**JAVA LAB 5**

PiyushRaj2200290100114

Design an interface named BankAccount with methods deposit(double amount) and withdraw(double amount). Implement this interface in two classes: SavingsAccount and CurrentAccount. The SavingsAccount class should apply an interest rate of 5% on deposits, and the CurrentAccount class should not apply any interest. Write a Java program to demonstrate the usage of these classes by performing deposit and withdrawal operations on both types of accounts.

Code:-

interface BankAccount {

    void deposit(double amount);

    void withdraw(double amount);

}

class SavingsAccount implements BankAccount {

    private double balance;

    private final double interestRate = 0.05;

    @Override

    public void deposit(double amount) {

        balance += amount + (amount \* interestRate);

        System.out.println("Amount Deposited: " + amount);

        System.out.println("Savings Account Balance: " + balance);

    }

    @Override

    public void withdraw(double amount) {

        if (balance >= amount) {

            balance -= amount;

            System.out.println("Amount Withdrawen: " + amount);

            System.out.println("Savings Account Balance: " + balance);

        } else {

            System.out.println("Insufficient funds.");

        }

    }

}

class CurrentAccount implements BankAccount {

    private double balance;

    @Override

    public void deposit(double amount) {

        balance += amount;

        System.out.println("Amount Deposited: " + amount);

        System.out.println("Current Account Balance: " + balance);

    }

    @Override

    public void withdraw(double amount) {

        if (balance >= amount) {

            balance -= amount;

            System.out.println("Amount Withdrawen: " + amount);

            System.out.println("Current Account Balance: " + balance);

        } else {

            System.out.println("Insufficient funds.");

        }

    }

}

public class Code1 {

    public static void main(String[] args) {

        SavingsAccount savingsAccount = new SavingsAccount();

        CurrentAccount currentAccount = new CurrentAccount();

        savingsAccount.deposit(1000);

        savingsAccount.withdraw(500);

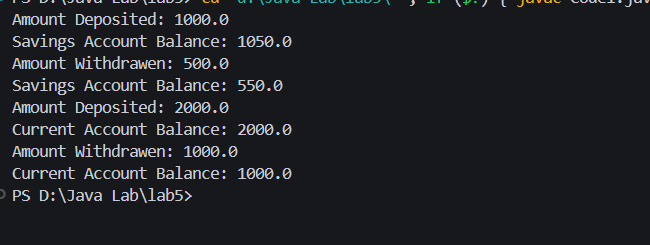
        currentAccount.deposit(2000);

        currentAccount.withdraw(1000);

    }

}

**Output:-**

****

2. Define an interface named Resizable with a method resize(double factor) that modifies the dimensions of a shape by the given factor. Implement this interface in classes Circle and Rectangle, allowing them to be resized accordingly. Ensure that resizing maintains the shape's proportions. Write a Java program to demonstrate the resizing functionality of both shapes.

Code:-

import java.text.DecimalFormat;

interface Resizable {

    void resize(double factor);

}

class Circle implements Resizable {

    private double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    @Override

    public void resize(double factor) {

        radius \*= Math.sqrt(factor);

    }

    public double getRadius() {

        return radius;

    }

}

class Rectangle implements Resizable {

    private double width;

    private double height;

    public Rectangle(double width, double height) {

        this.width = width;

        this.height = height;

    }

    @Override

    public void resize(double factor) {

        double ratio = Math.sqrt(factor);

        width \*= ratio;

        height \*= ratio;

    }

    public double getWidth() {

        return width;

    }

    public double getHeight() {

        return height;

    }

}

public class Code2 {

    public static void main(String[] args) {

        DecimalFormat df = new DecimalFormat("#.##");

        Circle circle = new Circle(5);

        System.out.println("Circle Radius: " + df.format(circle.getRadius()));

        circle.resize(2);

        System.out.println("Resized Circle Radius: " + df.format(circle.getRadius()));

        Rectangle rectangle = new Rectangle(4, 6);

        System.out.println("Rectangle Width: " + df.format(rectangle.getWidth()) + ", Height: " + df.format(rectangle.getHeight()));

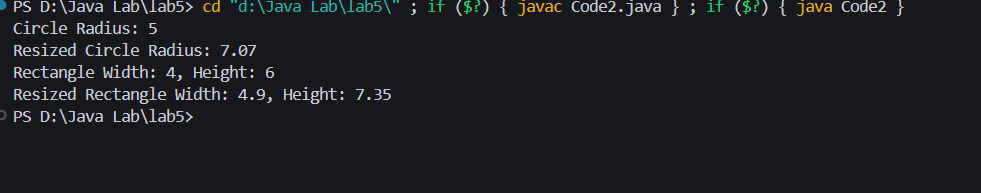
        rectangle.resize(1.5);

        System.out.println("Resized Rectangle Width: " + df.format(rectangle.getWidth()) + ", Height: " + df.format(rectangle.getHeight()));

    }

}

Output:-



3.  Create an interface named Worker with a method work(). Define two more interfaces, Eater with a method eat() and Sleeper with a method sleep(). Now, create a class named Human implementing all three interfaces. Write a Java program to demonstrate how Human class exhibits multiple inheritance through interfaces.

Code:-

interface Worker {

    void work();

}

interface Eater {

    void eat();

}

interface Sleeper {

    void sleep();

}

class Human implements Worker, Eater, Sleeper {

    @Override

    public void work() {

        System.out.println("Human is working.");

    }

    @Override

    public void eat() {

        System.out.println("Human is eating.");

    }

    @Override

    public void sleep() {

        System.out.println("Human is sleeping.");

    }

}

public class Code3 {

    public static void main(String[] args) {

        Human human = new Human();

        human.work();

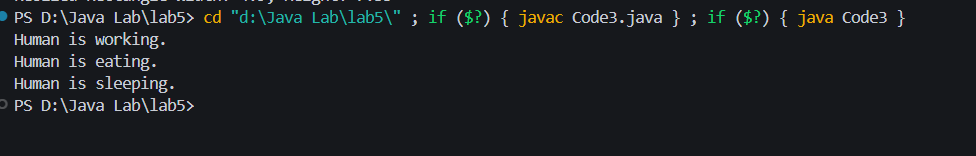
        human.eat();

        human.sleep();

    }

}

Output:-



4.  Write a program to implement Stack using Java.

Code:-

import java.util.EmptyStackException;

class Stack {

    private int maxSize;

    private int[] stackArray;

    private int top;

    public Stack(int size) {

        maxSize = size;

        stackArray = new int[maxSize];

        top = -1;

    }

    public void push(int value) {

        if (top == maxSize - 1) {

            System.out.println("Stack is full. Cannot push element.");

            return;

        }

        stackArray[++top] = value;

    }

    public void pop() {

        if (top == -1) {

            throw new EmptyStackException();

        }

        System.out.println("Popped Element is: " + stackArray[top--]);

        maxSize--;

    }

    public void peek() {

        if (top == -1) {

            throw new EmptyStackException();

        }

        System.out.println("Stack Top is: " + stackArray[top]);

    }

}

public class Code4 {

    public static void main(String[] args) {

        Stack stack = new Stack(5);

        stack.push(50);

        stack.push(40);

        stack.push(30);

        stack.push(20);

        stack.push(10);

        stack.peek();

        stack.pop();

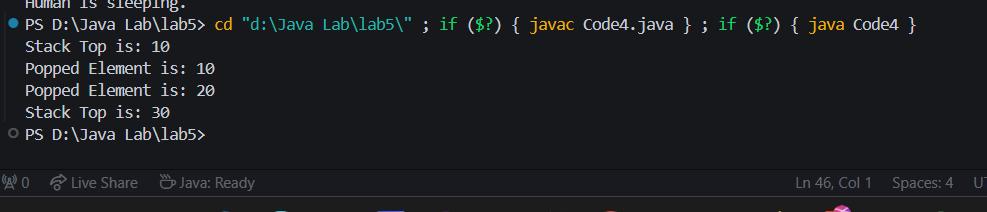
        stack.pop();

        stack.peek();

    }

}

Output:-



5. Write a program to implement Queue using Java.

Code:-

import java.util.EmptyStackException;

class Queue {

    private int maxSize;

    private int[] queueArray;

    private int front;

    private int rear;

    private int nItems;

    public Queue(int size) {

        maxSize = size;

        queueArray = new int[maxSize];

        front = 0;

        rear = -1;

        nItems = 0;

    }

    public void enqueue(int value) {

        if (rear == maxSize - 1) {

            System.out.println("Queue is full. Cannot enqueue element.");

            return;

        }

        queueArray[++rear] = value;

        nItems++;

    }

    public void dequeue() {

        if (nItems == 0) {

            throw new EmptyStackException();

        }

        int temp = queueArray[front++];

        if (front == maxSize) {

            front = 0;

        }

        nItems--;

        System.out.println("Dequeue element is : "+temp);

    }

    public void peek() {

        if (nItems==0) {

            throw new EmptyStackException();

        }

        System.out.println("Queue Front is : "+ queueArray[front]);

    }

}

public class Code5 {

    public static void main(String[] args) {

        Queue queue = new Queue(5);

        queue.enqueue(10);

        queue.enqueue(20);

        queue.enqueue(30);

       queue.peek();

       queue.dequeue();

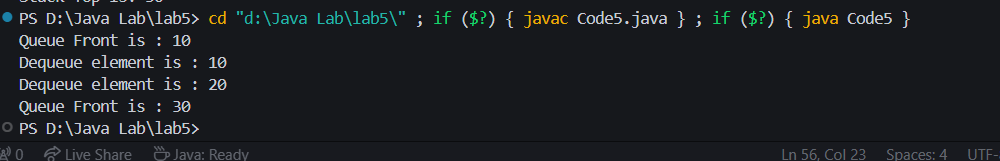
       queue.dequeue();

       queue.peek();

    }

}

Output:-



6. Define an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Extend this class to create a concrete subclass Triangle. Implement methods to calculate the area and perimeter of a triangle. Additionally, implement a method to determine if the triangle is equilateral, isosceles, or scalene. Write a Java program to demonstrate the functionality of the Triangle class

Code:-

abstract class Shape {

    public abstract double calculateArea();

    public abstract double calculatePerimeter();

}

class Triangle extends Shape {

    private double side1;

    private double side2;

    private double side3;

    public Triangle(double side1, double side2, double side3) {

        this.side1 = side1;

        this.side2 = side2;

        this.side3 = side3;

    }

    @Override

    public double calculateArea() {

        double s = (side1 + side2 + side3) / 2;

        return Math.sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

    }

    @Override

    public double calculatePerimeter() {

        return side1 + side2 + side3;

    }

    public String getType() {

        if (side1 == side2 && side2 == side3) {

            return "Equilateral";

        } else if (side1 == side2 || side2 == side3 || side1 == side3) {

            return "Isosceles";

        } else {

            return "Scalene";

        }

    }

}

public class Code6 {

    public static void main(String[] args) {

        Triangle triangle = new Triangle(3, 4, 5);

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

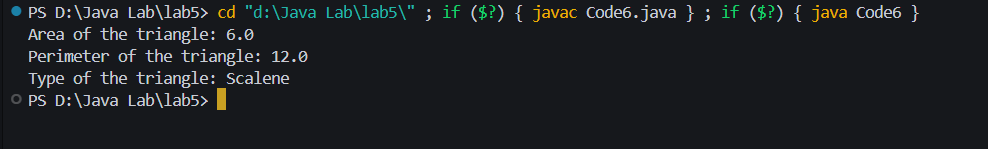
        System.out.println("Perimeter of the triangle: " + triangle.calculatePerimeter());

        System.out.println("Type of the triangle: " + triangle.getType());

    }

}

Output:-



7.  Design an abstract class Employee with instance variables name, age, and salary. Include concrete methods work() and rest(). Extend this class to create two concrete subclasses: Manager and Worker. The Manager class should have an additional instance variable department, and the Worker class should have an additional instance variable hoursWorked. Write a Java program to demonstrate the usage of these classes and their specific attributes.

Code:-

abstract class Employee {

    protected String name;

    protected int age;

    protected double salary;

    public Employee(String name, int age, double salary) {

        this.name = name;

        this.age = age;

        this.salary = salary;

    }

    public abstract void work();

    public void rest() {

        System.out.println(name + " is resting.");

    }

}

class Manager extends Employee {

    private String department;

    public Manager(String name, int age, double salary, String department) {

        super(name, age, salary);

        this.department = department;

    }

    @Override

    public void work() {

        System.out.println(name + " is managing the " + department + " department.");

    }

}

class Worker extends Employee {

    private int hoursWorked;

    public Worker(String name, int age, double salary, int hoursWorked) {

        super(name, age, salary);

        this.hoursWorked = hoursWorked;

    }

    @Override

    public void work() {

        System.out.println(name + " is working for " + hoursWorked + " hours.");

    }

}

public class Code7 {

    public static void main(String[] args) {

        Manager manager = new Manager("Rahul", 35, 5000, "Sales");

        Worker worker = new Worker("Suraj", 28, 3000, 8);

        manager.work();

        manager.rest();

        worker.work();

        worker.rest();

    }

}

Output:-

