Java Practical Programs with Code, Output, and Explanation

Practical 1: Sum and Average Using Loops

Description:

This program calculates the sum and average of numbers using a loop.

Code:

```
public class SumAverage {
    public static void main(String[] args) {
        int sum = 0;
        int n = 5;
        for (int i = 1; i <= n; i++) {
            sum += i;
        }
        double avg = sum / (double) n;
        System.out.println("Sum: " + sum);
        System.out.println("Average: " + avg);
    }
}
```

Output:

Output:

Sum: 15

Average: 3.0

Practical 2: Basic Calculator using switch-case

Description:

A calculator performing basic operations using switch-case.

Code:

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        int a = sc.nextInt();
        System.out.print("Enter second number: ");
        int b = sc.nextInt();
        System.out.print("Enter operation (+, -, *, /): ");
        char op = sc.next().charAt(0);
        switch(op) {
            case '+': System.out.println("Sum: " + (a + b)); break;
            case '-': System.out.println("Diff: " + (a - b)); break;
            case '*': System.out.println("Product: " + (a * b)); break;
             case '/': System.out.println("Quotient: " + (b != 0 ? (a / b) : "Division by
zero")); break;
            default: System.out.println("Invalid operator");
```

```
        }
    }
}
```

Output:

Output:

Enter first number: 10

Enter second number: 5

Enter operation: +

Sum: 15

Practical 3: Student Class with Attributes

Description:

Creates a student class with name, roll number and marks.

Code:

```
class Student {
    String name;
    int roll;
    float marks;

    Student(String name, int roll, float marks) {
        this.name = name;
        this.roll = roll;
        this.marks = marks;
    }

    void display() {
        System.out.println("Name: " + name + ", Roll: " + roll + ", Marks: " + marks);
    }

    public static void main(String[] args) {
        Student s = new Student("Ravi", 101, 89.5f);
        s.display();
    }
}
```

Output:

Output:

Name: Ravi, Roll: 101, Marks: 89.5

Practical 4: Method Overloading and this Keyword

Description:

Java Practical Programs with Code, Output, and Explanation

This program demonstrates method overloading and use of 'this' to refer to current object.

Code:

```
class Demo {
    int a, b;

    Demo(int a, int b) {
        this.a = a;
        this.b = b;
    }

    void show() {
        System.out.println("a: " + a + ", b: " + b);
    }

    void show(String msg) {
        System.out.println(msg + " a: " + a + ", b: " + b);
    }

    public static void main(String[] args) {
        Demo d = new Demo(10, 20);
        d.show();
        d.show("Values ->");
    }
}
```

Output:

Output:

a: 10, b: 20

Values -> a: 10, b: 20

Practical 5: Single and Multilevel Inheritance

Description:

Shows inheritance in Java with single and multilevel example.

Code:

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}
```

Java Practical Programs with Code, Output, and Explanation

```
class Puppy extends Dog {
    void weep() {
        System.out.println("The puppy weeps.");
    }

    public static void main(String[] args) {
        Puppy p = new Puppy();
        p.eat();
        p.bark();
        p.weep();
    }
}
```

Output:

Output:

This animal eats food.

The dog barks.

The puppy weeps.

Practical 6: Interface for Area Calculation

Description:

Uses interface to calculate area of different shapes.

Code:

```
interface Shape {
    double area();
}

class Circle implements Shape {
    double r;
    Circle(double r) { this.r = r; }
    public double area() { return Math.PI * r * r; }
}

class Rectangle implements Shape {
    double l, b;
    Rectangle(double l, double b) { this.l = l; this.b = b; }
    public double area() { return l * b; }
}

public class AreaDemo {
    public static void main(String[] args) {
        Shape c = new Circle(5);
        Shape r = new Rectangle(4, 6);
        System.out.println("Circle area: " + c.area());
        System.out.println("Rectangle area: " + r.area());
    }
```

```
}
```

Output:

Output:

Circle area: 78.5398...

Rectangle area: 24.0

Practical 7: Exception Handling: Division by Zero & Invalid Input

Description:

Handles exceptions like divide-by-zero and input mismatch.

Code:

```
import java.util.*;

public class ExceptionDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter numerator: ");
            int num = sc.nextInt();
            System.out.print("Enter denominator: ");
            int den = sc.nextInt();
            System.out.println("Result: " + (num / den));
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero.");
        } catch (InputMismatchException e) {
            System.out.println("Invalid input.");
        } finally {
            System.out.println("Program ended.");
        }
    }
}
```

Output:

Output:

Enter numerator: 10

Enter denominator: 0

Cannot divide by zero.

Program ended.

Practical 8: Banking System with Exception Handling

Description:

Java Practical Programs with Code, Output, and Explanation

Implements basic banking operations with exception handling.

Code:

```
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

class BankAccount {
    double balance = 1000;

    void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance)
            throw new InsufficientFundsException("Insufficient balance");
        balance -= amount;
        System.out.println("Withdrawal successful. Remaining: " + balance);
    }
}

public class Bank {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount();
        try {
            acc.withdraw(1500);
        } catch (InsufficientFundsException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

Output:

Insufficient balance

Practical 9: Applet to Display Welcome and Change Background

Description:

Simple Java Applet displaying a welcome message and changing background color.

Code:

```
import java.applet.*;
import java.awt.*;

// <applet code="WelcomeApplet.class" width=300 height=200></applet>

public class WelcomeApplet extends Applet {
    public void init() {
        setBackground(Color.cyan);
```

```
    }
    public void paint(Graphics g) {
        g.drawString("Welcome to Java Applet!", 50, 100);
    }
}
```

Output:

Output:

Applet window with cyan background and welcome text.

Practical 10: Swing-based GUI Calculator

Description:

Java Swing calculator with text fields and buttons.

Code:

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

public class GUICalc {
    public static void main(String[] args) {
        JFrame f = new JFrame("Calculator");
        JTextField t1 = new JTextField(); t1.setBounds(50, 50, 150, 20);
        JTextField t2 = new JTextField(); t2.setBounds(50, 80, 150, 20);
        JButton add = new JButton("+"); add.setBounds(50, 110, 50, 30);
        JLabel result = new JLabel("Result"); result.setBounds(50, 150, 200, 20);

        add.addActionListener(e -> {
            int a = Integer.parseInt(t1.getText());
            int b = Integer.parseInt(t2.getText());
            result.setText("Sum = " + (a + b));
        });

        f.add(t1); f.add(t2); f.add(add); f.add(result);
        f.setSize(300, 300); f.setLayout(null); f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Output:

Output:

GUI window with 2 inputs, button '+', and label showing result.

Practical 11: Singleton Design Pattern

Description:

Ensures only one instance of a class exists.

Java Practical Programs with Code, Output, and Explanation

Code:

```
class Singleton {
    private static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
}

public class SingletonDemo {
    public static void main(String[] args) {
        Singleton s1 = Singleton.getInstance();
        Singleton s2 = Singleton.getInstance();
        System.out.println(s1 == s2);  // true
    }
}
```

Output:

Output:

true (same instance used)

Practical 12: Multithreading with Synchronization

Description:

Creates threads to perform tasks and demonstrates synchronization.

Code:

```
class Task extends Thread {
    String task;
    Task(String task) { this.task = task; }

    public synchronized void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(task + " - " + i);
            try { Thread.sleep(100); } catch (Exception e) {}
        }
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        Task t1 = new Task("Reading");
        Task t2 = new Task("Calculating");
        t1.start();
        t2.start();
```

```
    }
}
```

Output:

Output:

Reading - 1...

Calculating - 1... (interleaved output from two threads)