INHERITANCE TRONG JAVA OOP — TOÀN DIỆN

Phần 2: Từ mục 5 đến mục 11

5. So sánh kế thừa với các đặc tính OOP khác

5.1. Kế thừa vs Đóng gói (Encapsulation)

- Kế thừa giúp lớp con tái sử dụng thuộc tính và hành vi từ lớp cha, tạo mối quan hệ "is-a" giữa các đối tượng.
- Đóng gói giúp bảo vệ dữ liệu và kiểm soát truy cập, che giấu chi tiết cài đặt.
- Quan hệ: Kế thừa là về mở rộng cấu trúc và hành vi; đóng gói là về bảo vệ, giới hạn truy cập.

5.2. Kế thừa vs Đa hình (Polymorphism)

- Kế thừa tạo nền tảng cho đa hình — một biến kiểu lớp cha có thể tham chiếu tới đối tượng lớp con.
- Đa hình cho phép gọi phương thức của lớp con thông qua tham chiếu lớp cha, giúp mở rộng linh hoạt (ví dụ: override).

Ví dụ:

```
Animal a = new Dog(); // Đa hình nhờ kế thừa
a.eat(); // Gọi eat() ở Dog nếu Dog override phương thức này
```

5.3. Kế thừa vs Trừu tượng hóa (Abstraction)

- Trừu tượng hóa là che giấu chi tiết không cần thiết, chỉ thể hiện giao diện bên ngoài.
- Kế thừa là một công cụ giúp cài đặt trừu tượng hóa (thường thông qua abstract class và interface).

6. Lợi ích và hạn chế của tính kế thừa

6.1. Lợi ích

- Tái sử dụng mã nguồn: Giảm trùng lặp code, tăng hiệu quả phát triển.
- Dễ mở rộng: Thêm tính năng mới qua lớp con mà không thay đổi lớp cha.
- Hỗ trợ đa hình: Dễ xây dựng các cấu trúc xử lý linh hoạt.
- Tổ chức hệ thống rõ ràng: Cấu trúc phân cấp logic, dễ bảo trì.

6.2. Hạn chế

- Lạm dụng kế thừa: Nếu dùng sai, hệ thống trở nên rối rắm, khó bảo trì (ví dụ: cây kế thừa quá sâu).
- Kết dính chặt (tight coupling): Lớp con phụ thuộc vào lớp cha, dễ phát sinh lỗi khi lớp cha thay đổi.
- Không phù hợp cho mọi mối quan hệ: Không phải cứ có chung thuộc tính là dùng kế thừa ("is-a" vs "has-a" — nên cân nhắc dùng composition khi thích hợp).
- Java không hỗ trợ đa kế thừa bằng class: Giới hạn về khả năng kết hợp nhiều lớp cha.

7. Các lỗi phổ biến và ví dụ thực tế về việc lạm dụng/hiểu sai kế thừa

7.1. Lỗi phổ biến

- Dùng kế thừa thay vì composition: Không phân biệt rõ "is-a" và "has-a" dẫn tới mô hình hóa sai quan hệ giữa các đối tượng.
- Tạo cây kế thừa quá sâu: Khiến code khó hiểu, khó bảo trì.
- Override phương thức lớp cha không hợp lý: Có thể phá vỡ logic lớp cha.
- Lạm dụng kế thừa chỉ để tái sử dụng code: Dẫn đến những lớp con "vô nghĩa" về mặt ngữ nghĩa.

7.2. Ví dụ thực tế

Lạm dụng kế thừa:

```
class Engine {
   void start() { ... }
}
class Car extends Engine { // Sai: Car không phải là Engine!
   // Nên dùng: class Car { private Engine engine; }
}
```

Đúng:

```
class Car {
   private Engine engine;
   void startCar() { engine.start(); }
}
```

Car có động cơ (has-a), không phải là động cơ (is-a).

Cây kế thừa quá sâu:

class A { }
class B extends A { }
class C extends B { }
class D extends C { }
// Dùng không hợp lý sẽ gây rối code, khó bảo trì.

8. Ứng dụng thực tiễn của kế thừa trong các dự án Java lớn

- Mô hình hóa domain: Ví dụ: User → Student, Admin, Guest.
- Framework Java (Spring, Hibernate): Các lớp base như AbstractController, BaseEntity,... giúp chuẩn hóa, tái sử dụng logic chung.
- Swing/AWT: Các component UI kế thừa từ các lớp gốc như JComponent, JFrame.
- Exception hierarchy: Toàn bộ exception trong Java kế thừa từ Throwable → Exception/Error → ...

9. Các mẫu thiết kế (Design Patterns) liên quan đến kế thừa

- Template Method Pattern: Lớp cha định nghĩa skeleton của thuật toán, các bước cụ thể override ở lớp con.
- Factory Method Pattern: Lớp cha định nghĩa interface để tạo object, lớp con quyết định đối tượng cụ thể được tạo.
- Strategy Pattern: Dùng interface/abstract class để các thuật toán khác nhau kế thừa và hoán đổi linh hoạt.
- Decorator Pattern: Lớp decorator kế thừa/làm việc với interface/lớp cha để mở rộng hành vi.

10. Câu hỏi phỏng vấn thường gặp về kế thừa

1. 1. Kế thừa là gì? Ý nghĩa của kế thừa trong Java?
2. 2. Phân biệt kế thừa (Inheritance) và giao diện (Interface) trong Java.
3. 3. Tại sao Java không hỗ trợ đa kế thừa bằng class?
4. 4. Giải thích “diamond problem” và cách Java xử lý?
5. 5. So sánh kế thừa với composition, khi nào nên dùng cái nào?
6. 6. Từ khóa super có vai trò gì?
7. 7. Override là gì? Khi nào nên override phương thức lớp cha?
8. 8. Trường hợp thực tế nào không nên dùng kế thừa?
9. 9. Các design pattern nào tận dụng kế thừa?

10. 10. Java cho phép một class vừa extends một class vừa implements nhiều interface không?

11. Tài liệu, nguồn tham khảo uy tín

- Oracle Java Tutorials – Inheritance: https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- Head First Java (O'Reilly, Kathy Sierra & Bert Bates) – Chương về Kế thừa.
- Effective Java (Joshua Bloch) – Item về Inheritance.
- Clean Code (Robert C. Martin) – Các chương về cấu trúc class và hierarchy.
- Java SE 8 Documentation: https://docs.oracle.com/javase/8/docs/
- Baeldung – Java Inheritance: https://www.baeldung.com/java-inheritance
- GeeksforGeeks – Inheritance in Java: https://www.geeksforgeeks.org/inheritance-in-java/
- TutorialsPoint – Java - Inheritance: https://www.tutorialspoint.com/java/java_inheritance.htm
- Design Patterns: Elements of Reusable Object-Oriented Software (GoF book)

Kết luận

Tính kế thừa giúp Java mô hình hóa mối quan hệ trong thế giới thực, thúc đẩy tái sử dụng code, chuẩn hóa và mở rộng hệ thống. Tuy nhiên, cần dùng kế thừa một cách hợp lý, kết hợp với các nguyên tắc như composition, interface, để xây dựng phần mềm hiệu quả và dễ bảo trì.