# ABSTRACTION TRONG JAVA OOP — TOÀN DIỆN

*Phần 2: Từ mục 5 đến mục 11*

## 5. So sánh trừu tượng với các đặc tính OOP khác

### 5.1. Trừu tượng vs Đóng gói (Encapsulation)

- Trừu tượng: Ẩn chi tiết cài đặt, chỉ cho bên ngoài biết giao diện (method, hành vi) cần thiết.
- Đóng gói: Bao bọc dữ liệu và phương thức bên trong đối tượng, kiểm soát quyền truy cập (private/protected).
- Liên hệ: Trừu tượng thiên về "giao tiếp" với bên ngoài, đóng gói thiên về bảo vệ dữ liệu bên trong.

### 5.2. Trừu tượng vs Kế thừa (Inheritance)

- Kế thừa: Cho phép lớp con nhận lại thuộc tính/phương thức từ lớp cha, hỗ trợ tái sử dụng và tổ chức phân cấp.
- Trừu tượng: Định nghĩa chuẩn hành vi, các lớp con buộc phải thực hiện đầy đủ các hành vi này (interface/abstract class).
- Liên hệ: Abstract class là công cụ thể hiện cả trừu tượng và kế thừa.

### 5.3. Trừu tượng vs Đa hình (Polymorphism)

- Trừu tượng: Định nghĩa ra interface/hành vi tổng quát, chưa cài đặt chi tiết.
- Đa hình: Sử dụng cùng một interface nhưng hành vi thực tế phụ thuộc vào đối tượng cụ thể lúc runtime.
- Liên hệ: Trừu tượng tạo điều kiện cho đa hình vận hành.

## 6. Lợi ích và hạn chế của tính trừu tượng

### 6.1. Lợi ích

- Giảm phụ thuộc: Code client chỉ phụ thuộc interface/abstract class.
- Mở rộng, bảo trì dễ dàng: Thay đổi cài đặt không ảnh hưởng code sử dụng.
- Tăng khả năng kiểm thử: Dễ dàng mock các implement cho unit test.
- Chuẩn hóa hành vi: Đảm bảo các class cùng loại phải tuân theo chuẩn chung.
- Tối ưu hóa thiết kế module: Dễ phát triển các hệ thống plugin, module mở rộng.

### 6.2. Hạn chế

- Có thể trừu tượng hóa quá mức: Làm hệ thống phức tạp, khó hiểu, tốn thời gian bảo trì.
- Gây khó khăn cho debug: Khi gặp lỗi, phải kiểm tra nhiều tầng interface/abstract class để tìm code thực thi.
- Khó truyền đạt ý tưởng: Nếu không đặt tên interface, abstract class hợp lý, dễ gây nhầm lẫn.

## 7. Các lỗi phổ biến và ví dụ thực tế về sử dụng chưa đúng trừu tượng

### 7.1. Lỗi phổ biến

- Tạo interface/abstract class "vô hồn": Không có ý nghĩa thực tiễn, chỉ để "cho có".
- Đặt quá nhiều phương thức vào interface: Khiến implement phức tạp, vi phạm nguyên lý Interface Segregation.
- Chỉ dùng 1 implement: Nếu chỉ có 1 class thực sự implement, có thể chưa cần abstraction.
- Lặp lại code giữa abstract class và interface.

### 7.2. Ví dụ thực tế

#### *Lỗi 1: Interface "quá tải"*

```
interface MultiFunctionDevice {
    void print();
    void scan();
    void fax();
    void email();
}
class SimplePrinter implements MultiFunctionDevice {
    public void print() { ... }
    public void scan() { /* Không hỗ trợ! */ }
    public void fax() { /* Không hỗ trợ! */ }
    public void email() { /* Không hỗ trợ! */ }
}
// => Vi phạm nguyên lý Interface Segregation.
```

#### *Lỗi 2: Tạo abstraction "thừa"*

```
interface Fruit {
    void eat();
```

```
}
class Apple implements Fruit {
    public void eat() { ... }
}
// Nếu hệ thống chỉ có Apple, interface là thừa.
```

## 8. Ứng dụng thực tiễn của trừu tượng trong các dự án Java lớn

- Framework/Library: Spring, Hibernate dùng interface/abstract class để tách biệt API với implement (ví dụ: JpaRepository, ApplicationContext...).
- Plugin System: Eclipse IDE, các hệ thống CMS dùng interface cho plugin mở rộng.
- Design API/SDK: Google Maps API, Java Collections (List, Map...) đều là interface.
- Test Automation: Mock các implement của interface để kiểm thử đơn vị.

## 9. Các mẫu thiết kế (Design Pattern) liên quan đến trừu tượng

- Factory Pattern: Tạo object qua interface/abstract class, client không biết class cụ thể.
- Strategy Pattern: Định nghĩa family of algorithms qua interface, runtime chọn implement phù hợp.
- Adapter Pattern: Chuyển đổi interface cũ/new thành interface mà client mong muốn.
- Template Method Pattern: Abstract class định nghĩa khung thuật toán, subclass triển khai bước cụ thể.
- Bridge Pattern: Tách abstraction và implementation thành hai class hierarchy riêng biệt.

## 10. Câu hỏi phỏng vấn thường gặp về trừu tượng

1. 1. Abstraction là gì? Cho ví dụ thực tiễn trong Java.
2. 2. Phân biệt abstract class và interface (trước/sau Java 8).
3. 3. Khi nào dùng interface, khi nào dùng abstract class?
4. 4. Có thể có method body trong interface không?
5. 5. Có thể implement nhiều interface, extend nhiều abstract class không?
6. 6. Tại sao abstraction giúp tăng khả năng mở rộng phần mềm?
7. 7. Sự khác biệt giữa trừu tượng và đóng gói?
8. 8. Tại sao không nên nhồi nhét nhiều method vào một interface?
9. 9. Cho ví dụ về Design Pattern dùng abstraction?
10. 10. Java có cho phép interface có biến instance không?

## 11. Tài liệu, nguồn tham khảo uy tín


- Oracle Java Tutorials – Abstraction: https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html
- Head First Java (O'Reilly, Kathy Sierra & Bert Bates) – Chương về abstract class và interface.
- Effective Java (Joshua Bloch) – Item về Interface và Abstract class.
- Clean Code (Robert C. Martin) – Interface Segregation Principle.
- Java SE Documentation: https://docs.oracle.com/javase/
- Baeldung – Abstraction in Java: https://www.baeldung.com/java-abstraction
- GeeksforGeeks – Abstraction in Java: https://www.geeksforgeeks.org/abstraction-in-java/
- TutorialsPoint – Java - Abstraction: https://www.tutorialspoint.com/java/java_abstraction.htm
- GoF Design Patterns book (Factory, Adapter, Bridge, ...)


### Kết luận

Trừu tượng là chìa khóa cho OOP hiện đại, giúp giảm phụ thuộc, tăng khả năng mở rộng, test, và phát triển module lớn. Tuy nhiên, cần vận dụng vừa đủ, tránh tạo abstraction "thừa" hoặc quá tải.