

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_2028\_OOPS using Java\_Week 7\_MCQ

Attempt : 1  
Total Mark : 15  
Marks Obtained : 13

#### Section 1 : MCQ

1. Which of the following is the correct way to declare an interface in Java?

**Answer**

```
interface Vehicle { void start();}
```

**Status : Correct**

**Marks : 1/1**

2. What is the output of the following code?

```
interface A {  
    default void show() {  
        System.out.println("A's Default Method");  
    }  
}
```

```
interface B {  
    default void show() {  
        System.out.println("B's Default Method");  
    }  
}
```

```
class C implements A, B {  
    public void show() {  
        A.super.show();  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        C obj = new C();  
        obj.show();  
    }  
}
```

**Answer**

A's Default Method

**Status : Correct**

**Marks : 1/1**

3. What is the primary purpose of static methods in Java interfaces?

**Answer**

They allow an interface to provide helper methods without requiring an implementing class.

**Status : Correct**

**Marks : 1/1**

4. What happens when an implementing class does not override a default method from an interface?

**Answer**

The default method's implementation from the interface will be used.

**Status :** Correct

**Marks :** 1/1

5. What is the output of the following code?

```
interface A {  
    static void display() {  
        System.out.println("Static method in A");  
    }  
}
```

```
class B implements A {  
    static void display() {  
        System.out.println("Static method in B");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        B.display();  
    }  
}
```

**Answer**

Static method in B

**Status :** Correct

**Marks :** 1/1

6. How do you call a static method from an interface MyInterface?

**Answer**

MyInterface.staticMethod();

**Status :** Correct

**Marks :** 1/1

7. What is the output of the following code?

```
interface X {  
    default void show() {
```

```
        System.out.println("X's Default Method");
    }
}
```

```
interface Y {
    default void show() {
        System.out.println("Y's Default Method");
    }
}
```

```
class Z implements X, Y {
    public void show() {
        System.out.println("Z's Method");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Z obj = new Z();
        obj.show();
    }
}
```

**Answer**

Z's Method

**Status :** Correct

**Marks :** 1/1

8. If a class implements two interfaces that have the same default method, what must the class do?

**Answer**

The class must override the method to resolve ambiguity.

**Status :** Correct

**Marks :** 1/1

9. Which of the following statements about Java interfaces is true?

**Answer**

A class can implement multiple classes and interfaces together.

**Status :** Wrong

**Marks :** 0/1

10. Can a Java interface contain both default and static methods?

**Answer**

Yes, an interface can have both default and static methods.

**Status :** Correct

**Marks :** 1/1

11. What is the output of the following code?

```
interface A {  
    default void show() {  
        System.out.println("A's Default Method");  
    }  
}
```

```
class B {  
    public void show() {  
        System.out.println("B's Method");  
    }  
}
```

```
class C extends B implements A {  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        C obj = new C();  
        obj.show();  
    }  
}
```

**Answer**

B's Method

**Status :** Correct

**Marks :** 1/1

12. How can a class explicitly call a default method from an interface if there is a naming conflict?

**Answer**

Using InterfaceName.super.methodName();

**Status :** Correct

**Marks :** 1/1

13. What is the output of the following code?

```
interface MathOperations {  
    static int square(int x) {  
        return x * x;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        System.out.println(MathOperations.square(5));  
    }  
}
```

**Answer**

25

**Status :** Correct

**Marks :** 1/1

14. Which of the following statements is true regarding default methods in Java interfaces?

**Answer**

A default method must be implemented in all classes that implement the interface.

**Status :** Wrong

**Marks :** 0/1

15. Consider a class implementing an interface and extending a class, both having a method with the same name. Which method gets called?

**Answer**

The method from the superclass

**Status :** Correct

**Marks :** 1/1

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 7\_Q1

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement:

Rajiv is analyzing the energy consumption in his household and wants to calculate the total cost based on the daily energy usage. He is given the rate per unit of electricity and the energy consumed for multiple days. To structure this calculation efficiently, he decides to use an interface-based approach.

Implement an interface CostCalculator with the necessary methods to retrieve energy details and compute the cost. The calculations should be handled in the EnergyConsumptionTracker class, while the EnergyConsumptionApp class should only handle input and output.

##### Formula

Energy Cost for one day = Energy Consumed per day \* Rate Per Unit

### ***Input Format***

The first line of input consists of the rate per unit as an 'R' (a double value).

The second line of input consists of the number of days 'N' (an integer).

The third line of input consists of the daily energy consumption values for each day 'D' (double values), separated by space.

### ***Output Format***

The first line of the output prints: "Day-wise Energy Cost:"

The next N lines of the output print the day-wise energy costs(double type) and the total energy cost (double type) in Indian Rupees in the following format: "Day [day\_number]: Rs. [energy\_cost]"

The last line of the output prints: "Total Energy Cost: Rs. [total\_cost]"

Note: energy\_cost and total\_cost are rounded off to two decimal points

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 0.01

3

10.0 20.0 30.0

Output: Day-wise Energy Cost:

Day 1: Rs. 0.10

Day 2: Rs. 0.20

Day 3: Rs. 0.30

Total Energy Cost: Rs. 0.60

### ***Answer***

```
import java.util.Scanner;
```

```
interface CostCalculator {
```

```

    void getEnergyDetails(Scanner scanner);
    void calculateAndDisplayCost();
}

class EnergyConsumptionTracker implements CostCalculator {
    private double ratePerUnit;
    private int numDays;
    private double[] dailyConsumption;

    public EnergyConsumptionTracker(double ratePerUnit, int numDays) {
        this.ratePerUnit = ratePerUnit;
        this.numDays = numDays;
        this.dailyConsumption = new double[numDays];
    }

    @Override
    public void getEnergyDetails(Scanner scanner) {
        for (int i = 0; i < numDays; i++) {
            dailyConsumption[i] = scanner.nextDouble();
        }
    }

    @Override
    public void calculateAndDisplayCost() {
        System.out.print("Day-wise Energy Cost: ");
        double totalCost = 0.0;

        for (int i = 0; i < numDays; i++) {
            double dayCost = dailyConsumption[i] * ratePerUnit;
            totalCost += dayCost;
            System.out.printf("Day %d: Rs. %.2f ", (i + 1), dayCost);
        }

        System.out.printf("Total Energy Cost: Rs. %.2f ", totalCost);
    }
}

class EnergyConsumptionApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        double ratePerUnit = scanner.nextDouble();
        int numDays = scanner.nextInt();
    }
}

```

```
CostCalculator tracker = new EnergyConsumptionTracker(ratePerUnit,  
numDays);
```

```
    tracker.getEnergyDetails(scanner);  
    tracker.calculateAndDisplayCost();
```

```
    scanner.close();
```

```
    }  
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 7\_Q2

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

Jaheer is working on a health monitoring system to help individuals calculate their Body Mass Index (BMI). He has implemented a basic BMI calculator and an interface called HealthCalculator. It should have a method called calculateBMI.

You are tasked with creating a program that takes weight and height as input, calculates the BMI using the BMICalculator class, and displays the result. If the height or weight is less than or equal to zero, then return -1.

Formula:  $BMI = \text{weight} / (\text{height} * \text{height})$

##### ***Input Format***

The first line of input consists of a double value W, the person's weight in kilograms.

The second line consists of a double value H, the height of the person in meters.

### **Output Format**

The output displays "BMI: " followed by a double value, representing the calculated BMI, rounded off to two decimal places.

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 70.0

1.75

Output: BMI: 22.86

### **Answer**

```
import java.util.Scanner;

interface HealthCalculator {
    double calculateBMI(double weight, double height);
}

class BMICalculator implements HealthCalculator {
    @Override
    public double calculateBMI(double weight, double height) {
        if (weight <= 0 || height <= 0) {
            return -1;
        }
        return weight / (height * height);
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        double weight = scanner.nextDouble();
        double height = scanner.nextDouble();

        BMICalculator bmiCalculator = new BMICalculator();
```

```
double bmi = bmiCalculator.calculateBMI(weight, height);  
System.out.printf("BMI: %.2f\n", bmi);  
  
    scanner.close();  
}  
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 7\_Q3

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

A financial analyst, Alex, needs a program to calculate simple interest for various financial transactions. He requires a straightforward tool that takes in the principal amount, interest rate, and time in years and computes the interest.

The formula to be used is:  $\text{Interest} = \text{Principal} \times \text{Rate} \times \text{Time} / 100$

Implement this functionality using the InterestCalculator interface and the SimpleInterestCalculator class.

##### ***Input Format***

The first line of input consists of the principal amount P as a double value.

The second line of input consists of the annual interest rate  $r$  as a double value.

The third line of input consists of the number of years  $t$  as a positive integer, which is an integer value.

### ***Output Format***

The output displays the calculated simple interest in the following format: "Simple Interest: [interest\_value]", Here, [interest\_value] should be replaced with the actual interest value calculated by the program.

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 1000.00

5.00

2

Output: Simple Interest: 100.0

### ***Answer***

```
import java.util.Scanner;

interface InterestCalculator {
    double simpleInterest(double principal, double rate, int time);
}

class SimpleInterestCalculator implements InterestCalculator {
    public double simpleInterest(double principal, double rate, int time) {
        return (principal * rate * time) / 100;
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        double principal = scanner.nextDouble();
        double rate = scanner.nextDouble();
        int time = scanner.nextInt();
```

```
InterestCalculator calculator = new SimpleInterestCalculator();  
double interest = calculator.simpleInterest(principal, rate, time);  
  
System.out.println("Simple Interest: " + interest);  
  
}  
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 7\_Q4

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

Maria, a software developer, is working on an inventory management system project using Java that utilizes an inventory interface to manage a store's products.

The interface should define two methods: `addProduct`, which adds a product by accepting its name, price, and quantity, and `calculateTotalValue`, which computes the total value of all products in the inventory. Implement the interface in a class called `SimpleInventory`, which internally manages a list of `Product` objects.

Each `Product` object should encapsulate the product's name, price, and quantity and include a method to calculate its value as  $\text{price} \times \text{quantity}$ . The system should allow users to dynamically add products to the inventory and calculate the total value of all products stored.

Help Maria achieve the task.

### ***Input Format***

The first line of input consists of an integer to choose one of the following options:

- 1 - to add a product to the inventory.
- 2 - to calculate and view the total inventory value.
- 3 - to exit the program.

For Choice 1 (Add Product):

The next input line is the string representing the product name as a string (single or multi-word, without quotes).

The next line is a double value representing the price as a decimal value

The next line is an integer value representing the quantity as an integer

For Choices 2 and 3, no additional input is required

### ***Output Format***

The output displays the results of the commands as follows:

- For the addProduct command, the program should display "Product added to inventory."
- For choice 2, the program should display "Total inventory value [totalvalue].  
"The total value should be displayed with one decimal place. If there is no product in the inventory, print the total as 0.0.
- For choice 3, the program should exit

If the choice is not 1, 2, or 3, then print "Invalid choice. Please select a valid option (1/2/3).".

Refer to the sample output for the formatting specifications.

### Sample Test Case

Input: 1

Laptop

800.0

3

2

5

3

Output: Product added to inventory.

Total inventory value: \$2400.0

Invalid choice. Please select a valid option (1/2/3).

### Answer

```
import java.util.Scanner;
```

```
interface Inventory {  
    void addProduct(String name, double price, int quantity);  
    double calculateTotalValue();  
}
```

```
class Product {  
    private String name;  
    private double price;  
    private int quantity;
```

```
    public Product(String name, double price, int quantity) {  
        this.name = name;  
        this.price = price;  
        this.quantity = quantity;  
    }
```

```
    public double getValue() {  
        return price * quantity;  
    }  
}
```

```
class SimpleInventory implements Inventory {  
    private Product[] products;  
    private int count;  
    public SimpleInventory(int capacity) {  
        products = new Product[capacity]; count = 0;  
    }
```

```

public void addProduct(String name, double price, int quantity) {
    if (count < products.length) {
        products[count++] = new Product(name, price, quantity);
        System.out.println("Product added to inventory.");
    } else {
        System.out.println("Inventory is full. Cannot add more products.");
    }
}

public double calculateTotalValue() {
    double total = 0.0;
    for (int i = 0; i < count; i++) {
        total += products[i].getValue();
    }
    return total;
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Inventory inventory = new SimpleInventory(10);
        while (true) {
            int choice = scanner.nextInt();
            if (choice == 1) {
                scanner.nextLine();
                String productName = scanner.nextLine();
                double price = scanner.nextDouble();
                int quantity = scanner.nextInt();
                inventory.addProduct(productName, price, quantity);
            } else if (choice == 2) {
                double totalValue = inventory.calculateTotalValue();
                System.out.println("Total inventory value: $" + totalValue);
            } else if (choice == 3) {
                break;
            } else {
                System.out.println("Invalid choice. Please select a valid option (1/2/3).");
            }
        }
        scanner.close();
    }
}

```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 7\_Q5

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

Raj is curious about how old he is in the current year.

He has asked you to create a simple program that calculates a person's age based on their birth year. You decide to implement this functionality using the AgeCalculator interface and the HumanAgeCalculator class.

Note: The current year is 2024. Calculate the current age by using the formula: current year - birth year.

##### ***Input Format***

The input consists of an integer representing the birth year.

##### ***Output Format***

The output displays "You are X years old." where X is an integer representing the calculated age based on the entered birth year.

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 1934

Output: You are 90 years old.

### **Answer**

```
import java.util.Scanner;

interface AgeCalculator {
    int calculateAge(int birthYear);
}

class HumanAgeCalculator implements AgeCalculator {
    @Override
    public int calculateAge(int birthYear) {
        int currentYear = 2024;
        return currentYear - birthYear;
    }
}

class AgeCalculatorApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        AgeCalculator ageCalculator = new HumanAgeCalculator();

        int birthYear = scanner.nextInt();
        int age = ageCalculator.calculateAge(birthYear);

        System.out.println("You are " + age + " years old.");
    }
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_2028\_OOPS using Java\_Week 7\_PAH

Attempt : 1  
Total Mark : 40  
Marks Obtained : 40

#### Section 1 : Coding

##### 1. Problem Statement:

Alice has been tasked with implementing a simple calculator interface and a corresponding class for performing basic addition and subtraction operations. The task is to create an interface called Calculator with two methods: add and subtract. The add method should take two numbers as input and return their sum, while the subtract method should take two numbers as input and return their difference.

Implement a class called SimpleCalculator that implements the Calculator interface. This class should provide the functionality for adding and subtracting numbers. Write a code that satisfies the above requirements.

##### ***Input Format***

The first line of input consists of a single integer, representing the choice

If the choice is 1 or 2, the next two lines consist of 2 double values, representing the numbers to do addition or subtraction.

### ***Output Format***

The output prints a float-value with one decimal value representing the sum of two number or difference of two number.

Refer to the sample output for format specification.

### ***Sample Test Case***

Input: 1

5.5

3.5

Output: Result: 9.0

### ***Answer***

```
import java.util.Scanner;
```

```
interface Calculator {  
    double add(double a, double b);  
    double subtract(double a, double b);  
}
```

```
class SimpleCalculator implements Calculator {  
    public double add(double a, double b) {  
        return a + b;  
    }  
}
```

```
    public double subtract(double a, double b) {  
        return a - b;  
    }  
}
```

```
class MathOperationsProgram {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);
```

```

SimpleCalculator calculator = new SimpleCalculator();

int choice = scanner.nextInt();

if (choice == 1) {
    double num1 = scanner.nextDouble();
    double num2 = scanner.nextDouble();
    double result = calculator.add(num1, num2);
    System.out.println("Result: " + result);
} else if (choice == 2) {
    double num1 = scanner.nextDouble();
    double num2 = scanner.nextDouble();
    double result = calculator.subtract(num1, num2);
    System.out.println("Result: " + result);
} else {
    System.out.println("Invalid choice. Please choose 1 for addition or 2 for subtraction.");
}

scanner.close();
}
}

```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Oviya is fascinated by automorphic numbers and wants to create a program to determine whether a given number is an automorphic number or not.

An automorphic number is a number whose square ends with the same digits as the number itself. For example,  $25 = (25)^2 = 625$

Oviya has defined two interfaces: NumberInput for taking user input and AutomorphicChecker for checking if a given number is automorphic. The class AutomorphicNumber implements both interfaces.

Help her complete the task.

### ***Input Format***

The input consists of a single integer n.

### ***Output Format***

If the input number is an automorphic number, print "n is an automorphic number". Otherwise, print "n is not an automorphic number".

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 25

Output: 25 is an automorphic number

### ***Answer***

```
import java.util.Scanner;

interface NumberInput {
    int getInput();
}

interface AutomorphicChecker {
    boolean checkAutomorphic(int n);
}

class AutomorphicNumber implements NumberInput, AutomorphicChecker {

    public int getInput() {
        Scanner sc = new Scanner(System.in);
        return sc.nextInt();
    }

    public boolean checkAutomorphic(int n) {
        int square = n * n;

        String numStr = String.valueOf(n);
        String squareStr = String.valueOf(square);

        return squareStr.endsWith(numStr);
    }
}
```

```

    }
}

public class Main {
    public static void main(String[] args) {
        AutomorphicNumber automorphicNumber = new AutomorphicNumber();
        int inputNumber = automorphicNumber.getInput();

        boolean isAutomorphic =
        automorphicNumber.checkAutomorphic(inputNumber);

        if (isAutomorphic) {
            System.out.println(inputNumber+" is an automorphic number");
        } else {
            System.out.println(inputNumber+" is not an automorphic number");
        }
    }
}

```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

Sophia is developing a matrix analysis tool for a data analytics company. The tool needs to analyze square matrices and extract insights from the matrix diagonals.

To organize the code properly, Sophia creates an interface named Matrix that declares a method for finding the smallest and largest elements along the principal and secondary diagonals of the matrix.

Sophia then creates a class named MatrixAnalyzer that implements the Matrix interface. This class provides the logic to process a given square matrix and print:

The smallest and largest elements in the principal diagonal (from top-left to bottom-right). The smallest and largest elements in the secondary diagonal (from top-right to bottom-left).

Your task is to implement the Matrix interface and the MatrixAnalyzer

class. The main driver program (in the class Main) will read the input matrix, create an instance of MatrixAnalyzer, and invoke its method to display the results.

### ***Input Format***

The first line contains an integer n, representing the size of the square matrix.

The next n lines each contain n integers separated by spaces, representing the elements of the matrix.

### ***Output Format***

The output prints the four lines:

"Smallest Element - 1: <smallest element in the principal diagonal>" (integer)

"Largest Element - 1: <largest element in the principal diagonal>" (integer)

"Smallest Element - 2: <smallest element in the secondary diagonal>" (integer)

"Largest Element - 2: <largest element in the secondary diagonal>" (integer)

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 5

7 8 9 0 1

2 3 4 5 6

5 4 2 0 8

23 5 6 8 9

12 5 6 7 32

Output: Smallest Element - 1: 2

Largest Element - 1: 32

Smallest Element - 2: 1

Largest Element - 2: 12

### ***Answer***

```
import java.util.Scanner;
```

```
interface Matrix {  
    void diagonalsMinMax(int[][] matrix);  
}
```

```
class MatrixAnalyzer implements Matrix {
```

```
    @Override
```

```
    public void diagonalsMinMax(int[][] matrix) {  
        int n = matrix.length;
```

```
        int smallest1 = matrix[0][0];  
        int largest1 = matrix[0][0];
```

```
        int smallest2 = matrix[0][n - 1];  
        int largest2 = matrix[0][n - 1];
```

```
        for (int i = 0; i < n; i++) {  
            int principal = matrix[i][i];  
            int secondary = matrix[i][n - 1 - i];
```

```
            if (principal < smallest1) smallest1 = principal;  
            if (principal > largest1) largest1 = principal;
```

```
            if (secondary < smallest2) smallest2 = secondary;  
            if (secondary > largest2) largest2 = secondary;
```

```
        }
```

```
        System.out.println("Smallest Element - 1: " + smallest1);  
        System.out.println("Largest Element - 1: " + largest1);  
        System.out.println("Smallest Element - 2: " + smallest2);  
        System.out.println("Largest Element - 2: " + largest2);
```

```
    }  
}
```

```
public class Main {
```

```
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);
```

```
        int n = sc.nextInt();
```

```
        int[][] matrix = new int[n][n];
```

```
        for (int i = 0; i < n; i++) {  
            for (int j = 0; j < n; j++) {  
                matrix[i][j] = sc.nextInt();
```

```
            }
```

```
}  
    MatrixAnalyzer analyzer = new MatrixAnalyzer();  
    analyzer.diagonalsMinMax(matrix);  
}  
}
```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Develop a program for managing employee information that caters to both full-time and part-time employees. The program should be capable of computing the salary for each category of employee and presenting their particulars. To achieve this, create two classes, FullTimeEmployee and PartTimeEmployee, that adhere to the Employee interface.

The program is expected to accept input data, including the name and monthly salary for full-time employees, as well as the name, hourly rate, and hours worked for part-time employees. Subsequently, it should calculate and exhibit the employee details and their respective salaries.

For Full-Time employees, the annual salary should be calculated as 12 times the monthly salary.

For Part-Time employees, the salary calculation should be based on the formula: hourly rate \* hours worked.

##### **Input Format**

The first line of input should be a string representing the name of a full-time employee.

The second line of input should be an integer representing the monthly salary of the full-time employee.

The third line of input should be a string representing the name of a part-time employee.

The fourth line of input should be an integer representing the hourly rate of the part-time employee.

The fifth line of input should be an integer representing the number of hours worked by the part-time employee.

### ***Output Format***

The output displays the following details:

Full-Time Employee Details:

Name: [Full-Time Employee Name] (string)

Monthly Salary: \$[Monthly Salary] (integer)

Annual Salary: \$[12 times Monthly Salary] (integer)

Part-Time Employee Details:

Name: [Part-Time Employee Name] (string)

Hourly Rate: \$[Hourly Rate] (integer)

Hours Worked: [Hours Worked] hours (integer)

Monthly Salary: \$[Calculated Monthly Salary] (integer)

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: John Smith

15000

Mary Johnson

100

100

Output: Full-Time Employee Details:

Name: John Smith

Monthly Salary: \$15000

Annual Salary: \$180000

Part-Time Employee Details:

Name: Mary Johnson

Hourly Rate: \$100

Hours Worked: 100 hours

Monthly Salary: \$10000

### **Answer**

```
import java.util.Scanner;
```

```
interface Employee {  
    void displayDetails();  
}
```

```
class FullTimeEmployee implements Employee {  
    private String name;  
    private int monthlySalary;  
  
    public FullTimeEmployee(String name, int monthlySalary) {  
        this.name = name;  
        this.monthlySalary = monthlySalary;  
    }  
  
    public int getAnnualSalary() {  
        return monthlySalary * 12;  
    }  
}
```

```
@Override  
public void displayDetails() {  
    System.out.println("Full-Time Employee Details:");  
    System.out.println("Name: " + name);  
    System.out.println("Monthly Salary: $" + monthlySalary);  
    System.out.println("Annual Salary: $" + getAnnualSalary());  
}  
}
```

```
class PartTimeEmployee implements Employee {  
    private String name;  
    private int hourlyRate;
```

```

private int hoursWorked;

public PartTimeEmployee(String name, int hourlyRate, int hoursWorked) {
    this.name = name;
    this.hourlyRate = hourlyRate;
    this.hoursWorked = hoursWorked;
}

public int getMonthlySalary() {
    return hourlyRate * hoursWorked;
}

@Override
public void displayDetails() {
    System.out.println("Part-Time Employee Details:");
    System.out.println("Name: " + name);
    System.out.println("Hourly Rate: $" + hourlyRate);
    System.out.println("Hours Worked: " + hoursWorked + " hours");
    System.out.println("Monthly Salary: $" + getMonthlySalary());
}
}

class EmployeeInheritanceDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String fullName = scanner.nextLine();
        int fullTimeSalary = scanner.nextInt();
        scanner.nextLine();
        String partTimeName = scanner.nextLine();
        int hourlyRate = scanner.nextInt();
        int hoursWorked = scanner.nextInt();
        FullTimeEmployee fullTimeEmployee = new FullTimeEmployee(fullName,
fullTimeSalary);
        PartTimeEmployee partTimeEmployee = new
PartTimeEmployee(partTimeName, hourlyRate, hoursWorked);
        fullTimeEmployee.displayDetails();
        System.out.println();
        partTimeEmployee.displayDetails();
        scanner.close();
    }
}

```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Ubendra L  
Email: 241901120@rajalakshmi.edu.in  
Roll no: 241901120  
Phone: 8610285702  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_2028\_OOPS using Java\_Week 7\_CY

Attempt : 1  
Total Mark : 40  
Marks Obtained : 40

#### Section 1 : Coding

##### 1. Problem Statement:

Sam is developing a geometry application and needs a class for trapezoid calculations. Create a "Trapezoid" class implementing a "ShapeInput" interface with a method to input trapezoid dimensions.

Also, implement a "ShapeCalculator" interface with methods to compute area and perimeter. In the "Main" class, instantiate Trapezoid, gather user input, and display the calculated area and perimeter with two decimal places.

##### Note

Area of Trapezoid =  $(1/2) * (base1 + base2) * height$

Perimeter of Trapezoid =  $base1 + base2 + side1 + side2$

### ***Input Format***

The first line of input is a double-point value representing base1 of the trapezoid.

The second line of input is a double-point value representing base2 of the trapezoid.

The third line of input is a double-point value representing the height of the trapezoid.

The fourth line of input is a double-point value representing side1 of the trapezoid.

The fifth line of input is a double-point value representing side2 of the trapezoid.

### ***Output Format***

The output displays the two lines of the calculated area (double type) and perimeter (double type) of the trapezoid, each rounded to two decimal places in the following format:

"Area of the Trapezoid: <<calculated area>>".

Perimeter of the Trapezoid: <<calculated perimeter>>".

Refer to the sample output for the formatting specifications.

### ***Sample Test Case***

Input: 1.0

2.0

1.0

3.0

1.0

Output: Area of the Trapezoid: 1.50

Perimeter of the Trapezoid: 7.00

### ***Answer***

```
import java.util.Scanner;
```

```
interface ShapeInput {  
    void getInput();  
}
```

```
interface ShapeCalculator {  
    double calculateArea();  
    double calculatePerimeter();  
}
```

```
class Trapezoid implements ShapeInput, ShapeCalculator {  
    private double base1;  
    private double base2;  
    private double height;  
    private double side1;  
    private double side2;
```

```
    @Override  
    public void getInput() {  
        Scanner sc = new Scanner(System.in);  
        base1 = sc.nextDouble();  
        base2 = sc.nextDouble();  
        height = sc.nextDouble();  
        side1 = sc.nextDouble();  
        side2 = sc.nextDouble();  
    }
```

```
    @Override  
    public double calculateArea() {  
        return 0.5 * (base1 + base2) * height;  
    }
```

```
    @Override  
    public double calculatePerimeter() {  
        return base1 + base2 + side1 + side2;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Trapezoid trapezoid = new Trapezoid();  
        trapezoid.getInput();  
    }  
}
```

```
double area = trapezoid.calculateArea();
double perimeter = trapezoid.calculatePerimeter();

System.out.println("Area of the Trapezoid: " + String.format("%.2f", area));
System.out.println("Perimeter of the Trapezoid: " + String.format("%.2f",
perimeter));
}
```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Maria, an online store owner, is looking to implement a pricing system that calculates the final price of products after applying discounts. She needs a program that takes the original price of a product and the discount percentage as input and computes the final discounted price. The discount is applied as a percentage of the original price. Maria wants to ensure that the final price is formatted to display exactly two decimal places.

Implement this functionality using the PriceCalculator interface and the DiscountCalculator class.

### **Input Format**

The first line of input consists of the original price (a double value).

The second line of input consists of a discount percentage (a double value).

### **Output Format**

The output displays the final price after the discount, adhering to the following format: "Final Price after discount: \$[final\_price]".

Here, [final\_price] should be replaced with the calculated final price, formatted as a currency value with two decimal places.

Refer to the sample output for the formatting specifications.

### Sample Test Case

Input: 100.0

10.0

Output: Final Price after discount: \$90.00

### Answer

```
import java.util.Scanner;

interface PriceCalculator {
    double calculatePrice(double originalPrice, double discountPercentage);
}

class DiscountCalculator implements PriceCalculator {
    @Override
    public double calculatePrice(double originalPrice, double discountPercentage)
    {
        double discountAmount = originalPrice * (discountPercentage / 100.0);
        return originalPrice - discountAmount;
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double originalPrice = scanner.nextDouble();
        double discount = scanner.nextDouble();
        PriceCalculator calculator = new DiscountCalculator();
        double finalPrice = calculator.calculatePrice(originalPrice, discount);
        System.out.printf("Final Price after discount: $%.2f\n", finalPrice); //
        Formats output to 2 decimal places
    }
}
```

Status : Correct

Marks : 10/10

### 3. Problem Statement

A developer aims to create a budget management system using two interfaces, ExpenseRecorder for recording expenses and BudgetCalculator for calculating remaining budgets.

The ExpenseTracker class implements these interfaces, allowing users to input an initial budget and record expenses iteratively until entering 0.0 as a sentinel value.

The program then computes and displays the remaining budget or notifies of budget exceedance.

Example

Input

100.0

20.0 30.0 10.0 0.0

Output

Remaining budget: Rs. 40.00

Explanation

The initial budget is 100.0. Expenses of 20.0, 30.0, and 10.0 are recorded.

Remaining budget is calculated ( $100.0 - 20.0 - 30.0 - 10.0 = 40.0$ ).

### ***Input Format***

The first line of input is the initial budget as a double-point number (double type). The budget is a positive number.

The second line of input consists of individual expenses as double-point numbers. Each expense is separated by space.

To end the input, an expense of 0.0 is used.

### ***Output Format***

The output displays the remaining budget, formatted to two decimal places, in the following format:

If the remaining budget (double type) is non-negative, it prints "Remaining

budget: Rs. [remainingBudget]".

If the remaining budget is negative, it prints "No remaining budget, You've exceeded your budget!"

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 100.0

20.0 30.0 10.0 0.0

Output: Remaining budget: Rs. 40.00

### **Answer**

```
import java.util.Scanner;
```

```
interface ExpenseRecorder {  
    void recordExpense(double expense);  
}
```

```
interface BudgetCalculator {  
    double calculateRemainingBudget();  
}
```

```
class ExpenseTracker implements ExpenseRecorder, BudgetCalculator {  
    private double initialBudget;  
    private double totalExpenses;
```

```
    public ExpenseTracker(double budget) {  
        this.initialBudget = budget;  
        this.totalExpenses = 0.0;  
    }
```

```
    @Override
```

```
    public void recordExpense(double expense) {  
        if (expense != 0.0) {  
            totalExpenses += expense;  
        }  
    }
```

```

@Override
public double calculateRemainingBudget() {
    return initialBudget - totalExpenses;
}
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double budget = scanner.nextDouble();

        ExpenseTracker tracker = new ExpenseTracker(budget);

        double expense;
        do {
            expense = scanner.nextDouble();
            tracker.recordExpense(expense);
        } while (expense != 0.0);

        double remainingBudget = tracker.calculateRemainingBudget();
        if (remainingBudget >= 0) {
            System.out.printf("Remaining budget: Rs. %.2f", remainingBudget);
        } else {
            System.out.println("No remaining budget, You've exceeded your
budget!");
        }
    }
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

John is developing a car loan calculator and has structured his program using two interfaces, Principal and InterestRate, defining methods for principal and interest rate retrieval.

The Loan class implements these interfaces, taking principal and annual interest rates as parameters. User input is solicited for these values, and the program ensures their validity before performing calculations. If input

values are invalid (less than or equal to zero), an error message is displayed.

Note: Total interest = principal \* interest rate \* years

### ***Input Format***

The first line of input consists of a double value P, representing the principal.

The second line consists of a double value R, representing the annual interest rate.

The third line consists of an integer value N, representing the loan duration in years.

### ***Output Format***

If the input values are valid, print "Total interest paid: Rs. " followed by a double value, representing the total interest paid, rounded off to two decimal places.

If the input values are invalid (negative or zero values for principal, annual interest rate, or loan duration), print "Invalid input values!".

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 20000.00  
0.05  
5

Output: Total interest paid: Rs.5000.00

### ***Answer***

```
import java.util.Scanner;

interface Principal {
    double getPrincipal();
}

interface InterestRate {
    double getInterestRate();
}
```

```
}
```

```
class Loan implements Principal, InterestRate {  
    private double principal;  
    private double interestRate;
```

```
  
    public Loan(double principal, double interestRate) {  
        this.principal = principal;  
        this.interestRate = interestRate;  
    }  
  
    @Override  
    public double getPrincipal() {  
        return principal;  
    }  
  
    @Override  
    public double getInterestRate() {  
        return interestRate;  
    }  
  
    public double calculateTotalInterest(int years) {  
        return principal * interestRate * years;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        double carPrice = scanner.nextDouble();  
  
        double annualInterestRate = scanner.nextDouble();  
  
        int loanDuration = scanner.nextInt();  
  
        if (carPrice <= 0 || annualInterestRate <= 0 || loanDuration <= 0) {  
            System.out.println("Invalid input values!");  
            return;  
        }  
  
        Loan carLoan = new Loan(carPrice, annualInterestRate);  
        double totalInterest = carLoan.calculateTotalInterest(loanDuration);  
    }  
}
```

```
    System.out.printf("Total interest paid: Rs.%.2f%n", totalInterest);  
  }  
}
```

**Status :** Correct

**Marks : 10/10**