1. Tính chất quan trọng của hướng đối tượng

Hướng đối tượng (OOP) có các tính chất quan trọng sau:

- \*\*Encapsulation (Đóng gói)\*\*: Bảo vệ dữ liệu bằng cách ẩn các chi tiết thực hiện và chỉ cung cấp giao diện để tương tác. Dữ liệu được bảo vệ thông qua các access modifier.

- \*\*Inheritance (Kế thừa)\*\*: Cho phép một lớp (class) kế thừa các thuộc tính và phương thức từ lớp khác. Điều này giúp tái sử dụng mã và xây dựng cấu trúc phân cấp.

- \*\*Polymorphism (Đa hình)\*\*: Cho phép các phương thức cùng tên hoạt động khác nhau dựa trên đối tượng gọi chúng. Có hai loại đa hình: compile-time (method overloading) và runtime (method overriding).

- \*\*Abstraction (Trừu tượng)\*\*: Giúp làm rõ các khái niệm bằng cách chỉ hiển thị các thông tin cần thiết và ẩn đi các chi tiết không cần thiết.

2. Access modifier trong Java

Java có bốn loại access modifier:

- \*\*public\*\*: Có thể truy cập từ mọi lớp.

- \*\*protected\*\*: Có thể truy cập từ lớp hiện tại, lớp con, và các lớp trong cùng gói.

- \*\*default\*\* (không khai báo gì): Chỉ có thể truy cập từ các lớp trong cùng gói.

- \*\*private\*\*: Chỉ có thể truy cập từ lớp hiện tại.

3. Phân biệt class và instance

- \*\*Class\*\*: Là một bản thiết kế cho các đối tượng, định nghĩa thuộc tính và phương thức chung.

- \*\*Instance\*\*: Là một thể hiện cụ thể của một lớp. Khi bạn tạo một đối tượng từ lớp, bạn đang tạo một instance.

4. Phân biệt Abstract và Interface

- \*\*Abstract Class\*\*:

- Có thể có cả phương thức đã được cài đặt và chưa được cài đặt.

- Có thể có trạng thái (thuộc tính).

- Chỉ có thể kế thừa một lần (single inheritance).

- \*\*Interface\*\*:

- Chỉ chứa các phương thức chưa được cài đặt (từ Java 8 trở đi có thể có default method).

- Không có trạng thái.

- Một lớp có thể triển khai nhiều interface (multiple inheritance).

Trường hợp sử dụng cụ thể:

- Sử dụng \*\*abstract class\*\* khi bạn cần chia sẻ mã giữa các lớp nhưng vẫn muốn hạn chế cách mà các lớp con triển khai.

- Sử dụng \*\*interface\*\* khi bạn cần định nghĩa một bộ hành vi mà nhiều lớp có thể thực hiện mà không cần liên quan đến nhau.

Xử lý phương thức cùng tên:

- Nếu hai interface hoặc một abstract và một interface có phương thức cùng tên, một lớp có thể kế thừa từ chúng mà không gặp vấn đề gì, miễn là kiểu trả về tương thích. Tuy nhiên, nếu không tương thích, sẽ có lỗi biên dịch.

5. Overriding và Overloading

- \*\*Overriding\*\*: Là việc định nghĩa lại một phương thức trong lớp con có cùng tên và tham số với phương thức trong lớp cha. Điều này xảy ra khi bạn muốn thay đổi hành vi của phương thức kế thừa.

- \*\*Overloading\*\*: Là việc định nghĩa nhiều phương thức có cùng tên nhưng khác tham số trong cùng một lớp. Điều này xảy ra khi bạn muốn có nhiều cách để gọi một phương thức với các tham số khác nhau.

6. Overriding với Access Modifier là private hoặc static

- \*\*private\*\*: Không thể overriding vì phương thức private không thể truy cập từ lớp con.

- \*\*static\*\*: Không thể overriding, vì static phương thức được liên kết với lớp thay vì với instance.

7. Phương thức final có thể kế thừa không?

Phương thức `final` không thể bị overriding, nhưng có thể được kế thừa. Lớp con có thể sử dụng phương thức `final` nhưng không thể thay đổi nó.

8. Phân biệt từ khóa `this` và `super`

- \*\*this\*\*: Được sử dụng để tham chiếu đến đối tượng hiện tại của lớp. Nó thường được sử dụng để phân biệt giữa biến instance và biến tham số.

- \*\*super\*\*: Được sử dụng để tham chiếu đến lớp cha. Nó thường được dùng để gọi phương thức hoặc constructor của lớp cha.

Demo Code

Dưới đây là đoạn mã Java minh họa cho các khái niệm trên:

// Abstract class and Interface example

abstract class Animal {

abstract void sound(); // Abstract method

}

interface Pet {

void play(); // Interface method

}

class Dog extends Animal implements Pet {

@Override

void sound() {

System.out.println("Woof");

}

@Override

public void play() {

System.out.println("Dog is playing");

}

}

// Overriding and Overloading example

class MathUtils {

void add(int a, int b) { // Method Overloading

System.out.println("Sum: " + (a + b));

}

void add(double a, double b) {

System.out.println("Sum: " + (a + b));

}

}

class Base {

void display() {

System.out.println("Display from Base");

}

}

class Derived extends Base {

@Override

void display() { // Method Overriding

System.out.println("Display from Derived");

}

}

// Main class

public class Main {

public static void main(String[] args) {

// Abstract and Interface

Dog dog = new Dog();

dog.sound(); // Woof

dog.play(); // Dog is playing

// Overriding and Overloading

MathUtils mathUtils = new MathUtils();

mathUtils.add(5, 10); // Method Overloading

mathUtils.add(5.5, 10.5); // Method Overloading

Base base = new Derived();

base.display(); // Display from Derived (Method Overriding)

// this and super

System.out.println("End of demonstration.");

}

}