ABSTRACTION 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 trừu tượng

1.1. Định nghĩa

Trừu tượng (Abstraction) trong lập trình hướng đối tượng là quá trình ẩn giấu các chi tiết cài đặt, chỉ cung cấp cho bên ngoài những thông tin, hành vi "cần thiết" thông qua một giao diện (interface) hoặc lớp trừu tượng (abstract class). Trong Java, abstraction cho phép lập trình viên "mô tả" những gì một đối tượng có thể làm mà không cần biết làm như thế nào bên trong.

Tài liệu: Oracle Java Tutorials – What Is Abstraction?
https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

1.2. Nguồn gốc lý thuyết

Trừu tượng là khái niệm nền tảng trong toán học, khoa học máy tính, OOP. Ý tưởng đến từ quan sát thế giới thực: Khi nhìn vào một chiếc xe, ta quan tâm đến hành vi "chạy", "phanh" thay vì chi tiết từng bánh răng, cơ chế vận hành bên trong.

1.3. Ý nghĩa triết lý OOP

- Tách biệt giữa "cái gì" (what) và "như thế nào" (how): Người dùng chỉ cần quan tâm chức năng, không cần biết chi tiết cài đặt.
- Tạo chuẩn mực: Tất cả các class cùng loại đều phải triển khai các hành vi đã "cam kết".
- Đảm bảo tính mở rộng: Hệ thống dễ mở rộng, bảo trì vì các chi tiết có thể thay đổi mà không ảnh hưởng code sử dụng.

2. Vai trò, tác dụng của tính trừu tượng trong phát triển phần mềm

- Giảm phụ thuộc: Code sử dụng chỉ phụ thuộc vào interface/abstract class, không phụ thuộc implement cụ thể.
- Dễ bảo trì, mở rộng: Thay đổi logic bên trong không ảnh hưởng code client.
- Chuẩn hóa quy tắc: Tất cả class cùng loại (implement interface/abstract) đều buộc phải tuân theo chuẩn chung.
- Tăng khả năng tái sử dụng: Nhóm các đối tượng cùng "hành vi" dưới một giao diện chung, dễ tổ chức hệ thống lớn.
- Hỗ trợ test và phát triển module: Dễ thay thế giả lập (mock), cắm plugin mới vào hệ thống.

3. Cách hiện thực hóa (triển khai) trừu tượng trong Java

3.1. abstract class là gì?

abstract class là lớp không thể khởi tạo đối tượng trực tiếp, có thể chứa:
- phương thức trừu tượng (chỉ khai báo, không code thân hàm)
- phương thức đã cài đặt cụ thể (concrete)
- thuộc tính, constructor, static method...

Lớp con kế thừa abstract class phải override (cài đặt lại) tất cả các phương thức trừu tượng.

```
abstract class Animal {
    abstract void makeSound();
    void sleep() { System.out.println("Sleeping..."); }
}
```

3.2. interface là gì?

interface là bản "cam kết" các phương thức, chỉ chứa khai báo hàm (từ Java 8 trở đi có thể có default method, static method). Không có constructor, không có biến instance (chỉ constant). Một class có thể implement nhiều interface.

```
interface Flyable {
    void fly();
}
class Bird implements Flyable {
    public void fly() { System.out.println("Bird flies"); }
}
```

3.3. Sự khác biệt giữa abstract class và interface

Tiêu chí	abstract class	interface	Có constructor	Có biến instance	Có code cụ thể
Khởi tạo đối tượng	Không	Không	Có	Có	Có thể
Kế thừa	1 lớp cha	Nhiều interface	Có	Không	Từ Java 8 (default/static method)
Dùng khi	Có logic dùng chung, có trạng thái	Chỉ định cam kết hành vi	Có	Không	Có thể

3.4. Khi nào dùng abstract class, khi nào dùng interface?

- abstract class: Khi các class con cùng loại, có "chung bản chất", chia sẻ code hoặc thuộc tính chung.
- interface: Khi nhiều class không liên quan muốn cam kết hành vi chung, hoặc khi cần hỗ trợ đa kế thừa.

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

4.1. Ví dụ abstract class cơ bản

```
abstract class Animal {
    abstract void makeSound();
    void eat() { System.out.println("Ăn thức ăn"); }
}
class Dog extends Animal {
    void makeSound() { System.out.println("Gâu gâu"); }
}


Animal a = new Dog();
a.makeSound(); // "Gâu gâu"
a.eat();       // "Ăn thức ăn"
```

4.2. Ví dụ interface cơ bản

```
interface Printable {
    void print();
}
class Document implements Printable {
    public void print() { System.out.println("In tài liệu"); }
```

```
}


Printable p = new Document();
p.print(); // "In tài liệu"
```

4.3. Kết hợp abstract class và interface (nâng cao)

```
interface Swimmable {
    void swim();
}
abstract class Animal {
    abstract void makeSound();
}
class Dolphin extends Animal implements Swimmable {
    void makeSound() { System.out.println("Click-click"); }
    public void swim() { System.out.println("Dolphin swims"); }
}


Animal d = new Dolphin();
d.makeSound(); // "Click-click"
((Swimmable)d).swim(); // "Dolphin swims"
```

4.4. Phân tích các trường hợp thực tế

- Abstract class phù hợp cho nhóm đối tượng "cùng bản chất" (ví dụ: Vehicle, Animal, Person…)
- Interface phù hợp cho hành vi "khả năng", đa dạng nhóm đối tượng đều có thể "implement" (ví dụ: Runnable, Comparable, Serializable…)