subject_name>AJ</subject_name>

    <subject_marks>85</subject_marks>

</subject>

<subject>

    <subject_name>WT</subject_name>

    <subject_marks>75</subject_marks>

</subject>

</student>

</students>
```

student2.xml

```
<?xml version="1.0"?>
```

```
<!--
```

To change this license header, choose License Headers in Project Properties.

To change this template file, choose Tools | Templates

and open the template in the editor.

```
-->
```

```
<xs:schema version = "1.0" xmlns:xs = http://www.w3.org/2001/XMLSchema elementFormDefault = "qualified">
```

```
<xs:element name="students">
```

```
    <xs:complexType>
```

```
        <xs:sequence>
```

```
            <xs:element name="student" maxOccurs="unbounded">
```

```
                <xs:complexType>
```

```
                    <xs:sequence>
```

```
                        <xs:element name="id" type="xs:decimal"/>
```

```

        <xs:element name="name"
type="xs:string"/>

        <xs:element name="subject"
maxOccurs="unbounded">

            <xs:complexType>

                <xs:sequence>

                    <xs:element
name="sub_name"
type="xs:string"/>

                    <xs:element
name="sub_marks"
type="xs:decimal"/>

                </xs:sequence>

            </xs:complexType>

        </xs:element>

    </xs:sequence>

</xs:complexType>

</xs:element>

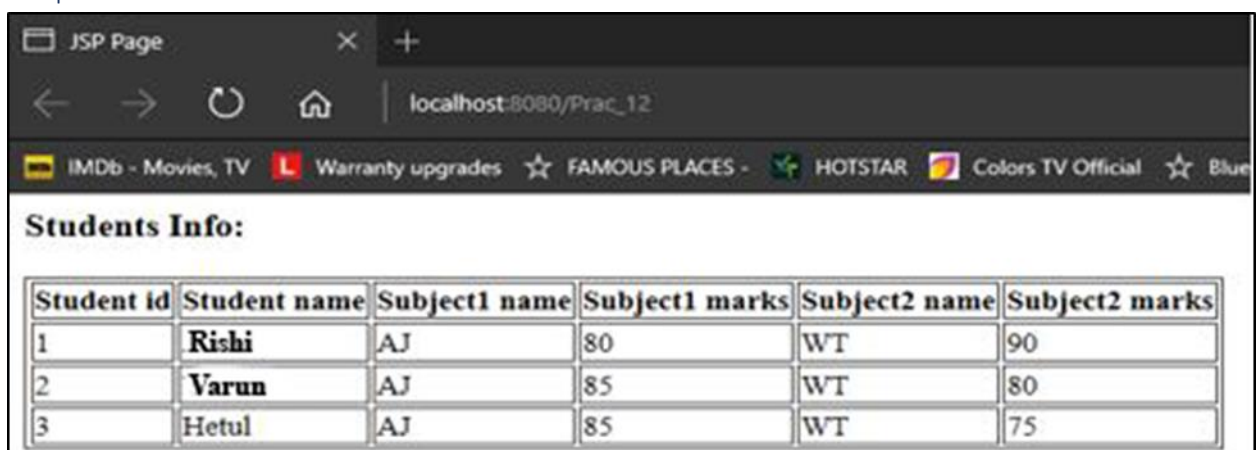
</xs:sequence>

</xs:complexType>

</xs:element></xs:schema>

```

Output:



The screenshot shows a web browser window with the address bar at localhost:8080/Prac_12. The page title is 'JSP Page'. Below the browser window, there is a table titled 'Students Info:' with the following data:

Student id	Student name	Subject1 name	Subject1 marks	Subject2 name	Subject2 marks
1	Rishi	AJ	80	WT	90
2	Varun	AJ	85	WT	80
3	Hetul	AJ	85	WT	75

Practical 9

Aim: Implement Action tags using JSP.

JSP Action

JSP actions use the construct in XML syntax to control the behavior of the servlet engine.

We can dynamically insert a file, reuse the beans components, forward user to another page, etc. through JSP Actions like include and forward.

Unlike directives, actions are re-evaluated each time the page is accessed.

Syntax:

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

There are 11 types of action names as following:

jsp:useBean

jsp:include

jsp:setProperty

jsp:getProperty

jsp:forward

jsp:plugin

jsp:attribute

jsp:body

jsp:text

jsp:param

jsp:attribute

jsp:output

jsp:useBean:

This action name is used when we want to use beans in the JSP page.

With this tag, we can easily invoke a bean.

Syntax of jsp: UseBean:

```
<jsp:useBean id="" class="" />
```

Here it specifies the identifier for this bean and class is full path of the bean class

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Action JSP1</title>

</head>

<body>

<jsp:useBean id="name" class="demotest.DemoClass">

</body>

</html>
```

Explanation of the code:

Code Line 10: In the above code we use "bean id" and "class path" of the bean.

include

It also used to insert a jsp file into another file, just like include directive.

It is added during request processing phase

Syntax of jsp:include

```
<jsp:include page="page URL" flush="true/false">
```

Example:

Action_jsp2 (Code Line 10) we are including a date.jsp file

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Date JSP</title>
```

```
</head>
```

```
<body>
```

```
<jsp:include page="date.jsp" flush="true" />
```

```
</body>
```

```
</html>
```

Date.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
<p>
```

```
Today's date: <%= {new java.util.Date()}.toLocaleString()%>
```

```
</p>
```

```
</body>
```

```
</html>
```

Explanation of the code:

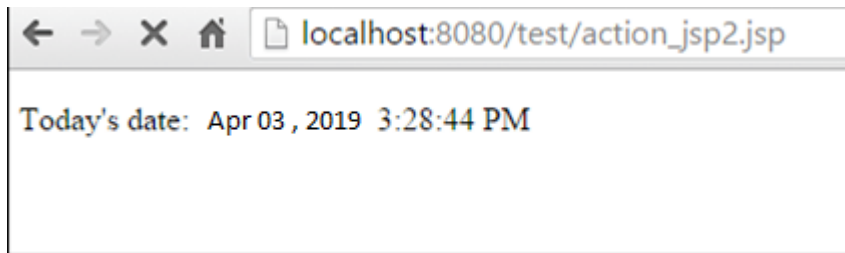
Action_jsp2.jsp

Code Line 10: In the first file we are including the date.jsp file in action_jsp2.jsp

Date.jsp:

Code Line 11: We are printing today's date in code line 11 in date.jsp

When you execute the code following is the output.



Output:

It displays today's date with time as date file is included in the main jsp

setProperty

This property is used to set the property of the bean.

We need to define a bean before setting the property

Syntax:

```
<jsp:setproperty name="" property="" >
```

Here, the name defines the bean whose property is set and property which we want to set.

Also, we can set value and param attribute.

Here value is not mandatory, and it defines the value which is assigned to the property.

Here param is the name of the request parameter using which value can be fetched.

The example of setproperty will be demonstrated below with getproperty

getProperty

This property is used to get the property of the bean.

It converts into a string and finally inserts into the output.

Syntax:

```
<jsp:getAttribute name="" property="" >
```

Here, the name of the bean from which the property has to be retrieved and bean should be defined. The property attribute is the name of the bean property to be retrieved.

Example of setProperty and getProperty:

TestBean.java:

```
package demotest;

import java.io.Serializable;

public class TestBean implements Serializable{

    private String msg = "null";
```

```
        public String getMsg() {  
            return msg;  
        }  
  
        public void setMsg(String msg) {  
            this.msg = msg;  
        }  
    }  
}
```

Action_jsp3.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
    "http://www.w3.org/TR/html4/loose.dtd">  
  
<html>  
  
<head>  
  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
  
<title> Action 3</title>  
  
</head>  
  
<body>  
  
<jsp:useBean id="Test" class="demotest.TestBean" />  
  
<jsp:setProperty name="Test" property="msg" value="Tutorial" />  
  
<jsp:getProperty name="Test" property="msg" />  
  
</body>  
  
</html>
```

Explanation of the code:

TestBean.java:

Code Line 5: TheTestBean is implementing the serializable class. It is a bean class with getters setters in the code.

Code Line 7: Here we are taking private string variable msg as "null"

Code Line 9-14: Here we are using getters and setters of variable "msg".

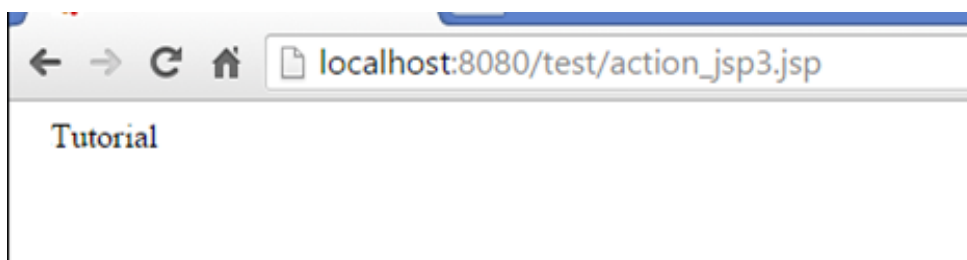
Action_jsp3.jsp

Code Line 10: Here we are using "useBean" tag, where it specifies the bean i.e TestBean which has to be used in this jsp class

Code Line 11: Here we are setting the value for the property msg for bean TestBean as "Tutorial."

CodeLine12: Here using getProperty, we are getting the value of property msg for bean TestBean i.e Tutorial which is there in the output

When you execute the above code you get the following output:



Output:

In this example, using TestBean we are trying to set the property "test" using setProperty and get the value of property using getProperty as "Tutorial"

forward:

It is used to forward the request to another jsp or any static page.

Here the request can be forwarded with no parameters or with parameters.

Syntax:

```
<jsp:forward page="value">
```

Here value represents where the request has to be forwarded.

Example:

Action_jsp41.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title> Action JSP1</title>

</head>

<body>

<jsp:forward page="jsp_action_42.jsp" />

</body>

</html>
```

Jsp_action_42.jsp

```
Action JSP2</title>

</head>

<body>

<a>This is after forward page</a>

</body>

</html>
```

[Explanation of the code](#)

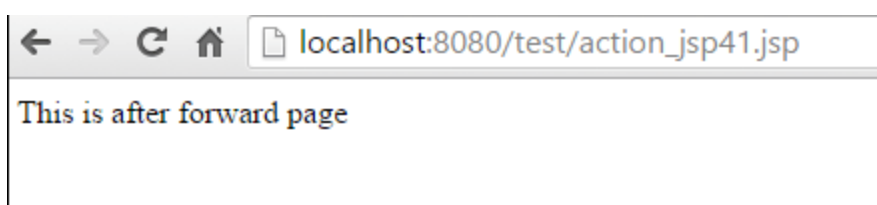
Action_jsp41.jsp

Code Line 10: Here we are using forward JSP Action to forward the request to the page mentioned in the attribute, i.e., jsp_action_42.jsp

Jsp_action_42.jsp

Code Line 10: Once we call action_jsp41.jsp, the request gets forwarded to this page, and we get the output as "This is after forward page."

When we execute the above code, we get the following output



Output:

We call action_jsp41.jsp but the request gets forwarded to jsp_action_42.jsp, and we get the output from that page as "This is after forward page".

plugin

It is used to introduce [Java](#) components into jsp, i.e., the java components can be either an applet or bean.

It detects the browser and adds <object> or <embed> tags into the file

Syntax:

```
<jsp:plugin type="applet/bean" code="objectcode" codebase="objectcodebase">
```

Here the type specifies either an object or a bean

Code specifies class name of applet or bean

Code base contains the base URL that contains files of classes

7) param

This is child object of the plugin object described above

It must contain one or more actions to provide additional parameters.

Syntax:

```
<jsp:params>
```

```
<jsp:param name="val" value="val" / >
```

```
</jsp:params>
```

Example of plugin and param

Action_jsp5.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Action_jsp5</title>
```

```
</head>
```

```
<body>
```

```
<jsp:plugin type="bean" code="Student.class" codebase="demotest.Student">
```

```
<jsp:params>
  <jsp:param name="id" value="5" />
  <jsp:param name="name" value="" />
</jsp:params>
</jsp:plugin>
</body>
</html>
```

Student.java

```
package demotest;

import java.io.Serializable;

public class Student implements Serializable {
    public String getName () {
        return name;
    }

    public void setName (String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId (int id) {
        this.id = id;
    }

    private String name = "null";

    private int id = 0;

}
```

Explanation of the code:

Action_jsp5.jsp

Code Line 10: Here we are taking jsp: plugin object where we are taking three attributes

Type – in this case it is bean

Code- name of the file

Codebase – path with the package name

Code Line 11-14: Here we are taking jsp: params object under which there is a child param object with the attributes of name and value, and we are setting the values of id and name in this attributes.

Student.java

Code 7- 17: We are using getters and setters for variables id and name

Code 19-20: we are initializing variables id and name.

Here we will get output in the case when the set values of param will be used in Student Bean. In this case, we won't have any output as we are just setting and getting values of param but not printing it anywhere.

8) body:

This tag is used to define the XML dynamically i.e., the elements can generate during request time than compilation time.

It actually defines the XML, which is generated dynamically element body.

Syntax:

```
<jsp:body></jsp:body>
```

Here we write XML body tag within this tags

9) attribute:

This tag is used to define the XML dynamically i.e. the elements can be generated during request time than compilation time

It actually defines the attribute of XML which will be generated dynamically.

Syntax:

```
<jsp:attribute></jsp:attribute>
```

Here we write attribute tag of XML.

Example of body and attribute:

Action_jsp6.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Action JSP6</title>

</head>

<body>

<jsp:element name="XMLElement">

<jsp:attribute name="XMLAttribute">

Value

</jsp:attribute>

<jsp:body> XML</jsp:body>

</jsp:element>

</body>

</html>
```

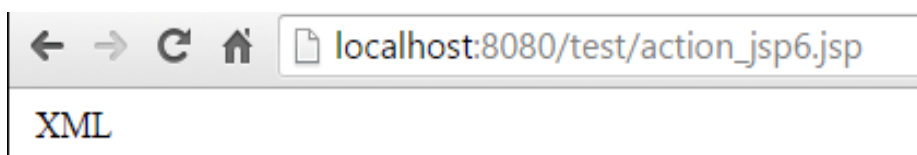
Explanation of the code:

Code Line 10: Here we are defining element, which dynamically generated as XML, and its name will be XMLElement

Code Line 11-13: Here we are defining an attribute which will XML attribute of the dynamically generated XML.

Code Line 14: Here we have body action where we are writing the XML body which will be generated in dynamically XML.

When you execute the above code, you get the following output:



Output:

Here we get the output from the body tag of generated XML.

10) text

It is used to template text in JSP pages.

Its body does not contain any other elements, and it contains only text and EL expressions.

Syntax:

```
<jsp:text>template text</jsp:text>
```

Here template text refers to only template text (which can be any generic text which needs to be printed on jsp) or any EL expression.

Example:

Action_jsp7.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title> Action JSP7</title>

</head>

<body>

<jsp:text>Template Text</jsp:text>

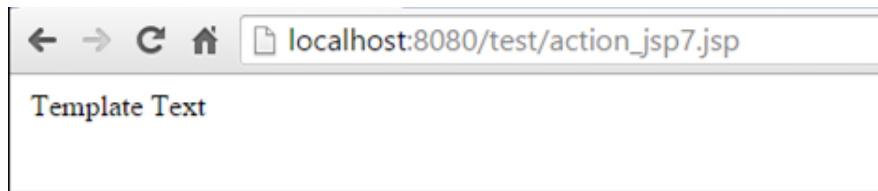
</body>

</html>
```

Explanation of the code:

Code Line 10: Here we are taking text object to print the template text

When you execute the above code, you get the following output



Output:

We are getting Template Text, which is placed within text action objects.

11) output:

It specifies the XML declaration or the DOCTYPE declaration of jsp

The XML declaration and DOCTYPE are declared by the output

Syntax:

```
<jsp:output doctype-root-element="" doctype-system="">
```

Here, doctype-root-element indicates the root element of XML document in DOCTYPE.

Doctype-system indicates doctype which is generated in output and gives system literal

Example:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Action JSP8</title>
```

```
</head>
```

```
<body>
```

```
<jsp:output doctype-root-element="html PUBLIC" doctype-  
system="http://www.w3.org/TR/html4/loose.dtd"/>
```

```
</body>
```

```
</html>
```

Explanation of the code:

Code Line 10: Here we are using output action object to generate a DOCTYPE, and internally it will be generated in this format:

```
<!DOCTYPE html "http://www.w3.org/TR/html4/loose.dtd">
```

There won't be any output for this as this will be generated internally.

Practical 10 (a)

Aim: Create web service which provides student information.

```
package abc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "NewWebService")

public class NewWebService {

    @WebMethod(operationName = "View_data")

    public String View_data(@WebParam(name = "name") String id) {

        int sid = 0;

        String name = null;

        try

        {

            Class.forName("org.apache.derby.jdbc.ClientDriver");

            Connection

con=DriverManager.getConnection("jdbc:derby://localhost:1527/sample","app","app");

            Statement st=con.createStatement();

            ResultSet rs=st.executeQuery("select * from student where id="+id+"");

            if(rs.next())

            {

                sid = rs.getInt("id");

                name = rs.getString("name");
```

```
    }  
    }  
    catch(Exception e)  
    {  
    }  
    return "ID:" + sid + "      " + "Name: " + name;  
  }  
}
```

Jsp file:

<%--

Document : newjsp

Created on: Mar 30, 2018, 2:06:23 PM

Author : admin

--%>\

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>JSP Page</title>

</head>

<body>

<form action="index.jsp">

Id: <input type="text" name="sid"/>

<input type="submit" value="view_data"/>

</form>

</body>

</html>

Output:

The screenshot shows a web browser window with the title "StudentService Web Ser" and "Process Page". The address bar shows "localhost:8080/StudentInfo/StudentService?Tester". The page has a header "StudentService Web Service Tester". Below the header, there is a text area with the following content:

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.lang.String studentdata.StudentService.getData(java.lang.String)

getData (Aarsh)

The screenshot shows a web browser window with the title "Method invocation tra..." and "Process Page". The address bar shows "localhost:8080/StudentInfo/StudentService?Tester". The page has a header "Method invocation tra...". Below the header, there is a section titled "getData Method invocation".

Method parameter(s)

Type	Value
java.lang.String	Aarsh

Method returned

java.lang.String : "Name:Aarsh ID:43"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:getData xmlns:ns2="http://StudentData/">
      <name>Aarsh</name>
    </ns2:getData>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:getDataResponse xmlns:ns2="http://StudentData/">
      <return>Name:Aarsh ID:43</return>
    </ns2:getDataResponse>
  </S:Body>
</S:Envelope>
```

Practical 10 (b)

Aim: Create Web Service client which consume above service and display student data by entering student id.

Html file:

```
<!DOCTYPE html>
```

```
<!--
```

To change this license header, choose License Headers in Project Properties.

To change this template file, choose Tools | Templates

and open the template in the editor.

```
-->
```

```
<html>
```

```
  <head>
```

```
    <title>TODO supply a title</title>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  </head>
```

```
  <body>
```

```
    <form action="index.jsp">
```

```
      Id: <input type="text" name="sid"/><br>
```

```
      <input type="submit" value="view_data"/>
```

```
    </form>
```

```
  </body>
```

```
</html>
```

Java file:

```
package abc;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.ResultSet;

import java.sql.Statement;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "NewWebService")

public class NewWebService {

    @WebMethod(operationName = "View_data")

    public String View_data(@WebParam(name = "name") String id) {

        int sid = 0;

        String name = null;

        try

        {

            Class.forName("org.apache.derby.jdbc.ClientDriver");

            Connection

con=DriverManager.getConnection("jdbc:derby://localhost:1527/sample","app","app");

            Statement st=con.createStatement();

            ResultSet rs=st.executeQuery("select * from student where id="+id+"");

            if(rs.next())

            {

                sid = rs.getInt("id");

                name = rs.getString("name");

            }

        }

        catch(Exception e)

        {

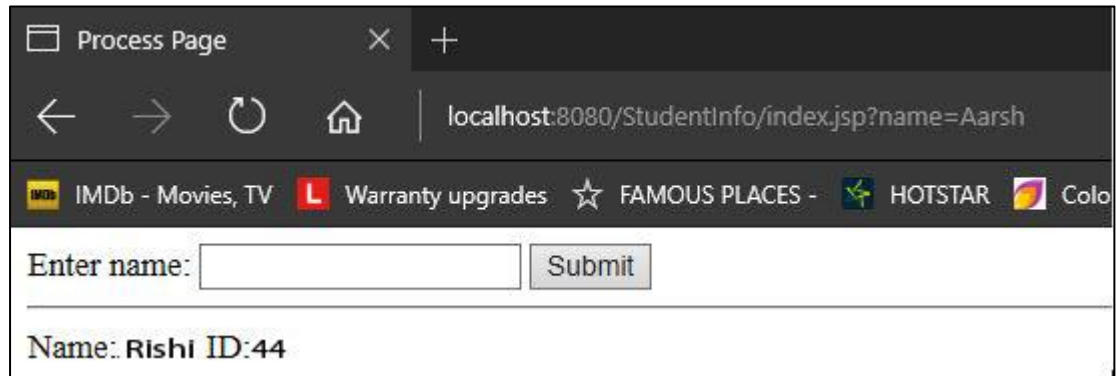
        }

        return "ID:" + sid + "      "+ "Name: " + name;
```



```
}  
}
```

Output:



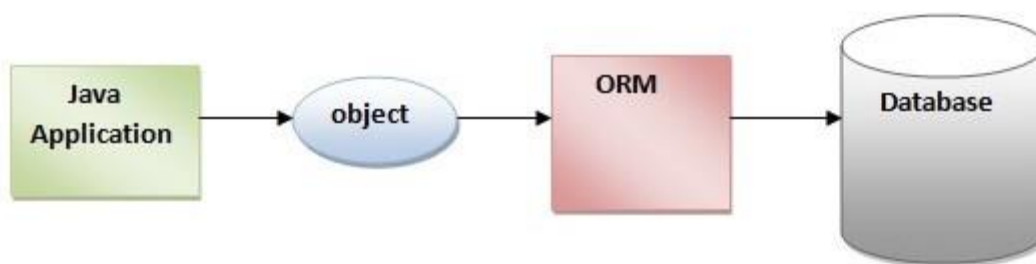
Practical 11

Aim: Study and implement Hibernate

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) 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.

3) 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. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) 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.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

1) Create the Persistent class

A simple Persistent class should follow some rules:

A no-arg constructor: It is recommended that you have a default constructor at least package visibility so that hibernate can create the instance of the Persistent class by newInstance() method.

Provide an identifier property: It is better to assign an attribute as id. This attribute behaves as a primary key in database.

Declare getter and setter methods: The Hibernate recognizes the method by getter and setter method names by default.

Prefer non-final class: Hibernate uses the concept of proxies, that depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.

Let's create the simple Persistent class:

Employee.java

```
package com.javatpoint.mypackage;
```

```
public class Employee {
```

```
    private int id;
```

```
    private String firstName,lastName;
```

```
    public int getId() {
```

```
        return id;
```

```
    }
```

```
    public void setId(int id) {
```

```
        this.id = id;
```

```
    }
```

```
    public String getFirstName() {
```

```
        return firstName;
```

```
}  
  
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}  
  
public String getLastName() {  
    return lastName;  
}  
  
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}  
  
}
```

2) Create the mapping file for Persistent class

The mapping file name conventionally, should be class_name.hbm.xml. There are many elements of the mapping file.

hibernate-mapping : It is the root element in the mapping file that contains all the mapping elements.

class : It is the sub-element of the hibernate-mapping element. It specifies the Persistent class.

id : It is the subelement of class. It specifies the primary key attribute in the class.

generator : It is the sub-element of id. It is used to generate the primary key. There are many generator classes such as assigned, increment, hilo, sequence, native etc. We will learn all the generator classes later.

property : It is the sub-element of class that specifies the property name of the Persistent class.

Let's see the mapping file for the Employee class:

employee.hbm.xml

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

```
<!DOCTYPE hibernate-mapping PUBLIC
```

```
"-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>

  <class name="com.javatpoint.mypackage.Employee" table="emp1000">

    <id name="id">

      <generator class="assigned"></generator>

    </id>

    <property name="firstName"></property>

    <property name="lastName"></property>

  </class>

</hibernate-mapping>
```

3) Create the Configuration file

The configuration file contains information about the database and mapping file. Conventionally, its name should be hibernate.cfg.xml .

hibernate.cfg.xml

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

<!DOCTYPE hibernate-configuration PUBLIC

  "-//Hibernate/Hibernate Configuration DTD 5.3//EN"

  "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

  <session-factory>
```

```
<property name="hbm2ddl.auto">update</property>

<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>

<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>

<property name="connection.username">system</property>

<property name="connection.password">jtp</property>

<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

<mapping resource="employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>
```

4) Create the class that retrieves or stores the object

In this class, we are simply storing the employee object to the database.

```
package com.javatpoint.mypackage;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

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

```
//Create typesafe ServiceRegistry object

StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml")
).build();

Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();


Employee e1=new Employee();

e1.setId(101);

e1.setFirstName("Gaurav");

e1.setLastName("Chawla");


session.save(e1);

t.commit();

System.out.println("successfully saved");

factory.close();

session.close();

}

}
```

5) Load the jar file

For successfully running the hibernate application, you should have the hibernate5.jar file.

[Download the required jar files for hibernate](#)

6) How to run the first hibernate application without IDE

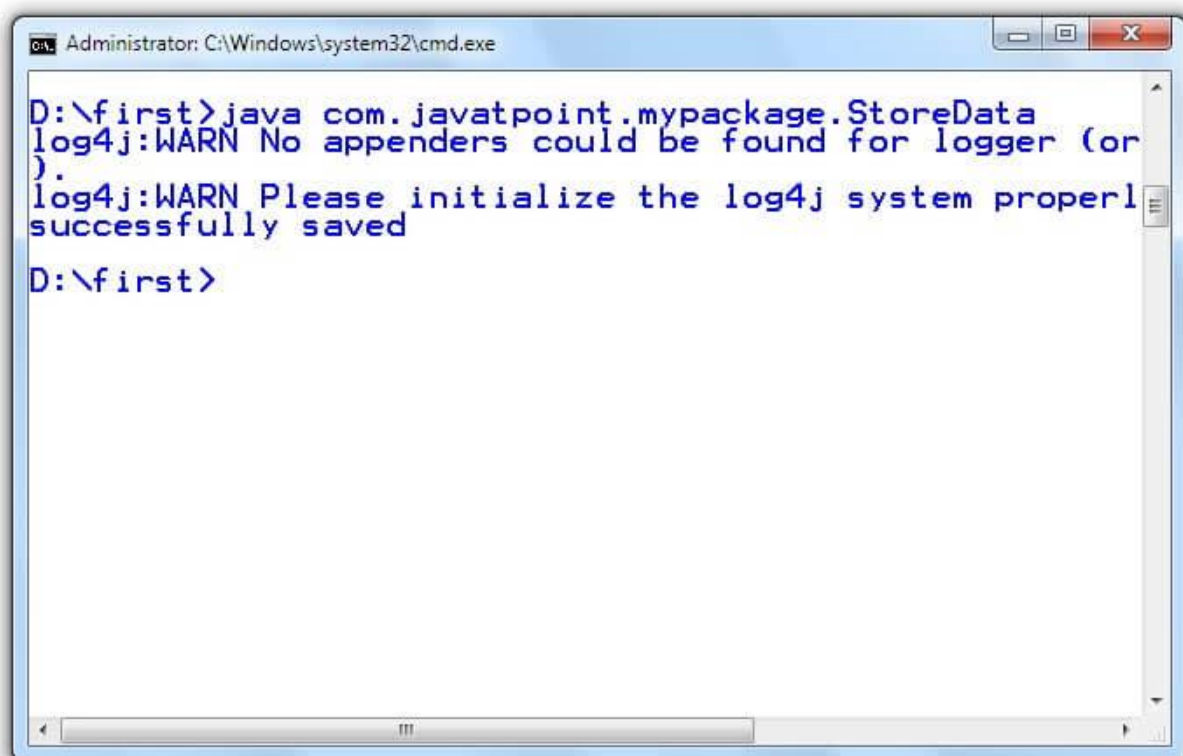
We may run this hibernate application by IDE (e.g. Eclipse, Myeclipse, Netbeans etc.) or without IDE. We will learn about creating hibernate application in Eclipse IDE in next chapter.

To run the hibernate application without IDE:

Install the oracle10g for this example.

Load the jar files for hibernate. (One of the way to load the jar file is copy all the jar files under the JRE/lib/ext folder). It is better to put these jar files inside the public and private JRE both.

Now, run the StoreData class by **java com.javatpoint.mypackage.StoreData**



```
Administrator: C:\Windows\system32\cmd.exe
D:\first>java com.javatpoint.mypackage.StoreData
log4j:WARN No appenders could be found for logger (or
).
log4j:WARN Please initialize the log4j system properl
successfully saved
D:\first>
```


Practical 12

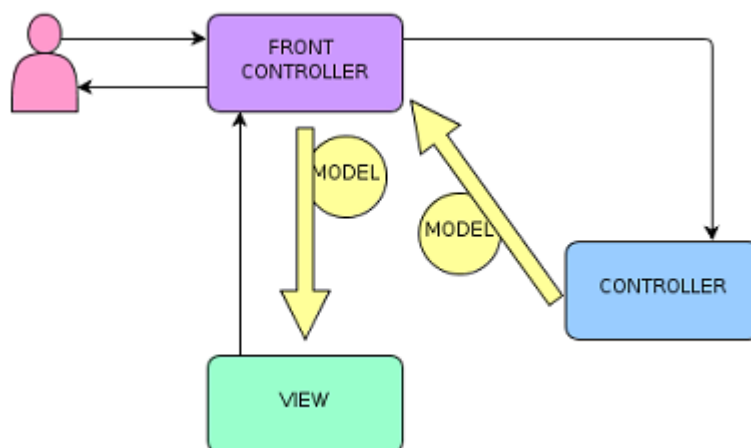
Aim: Study and Implement MVC using Spring Framework

Model View Controller (MVC)

Model view controller is a software architecture design pattern. It provides solution to layer an application by separating three concerns business, presentation and control flow. Model contains business logic, controller takes care of the interaction between view and model. Controller gets input from view and converts it in preferable format for the model and passes to it. Then gets the response and forwards to view. View contains the presentation part of the application.

Spring MVC

Spring MVC is a module for enabling us to implement the MVC pattern. Following image shows the Spring's MVC architecture



DispatcherServlet

This class is a front controller that acts as central dispatcher for HTTP web requests. Based on the handler mappings we provide in spring configuration, it routes control from view to other controllers and gets processed result back and routes it back to view.

Control Flow in Spring MVC

Based on servletmapping from web.xml request gets routed by servlet container to a front controller (DispatcherServlet).

DispatcherServlet uses handler mapping and forwards the request to matching controller.

Controller processes the request and returns ModelAndView back to front controller (DispatcherServlet). Generally controller uses a Service to perform the rules.

DispatcherServlet uses the view resolver and sends the model to view.

Response is constructed and control sent back to DispatcherServlet.

DispatcherServlet returns the response.

```
<?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"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"
  id="WebApp_ID" version="2.5">

  <display-name>Spring Hello World</display-name>

  <welcome-file-list>
    <welcome-file>/</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>springDispatcher</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/config/spring-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springDispatcher</servlet-name>
```

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

</servlet-mapping>

</web-app>
```

Spring Configuration

This spring configuration file provides context information to spring container. In our case,

The tag `mvc:annotation-driven` says that we are using annotation based configurations. `context:component-scan` says that the annotated components like Controller, Service are to be scanned automatically by Spring container starting from the given package.

```
<?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:mvc="http://www.springframework.org/schema/mvc"

       xmlns:context="http://www.springframework.org/schema/context"

       xsi:schemaLocation="

           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd

           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.javapapers.spring.mvc" />

    <mvc:annotation-driven />

    <bean

        class="org.springframework.web.servlet.view.InternalResourceViewResolver">

        <property name="prefix" value="/WEB-INF/view/" />

        <property name="suffix" value=".jsp" />

    </bean>
```

```
</beans>
```

Library files needed

commons-logging.jar

org.springframework.asm-3.1.2.RELEASE.jar

org.springframework.beans-3.1.2.RELEASE.jar

org.springframework.context-3.1.2.RELEASE.jar

org.springframework.core-3.1.2.RELEASE.jar

org.springframework.expression-3.1.2.RELEASE.jar

org.springframework.web.servlet-3.1.2.RELEASE.jar

org.springframework.web-3.1.2.RELEASE.jar

javax.servlet.jsp.jstl-1.2.1.jar (<http://jstl.java.net/download.html> -> JSTL Implementation)

These jars are part of standard spring framework download except the jstl.

Controller

Following controller class has got methods. First one hello maps to the url '/'. Annotation has made our job easier by just declaring the annotation we order Spring container to invoke this method whenever this url is called. return "hello" means this is used for selecting the view. In spring configuration we have declared InternalResourceViewResolver and have given a prefix and suffix. Based on this, the prefix and suffix is added to the returned word "hello" thus making it as "/WEB-INF/view/hello.jsp". So on invoking of "/" the control gets forwarded to the hello.jsp. In hello.jsp we just print "Hello World".

org.springframework.web.servlet.view.InternalResourceViewResolver

Added to printing Hello World I wanted to give you a small bonus with this sample. Just get an input from user and send a response back based on it. The second method "hi" serves that purpose in controller. It gets the user input and concatenates "Hi" to it and sets the value in model object. 'model' is a special [hashmap](#) passed by spring container while invoking this method. When we set a key-value pair, it becomes available in the view. We can use the key to get the value and display it in the view which is our hi.jsp. As explained above InternalResourceViewResolver is used to resolve the view.

```
package com.javapapers.spring.mvc;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller

public class HelloWorldController {

    @RequestMapping("/")
    public String hello() {
        return "hello";
    }

    @RequestMapping(value = "/hi", method = RequestMethod.GET)
    public String hi(@RequestParam("name") String name, Model model) {
        String message = "Hi " + name + "!";
        model.addAttribute("message", message);
        return "hi";
    }

}
```

View – Hello World

```
<html>

<head>

<title>Home</title>

</head>

<body>

    <h1>Hello World!</h1>
```

```
<hr/>
```

```
<form action="hi">
```

```
    Name: <input type="text" name="name"> <input type="submit" value="Submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

Use key “message” in model to get the value and print it.

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

```
<%@ page session="false" %>
```

```
<html>
```

```
    <head>
```

```
        <title>Result</title>
```

```
    </head>
```

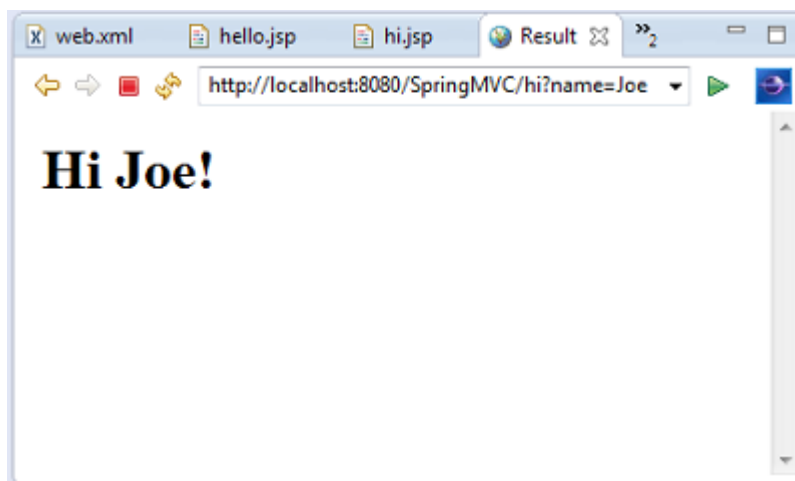
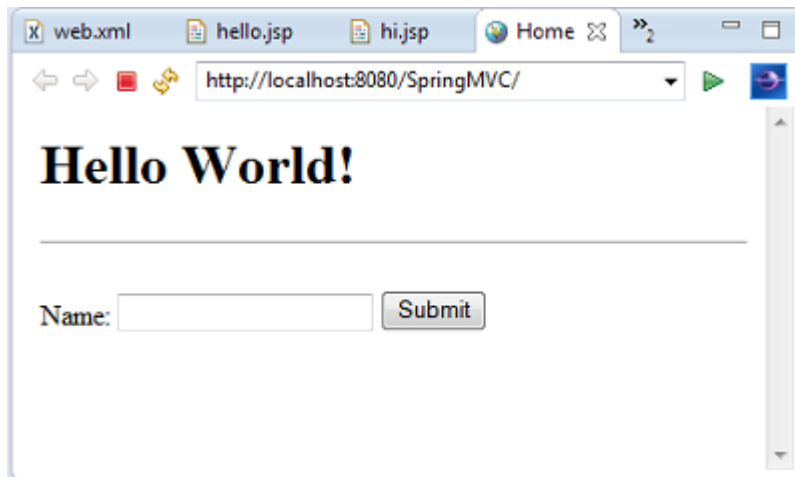
```
    <body>
```

```
        <h1><c:out value="${message}"></c:out></h1>
```

```
    </body>
```

```
</html>
```

Hello World Output



BEYOND SYLLABUS

Aim: Create a simple program using RMI.

****AddI.java Code****

```
import java.rmi.Remote;  
public interface AddI extends Remote  
{  
    public int add(int x,int y) throws Exception;  
}
```

****AddC.java Code****

```
import java.rmi.*;  
import java.rmi.server.*;  
public class AddC extends UnicastRemoteObject implements AddI  
{  
    AddC() throws Exception
```

```
{
    super();
}
public int add(int x,int y)
{
    return x+y;
}
}
```

****Server.java Code****

```
import java.rmi.*;
public class Server
{
    public static void main(String args[]) throws Exception
    {
        AddI obj= new AddC();
        Naming.rebind("Adder",obj);
        System.out.println("Server Started");
    }
}
```

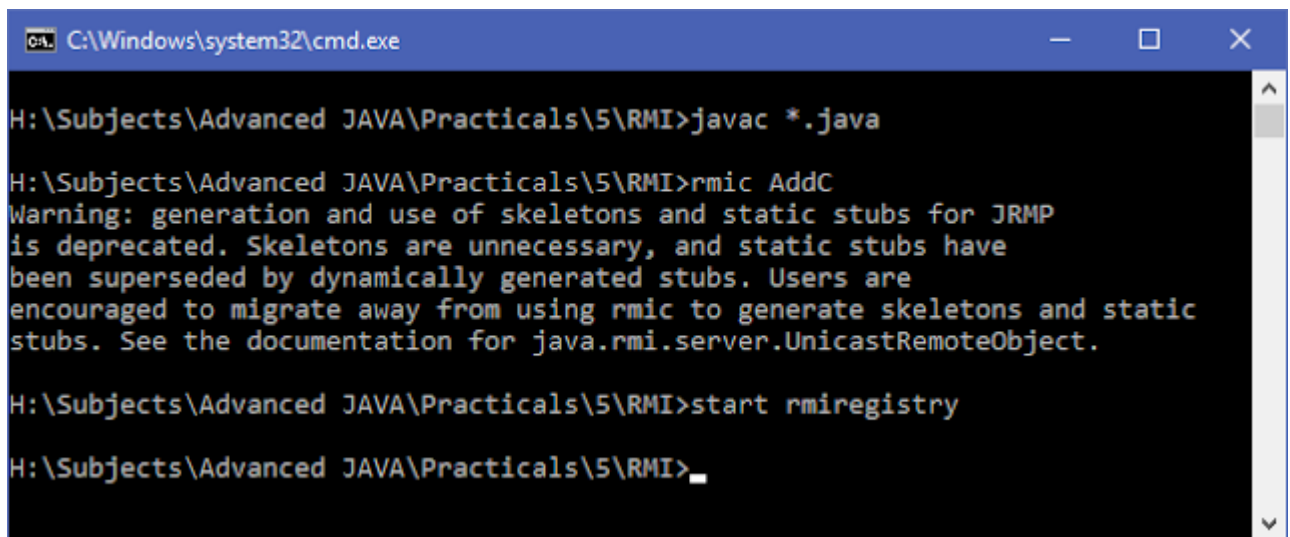
****Client.java Code****

```
import java.rmi.*;
import java.util.Scanner;

public class Client
{
    public static void main(String args[]) throws Exception
    {
        AddI obj=(AddI)Naming.lookup("Adder");
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter 2 numbers to do addition:");
        int a=scan.nextInt();
        int b=scan.nextInt();
        int sum=obj.add(a,b);
        System.out.println("Addition :\t" + sum);
    }
}
```

Output:

Commands



```
C:\Windows\system32\cmd.exe

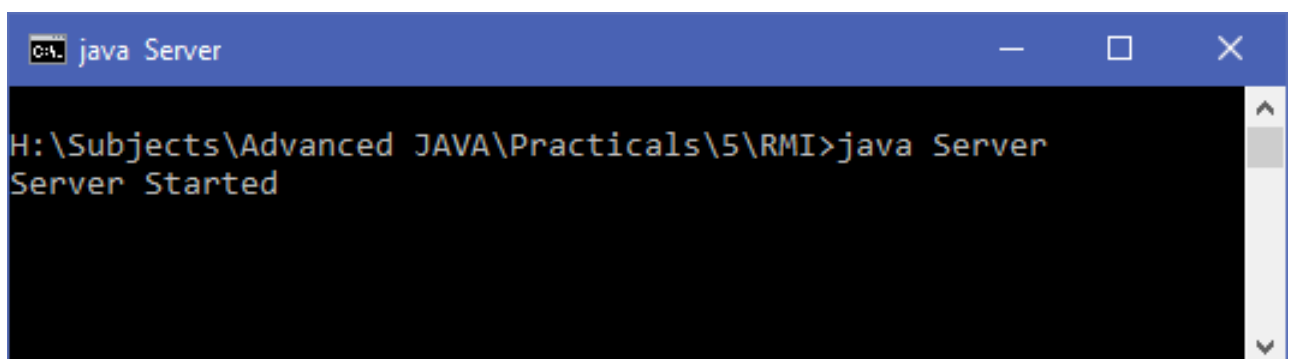
H:\Subjects\Advanced JAVA\Practicals\5\RMI>javac *.java

H:\Subjects\Advanced JAVA\Practicals\5\RMI>rmic AddC
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

H:\Subjects\Advanced JAVA\Practicals\5\RMI>start rmiregistry

H:\Subjects\Advanced JAVA\Practicals\5\RMI>
```

Server



```
java Server

H:\Subjects\Advanced JAVA\Practicals\5\RMI>java Server
Server Started
```