**22MCA0139**

**Rajat Singh**

**Object Oriented Programming with Java**

**Digital Assignment**

**Code Link (Github): (**[**Link**](https://github.com/rajat-singh1999/MCA/tree/main/SEM_2/java/DA)**) 🡨Click on the link to see the codes used for this assignment.**

**Q1) A class called Circle is designed as given below. It contains one private instance variable: radius (of type double). Two constructors, Two public methods: getRadius() and findArea().Define the main class and test the program.**

**Solution:**

class Circle {

    private double radius;

    public Circle() {

        radius = 0.0;

    }

    public Circle(double radius) {

        this.radius = radius;

    }

    public double getRadius() {

        return radius;

    }

    public double findArea() {

        return Math.PI \* radius \* radius;

    }

}

public class CircleMain {

    public static void main(String[] args) {

        Circle circle1 = new Circle();

        System.out.println("Circle 1 - Radius: " + circle1.getRadius());

        System.out.println("Circle 1 - Area: " + circle1.findArea());

        Circle circle2 = new Circle(5.0);

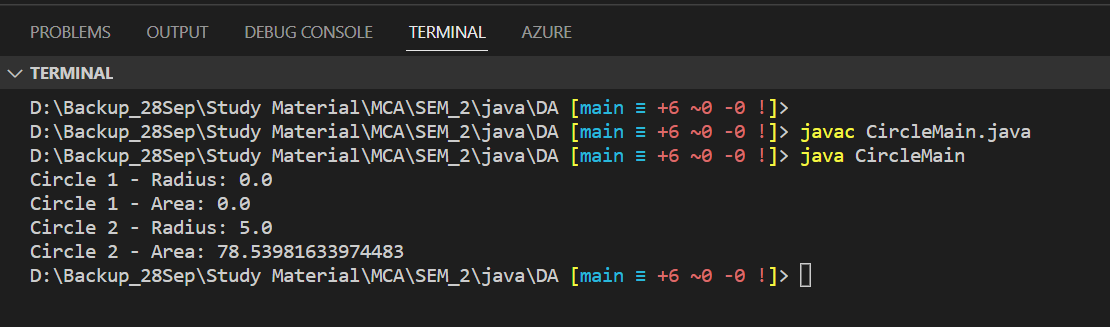
        System.out.println("Circle 2 - Radius: " + circle2.getRadius());

        System.out.println("Circle 2 - Area: " + circle2.findArea());

    }

}

**Output:**

****

**Q2) Create a Line class with 2 points (say, x and y) as data members. Write suitable constructors. Write a method to compute the length of the line. Calculate length using Euclidian distance formula.**

**Solution:**

import java.lang.Math;

class Line {

    private double x1, y1;  // First point

    private double x2, y2;  // Second point

    public Line() {

        x1 = y1 = x2 = y2 = 0.0;

    }

    public Line(double x1, double y1, double x2, double y2) {

        this.x1 = x1;

        this.y1 = y1;

        this.x2 = x2;

        this.y2 = y2;

    }

    public double computeLength() {

        double length = Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

        return length;

    }

}

public class EuclidianDistance {

    public static void main(String[] args) {

        Line line1 = new Line();

        System.out.println("Line 1 Length: " + line1.computeLength());

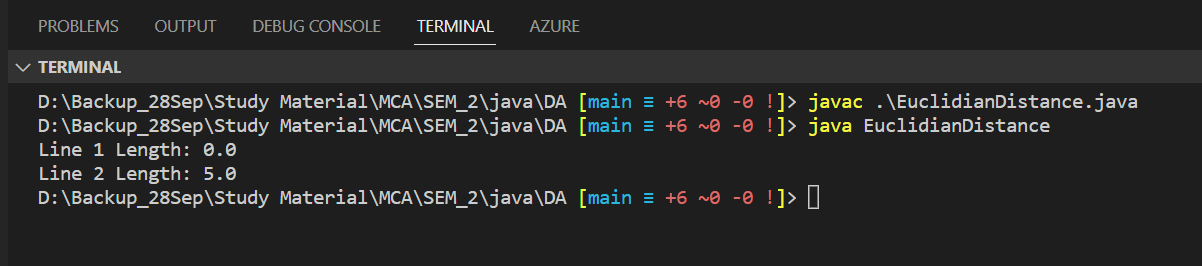
        Line line2 = new Line(1.0, 2.0, 4.0, 6.0);

        System.out.println("Line 2 Length: " + line2.computeLength());

    }

}

**Output:**

****

**Q3) Develop a Java application to generate Electricity bills. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, type of EB connection (i.e. domestic or commercial).**

**Compute the bill amount using the following tariff.**

**If the type of the EB connection is domestic, calculate the amount to be paid as follows:**

**• First 100 units - Rs. 1 per unit**

**• 101-200 units - Rs. 2.50 per unit**

**• 201 -500 units - Rs. 4 per unit**

**• > 501 units - Rs. 6 per unit**

**If the type of the EB connection is commercial, calculate the amount to be paid as follows:**

**• First 100 units - Rs. 2 per unit**

**• 101-200 units - Rs. 4.50 per unit**

**• 201 -500 units - Rs. 6 per unit**

**• > 501 units - Rs. 7 per unit**

**Solution:**

import java.util.Scanner;

class ElectricityBill {

    private int consumerNo;

    private String consumerName;

    private double previousMonthReading;

    private double currentMonthReading;

    private String connectionType;

    public ElectricityBill(int consumerNo, String consumerName, double previousMonthReading, double currentMonthReading, String connectionType) {

        this.consumerNo = consumerNo;

        this.consumerName = consumerName;

        this.previousMonthReading = previousMonthReading;

        this.currentMonthReading = currentMonthReading;

        this.connectionType = connectionType;

    }

    public double calculateBillAmount() {

        double unitsConsumed = currentMonthReading - previousMonthReading;

        double billAmount = 0.0;

        if (connectionType.equalsIgnoreCase("domestic")) {

            if (unitsConsumed <= 100) {

                billAmount = unitsConsumed \* 1.0;

            } else if (unitsConsumed <= 200) {

                billAmount = 100 \* 1.0 + (unitsConsumed - 100) \* 2.5;

            } else if (unitsConsumed <= 500) {

                billAmount = 100 \* 1.0 + 100 \* 2.5 + (unitsConsumed - 200) \* 4.0;

            } else {

                billAmount = 100 \* 1.0 + 100 \* 2.5 + 300 \* 4.0 + (unitsConsumed - 500) \* 6.0;

            }

        } else if (connectionType.equalsIgnoreCase("commercial")) {

            if (unitsConsumed <= 100) {

                billAmount = unitsConsumed \* 2.0;

            } else if (unitsConsumed <= 200) {

                billAmount = 100 \* 2.0 + (unitsConsumed - 100) \* 4.5;

            } else if (unitsConsumed <= 500) {

                billAmount = 100 \* 2.0 + 100 \* 4.5 + (unitsConsumed - 200) \* 6.0;

            } else {

                billAmount = 100 \* 2.0 + 100 \* 4.5 + 300 \* 6.0 + (unitsConsumed - 500) \* 7.0;

            }

        }

        return billAmount;

    }

    public void displayBill() {

        System.out.println("Electricity Bill");

        System.out.println("Consumer No.: " + consumerNo);

        System.out.println("Consumer Name: " + consumerName);

        System.out.println("Previous Month Reading: " + previousMonthReading);

        System.out.println("Current Month Reading: " + currentMonthReading);

        System.out.println("Connection Type: " + connectionType);

        System.out.println("Bill Amount: " + calculateBillAmount());

    }

}

public class ElectricBilling {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter Consumer No.: ");

        int consumerNo = scanner.nextInt();

        System.out.print("Enter Consumer Name: ");

        String consumerName = scanner.next();

        System.out.print("Enter Previous Month Reading: ");

        double previousMonthReading = scanner.nextDouble();

        System.out.print("Enter Current Month Reading: ");

        double currentMonthReading = scanner.nextDouble();

        System.out.print("Enter Connection Type (domestic or commercial): ");

        String connectionType = scanner.next();

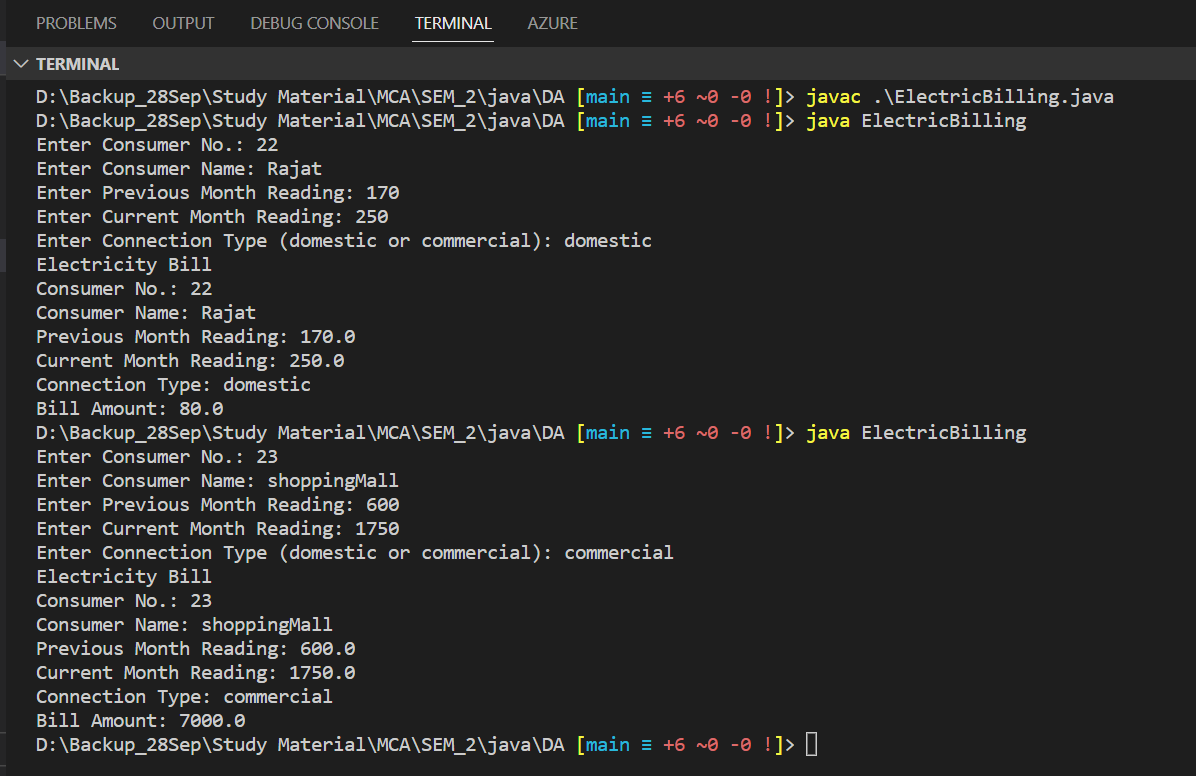
        ElectricityBill bill = new ElectricityBill(consumerNo, consumerName, previousMonthReading, currentMonthReading, connectionType);

        bill.displayBill();

        scanner.close();

    }

**Output:**

****

**Q4) Define a class ‘Student\_Results’ with the following attributes: Reg.No, Name, branch & CGPA. You have to define appropriate constructors and methods. A Software company directly calls the first two toppers for the HR round in each branch. Write a Java program to display the shortlisted students’ details.**

**Solution:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Student\_Results {

    private int regNo;

    private String name;

    private String branch;

    private double cgpa;

    public Student\_Results(int regNo, String name, String branch, double cgpa) {

        this.regNo = regNo;

        this.name = name;

        this.branch = branch;

        this.cgpa = cgpa;

    }

    public int getRegNo() {

        return regNo;

    }

    public String getName() {

        return name;

    }

    public String getBranch() {

        return branch;

    }

    public double getCgpa() {

        return cgpa;

    }

    public static List<Student\_Results> getTopperStudents(List<Student\_Results> students, String branch, int count) {

        List<Student\_Results> topperStudents = new ArrayList<>();

        for (Student\_Results student : students) {

            if (student.getBranch().equalsIgnoreCase(branch)) {

                topperStudents.add(student);

            }

        }

        Collections.sort(topperStudents, new Comparator<Student\_Results>() {

            @Override

            public int compare(Student\_Results s1, Student\_Results s2) {

                return Double.compare(s2.getCgpa(), s1.getCgpa());

            }

        });

        topperStudents = topperStudents.subList(0, Math.min(count, topperStudents.size()));

        return topperStudents;

    }

    public String toString() {

        return "Reg.No: " + regNo + "\nName: " + name + "\nBranch: " + branch + "\nCGPA: " + cgpa + "\n";

    }

}

public class HR\_Shortlisting {

    public static void main(String[] args) {

        List<Student\_Results> students = new ArrayList<>();

        // Add sample student records

        students.add(new Student\_Results(1, "Shankar", "CSE", 9.5));

        students.add(new Student\_Results(2, "Bhola", "CSE", 9.8));

        students.add(new Student\_Results(3, "Ravi", "ECE", 9.2));

        students.add(new Student\_Results(4, "Kishan", "ECE", 9.7));

        students.add(new Student\_Results(5, "Chandu", "CSE", 9.6));

        students.add(new Student\_Results(6, "Chatur", "EEE", 9.3));

        String branch = "CSE"; // Specify the branch for shortlisting

        int count = 2; // Specify the number of top students to select

        List<Student\_Results> shortlistedStudents = Student\_Results.getTopperStudents(students, branch, count);

        System.out.println("Shortlisted Students' Details:");

        for (Student\_Results student : shortlistedStudents) {

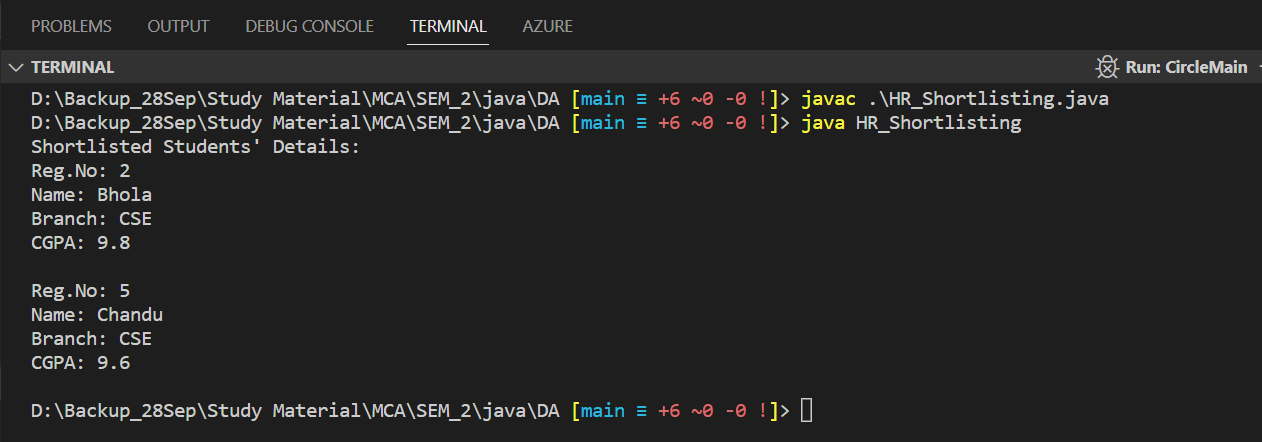
            System.out.println(student);

        }

    }

}

**Output:**

****

**Q5) A group of ‘n’ candidates has applied for faculty recruitment. Their Name, qualification, experience, dob and gender are stored in a class “Recruitment”. Write a java program to sort the objects based on their experience and display their details. If the experience is equal, use the name as the second sorting criteria.**

**Solution:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Recruitment {

    private String name;

    private String qualification;

    private int experience;

    private String dob;

    private String gender;

    public Recruitment(String name, String qualification, int experience, String dob, String gender) {

        this.name = name;

        this.qualification = qualification;

        this.experience = experience;

        this.dob = dob;

        this.gender = gender;

    }

    public String getName() {

        return name;

    }

    public String getQualification() {

        return qualification;

    }

    public int getExperience() {

        return experience;

    }

    public String getDob() {

        return dob;

    }

    public String getGender() {

        return gender;

    }

    public String toString() {

        return "Name: " + name + "\nQualification: " + qualification + "\nExperience: " + experience +

                "\nDOB: " + dob + "\nGender: " + gender + "\n";

    }

}

public class RecruitmentMain {

    public static void main(String[] args) {

        List<Recruitment> candidates = new ArrayList<>();

        // Add sample candidate records

        candidates.add(new Recruitment("John", "Ph.D.", 5, "1990-01-01", "Male"));

        candidates.add(new Recruitment("Alice", "M.Tech.", 3, "1992-05-15", "Female"));

        candidates.add(new Recruitment("Bob", "B.Tech.", 5, "1991-08-20", "Male"));

        candidates.add(new Recruitment("Emma", "Ph.D.", 2, "1993-04-10", "Female"));

        candidates.add(new Recruitment("Sam", "M.Tech.", 3, "1990-11-30", "Male"));

        Collections.sort(candidates, new Comparator<Recruitment>() {

            @Override

            public int compare(Recruitment r1, Recruitment r2) {

                if (r1.getExperience() == r2.getExperience()) {

                    return r1.getName().compareTo(r2.getName());

                }

                return Integer.compare(r2.getExperience(), r1.getExperience());

            }

        });

        System.out.println("Sorted Candidates' Details:");

        for (Recruitment candidate : candidates) {

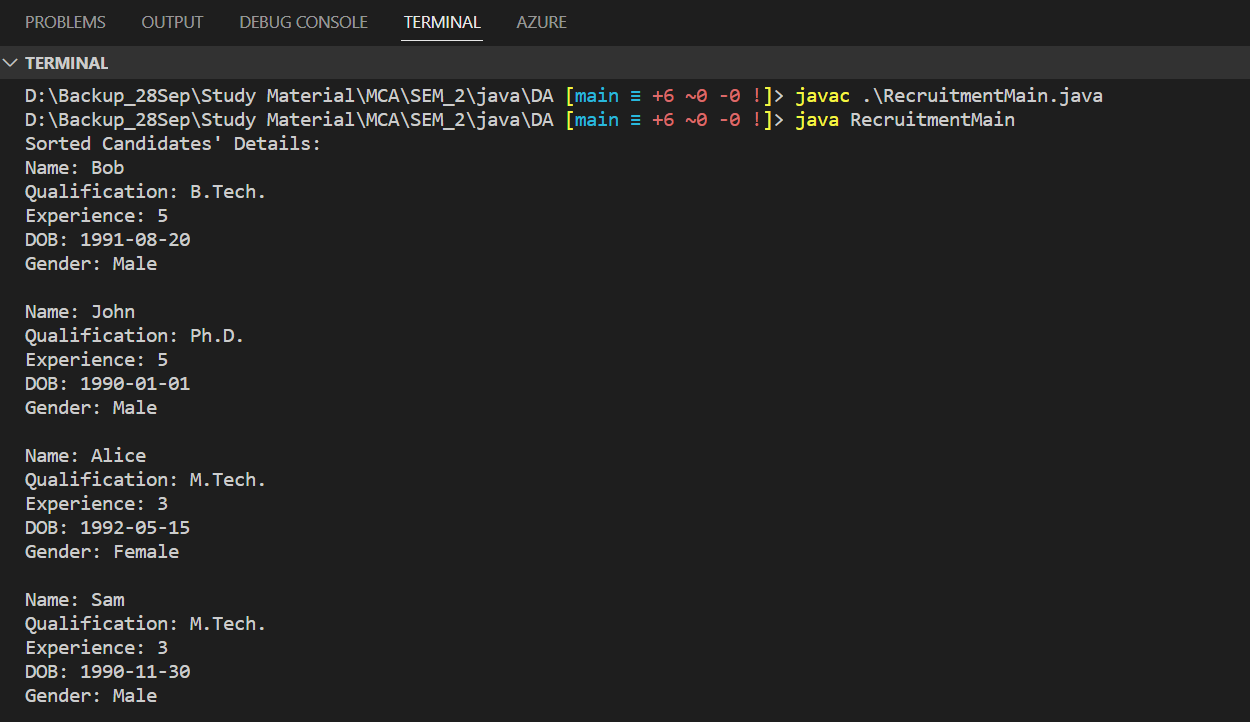
            System.out.println(candidate);

        }

    }

}

**Output:**

****

**A picture containing text, font, screenshot

Description automatically generated**

**Q6) Create a class ComplexArithmetic to perform Complex number arithmetic operations. A Complex object should be created in the main method enclosed within a class ComplexMain. The complex arithmetic operations should be executed in a menu-driven way.**

**Solution:**

import java.util.Scanner;

class Complex {

    private double real;

    private double imaginary;

    public Complex(double real, double imaginary) {

        this.real = real;

        this.imaginary = imaginary;

    }

    public double getReal() {

        return real;

    }

    public double getImaginary() {

        return imaginary;

    }

    public Complex add(Complex other) {

        double realSum = real + other.real;

        double imaginarySum = imaginary + other.imaginary;

        return new Complex(realSum, imaginarySum);

    }

    public Complex subtract(Complex other) {

        double realDiff = real - other.real;

        double imaginaryDiff = imaginary - other.imaginary;

        return new Complex(realDiff, imaginaryDiff);

    }

    public Complex multiply(Complex other) {

        double realProduct = (real \* other.real) - (imaginary \* other.imaginary);

        double imaginaryProduct = (real \* other.imaginary) + (imaginary \* other.real);

        return new Complex(realProduct, imaginaryProduct);

    }

    public Complex divide(Complex other) {

        double divisor = (other.real \* other.real) + (other.imaginary \* other.imaginary);

        double realQuotient = ((real \* other.real) + (imaginary \* other.imaginary)) / divisor;

        double imaginaryQuotient = ((imaginary \* other.real) - (real \* other.imaginary)) / divisor;

        return new Complex(realQuotient, imaginaryQuotient);

    }

    public String toString() {

        return real + " + " + imaginary + "i";

    }

}

public class ComplexMain {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Read the first complex number

        System.out.print("Enter the real part of the first complex number: ");

        double real1 = scanner.nextDouble();

        System.out.print("Enter the imaginary part of the first complex number: ");

        double imaginary1 = scanner.nextDouble();

        Complex complex1 = new Complex(real1, imaginary1);

        System.out.print("Enter the real part of the second complex number: ");

        double real2 = scanner.nextDouble();

        System.out.print("Enter the imaginary part of the second complex number: ");

        double imaginary2 = scanner.nextDouble();

        Complex complex2 = new Complex(real2, imaginary2);

        while (true) {

            System.out.println("\nMenu:");

            System.out.println("1. Add");

            System.out.println("2. Subtract");

            System.out.println("3. Multiply");

            System.out.println("4. Divide");

            System.out.println("5. Exit");

            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();

            Complex result;

            switch (choice) {

                case 1:

                    result = complex1.add(complex2);

                    System.out.println("Result: " + result);

                    break;

                case 2:

                    result = complex1.subtract(complex2);

                    System.out.println("Result: " + result);

                    break;

                case 3:

                    result = complex1.multiply(complex2);

                    System.out.println("Result: " + result);

                    break;

                case 4:

                    result = complex1.divide(complex2);

                    System.out.println("Result: " + result);

                    break;

                case 5:

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

                    scanner.close();

                    return;

                default:

                    System.out.println("Invalid choice. Try again.");

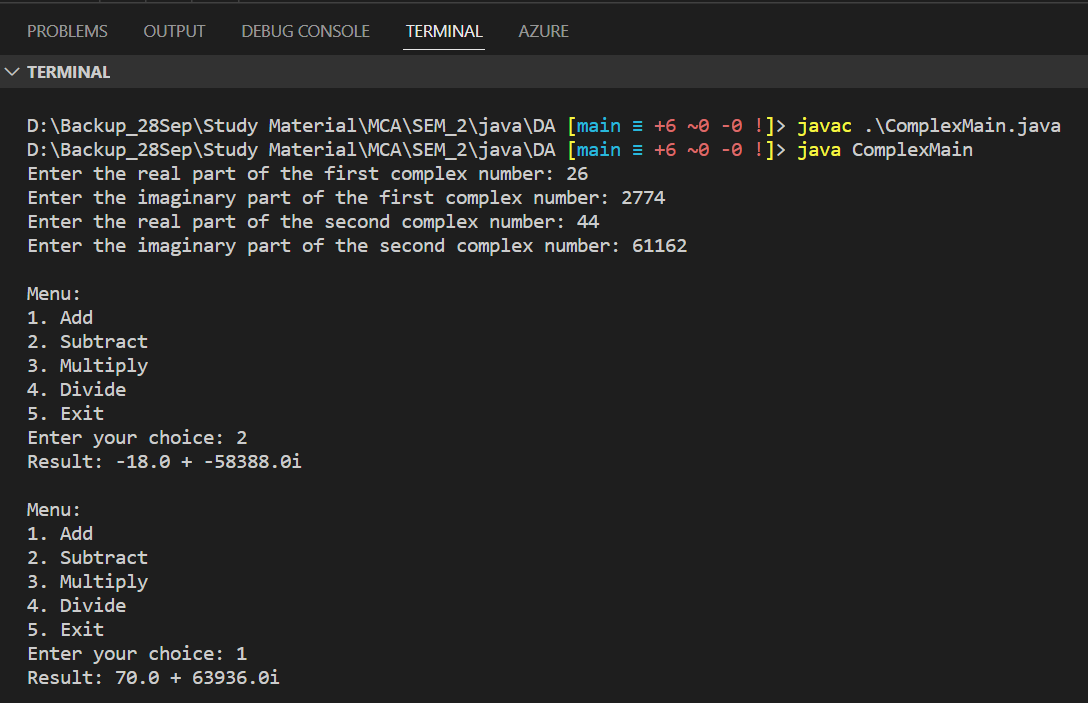
            }

        }

    }

}

**Output:**

****

**A picture containing text, screenshot, software, multimedia software

Description automatically generated**

**Q7) Develop a class TelephoneIndex with two String objects as members. One should hold subscribers’ names, and the other should hold their phone numbers. The class should have appropriate constructor, input and display methods. Create an array of objects for TelephoneIndex and do the following:**

**a) Your program should ask the user to enter a name or the first few characters of a name to search for it in the array and display the corresponding phone number.**

**b) The program should display all the names that match the user’s input and corresponding phone numbers.**

**Solution:**

import java.util.Scanner;

class TelephoneIndex {

    private String[] names;

    private String[] phoneNumbers;

    public TelephoneIndex(int size) {

        names = new String[size];

        phoneNumbers = new String[size];

    }

    public void input() {

        Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < names.length; i++) {

            System.out.print("Enter subscriber's name: ");

            names[i] = scanner.nextLine();

            System.out.print("Enter phone number: ");

            phoneNumbers[i] = scanner.nextLine();

        }

    }

    public void display() {

        System.out.println("Telephone Index:");

        for (int i = 0; i < names.length; i++) {

            System.out.println("Name: " + names[i] + ", Phone Number: " + phoneNumbers[i]);

        }

    }

    public void searchByName(String searchName) {

        boolean found = false;

        System.out.println("Search results:");

        for (int i = 0; i < names.length; i++) {

            if (names[i].toLowerCase().contains(searchName.toLowerCase())) {

                System.out.println("Name: " + names[i] + ", Phone Number: " + phoneNumbers[i]);

                found = true;

            }

        }

        if (!found) {

            System.out.println("No matching names found.");

        }

    }

}

public class TelephoneIndexMain {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of subscribers: ");

        int size = scanner.nextInt();

        scanner.nextLine(); // Consume the newline character

        TelephoneIndex telephoneIndex = new TelephoneIndex(size);

        telephoneIndex.input();

        System.out.println();

        telephoneIndex.display();

        System.out.print("\nEnter a name or the first few characters of a name to search: ");

        String searchName = scanner.nextLine();

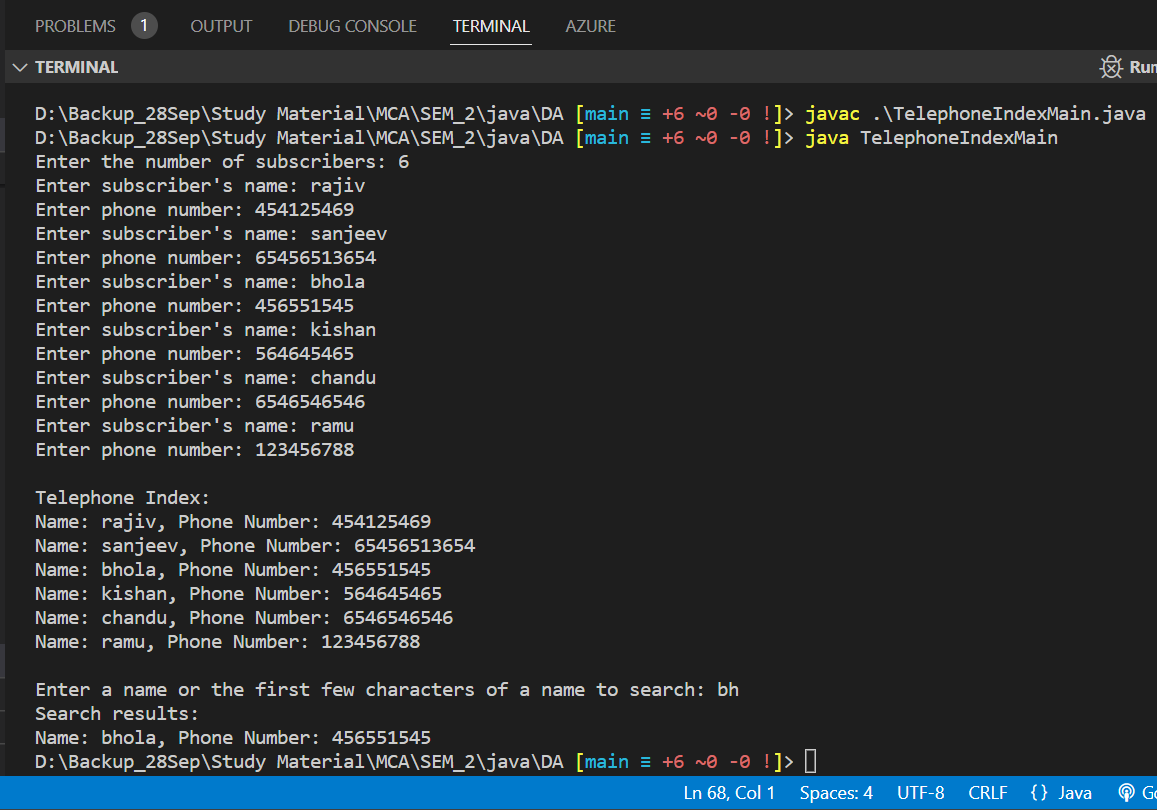
        telephoneIndex.searchByName(searchName);

        scanner.close();

    }

}

**Output:**

****

**Q8) Implement profit/loss calculation for each of the products provided and also the total profit/loss using inheritance in java. The following classes and class members have to be part of the solution 1) ProfitLossCalculation -- child class of productData which implements the function calculate() to calculate the profit or loss 2) super class -- ProductData -- which stores the product information and implements printProduct() to print product information.**

**Solution:**

class ProductData {

    public String productName;

    public double costPrice;

    public double sellingPrice;

    public ProductData(String productName, double costPrice, double sellingPrice) {

        this.productName = productName;

        this.costPrice = costPrice;

        this.sellingPrice = sellingPrice;

    }

    public void printProduct() {

        System.out.println("Product Name: " + productName);

        System.out.println("Cost Price: " + costPrice);

        System.out.println("Selling Price: " + sellingPrice);

    }

}

class ProfitLossCalculation extends ProductData {

    public ProfitLossCalculation(String productName, double costPrice, double sellingPrice) {

        super(productName, costPrice, sellingPrice);

    }

    public void calculate() {

        double profitLoss = sellingPrice - costPrice;

        String result = (profitLoss > 0) ? "Profit" : "Loss";

        System.out.println("Profit/Loss: " + result);

        System.out.println("Amount: " + Math.abs(profitLoss));

    }

}

public class ProfitLoss {

    public static void main(String[] args) {

        // Create product instances

        ProductData product1 = new ProductData("Product 1", 100.0, 120.0);

        ProductData product2 = new ProductData("Product 2", 50.0, 40.0);

        ProductData product3 = new ProductData("Product 3", 200.0, 180.0);

        // Print product information

        product1.printProduct();

        System.out.println();

        product2.printProduct();

        System.out.println();

        product3.printProduct();

        System.out.println();

        // Calculate profit/loss for each product

        ProfitLossCalculation p1 = new ProfitLossCalculation("Product 1", 100.0, 120.0);

        p1.calculate();

        System.out.println();

        ProfitLossCalculation p2 = new ProfitLossCalculation("Product 2", 50.0, 40.0);

        p2.calculate();

        System.out.println();

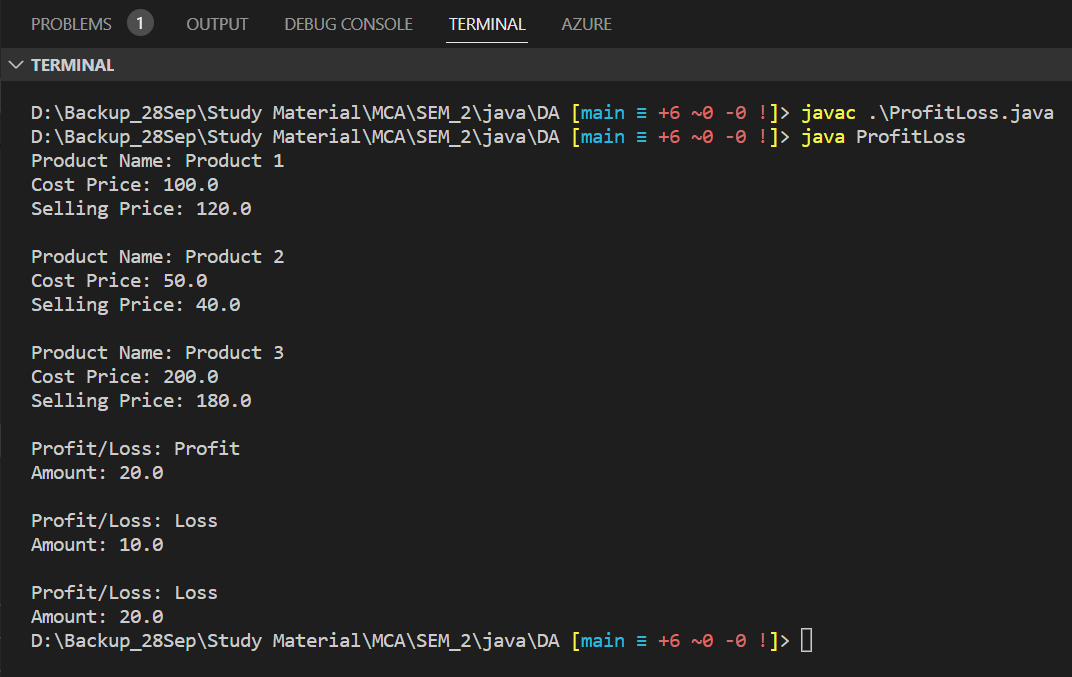
        ProfitLossCalculation p3 = new ProfitLossCalculation("Product 3", 200.0, 180.0);

        p3.calculate();

    }

}

**Output:**

****

**Q9) Create a class CARD with cardno, cust\_name, bank\_name as data members. Provide suitable constructors to create a CARD object. Create another class Creditcard that inherits the properties of CARD. In addition to the CARD properties, Creditcard will have an additional ‘limit’ data member. Provide suitable constructors. Include methods ‘display’ and ‘use’. Display method should display the details of a credit card. The use method in Creditcard should decrease the limit by the amount used. Make sure that the user never exceeds the available limit. Create a demo class to test for ‘n’ Creditcard objects.**

**Solution:**

class Card {

    protected String cardNo;

    protected String custName;

    protected String bankName;

    public Card(String cardNo, String custName, String bankName) {

        this.cardNo = cardNo;

        this.custName = custName;

        this.bankName = bankName;

    }

}

class CreditCard extends Card {

    private double limit;

    public CreditCard(String cardNo, String custName, String bankName, double limit) {

        super(cardNo, custName, bankName);

        this.limit = limit;

    }

    public void display() {

        System.out.println("Credit Card Details:");

        System.out.println("Card Number: " + cardNo);

        System.out.println("Customer Name: " + custName);

        System.out.println("Bank Name: " + bankName);

        System.out.println("Limit: " + limit);

    }

    public void use(double amount) {

        if (amount <= limit) {

            limit -= amount;

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

            System.out.println("Remaining Limit: " + limit);

        } else {

            System.out.println("Insufficient limit. Transaction cannot be completed.");

        }

    }

}

public class CreditCardDemo {

    public static void main(String[] args) {

        CreditCard card1 = new CreditCard("123456789", "John Doe", "ABC Bank", 5000.0);

        CreditCard card2 = new CreditCard("987654321", "Jane Smith", "XYZ Bank", 10000.0);

        card1.display();

        System.out.println();

        card2.display();

        System.out.println();

        card1.use(2000.0);

        System.out.println();

        card2.use(15000.0);

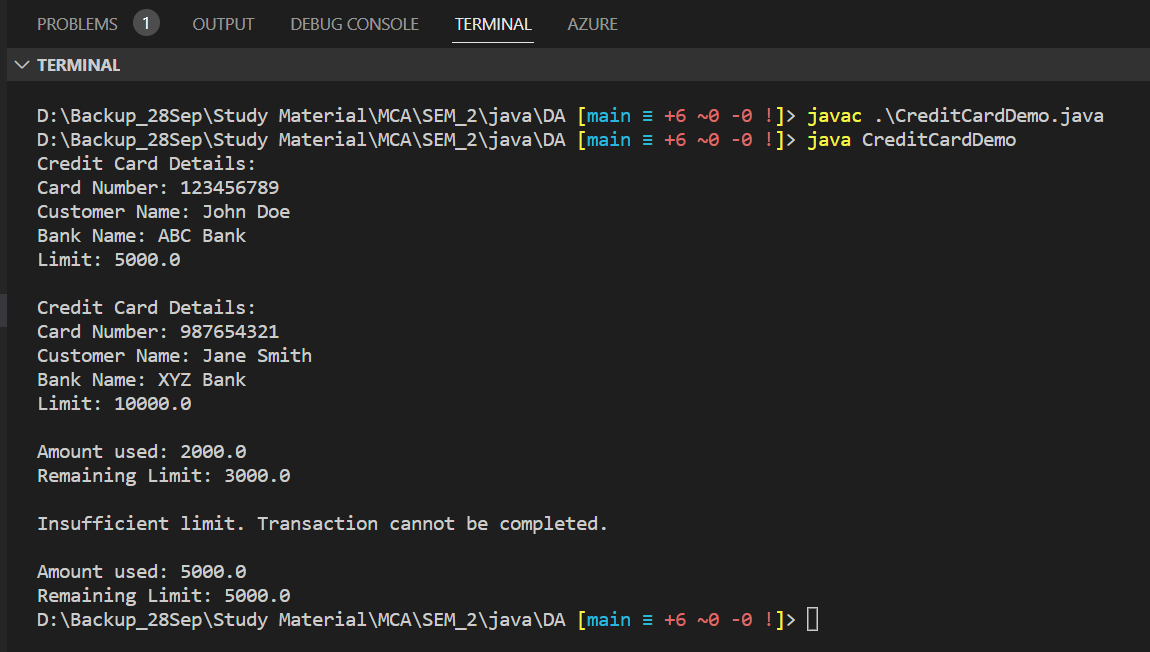
        System.out.println();

        card2.use(5000.0);

    }

}

**Output:**

****

**Q10) Write an inheritance hierarchy for classes Quadrilateral, Trapezoid, Parallelogram, Rectangle and Square. Use Quadrilateral as the superclass of the hierarchy. Create and use a Point class to represent the points in each shape. Make the hierarchy as deep (i.e., as many levels) as possible. Specify the instance variables and methods for each class. The private instance variables of the Quadrilateral should be the x-y coordinate pairs for the four endpoints of the Quadrilateral. Write a program that instantiates objects of your classes and outputs each object’s area (except Quadrilateral).**

**Solution:**

class Point {

    private double x;

    private double y;

    public Point(double x, double y) {

        this.x = x;

        this.y = y;

    }

    public double getX() {

        return x;

    }

    public double getY() {

        return y;

    }

}

class Quadrilateral {

    public Point point1;

    public Point point2;

    public Point point3;

    public Point point4;

    public Quadrilateral(Point point1, Point point2, Point point3, Point point4) {

        this.point1 = point1;

        this.point2 = point2;

        this.point3 = point3;

        this.point4 = point4;

    }

    // Calculate the area of a Quadrilateral

    public double calculateArea() {

        // The area of a general quadrilateral cannot be determined without specific shape information

        // Return 0.0 or throw an exception to indicate that the calculation is not applicable

        return 0.0;

    }

}

class Trapezoid extends Quadrilateral {

    public Trapezoid(Point point1, Point point2, Point point3, Point point4) {

        super(point1, point2, point3, point4);

    }

    // Implement the area calculation method for Trapezoid

    public double calculateArea() {

        // Area calculation logic for Trapezoid

        double base1 = Math.abs(point1.getX() - point2.getX());

        double base2 = Math.abs(point3.getX() - point4.getX());

        double height = Math.abs(point2.getY() - point4.getY());

        return (base1 + base2) \* height / 2;

    }

}

class Parallelogram extends Quadrilateral {

    public Parallelogram(Point point1, Point point2, Point point3, Point point4) {

        super(point1, point2, point3, point4);

    }

    // Implement the area calculation method for Parallelogram

    public double calculateArea() {

        // Area calculation logic for Parallelogram

        double base = Math.abs(point1.getX() - point2.getX());

        double height = Math.abs(point2.getY() - point4.getY());

        return base \* height;

    }

}

class Rectangle extends Parallelogram {

    public Rectangle(Point point1, Point point2, Point point3, Point point4) {

        super(point1, point2, point3, point4);

    }

    // Implement the area calculation method for Rectangle

    public double calculateArea() {

        // Area calculation logic for Rectangle

        double width = Math.abs(point1.getX() - point2.getX());

        double height = Math.abs(point2.getY() - point4.getY());

        return width \* height;

    }

}

class Square extends Rectangle {

    public Square(Point point1, Point point2, Point point3, Point point4) {

        super(point1, point2, point3, point4);

    }

    // Implement the area calculation method for Square

    public double calculateArea() {

        // Area calculation logic for Square

        double side = Math.abs(point1.getX() - point2.getX());

        return side \* side;

    }

}

public class ShapesDemo {

    public static void main(String[] args) {

        // Create objects and calculate areas

        Point point1 = new Point(24, 33);

        Point point2 = new Point(12, 14);

        Point point3 = new Point(15, 20);

        Point point4 = new Point(10, 18);

        Quadrilateral quadrilateral = new Quadrilateral(point1, point2, point3, point4);

        double quadrilateralArea = quadrilateral.calculateArea();

        System.out.println("Quadrilateral Area: " + quadrilateralArea);

        Trapezoid trapezoid = new Trapezoid(point1, point2, point3, point4);

        double trapezoidArea = trapezoid.calculateArea();

        System.out.println("Trapezoid Area: " + trapezoidArea);

        Parallelogram parallelogram = new Parallelogram(point1, point2, point3, point4);

        double parallelogramArea = parallelogram.calculateArea();

        System.out.println("Parallelogram Area: " + parallelogramArea);

        Rectangle rectangle = new Rectangle(point1, point2, point3, point4);

        double rectangleArea = rectangle.calculateArea();

        System.out.println("Rectangle Area: " + rectangleArea);

        Square square = new Square(point1, point2, point3, point4);

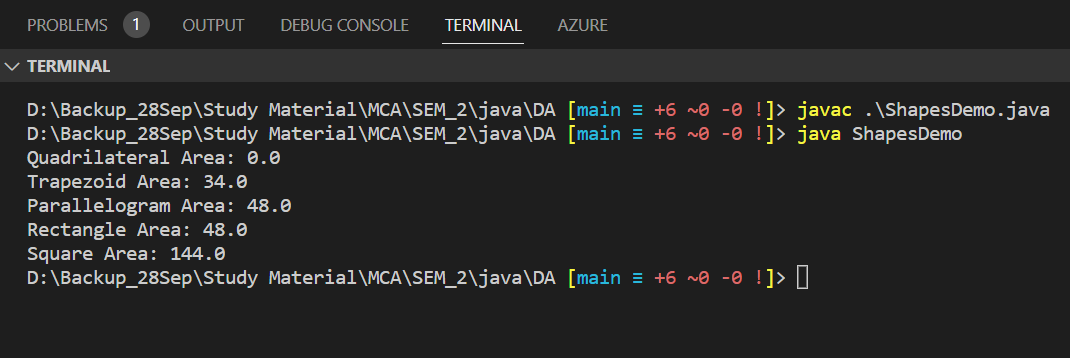
        double squareArea = square.calculateArea();

        System.out.println("Square Area: " + squareArea);

    }

}

**Output:**

****

**Q11) The interface GCD contains an abstract method computeGCD(int num1, int num2). Class APPROACH1 implements the interface by following Euclid’s algorithm and class APPROACH2 implements the interface by listing all the factors (need not be prime factors) of the two numbers and choosing the highest common factor. Write a Java program to do the above-said operations.**

**Solution:**

interface GCD {

    int computeGCD(int num1, int num2);

}

class APPROACH1 implements GCD {

    @Override

    public int computeGCD(int num1, int num2) {

        // Implementing Euclid's algorithm to compute GCD

        while (num2 != 0) {

            int temp = num2;

            num2 = num1 % num2;

            num1 = temp;

        }

        return num1;

    }

}

class APPROACH2 implements GCD {

    @Override

    public int computeGCD(int num1, int num2) {

        int gcd = 1;

        for (int i = 1; i <= num1 && i <= num2; i++) {

            if (num1 % i == 0 && num2 % i == 0) {

                gcd = i;

            }

        }

        return gcd;

    }

}

public class GCDExample {

    public static void main(String[] args) {

        int number1 = 36;

        int number2 = 48;

        // Using APPROACH1 (Euclid's algorithm) to compute GCD

        GCD approach1 = new APPROACH1();

        int gcd1 = approach1.computeGCD(number1, number2);

        System.out.println("GCD using APPROACH1: " + gcd1);

        // Using APPROACH2 (listing factors) to compute GCD

        GCD approach2 = new APPROACH2();

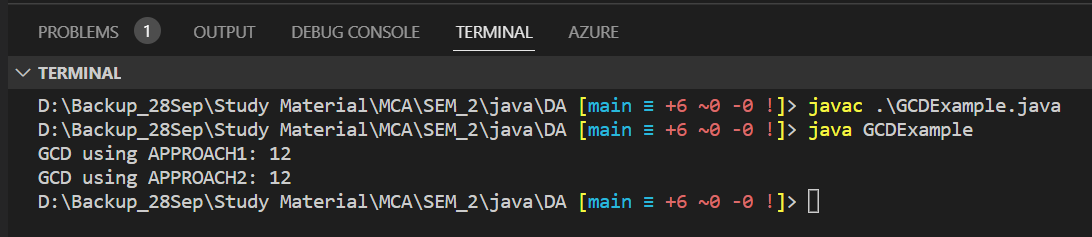
        int gcd2 = approach2.computeGCD(number1, number2);

        System.out.println("GCD using APPROACH2: " + gcd2);

    }

}

**Output:**

****

**Q12) Write an abstract class Special with an abstract method double process (doubleP,double R). Create a subclass Discount and implement the process() method. Return the process() method with the following formula: total=P+P\*R/100. Return the total.**

**Solution:**

abstract class Special {

    public abstract double process(double P, double R);

}

class Discount extends Special {

    @Override

    public double process(double P, double R) {

        double total = P + (P \* R / 100);

        return total;

    }

}

public class SpecialExample {

    public static void main(String[] args) {

        double price = 100.0;

        double rate = 10.0;

        Special special = new Discount();

        double result = special.process(price, rate);

        System.out.println("Total with discount: " + result);

    }

}

**Output:**

**A screen shot of a computer

Description automatically generated with low confidence**

**Q13) Create a package called pack1. Add two classes Sum and Difference (calculate the sum and difference of two numbers) to it. Create a subpackage called subpack1. Add two classes Product and Quotient (calculate the product and quotient of two numbers). Write a program to read values from the user and perform the arithmetic operations using the package classes.**

**Solution:**

**//Summation.java (/pack1)**

package pack1;

public class Summation {

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

        return a + b;

    }

}

**//Difference.java (/pack1)**

package pack1;

public class Difference {

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

        return a - b;

    }

}

**//Product.java (pack1/subpack1)**

package pack1.subpack1;

public class Product {

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

        return a \* b;

    }

}

**//Quotient.java (pack1/subpack1)**

package pack1.subpack1;

public class Quotient {

    public static double calculate(double a, double b) {

        return a / b;

    }

}

**//PackageTesting.java (Main method)**

import pack1.Summation;

import pack1.Difference;

import pack1.subpack1.Product;

import pack1.subpack1.Quotient;

import java.util.Scanner;

public class PackageTesting {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter two numbers: ");

        int num1 = scanner.nextInt();

        int num2 = scanner.nextInt();

        int sum = Summation.calculate(num1, num2);

        int difference = Difference.calculate(num1, num2);

        int product = Product.calculate(num1, num2);

        double quotient = Quotient.calculate(num1, num2);

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

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

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

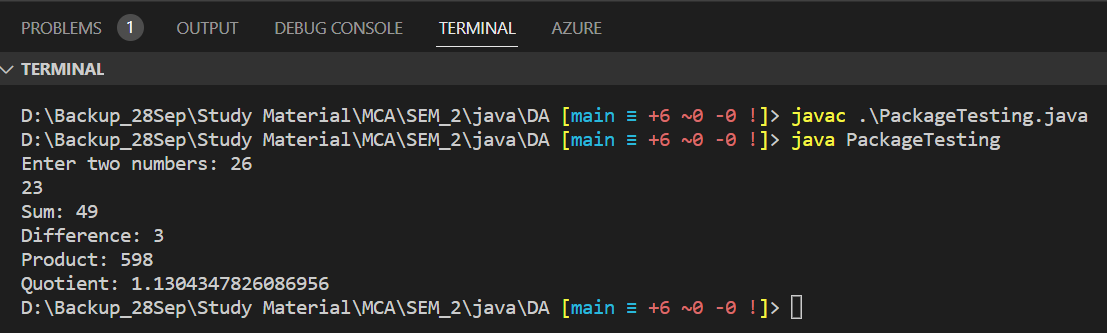
        System.out.println("Quotient: " + quotient);

        scanner.close();

    }

}

**Output:**

****

**Q14) Create an interface with methods add () and sub () in a package called ‘pack1’. Create another package, ‘pack2’ with an interface with methods multiply () and divide (). By implementing both interfaces, write a main class to perform arithmetic operations on integer numbers.**

**Solution:**

**//Arithmetic.java (/pack1)**

package pack1;

public interface Arithmetic {

    int add(int a, int b);

    int sub(int a, int b);

}

**//Calculator.java (/pack2)**

package pack2;

public interface Calculator {

    int multiply(int a, int b);

    int divide(int a, int b);

}

**//PackageTesting2.java**

import pack1.Arithmetic;

import pack2.Calculator;

public class PackageTesting2 implements Arithmetic, Calculator {

    public int add(int a, int b) {

        return a + b;

    }

    public int sub(int a, int b) {

        return a - b;

    }

    public int multiply(int a, int b) {

        return a \* b;

    }

    public int divide(int a, int b) {

        if (b != 0) {

            return a / b;

        }

        return 0;

    }

    public static void main(String[] args) {

        PackageTesting2 obj = new PackageTesting2();

        int num1 = 10;

        int num2 = 5;

        System.out.println("Addition: " + obj.add(num1, num2));

        System.out.println("Subtraction: " + obj.sub(num1, num2));

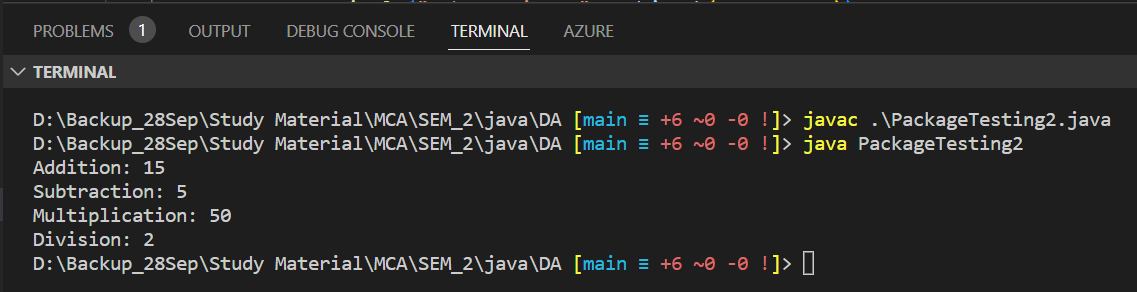
        System.out.println("Multiplication: " + obj.multiply(num1, num2));

        System.out.println("Division: " + obj.divide(num1, num2));

    }

}

**Output:**

****

**Q15) You have been asked to create a hospital management system that can keep track of patients and their medical records. The system should be able to handle different types of patients, such as inpatients and outpatients, and different types of medical records, such as laboratory reports and radiology images.**

**Task:**

**a) Create an abstract class called Patient with the following properties:**

**o String patientName**

**o int patientAge**

**o String patientGender**

**o String patientID**

**o String patientAddress**

**b) Create two subclasses of Patient: Inpatient and Outpatient. Each subclass should have its own unique property:**

**o Inpatient: int bedNumber**

**o Outpatient: String appointmentDate**

**c) Create an interface called MedicalRecord with the following methods:**

**o void addRecord()**

**o void deleteRecord()**

**o void viewRecord()**

**d) Have the Patient class implement the MedicalRecord interface.**

**e) Create two subclasses of MedicalRecord: LaboratoryReport and RadiologyImage. Each subclass should have its own unique property:**

**o LaboratoryReport: String testName**

**o RadiologyImage: String imageType**

**f) Implement the addRecord(), deleteRecord(), and viewRecord() methods for each of the subclasses. When a medical record is added, it should be stored in a list of medical records for the patient. When a medical record is deleted, it should be removed from the list of medical records. When a medical record is viewed, its properties should be displayed.**

**g) Create a Hospital class that will keep track of all the patients in the hospital. It should have the following methods:**

**o admitPatient(Patient patient): Add a patient to the hospital**

**o dischargePatient(Patient patient): Discharge a patient from the hospital**

**o displayPatients(): Display a list of all patients in the hospital**

**o displayMedicalRecords(Patient patient): Display a list of all medical records for a given patient.**

**h) Create a Main class with a main method that will test the functionality of the hospital management system. The main method should:**

**o Create several inpatients and outpatients and admit them to the hospital**

**o Display the list of patients in the hospital**

**o Add some medical records for the patients, both laboratory reports and radiology images**

**o Display the list of medical records for a given patient**

**o Delete some medical records for the patients**

**o Discharge some of the patients from the hospital**

**o Display the list of patients in the hospital again to show the updated list of patients**

**Solution:**

import java.util.ArrayList;

import java.util.List;

abstract class Patient {

    private String patientName;

    private int patientAge;

    private String patientGender;

    private String patientID;

    private String patientAddress;

    private List<MedicalRecord> medicalRecords;

    public Patient(String patientName, int patientAge, String patientGender, String patientID, String patientAddress) {

        this.patientName = patientName;

        this.patientAge = patientAge;

        this.patientGender = patientGender;

        this.patientID = patientID;

        this.patientAddress = patientAddress;

        this.medicalRecords = new ArrayList<>();

    }

    public String getPatientName() {

        return patientName;

    }

    public int getPatientAge() {

        return patientAge;

    }

    public String getPatientGender() {

        return patientGender;

    }

    public String getPatientID() {

        return patientID;

    }

    public String getPatientAddress() {

        return patientAddress;

    }

    public void addRecord(MedicalRecord record) {

        medicalRecords.add(record);

    }

    public void deleteRecord(MedicalRecord record) {

        medicalRecords.remove(record);

    }

    public void viewRecord() {

        for (MedicalRecord record : medicalRecords) {

            record.displayRecord();

        }

    }

}

class Inpatient extends Patient {

    private int bedNumber;

    public Inpatient(String patientName, int patientAge, String patientGender, String patientID, String patientAddress, int bedNumber) {

        super(patientName, patientAge, patientGender, patientID, patientAddress);

        this.bedNumber = bedNumber;

    }

    public int getBedNumber() {

        return bedNumber;

    }

}

class Outpatient extends Patient {

    private String appointmentDate;

    public Outpatient(String patientName, int patientAge, String patientGender, String patientID, String patientAddress, String appointmentDate) {

        super(patientName, patientAge, patientGender, patientID, patientAddress);

        this.appointmentDate = appointmentDate;

    }

    public String getAppointmentDate() {

        return appointmentDate;

    }

}

interface MedicalRecord {

    void addRecord();

    void deleteRecord();

    void displayRecord();

}

class LaboratoryReport implements MedicalRecord {

    private String testName;

    public LaboratoryReport(String testName) {

        this.testName = testName;

    }

    public void addRecord() {

        // Logic to add laboratory report

        System.out.println("Laboratory report added.");

    }

    public void deleteRecord() {

        // Logic to delete laboratory report

        System.out.println("Laboratory report deleted.");

    }

    public void displayRecord() {

        System.out.println("Laboratory Report: " + testName);

    }

}

class RadiologyImage implements MedicalRecord {

    private String imageType;

    public RadiologyImage(String imageType) {

        this.imageType = imageType;

    }

    public void addRecord() {

        // Logic to add radiology image

        System.out.println("Radiology image added.");

    }

    public void deleteRecord() {

        // Logic to delete radiology image

        System.out.println("Radiology image deleted.");

    }

    public void displayRecord() {

        System.out.println("Radiology Image: " + imageType);

    }

}

class Hospital {

    private List<Patient> patients;

    public Hospital() {

        patients = new ArrayList<>();

    }

    public void admitPatient(Patient patient) {

        patients.add(patient);

    }

    public void dischargePatient(Patient patient) {

        patients.remove(patient);

    }

    public void displayPatients() {

        System.out.println("List of Patients in the Hospital:");

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

        for (Patient patient : patients) {

            System.out.println("Patient ID: " + patient.getPatientID());

            System.out.println("Patient Name: " + patient.getPatientName());

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

        }

    }

    public void displayMedicalRecords(Patient patient) {

        System.out.println("Medical Records for patient: " + patient.getPatientName());

        patient.viewRecord();

    }

}

public class HospitalManage {

    public static void main(String[] args) {

        Hospital hospital = new Hospital();

        Inpatient inpatient1 = new Inpatient("John", 40, "Male", "P001", "123 Street", 101);

        Inpatient inpatient2 = new Inpatient("Alice", 35, "Female", "P002", "456 Street", 102);

        Outpatient outpatient1 = new Outpatient("David", 50, "Male", "P003", "789 Street", "2023-06-01");

        Outpatient outpatient2 = new Outpatient("Sarah", 28, "Female", "P004", "321 Street", "2023-06-02");

        hospital.admitPatient(inpatient1);

        hospital.admitPatient(inpatient2);

        hospital.admitPatient(outpatient1);

        hospital.admitPatient(outpatient2);

        hospital.displayPatients();

        LaboratoryReport labReport1 = new LaboratoryReport("Blood Test");

        LaboratoryReport labReport2 = new LaboratoryReport("Urinalysis");

        RadiologyImage radiologyImage1 = new RadiologyImage("X-Ray");

        RadiologyImage radiologyImage2 = new RadiologyImage("MRI");

        // Add medical records for patients

        inpatient1.addRecord(labReport1);

        inpatient1.addRecord(radiologyImage1);

        outpatient1.addRecord(labReport2);

        outpatient2.addRecord(radiologyImage2);

        // Display medical records for a patient

        hospital.displayMedicalRecords(inpatient1);

        // Delete medical records for patients

        inpatient1.deleteRecord(labReport1);

        outpatient2.deleteRecord(radiologyImage2);

        hospital.dischargePatient(inpatient1);

        hospital.dischargePatient(outpatient2);

        hospital.displayPatients();

    }

}

**Q16) Create a suitable GUI using AWT/Swing and demonstrate the CRUD operations using JDBC.**

**Solution:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.\*;

public class CRUDApplication extends JFrame {

    private JLabel nameLabel, ageLabel, genderLabel;

    private JTextField nameField, ageField, genderField;

    private JButton addButton, updateButton, deleteButton, viewButton;

    private Connection connection;

    private Statement statement;

    public CRUDApplication() {

        initializeDatabase();

        createUI();

        bindActions();

    }

    private void initializeDatabase() {

        try {

            // Establish the connection with the database

            connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb", "root", "password");

            // Create a statement object

            statement = connection.createStatement();

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

    private void createUI() {

        // Set frame properties

        setTitle("CRUD Application");

        setSize(400, 200);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        // Create labels

        nameLabel = new JLabel("Name:");

        ageLabel = new JLabel("Age:");

        genderLabel = new JLabel("Gender:");

        // Create text fields

        nameField = new JTextField();

        ageField = new JTextField();

        genderField = new JTextField();

        // Create buttons

        addButton = new JButton("Add");

        updateButton = new JButton("Update");

        deleteButton = new JButton("Delete");

        viewButton = new JButton("View");

        // Create panel and set layout

        JPanel panel = new JPanel();

        panel.setLayout(new GridLayout(4, 2));

        // Add components to the panel

        panel.add(nameLabel);

        panel.add(nameField);

        panel.add(ageLabel);

        panel.add(ageField);

        panel.add(genderLabel);

        panel.add(genderField);

        panel.add(addButton);

        panel.add(updateButton);

        panel.add(deleteButton);

        panel.add(viewButton);

        // Add panel to the frame

        add(panel);

        // Set frame visibility

        setVisible(true);

    }

    private void bindActions() {

        addButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                String name = nameField.getText();

                int age = Integer.parseInt(ageField.getText());

                String gender = genderField.getText();

                // Insert record into the database

                try {

                    statement.executeUpdate("INSERT INTO patients (name, age, gender) VALUES ('" + name + "', " + age + ", '" + gender + "')");

                    JOptionPane.showMessageDialog(null, "Record added successfully!");

                } catch (SQLException ex) {

                    ex.printStackTrace();

                }

            }

        });

        updateButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                String name = nameField.getText();

                int age = Integer.parseInt(ageField.getText());

                String gender = genderField.getText();

                // Update record in the database

                try {

                    statement.executeUpdate("UPDATE patients SET age=" + age + ", gender='" + gender + "' WHERE name='" + name + "'");

                    JOptionPane.showMessageDialog(null, "Record updated successfully!");

                } catch (SQLException ex) {

                    ex.printStackTrace();

                }

            }

        });

        deleteButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                String name = nameField.getText();

                // Delete record from the database

                try {

                    statement.executeUpdate("DELETE FROM patients WHERE name='" + name + "'");

                    JOptionPane.showMessageDialog(null, "Record deleted successfully!");

                } catch (SQLException ex) {

                    ex.printStackTrace();

                }

            }

        });

        viewButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                // Fetch records from the database

                try {

                    ResultSet resultSet = statement.executeQuery("SELECT \* FROM patients");

                    StringBuilder records = new StringBuilder();

                    while (resultSet.next()) {

                        String name = resultSet.getString("name");

                        int age = resultSet.getInt("age");

                        String gender = resultSet.getString("gender");

                        records.append("Name: ").append(name).append(", Age: ").append(age).append(", Gender: ").append(gender).append("\n");

                    }

                    JOptionPane.showMessageDialog(null, records.toString());

                } catch (SQLException ex) {

                    ex.printStackTrace();

                }

            }

        });

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                new CRUDApplication();

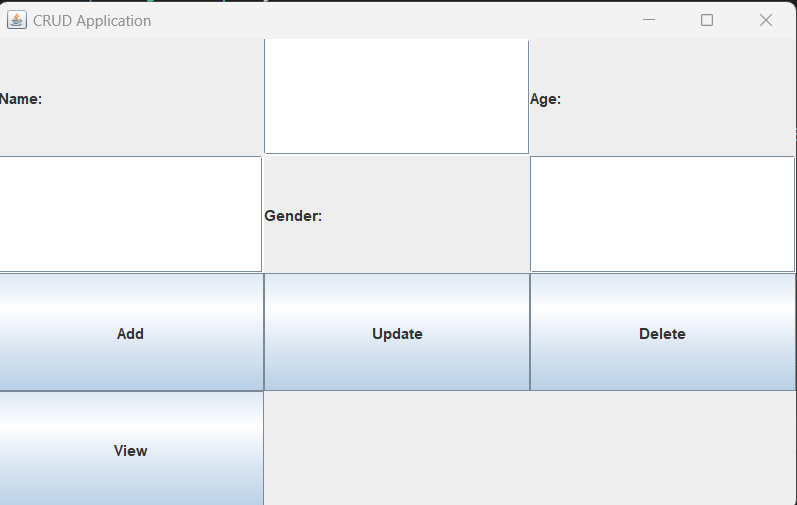
            }

        });

    }

}

**Output:**

****

**Q17) Demonstrate how a Form submission can be done using Java servlets.**

**Solution:**

**//index.html**

<!DOCTYPE html>

<html>

<head>

    <title>Form Submission</title>

</head>

<body>

    <h2>Submit Form</h2>

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

        <label for="name">Name:</label>

        <input type="text" id="name" name="name"><br><br>

        <label for="email">Email:</label>

        <input type="email" id="email" name="email"><br><br>

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

    </form>

</body>

</html>

**//web.xml**

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

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

                             http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

         version="3.1">

    <servlet>

        <servlet-name>SubmitServlet</servlet-name>

        <servlet-class>SubmitServlet</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>SubmitServlet</servlet-name>

        <url-pattern>/submit</url-pattern>

    </servlet-mapping>

    <welcome-file-list>

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

    </welcome-file-list>

</web-app>

**//SubmitServlet.java**

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class SubmitServlet extends HttpServlet {

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

        // Retrieve form data

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

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

        // Process the form data (e.g., store in a database, perform calculations, etc.)

        // Set response content type

        response.setContentType("text/html");

        // Display a response message

        response.getWriter().println("<h2>Form Submitted Successfully!</h2>");

        response.getWriter().println("<p>Name: " + name + "</p>");

        response.getWriter().println("<p>Email: " + email + "</p>");

    }

}