**Amity University Uttar Pradesh**

**Lab File**

**Advance Java Programming**

**IT-404**



**Amity School of Engineering and Technology**

**Submitted To: Submitted By:**

**Dr. Shilpi Sharma Kuldeep Dwivedi**

**ASET A2305218477**

**B. Tech (6CSE7Y)**

**INDEX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Exp no. | Name of Experiment | Date of conduct | Date of submission | Max. marks | Marks obtained | Faculty  signature |
| 1 | Create a biodata using java swing and implementing exception handling | 21/12/2020 | 01/2/2021 | 2 |  | checked |
| 2 | Create 3 interconnected frames using java swing. Home page(static page), login page and catalogue for amity library | 11/1/2021 | 01/2/2021 | 2 |  | checked |
| 3 | To create an employee table and to store and fetch all records using Select Query | 18/1/2021 | 15/02/2021 | 2 |  | checked |
| 4 | Write a program to store, fetch and delete the records using statement and Prepared Statement. | 25/1/2021 | 15/02/2021 | 2 |  | checked |
| 5 | Write a servlet program to print hello world and get input from user. Use cookies to track the number of times user access the page. | 08/02/2021 | 22/02/2021 | 2 |  | checked  Good |
| 6 | Write a program to get input from user and print a welcome message using JSP. | 15/02/2021 | 22/02/2021 | 2 |  | checked |
| 7 | Write a program to create a Loan Calculator which can compute Simple and Compound interest using JSP. | 15/02/2021 | 22/02/2021 | 2 |  | checked |
| 8 | Write a program to get input from user using servlet and connect it with a database. | 22/02/2021 | 15/03/2021 | 2 |  | Checked |
| 9 | Write a program to print 10 numbers, get input from user, calculating the salary and handling the error using Scriptlet. | 1/03/2021 | 15/03/2021 | 2 |  | Checked  Good |
| 10  11  12  13 | Write a program to create a Calculator using statefull Enterprise JavaBeans.  Write a program to develop a deposit and withdrawal form using stateful EJB.  Write a program to implement Entity Bean  Write a program to implement Struts. | 8/03/2021  8/03/2021  15/03/2021  22/03/2021 | 15/03/2021  22/03/2021  22/03/2021  22/03/2021 | 2  2  2  2 |  | Checked  Checked  Checked  Good  Checked  Good |

Date:21/12/2020

**Experiment 1**

**Objective:**

Creating a Biodata form using Java GUI elements.

**Theory:**

**Java AWT:** The Abstract Window Toolkit (**AWT**) is Java's original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing. The **AWT** is part of the Java Foundation Classes (JFC), the standard API for providing a graphical user interface (GUI) for a Java program.

**Java Swing:** Swing in Java is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window-based applications. It is a part of the JFC (Java Foundation Classes). It is built on top of the AWT API and entirely written in java. It is platform independent unlike AWT and has lightweight components.

**Code :**

package com.company;  
  
import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
import javax.swing.event.\*;  
import javax.swing.table.\*;  
  
class biodata extends JFrame {  
  
 JFrame f;  
 JLabel l,l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11;  
 JTextField t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,td;  
 JTextArea ta1,ta2;  
 JComboBox c1;  
 JButton b1,b2;  
 JRadioButton rb1,rb2;  
 ButtonGroup bg1;  
 JCheckBox ch1,ch2,ch3;  
 JTable tb1;  
 TableColumn sc,tc;  
 DefaultTableModel model,dtm;  
  
 public biodata() {  
 f=new JFrame("BIODATA");  
 l=new JLabel("BioData");  
 l.setFont(new Font("Algerian",Font.*BOLD*,40));  
 l1=new JLabel("Name :-");  
 l2=new JLabel("Address :-");  
 l3=new JLabel("Contact Number :-");  
 l4=new JLabel("E-Mail Address :-");  
 l5=new JLabel("Date OF Birth :-");  
 l6=new JLabel("Marital Status :-");  
 l7=new JLabel("Gender :-");  
 l8=new JLabel("Nationality :-");  
 l9=new JLabel("Known Language :-");  
 l10=new JLabel("Qualification :-");  
 l11=new JLabel("Other Qualification :-");  
 t1=new JTextField(20);  
 ta1=new JTextArea(20,20);  
 int v=ScrollPaneConstants.*VERTICAL\_SCROLLBAR\_ALWAYS*;  
 int h=ScrollPaneConstants.*HORIZONTAL\_SCROLLBAR\_ALWAYS*;  
 JScrollPane s=new JScrollPane(ta1,v,h);  
 t2=new JTextField(20);  
 t3=new JTextField(20);  
 t4=new JTextField(20);  
 c1=new JComboBox();  
 c1.addItem("Married");  
 c1.addItem("Unmarried");  
 rb1=new JRadioButton("Male");  
 rb2=new JRadioButton("Female");  
 bg1=new ButtonGroup();  
 bg1.add(rb1);  
 bg1.add(rb2);  
 t5=new JTextField(20);  
 ch1=new JCheckBox("Marathi");  
 ch2=new JCheckBox("Hindi");  
 ch3=new JCheckBox("English");  
 ta2=new JTextArea(20,20);  
  
 String col[]={"Sr No.","Exam Detail","University/Boards","Year Of Passing","Grade/Class","Percentage"};  
 String row[][]=new String[4][6];  
 model = new DefaultTableModel(row, col);  
 tb1 = new JTable(model);  
  
  
  
 JScrollPane s1=new JScrollPane(tb1,v,h);  
 JScrollPane s2=new JScrollPane(ta2,v,h);  
 b1=new JButton("Submit");  
 b2=new JButton("Cancel");  
 JPanel p=new JPanel();  
 p.setLayout(null);  
  
 l.setBounds(200,30,200,50);  
 l1.setBounds(80,80,150,30);  
 l2.setBounds(80,130,150,30);  
 l3.setBounds(80,180,150,30);  
 l4.setBounds(80,220,150,30);  
 l5.setBounds(80,260,150,30);  
 l6.setBounds(80,300,150,30);  
 l7.setBounds(80,340,150,30);  
 l8.setBounds(80,380,150,30);  
 l9.setBounds(80,420,150,30);  
 l10.setBounds(80,460,150,30);  
 l11.setBounds(80,600,150,30);  
  
 t1.setBounds(250,80,200,25);  
 s.setBounds(250,120,200,50);  
 t2.setBounds(250,180,200,25);  
 t3.setBounds(250,220,200,25);  
 t4.setBounds(250,260,200,25);  
 c1.setBounds(250,300,200,25);  
 rb1.setBounds(250,340,100,25);  
 rb2.setBounds(350,340,200,25);  
 t5.setBounds(250,380,200,25);  
 ch1.setBounds(250,420,80,20);  
 ch2.setBounds(330,420,60,20);  
 ch3.setBounds(400,420,80,20);  
 s1.setBounds(20,500,600,100);  
 s2.setBounds(250,610,300,70);  
 b1.setBounds(200,700,100,30);  
 b2.setBounds(350,700,100,30);  
  
 p.add(l);  
 p.add(l1);p.add(l2);  
 p.add(l3);p.add(l4);  
 p.add(l5);p.add(l6);  
 p.add(l7);p.add(l8);  
 p.add(l9);p.add(l10);  
 p.add(l11);p.add(t1);  
 p.add(s);p.add(t2);  
 p.add(t3);p.add(t4);  
 p.add(c1);p.add(rb1);  
 p.add(rb2);p.add(t5);  
 p.add(ch1);p.add(ch2);  
 p.add(ch3);  
 p.add(s1);p.add(s2);  
 p.add(b1);p.add(b2);  
  
 f.add(p,BorderLayout.*CENTER*);  
 f.add(p);  
  
 f.setSize(650,780);  
 f.setVisible(true);  
 f.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
  
 b1.addActionListener(new ActionListener()  
 {  
  
 public void actionPerformed(ActionEvent ae)  
 {  
 try{  
 int l3 = Integer.*parseInt*(t4.getText());  
 }  
 catch (Exception ex){  
 JOptionPane.*showMessageDialog*(null,"Only Numbers are allowed");  
 }  
 JOptionPane.*showMessageDialog*(null, "Data added Successfully ");  
  
 }  
 });  
  
 }  
 public static void main(String args[]){  
 biodata b=new biodata();  
 }  
}

**Output:**

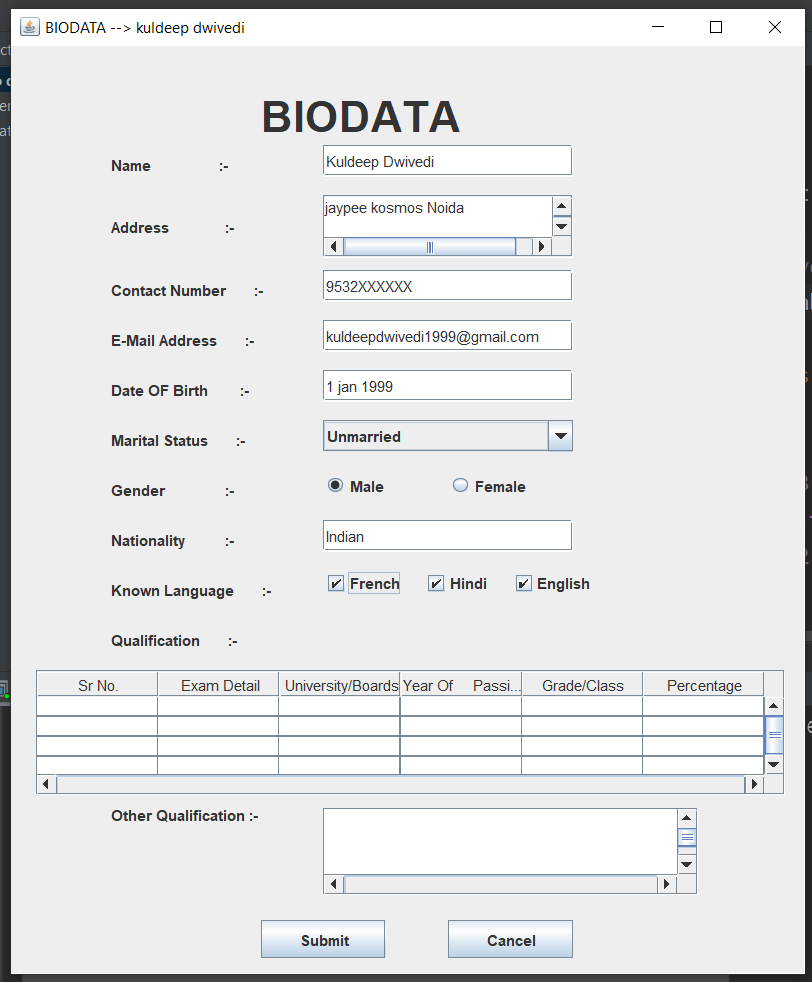
******

Fig : Bio data

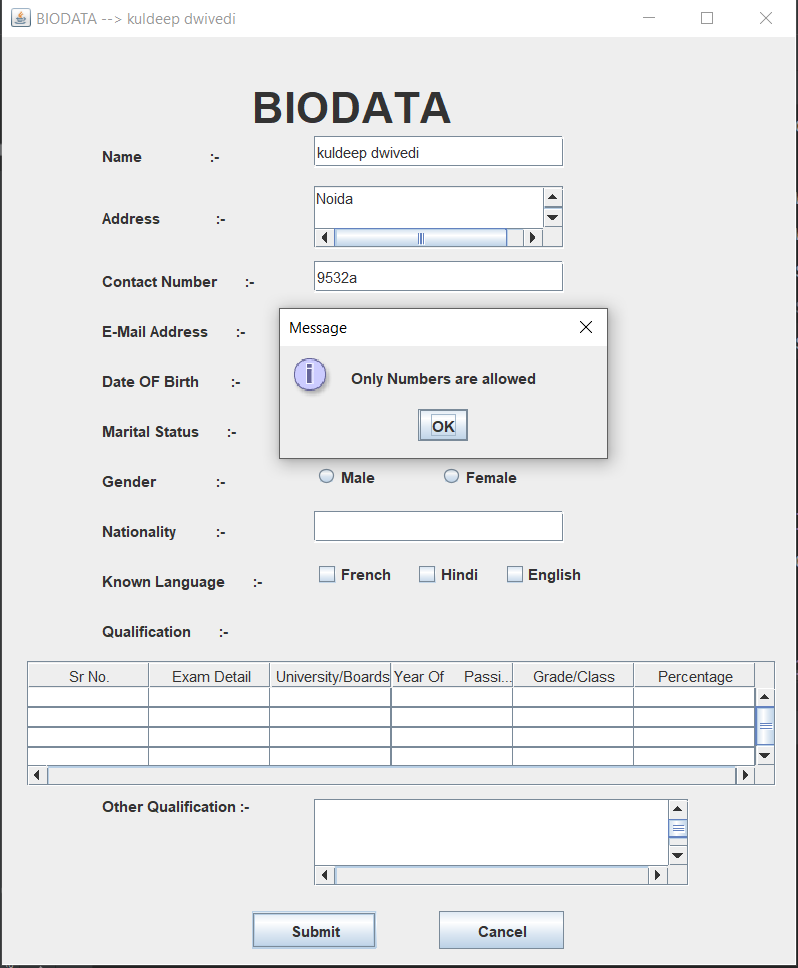


Fig: Exceptional handling

Date : 11/01/2021

**Experiment 2**

**Objective:**

Create a Library Management System using Java GUI elements and HTML.

**Theory:**

A **Library management system** is a software that uses to maintain the record of the library. It contains work like the number of available books in the library, the number of books is issued or returning or renewing a book or late fine charge record, etc.

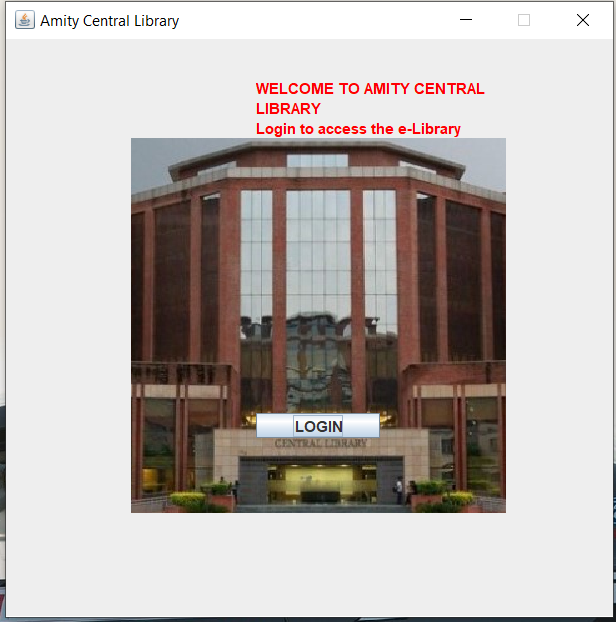
Swing in Java is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window based applications. It is a part of the JFC (Java Foundation Classes). It is built on top of the AWT API and entirely written in [java](https://www.edureka.co/blog/java-tutorial/). It is platform independent unlike AWT and has lightweight components.

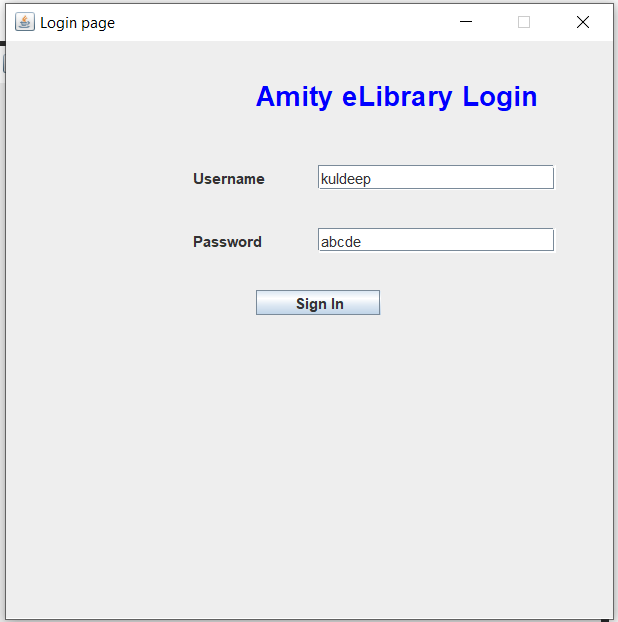
Frame in java is a window that has nice borders, various buttons along the top border, and other features. What you usually call a "window" Java calls a "frame". A frame is a container object, so GUI components can be placed in it. Java Swing provides the utility to show frame within another frame with the use of this you can display a JFrame-like window within another window.

**Code:**

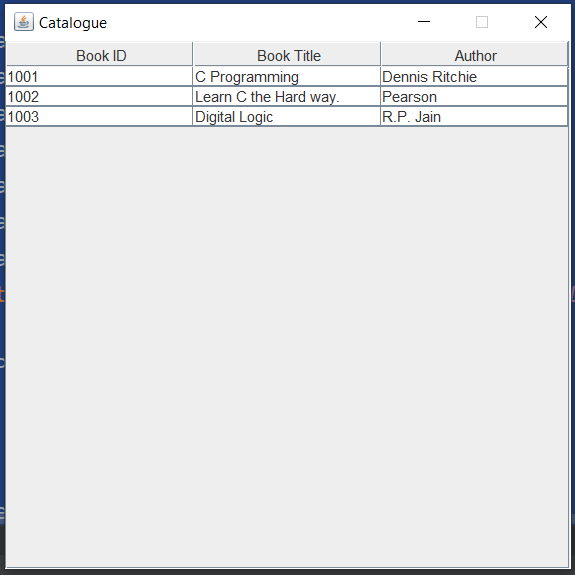
package com.company;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import javax.imageio.ImageIO;  
import static javax.swing.WindowConstants.*EXIT\_ON\_CLOSE*;  
  
class HomePage implements ActionListener {  
 JFrame load = new JFrame();  
 String s;  
 JLabel label;  
 BufferedImage img;  
 JButton login;  
 Container c;  
  
 public HomePage() {  
 load.setTitle("Amity Central Library");  
 load.setBounds(400, 90, 500, 500);  
 load.setResizable(false);  
 c = load.getContentPane();  
 c.setLayout(null);  
  
 s = "<html><strong> Welcome to Amity Central Library<br>";  
 s += " Login to access the e-Library</strong></html>";  
 try {  
 img = ImageIO.*read*((new File("C:\\Users\\User\\Downloads\\Amitylib.jpg")));  
 JLabel picLabel = new JLabel(new ImageIcon(img));  
 picLabel.setSize(300, 300);  
 picLabel.setLocation(100, 80);  
 c.add(picLabel);  
 } catch (Exception e) {  
 System.*out*.println("File not found!!");  
 }  
  
 label = new JLabel(s);  
 label.setForeground(Color.*BLACK*);  
 label.setSize(200, 50);  
 label.setLocation(200, 30);  
 c.add(label);  
  
  
 login = new JButton("Login");  
 login.addActionListener(this);  
 login.setSize(100, 20);  
 login.setLocation(200, 300);  
 c.add(login);  
  
 load.setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 load.setVisible(true);  
 }  
  
 public static void main(String[] args) {  
 // *TODO code application logic here* HomePage hp = new HomePage();  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae) {  
 if (ae.getSource() == login) {  
 load.setVisible(false);  
 new LoginPage();  
 }  
 }  
}  
  
class LoginPage implements ActionListener{  
 JFrame loginPage;  
 JLabel username;  
 JLabel password;  
 JTextField user;  
 JTextField key;  
 JButton signin;  
  
 JLabel title = new JLabel("Amity eLibrary Login");  
 Container cont;  
 public LoginPage()  
 {  
 username = new JLabel("Username");  
 password = new JLabel("Password");  
  
 user = new JTextField(15);  
 key = new JTextField(15);  
  
 loginPage = new JFrame();  
 loginPage.setTitle("Login page");  
 loginPage.setBounds(400,90,500,500);  
 loginPage.setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 loginPage.setResizable(false);  
  
 cont = loginPage.getContentPane();  
 cont.setLayout(null);  
  
 title.setFont(new Font("Arial", Font.*BOLD*, 22));  
 title.setForeground(Color.*BLUE*);  
 title.setSize(300,30);  
 title.setLocation(200, 30);  
 cont.add(title);  
  
 username.setSize(100, 20);  
 username.setLocation(150, 100);  
 cont.add(username);  
 user.setSize(190, 20);  
 user.setLocation(250, 100);  
 cont.add(user);  
  
 password.setSize(190, 20);  
 password.setLocation(150, 150);  
 cont.add(password);  
 key.setSize(190, 20);  
 key.setLocation(250, 150);  
 cont.add(key);  
  
 signin = new JButton("Sign In");  
 signin.setSize(100,20);  
 signin.setLocation(200, 200);  
 signin.addActionListener(this);  
 cont.add(signin);  
  
 loginPage.setLocationRelativeTo(null);  
 loginPage.setVisible(true);  
  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent ae) {  
 if(ae.getSource() == signin)  
 {  
 if(user.getText().equals("kuldeep") && key.getText().equals("abcde"))  
 {  
 loginPage.setVisible(false);  
 new Catalogue();  
 }  
 else  
 {  
 JLabel access = new JLabel("Access Denied!!!");  
 access.setForeground(Color.*RED*);  
 access.setFont(new Font("Arial", Font.*BOLD*, 20));  
 access.setSize(190, 20);  
 access.setLocation(175,250);  
 cont.add(access);  
 loginPage.repaint();  
 }  
 }  
 }  
}  
  
class Catalogue extends JFrame{  
 JFrame frame = new JFrame();  
 String data[][] = new String[][]{{"1001","C Programming","Dennis Ritchie"},  
 {"1002", "Learn C the Hard way.","Pearson"},  
 {"1003", "Digital Logic","R.P. Jain"}};  
 String column[] = new String[]{"Book ID", "Book Title","Author"};  
  
 Container layout;  
 public Catalogue()  
 {  
 JTable cat = new JTable(data,column);  
 frame.setLocationRelativeTo(null);  
 frame.setResizable(false);  
 frame.setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
  
 frame.add(new JScrollPane(cat));  
 frame.setTitle("Catalogue");  
 frame.pack();  
 frame.setVisible(true);  
 }  
}

**Output:**





**Figure 3 Login page**



**Experiment 3**

DATE: 18/1/2021

**Objective:**

To create an employee table and to store and fetch all records using Select Query

**Theory:**

**Statement:** Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement () method. Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

**PreparedStatement:** Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object. This statement gives you the flexibility of supplying arguments dynamically.

**ResultSet:** The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query. A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object**.**

**Program:**

import java.sql.\*;

import java.util.logging.Level;

import java.util.logging.Logger;

import java.sql.Statement;

import java.sql.PreparedStatement;

public class JavaApplication1 {

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

String stname,stdept,printrow;

int stroll;

Connection conn;

PreparedStatement stmt;

Statement st1,st2,st3;

ResultSet rs ;

String sql;

try {

// TODO code application logic here

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

conn = DriverManager.getConnection("jdbc:derby://localhost:1527/Student");

conn.setReadOnly(false);

st1 = conn.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

st2 = conn.createStatement();

System.out.println("VIEWING THE RECORDS");

System.out.println("Roll Name Dept");

rs = st1.executeQuery("SELECT \* FROM Student");

while(rs.next())

{

stroll = rs.getInt(1);

stname = rs.getString(2);

stdept = rs.getString(3);

printrow = stroll + " " + stname+ " "+ stdept;

System.out.println(printrow);

}

System.out.println("AFTER ADDING NEW RECORD");

st2 = conn.createStatement();

sql = "INSERT INTO Student(Roll, Name, Dept) VALUES (100, 'Record', 'ASET')";

st2.executeUpdate(sql);

System.out.println("Roll Name Dept");

rs = st1.executeQuery("SELECT \* FROM Student");

while(rs.next())

{

stroll = rs.getInt(1);

stname = rs.getString(2);

stdept = rs.getString(3);

printrow = stroll + " " + stname+ " "+ stdept;

System.out.println(printrow);

}

System.out.println("AFTER DELETION");

st3 = conn.createStatement();

sql = "DELETE FROM Student WHERE Roll = 5";

st3.executeUpdate(sql);

System.out.println("Roll Name Dept");

rs = st1.executeQuery("SELECT \* FROM Student");

while(rs.next())

{

stroll = rs.getInt(1);

stname = rs.getString(2);

stdept = rs.getString(3);

printrow = stroll + " " + stname+ " "+ stdept;

System.out.println(printrow);

}

}

catch (ClassNotFoundException ex) {

Logger.getLogger(JavaApplication1.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

**Output:**

**Employee table:**

****

DATE: 25/1/2021

**Experiment 4**

**Objective :**

Write a program to store, fetch and delete the records using statement and Prepared Statement.

**Theory :**

**Statement :** Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement () method. Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

**PreparedStatement :** Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object. This statement gives you the flexibility of supplying arguments dynamically.

**ResultSet:** The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query. A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object**.**

**Code:**

package javaapplication1;

import java.sql.\*;

import java.util.logging.Level;

import java.util.logging.Logger;

import java.sql.Statement;

import java.sql.PreparedStatement;

public class JavaApplication1 {

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

String stname,stdept,printrow;

int stroll;

Connection conn;

PreparedStatement stmt;

Statement st1,st2,st3;

ResultSet rs ;

String sql;

try {

// TODO code application logic here

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

conn = DriverManager.getConnection("jdbc:derby://localhost:1527/Student");

conn.setReadOnly(false);

st1 = conn.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE, ResultSet.CONCUR\_READ\_ONLY);

st2 = conn.createStatement();

System.out.println("VIEWING THE RECORDS");

System.out.println("Roll Name Dept");

rs = st1.executeQuery("SELECT \* FROM Student");

while(rs.next())

{

stroll = rs.getInt(1);

stname = rs.getString(2);

stdept = rs.getString(3);

printrow = stroll + " " + stname+ " "+ stdept;

System.out.println(printrow);

}

System.out.println("AFTER ADDING NEW RECORD");

st2 = conn.createStatement();

sql = "INSERT INTO Student(Roll, Name, Dept) VALUES (100, 'Record', 'ASET')";

st2.executeUpdate(sql);

System.out.println("Roll Name Dept");

rs = st1.executeQuery("SELECT \* FROM Student");

while(rs.next())

{

stroll = rs.getInt(1);

stname = rs.getString(2);

stdept = rs.getString(3);

printrow = stroll + " " + stname+ " "+ stdept;

System.out.println(printrow);

}

System.out.println("AFTER DELETION");

st3 = conn.createStatement();

sql = "DELETE FROM Student WHERE Roll = 5";

st3.executeUpdate(sql);

System.out.println("Roll Name Dept");

rs = st1.executeQuery("SELECT \* FROM Student");

while(rs.next())

{

stroll = rs.getInt(1);

stname = rs.getString(2);

stdept = rs.getString(3);

printrow = stroll + " " + stname+ " "+ stdept;

System.out.println(printrow);

}

}

catch (ClassNotFoundException ex) {

Logger.getLogger(JavaApplication1.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

**Output:**

****

DATE: 08/02/2021

**Experiment 5**

**Objective:**

Write a servlet program to print hello world and get input from user. Use cookies to track the number of times user access the page.

**Theory:**

**Servlets:** Servlets are the Java programs that runs on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.

**Properties of Servlets:**

* Servlets work on the server-side.
* Servlets are capable of handling complex requests obtained from web server.

**Cookies:** A cookie is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

There are 2 types of cookies in servlets.

* **Non-persistent cookie:** It is valid for single session only. It is removed each time when user closes the browser.
* **Persistent cookie**: It is valid for multiple session. It is not removed each time when user closes the browser. It is removed only if user logout or sign out.

**Code :**

**Index.html :**

<html>

<head>

<title>HOME PAGE</title>

<meta charset="UTF-8">

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

</head>

<body background="image.jpg" style=" background-repeat:no-repeat;background-size:100%">

<h1>Hello Users! WELCOME TO THE HOME PAGE</h1>

<form name="LoginForm" method="post" action="form1">

<input type="submit" value="Click Next" />

</form>

</body>

</html>

**Form1.java :**

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/form1")

public class form1 extends HttpServlet {

private int hitCount;

public void init()

{

hitCount=0;

}

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

hitCount++;

String title = "Total Number of Hits";

try(PrintWriter out = response.getWriter()){

out.print("<html><body background=\"image.jpg\" style=\" background-repeat:no-repeat;background-size:100%\">"

+"<h1> REGISTRATION FORM</h1><hr>"

+"<h3> Enter the details</h3>"

+"<input type='text' placeholder='Name' name='name' required/><br><br>"

+"<input type='number' placeholder='Age' name='age' required/><br><br>"

+"<input type='text' placeholder='gender' name='gender' required/><br><br>"

+"<form name='loginform' method='post' action='form.html'>"

+" <input type='submit' value='submit'/>"

+"<h1 align = \"center\">" + title + "</h1>\n"

+"<h2 align = \"center\">" + hitCount + "</h2>\n"

+"</body></html>");

}}}

**Form.html :**

<html>

<head>

<title>login</title>

<meta charset="UTF-8">

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

</head>

<body background="hi.jpg" style=" background-repeat:no-repeat;background-size:100%">

<form name="LoginForm" method="post" action="form">

Username: <input type="text" name="username"/> <br/>

Password: <input type="password" name="password"/> <br/>

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

</form>

</body>

</html>

**Form.java :**

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class form extends HttpServlet {

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.print("<html><body background=\"image.jpg\" style=\" background-repeat:no-repeat;background-size:100%\">"

+"</html></body");

String docType = "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

String p = request.getParameter("username");

String n = request.getParameter("password");

if(p.equals("kuldeep")&n.equals("qwerty")){

RequestDispatcher rd=request.getRequestDispatcher("welcome");

rd.forward(request, response);

}

else{

out.print("Sorry UserName or Password Error!");

out.print(" not a valid user");

RequestDispatcher rd=request.getRequestDispatcher("/form.html");

rd.include(request, response);

}

}

}

**Welcome.java :**

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class welcome extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.print("<html><body background=\"image.jpg\" style=\" background-repeat:no-repeat;background-size:100%\">"

+"</html></body");

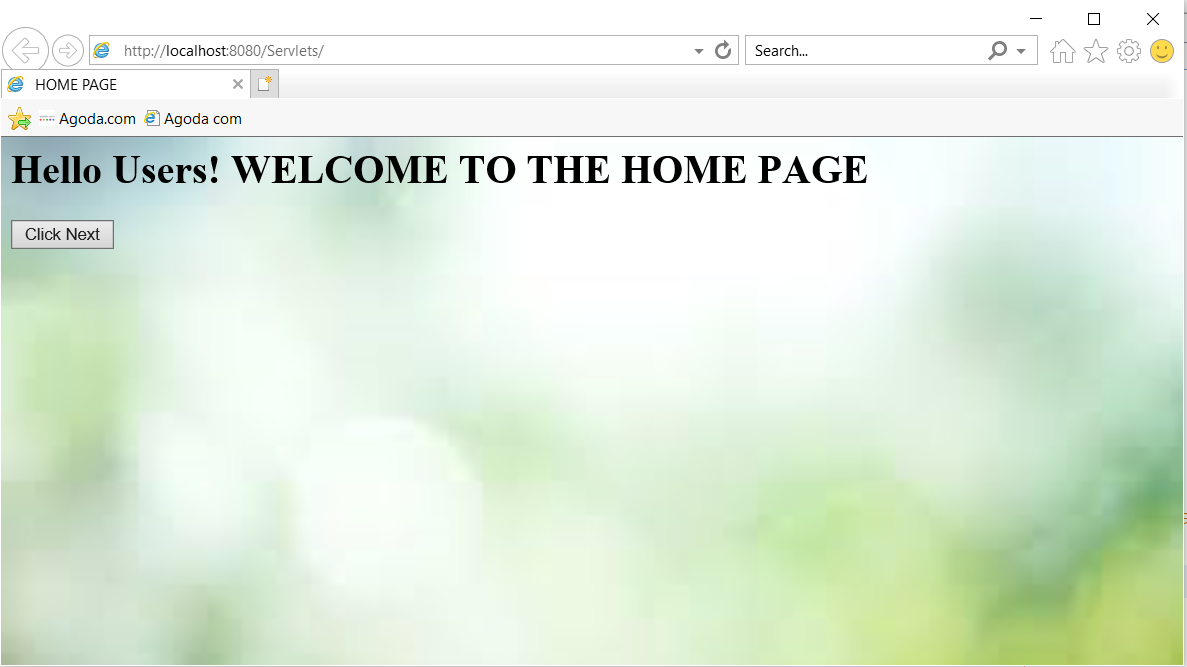
String n=request.getParameter("username");

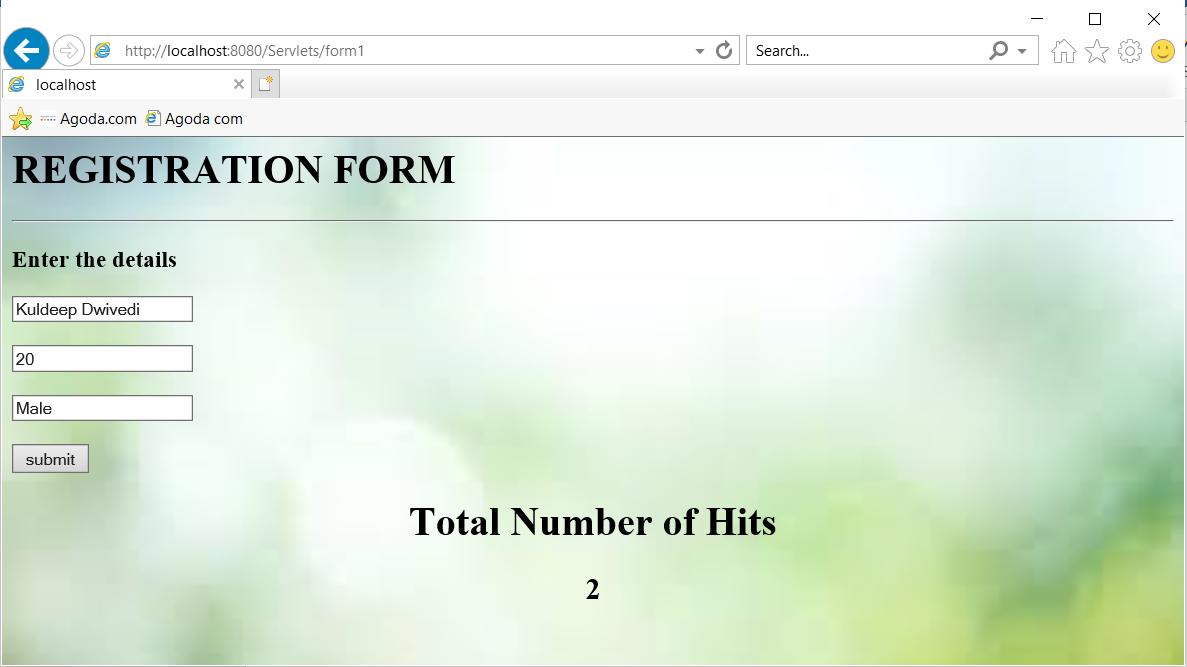
out.println("VALID USER!!" );

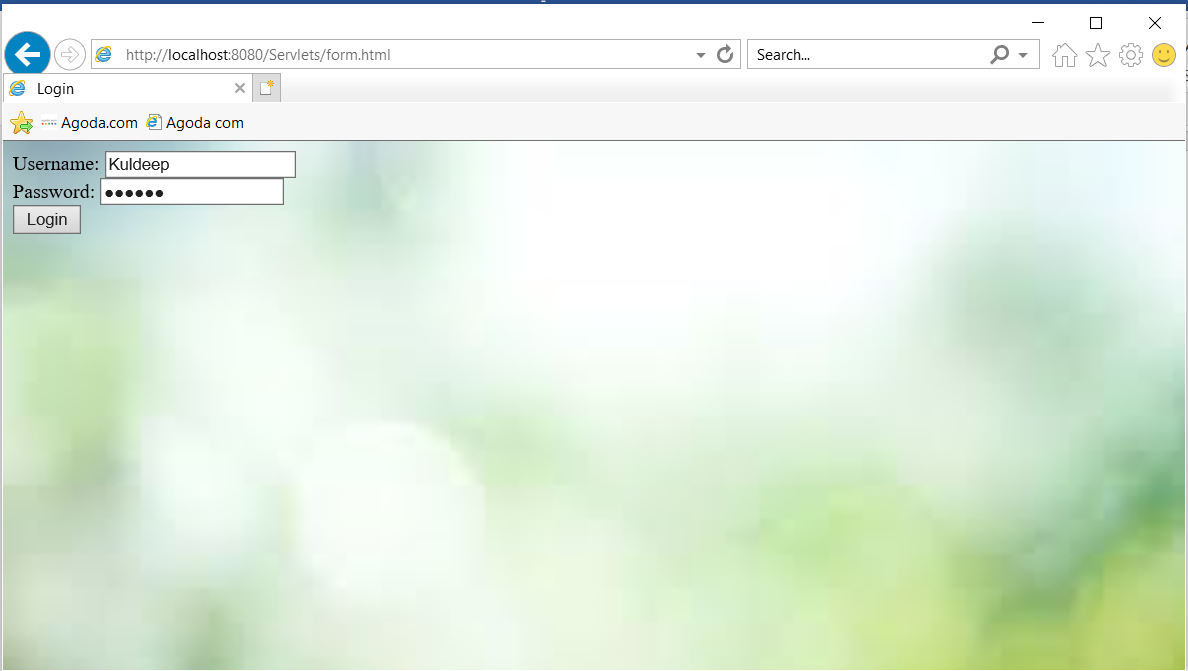
out.print(" Welcome " + n);

} }

**Output:**

****

****

****

Bottom of Form

Bottom of Form

Bottom of Form

Date:15/2/2021

**Experiment 6**

**Objective:**

Write a program to get input from user and print a welcome message using JSP.

**Theory:**

**Java Server Pages** is a server-side web technology which is used to create web applications. JSP pages are the text documents which contain two types of text namely static content and dynamic content. Static content can be expressed in any text-based format like HTML whereas the dynamic content comprises of the Java code. JSP technology here combines the static content with the Java code, hence making it a dynamic web page.

**Features of JSP technology:**

* JSP allows tag-based programming. Hence, there is no need for expertise in the Java language. HTML tags are easy to use and makes the code readable.
* The building of dynamic web pages that are able to interact with the users in a real-time environment.
* It allows user an easy connection to the database as it mainly connects with the server.

The file extension for the source file of a JSP page is .jsp.

**Code :**

**first.jsp:**

<%@ page language="Java" %>

<html>

<head>

<title>

Input Form

</title>

</head>

<body>

<form method = "post" action ="second.jsp">

Enter your Name :

<Input type= "text" name= "username">

Enter your Email :

<Input type="text" Email="email">

<input type = "submit" value= "Submit">

</form>

</body>

</html>

**second.jsp:**

<%@ page language= "Java" %>

<% String name =request.getParameter ("username");

if (name==null)

name=" ";

session.setAttribute("username", name);

%>

<% String email=request.getParameter ("email");

if(email==null)

email = " ";

session.setAttribute("Email", email);

%>

<body>

<a href = "third.jsp">

Click on Next page </a>

</body>

</html>

**third.jsp:**

<%@ page language="Java" %>

<%String uname=(String)session.getAttribute("username");

if(uname==null)

uname=" ";

%>

<html>

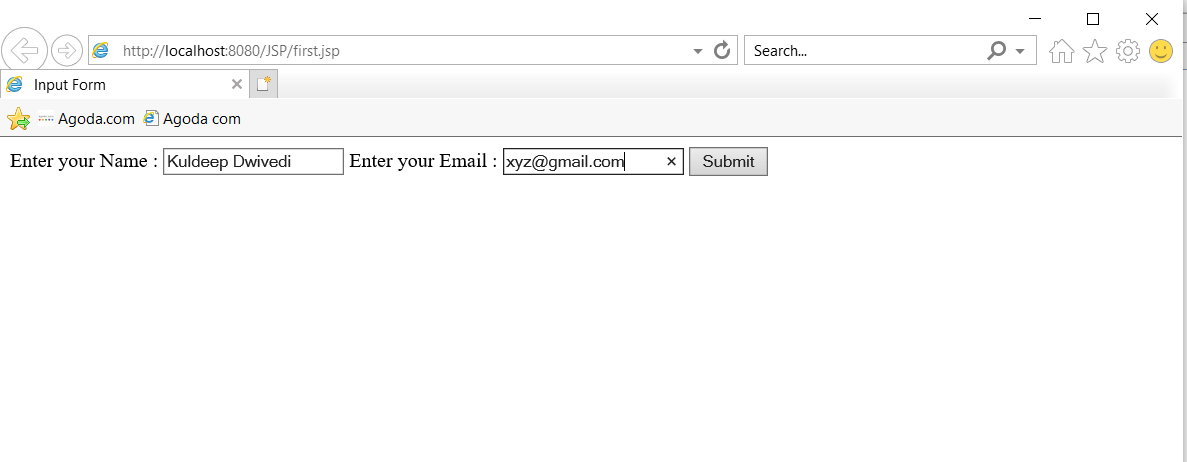
<body>

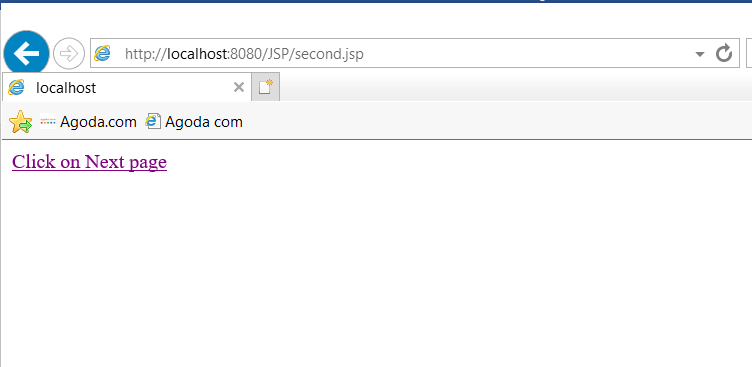
Welcome <%=uname%>

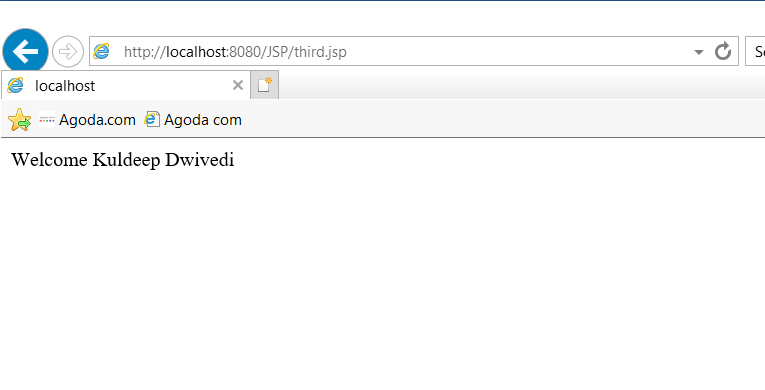
</body>

</html>

**Output :**







Date:15/2/2021

**Experiment 7**

**Objective:**

Write a program to create a Loan Calculator which can compute Simple and Compound interest using JSP.

**Theory:**

**Java Server Pages** is a server-side web technology which is used to create web applications. JSP pages are the text documents which contain two types of text namely static content and dynamic content. Static content can be expressed in any text-based format like HTML whereas the dynamic content comprises of the Java code. JSP technology here combines the static content with the Java code, hence making it a dynamic web page.

**Features of JSP technology:**

* JSP allows tag-based programming. Hence, there is no need for expertise in the Java language. HTML tags are easy to use and makes the code readable.
* The building of dynamic web pages that are able to interact with the users in a real-time environment.
* It allows user an easy connection to the database as it mainly connects with the server.

The file extension for the source file of a JSP page is .jsp.

**Code:**

**index.jsp:**

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

<!DOCTYPE html>

<html>

<head>

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

<title>include</title>

</head>

<body style = "font-family:verdana; font-size: 10pt;">

<%@include file = "header.jsp" %>

<form action = "controller.jsp">

<table border = "0" style = "font-family:verdana; font-size: 10pt;">

<tr>

<td>Amount:</td>

<td><input type = "text" name = "amount"/></td>

</tr>

<tr>

<td>Interest in %:</td>

<td><input type = "text" name = "interest"/></td>

</tr>

<tr>

<td>Compound:</td>

<td><input type = "radio" name = "type" value="C" checked/></td>

</tr>

<tr>

<td>Simple:</td>

<td><input type ="radio" name = "type" value ="S"/></td>

</tr>

<tr>

<td>Period:</td>

<td><input type="text" name ="period"/></td>

</tr>

</table>

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

</form>

<jsp:include page="footer.jsp"/>

</body>

</html>

**controller.jsp:**

<%

String type = request.getParameter("type");

if(type.equals("S")){

%>

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

<%

} else {

%>

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

<%

}

%>

**header.jsp:**

<h3>Loan Calculator</h3>

**footer.jsp:**

<%= new java.util.Date()%>

**error.jsp:**

<%@page isErrorPage = "true"%>

<html>

<head>

<title>Simple</title>

</head>

<body style ="font-family: verdana;font-size:10pt;">

<%@ include file="header.jsp"%>

<p><b><%= exception.getMessage()%></b></p>

<jsp:include page = "footer.jsp"/>

</body>

</html>

**simple.jsp:**

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

<%!

public double calculate(double amount, double interest, int period){

if(amount <= 0){

throw new IllegalArgumentException("Amount should be greater than 0:"+ amount);

}

if(interest <= 0){

throw new IllegalArgumentException("Interest should be greater than 0:"+ interest);

}

if(period <= 0){

throw new IllegalArgumentException("Period should be greater than 0:"+ period);

}

return amount\*(1+period\*interest/100);

}

%>

<html>

<head>

<title>Simple</title>

</head>

<body style="font-family: verdana; font-size: 10pt;">

<%@include file="header.jsp" %>

<%

double amount = Double.parseDouble(request.getParameter("amount"));

double interest = Double.parseDouble(request.getParameter("interest"));

int period = Integer.parseInt(request.getParameter("period"));

%>

<b>Principal using Simple Interest:</b>

<%= calculate(amount,interest,period)%>

<br/><br/>

<jsp:include page = "footer.jsp"/>

</body>

</html>

**compound.jsp:**

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

<%!

public double calculate(double amount, double interest, int period){

if(amount <= 0){

throw new IllegalArgumentException("Amount should be greater than 0:"+ amount);

}

if(interest <= 0){

throw new IllegalArgumentException("Interest should be greater than 0:"+ interest);

}

if(period <= 0){

throw new IllegalArgumentException("Period should be greater than 0:"+ period);

}

return amount\*Math.pow(1+interest/100,period);

}

%>

<html>

<head>

<title>Compound</title>

</head>

<body style="font-family: verdana; font-size: 10pt;">

<%@include file="header.jsp" %>

<%

double amount = Double.parseDouble(request.getParameter("amount"));

double interest = Double.parseDouble(request.getParameter("interest"));

int period = Integer.parseInt(request.getParameter("period"));

%>

<b>Principal using compound Interest:</b>

<%= calculate(amount,interest,period)%>

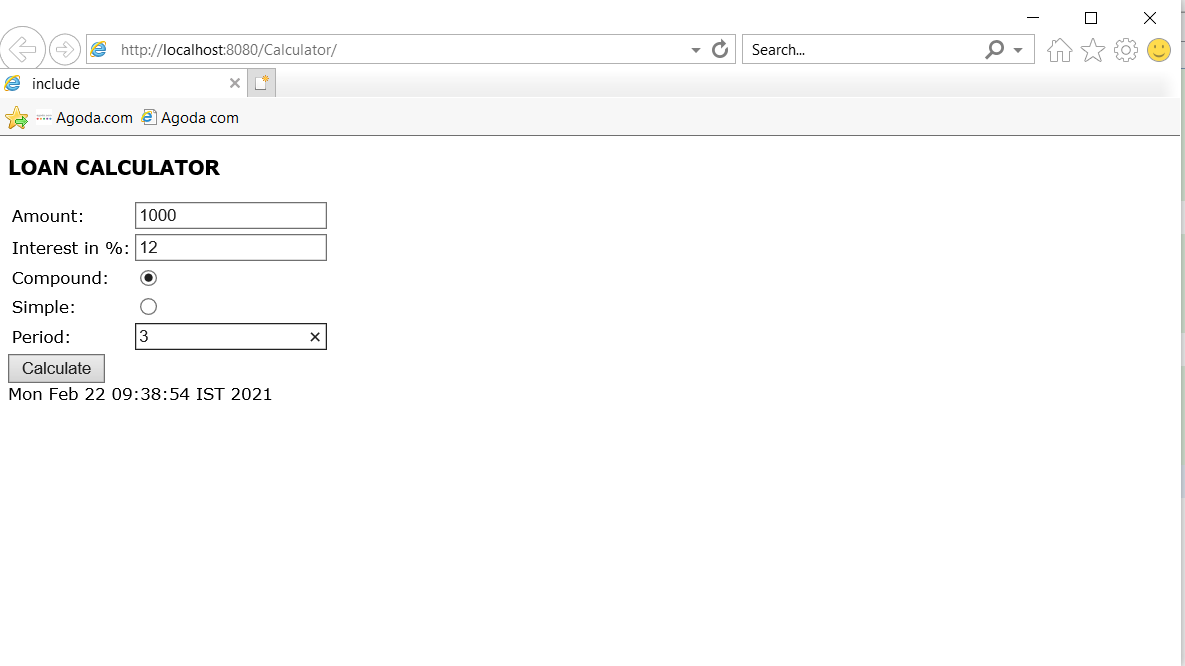
<br/><br/>

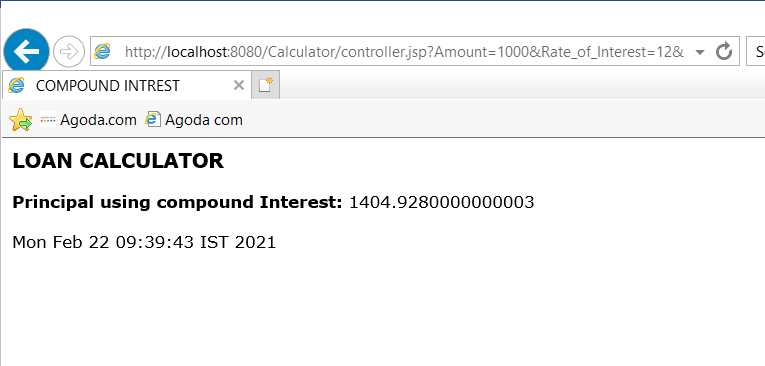
<jsp:include page = "footer.jsp"/>

</body>

</html>

**Output :**

****

****

DATE: 22/2/2021

**Experiment 8**

**Objective:**

Write a program to get input from user using servlet and connect it with a database.

**Theory:**

**Servlets:** Servlets are the Java programs that runs on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.

**Properties of Servlets:**

* Servlets work on the server-side.
* Servlets are capable of handling complex requests obtained from web server.

**PreparedStatement:** Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object. This statement gives you the flexibility of supplying arguments dynamically.

**Code:**

**index.html:**

<html>

<head>

<title>Register Form</title>

</head>

<body>

<form method= "post" action = '<%=request.getContextPath()%>/Register'>

Name: <input type ="text" name="name"/><br/>

Email ID: <input type="text" name ="email"/><br/>

Password: <input type ="text" name ="pass"/><br/>

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

</form>

</body>

</html>

**Register.java:**

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.sql.\*;

public class Register extends HttpServlet {

protected void processRequest(HttpServletRequest req, HttpServletResponse res)

throws ServletException, IOException {

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

try (PrintWriter out = res.getWriter()) {

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

String name = req.getParameter("name");

String email = req.getParameter("email");

String pass = req.getParameter("pass");

try {

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

Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/Stud");

PreparedStatement ps = conn.prepareStatement("INSERT INTO Student VALUE (?,?,?)");

ps.setString(1, name);

ps.setString(2, email);

ps.setString(3, pass);

int i = ps.executeUpdate();

if (i > 0) {

out.println("Successfully Registered");

}

} catch (ClassNotFoundException | SQLException e) {

System.out.println(e);

}

}

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException{

processRequest(req, res);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

**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">

<servlet>

<servlet-name>Register</servlet-name>

<servlet-class>C:\Users\hp\Documents\NetBeansProjects\DB\_Servlet\src\java\Register.java</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Register</servlet-name>

<url-pattern>/Register</url-pattern>

</servlet-mapping>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

**Output:**

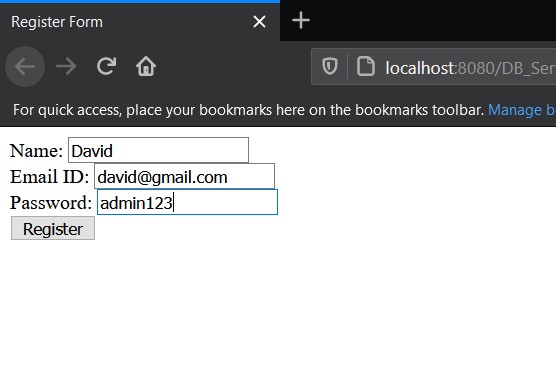


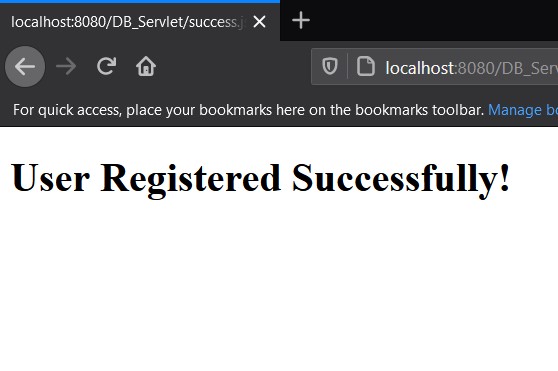
Figure 8 **HomePage.**

Figure 9 **Confirmation Message after the registration is done.**

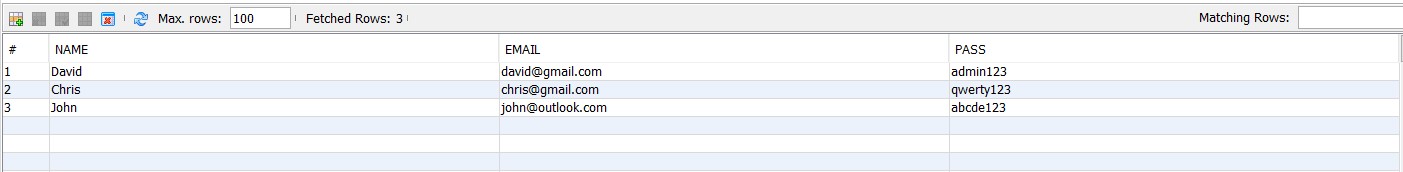


Figure 10 **Entries added to the database.**

DATE: 1/3/2021

**Experiment 9**

**Objective:**

Write a program to print 10 numbers, get input from user, calculating the salary and handling the error using Scriptlet.

**Theory:**

**Java Server Pages** is a server-side web technology which is used to create web applications. JSP pages are the text documents which contain two types of text namely static content and dynamic content. Static content can be expressed in any text-based format like HTML whereas the dynamic content comprises of the Java code. JSP technology here combines the static content with the Java code, hence making it a dynamic web page.

**JSP Scriptlet Tags:** The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

* scriptlet tag
* expression tag
* declaration tag

**Syntax:** *<%* ***Java Source Code****%>*

**Code:**

**JSPTags.jsp:**

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

<!DOCTYPE html>

<html>

<head>

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

<title>10 Numbers</title>

</head>

<body>

<form method="Post" action="print.jsp">

<% for(int i = 1; i <= 10; i++){%>

<p>Number = <%=i%></p>

<%}%>

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

</form>

</body>

</html>

**print.jsp:**

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

<!DOCTYPE html>

<html>

<head>

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

<title>Enter values</title>

</head>

<body style = "font-family:verdana; font-size: 10pt;">

<form method = "post" action ="Calculate.jsp">

<table border = "1" style = "font-family:verdana; font-size: 10pt; border-collapse: collapse; padding: 15px;">

<tr>

<th>Name:</th>

<td><input type = "text" name = "name"/></td>

</tr>

<tr>

<th>Basic Salary:</th>

<td><input type = "text" name = "basic"/></td>

</tr>

<tr>

<th>HR Allowance:</th>

<td><input type = "text" name = "hr"/></td>

</tr>

<tr>

<th>Travel Allowance:</th>

<td><input type = "text" name = "ta"/></td>

</tr>

<tr>

<th>Provident Fund(in %):</th>

<td><input type ="text" name="pf"/></td>

</tr>

</table>

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

</form>

</body>

</html>

**Calculate.jsp:**

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

<%!

public double total(double basic, double hr, double ta, double pf){

if(basic <= 0){

throw new IllegalArgumentException("Amount should be greater than 0:"+ basic);

}

if(hr <= 0){

throw new IllegalArgumentException("Interest should be greater than 0:"+ hr);

}

if(ta <= 0){

throw new IllegalArgumentException("Period should be greater than 0:"+ ta);

}

if(pf <= 0){

throw new IllegalArgumentException("Period should be greater than 0:"+ pf);

}

double total = basic + hr + ta;

total -= total\*0.01\*pf;

return total;

}

%>

<html>

<head>

<title>Salary Calculate</title>

</head>

<style>

h1{text-align: center;}

p{text-align: center;}

table.center{

margin-left: auto;

margin-right: auto;

}

</style>

<body style="font-family: verdana; font-size: 10pt;">

<%

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

double basic = Double.valueOf(request.getParameter("basic"));

double hr = Double.valueOf(request.getParameter("hr"));

double ta = Double.valueOf(request.getParameter("ta"));

double pf = Double.valueOf(request.getParameter("pf"));

double gross = basic+hr+ta;

%>

<h1>ABC Company</h1>

<p>Hello <%=name%></p>

<table class="center" border = "1" style = "font-family:verdana; font-size: 10pt; border-collapse: collapse; width: 50%; background-color: #cccccc;">

<tr style="background-color: #cccccc">

<th>Earnings and Deductions</th>

<th>Amount</th>

</tr>

<tr style="background-color: #ffffff">

<th>Basic Salary</th>

<td><%=basic%></td>

</tr>

<tr style="background-color: #cccccc">

<th>HR Allowance</th>

<td><%=hr%></td>

</tr>

<tr style="background-color: #ffffff">

<th>Travel Allowance</th>

<td><%=ta%></td>

</tr>

<tr style="background-color: #cccccc">

<th>PF Deduction(in %)</th>

<td><%=pf%></td>

</tr>

<tr style="background-color: #ffffff">

<th>Gross Salary</th>

<td><%=gross%></td>

</tr>

<tr style="background-color: #cccccc">

<th>Net Salary</th>

<td><%=total(basic,hr,ta,pf)%></td>

</tr>

</table>

</body>

</html>

**error.jsp:**

<%@page isErrorPage = "true"%>

<html>

<head>

<title>Error Page</title>

</head>

<body style ="font-family: verdana;font-size:10pt;">

<p><b><%= exception.getMessage()%></b></p>

</body>

</html>

**Output:**

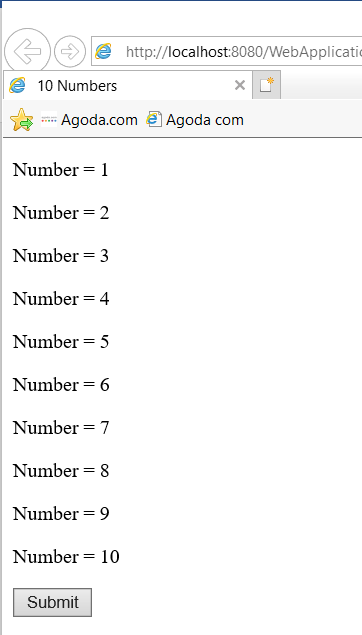


Figure 11 **Printing first 10 numbers using JSP Scriptlet tag.**

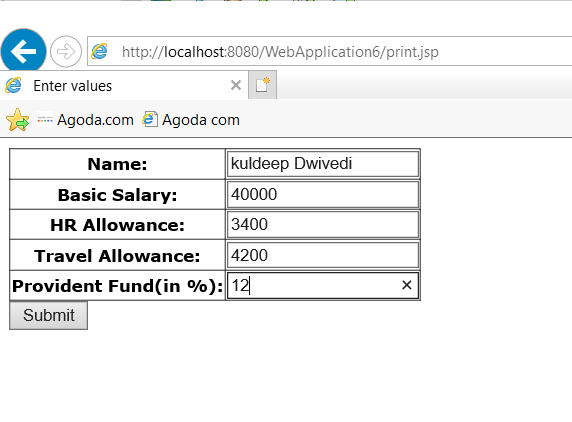


Figure 12 **Taking inputs from user for calculating salary.**

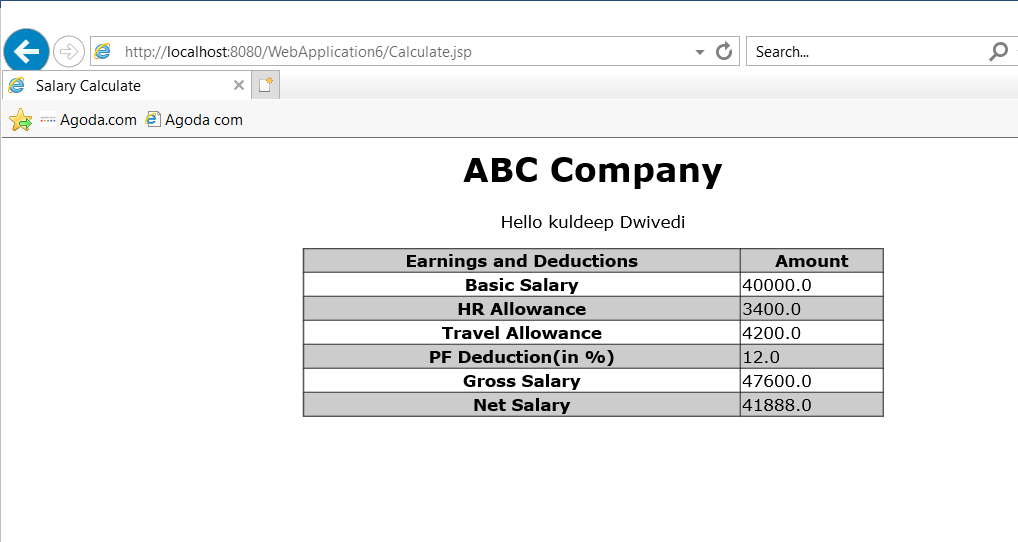


Figure 13 **Printing the calculated and other values.**



Figure 14 **Error Page.**

DATE: 8/3/2021

**Experiment 10**

**Objective:**

Write a program to create a Calculator using stateful Enterprise JavaBeans.

**Theory:** Enterprise Java Beans (EJB) is one of the several Java APIs for standard manufacture of enterprise software. EJB is a server-side software element that summarizes business logic of an application. Enterprise Java Beans web repository yields a runtime domain for web related software elements including computer reliability, Java Servlet Lifecycle (JSL) management, transaction procedure and other web services

**1. *Session Bean:*** Session bean contains business logic that can be invoked by local, remote or webservice client. There are two types of session beans: (i) Stateful session bean and (ii) Stateless session bean.

* **(i) Stateful Session bean :**  
  Stateful session bean performs business task with the help of a state. Stateful session bean can be used to access various method calls by storing the information in an instance variable. Some of the applications require information to be stored across separate method calls. In a shopping site, the items chosen by a customer must be stored as data is an example of stateful session bean.
* **(ii) Stateless Session bean :**  
  Stateless session bean implement business logic without having a persistent storage mechanism, such as a state or database and can used shared data. Stateless session bean can be used in situations where information is not required to used across call methods.

**Code:**

**index.jsp:**

<html>

<head>

<title>Ejb3 Stateless Tutorial</title>

</head>

<body bgcolor="#FFFFCC">

<p align="center"><font size="6" color="#800000"><b>Welcome to <br>

Ejb3-Jboss 4.2.0 Tutorial</b></font>

Click <a href="Form.jsp">Calculator Example</a> to execute Calculator<br></p>

</body>

</html>

**Form.jsp:**

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

<!DOCTYPE html>

<html>

<head>

<title>Calculator</title>

</head>

<body bgcolor="pink">

<h1>Calculator</h1>

<hr>

<form action="WebClient.jsp" method="POST">

<p>Enter first value:

<input type="text" name="num1" size="25"></p>

<br>

<p>Enter second value:

<input type="text" name="num2" size="25"></p>

<br>

<b>Select your choice:</b><br>

<input type="radio" name="group1" value ="add">Addition<br>

<input type="radio" name="group1" value ="sub">Subtraction<br>

<input type="radio" name="group1" value ="multi">Multiplication<br>

<input type="radio" name="group1" value ="div">Division<br>

<p>

<input type="submit" value="Submit">

<input type="reset" value="Reset"></p>

</form>

</body>

</html>

**WebClient.jsp:**

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

<!DOCTYPE html>

<%@ page import="com.test.\*,

javax.naming.\*"%>

<%!

private NewSessionBeanLocal calculator = null;

float result=0;

public void jspInit() {

try {

InitialContext ic = new InitialContext();

calculator = (NewSessionBeanLocal)ic.lookup("java:global/EJB/NewSessionBean");

System.out.println("Loaded Calculator Bean");

//CalculatorBean

} catch (Exception ex) {

System.out.println("Error:"+

ex.getMessage());

}

}

public void jspDestroy() {

calculator = null;

}

%>

<%

try {

String s1 = request.getParameter("num1");

String s2 = request.getParameter("num2");

String s3 = request.getParameter("group1");

System.out.println(s3);

if ( s1 != null && s2 != null ) {

Float num1 = new Float(s1);

Float num2 = new Float(s2);

if(s3.equals("add"))

result=calculator.add(num1.floatValue(),

num2.floatValue());

else if(s3.equals("sub"))

result=calculator.subtract(num1.floatValue(),

num2.floatValue());

else if(s3.equals("multi"))

result=calculator.multiply(num1.floatValue(),

num2.floatValue());

else

result=calculator.division(num1.floatValue(),

num2.floatValue());

%>

<p>

<b>The result is:</b> <%= result %>

<p>

<%

}

}// end of try

catch (Exception e) {

e.printStackTrace ();

//result = "Not valid";

}

%>

**NewSessionBean.java:**

import javax.ejb.Stateless;

@Stateless

public class NewSessionBean implements NewSessionBeanLocal {

public float add(float x, float y){

return x + y;

}

public float subtract(float x, float y){

return x - y;

}

public float multiply(float x, float y){

return x \* y;

}

public float division(float x, float y){

return x / y;

}

}

**NewSessionBeanLocal.java:**

import javax.ejb.Local;

@Local

public interface NewSessionBeanLocal {

public float add(float x, float y);

public float subtract(float x, float y);

public float multiply(float x, float y);

public float division(float x, float y);

}

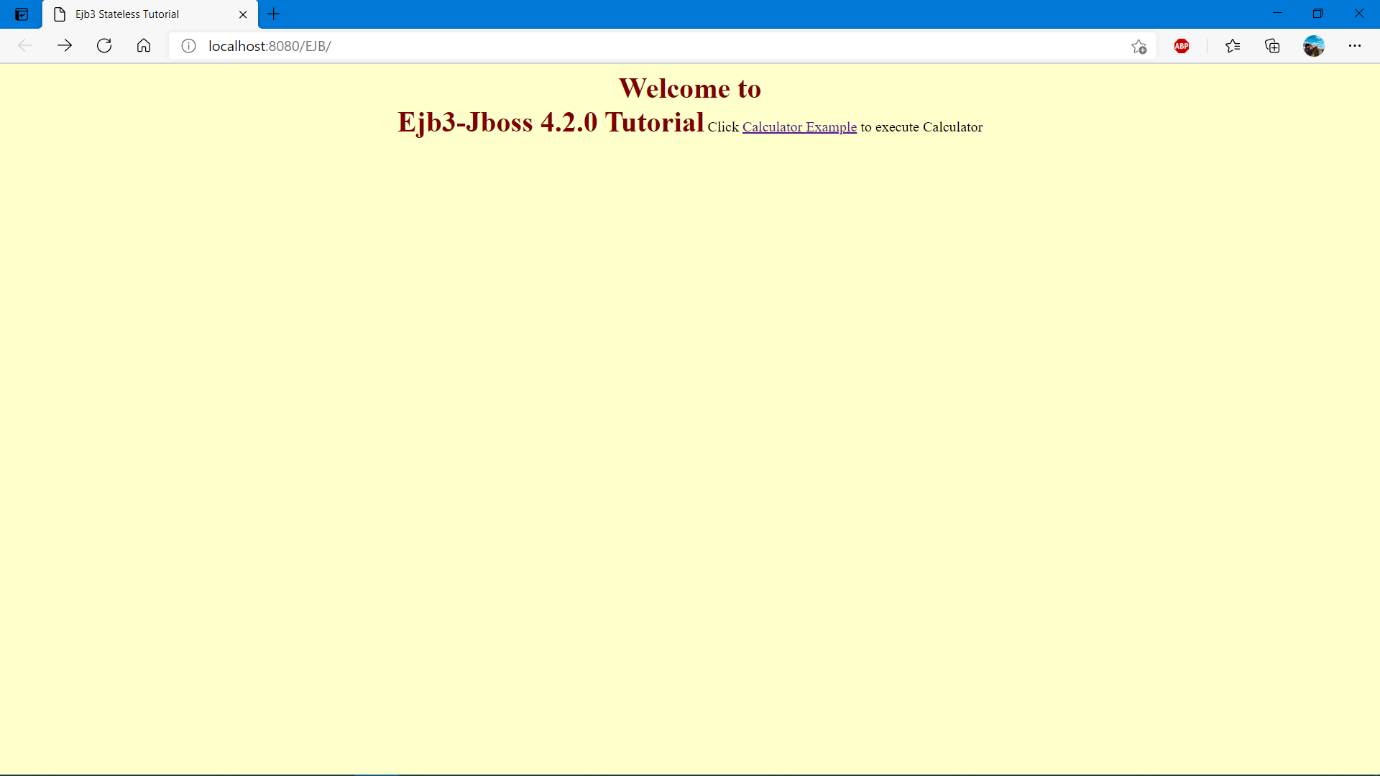
**Output:**

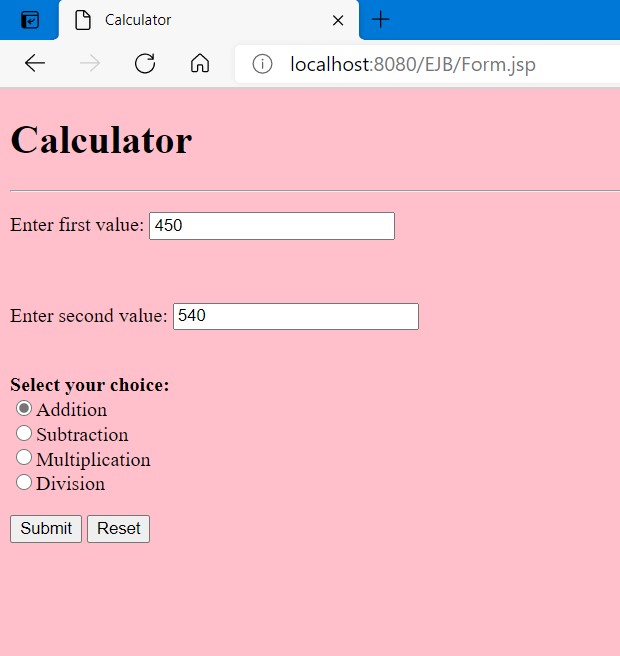
Figure 15 **Welcome Page**

Figure 16 **User will enter values and select the operation**

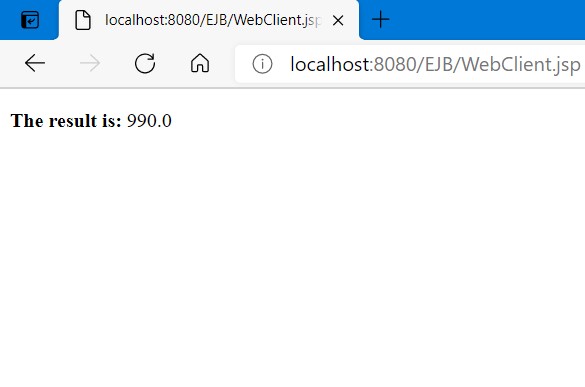


Figure 17 **Result of the operation.**

DATE: 8/3/2021

**Experiment 11**

**Objective:**

Write a program to develop a deposit and withdrawal form using stateful EJB.

**Theory:**

**EJB**: EJB stands for **Enterprise** **Java** **Beans**. EJB is an essential part of a J2EE platform. J2EE platform has component-based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.

EJB provides an architecture to develop and deploy component-based enterprise applications considering robustness, high scalability, and high performance.

EJB is primarily divided into three categories:

* **Session Bean:** Session bean stores data of a particular user for a single session. It can be stateful or stateless. It is less resource intensive as compared to entity bean. Session bean gets destroyed as soon as user session terminates.
* **Entity Bean:** Entity beans represent persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.
* **Message-Driven Bean:** Message driven beans are used in context of JMS (Java Messaging Service). Message Driven Beans can consume JMS messages from external entities and act accordingly.

**Stateful Session Bean:** A stateful session bean is a type of enterprise bean, which preserve the conversational state with client. A stateful session bean as per its name keeps associated client state in its instance variables. EJB Container creates a separate stateful session bean to process client's each request. As soon as request scope is over, stateful session bean is destroyed.

**Code:**

**Index.jsp:**

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

<!DOCTYPE html>

<html>

<head>

<title>Ejb3 Stateful Tutorial</title>

</head>

<body bgcolor="#FFFFCC">

<p align="center"><font size="6" color="#800000"><b>Welcome to <br>

Ejb3-Jboss 4.2.0 Tutorial</b></font>

Click <a href="Form.jsp">Bank Transaction Example</a> to execute Bank Bean<br></p>

</body>

</html>

**Form.jsp:**

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

<html>

<head>

<title>Bank Account</title>

</head>

<body>

<h1><p align="center"><font size="6" color="#800000">Bank Transaction Request Form</h1>

<hr><br>

<table bgcolor="#FFFFCC" align="center">

<form action="WebClient.jsp" method="POST">

<tr><td></tr></td>

<tr><td>Enter the amount in rupees:

<input type="text" name="amt" size="10"></tr></td>

<br>

<tr><td><b>Select your choice:</b></tr></td>

<tr><td><input type="radio" name="group1" value ="dep">Deposit</tr></td>

<tr><td><input type="radio" name="group1" value ="with">Withdraw<br></tr></td>

<tr><td>

<input type="submit" value="Transmit">

<input type="reset" value="Reset"></tr></td>

<tr><td></tr></td>

</form>

</table>

</body>

</html>

**WebClient.jsp:**

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

<%@ page import="com.test.ex.\*,

javax.naming.\*"%>

<%!

public NewSessionBeanLocal account = null;

float bal=0;

public void jspInit() {

try {

InitialContext ic = new InitialContext();

account = (NewSessionBeanLocal) ic.lookup("java:global/EJB\_Stateful/NewSessionBean");

System.out.println("Loaded Account Bean");

}

catch (Exception ex) {

System.out.println("Error:"+

ex.getMessage());

}

}

public void jspDestroy() {

account = null;

}

%>

<%

try {

String s1 = request.getParameter("amt");

String s2 = request.getParameter("group1");

if ( s1 != null) {

Float amt = new Float(s1);

if(s2.equals("dep"))

bal=account.deposit(amt.floatValue());

else if(s2.equals("with"))

bal=account.withdraw(amt.floatValue());

else

%>

<p>Please select your choice</p>

<%

}

else

%>

<br>Please enter the amount<br>

<p>

The Transaction is complete<br>

<b>Your Current Balance is:</b> <%= bal%>

<p>

<%

}// end of try

catch (Exception e) {

e.printStackTrace ();

}

%>

**NewSessionBean.java:**

package com.test.ex;

import javax.ejb.Stateful;

@Stateful

public class NewSessionBean implements NewSessionBeanLocal {

float balance = 0;

public float deposit(float amount){

balance += amount;

return balance;

}

public float withdraw(float amount){

balance -= amount;

return balance;

}

public void remove() {

balance = 0;

}

}

**NewSessionBeanLocal.java:**

package com.test.ex;

import javax.ejb.Local;

@Local

public interface NewSessionBeanLocal {

public float deposit(float amount);

public float withdraw(float amount);

public void remove();

}

**Output:**

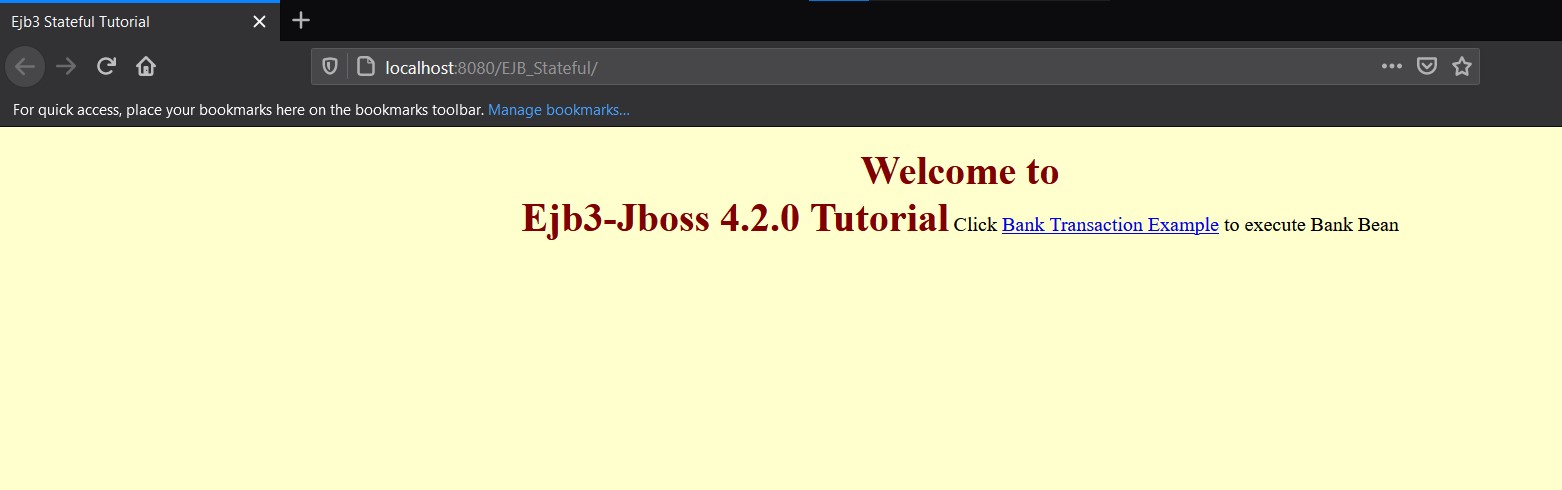


Figure 21 **HomePage.**

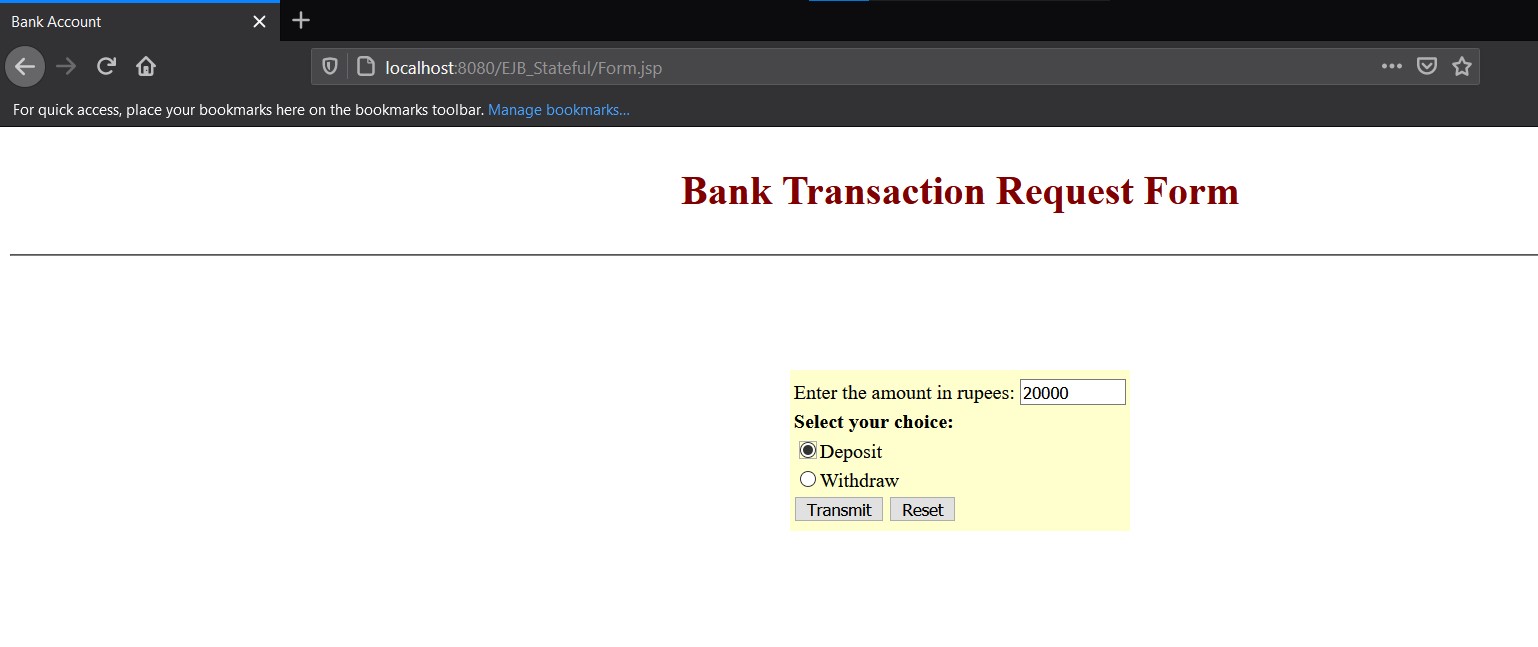


Figure 22 **User gives the input.**

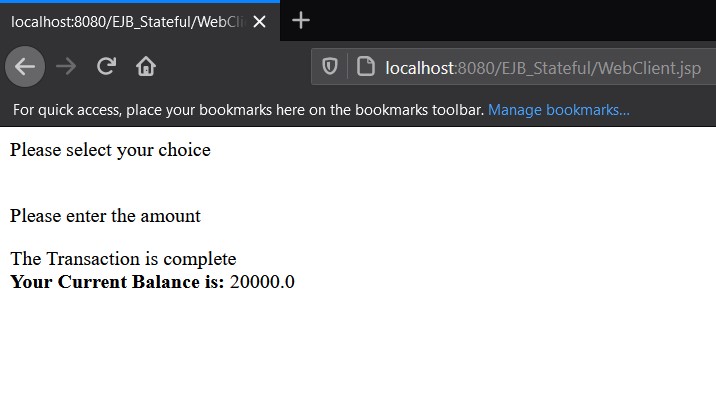


Figure 23 **Confirmation Page.**

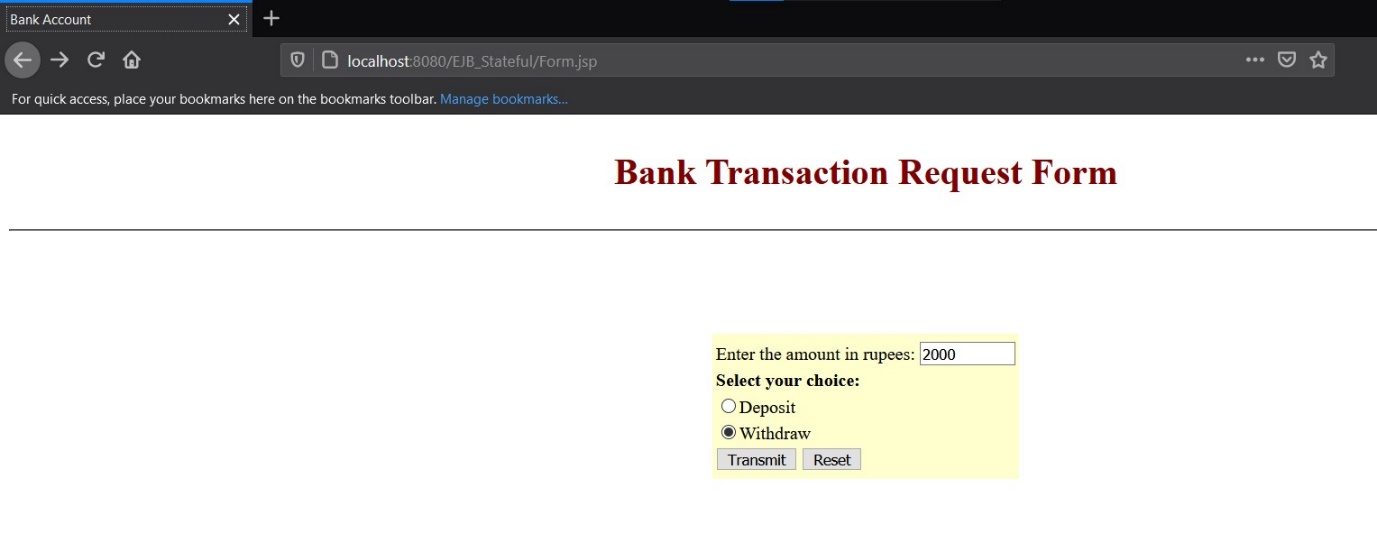


Figure 24 **User can withdraw money as well and system will keep track of the transaction.**

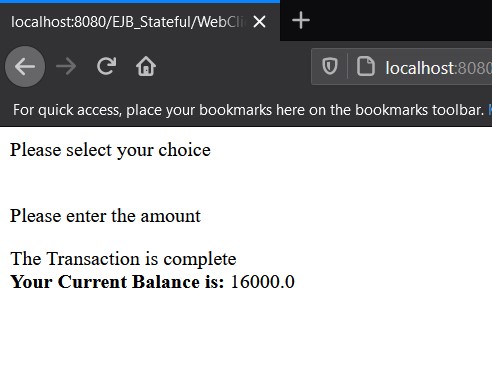


Figure 25 **Final Page.**

08/03/2021

**Experiment 12**

**Objective :** Write a program to implement Entity Bean

**Theory:** An entity bean is a remote object that manages persistent data, performs complex business logic, potentially uses several dependent Java objects, and can be uniquely identified by a primary key. Entity beans are normally coarse-grained persistent objects, in that they utilize persistent data stored within several fine-grained persistent Java objects.

### When to Use Entity Beans

You should probably use an entity bean under the following conditions:

* The bean represents a business entity, not a procedure. For example, CreditCardEJB would be an entity bean, but CreditCardVerifierEJB would be a session bean.
* The bean's state must be persistent. If the bean instance terminates or if the J2EE server is shut down, the bean's state still exists in persistent storage (a database).

**Code:**

**index.html**

<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 = "Servlet">

<input type="text" name="enroll" value="" />

<input type="text" name="name" value="" />

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

</form>

</body>

</html>

**Servlet.java**

/\*

\* 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.

\*/

import Student.info;

import Student.infoFacadeLocal;

import java.io.IOException;

import java.io.PrintWriter;

import javax.ejb.EJB;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\*

\* @author RexTire

\*/

public class Servlet extends HttpServlet {

@EJB

private infoFacadeLocal infoFacade;

info std = new info();

/\*\*

\* Processes requests for both HTTP <code>GET</code> and <code>POST</code>

\* methods.

\*

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

try (PrintWriter out = response.getWriter()) {

/\* TODO output your page here. You may use following sample code. \*/

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet Servlet</title>");

out.println("</head>");

out.println("<body>");

std.setENROLL(request.getParameter("enroll"));

std.setNAME(request.getParameter("name"));

infoFacade.create(std);

out.println("Data Inserted Sucessfully!");

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

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

}

}

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">

/\*\*

\* Handles the HTTP <code>GET</code> method.

\*

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

/\*\*

\* Handles the HTTP <code>POST</code> method.

\*

\* @param request servlet request

\* @param response servlet response

\* @throws ServletException if a servlet-specific error occurs

\* @throws IOException if an I/O error occurs

\*/

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

/\*\*

\* Returns a short description of the servlet.

\*

\* @return a String containing servlet description

\*/

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

**info.java**

/\*

\* 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.

\*/

package Student;

import java.io.Serializable;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

/\*\*

\*

\* @author RexTire

\*/

@Entity

public class info implements Serializable {

private String ENROLL;

private String NAME;

public String getENROLL() {

return ENROLL;

}

public void setENROLL(String ENROLL) {

this.ENROLL = ENROLL;

}

public String getNAME() {

return NAME;

}

public void setNAME(String NAME) {

this.NAME = NAME;

}

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

@Override

public int hashCode() {

int hash = 0;

hash += (id != null ? id.hashCode() : 0);

return hash;

}

@Override

public boolean equals(Object object) {

// TODO: Warning - this method won't work in the case the id fields are not set

if (!(object instanceof info)) {

return false;

}

info other = (info) object;

if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {

return false;

}

return true;

}

@Override

public String toString() {

return "Student.info[ id=" + id + " ]";

}

}

**infoFacade.java**

/\*

\* 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.

\*/

package Student;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

/\*\*

\*

\* @author RexTire

\*/

@Stateless

public class infoFacade extends AbstractFacade<info> implements infoFacadeLocal {

@PersistenceContext(unitName = "bean-ejbPU")

private EntityManager em;

@Override

protected EntityManager getEntityManager() {

return em;

}

public infoFacade() {

super(info.class);

}

}

**AbstractFacade.java**

/\*

\* 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.

\*/

package Student;

import java.util.List;

import javax.persistence.EntityManager;

/\*\*

\*

\* @author RexTire

\*/

public abstract class AbstractFacade<T> {

private Class<T> entityClass;

public AbstractFacade(Class<T> entityClass) {

this.entityClass = entityClass;

}

protected abstract EntityManager getEntityManager();

public void create(T entity) {

getEntityManager().persist(entity);

}

public void edit(T entity) {

getEntityManager().merge(entity);

}

public void remove(T entity) {

getEntityManager().remove(getEntityManager().merge(entity));

}

public T find(Object id) {

return getEntityManager().find(entityClass, id);

}

public List<T> findAll() {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

cq.select(cq.from(entityClass));

return getEntityManager().createQuery(cq).getResultList();

}

public List<T> findRange(int[] range) {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

cq.select(cq.from(entityClass));

javax.persistence.Query q = getEntityManager().createQuery(cq);

q.setMaxResults(range[1] - range[0] + 1);

q.setFirstResult(range[0]);

return q.getResultList();

}

public int count() {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

javax.persistence.criteria.Root<T> rt = cq.from(entityClass);

cq.select(getEntityManager().getCriteriaBuilder().count(rt));

javax.persistence.Query q = getEntityManager().createQuery(cq);

return ((Long) q.getSingleResult()).intValue();

}

}

**infoFacadeLocal.java**

/\*

\* 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.

\*/

package Student;

import java.util.List;

import javax.ejb.Local;

/\*\*

\*

\* @author RexTire

\*/

@Local

public interface infoFacadeLocal {

void create(info info);

void edit(info info);

void remove(info info);

info find(Object id);

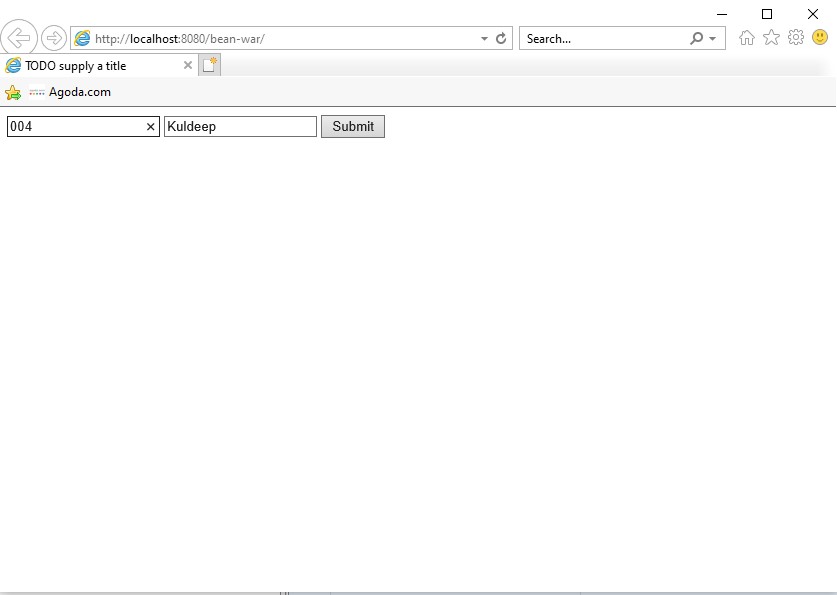
List<info> findAll();

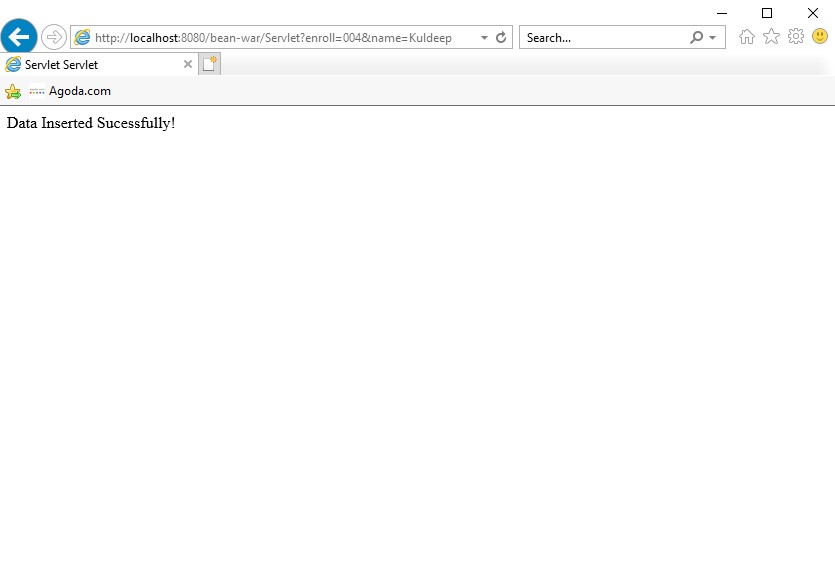
List<info> findRange(int[] range);

int count();

}

**Output:**





Date:22/03/2020

**Experiment 13**

**Objective:**

Write a program to implement Struts.

**Theory:**

**Struts** is used to create a web applications based on servlet and JSP. Struts depend on the MVC (Model View Controller) framework. Struts application is a genuine web application. Struts are thoroughly useful in building J2EE (Java 2 Platform, Enterprise Edition) applications because struts takes advantage of J2EE design patterns. Struts follows these J2EE design patterns including MVC.

**Features of Struts:** Struts has the following features:

* Struts encourages good design practices and modeling because the framework is designed with “time-proven” design patterns.
* Struts is almost simple, so easy to learn and use.
* It supports many convenient features such as input validation and internationalization.
* It takes much of the complexity out as instead of building your own MVC framework, you can use struts.
* Struts is very well integrated with J2EE.
* Struts has large user community.

**Code:**

**Index.jsp**

<%@page contentType="text/html"%>

<%@page pageEncoding="UTF-8"%>

<jsp:forward page="Welcome.do"/>

**Login.jsp**

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

<!DOCTYPE html>

<html>

<head>

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

<title>Login Page</title>

</head>

<body>

<h1>Login Form</h1>

<form action="loginform.do" method="post">

Username <input type="text" name="uname"/><br/>

Password <input type="text" name="upass"/><br/>

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

</form>

</body>

</html>

**Success.jsp**

<%@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>

<h1>Login Successful!</h1>

</body>

</html>

**Failure.jsp**

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

<!DOCTYPE html>

<html>

<head>

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

<title>Failure Page</title>

</head>

<body>

<h1>Login Failed!</h1>

<p>Try again!</p>

</body>

</html>

**Loginbean.java**

package com.myapp.struts;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;

import org.apache.struts.action.ActionMapping;

import org.apache.struts.action.ActionMessage;

/\*\*

\*

\* @author hp

\*/

public class loginbean extends org.apache.struts.action.ActionForm {

private String uname;

private String upass;

public String getUname() {

return uname;

}

public void setUname(String uname) {

this.uname = uname;

}

public String getUpass() {

return upass;

}

public void setUpass(String upass) {

this.upass = upass;

}

public loginbean() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* This is the action called from the Struts framework.

\*

\* @param mapping The ActionMapping used to select this instance.

\* @param request The HTTP Request we are processing.

\* @return

\*/

public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {

ActionErrors errors = new ActionErrors();

return errors;

}

}

**Loginform.java**

package com.myapp.struts;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.ActionForm;

import org.apache.struts.action.ActionForward;

import org.apache.struts.action.ActionMapping;

/\*\*

\*

\* @author hp

\*/

public class loginform extends org.apache.struts.action.Action {

/\* forward name="success" path="" \*/

private static final String SUCCESS = "success";

private static final String FAILURE = "failure";

/\*\*

\* This is the action called from the Struts framework.

\*

\* @param mapping The ActionMapping used to select this instance.

\* @param form The optional ActionForm bean for this request.

\* @param request The HTTP Request we are processing.

\* @param response The HTTP Response we are processing.

\* @throws java.lang.Exception

\* @return

\*/

@Override

public ActionForward execute(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response)

throws Exception {

loginbean lb = (loginbean)form;

if(lb.getUname().equals("admin")&&lb.getUpass().equals("admin123"))

return mapping.findForward(SUCCESS);

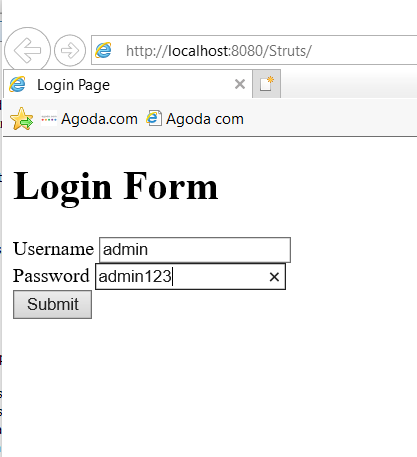
else

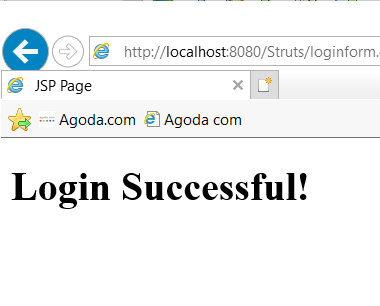
return mapping.findForward(FAILURE);

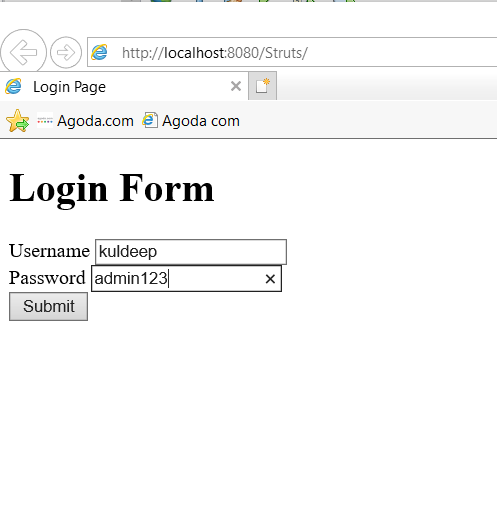
}

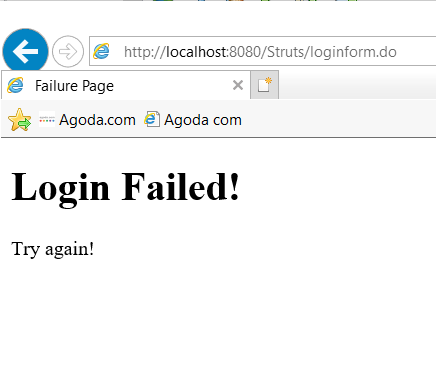
}

**Output:**

****

****

****

****