**Practical No : 01**

**a) Write a program to create a class and implement a default, overloaded and copy Constructor**

**Program:**

import java.util.stream.StreamSupport;

public class Constructor {

private String name;

private int age;

public Constructor() {

System.out.println("Default constructor called.");

name = "unknown";

age = 0;

}

public Constructor(String name, int age) {

System.out.println("overloading constructor called.");

this.name = name;

this.age = age;

}

public Constructor(Constructor other) {

System.out.println("copy constructor called.");

this.name = other.name;

this.age = other.age;

}

public void displayInfo() {

System.out.println("Name:" + name + ",age:" + age);

}

public static void main(String[] args) {

Constructor cst1 = new Constructor();

Constructor cst2 = new Constructor("Adam", 25);

Constructor cst3 = new Constructor(cst2);

cst1.displayInfo();

cst2.displayInfo();

cst3.displayInfo();

}

}

**Output:**

Default constructor called.

overloading constructor called.

copy constructor called.

Name:unknown,age:0

Name:Adam,age:25

Name:Adam,age:25

**b) Write a program to create a class and implement the concepts of Method Overloading**

**Program:**

public class MethodOverloadingExample {

// Method with two integer parameters

public int add(int a, int b) {

return a + b;

}

// Method with three integer parameters

public int add(int a, int b, int c) {

return a + b + c;

}

// Method with two double parameters

public double add(double a, double b) {

return a + b;

}

// Method with a String parameter

public String concatenate(String str1, String str2) {

return str1 + str2;

}

public static void main(String[] args) {

MethodOverloadingExample example = new MethodOverloadingExample();

// Call methods with different parameter combinations

int sum1 = example.add(5, 10);

int sum2 = example.add(5, 10, 15);

double sum3 = example.add(3.5, 7.5);

String result = example.concatenate("Hello, ", "World!");

// Display the results

System.out.println("Sum 1: " + sum1);

System.out.println("Sum 2: " + sum2);

System.out.println("Sum 3: " + sum3);

System.out.println("Concatenated String: " + result);

}

}

**Output:**

Sum 1: 15

Sum 2: 30

Sum 3: 11.0

Concatenated String: Hello, World!

**C) Write a program to create a class and implement the concepts of Static methods**

**Program:**

public class MathOperations {

// Static method to calculate the sum of two numbers

public static int add(int a, int b) {

return a + b;

}

// Static method to calculate the difference of two numbers

public static int subtract(int a, int b) {

return a - b;

}

// Static method to calculate the product of two numbers

public static int multiply(int a, int b) {

return a \* b;

}

public static void main(String[] args) {

// Calling static methods without creating an instance

int sumResult = MathOperations.add(5, 3);

int differenceResult = MathOperations.subtract(8, 4);

int productResult = MathOperations.multiply(2, 6);

// Displaying results

System.out.println("Sum: " + sumResult);

System.out.println("Difference: " + differenceResult);

System.out.println("Product: " + productResult);

}

}

**Output :**

Sum: 8

Difference: 4

Product: 12

**Practical No : 02**

**a) Write a program to implement the concepts of Inheritance and Method overriding**

**Program:**

**class Vehicle {**

**public void move() {**

**System.out.println("Vehicle is moving...");**

**}**

**}**

**class Car extends Vehicle {**

**@Override**

**public void move() {**

**System.out.println("Car is driving...");**

**}**

**}**

**class Bike extends Vehicle {**

**@Override**

**public void move() {**

**System.out.println("Bike is riding...");**

**}**

**}**

**public class Main {**

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

**Vehicle vehicle = new Vehicle();**

**Car car = new Car();**

**Bike bike = new Bike();**

**vehicle.move(); // Output: Vehicle is moving...**

**car.move(); // Output: Car is driving...**

**bike.move(); // Output: Bike is riding...**

**// Polymorphism example**

**Vehicle myVehicle1 = new Car(); // Upcasting**

**Vehicle myVehicle2 = new Bike(); // Upcasting**

**myVehicle1.move(); // Output: Car is driving... (due to overriding)**

**myVehicle2.move(); // Output: Bike is riding... (due to overriding)**

**}**

**}**

**Output:**

Vehicle is moving...

Car is driving...

Bike is riding...

Car is driving...

Bike is riding...

**b) Write a program to implement the concepts of Abstract classes and methods**

**Program :**

abstract class Shape {

// Abstract method with no implementation

public abstract double calculateArea();

}

class Square extends Shape {

private double sideLength;

public Square(double sideLength) {

this.sideLength = sideLength;

}

@Override

public double calculateArea() {

return sideLength \* sideLength;

}

}

class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

public class waqar {

public static void main(String[] args) {

// You cannot create objects of abstract classes

// Shape shape = new Shape(); // This will cause an error

Square square = new Square(5);

Circle circle = new Circle(3);

System.out.println("Area of square: " + square.calculateArea());

System.out.println("Area of circle: " + circle.calculateArea());

}

}

**Output :**

Area of square: 25.0

Area of circle: 28.274333882308138

**c) Write a program to implement the concept of interfaces.**

**Program :**

interface Shape {

// Interface methods (no implementation)

double getArea();

void draw();

}

class Circle implements Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double getArea() {

return Math.PI \* radius \* radius;

}

@Override

public void draw() {

System.out.println("Drawing a circle with radius " + radius);

}

}

class Rectangle implements Shape {

private double width, height;

public Rectangle(double width, double height) {

this.width = width;

this.height = height;

}

@Override

public double getArea() {

return width \* height;

}

@Override

public void draw() {

System.out.println("Drawing a rectangle with width " + width + " and height " + height);

}

}

public class Main {

public static void main(String[] args) {

Shape circle = new Circle(5);

Shape rectangle = new Rectangle(4, 6);

printShapeInfo(circle);

printShapeInfo(rectangle);

}

static void printShapeInfo(Shape shape) {

System.out.println("Shape area: " + shape.getArea());

shape.draw();

}

}

**Output :**

Shape area : 78.53981633974483

Drawing a circle with radius 5.0

Shape area : 24.0

Drawing a rectangle with width 4.0 and height 6.0

**Practical 7a**

**AIM:** Flow Layout

**Program:**

import javax.swing.\*;

import java.awt.\*;

public class FlowLayoutExample {

public static void main(String[] args) {

JFrame frame = new JFrame("Flow Layout Example");

frame.setLayout(new FlowLayout());

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

JButton button = new JButton("Button " + i);

frame.add(button);

}

frame.setSize(300, 200);

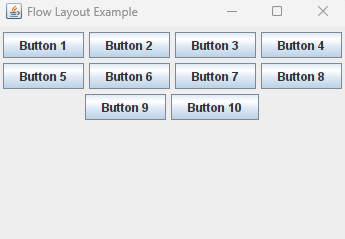
frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

**Output:**



**Practical 7b**

**AIM:** Grid Layout

**Program:**

import javax.swing.\*;

import java.awt.\*;

public class GridLayoutExample {

public static void main(String[] args) {

JFrame frame = new JFrame("Grid Layout Example");

frame.setLayout(new GridLayout(3, 3));

for (int i = 1; i <= 9; i++) {

JButton button = new JButton("Button " + i);

frame.add(button);

}

frame.setSize(300, 200);

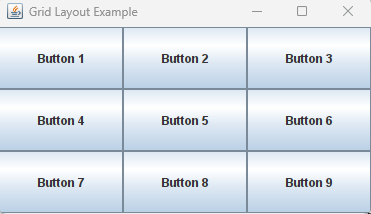
frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

**Output:**



**Practical 7c**

**AIM:** Border Layout

**Program:**

import javax.swing.\*;

import java.awt.\*;

public class BorderLayoutExample {

public static void main(String[] args) {

JFrame frame = new JFrame("Border Layout Example");

frame.setLayout(new BorderLayout());

JButton northButton = new JButton("North");

JButton southButton = new JButton("South");

JButton eastButton = new JButton("East");

JButton westButton = new JButton("West");

JButton centerButton = new JButton("Center");

frame.add(northButton, BorderLayout.NORTH);

frame.add(southButton, BorderLayout.SOUTH);

frame.add(eastButton, BorderLayout.EAST);

frame.add(westButton, BorderLayout.WEST);

frame.add(centerButton, BorderLayout.CENTER);

frame.setSize(400, 300);

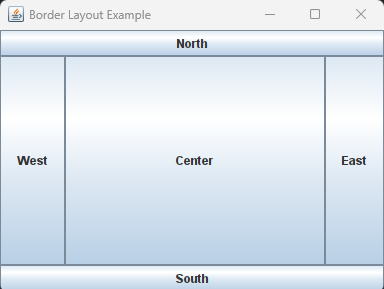
frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

**Output:**



**Practical 8a**

**AIM:** ActionEvent

**Program:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class ActionEventDemo extends JFrame {

private JButton button;

public ActionEventDemo() {

setTitle("ActionEvent Demo");

setSize(300, 200);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

button = new JButton("Click Me");

button.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(ActionEventDemo.this, "Button clicked!");

}

});

getContentPane().add(button, BorderLayout.CENTER);

setVisible(true);

}

public static void main(String[] args) {

new ActionEventDemo();

}

}

**Output:**

