

Java Object-Oriented Programming (OOP) – Complete Guide with Comments

This file is a **full beginner-to-strong-intermediate guide** to Java OOP. Every code snippet now includes detailed **comments explaining each line**.

1. Why Object-Oriented Programming Exists

Before OOP, programs were written as long lists of instructions (procedural programming). As software grew, code became:

- Hard to maintain
- Hard to reuse
- Easy to break

OOP solves this by modeling software the same way we think about the real world — using objects.

An object:

- Has **state** (data)
- Has **behavior** (actions)

2. Classes and Objects

Example

```
// Define a class called Student
class Student {
    String name; // Field to store the student's name
    int age;     // Field to store the student's age

    // Method representing behavior
    void study() {
        System.out.println(name + " is
studying"); // Prints a message including the student's name
    }
}

// Main method to run code
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(); // Create a new Student object
        s1.name = "Alex";          // Set the student's name
        s1.age = 20;               // Set the student's age
        s1.study();                // Call the study method
    }
}
```

3. Calculator Example

A simple calculator class demonstrating **OOP concepts** like methods, fields, encapsulation, and usage.

```
// Calculator class
class Calculator {
    private double result; // Stores the last calculation result

    // Adds two numbers
    public double add(double a, double b) {
        result = a + b;
        return result; // Return the sum
    }

    // Subtracts second number from first
    public double subtract(double a, double b) {
        result = a - b;
        return result; // Return the difference
    }

    // Multiplies two numbers
    public double multiply(double a, double b) {
        result = a * b;
        return result; // Return the product
    }

    // Divides first number by second
    public double divide(double a, double b) {
        if (b != 0) {
            result = a / b;
            return result; // Return the quotient
        } else {
            System.out.println("Error: Division by zero");
            return 0;
        }
    }

    // Returns the last result
    public double getResult() {
        return result;
    }
}

// Main class to test Calculator
public class Main {
```

```
public static void main(String[] args) {  
    Calculator calc = new Calculator(); // Create Calculator object  
  
    System.out.println("Add: " + calc.add(10, 5)); // 15  
    System.out.println("Subtract: " + calc.subtract(10, 5)); // 5  
    System.out.println("Multiply: " + calc.multiply(10, 5)); // 50  
    System.out.println("Divide: " + calc.divide(10, 5)); // 2  
  
    System.out.println("Last result: " + calc.getResult()); // 2 (from last  
operation)  
}  
}
```

Explanation: - `Calculator` class encapsulates the calculation logic. - `result` field stores last calculation result (encapsulation example). - Methods perform individual operations. - `Main` class demonstrates **object creation** and calling methods. - `divide` checks for division by zero (safety).
