BÁO CÁO

Họ và tên: Nguyễn Hữu Minh

Contents

Khái niệm lập trình hướng đối tượng

Lập trình hướng đối tượng (OOP) là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code để trừu tượng hóa các đối tượng thực tế trong cuộc sống.

Các tính chất của lập trình hướng đối tượng

1. Tính đóng gói

- **Khái niệm:** Là tính chất cho phép ẩn thông tin bên trong đối tượng đối với bên ngoài.
- **Mục đích:** Bảo vệ dữ liệu. Các đối tượng khác chỉ có thể truy cập thuộc tính thông qua các phương thức công khai.

- **Thể hiện trong Java:**

```
public class Circle {
    private int radius;
    private float area;

    public Circle(float radius) {
        this.radius = radius;
        this.area = radius * radius * 3.14f;
    }

    public float getArea() {
        return area;
    }
}
```

2. Tính kế thừa

- **Khái niệm:** Cho phép một đối tượng kế thừa lại những gì mà một đối tượng khác đã có.
- **Mục đích:** Tránh trùng lặp mã nguồn, dễ bảo trì.
- **Thể hiện trong Java:**

```
public class Rectangle {
    private int height, width;

    public Rectangle(int height, int width) {
        this.height = height;
        this.width = width;
    }
}

public class Square extends Rectangle {
    public Square(int side) {
        super(side, side);
    }
}
```

3. Tính đa hình

- **Khái niệm:** Cho phép một đối tượng thuộc nhiều lớp khác nhau thực hiện phương thức theo nhiều cách khác nhau.
- **Mục đích:**
  - Tăng tính linh hoạt.
  - Giảm trùng lặp.
- **Thể hiện trong Java:**
  - **Đa hình tĩnh (compile):**

```
public int getSum(int a, int b) {
    return a + b;
}

public int getSum(int a, int b, int c) {
    return a + b + c;
```

```
}
```

- **Đa hình động (run-time):**

```
public class Animal {
    public void say() {
        System.out.println("...");
    }
}

public class Cat extends Animal {
    @Override
    public void say() {
        System.out.println("Meo meo");
    }
}

public class Dog extends Animal {
    @Override
    public void say() {
        System.out.println("Gau gau");
    }
}

Animal animal = new Animal();
Animal cat = new Cat();
Animal dog = new Dog();

animal.say(); // in ra "..."
cat.say(); // in ra "Meo meo"
dog.say(); // in ra "Gau gau"
```

4. Tính trừu tượng

- **Khái niệm:** Cho phép ẩn đi các chi tiết không cần thiết, chỉ tập trung vào những chi tiết cần thiết.
- **Mục đích:**
  - Giảm độ phức tạp.
  - Khuyến khích tái sử dụng code.
- **Thể hiện trong Java:**
  - **Interface:**

```
public interface Shape {
    float getArea();
}

public class Circle implements Shape {
    @Override
    public float getArea() {
        return ...; // tính diện tích
    }
}
```

  - **Abstract class (tương tự)**

Nguyên lý SOLID

1. Single Responsibility Principle

- **Khái niệm:** Một class chỉ nên có một trách nhiệm duy nhất.
- **Mục đích:** Tránh việc một class có nhiều trách nhiệm trở nên cồng kềnh.
- **Thể hiện trong Java Spring:**
  - Lớp `OncePerRequestFilter` chỉ có một nhiệm vụ duy nhất, là đảm bảo filter chạy đúng một lần cho mỗi request.

2. Open/Closed Principle

- **Khái niệm:** Có thể mở rộng một class nhưng hạn chế sửa đổi.
- **Mục đích:** Hạn chế sửa đổi code trước đó.
- **Thể hiện trong Java Core:**
  - Lớp `List` và hai lớp `LinkedList`, `ArrayList` được mở rộng từ lớp `List`.

3. Liskov Substitution Principle

- **Khái niệm:** Bất kỳ instance nào của class cha đều có thể thay thế được instance của class con.
- **Mục đích:** Tránh lỗi khi kế thừa.
- **Thể hiện trong Java Spring:**
  - Lớp `OncePerRequestFilter` có thể thay thế bởi các lớp con mà không làm phá vỡ chương trình.

4. Interface Segregation Principle

- **Khái niệm:** Tách thành nhiều interface nhỏ thay vì một interface lớn.
- **Mục đích:** Tránh việc client phải implement các method không cần thiết.
- **Thể hiện trong Java Spring:**
  - Lớp `GenericFilterBean` triển khai nhiều interface như `Filter`, `BeanNameAware`,...

5. Dependency Inversion Principle

- **Khái niệm:** Các module cấp cao không nên phụ thuộc vào các module cấp thấp, cả hai nên phụ thuộc vào abstract class/interface.

- **Mục đích:** Giúp các module độc lập với nhau.
- **Thể hiện trong Java Core:**
  - Các lớp filter trong Spring đều triển khai dựa vào interface `Filter`.