**Java OOP Made Easy**

**Java OOP Basics**

Object-Oriented Programming (OOP) in Java revolves around four key principles:

1. **Encapsulation** – Wrapping data (variables) and code (methods) into a single unit (class).
2. **Abstraction** – Hiding complex implementation details and exposing only necessary parts.
3. **Inheritance** – One class can inherit properties and behaviors from another.
4. **Polymorphism** – A method or object can take many forms (method overloading and method overriding).

**1. Classes and Objects**

A **class** is like a blueprint, and an **object** is an instance of that class.

**Example: Creating a Class and an Object**

// Define a class

class Car {

String brand; // Attribute

int speed; // Attribute

// Method to display car details

void displayInfo() {

System.out.println("Brand: " + brand + ", Speed: " + speed);

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car(); // Create an object

myCar.brand = "Toyota";

myCar.speed = 120;

myCar.displayInfo(); // Output: Brand: Toyota, Speed: 120

}

}

**2. Encapsulation (Data Hiding)**

Encapsulation means using **private variables** and providing **public getter/setter methods**.

**Example: Using Getters and Setters**

class Person {

private String name; // Private variable

// Setter method

public void setName(String newName) {

name = newName;

}

// Getter method

public String getName() {

return name;

}

}

public class Main {

public static void main(String[] args) {

Person p = new Person();

p.setName("John"); // Set name

System.out.println(p.getName()); // Get name -> Output: John

}

}

**3. Inheritance (Extending a Class)**

Inheritance lets one class acquire properties and behaviors of another.

**Example: Child Class Inheriting Parent Class**

// Parent class

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

// Child class (inherits Animal)

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.makeSound(); // Inherited method

myDog.bark(); // Child class method

}

}

**4. Polymorphism (One Method, Different Behaviors)**

Polymorphism allows methods to behave differently based on the object.

**Example 1: Method Overriding (Changing Behavior in Child Class)**

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Cat extends Animal {

@Override

void makeSound() {

System.out.println("Cat meows");

}

}

public class Main {

public static void main(String[] args) {

Animal myAnimal = new Cat();

myAnimal.makeSound(); // Output: Cat meows

}

}

**Example 2: Method Overloading (Same Method, Different Parameters)**

class MathOperations {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

MathOperations math = new MathOperations();

System.out.println(math.add(2, 3)); // Output: 5

System.out.println(math.add(2.5, 3.5)); // Output: 6.0

}

}

**5. Abstraction (Hiding Details)**

Abstraction is achieved using **abstract classes** and **interfaces**.

**Example: Abstract Class in Java**

// Abstract class

abstract class Animal {

// Abstract method (no implementation)

abstract void makeSound();

// Concrete method (with implementation)

void sleep() {

System.out.println("Sleeping...");

}

}

// Child class (must implement abstract method)

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.makeSound(); // Output: Dog barks

myDog.sleep(); // Output: Sleeping...

}

}

**Example: Interface in Java**

// Define an interface

interface Vehicle {

void start(); // Abstract method (no implementation)

}

// Implement the interface in a class

class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car is starting...");

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.start(); // Output: Car is starting...

}

}

**Key Differences: Abstract Class vs. Interface**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| Methods | Can have abstract & concrete methods | Only abstract methods (before Java 8) |
| Variables | Can have instance variables | Only constants (final static) |
| Constructor | Can have a constructor | No constructor |
| Multiple Inheritance | Not allowed | Allowed (can implement multiple interfaces) |

**Summary**

1. **Encapsulation** → Use private variables and public getter/setter methods.
2. **Abstraction** → Use abstract classes or interfaces.
3. **Inheritance** → Use extends to make a child class inherit from a parent class.
4. **Polymorphism** → Use method overloading and method overriding.