SWASTIK COLLEGE

Tribhuvan University

Faculty of Humanities and Social Sciences

Bachelor of Computer Application

(BCA)

Course: Advance Java

Semester: 6th

A Lab Report On:

Advance Java

Submitted by:

Name: Swosti Makaju

Roll No : 33

Submitted to:

Department Of BCA

1. Write a GUI program using components to find sum and difference of two numbers.Use two text fields for giving input and a label for output. The program should display sum if user presses mouse and difference if user release mouse.

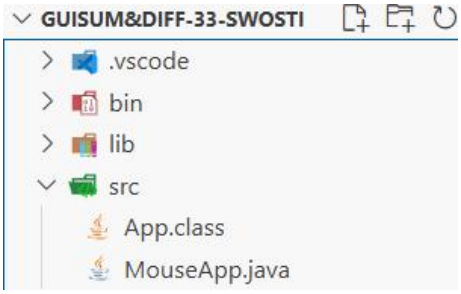
Objectives :

The objective of this program is to demonstrate the use of **MouseListener in Java AWT**. The program takes two numbers as input from text fields and performs arithmetic operations based on mouse events:

- **Displays the sum** of the two numbers when the mouse is pressed.
- **Displays the difference** of the two numbers when the mouse is released.

This illustrates how mouse interactions can be handled in GUI applications using event-driven programming.

Path :



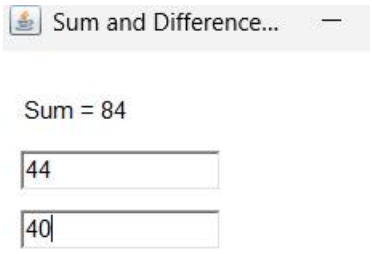
Source Code :

```
import java.awt.*;
import java.awt.event.*;
public class MouseApp implements MouseListener {
Label lblOutput;
TextField txtOne, txtTwo;
MouseApp() {
Frame f = new Frame("Sum and Difference using Mouse Events");
lblOutput = new Label("Result will be shown here");
lblOutput.setBounds(20, 50, 200, 20);
txtOne = new TextField();
txtOne.setBounds(20, 80, 100, 20);
txtTwo = new TextField();
txtTwo.setBounds(20, 110, 100, 20);
f.addMouseListener(this);
f.add(lblOutput);
f.add(txtOne);
```

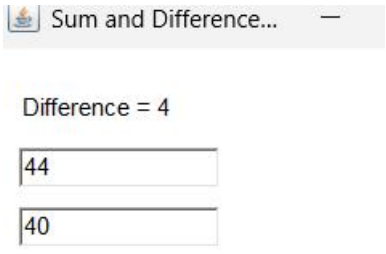
```
f.add(txtTwo);
f.setSize(300, 300);
f.setLayout(null);
f.setVisible(true); }
public void mousePressed(MouseEvent e) {
try {
int a = Integer.parseInt(txtOne.getText());
int b = Integer.parseInt(txtTwo.getText());
int sum = a + b;
lblOutput.setText("Sum = " + sum);
} catch (Exception ex) {
lblOutput.setText("Invalid Input!"); }  }
public void mouseReleased(MouseEvent e) {
try {
int a = Integer.parseInt(txtOne.getText());
int b = Integer.parseInt(txtTwo.getText());
int diff = a - b;
lblOutput.setText("Difference = " + diff);
} catch (Exception ex) {
lblOutput.setText("Invalid Input!");  }}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public static void main(String[] args) {
new MouseApp();  }   }
```

Output :

When mouse pressed :



When mouse released :



Conclusion :

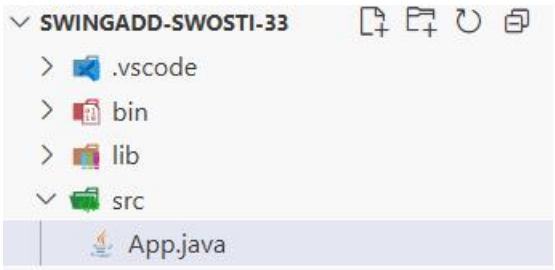
This program demonstrates how **event-driven programming** works in Java AWT using the **MouseListener interface**. By handling the mousePressed and mouseReleased events, the program performs addition and subtraction dynamically based on user interactions .

2. Write a Program using swing components to add two digit. Use text fields for inputs and output. Your program should display the result when the user presses a button.

Objectives :

The objective of this program is to create a simple **Java Swing application** that performs the addition of two numbers. The program uses **JTextField** components to take input, a **JButton** to trigger the addition, and another **JTextField** (or JLabel) to display the result when the button is pressed.

Path :

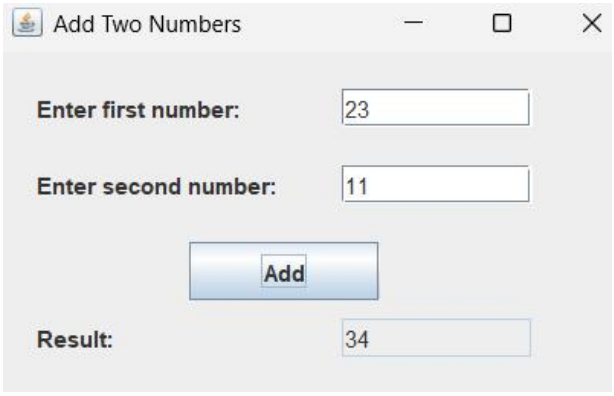


Source Code :

```
import javax.swing.*;
import java.awt.event.*;
public class App {
public static void main(String[] args) {
JFrame frame = new JFrame("Add Two Numbers");
JLabel lbl1 = new JLabel("Enter first number:");
lbl1.setBounds(20, 20, 150, 20);
JLabel lbl2 = new JLabel("Enter second number:");
lbl2.setBounds(20, 60, 150, 20);
JLabel lblResult = new JLabel("Result:");
lblResult.setBounds(20, 140, 200, 20);
JTextField txtOne = new JTextField();
txtOne.setBounds(180, 20, 100, 20);
JTextField txtTwo = new JTextField();
txtTwo.setBounds(180, 60, 100, 20);
JTextField txtResult = new JTextField();
txtResult.setBounds(180, 140, 100, 20);
txtResult.setEditable(false);
JButton btnAdd = new JButton("Add");
```

```
btnAdd.setBounds(100, 100, 100, 30);
btnAdd.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
try {
int a = Integer.parseInt(txtOne.getText());
int b = Integer.parseInt(txtTwo.getText());
int sum = a + b;
txtResult.setText(String.valueOf(sum));
} catch (Exception ex) {
txtResult.setText("Invalid Input"); } } });
frame.add(lbl1);
frame.add(lbl2);
frame.add(txtOne);
frame.add(txtTwo);
frame.add(btnAdd);
frame.add(lblResult);
frame.add(txtResult);
frame.setSize(350, 250);
frame.setLayout(null);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true); }}
```

Output :



Conclusion :

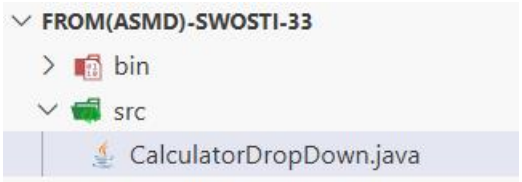
This program demonstrates how to use **Swing components** (JFrame, JLabel, JTextField, and JButton) to design a simple GUI-based calculator for addition. By attaching an ActionListener to the button, the program performs addition when clicked, showing the interactive capability of Swing in creating user-friendly applications.

3. Write a program to Create a Form Below using Swing and perform Add, Subtract, Multiply and Divide as user click button with selected option.

Objectives :

The objective of this program is to design a **Java Swing form** where users can input two numbers and perform **Addition, Subtraction, Multiplication, and Division** using buttons. The program demonstrates how multiple **ActionListeners** can be used to perform different operations in response to user interactions.

Path :



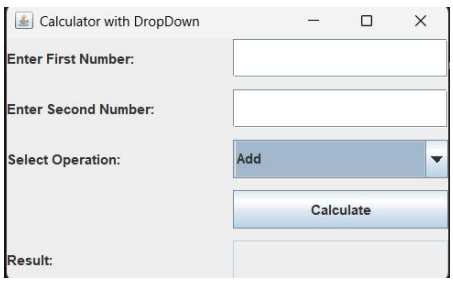
Source Code :

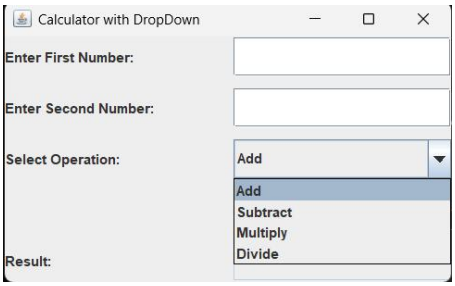
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class CalculatorDropDown extends JFrame implements ActionListener {
JTextField num1Field, num2Field, resultField;
JComboBox<String> operationBox;
JButton calcBtn;
public CalculatorDropDown() {
setTitle("Calculator with DropDown");
setSize(400, 250);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new GridLayout(5, 2, 10, 10));
JLabel num1Label = new JLabel("Enter First Number:");
JLabel num2Label = new JLabel("Enter Second Number:");
JLabel operationLabel = new JLabel("Select Operation:");
JLabel resultLabel = new JLabel("Result:");
num1Field = new JTextField();
num2Field = new JTextField();
resultField = new JTextField();
resultField.setEditable(false);
```

```
String[] operations = {"Add", "Subtract", "Multiply", "Divide"};
operationBox = new JComboBox<>(operations);
calcBtn = new JButton("Calculate");
calcBtn.addActionListener(this);
add(num1Label);
add(num1Field);
add(num2Label);
add(num2Field);
add(operationLabel);
add(operationBox);
add(new JLabel("")); // empty space
add(calcBtn);
add(resultLabel);
add(resultField);
setVisible(true);   }
public void actionPerformed(ActionEvent e) {
try {
double num1 = Double.parseDouble(num1Field.getText());
double num2 = Double.parseDouble(num2Field.getText());
double result = 0;
String operation = (String) operationBox.getSelectedItem();
switch (operation) {
case "Add":
result = num1 + num2;
break;
case "Subtract":
result = num1 - num2;
break;
case "Multiply":
result = num1 * num2;
break;
case "Divide":
if (num2 == 0) {
```

```
JOptionPane.showMessageDialog(this, "Division by zero not allowed!");
return;
}
result = num1 / num2;
break;
}
resultField.setText(String.valueOf(result));

} catch (NumberFormatException ex) {
JOptionPane.showMessageDialog(this, "Please enter valid numbers!");
}
}
public static void main(String[] args) {
new CalculatorDropDown();
}
}
```

Output :





Conclusion :

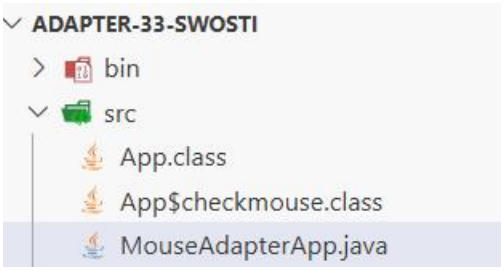
This program successfully demonstrates how to create a **GUI-based calculator using Swing components**. The use of JButton, JTextField, and JLabel allows users to interact with the program easily, while ActionListener handles different button clicks to perform **addition, subtraction, multiplication, and division**. This shows the effectiveness of Swing in developing interactive, event-driven desktop applications.

4. Write a GUI program using components to find sum and difference of two numbers.Use two text fields for giving input and a label for output. The program should display sum if user presses mouse and difference if user release mouse using adapter class.

Objectives :

To create a GUI program using AWT where two numbers are input through text fields, and the **sum** is shown on mouse press while the **difference** is shown on mouse release using a **MouseAdapter**.

Path :



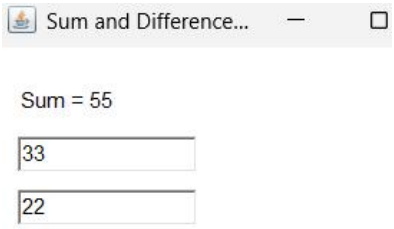
Source Code :

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterApp {
Label lblOutput;
TextField txtOne, txtTwo;
MouseAdapterApp() {
Frame f = new Frame("Sum and Difference using MouseAdapter");
lblOutput = new Label("Result will be shown here");
lblOutput.setBounds(20, 50, 200, 20);
txtOne = new TextField();
txtOne.setBounds(20, 80, 100, 20);
txtTwo = new TextField();
txtTwo.setBounds(20, 110, 100, 20);
f.addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent e) {
try {
int a = Integer.parseInt(txtOne.getText());
int b = Integer.parseInt(txtTwo.getText());
```

```
lblOutput.setText("Sum = " + (a + b));
} catch (Exception ex) {
lblOutput.setText("Invalid Input!");
}
}
public void mouseReleased(MouseEvent e) {
try {
int a = Integer.parseInt(txtOne.getText());
int b = Integer.parseInt(txtTwo.getText());
lblOutput.setText("Difference = " + (a - b));
} catch (Exception ex) {
lblOutput.setText("Invalid Input!");  }}});
f.add(lblOutput);
f.add(txtOne);
f.add(txtTwo);
f.setSize(300, 300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
new MouseAdapterApp();  }}
```

Output :

When mouse pressed : When mouse released :



Conclusion :

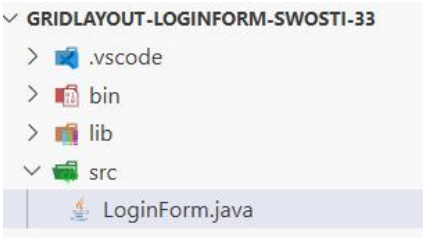
The program shows how **MouseAdapter** simplifies event handling by overriding only required methods, successfully displaying sum and difference based on mouse actions.

5. Create a Simple Login Form using GridLayout in Java.

Objectives :

To design a simple **Login Form** in Java using **Swing components** and arrange them with **GridLayout**. The form takes a username and password as input and validates them when the user clicks the Login button.

Path :

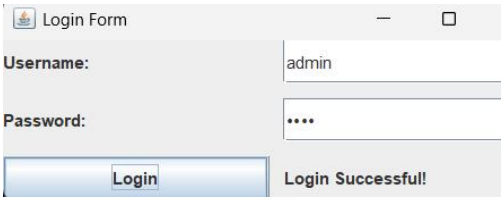


Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LoginForm {
public static void main(String[] args) {
JFrame frame = new JFrame("Login Form");
frame.setLayout(new GridLayout(3, 2, 10, 10));
JLabel lblUser = new JLabel("Username:");
JLabel lblPass = new JLabel("Password:");
JTextField txtUser = new JTextField();
JPasswordField txtPass = new JPasswordField();
JButton btnLogin = new JButton("Login");
JLabel lblMessage = new JLabel("");
btnLogin.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String user = txtUser.getText();
String pass = new String(txtPass.getPassword());
if (user.equals("admin") && pass.equals("1234")) {
lblMessage.setText("Login Successful!");
} else {
lblMessage.setText("Invalid Username or Password"); } } });
frame.add(lblUser);
```

```
frame.add(txtUser);
frame.add(lblPass);
frame.add(txtPass);
frame.add(btnLogin);
frame.add(lblMessage);
frame.setSize(350, 150);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true); } }
```

Output :



Conclusion :

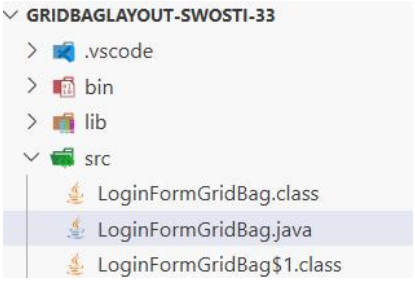
This program demonstrates how to use **GridLayout** to create a neat and simple **Login Form** in Java. It shows the arrangement of labels, text fields, and a button, and validates user credentials with a message.

6. Create a Simple Login Form using GridBagLayout in Java.

Objectives :

To design a simple **Login Form** in Java using **Swing components** arranged with **GridBagLayout**, demonstrating flexible GUI alignment.

Path :



Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LoginFormGridBag {
public static void main(String[] args) {
JFrame frame = new JFrame("Login Form using GridBagLayout");
frame.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.insets = new Insets(5, 5, 5, 5); // padding
JLabel lblUser = new JLabel("Username:");
JLabel lblPass = new JLabel("Password:");
JTextField txtUser = new JTextField(15);
JPasswordField txtPass = new JPasswordField(15);
JButton btnLogin = new JButton("Login");
JLabel lblMessage = new JLabel("");    gbc.gridx = 0; gbc.gridy = 0;
frame.add(lblUser, gbc);         gbc.gridx = 1; gbc.gridy = 0;
frame.add(txtUser, gbc);         gbc.gridx = 0; gbc.gridy = 1;
frame.add(lblPass, gbc);         gbc.gridx = 1; gbc.gridy = 1;
frame.add(txtPass, gbc);         gbc.gridx = 1; gbc.gridy = 2;
frame.add(btnLogin, gbc);     gbc.gridx = 1; gbc.gridy = 3;
frame.add(lblMessage, gbc);
btnLogin.addActionListener(new ActionListener() {
```

```
public void actionPerformed(ActionEvent e) {
String user = txtUser.getText();
String pass = new String(txtPass.getPassword());
if (user.equals("admin") && pass.equals("1234")) {
lblMessage.setText("Login Successful!");
} else {
lblMessage.setText("Invalid Username or Password"); } } });
frame.setSize(400, 200);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);  }  }
```

Output :

Conclusion :

This program shows how **GridBagLayout** provides flexibility in arranging components compared to GridLayout. It successfully creates a **Login Form** with aligned labels, text fields, and a button, demonstrating fine control over GUI layouts.

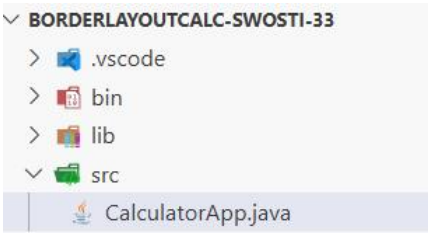
7. Write a program that divides the frame into five regions by using border layout and then add panels in the east, north and center region. Finally add some descriptive label in the north panel, buttons with icon in the east panel and a sample form in the center panel. You can further subdivide the center panel, if necessary. Prepare a program with three text boxes First Number, Second Number, and Result and four buttons add, subtract, multiply and divide. Handle the events to perform the required operation and display results.

Objectives :

To design a Java GUI using **BorderLayout** that places components in different regions:

- A label in the north panel,
- buttons with icons in the east panel,
- A calculator form in the center panel with text fields and buttons to perform arithmetic operations.

Path :



Source Code :

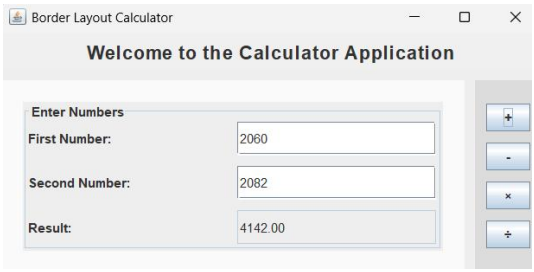
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class CalculatorApp extends JFrame implements ActionListener {
private final JTextField firstNumberField;
private final JTextField secondNumberField;
private final JTextField resultField;
private final JButton addButton;
private final JButton subtractButton;
private final JButton multiplyButton;
private final JButton divideButton;
public CalculatorApp() {
```

```
super("Border Layout Calculator");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setLayout(new BorderLayout(10, 10));
JPanel northPanel = new JPanel();
northPanel.setBackground(new Color(240, 240, 240));
JLabel northLabel = new JLabel("Welcome to the Calculator Application", SwingConstants.CENTER);
northLabel.setFont(new Font("SansSerif", Font.BOLD, 18));
northPanel.add(northLabel);
this.add(northPanel, BorderLayout.NORTH);
JPanel eastPanel = new JPanel(new GridLayout(4, 1, 10, 10));
eastPanel.setBorder(BorderFactory.createEmptyBorder(20, 10, 20, 10));
eastPanel.setBackground(new Color(220, 220, 220));
this.addButton = new JButton("+");
this.subtractButton = new JButton("-");
this.multiplyButton = new JButton("×");
this.divideButton = new JButton("÷");
this.addButton.addActionListener(this);
this.subtractButton.addActionListener(this);
this.multiplyButton.addActionListener(this);
this.divideButton.addActionListener(this);
this.addButton.setToolTipText("Add numbers");
this.subtractButton.setToolTipText("Subtract numbers");
this.multiplyButton.setToolTipText("Multiply numbers");
this.divideButton.setToolTipText("Divide numbers");
eastPanel.add(this.addButton);
eastPanel.add(this.subtractButton);
eastPanel.add(this.multiplyButton);
eastPanel.add(this.divideButton);
this.add(eastPanel, BorderLayout.EAST);
JPanel centerPanel = new JPanel(new BorderLayout(20, 20));
centerPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
centerPanel.setBackground(new Color(250, 250, 250));
```

```
JPanel formPanel = new JPanel(new GridLayout(3, 2, 10, 10));
formPanel.setBorder(BorderFactory.createTitledBorder("Enter Numbers"));
this.firstNumberField = new JTextField(15);
this.secondNumberField = new JTextField(15);
this.resultField = new JTextField(15);
this.resultField.setEditable(false);
formPanel.add(new JLabel("First Number:"));
formPanel.add(this.firstNumberField);
formPanel.add(new JLabel("Second Number:"));
formPanel.add(this.secondNumberField);
formPanel.add(new JLabel("Result:"));
formPanel.add(this.resultField);
centerPanel.add(formPanel, BorderLayout.CENTER);
this.add(centerPanel, BorderLayout.CENTER);
this.setSize(600, 350);
this.setLocationRelativeTo(null);
this.setVisible(true);
}
@Override
public void actionPerformed(ActionEvent e) {
try {
double num1 = Double.parseDouble(firstNumberField.getText());
double num2 = Double.parseDouble(secondNumberField.getText());
double result = 0;
if (e.getSource() == addButton) {
result = num1 + num2;
} else if (e.getSource() == subtractButton) {
result = num1 - num2;
} else if (e.getSource() == multiplyButton) {
result = num1 * num2;
} else if (e.getSource() == divideButton) {
// Handle division by zero
if (num2 == 0) {
```

```
resultField.setText("Error: Div by zero");
return;
}
result = num1 / num2;
}
resultField.setText(String.format("%.2f", result));
} catch (NumberFormatException ex) {
resultField.setText("Invalid Input");
}
}
public static void main(String[] args) {
SwingUtilities.invokeLater(CalculatorApp::new);}}
```

Output :



Conclusion :

This program demonstrates how to use **BorderLayout** to organize different GUI components in Java. The north panel displays a label, the east panel holds icon buttons, and the center panel contains a calculator form with text fields and operation buttons. It successfully performs addition, subtraction, multiplication, and division with proper event handling.

8. Design a form with three buttons with captions “RED,” “BLUE,” and “GREEN,” respectively. Then write a program to handle the event such that when the user clicks the button, the color of that button will be the same as its caption.

Objectives :

To design a simple GUI program with three buttons (**RED, BLUE, GREEN**) and handle events such that clicking a button changes its background color to match its caption.

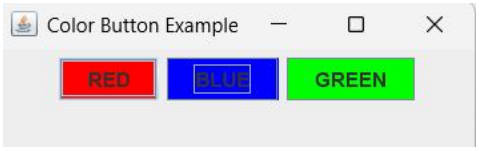
Path :

Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ColorButtonApp extends JFrame implements ActionListener {
private final JButton redButton, blueButton, greenButton;
public ColorButtonApp() {
super("Color Button Example");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new FlowLayout());
redButton = new JButton("RED");
blueButton = new JButton("BLUE");
greenButton = new JButton("GREEN");
redButton.addActionListener(this);
blueButton.addActionListener(this);
greenButton.addActionListener(this);
add(redButton);
add(blueButton);
add(greenButton);
setSize(300, 150);
setLocationRelativeTo(null);
```

```
setVisible(true);  }
@Override
public void actionPerformed(ActionEvent e) {
Object src = e.getSource();
if (src == redButton) redButton.setBackground(Color.RED);
else if (src == blueButton) blueButton.setBackground(Color.BLUE);
else if (src == greenButton) greenButton.setBackground(Color.GREEN);
}
public static void main(String[] args) {
SwingUtilities.invokeLater(ColorButtonApp::new);
} }
```

Output :



Conclusion :

This program demonstrates **event handling in Java Swing** by associating button clicks with specific actions. It shows how to use listeners to change component properties dynamically.

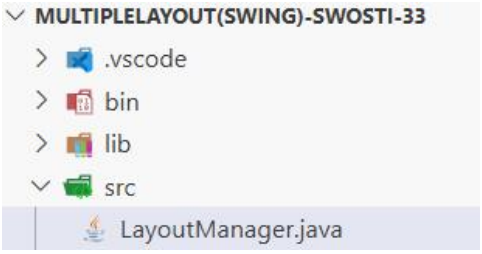
9. Write a Java program using Swing that demonstrates the use of different **layout managers** (FlowLayout, GridLayout, and BorderLayout).

- Divide the main frame into **three panels** using BorderLayout:
    - **North Panel**: Use FlowLayout to arrange three labels (Name, Roll No, Class) in a single row.
    - **Center Panel**: Use GridLayout (3x2) to create a simple login form with two text fields (Username, Password), a JCheckBox ("Remember Me"), and a JButton ("Login").
    - **South Panel**: Use FlowLayout to add two buttons (Reset and Exit).
- Handle events so that:
    - When **Login** is pressed, display the entered username in a message dialog.
    - When **Reset** is pressed, clear the text fields.
    - When **Exit** is pressed, close the program.

Objectives :

To demonstrate the use of **FlowLayout, GridLayout, and BorderLayout** in a single Java Swing application and handle events for login, reset, and exit functionalities.

Path :



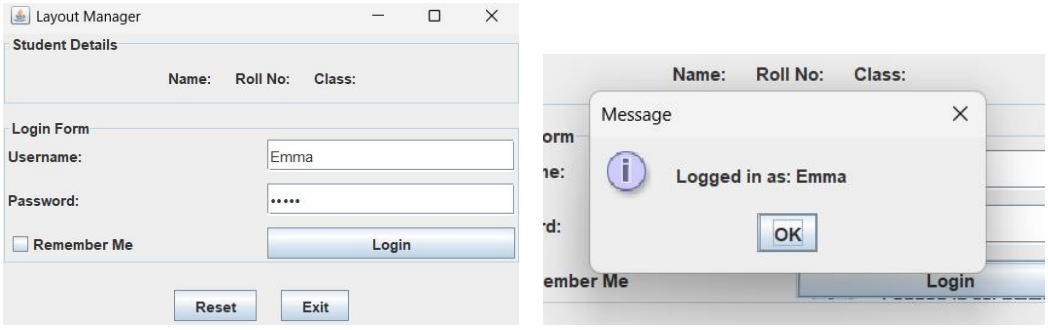
Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class LayoutManager extends JFrame implements ActionListener {
private JTextField usernameField;
```

```
        private JPasswordField passwordField;
        public LayoutManager() {
        super("Layout Manager");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new BorderLayout(10, 10));
        JPanel northPanel = new JPanel();
        northPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));
        northPanel.setBorder(BorderFactory.createTitledBorder("Student Details"));
        northPanel.add(new JLabel("Name:"));
        northPanel.add(new JLabel("Roll No:"));
        northPanel.add(new JLabel("Class:"));
        this.add(northPanel, BorderLayout.NORTH);
        JPanel centerPanel = new JPanel();
        centerPanel.setLayout(new GridLayout(3, 2, 10, 10));
        centerPanel.setBorder(BorderFactory.createTitledBorder("Login Form"));
        this.usernameField = new JTextField(20);
        this.passwordField = new JPasswordField(20);
        centerPanel.add(new JLabel("Username:"));
        centerPanel.add(this.usernameField);
        centerPanel.add(new JLabel("Password:"));
        centerPanel.add(this.passwordField);
        centerPanel.add(new JCheckBox("Remember Me"));
        JButton loginButton = new JButton("Login");
        loginButton.addActionListener(this);
        centerPanel.add(loginButton);
        this.add(centerPanel, BorderLayout.CENTER);
        JPanel southPanel = new JPanel();
        southPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));
        JButton resetButton = new JButton("Reset");
        JButton exitButton = new JButton("Exit");
        resetButton.addActionListener(this);
        exitButton.addActionListener(this);
southPanel.add(resetButton);
        southPanel.add(exitButton);
        this.add(southPanel, BorderLayout.SOUTH);
```

```
this.pack();
this.setLocationRelativeTo(null);
this.setVisible(true);
}
@Override
public void actionPerformed(ActionEvent e) {
String command = e.getActionCommand();
switch (command) {
case "Login":
String username = usernameField.getText();
JOptionPane.showMessageDialog(this, "Logged in as: " + username);
break;
case "Reset":
usernameField.setText("");
passwordField.setText("");
break;
case "Exit":
System.exit(0);
break;   } }
public static void main(String[] args) {
SwingUtilities.invokeLater(LayoutManager::new);}          }
```

Output :



Conclusion :

This program shows how to combine **different layout managers** in a Swing GUI. It also demonstrates event handling for buttons, making the application interactive and functional.

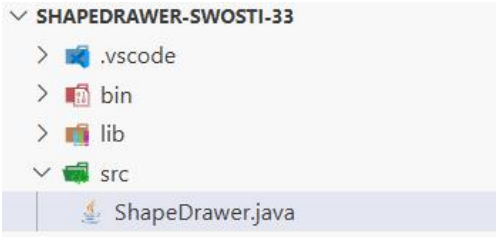
10. Design a form with three buttons with captions **“Circle,” “Square,” and “Triangle.”**

- When the user clicks the **“Circle”** button, draw a red circle in the center of the frame.
- When the user clicks the **“Square”** button, draw a blue square in the center of the frame.
- When the user clicks the **“Triangle”** button, draw a green triangle in the center of the frame.

Objectives :

To design a GUI program with three buttons (**Circle, Square, Triangle**) and handle events such that clicking each button draws the corresponding shape with a specific color in the center of the frame.

Path :



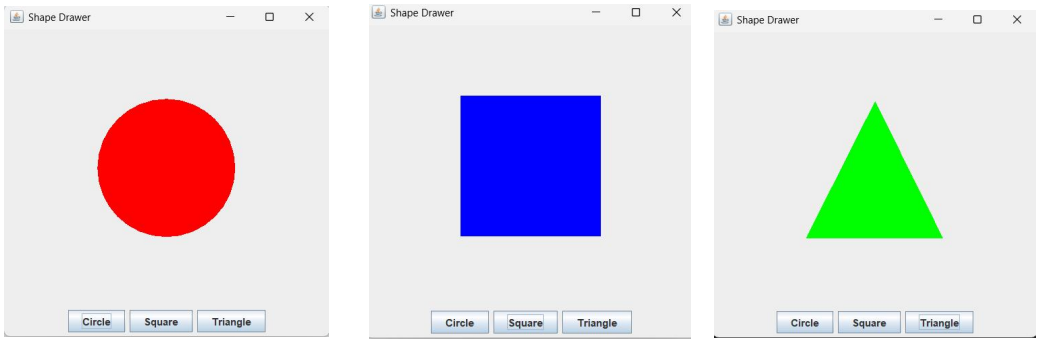
Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ShapeDrawer extends JFrame implements ActionListener {
private String shape = "";
private final DrawingPanel drawingPanel;
public ShapeDrawer() {
super("Shape Drawer");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new BorderLayout());
JPanel buttonPanel = new JPanel(new FlowLayout());
JButton circleButton = new JButton("Circle");
```

```
JButton squareButton = new JButton("Square");
JButton triangleButton = new JButton("Triangle");
circleButton.addActionListener(this);
squareButton.addActionListener(this);
triangleButton.addActionListener(this);
buttonPanel.add(circleButton);
buttonPanel.add(squareButton);
buttonPanel.add(triangleButton);
add(buttonPanel, BorderLayout.SOUTH);
drawingPanel = new DrawingPanel();
add(drawingPanel, BorderLayout.CENTER);
setSize(400, 400);
setLocationRelativeTo(null);
setVisible(true);    }
@Override
public void actionPerformed(ActionEvent e) {
shape = e.getActionCommand();
drawingPanel.repaint(); }
class DrawingPanel extends JPanel {
@Override
protected void paintComponent(Graphics g) {
super.paintComponent(g);
int w = getWidth();
int h = getHeight();
int size = Math.min(w, h) / 2;
int x = (w - size) / 2;
int y = (h - size) / 2;
switch (shape) {
case "Circle": g.setColor(Color.RED);
g.fillOval(x, y, size, size);
break;
case "Square": g.setColor(Color.BLUE);
g.fillRect(x, y, size, size);
```

```
break;
case "Triangle": g.setColor(Color.GREEN);
int[] xPoints = {w / 2, x, x + size};
int[] yPoints = {y, y + size, y + size};
g.fillPolygon(xPoints, yPoints, 3);
break; } } }
public static void main(String[] args) {
SwingUtilities.invokeLater(ShapeDrawer::new);
}
  }
```

Output :



Conclusion :

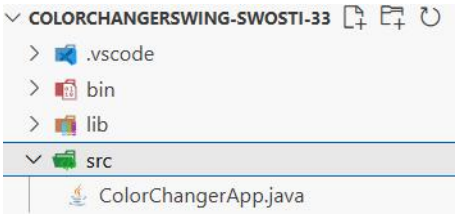
This program demonstrates how to use **Swing event handling** with **custom graphics (paintComponent)** to create an interactive shape drawer.

11. Design a Swing application that contains three radio buttons labeled "Red", "Green", and "Blue". When the user selects a radio button, the background color of the window should change to the selected color. Implement event handling using an ItemListener.

Objectives :

To create a Swing application that changes the window background color based on the selected radio button, using **ItemListener** for event handling.

Path :



Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ColorChangerApp extends JFrame implements ItemListener {
private JRadioButton redButton, greenButton, blueButton;
private ButtonGroup colorGroup;
public ColorChangerApp() {
super("Radio Button Color Changer");
setLayout(new FlowLayout());
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
redButton = new JRadioButton("Red");
greenButton = new JRadioButton("Green");
blueButton = new JRadioButton("Blue");
colorGroup = new ButtonGroup();
colorGroup.add(redButton);
colorGroup.add(greenButton);
colorGroup.add(blueButton);
```

```
redButton.addItemListener(this);
greenButton.addItemListener(this);
blueButton.addItemListener(this);
add(redButton);
add(greenButton);
add(blueButton);
setSize(300, 150);
setLocationRelativeTo(null);
setVisible(true);
}
@Override
public void itemStateChanged(ItemEvent e) {
if (redButton.isSelected()) {
getContentPane().setBackground(Color.RED);
}
else if (greenButton.isSelected()) {
getContentPane().setBackground(Color.GREEN);
}
else if (blueButton.isSelected()) {
getContentPane().setBackground(Color.BLUE);}   }
public static void main(String[] args) {
SwingUtilities.invokeLater(ColorChangerApp::new); }  }
```

Output :

Before clicking :

After clicking on red button :

Conclusion :

This program demonstrates how **radio buttons** and **ItemListener** can be used to perform actions based on user selection, allowing dynamic interface color changes.

12. Write a Java Swing program to create a toolbar with three buttons. Each button should display an **icon** (New, Login, and Logout). Place the toolbar at the top of the window using a suitable layout. When a button is clicked, display a message dialog showing the name of the action (e.g., “New File Created”, “Login Successful”, “Logged Out”).

Objectives :

To create a Swing application that uses a **toolbar** with buttons, displayed at the top, and shows appropriate messages when each button is clicked.

Path :

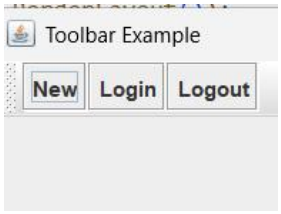
Source Code :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Toolbar extends JFrame implements ActionListener {
private JButton newButton, loginButton, logoutButton;
public Toolbar() {
super("Toolbar Example");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new BorderLayout());
JToolBar toolbar = new JToolBar();
newButton = new JButton("New");
loginButton = new JButton("Login");
logoutButton = new JButton("Logout");
newButton.addActionListener(this);
loginButton.addActionListener(this);
logoutButton.addActionListener(this);
```

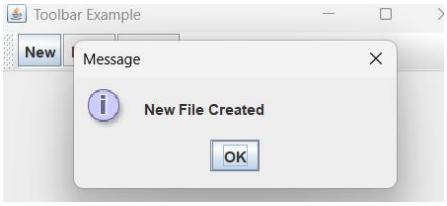
```
toolbar.add(newButton);
toolbar.add(loginButton);
toolbar.add(logoutButton);
add(toolbar, BorderLayout.NORTH);
setSize(400, 200);
setLocationRelativeTo(null);
setVisible(true);     }
@Override
public void actionPerformed(ActionEvent e) {
if (e.getSource() == newButton)
JOptionPane.showMessageDialog(this, "New File Created");
else if (e.getSource() == loginButton)
JOptionPane.showMessageDialog(this, "Login Successful");
else if (e.getSource() == logoutButton)
JOptionPane.showMessageDialog(this, "Logged Out");     }
public static void main(String[] args) {
SwingUtilities.invokeLater(Toolbar::new); }   }
```

Output :

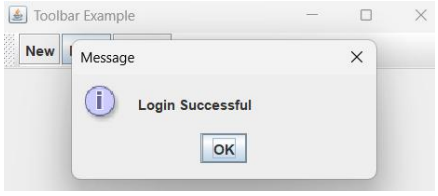
Before clicking on buttons:



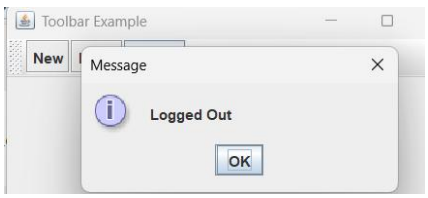
After clicking on new button:



After clicking on login:



After clicking on logout:



Conclusion :

This program demonstrates the use of **JToolBar** and **ActionListener** to create interactive toolbar buttons in a Java Swing interface.

13. Write a java program to inset 5 student records and display record in name ascending order, given table student(Id,Name,Gender,Address) using JDBC.

Objectives :

The objective of this program is to demonstrate how JDBC is used to connect a Java application with a MySQL database. It allows inserting multiple student records and retrieving the stored data in **alphabetical order by name**, showing both database insertion and data retrieval operations.

Path :

Source Code :

```
import java.sql.*;
import java.util.Scanner;
public class App {
public static void main(String[] args) {
Connection con = null;
PreparedStatement psInsert = null;
Statement stmt = null;
ResultSet rs = null;
Scanner sc = new Scanner(System.in);
try {
Class.forName("com.mysql.cj.jdbc.Driver");
con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/studentdb?useSSL=false",
"root",""  );
```

```
String insertSql = "INSERT INTO student(name, gender, address) VALUES (?, ?, ?)";
psInsert = con.prepareStatement(insertSql);
System.out.println("Enter details for 5 students:");
for(int i = 1; i <= 5; i++) {
System.out.println("\nStudent " + i + ":");
System.out.print("Name: ");
String name = sc.nextLine();
System.out.print("Gender: ");
String gender = sc.nextLine();
System.out.print("Address: ");
String address = sc.nextLine();
psInsert.setString(1, name);
psInsert.setString(2, gender);
psInsert.setString(3, address);
psInsert.executeUpdate();  }
System.out.println("\n✅ Records Inserted Successfully!\n");
stmt = con.createStatement();
rs = stmt.executeQuery("SELECT * FROM student ORDER BY name ASC");
System.out.println("---- Student Records (Sorted by Name) ----");
while(rs.next()) {
System.out.println(
rs.getInt("id") + "\t"
+ rs.getString("name") + "\t"
+ rs.getString("gender") + "\t"
+ rs.getString("address")  );  }
} catch(Exception e) {
System.out.println("Error: " + e.getMessage());
} finally {
try { con.close(); } catch(Exception ignored) {}
sc.close();
  }          }       }
```

Output :

```
PS D:\Advanced-Java\insertjdbc-swosti-33>  d:; cd 'd:\Advanced-Java\inser
jdbc-swosti-33'; & 'C:\Program Files\Java\jdk-24\bin\java.exe' '@C:\Users
swost\AppData\Local\Temp\cp_bd06mwiga3q9zzyllt3ecbco2.argfile' 'App'
Enter details for 5 students:

Student 1:
Name: swosti
Gender: female
Address: bhaktapur

Student 2:
Name: srashta
Gender: female
Address: bhaktapur

Student 3:
Name: suneel
Gender: male
Address: pokhara

Student 4:
Name: suman
Gender: male
Address: kathmandu

Student 5:
Name: alice
Gender: female
Address: kathmandu

? Records Inserted Successfully!

---- Student Records (Sorted by Name) ----
5       alice   female  kathmandu
2       srashta female  bhaktapur
4       suman   male    kathmandu
3       suneel  male    pokhara
1       swosti  female  bhaktapur
```

In MySql :

Id	Name	Gender	Address
1	swosti	female	bhaktapur
2	srashta	female	bhaktapur
3	suneel	male	pokhara
4	suman	male	kathmandu
5	alice	female	kathmandu

Conclusion :

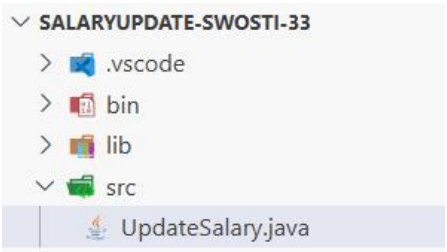
This program successfully shows how data can be inserted into and retrieved from a database using JDBC. By sorting records using SQL, it demonstrates the ability to integrate Java with database queries for effective data management and display.

14. Write a java program update salary by 15% wholes salary less than 20000 from given table student(Id,Name,Salary,Gender) using JDBC.

Objectives :

To write a JDBC program that connects to the database and updates the salary of employees. If an student's salary is less than 20000, it should be increased by 15%.

Path :



Source Code :

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

public class UpdateSalary {

public static void main(String[] args) {

try {

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

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/swastik?useSSL=false",

"root",

""

);
```

```
String sql = "UPDATE student SET salary = salary + (salary * 0.15) WHERE salary < 20000";

PreparedStatement ps = con.prepareStatement(sql);

int rows = ps.executeUpdate();

System.out.println(rows + " record(s) updated successfully!");

con.close();

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}

}

}
```

Output :

Before update:

Id	Name	Salary	Gender
1	Rahul	18000	Male
2	Sita	25000	Female
3	Aman	15000	Male
4	Rina	22000	Female
5	Kamal	17000	Male

After update:

Id	Name	Salary	Gender
1	Rahul	20700	Male
2	Sita	25000	Female
3	Aman	17250	Male
4	Rina	22000	Female
5	Kamal	19550	Male

Conclusion :

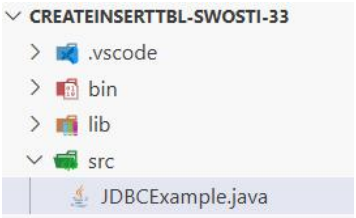
This program demonstrates how to use JDBC to update records in a MySQL database using an SQL UPDATE query and PreparedStatement. It successfully increases salary values based on a condition.

15. Write a Java program using JDBC to connect to a MySQL database, create a table named tblStudent with columns id, name, and email, and insert a record into the table using a prepared statement.

Objectives :

To demonstrate how to connect Java with a MySQL database, create a table dynamically, and insert records securely using JDBC and PreparedStatement.

Path :



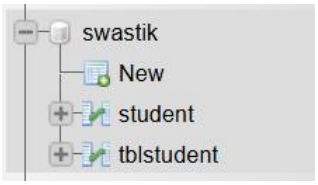
Source Code :

```
import java.sql.*;

public class JDBCExample {

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

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/swastik?useSSL=false",

"root", ""

);

con.createStatement().execute(

"CREATE TABLE IF NOT EXISTS tblStudent (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50), email VARCHAR(50))"

);

PreparedStatement ps = con.prepareStatement("INSERT INTO tblStudent (name, email) VALUES (?, ?)");
```

```
ps.setString(1, "Rahul");

ps.setString(2, "rahul@example.com");

ps.executeUpdate();

ps.close();

con.close();

 }

 }
```

Output :

Created table “tblstudent”



Inserted table on “tblstudent”

id	name	email
1	Rahul	rahul@example.com

Conclusion :

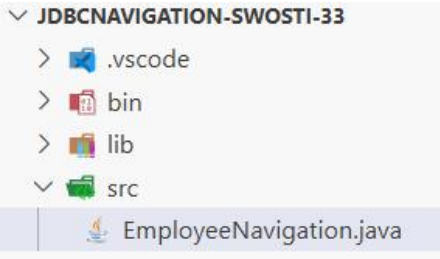
This program shows how to manage database operations in Java using JDBC. It demonstrates creating a table and inserting records safely, which is a key part of database-driven applications.

16. Write a Java program using JDBC to connect to a MySQL database and display employee details from the **emp** table. The program should demonstrate how to move through the result set using navigation methods such as **first()**, **last()**, **previous()**, and **next()**, and print the corresponding employee records on the console.

Objectives :

To demonstrate **JDBC ResultSet navigation methods** (first(), last(), previous(), next()) for iterating over database records in Java.

Path :



Source Code :

```
import java.sql.*;

public class EmployeeNavigation {

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

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/swastik?useSSL=false",

"root","" );

Statement stmt = con.createStatement(

ResultSet.TYPE_SCROLL_INSENSITIVE,

ResultSet.CONCUR_READ_ONLY );

ResultSet rs = stmt.executeQuery("SELECT * FROM emp");

if (rs.first())
```

```
System.out.println("First Employee: " + rs.getInt("id") + "\t" + rs.getString("name") + "\t" + rs.getString("department"));

if (rs.last())

System.out.println("Last Employee: " + rs.getInt("id") + "\t" + rs.getString("name") + "\t" + rs.getString("department"));

if (rs.previous())

System.out.println("Previous Employee: " + rs.getInt("id") + "\t" + rs.getString("name") + "\t" + rs.getString("department"));

if (rs.next())

System.out.println("Next Employee: " + rs.getInt("id") + "\t" + rs.getString("name") + "\t" + rs.getString("department"));

rs.close();

stmt.close();

con.close();

}

}
```

Output :

```
PS D:\Advanced-Java\jdbcnavigation-swosti-33>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '@C:\Users\swost\AppData\Local\Temp\cp_eob5i0ru39z0pu22yj525us8o.argfile' 'EmployeeNavigation'
First Employee: 1       Rahul   IT
Last Employee: 5        Kamal   IT
Previous Employee: 4    Rina    Marketing
Next Employee: 5        Kamal   IT
```

Conclusion :

This program shows how to navigate through database query results using **scrollable ResultSet**. It helps in accessing records in different orders and positions without repeatedly querying the database.

17. Write a Java program using JDBC to connect to a MySQL database and retrieve an employee record from the emp table based on a given ID using a prepared statement. The program should use an **updatable ResultSet** to modify the employee's name and update the record in the database.

Objectives :

To demonstrate how to retrieve a specific record using a **PreparedStatement** and modify it using an **updatable ResultSet** in JDBC.

Path :

Source Code :

```
import java.sql.*;
import java.util.Scanner;
public class UpdateEmployee {
public static void main(String[] args) throws Exception {
Scanner sc = new Scanner(System.in);
System.out.print("Enter Employee ID to update: ");
int empId = sc.nextInt();
sc.nextLine();
System.out.print("Enter new Employee Name: ");
String newName = sc.nextLine();
Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/swastik?useSSL=false","root","");
PreparedStatement ps = con.prepareStatement(
"SELECT * FROM emp WHERE id = ?",
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
ps.setInt(1, empId);
ResultSet rs = ps.executeQuery();
if (rs.next()) {
```

```
System.out.println("Current Employee: " + rs.getInt("id") + "\t" + rs.getString("name") + "\t" + rs.getString("department"));

rs.updateString("name", newName);

rs.updateRow();

System.out.println("Employee name updated successfully."); } else {

System.out.println("Employee with ID " + empId + " not found.");}

rs.close();

ps.close();

con.close();

sc.close(); } }
```

Output :

```
PS D:\Advanced-Java\updatejdbc-swosti-33> & 'C:\Program
 Files\Java\jdk-24\bin\java.exe' '@C:\Users\swost\AppDat
a\Local\Temp\cp_cvopvnazyt6xs7bh89fzgfkoq.argfile' 'Upda
teEmployee'
Enter Employee ID to update: 1
Enter new Employee Name: Nancy
Current Employee: 1     Rahul   IT
Employee name updated successfully.
```

In MySql:

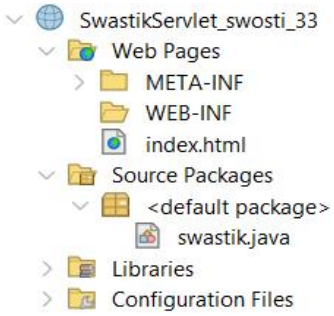
id	name	department
1	Nancy	IT
2	Sita	HR
3	Aman	Finance
4	Rina	Marketing
5	Kamal	IT

Conclusion :

This program shows how to update database records dynamically in Java. By using **updatable ResultSet**, you can modify selected fields directly without writing a separate UPDATE SQL query, demonstrating an advanced JDBC feature.

18. Write a servlet program to print Swastik College 10 times.

Path :



Objective :

The objective of this experiment is to develop a simple Java Servlet that runs on an Apache Tomcat server and prints **"Swastik College"** ten times in the browser. This task helps in understanding how servlets handle client requests, generate dynamic responses, and use the HttpServletResponse object to send formatted output to the client.

Source Code :

index.html :

```
<!DOCTYPE html>

<html>  <head>

    <title>Print swastik college</title>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">  </head>
<body>  <form action="swastik" method="post">

        <input type="submit" value="invoke life cycle servlet">  </form>  </body>

</html>
```

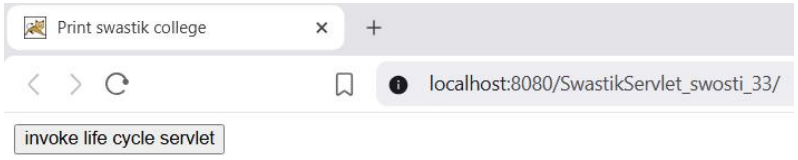
swastik.java :

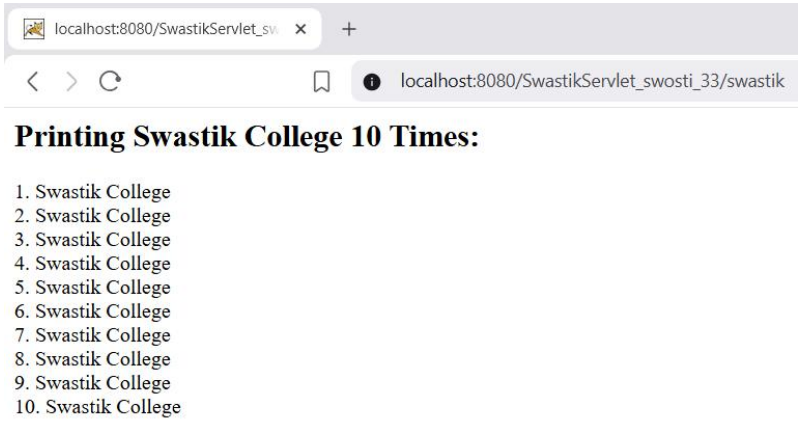
```
import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;
```

```
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(urlPatterns = {"/swastik"})
public class swastik extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       try (PrintWriter out = response.getWriter()) {
       out.println("<!DOCTYPE html>");
           out.println("<html>");
           out.println("<head><title>Print Swastik College</title></head>");
           out.println("<body>");
           out.println("<h1>Servlet swastik at " + request.getContextPath() + "</h1>");
           out.println("<h2>Printing 'Swastik College' 10 times:</h2>");
           out.println("</body>");
           out.println("</html>");   }  }
      ServletConfig config = null;   @Override
   public void init(ServletConfig sc) throws ServletException {
       config = sc;
       System.out.println("Init Method Executed");   }    @Override
   public ServletConfig getServletConfig() {
       return config;   }   @Override
```

```
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {    res.setContentType("text/html");

    PrintWriter pw = res.getWriter();

    pw.println("<h2>Printing Swastik College 10 Times:</h2>");

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

        pw.println(i + ". Swastik College<br>");   }

    System.out.println("Service Method Executed");   }   @Override

public String getServletInfo() {

    return "MyServlet for Printing Swastik College";  }   @Override

public void destroy() {

    System.out.println("Destroy Method Executed"); }}
```

Output :





Conclusion :

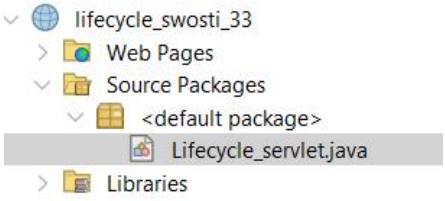
The servlet successfully displays **"Swastik College"** ten times, showing how servlets process requests and send HTML output. This program helps understand basic servlet operation and server-side response handling.

19. Write a program to demonstrate servlet life cycle.

Objectives :

To demonstrate the servlet life cycle by implementing and observing the execution of init(), doGet(), and destroy() methods.

Path :



Source Code :

index.html:

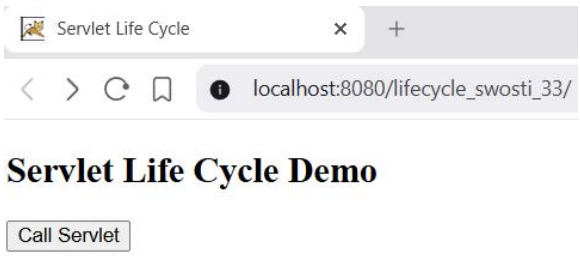
```
<!DOCTYPE html>
<html>
<head><title>Servlet Life Cycle</title> </head>
<body>
<h2>Servlet Life Cycle Demo</h2>
<form action="ServletLifeCycle" method="get">
<button type="submit">Call Servlet</button>
</form>
</body>
</html>
```

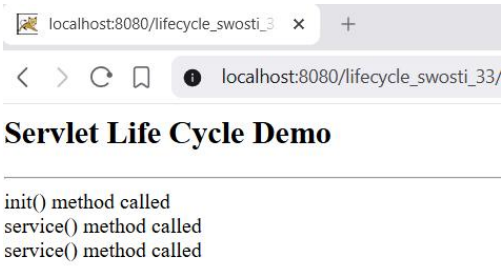
Lifecycle_servlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ServletLifeCycle")
public class Lifecycle_servlet extends HttpServlet {
   String message = "";
   @Override
```

```
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        message += "init() method called<br>";
    }
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        message += "service() method called<br>";
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h2>Servlet Life Cycle Demo</h2>");
        out.println("<hr>");
        out.println(message);
    }
    @Override
    public void destroy() {
        message += "destroy() method called<br>";
    }
}
```

Output :





Conclusion :

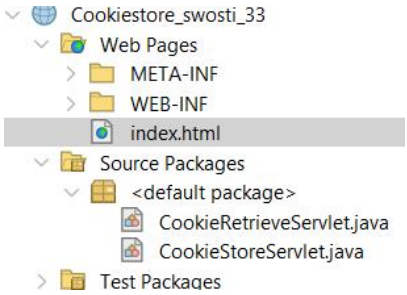
The program successfully explains the servlet life cycle in a simple way. It shows that init() is called only once, service() is called for every client request, and destroy() is called before the servlet is removed. This program helps in understanding how the servlet container manages a servlet.

20. Write a Java program to store Username and Password from two input field in cookies and retrieve value from cookies using Servlet.

Objectives :

The objective of this program is to understand how cookies are used in Java Servlets to store and retrieve client-side data. This experiment demonstrates storing username and password values from input fields into cookies and later retrieving those values in a servlet, helping to learn how servlets maintain user-related information across HTTP requests.

Path :



Source Code :

index.html:

```
<!DOCTYPE html> <html>
   <head> <title>cookie store</title> </head>
   <body> <h2>Enter Username and Password</h2>
   <form action="CookieStoreServlet" method="post">
      Username: <input type="text" name="username" required><br><br>
      Password: <input type="password" name="password" required><br><br>
      <input type="submit" value="Submit">
   </form> </body>
</html>
```

CookieRetriveServlet.jsp:

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
@WebServlet("/CookieRetrieveServlet")
```

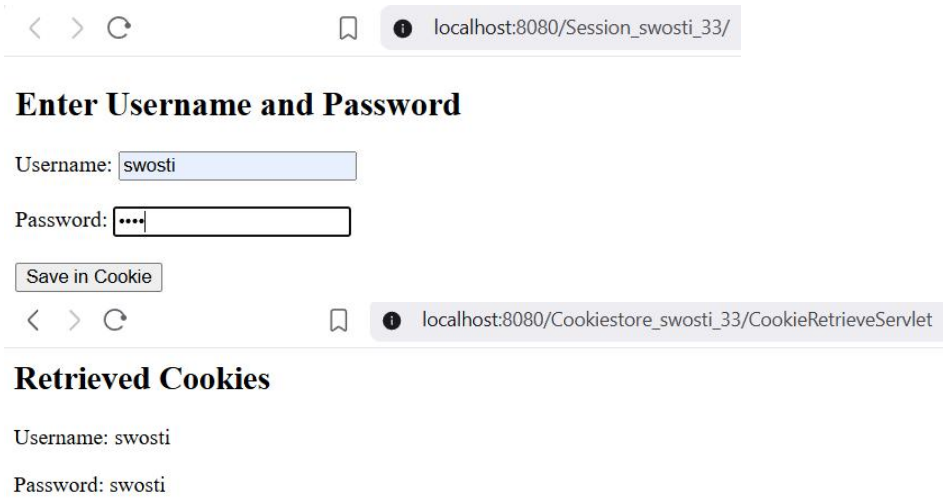
```
public class CookieRetrieveServlet extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      Cookie[] cookies = request.getCookies();
      String username = "";
      String password = "";
      if (cookies != null) {
         for (Cookie cookie : cookies) {
            if (cookie.getName().equals("username")) {
               username = cookie.getValue();
            } if (cookie.getName().equals("password")) {
               password = cookie.getValue();
            } } }
      out.println("<html><body>");
      out.println("<h2>Retrieved Cookies</h2>");
      out.println("<p>Username: " + username + "</p>");
      out.println("<p>Password: " + password + "</p>");
      out.println("</body></html>");
   }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {  processRequest(request, response); }
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException { processRequest(request, response); }
   @Override
```

```
    public String getServletInfo() {
        return "Short description";
} }
```

CookieStoreServlet.jsp:

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/CookieStoreServlet")
public class CookieStoreServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        Cookie userCookie = new Cookie("username", username);
        Cookie passCookie = new Cookie("password", password);
userCookie.setMaxAge(24 * 60 * 60);
        passCookie.setMaxAge(24 * 60 * 60);        response.addCookie(userCookie);
        response.addCookie(passCookie);
response.sendRedirect("CookieRetrieveServlet");
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

Output :



Conclusion :

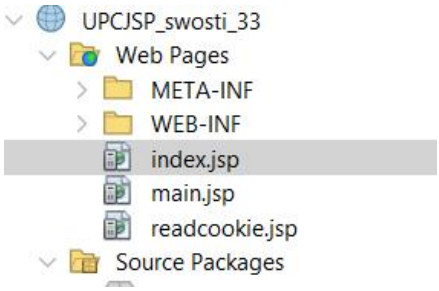
This program successfully demonstrates the use of cookies in servlets to store and retrieve user data. It shows how cookies help maintain state between client and server in a stateless HTTP environment, providing a basic understanding of session management in web applications.

21. Write a Java program to store Username and Password from two input field in cookies and retrieve value from cookies using JSP.

Objectives :

The objective of this program is to understand how cookies are used in JSP to store and retrieve user information. This experiment demonstrates storing username and password from input fields into cookies and accessing those values later, helping to learn basic state management in JSP applications.

Path :



Source 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>JSP Page</title>
   </head>
   <body>
       <form action = "main.jsp" method = "Post">
      Username: <input type = "text" name = "username">
      <br />
      Password: <input type = "text" name = "password" />
      <input type = "submit" value = "Submit" />
    </form>
   </body>
</html>
```

main.jsp:

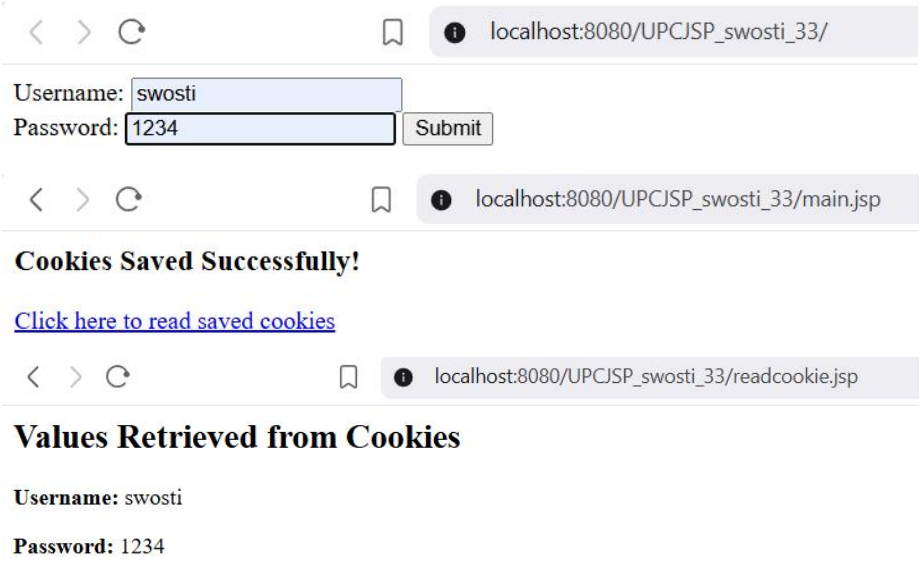
```
<%@ page import="javax.servlet.http.Cookie" %>
<%
   String username = request.getParameter("username");
   String password = request.getParameter("password");
   Cookie userCookie = new Cookie("username", username);
   Cookie passCookie = new Cookie("password", password);
   userCookie.setMaxAge(60 * 60 * 24); // 1 day
   passCookie.setMaxAge(60 * 60 * 24);
   response.addCookie(userCookie);
   response.addCookie(passCookie);
%>
<html>
<body>
<h3>Cookies Saved Successfully!</h3>
<a href="readcookie.jsp">Click here to read saved cookies</a>
</body>
</html>
```

readcookie.jsp:

```
<%@ page import="javax.servlet.http.Cookie" %>
<%
   String username = "";
   String password = "";
   Cookie[] cookies = request.getCookies();
   if (cookies != null) {
      for (Cookie c : cookies) {
         if (c.getName().equals("username")) {
            username = c.getValue();
         }
         if (c.getName().equals("password")) {
            password = c.getValue();}}}
```

```
%>
<html>
<body>
<h2>Values Retrieved from Cookies</h2>
<p><b>Username:</b> <%= username %></p>
<p><b>Password:</b> <%= password %></p>
</body>
</html>
```

Output :



Conclusion :

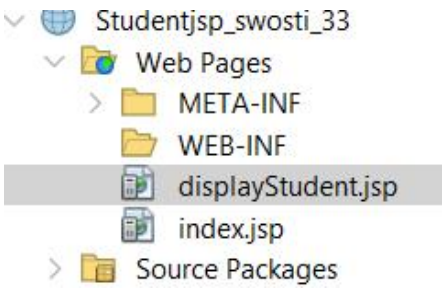
This program successfully demonstrates the use of cookies in JSP to store and retrieve user data. It helps in understanding how cookies maintain information between client and server across multiple requests in a stateless web environment.

22. Write a program to a JSP web form to take input of a student and submit it to second JSP file which may simply print the values of form submission.

Objectives :

The objective of this program is to demonstrate how a JSP web form collects student information and submits the entered data to another JSP file. This helps in understanding form handling, request processing, and data transfer between JSP pages using request parameters.

Path :



Source Code :

index.jsp:

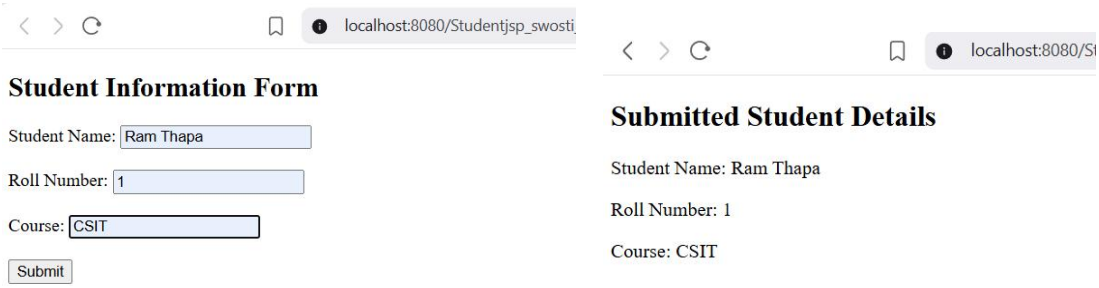
```
<!DOCTYPE html> <html>
 <head> <title>JSP Page</title> </head>
   <body> <h2>Student Information Form</h2>
   <form action="displayStudent.jsp" method="post">
      Student Name:
      <input type="text" name="name"><br><br>
      Roll Number:
      <input type="text" name="rollno"><br><br>
      Course:
      <input type="text" name="course"><br><br>
<input type="submit" value="Submit">
   </form>  </body> </html>
```

displayStudent.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>  <h2>Submitted Student Details</h2>
<%
        String name = request.getParameter("name");
        String rollno = request.getParameter("rollno");
        String course = request.getParameter("course");
    %>
    <p>Student Name: <%= name %></p>
    <p>Roll Number: <%= rollno %></p>
    <p>Course: <%= course %></p>
    </body> </html>
```

Output :



Conclusion :

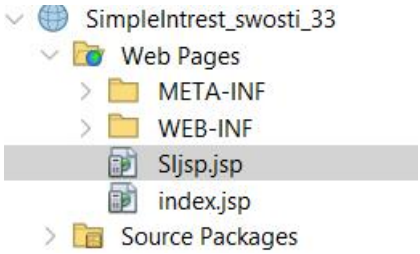
This program successfully demonstrates passing data from one JSP page to another using form submission. It helps in understanding how JSP handles user input and displays submitted values, which is essential for building dynamic web applications.

23. Write a program to a JSP web form to take principle,Time and Rate as input from textfield and calculate it and display Simple interest in second JSP.

Objectives :

The objective of this program is to create a JSP web form that accepts Principal, Time, and Rate as input values and submits them to another JSP page to calculate and display the Simple Interest. This helps in understanding form handling, request parameter processing, and performing calculations using JSP.

Path :



Source Code :

index.jsp:

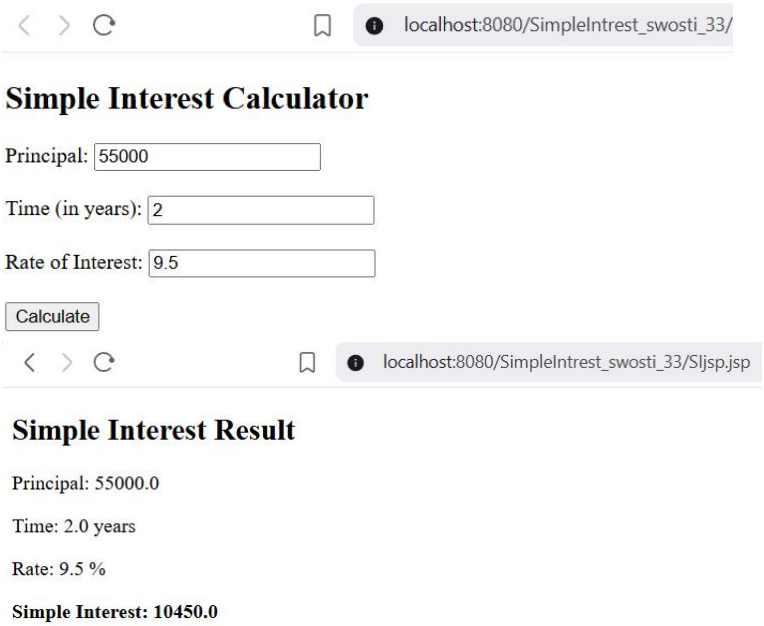
```
<!DOCTYPE html> <html>
   <head> <title>Simple Interest Form</title> </head>
<body> <h2>Simple Interest Calculator</h2>
   <form action="SIjsp.jsp" method="post">
      Principal: <input type="text" name="principal" required><br><br>
      Time (in years): <input type="text" name="time" required><br><br>
      Rate of Interest: <input type="text" name="rate" required><br><br>
      <input type="submit" value="Calculate">
   </form>
</body> </html>
```

SIjsp.jsp:

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
```

```
<body>  <h2>Simple Interest Result</h2>
 <%
    double principal = Double.parseDouble(request.getParameter("principal"));
    double time = Double.parseDouble(request.getParameter("time"));
    double rate = Double.parseDouble(request.getParameter("rate"));
    double simpleInterest = (principal * time * rate) / 100;
 %>
 <p>Principal: <%= principal %></p>
 <p>Time: <%= time %> years</p>
 <p>Rate: <%= rate %> %</p>
 <p><b>Simple Interest: <%= simpleInterest %></b></p>
</body> </html>
```

Output :



Conclusion :

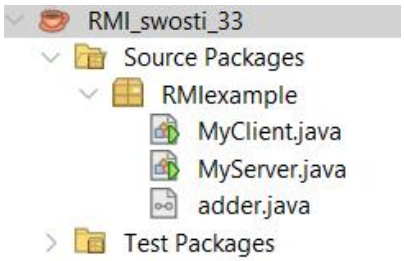
This program successfully demonstrates how to process user input in JSP and perform calculations in a second JSP page. It provides a clear understanding of JSP form submission and dynamic content generation for basic web-based calculations.

24. Write a java program to add two digit using RMI.

Objectives :

The objective of this program is to understand the concept of Remote Method Invocation (RMI) in Java. This experiment demonstrates how a client can invoke a method on a remote server object to add two numbers, helping to learn basic distributed application communication using Java RMI.

Path :



Source Code :

MyClient.java:

```
package RMIexample;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

public class MyClient {

    public static void main(String[] args) {

try {

Registry reg=LocateRegistry.getRegistry("localhost",5000);

adder ad=(adder)reg.lookup("hi_server");

System.out.println("Addition:"+ad.add(20, 30));

} catch (Exception e) {

System.out.println(e);

}}}
```

MyServer.java:

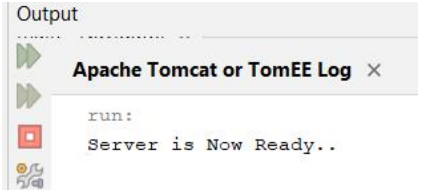
```
package RMIexample;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;
```

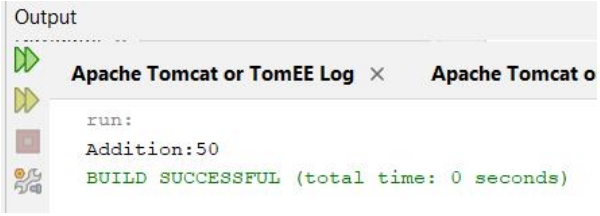
```
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
public class MyServer extends UnicastRemoteObject implements adder {
protected MyServer() throws RemoteException {
super();
}
public static void main(String[] args) throws Exception {
Registry reg = LocateRegistry.createRegistry(5000);
reg.rebind("hi_server", new MyServer());
System.out.println("Server is Now Ready..");
}
@Override
public int add(int x, int y) throws RemoteException {
return x + y;
}}
```

adder.java:

```
package RMIexample;
import java.rmi.*;
public interface adder extends Remote {
   public int add(int x,int y)throws RemoteException; }
```

Output :





Conclusion :

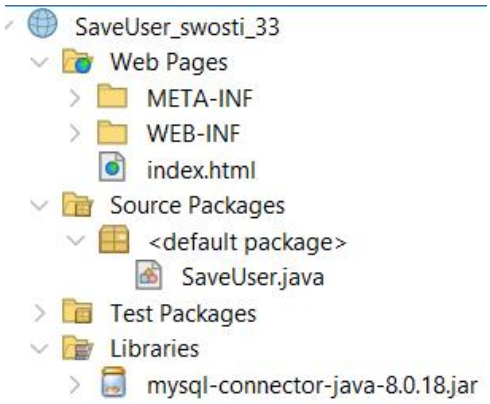
This program successfully demonstrates the use of Java RMI to perform addition of two numbers remotely. It shows how methods can be executed on a server from a client machine, providing a clear understanding of remote communication and distributed computing in Java.

25. Write a Java Servlet program to demonstrate data accessed using JDBC. Create a HTML form that accepts name, email, and address from user when the form is submitted. Data should be stored in table using POST method.

Objective :

The objective of this program is to demonstrate how a **Java Servlet** interacts with a **MySQL database using JDBC**. The program accepts **name, email, and address** from an HTML form using the **POST method** and stores the submitted data into a database table, helping to understand server-side form handling and database connectivity.

Path :



Source Code :

index.html :

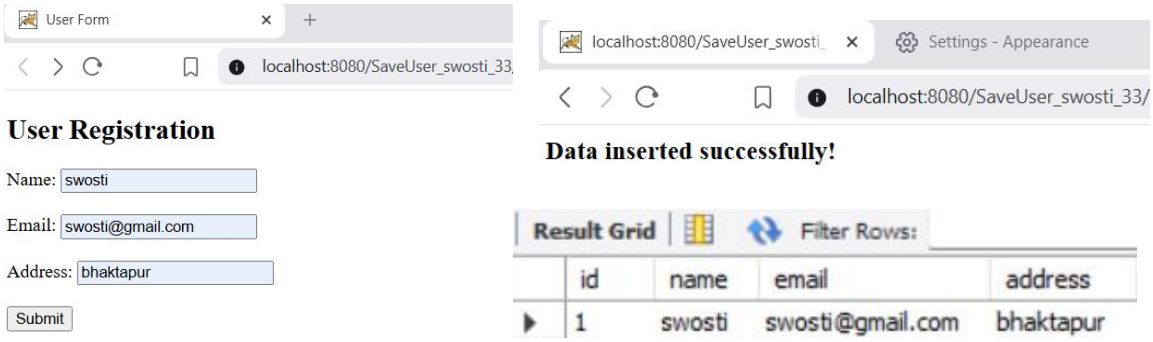
```
<html>

<head>  <title>User Form</title></head>

<body>

   <h2>User Registration</h2>

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

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

      Email: <input type="email" name="email" required><br><br>

      Address: <input type="text" name="address" required><br><br>

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

</body></html>
```

SaveUser.java :

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/InsertServlet")
public class SaveUser extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String name = request.getParameter("name");
      String email = request.getParameter("email");
      String address = request.getParameter("address");
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/lab?useSSL=false&serverTimezone=UTC",    "root",    "root"  );
         String sql = "INSERT INTO user(name, email, address) VALUES (?, ?, ?)";
         PreparedStatement ps = con.prepareStatement(sql);
         ps.setString(1, name);
         ps.setString(2, email);
         ps.setString(3, address);
         int result = ps.executeUpdate();
```

```
            if (result > 0) {
                out.println("<h3>Data inserted successfully!</h3>");
            } else {
                out.println("<h3>Data insertion failed.</h3>");      }
            ps.close();
            con.close();        } catch (Exception e) {
            out.println("Error: " + e.getMessage());
    }
}
}
```

Output :



Conclusion :

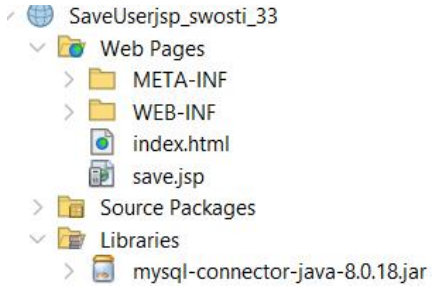
This program successfully demonstrates accessing and storing user data in a database using **JDBC within a servlet**. It shows how servlets process form data, connect to a database, and execute SQL queries, providing a basic understanding of database-driven web applications.

26. Write a jsp program to demonstrate data accessed using JDBC. Create a HTML form that accepts name, email, and address from user when the form is submitted. Data should be stored in table using POST method.

Objective :

To demonstrate how a **JSP page** uses **JDBC** to store user data (name, email, and address) submitted through an HTML form using the **POST method**.

Path :



Source Code :

index.html :

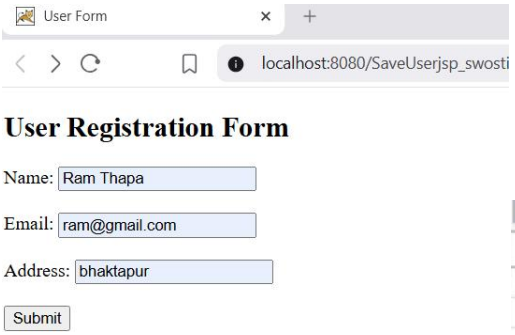
```
<body>
   <h2>User Registration Form</h2>
   <form action="save.jsp" method="post">
      Name: <input type="text" name="name" required><br><br>
      Email: <input type="email" name="email" required><br><br>
      Address: <input type="text" name="address" required><br><br>
      <input type="submit" value="Submit">
   </form>
</body>
```

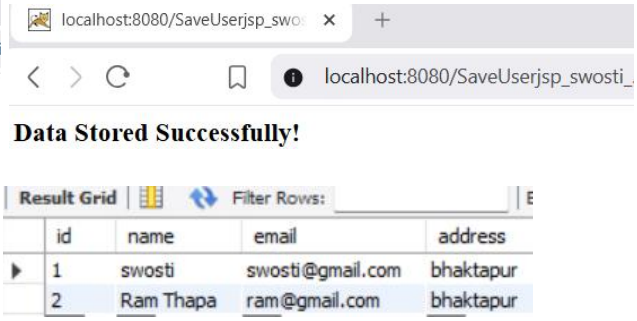
save.jsp :

```
<%@ page import="java.sql.*" %>
<%
   String name = request.getParameter("name");
   String email = request.getParameter("email");
   String address = request.getParameter("address");
```

```
try {
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/lab?useSSL=false",  "root", "root"        );
    String sql = "INSERT INTO user(name, email, address) VALUES (?, ?, ?)";
    PreparedStatement ps = con.prepareStatement(sql);
    ps.setString(1, name);
    ps.setString(2, email);
    ps.setString(3, address);
    ps.executeUpdate();
    out.println("<h3>Data Stored Successfully!</h3>");
    ps.close();
    con.close();
} catch(Exception e) {
    out.println("Error: " + e.getMessage()); }
%>
```

Output :





Conclusion :

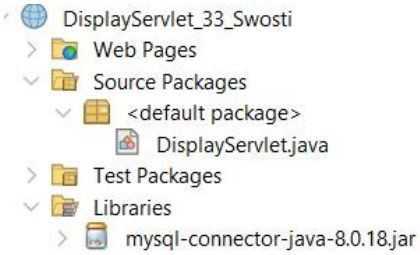
This program successfully shows how JSP processes form data and stores it in a database using JDBC, helping to understand basic database connectivity in JSP applications.

27. Write a java servlet program to display all the records in html table from database.

Objectives :

To develop a Java Servlet that connects to a database using JDBC, retrieves all records from a table, and displays them in an HTML table format on a web browser.

Path :



Source Code :

DisplayServlet.java :

```
import java.io.*;

import java.sql.*;

import javax.servlet.*;

import javax.servlet.http.*;

import javax.servlet.annotation.WebServlet;

@WebServlet("/DisplayServlet")

public class DisplayServlet extends HttpServlet { protected void doGet(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    out.println("<html><head><title>User Records</title></head><body>");

    out.println("<h2>User Details</h2>");

    out.println("<table border='1'>");

  out.println("<tr><th>ID</th><th>Name</th><th>Email</th><th>Address</th></tr>");

    try {

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

        Connection con = DriverManager.getConnection(
```

```
"jdbc:mysql://localhost:3306/lab?useSSL=false&serverTimezone=UTC","root",   "root" );
    Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM user_details");
            while (rs.next()) {
               out.println("<tr>");
               out.println("<td>" + rs.getInt("id") + "</td>");
               out.println("<td>" + rs.getString("name") + "</td>");
               out.println("<td>" + rs.getString("email") + "</td>");
               out.println("<td>" + rs.getString("address") + "</td>");
               out.println("</tr>");
            }
            rs.close();
            stmt.close();
            con.close();
        } catch (Exception e) {
            out.println("<tr><td colspan='4'>Error: " + e.getMessage() + "</td></tr>");
        }
        out.println("</table>");
        out.println("</body></html>");}}
```

Output :

Conclusion :

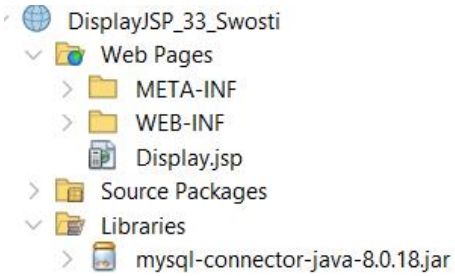
This program successfully demonstrates how a servlet accesses database records using JDBC and dynamically displays them in an HTML table. It helps in understanding database connectivity, result set processing, and dynamic web page generation using servlets.

28. Write a jsp program to display all the records in html table from database.

Objectives :

To demonstrate how JSP can be used with JDBC to retrieve records from a database and dynamically display them in an HTML table.

Path :



Source Code :

Display.jsp :

```
<%@ page import="java.sql.*" %>
<html>
<body>
<h2>User Details</h2>
<table border="1">
   <tr>
      <th>ID</th>
      <th>Name</th>
      <th>Email</th>
      <th>Address</th>
   </tr> <%
   try {
      Class.forName("com.mysql.cj.jdbc.Driver");
      Connection con = DriverManager.getConnection(
         "jdbc:mysql://localhost:3306/lab?useSSL=false&serverTimezone=UTC",
         "root","root"        );
      Statement stmt = con.createStatement();
```

```
        ResultSet rs = stmt.executeQuery("SELECT * FROM user");
        while (rs.next()) { %>
    <tr> <td><%= rs.getInt("id") %></td>
                <td><%= rs.getString("name") %></td>
                <td><%= rs.getString("email") %></td>
                <td><%= rs.getString("address") %></td> </tr>
<%  }
        rs.close();
        stmt.close();
        con.close();
    } catch (Exception e) {
        out.println("Error: " + e.getMessage());
    } %>
</table>
</body>
</html>
```

Output :

Conclusion :

This JSP program successfully fetches data from the database using JDBC and presents it in a structured HTML table, showing how JSP simplifies dynamic web page creation with database integration.