**OOPJ 2023-2024 Paper (Assignment)**

**Name: Rushabh Abhay Shah**

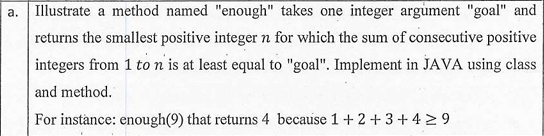
**Roll No.: C126**

**Division: D1**

**Date: 15-04-2025**

**Subject: OOPJ**

**Q1 a.**

****

**CODE**

public class q1a {

// Method that returns the smallest positive integer n where sum from 1 to n >= goal

public static int enough(int goal) {

int sum = 0;

int n = 0;

while (sum < goal) {

n++;

sum += n;

}

return n;

}

public static void main(String[] args) {

// Test case from the problem

int result = enough(9);

System.out.println("enough(9) returns " + result);

// Verification

int sum = 0;

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

sum += i;

}

System.out.println("Sum from 1 to " + result + " = " + sum);

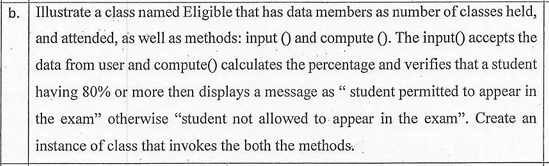
}

}

**OUTPUT**

****

**Q1 b.**

****

**CODE**

public class q1b {

// Data members

private int classesHeld;

private int classesAttended;

// Constructor

public q1b() {

classesHeld = 0;

classesAttended = 0;

}

// Method to input data from user

public void input() {

// In a real application, you would use Scanner class

// For demonstration purposes:

java.util.Scanner scanner = new java.util.Scanner(System.in);

System.out.print("Enter number of classes held: ");

classesHeld = scanner.nextInt();

System.out.print("Enter number of classes attended: ");

classesAttended = scanner.nextInt();

}

// Method to compute attendance percentage and verify eligibility

public void compute() {

// Calculate attendance percentage

double percentage = (double) classesAttended / classesHeld \* 100;

// Display the percentage

System.out.println("Attendance Percentage: " + percentage + "%");

// Check if student is eligible (80% or more attendance)

if (percentage >= 80) {

System.out.println("Student permitted to appear in the exam");

} else {

System.out.println("Student not allowed to appear in the exam");

}

}

// Main method to create instance and invoke methods

public static void main(String[] args) {

// Create an instance of Eligible class

q1b student = new q1b();

// Call input method to get data

student.input();

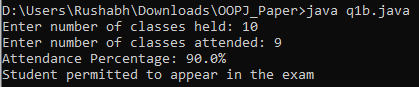
// Call compute method to determine eligibility

student.compute();

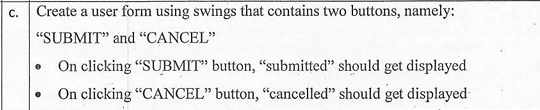
}

}

**OUTPUT**

****

**Q1 c.**

****

**CODE**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class q1c extends JFrame {

private JLabel statusLabel;

public q1c() {

// Set up the frame

setTitle("Simple Form");

setSize(300, 150);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

// Create buttons panel

JPanel buttonPanel = new JPanel();

JButton submitButton = new JButton("SUBMIT");

JButton cancelButton = new JButton("CANCEL");

// Add action listeners to buttons

submitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

statusLabel.setText("submitted");

}

});

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

statusLabel.setText("cancelled");

}

});

// Add buttons to panel

buttonPanel.add(submitButton);

buttonPanel.add(cancelButton);

// Create status label

statusLabel = new JLabel("", JLabel.CENTER);

// Add components to frame

add(buttonPanel, BorderLayout.CENTER);

add(statusLabel, BorderLayout.SOUTH);

// Display the frame

setLocationRelativeTo(null);

setVisible(true);

}

public static void main(String[] args) {

// Create the form on the Event Dispatch Thread

SwingUtilities.invokeLater(new Runnable() {

public void run() {

new q1c();

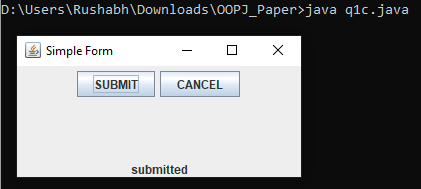
}

});

}

}

**OUTPUT**

****

**Q1 d.**

****

**CODE**

public class q1d {

public static void main(String[] args) {

// Create an array

int[] numbers = {1, 2, 3, 4, 5};

System.out.println("Array created with 5 elements (indexes 0-4)");

// Demonstrate safe access (within bounds)

try {

System.out.println("\nTrying to access a valid index (index 2):");

int value = numbers[2];

System.out.println("Value at index 2: " + value);

} catch (ArrayIndexOutOfBoundsException e) {

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

}

// Demonstrate exception (beyond upper bound)

try {

System.out.println("\nTrying to access beyond upper bound (index 10):");

int value = numbers[10]; // This will cause ArrayIndexOutOfBoundsException

System.out.println("This line won't execute");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Array index out of bounds");

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

}

// Demonstrate exception (negative index)

try {

System.out.println("\nTrying to access negative index (index -1):");

int value = numbers[-1]; // This will cause ArrayIndexOutOfBoundsException

System.out.println("This line won't execute");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Negative index not allowed");

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

} finally {

System.out.println("\nFinally block executes regardless of exception");

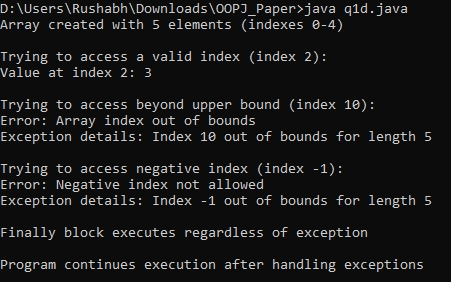
}

System.out.println("\nProgram continues execution after handling exceptions");

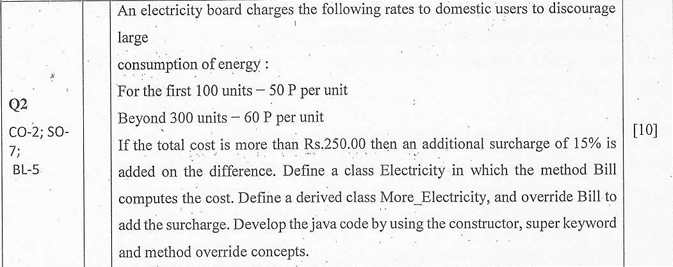
}

}

**OUTPUT**

****

**Q2.**

****

**CODE**

// Base class for electricity billing

class Electricity {

protected int units;

// Constructor

public Electricity(int units) {

this.units = units;

}

// Method to compute the bill

public double Bill() {

double cost = 0;

if (units <= 100) {

cost = units \* 0.50;

} else if (units <= 300) {

cost = 100 \* 0.50 + (units - 100) \* 0.60;

} else {

cost = 100 \* 0.50 + 200 \* 0.60 + (units - 300) \* 0.60;

}

return cost;

}

}

// Derived class that adds surcharge for high bills

class More\_Electricity extends Electricity {

// Constructor using super keyword to call the parent constructor

public More\_Electricity(int units) {

super(units);

}

// Override the Bill method to add surcharge

@Override

public double Bill() {

// Call the parent class Bill method

double cost = super.Bill();

// Add surcharge if cost exceeds Rs. 250

if (cost > 250.0) {

double surcharge = (cost - 250.0) \* 0.15;

cost += surcharge;

}

return cost;

}

}

// Main class to test the implementation

public class q2 {

public static void main(String[] args) {

// Test with different consumption levels

int[] testUnits = {80, 200, 400};

for (int units : testUnits) {

// Create base class object

Electricity regular = new Electricity(units);

double regularBill = regular.Bill();

// Create derived class object

More\_Electricity withSurcharge = new More\_Electricity(units);

double totalBill = withSurcharge.Bill();

System.out.println("Units consumed: " + units);

System.out.println("Regular bill: Rs. " + regularBill);

System.out.println("Bill with surcharge: Rs. " + totalBill);

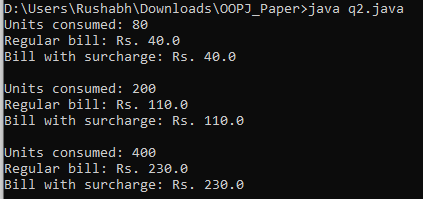
System.out.println();

}

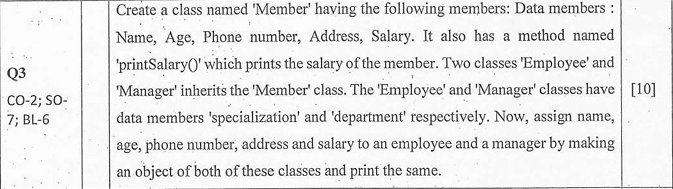
}

}

**OUTPUT**

****

**Q3.**

****

**CODE**

// Base class Member

class Member {

// Data members

protected String name;

protected int age;

protected String phoneNumber;

protected String address;

protected double salary;

// Constructor

public Member(String name, int age, String phoneNumber, String address, double salary) {

this.name = name;

this.age = age;

this.phoneNumber = phoneNumber;

this.address = address;

this.salary = salary;

}

// Method to print salary

public void printSalary() {

System.out.println("Salary: Rs. " + salary);

}

// Method to display member details

public void displayDetails() {

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

System.out.println("Age: " + age);

System.out.println("Phone: " + phoneNumber);

System.out.println("Address: " + address);

printSalary();

}

}

// Derived class Employee

class Employee extends Member {

// Additional data member

private String specialization;

// Constructor

public Employee(String name, int age, String phoneNumber, String address,

double salary, String specialization) {

// Call parent constructor using super

super(name, age, phoneNumber, address, salary);

this.specialization = specialization;

}

// Method to display employee details

public void displayDetails() {

super.displayDetails();

System.out.println("Specialization: " + specialization);

}

}

// Derived class Manager

class Manager extends Member {

// Additional data member

private String department;

// Constructor

public Manager(String name, int age, String phoneNumber, String address,

double salary, String department) {

// Call parent constructor using super

super(name, age, phoneNumber, address, salary);

this.department = department;

}

// Method to display manager details

public void displayDetails() {

super.displayDetails();

System.out.println("Department: " + department);

}

}

// Main class to test the implementation

public class q3 {

public static void main(String[] args) {

// Create an Employee object

Employee emp = new Employee("John Smith", 28, "9876543210",

"123 Main St, City", 50000.0, "Software Development");

// Create a Manager object

Manager mgr = new Manager("Jane Doe", 35, "8765432109",

"456 Park Ave, City", 75000.0, "Information Technology");

// Display Employee details

System.out.println("EMPLOYEE DETAILS:");

emp.displayDetails();

System.out.println();

// Display Manager details

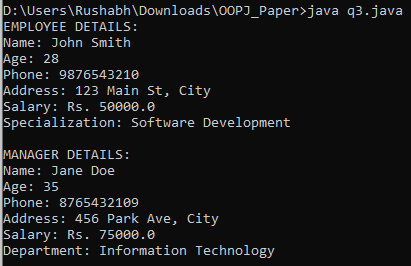
System.out.println("MANAGER DETAILS:");

mgr.displayDetails();

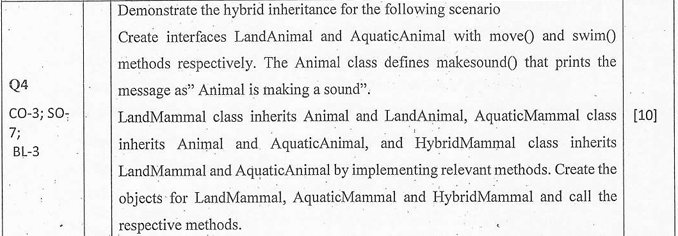
}

}

**OUTPUT**

****

**Q4.**

****

**CODE**

// Interface for land animals

interface LandAnimal {

void move();

}

// Interface for aquatic animals

interface AquaticAnimal {

void swim();

}

// Base Animal class

class Animal {

public void makeSound() {

System.out.println("Animal is making a sound");

}

}

// Class for land mammals - inherits Animal class and implements LandAnimal interface

class LandMammal extends Animal implements LandAnimal {

@Override

public void move() {

System.out.println("Land mammal is moving on land");

}

@Override

public void makeSound() {

System.out.println("Land mammal is making a sound");

}

}

// Class for aquatic mammals - inherits Animal class and implements AquaticAnimal interface

class AquaticMammal extends Animal implements AquaticAnimal {

@Override

public void swim() {

System.out.println("Aquatic mammal is swimming");

}

@Override

public void makeSound() {

System.out.println("Aquatic mammal is making a sound");

}

}

// Hybrid mammal class - inherits from LandMammal and implements AquaticAnimal interface

// Note: Java doesn't support multiple inheritance of classes, so we inherit from one class

// and implement the other functionality through an interface

class HybridMammal extends LandMammal implements AquaticAnimal {

@Override

public void swim() {

System.out.println("Hybrid mammal is swimming");

}

@Override

public void move() {

System.out.println("Hybrid mammal is moving on land");

}

@Override

public void makeSound() {

System.out.println("Hybrid mammal is making a sound");

}

}

// Main class to demonstrate the hybrid inheritance

public class q4 {

public static void main(String[] args) {

// Create objects of each class

LandMammal landMammal = new LandMammal();

AquaticMammal aquaticMammal = new AquaticMammal();

HybridMammal hybridMammal = new HybridMammal();

// Test LandMammal methods

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

landMammal.makeSound();

landMammal.move();

// Test AquaticMammal methods

System.out.println("\n--- Aquatic Mammal ---");

aquaticMammal.makeSound();

aquaticMammal.swim();

// Test HybridMammal methods

System.out.println("\n--- Hybrid Mammal ---");

hybridMammal.makeSound();

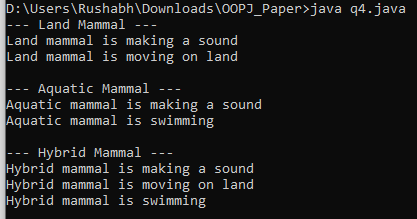
hybridMammal.move();

hybridMammal.swim();

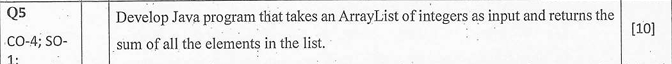
}

}

**OUTPUT**

****

**Q5.**

****

**CODE**

import java.util.ArrayList;

import java.util.Arrays;

public class q5 {

// Method to calculate sum of all elements in an ArrayList of integers

public static int sumArrayList(ArrayList<Integer> list) {

int sum = 0;

// Iterate through each element and add to sum

for (Integer num : list) {

sum += num;

}

return sum;

}

public static void main(String[] args) {

// Create and initialize several ArrayLists for testing

// Test case 1: ArrayList with positive integers

ArrayList<Integer> list1 = new ArrayList<>(Arrays.asList(5, 10, 15, 20, 25));

System.out.println("List 1: " + list1);

System.out.println("Sum of List 1: " + sumArrayList(list1));

// Test case 2: ArrayList with negative integers

ArrayList<Integer> list2 = new ArrayList<>(Arrays.asList(-3, -6, -9, -12));

System.out.println("\nList 2: " + list2);

System.out.println("Sum of List 2: " + sumArrayList(list2));

// Test case 3: ArrayList with mixed integers

ArrayList<Integer> list3 = new ArrayList<>(Arrays.asList(7, -4, 9, -2, 5, -8));

System.out.println("\nList 3: " + list3);

System.out.println("Sum of List 3: " + sumArrayList(list3));

// Test case 4: Empty ArrayList

ArrayList<Integer> list4 = new ArrayList<>();

System.out.println("\nList 4: " + list4);

System.out.println("Sum of List 4: " + sumArrayList(list4));

// Alternative implementation using Java 8 Stream API

System.out.println("\n--- Using Stream API ---");

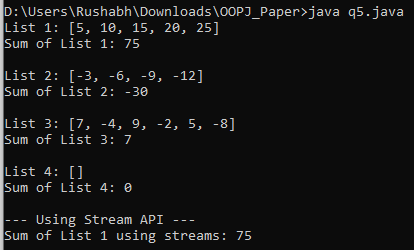
int sum1 = list1.stream().mapToInt(Integer::intValue).sum();

System.out.println("Sum of List 1 using streams: " + sum1);

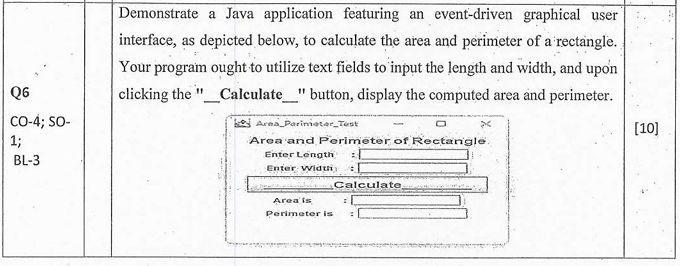
}

}

**OUTPUT**

****

**Q6.**

****

**CODE**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class q6 extends JFrame implements ActionListener {

// Declare the components

private JLabel titleLabel, lengthLabel, widthLabel, areaLabel, perimeterLabel;

private JTextField lengthField, widthField, areaField, perimeterField;

private JButton calculateButton;

public q6() {

// Set up the frame

super("Area\_Perimeter\_Test");

setSize(400, 250);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(null); // Using absolute positioning

// Create the title label

titleLabel = new JLabel("Area and Perimeter of Rectangle");

titleLabel.setBounds(100, 10, 250, 20);

add(titleLabel);

// Create input fields and labels

lengthLabel = new JLabel("Enter Length:");

lengthLabel.setBounds(30, 40, 100, 20);

add(lengthLabel);

lengthField = new JTextField();

lengthField.setBounds(150, 40, 200, 20);

add(lengthField);

widthLabel = new JLabel("Enter Width:");

widthLabel.setBounds(30, 70, 100, 20);

add(widthLabel);

widthField = new JTextField();

widthField.setBounds(150, 70, 200, 20);

add(widthField);

// Create calculate button

calculateButton = new JButton("\_\_\_\_\_\_\_\_\_Calculate\_\_\_\_\_\_\_\_\_");

calculateButton.setBounds(50, 100, 300, 30);

calculateButton.addActionListener(this);

add(calculateButton);

// Create output fields and labels

areaLabel = new JLabel("Area is:");

areaLabel.setBounds(30, 140, 100, 20);

add(areaLabel);

areaField = new JTextField();

areaField.setBounds(150, 140, 200, 20);

areaField.setEditable(false);

add(areaField);

perimeterLabel = new JLabel("Perimeter is:");

perimeterLabel.setBounds(30, 170, 100, 20);

add(perimeterLabel);

perimeterField = new JTextField();

perimeterField.setBounds(150, 170, 200, 20);

perimeterField.setEditable(false);

add(perimeterField);

// Center the frame on screen

setLocationRelativeTo(null);

setVisible(true);

}

@Override

public void actionPerformed(ActionEvent e) {

if (e.getSource() == calculateButton) {

try {

// Get the length and width from the text fields

double length = Double.parseDouble(lengthField.getText());

double width = Double.parseDouble(widthField.getText());

// Calculate area and perimeter

double area = length \* width;

double perimeter = 2 \* (length + width);

// Display the results

areaField.setText(String.valueOf(area));

perimeterField.setText(String.valueOf(perimeter));

} catch (NumberFormatException ex) {

// Handle invalid input

JOptionPane.showMessageDialog(this,

"Please enter valid numeric values for length and width.",

"Input Error",

JOptionPane.ERROR\_MESSAGE);

}

}

}

public static void main(String[] args) {

// Create the application on the Event Dispatch Thread

SwingUtilities.invokeLater(new Runnable() {

public void run() {

new q6();

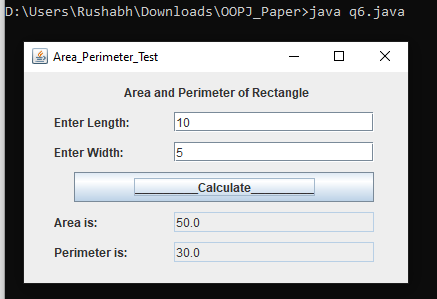
}

});

}

}

**OUTPUT**

****