# Question 1

## Problem: Demonstrate Inheritance, Method Overriding, Constructor Chaining, and `super` keyword.

Ans –

```
import java.util.Scanner;


class Employee {

   int id;

   String name;

   double baseSalary;


   Employee(int id, String name, double baseSalary) {

      this.id = id;

      this.name = name;

      this.baseSalary = baseSalary;

   }


   double calculateSalary() {

      return baseSalary;

   }


   void display() {

      System.out.println("ID: " + id + ", Name: " + name + ", Salary: " + calculateSalary());

   }

}
```

```
class Manager extends Employee {
    double bonus;

    Manager(int id, String name, double baseSalary, double bonus) {
        super(id, name, baseSalary);  // Constructor chaining using super
        this.bonus = bonus;
    }

    @Override
    double calculateSalary() {
        return baseSalary + bonus;
    }
}

class Developer extends Employee {
    double allowance;

    Developer(int id, String name, double baseSalary, double allowance) {
        super(id, name, baseSalary);
        this.allowance = allowance;
    }

    @Override
    double calculateSalary() {
        return baseSalary + allowance;

    }

}


public class InheritanceDemo {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);


        System.out.println("Enter Manager details: id name baseSalary bonus");

        int mid = sc.nextInt();

        String mname = sc.next();

        double mbase = sc.nextDouble();

        double mbonus = sc.nextDouble();

        Manager m = new Manager(mid, mname, mbase, mbonus);


        System.out.println("Enter Developer details: id name baseSalary allowance");

        int did = sc.nextInt();

        String dname = sc.next();

        double dbase = sc.nextDouble();

        double dallow = sc.nextDouble();

        Developer d = new Developer(did, dname, dbase, dallow);


        System.out.println("\nSalary Details:");

        m.display();

        d.display();
    }
}
```

# Question 2

## Problem: Swing GUI with string operations (Reverse, Uppercase, Count Vowels)

```
import javax.swing.*;
import java.awt.event.*;

public class StringOperationsGUI extends JFrame implements ActionListener {
    JTextField inputField;
    JLabel resultLabel;
    JButton reverseBtn, upperBtn, vowelBtn;

    StringOperationsGUI() {
        setTitle("String Operations");
        setSize(400, 200);
        setLayout(null);

        inputField = new JTextField();
        inputField.setBounds(50, 30, 280, 25);
        add(inputField);

        reverseBtn = new JButton("Reverse");
        reverseBtn.setBounds(30, 70, 100, 30);
        add(reverseBtn);
        upperBtn = new JButton("Uppercase");

        upperBtn.setBounds(140, 70, 100, 30);

        add(upperBtn);


        vowelBtn = new JButton("Count Vowels");

        vowelBtn.setBounds(250, 70, 120, 30);

        add(vowelBtn);


        resultLabel = new JLabel("");

        resultLabel.setBounds(50, 120, 300, 25);

        add(resultLabel);


        reverseBtn.addActionListener(this);

        upperBtn.addActionListener(this);

        vowelBtn.addActionListener(this);


        setVisible(true);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }


    public void actionPerformed(ActionEvent e) {

        String text = inputField.getText();


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

            resultLabel.setText("Reversed: " + new StringBuilder(text).reverse());
        } else if (e.getSource() == upperBtn) {
            resultLabel.setText("Uppercase: " + text.toUpperCase());
        } else if (e.getSource() == vowelBtn) {
            int count = 0;
            for (char c : text.toLowerCase().toCharArray()) {
                if ("aeiou".indexOf(c) != -1) count++;
            }
            resultLabel.setText("Vowels Count: " + count);
        }
    }

    public static void main(String[] args) {
        new StringOperationsGUI();
    }
}
```

# Question 3

## Problem: Input an array and display sum, maximum, and even number count

```
import java.util.Scanner;

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

        System.out.print("Enter size of array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        int sum = 0, max = arr[0], evenCount = 0;

        for (int num : arr) {
            sum += num;
            if (num > max) max = num;
            if (num % 2 == 0) evenCount++;
        }

        System.out.println("Sum of elements: " + sum);
        System.out.println("Maximum element: " + max);
        System.out.println("Count of even numbers: " + evenCount);
    }
}
```

## Set b

## Question 1: Read and Write Student Details using Data I/O Streams

This program will write student details (ID, Name, Marks) to a file using `DataOutputStream` and then read and display them using `DataInputStream`.

Answer-

```
import java.io.*;
import java.util.Scanner;


public class StudentDetailsIO {


    private static final String FILE_NAME = "student_data.dat";


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


        // 1. Write Data to File
        System.out.println("--- Entering Student Details ---");
        try {
            writeStudentDetails(scanner);
        } catch (IOException e) {
            System.err.println("Error writing to file: " +
e.getMessage());
        }

        // 2. Read and Display Data from File
        System.out.println("\n--- Displaying Student Details
from File ---");
        try {
            readStudentDetails();
        } catch (IOException e) {
            System.err.println("Error reading from file: " +
e.getMessage());
        }

        scanner.close();
    }

    /**
```

```
     * Writes student data to a binary file using
DataOutputStream.
     */
    private static void writeStudentDetails(Scanner scanner)
throws IOException {
        try (DataOutputStream dos = new
DataOutputStream(new FileOutputStream(FILE_NAME))) {

            // Get data for one student
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            System.out.print("Enter Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Marks: ");
            double marks = scanner.nextDouble();

            // Write data to the stream
            dos.writeInt(id);
            dos.writeUTF(name); // writeUTF is good for Strings
            dos.writeDouble(marks);

            System.out.println("Student data written successfully.");
        }
    }

    /**
     * Reads and displays student data from a binary file using DataInputStream.
     */
    private static void readStudentDetails() throws IOException {
        try (DataInputStream dis = new DataInputStream(new FileInputStream(FILE_NAME))) {

            // Read data in the same order it was written
            int id = dis.readInt();

            String name = dis.readUTF();

            double marks = dis.readDouble();


            // Display the data

            System.out.println("ID: " + id);

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

            System.out.println("Marks: " + marks);


        } catch (EOFException e) {

            // This is expected when reading a single entry and reaching the end

            // If there were multiple students, this would handle the file end.

        }

    }

}
```

- Query successful

That's a great set of Java programming problems! Here are the simple Java codes for all three questions, focusing on the core concepts requested.

---

## Question 1: Read and Write Student Details using Data I/O Streams

This program will write student details (ID, Name, Marks) to a file using `DataOutputStream` and then read and display them using `DataInputStream`.

Java

```
import java.io.*;
import java.util.Scanner;

public class StudentDetailsIO {

    private static final String FILE_NAME = "student_data.dat";

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

        // 1. Write Data to File
        System.out.println("--- Entering Student Details ---");
        try {
            writeStudentDetails(scanner);
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }

        // 2. Read and Display Data from File
        System.out.println("\n--- Displaying Student Details from File ---");
        try {
            readStudentDetails();
        } catch (IOException e) {
            System.err.println("Error reading from file: " + e.getMessage());
        }

        scanner.close();
    }

    /**
     * Writes student data to a binary file using DataOutputStream.
     */
    private static void writeStudentDetails(Scanner scanner) throws
IOException {
        try (DataOutputStream dos = new DataOutputStream(new
FileOutputStream(FILE_NAME))) {

            // Get data for one student
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            System.out.print("Enter Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Marks: ");
            double marks = scanner.nextDouble();

            // Write data to the stream
            dos.writeInt(id);
            dos.writeUTF(name); // writeUTF is good for Strings
            dos.writeDouble(marks);

            System.out.println("Student data written successfully.");
        }
    }

    /**
     * Reads and displays student data from a binary file using
DataInputStream.
     */
    private static void readStudentDetails() throws IOException {
        try (DataInputStream dis = new DataInputStream(new
FileInputStream(FILE_NAME))) {

            // Read data in the same order it was written
            int id = dis.readInt();
            String name = dis.readUTF();
            double marks = dis.readDouble();

            // Display the data
            System.out.println("ID: " + id);
            System.out.println("Name: " + name);
            System.out.println("Marks: " + marks);

        } catch (EOFException e) {
            // This is expected when reading a single entry and reaching the
end
            // If there were multiple students, this would handle the file
end.
        }
    }
}
```

---

## Question 2: Simple Interest Calculator using Swing GUI

This program creates a basic Swing GUI with text fields for Principal, Rate, and Time, a "Calculate" button, and a label to display the result.

```
import javax.swing.*;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleInterestGUI extends JFrame implements ActionListener {

    // Components
    private JTextField principalField, rateField, timeField;
    private JLabel resultLabel;
    private JButton calculateButton;

    public SimpleInterestGUI() {
        // Frame setup
        setTitle("Simple Interest Calculator");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new java.awt.GridLayout(5, 2)); // Simple layout manager
```

```
// Initialize components
principalField = new JTextField(10);
rateField = new JTextField(10);
timeField = new JTextField(10);
calculateButton = new JButton("Calculate");
resultLabel = new JLabel("Simple Interest: 0.0");

// Add components to the frame
add(new JLabel("Principal (P):"));
add(principalField);
add(new JLabel("Rate (R):"));
add(rateField);
add(new JLabel("Time (T) in years:"));
add(timeField);
add(new JLabel("")); // Empty cell for spacing
add(calculateButton);
add(resultLabel);
```

```
        // Register the button with the ActionListener
        calculateButton.addActionListener(this);

        setVisible(true);
    }

    /**
     * Handles the button click event.
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == calculateButton) {
            calculateInterest();
        }
    }

    private void calculateInterest() {
        try {
            // Get input from text fields and convert to double
            double P =
Double.parseDouble(principalField.getText());
            double R = Double.parseDouble(rateField.getText());
            double T = Double.parseDouble(timeField.getText());

            // Calculate Simple Interest: SI = (P * R * T) / 100
            double simpleInterest = (P * R * T) / 100.0;

            // Display the result in the JLabel
            resultLabel.setText(String.format("Simple Interest:
%.2f", simpleInterest));

        } catch (NumberFormatException ex) {
            // Handle Type Conversion error (non-numeric input)
            resultLabel.setText("Error: Enter valid numbers.");
            JOptionPane.showMessageDialog(this, "Please enter
valid numeric values for Principal, Rate, and Time.", "Input
Error", JOptionPane.ERROR_MESSAGE);
```

```
        }
    }

    public static void main(String[] args) {
        // Use SwingUtilities.invokeLater to ensure thread-safety
        SwingUtilities.invokeLater(() -> new SimpleInterestGUI());
    }
}
```

- Query successful

That's a great set of Java programming problems! Here are the simple Java codes for all three questions, focusing on the core concepts requested.

---

## Question 1: Read and Write Student Details using Data I/O Streams

This program will write student details (ID, Name, Marks) to a file using `DataOutputStream` and then read and display them using `DataInputStream`.

Java

```java
import java.io.*;
import java.util.Scanner;

public class StudentDetailsIO {
    private static final String FILE_NAME = "student_data.dat";

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

        // 1. Write Data to File
        System.out.println("--- Entering Student Details ---");
        try {
            writeStudentDetails(scanner);
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }

        // 2. Read and Display Data from File
        System.out.println("\n--- Displaying Student Details from File ---");
        try {
            readStudentDetails();
        } catch (IOException e) {
            System.err.println("Error reading from file: " + e.getMessage());
        }

        scanner.close();
    }

    /**
     * Writes student data to a binary file using DataOutputStream.
     */
    private static void writeStudentDetails(Scanner scanner) throws
IOException {
        try (DataOutputStream dos = new DataOutputStream(new
FileOutputStream(FILE_NAME))) {

            // Get data for one student
            System.out.print("Enter ID: ");
            int id = scanner.nextInt();
            scanner.nextLine(); // Consume newline
            System.out.print("Enter Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Marks: ");
            double marks = scanner.nextDouble();

            // Write data to the stream
            dos.writeInt(id);
            dos.writeUTF(name); // writeUTF is good for Strings
            dos.writeDouble(marks);

            System.out.println("Student data written successfully.");
        }
    }

    /**
     * Reads and displays student data from a binary file using
DataInputStream.
     */
    private static void readStudentDetails() throws IOException {
        try (DataInputStream dis = new DataInputStream(new
FileInputStream(FILE_NAME))) {

            // Read data in the same order it was written
            int id = dis.readInt();
            String name = dis.readUTF();
            double marks = dis.readDouble();

            // Display the data
            System.out.println("ID: " + id);
            System.out.println("Name: " + name);
            System.out.println("Marks: " + marks);

        } catch (EOFException e) {
            // This is expected when reading a single entry and reaching the
end
            // If there were multiple students, this would handle the file
end.
        }
    }
}
```

---

# Question 2: Simple Interest Calculator using Swing GUI

This program creates a basic Swing GUI with text fields for Principal, Rate, and Time, a "Calculate" button, and a label to display the result.

Java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleInterestGUI extends JFrame implements ActionListener {

    // Components
    private JTextField principalField, rateField, timeField;
    private JLabel resultLabel;
    private JButton calculateButton;

    public SimpleInterestGUI() {
        // Frame setup
        setTitle("Simple Interest Calculator");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new java.awt.GridLayout(5, 2)); // Simple layout manager

        // Initialize components
        principalField = new JTextField(10);
        rateField = new JTextField(10);
        timeField = new JTextField(10);
        calculateButton = new JButton("Calculate");
        resultLabel = new JLabel("Simple Interest: 0.0");
        // Add components to the frame
        add(new JLabel("Principal (P):"));
        add(principalField);
        add(new JLabel("Rate (R):"));
        add(rateField);
        add(new JLabel("Time (T) in years:"));
        add(timeField);
        add(new JLabel("")); // Empty cell for spacing
        add(calculateButton);
        add(resultLabel);

        // Register the button with the ActionListener
        calculateButton.addActionListener(this);

        setVisible(true);
    }

    /**
     * Handles the button click event.
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == calculateButton) {
            calculateInterest();
        }
    }

    private void calculateInterest() {
        try {
            // Get input from text fields and convert to double
            double P = Double.parseDouble(principalField.getText());
            double R = Double.parseDouble(rateField.getText());
            double T = Double.parseDouble(timeField.getText());

            // Calculate Simple Interest: SI = (P * R * T) / 100
            double simpleInterest = (P * R * T) / 100.0;

            // Display the result in the JLabel
            resultLabel.setText(String.format("Simple Interest: %.2f",
simpleInterest));

        } catch (NumberFormatException ex) {
            // Handle Type Conversion error (non-numeric input)
            resultLabel.setText("Error: Enter valid numbers.");
            JOptionPane.showMessageDialog(this, "Please enter valid numeric
values for Principal, Rate, and Time.", "Input Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        // Use SwingUtilities.invokeLater to ensure thread-safety
        SwingUtilities.invokeLater(() -> new SimpleInterestGUI());
    }
}
```

---

## Question 3: String Analysis (Word Count, Longest Word, Digit Count)

This program takes a string input and calculates the total number of words, finds the longest word, and counts the total number of digits present.

```
import java.util.Scanner;


public class StringAnalysis {


    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string (e.g., This is year 2025):");
        String inputString = scanner.nextLine();
        scanner.close();


        // 1. Get the list of words using the split method
        // \\s+ splits the string by one or more whitespace characters
        String[] words = inputString.trim().split("\\s+");
        // Handle case where input is only whitespace (split might return [""])
        int wordCount = 0;
        if (words.length > 0 && !words[0].isEmpty()) {
            wordCount = words.length;
        }

        // 2. Find the Longest Word
        String longestWord = "";
        if (wordCount > 0) {
            longestWord = findLongestWord(words);
        }

        // 3. Count the number of digits
        int digitCount = countDigits(inputString);

        // Output Format
        System.out.println("\n--- Analysis Results ---");
        System.out.println("Total number of words: " + wordCount);
        System.out.println("Longest word: " + longestWord);
        System.out.println("Count of digits in the string: " + digitCount);
    }

    /**
     * Finds the longest word in an array of words.
     */
    private static String findLongestWord(String[] words) {
        String longest = "";
        for (String word : words) {
            // Use word.replaceAll("[^a-zA-Z0-9]", "") to remove punctuation
            // or just use word.length() if punctuation should be included
            if (word.length() > longest.length()) {
```

```
            longest = word;
        }
    }
    return longest;
}

/**
 * Counts the total number of digits (0-9) in the string.
 */
private static int countDigits(String str) {
    int count = 0;
    // Iterate over each character in the string
    for (int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        // Check if the character is a digit
        if (Character.isDigit(ch)) {
            count++;
        }
        }
        return count;
    }
}
```

Set c

## Question 1: Multilevel Inheritance with Method Overriding (Area and Volume)

This program demonstrates **multilevel inheritance** (`Shape` → `Rectangle` → `Cuboid`) and **method overriding** to calculate the area and volume of a cuboid. It also uses **constructor chaining**.

Ans-

```
import java.util.Scanner;


// 1. Base Class: Shape
class Shape {
    protected double length;
    protected double breadth;
    // Constructor for Shape
    public Shape(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
        System.out.println("Shape constructor called.");
    }

    // Method to be overridden (for Area)
    public double calculateArea() {
        return 0.0; // Base implementation
    }
}

// 2. Intermediate Class: Rectangle (Extends Shape)
class Rectangle extends Shape {

    // Constructor for Rectangle, chains to Shape constructor
    public Rectangle(double length, double breadth) {
        super(length, breadth); // Calls parent (Shape) constructor
        System.out.println("Rectangle constructor called.");
    }

    // Method Overriding: Calculate Area of Rectangle
    @Override
    public double calculateArea() {
        return length * breadth;
    }
}

// 3. Derived Class: Cuboid (Extends Rectangle)
class Cuboid extends Rectangle {
    private double height;

    // Constructor for Cuboid, chains to Rectangle constructor
    public Cuboid(double length, double breadth, double height) {
        super(length, breadth); // Calls parent (Rectangle) constructor
        this.height = height;
        System.out.println("Cuboid constructor called.");
    }

    // Method Overriding: Calculate Area of Cuboid (Total Surface Area)
    @Override
    public double calculateArea() {
        // Area = 2 * (lb + bh + lh)
        return 2 * (length * breadth + breadth * height + length * height);
    }

    // New Method: Calculate Volume of Cuboid
    public double calculateVolume() {
        // Volume = l * b * h
        return length * breadth * height;
    }
}

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

        System.out.println("Enter Length of Cuboid:");
        double l = scanner.nextDouble();
        System.out.println("Enter Breadth of Cuboid:");
        double b = scanner.nextDouble();
        System.out.println("Enter Height of Cuboid:");
        double h = scanner.nextDouble();

        scanner.close();
        System.out.println("------------------------------------");
        // Create an object of the final class
        Cuboid box = new Cuboid(l, b, h);

        // Display Area (Method Overridden in Cuboid for Total Surface Area)
        double area = box.calculateArea();

        // Display Volume (New Method in Cuboid)
        double volume = box.calculateVolume();

        System.out.println("\n*** Results for Cuboid ***");
        System.out.printf("Total Surface Area: %.2f\n", area);
        System.out.printf("Volume: %.2f\n", volume);
    }
}
```

- Query successful

That's another excellent set of Java programming problems! Here are the simple Java codes for all three questions, covering the requested concepts.

---

# Question 1: Multilevel Inheritance with Method Overriding (Area and Volume)

This program demonstrates **multilevel inheritance** (`Shape` → `Rectangle` → `Cuboid`) and **method overriding** to calculate the area and volume of a cuboid. It also uses **constructor chaining**.

Java

```java
import java.util.Scanner;

// 1. Base Class: Shape
class Shape {
    protected double length;
    protected double breadth;

    // Constructor for Shape
    public Shape(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
        System.out.println("Shape constructor called.");
    }

    // Method to be overridden (for Area)
    public double calculateArea() {
        return 0.0; // Base implementation
    }
}

// 2. Intermediate Class: Rectangle (Extends Shape)
class Rectangle extends Shape {

    // Constructor for Rectangle, chains to Shape constructor
    public Rectangle(double length, double breadth) {
        super(length, breadth); // Calls parent (Shape) constructor
        System.out.println("Rectangle constructor called.");
    }

    // Method Overriding: Calculate Area of Rectangle
    @Override
    public double calculateArea() {
        return length * breadth;
    }
}

// 3. Derived Class: Cuboid (Extends Rectangle)
class Cuboid extends Rectangle {
    private double height;

    // Constructor for Cuboid, chains to Rectangle constructor
    public Cuboid(double length, double breadth, double height) {
        super(length, breadth); // Calls parent (Rectangle) constructor
        this.height = height;
        System.out.println("Cuboid constructor called.");
    }

    // Method Overriding: Calculate Area of Cuboid (Total Surface Area)
    @Override
    public double calculateArea() {
        // Area = 2 * (lb + bh + lh)
        return 2 * (length * breadth + breadth * height + length * height);
    }

    // New Method: Calculate Volume of Cuboid
    public double calculateVolume() {
        // Volume = l * b * h
        return length * breadth * height;
    }
}

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

        System.out.println("Enter Length of Cuboid:");
        double l = scanner.nextDouble();
        System.out.println("Enter Breadth of Cuboid:");
        double b = scanner.nextDouble();
        System.out.println("Enter Height of Cuboid:");
        double h = scanner.nextDouble();

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

        // Create an object of the final class
        Cuboid box = new Cuboid(l, b, h);

        // Display Area (Method Overridden in Cuboid for Total Surface Area)
        double area = box.calculateArea();

        // Display Volume (New Method in Cuboid)
        double volume = box.calculateVolume();

        System.out.println("\n*** Results for Cuboid ***");
        System.out.printf("Total Surface Area: %.2f\n", area);
        System.out.printf("Volume: %.2f\n", volume);
    }
}
```

---

## Question 2: Swing GUI for Student Marks Entry

This program creates a basic Swing GUI to input three subject marks and calculate and display the **Total**, **Average**, and **Grade**.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class StudentMarksGUI extends JFrame implements ActionListener {

    private JTextField mark1Field, mark2Field, mark3Field;
    private JLabel totalLabel, averageLabel, gradeLabel;
    private JButton calculateButton;

    public StudentMarksGUI() {
        // Frame setup
        setTitle("Student Marks Calculator");
        setSize(400, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(7, 2, 10, 5)); // Rows, Cols, HGap, VGap

        // Initialize Input Fields
        mark1Field = new JTextField(5);
        mark2Field = new JTextField(5);
        mark3Field = new JTextField(5);

        // Initialize Labels for Output
        totalLabel = new JLabel("Total: ");
        averageLabel = new JLabel("Average: ");
        gradeLabel = new JLabel("Grade: ");

        // Initialize Button
        calculateButton = new JButton("Calculate");
        calculateButton.addActionListener(this); // Register ActionListener
```

```
    // Add Components to the Frame

    add(new JLabel("Mark 1:"));

    add(mark1Field);

    add(new JLabel("Mark 2:"));

    add(mark2Field);

    add(new JLabel("Mark 3:"));

    add(mark3Field);



    add(new JLabel("")); // Spacer

    add(calculateButton);



    add(totalLabel);

    add(averageLabel);

    add(gradeLabel);



    setVisible(true);

}
```

```
    /**
     * Handles the button click event.
     */
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == calculateButton) {
            calculateResults();
        }
    }

    private void calculateResults() {
        try {
            // Get input and convert to integers (Type
Conversion)
            int m1 = Integer.parseInt(mark1Field.getText());
            int m2 = Integer.parseInt(mark2Field.getText());
            int m3 = Integer.parseInt(mark3Field.getText());
```

```
// 1. Calculate Total
int total = m1 + m2 + m3;

// 2. Calculate Average (use double for precision)
double average = total / 3.0;

// 3. Determine Grade (Conditional Logic)
String grade;
if (average >= 75) {
    grade = "A";
} else if (average >= 60) {
    grade = "B";
} else if (average >= 50) {
    grade = "C";
} else {
    grade = "D (Fail)";
}
```

```
            // Display results in JLabels
            totalLabel.setText("Total: " + total);
            averageLabel.setText(String.format("Average: %.2f", average));
            gradeLabel.setText("Grade: " + grade);

        } catch (NumberFormatException ex) {
            // Handle error for non-numeric input
            JOptionPane.showMessageDialog(this, "Please enter valid integer marks in all fields.", "Input Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        // Ensure GUI is created on the Event Dispatch Thread (EDT)
        SwingUtilities.invokeLater(() -> new StudentMarksGUI());
    }

}
```

## Question 3: Prime Numbers between 1 and 100

This program uses **loops** and **conditional statements** to find and display all **prime numbers** between 1 and 100, along with their total count.

```
public class PrimeNumberFinder {



    public static void main(String[] args) {
        int count = 0;
        System.out.println("Prime Numbers between 1 and 100:");



        // Loop 1: Iterate from 2 up to 100 (1 is not prime)
        for (int i = 2; i <= 100; i++) {
            if (isPrime(i)) {
                System.out.print(i + " ");
                count++;
            }
        }
        // Display the total count
        System.out.println("\n\nTotal count of prime numbers: " + count);
    }

    /**
     * Checks if a given number is prime.
     * A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
     */
    public static boolean isPrime(int n) {
        // Handle case for n < 2 (though main loop starts at 2)
        if (n <= 1) {
            return false;
        }
        // Loop 2: Check for divisors from 2 up to the square root of n
        // Optimization: checking up to Math.sqrt(n) is sufficient
        for (int j = 2; j * j <= n; j++) {
            // Conditional Statement: If n is divisible by j, it's not prime
            if (n % j == 0) {
                return false;
            }
        }

        return true;
    }
}
```

Set d

## Question 1: Abstract Class and Interface Concepts

This program demonstrates an **Abstract Class** (`Shape`) to define common attributes and an **Interface** (`Measurable`) to define shared methods (`area()`, `perimeter()`). The concrete classes

(`Circle` and `Rectangle`) implement the interface and extend the abstract class, using **Method Overriding**.

Answer-

```
import java.util.Scanner;

// Interface: Defines methods for area and perimeter
interface Measurable {
    double area();
    double perimeter();
}

// Abstract Class: Defines common attributes and a constructor
abstract class Shape {
    protected String color;
    protected String name;

    public Shape(String name, String color) {
        this.name = name;
        this.color = color;
    }

    // Abstract method (optional here, but good practice for inheritance)
    // public abstract void displayInfo();
}

// Concrete Class 1: Circle
class Circle extends Shape implements Measurable {
    private double radius;

    public Circle(double radius, String color) {
        super("Circle", color); // Constructor Chaining
        this.radius = radius;
    }

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }

    @Override
    public double perimeter() {
        return 2 * Math.PI * radius; // Circumference
    }

    public void displayResults() {
        System.out.printf("--- %s (Color: %s) ---\n", name, color);
        System.out.printf("Area: %.2f\n", area());
        System.out.printf("Perimeter (Circumference): %.2f\n", perimeter());
    }
}
```

```
// Concrete Class 2: Rectangle
class Rectangle extends Shape implements Measurable {
    private double length;
    private double width;

    public Rectangle(double length, double width, String color) {
        super("Rectangle", color); // Constructor Chaining
        this.length = length;
        this.width = width;
    }

    @Override
    public double area() {
        return length * width;
    }

    @Override
    public double perimeter() {
        return 2 * (length + width);
    }

    public void displayResults() {
        System.out.printf("--- %s (Color: %s) ---\n", name,
color);
        System.out.printf("Area: %.2f\n", area());
        System.out.printf("Perimeter: %.2f\n", perimeter());
    }
}

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

        // Input for Circle
        System.out.println("Enter radius for Circle:");
```

```
double r = scanner.nextDouble();

// Input for Rectangle
System.out.println("Enter length for Rectangle:");
double l = scanner.nextDouble();
System.out.println("Enter width for Rectangle:");
double w = scanner.nextDouble();

scanner.close();

// Create objects and display results
System.out.println("\n*** Shape Calculations ***");

Circle circle = new Circle(r, "Blue");
circle.displayResults();

Rectangle rect = new Rectangle(l, w, "Green");
rect.displayResults();
```

```
    }
}
```

- Query successful

I'd be glad to provide simple Java code for all the questions in Set D, focusing on the core concepts requested.

---

# Question 1: Abstract Class and Interface Concepts

This program demonstrates an **Abstract Class** (`Shape`) to define common attributes and an **Interface** (`Measurable`) to define shared methods (`area()`, `perimeter()`). The concrete classes (`Circle` and `Rectangle`) implement the interface and extend the abstract class, using **Method Overriding**.

Java

```
import java.util.Scanner;

// Interface: Defines methods for area and perimeter
interface Measurable {
    double area();
    double perimeter();
}

// Abstract Class: Defines common attributes and a constructor
abstract class Shape {
    protected String color;
    protected String name;

    public Shape(String name, String color) {
        this.name = name;
        this.color = color;
    }

    // Abstract method (optional here, but good practice for inheritance)
    // public abstract void displayInfo();
}

// Concrete Class 1: Circle
class Circle extends Shape implements Measurable {
    private double radius;

    public Circle(double radius, String color) {
        super("Circle", color); // Constructor Chaining
        this.radius = radius;
    }

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }

    @Override
    public double perimeter() {
        return 2 * Math.PI * radius; // Circumference
    }

    public void displayResults() {
        System.out.printf("--- %s (Color: %s) ---\n", name, color);
        System.out.printf("Area: %.2f\n", area());
        System.out.printf("Perimeter (Circumference): %.2f\n", perimeter());
    }
}

// Concrete Class 2: Rectangle
class Rectangle extends Shape implements Measurable {
    private double length;
    private double width;

    public Rectangle(double length, double width, String color) {
        super("Rectangle", color); // Constructor Chaining
        this.length = length;
        this.width = width;
    }

    @Override
    public double area() {
        return length * width;
    }

    @Override
    public double perimeter() {
        return 2 * (length + width);
    }

    public void displayResults() {
        System.out.printf("--- %s (Color: %s) ---\n", name, color);
        System.out.printf("Area: %.2f\n", area());
        System.out.printf("Perimeter: %.2f\n", perimeter());
    }
}

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

        // Input for Circle
        System.out.println("Enter radius for Circle:");
        double r = scanner.nextDouble();

        // Input for Rectangle
        System.out.println("Enter length for Rectangle:");
        double l = scanner.nextDouble();
        System.out.println("Enter width for Rectangle:");
        double w = scanner.nextDouble();

        scanner.close();

        // Create objects and display results
        System.out.println("\n*** Shape Calculations ***");

        Circle circle = new Circle(r, "Blue");
        circle.displayResults();

        Rectangle rect = new Rectangle(l, w, "Green");
        rect.displayResults();
    }
}
```

---

## Question 2: Swing-based Mini Calculator

This program creates a basic Swing GUI with two text fields for input numbers and buttons for four basic **Arithmetic Operations**. It includes **basic validation** and displays the result in a **JLabel**.

Answer-

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class MiniCalculatorGUI extends JFrame implements ActionListener {
    private JTextField num1Field, num2Field;
    private JLabel resultLabel;
    private JButton addButton, subButton, mulButton, divButton;

    public MiniCalculatorGUI() {
        // Frame setup
        setTitle("Mini Calculator");
        setSize(300, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout()); // Simple layout

        // Input Fields
        num1Field = new JTextField(10);
        num2Field = new JTextField(10);

        // Result Label
        resultLabel = new JLabel("Result: ");
        resultLabel.setPreferredSize(new Dimension(280, 20)); // Fixed size for label

        // Operation Buttons
        addButton = new JButton("+");
        subButton = new JButton("-");
        mulButton = new JButton("*");
        divButton = new JButton("/");

        // Register Action Listeners
        addButton.addActionListener(this);
        subButton.addActionListener(this);
        mulButton.addActionListener(this);
        divButton.addActionListener(this);

        // Add Components
        add(new JLabel("Number 1:"));
        add(num1Field);
        add(new JLabel("Number 2:"));
        add(num2Field);

        JPanel buttonPanel = new JPanel(new GridLayout(1, 4,
5, 5));
        buttonPanel.add(addButton);
        buttonPanel.add(subButton);
        buttonPanel.add(mulButton);
        buttonPanel.add(divButton);
        add(buttonPanel);

        add(resultLabel);

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        calculate(command);
    }

    private void calculate(String operator) {
        try {
            // Basic Validation & Type Conversion
            double num1 =
Double.parseDouble(num1Field.getText());
            double num2 =
Double.parseDouble(num2Field.getText());
            double result = 0;

            switch (operator) {
                case "+":
                    result = num1 + num2;
                    break;
                case "-":
```

```
            result = num1 - num2;
            break;
        case "*":
            result = num1 * num2;
            break;
        case "/":
            // Division by zero check
            if (num2 == 0) {
                resultLabel.setText("Error: Division by zero!");
                return;
            }
            result = num1 / num2;
            break;
    }

    // Display Result
    resultLabel.setText(String.format("Result: %.2f",
result));
```

```
        } catch (NumberFormatException ex) {
            resultLabel.setText("Error: Enter valid numbers.");
            JOptionPane.showMessageDialog(this, "Please enter
valid numeric values.", "Input Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new
MiniCalculatorGUI());
    }
}
```

## Question 3: String Transformation and Analysis

This program takes a string and performs three tasks using **loops** and **conditional statements**:

1. Converts **alternate characters** to **uppercase**.
2. Counts **vowels and consonants**.
3. **Reverses** the original string.

Answer-

```
import java.util.Scanner;

public class StringTransformer {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string:");
        String originalString = scanner.nextLine();
        scanner.close();

        // 1. Convert alternate characters to uppercase
        String transformedString =
transformString(originalString);

        // 2. Count vowels and consonants
        countVowelsAndConsonants(originalString);

        // 3. Display reversed string
        String reversedString = reverseString(originalString);

        // Output Format
        System.out.println("\n--- String Analysis Results ---");
        System.out.println("Transformed String (Alternate
Uppercase): " + transformedString);
        // Vowel/Consonant count is printed inside its method
for simplicity
        System.out.println("Reversed String: " +
reversedString);
    }

    /**
     * Converts characters at odd indices (1, 3, 5, ...) to
uppercase.
     */
    private static String transformString(String str) {
        StringBuilder sb = new StringBuilder(str.toLowerCase());
        // Loop through the string
        for (int i = 0; i < sb.length(); i++) {
            // Conditional Statement: Check for alternate index
(0-based: 1, 3, 5 are alternate)
            if (i % 2 != 0) {
                char upperChar =
Character.toUpperCase(sb.charAt(i));
                sb.setCharAt(i, upperChar);
            }
        }
        return sb.toString();
    }

    /**
     * Counts and displays the number of vowels and
consonants.
     */
    private static void countVowelsAndConsonants(String str)
{
        int vowelCount = 0;
        int consonantCount = 0;

        // Clean the string: convert to lower case and remove spaces/punctuation
        String cleanStr = str.toLowerCase().replaceAll("[^a-z]", "");

        // Loop through the cleaned string
        for (int i = 0; i < cleanStr.length(); i++) {
            char ch = cleanStr.charAt(i);

            // Conditional Statement: Check if it's a vowel
            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                vowelCount++;
            } else {
                // If it's a letter but not a vowel, it's a consonant
                consonantCount++;
            }
        }

        System.out.println("Vowel Count: " + vowelCount);
        System.out.println("Consonant Count: " + consonantCount);
    }

    /**
     * Reverses the string using the StringBuilder's reverse method.
     */
    private static String reverseString(String str) {
        return new StringBuilder(str).reverse().toString();
    }
}
```