CodInClub

Powered By BridgeLabz

Before diving into the concepts of **classes** and **objects**, it's essential to understand why Object-Oriented Programming (OOP) is preferred over Procedural Programming in modern software development.

1. Procedural Programming Overview

Procedural Programming is a programming paradigm that relies on a structured sequence of instructions, organized into procedures, functions, or routines to perform tasks. It follows a top-down approach and emphasizes a clear sequence of commands and control flow.

- Procedural programming (PP) follows a **step-by-step approach**, where programs are divided into functions.
- Examples: C, Pascal.
- Focuses on **functions** or **procedures** to perform operations.

Sample Program

Write a Java program to calculate the total and average marks of a student based on marks in three subjects using a **procedural programming approach**.

```
import java.util.Scanner;
public class StudentMarks {
    // Function to calculate total marks
    public static int calculateTotal(int marks1, int marks2, int
marks3) {
        return marks1 + marks2 + marks3;
    }

    // Function to calculate average marks
    public static double calculateAverage(int total, int
numberOfSubjects) {
        return (double) total / numberOfSubjects;
    }

    // Function to display results
    public static void displayResults(String name, int total, double
average) {
        System.out.println("Student Name: " + name);
```

```
        System.out.println("Total Marks: " + total);
        System.out.println("Average Marks: " + average);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input student name
        System.out.print("Enter the student's name: ");
        String name = scanner.nextLine();

        // Input marks for three subjects
        System.out.print("Enter marks for Subject 1: ");
        int marks1 = scanner.nextInt();
        System.out.print("Enter marks for Subject 2: ");
        int marks2 = scanner.nextInt();
        System.out.print("Enter marks for Subject 3: ");
        int marks3 = scanner.nextInt();

        // Calculate total and average
        int total = calculateTotal(marks1, marks2, marks3);
        double average = calculateAverage(total, 3);

        // Display results
        displayResults(name, total, average);

        scanner.close();
    }
}
```

Output

Input:

```
Enter the student's name: Alice

Enter marks for Subject 1: 85
```

```
Enter marks for Subject 2: 90

Enter marks for Subject 3: 88
```

Output:

```
Student Name: Alice

Total Marks: 263

Average Marks: 87.66666666666667
```

Explanation

1. **Functions**:
   - `calculateTotal(int marks1, int marks2, int marks3)`: Computes the total marks.
   - `calculateAverage(int total, int numberOfSubjects)`: Computes the average marks.
   - `displayResults(String name, int total, double average)`: Displays the results.
2. **Procedural Nature**:
   - The program is divided into **functions** that focus on specific tasks.
   - Data (`name`, `marks1`, `marks2`, `marks3`) is passed between functions as parameters.
3. **Global Flow**:
   - The `main()` method organizes the flow of the program sequentially:
     - Input → Calculation → Output.

Key Features of Procedural Programming in This Example

1. **Functions**: Used to perform specific tasks like calculations and display.
2. **Data Flow**: Data is passed explicitly between functions via parameters.
3. **Linear Execution**: The program executes in a step-by-step manner.

Limitations of Procedural Programming

1. **Data Handling**: Data and behavior are separate, making the code less modular.
2. **Reusability**: Functions operate on data provided to them and cannot encapsulate related data and behavior.
3. **Scalability**: Difficult to maintain and extend for large, complex systems.

Comparison with Object-Oriented Approach

If this problem were solved using OOP, the **data (attributes)** and **behavior (methods)** would be encapsulated in a `Student` class. This modular approach would make the code more reusable and scalable.

2. Object-oriented Programming (OOP) Overview

An **Object-Oriented Programming (OOP) Language** is a programming paradigm based on the concept of **"objects"**, which are instances of **classes**. It allows developers to model real-world entities and solve problems by organizing data (attributes) and behaviors (methods) into objects.

The OOP paradigm organizes code into **classes** and **objects**, focusing on data (attributes) and behaviors (methods).

Advantages of OOP Over Procedural Programming

1. **Encapsulation**
   - Combines data and methods into a single unit (class).
   - Restricts direct access to certain components, ensuring data security.
   - Example: Sensitive data in a banking app, like an account balance, is accessible only through specific methods.
2. **Abstraction**
   - Hides complex implementation details, exposing only essential features.
   - Example: A car's `drive()` method abstracts engine mechanics.
3. **Reusability Through Inheritance**
   - Enables classes to inherit properties and behaviors from other classes.
   - Example: A `Car` class can inherit features from a `Vehicle` class.
4. **Polymorphism**
   - Allows objects to take multiple forms, enabling method overloading and overriding.

- Example: A method `draw()` can behave differently for `Circle` and `Rectangle`.

5. **Natural Modeling of Real-World Entities**
   - OOP maps objects in the real world to programming constructs.
   - Example: An employee can be represented as a class with attributes like `name`, `salary` and methods like `work()`.
6. **Modularity and Maintainability**
   - Divides the program into small, manageable pieces (classes).
   - Example: A food delivery app can have classes like `Restaurant`, `Order`, and `Customer`.

Comparison Between Procedural Programming and OOP

Feature	Procedural Programming	Object-Oriented Programming
Focus	Functions and procedures.	Classes and objects.
Data Security	Global data is prone to accidental modification.	Encapsulated, secure data access.
Reusability	Minimal reusability, and code duplication.	Reusable code through inheritance and polymorphism.
Scalability	Poor scalability for large systems.	Highly scalable and modular.
Real-World Mapping	Difficult to map real-world entities.	Naturally models real-world objects.
Maintenance	Difficult to maintain in large projects.	Easier to maintain with a modular design.