Introduction to Object-Oriented Programming (OOP)

Contents

1 What is a Programming Paradigm?

A **programming paradigm** is a way or style of thinking about writing programs. It provides a model or framework that tells us how to design and organize code.

Examples of paradigms include:

- **Procedural programming**: Organizing code as a sequence of instructions and functions.
- **Functional programming**: Writing programs using pure functions, avoiding shared state.
- **Object-Oriented Programming (OOP)**: Organizing code around objects that contain both data and behavior.

—

2 What is Object-Oriented Programming?

Object-Oriented Programming, often called **OOP**, is a popular paradigm that organizes software into:

- **Objects** – which represent real-world entities (like cars, users, books)
- **Classes** – blueprints to create objects

In OOP, instead of writing code as one long list of instructions, we build small self-contained pieces called objects, which:

- Store data (attributes)
- Provide behavior (methods/functions)

—

3 Why use OOP?

OOP helps us:

- Model real-world systems more naturally
- Reuse and organize code easily
- Make programs easier to maintain and extend

—

4 Four Core Principles of OOP

OOP is based on four main ideas, often called the **four pillars**:

1. **Encapsulation**
2. **Abstraction**
3. **Inheritance**
4. **Polymorphism**

Let's learn what each means.

—

5 Encapsulation

Encapsulation means bundling data and methods together inside a class, and restricting direct access to some parts of the object.

This helps protect data from accidental modification.

Example in Python

```
class Person:
    def __init__(self, name):
        self.__name = name  # private attribute

    def get_name(self):
        return self.__name

p = Person("Alice")
print(p.get_name())  # Output: Alice
```

Example in Java

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public static void main(String[] args) {
        Person p = new Person("Alice");
        System.out.println(p.getName()); // Output: Alice
    }
}
```

—

6 Abstraction

Abstraction means showing only the essential features of an object, while hiding the complex details.

For example, when we drive a car, we just use the steering wheel and pedals, without worrying about how the engine works.

Example in Python (using abstract class)

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        print("Woof!")
```

```
d = Dog()
d.make_sound()  # Output: Woof!
```

Example in Java

```
abstract class Animal {
    abstract void makeSound();
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.makeSound(); // Output: Woof!
    }
}
```

—

7 Inheritance

Inheritance lets a class (child) reuse the code of another class (parent). This helps avoid duplication and makes it easier to extend code.

Example in Python

```
class Vehicle:
    def move(self):
        print("Vehicle is moving")

class Car(Vehicle):
    def honk(self):
        print("Beep beep!")

c = Car()
c.move()   # Output: Vehicle is moving
c.honk()   # Output: Beep beep!
```

Example in Java

```
class Vehicle {
    void move() {
        System.out.println("Vehicle␣is␣moving");
    }
}

class Car extends Vehicle {
    void honk() {
        System.out.println("Beep␣beep!");
    }
}

public class Main {
    public static void main(String[] args) {
        Car c = new Car();
        c.move(); // Output: Vehicle is moving
        c.honk(); // Output: Beep beep!
    }
}
```

—

8 Polymorphism

Polymorphism means ”many forms”: the same method name can behave differently based on the object.

Example in Python

```
class Bird:
    def make_sound(self):
        print("Chirp")

class Duck(Bird):
    def make_sound(self):
        print("Quack")

animals = [Bird(), Duck()]
for a in animals:
    a.make_sound()
# Output:
# Chirp
# Quack
```

Example in Java

```java
class Bird {
    void makeSound() {
        System.out.println("Chirp");
    }
}

class Duck extends Bird {
    void makeSound() {
        System.out.println("Quack");
    }
}

public class Main {
    public static void main(String[] args) {
        Bird[] animals = { new Bird(), new Duck() };
        for (Bird a : animals) {
            a.makeSound();
        }
    }
}
// Output:
// Chirp
// Quack
```

—

9 Summary

- OOP is a way of organizing code using objects and classes.
- It helps make code easier to read, reuse, and maintain.
- The four core pillars of OOP are: Encapsulation, Abstraction, Inheritance, Polymorphism.

10 Learn More

- Python OOP tutorial: `https://docs.python.org/3/tutorial/classes.html`
- Java OOP concepts: `https://docs.oracle.com/javase/tutorial/java/concepts/`