INHERITANCE TRONG JAVA OOP — TOÀN DIỆN

Phần 1: Từ mục 1 đến mục 4

MỤC LỤC

1. Định nghĩa, nguồn gốc và ý nghĩa lý thuyết của kế thừa

1.1. Định nghĩa

Kế thừa (Inheritance) là cơ chế cho phép một lớp mới (lớp con/subclass/child class) thừa hưởng các thuộc tính (fields/biến) và phương thức (methods) của một lớp đã tồn tại (lớp cha/superclass/parent class). Lớp con có thể sử dụng, ghi đè (override), mở rộng hoặc bổ sung chức năng mới ngoài những gì kế thừa từ lớp cha. Kế thừa là nền tảng để xây dựng mối quan hệ "is-a" (là một loại) giữa các đối tượng.

Tài liệu tham khảo: Oracle Java Documentation: Inheritance
https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

1.2. Nguồn gốc lịch sử

Kế thừa được hình thành cùng với sự ra đời của lập trình hướng đối tượng từ ngôn ngữ Simula (1967), sau này hoàn thiện và phổ biến qua Smalltalk, C++, Java… Ý tưởng mô phỏng thế giới thực: Đối tượng "con" kế thừa tính chất từ đối tượng "cha" (ví dụ: Chó là Động Vật, Chó có thể sủa).

1.3. Ý nghĩa lý thuyết và triết lý OOP

- Tổ chức và mô hình hóa thế giới thực bằng cấu trúc phân cấp.
- Hỗ trợ tái sử dụng mã nguồn và giảm trùng lặp code.
- Cơ sở để xây dựng tính đa hình (polymorphism).
- Định nghĩa rõ ràng quan hệ giữa các lớp, tăng khả năng mở rộng và bảo trì phần mềm.

2. Vai trò, tác dụng của tính kế thừa trong phát triển phần mềm

- Tái sử dụng mã nguồn: Không phải viết lại logic chung cho mỗi lớp con; chỉ cần kế thừa rồi bổ sung/ghi đè những điểm khác biệt.
- Tổ chức phân cấp (Hierarchy): Tạo ra cấu trúc "cây" các lớp, giúp hệ thống logic, dễ quản lý và phát triển.
- Hỗ trợ mở rộng, bảo trì phần mềm: Có thể mở rộng hệ thống bằng cách thêm lớp con mới mà không ảnh hưởng lớp cha. Dễ dàng sửa chữa hoặc thay đổi logic chung tại một điểm trung tâm (lớp cha).
- Tăng tính linh hoạt và chuẩn hóa hệ thống: Xây dựng các chuẩn interface/abstract class, mọi lớp con tuân theo một khuôn mẫu chung.

3. Hiện thực hóa (triển khai) kế thừa trong Java

3.1. Từ khóa extends và super

- extends: Sử dụng khi khai báo lớp con, ví dụ: class Dog extends Animal { ... }
- super: Dùng để gọi thuộc tính, phương thức, hoặc constructor của lớp cha.

3.2. Các loại kế thừa trong Java

Loại kế thừa	Mô tả
Single Inheritance	Một lớp con kế thừa một lớp cha duy nhất
Multilevel Inheritance	Lớp con kế thừa lớp cha, lớp cha lại kế thừa lớp ông (nhiều cấp)
Hierarchical Inheritance	Một lớp cha được nhiều lớp con kế thừa

3.3. Lý do Java không hỗ trợ đa kế thừa bằng class

Java không cho phép một lớp kế thừa nhiều lớp cha cùng lúc (multiple inheritance) để tránh "diamond problem" — khi một phương thức/tính chất được định nghĩa trong nhiều lớp cha, compiler sẽ không biết nên kế thừa cái nào. Thay vào đó, Java hỗ trợ đa kế thừa qua interface.

3.4. Interface và đa kế thừa kiểu interface

Một lớp có thể implements nhiều interface, qua đó có thể thừa hưởng nhiều hành vi khác nhau mà không gây mâu thuẫn code. Interface hỗ trợ xây dựng các hợp đồng (contract) mà nhiều class cùng tuân theo.

4. Ví dụ mã nguồn minh họa từ cơ bản đến nâng cao

4.1. Ví dụ đơn giản — Single Inheritance

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
```

```
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking");
    }
}
public class TestInheritance {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();  // Kế thừa từ Animal
        d.bark(); // Phương thức riêng của Dog
    }
}
```

4.2. Ví dụ kế thừa đa cấp (Multilevel)

```
class Animal {
    void eat() { System.out.println("Animal eats"); }
}
class Mammal extends Animal {
    void walk() { System.out.println("Mammal walks"); }
}
class Dog extends Mammal {
    void bark() { System.out.println("Dog barks"); }
}
```

4.3. Ví dụ hierarchical

```
class Animal { void eat() { System.out.println("Eat"); } }
class Dog extends Animal { void bark() { System.out.println("Bark"); } }
class Cat extends Animal { void meow() { System.out.println("Meow"); } }
```

4.4. Ví dụ interface và đa kế thừa qua interface

```
interface Swimmer { void swim(); }
interface Runner { void run(); }
class Duck implements Swimmer, Runner {
    public void swim() { System.out.println("Duck swims"); }
    public void run() { System.out.println("Duck runs"); }
```

}

4.5. Phân tích code nâng cao

- Lớp con có thể override phương thức của lớp cha bằng cách khai báo lại với cùng tên, cùng tham số.
- Có thể gọi phương thức lớp cha bằng super.methodName() nếu cần.