POLYMORPHISM TRONG JAVA OOP — TOÀN DIỆN

Phần 2: Từ mục 5 đến mục 11

5. So sánh đa hình với các đặc tính OOP khác

5.1. Đa hình vs Kế thừa (Inheritance)

- Kế thừa là cơ chế cho phép lớp con nhận thuộc tính, phương thức của lớp cha, là nền tảng để thực thi đa hình.
- Đa hình là khả năng cùng một interface có thể biểu diễn nhiều hành vi khác nhau tùy theo object thực tế.
- Mối quan hệ: Kế thừa giúp đa hình vận hành, nhưng đa hình mới là đặc tính cho phép gọi cùng một method qua tham chiếu lớp cha hoặc interface, kết quả hành vi khác nhau tùy object.

5.2. Đa hình vs Đóng gói (Encapsulation)

- Đóng gói giúp bảo vệ dữ liệu, che giấu chi tiết triển khai và kiểm soát truy cập.
- Đa hình giúp mở rộng hành vi, cho phép sử dụng chung interface, tăng tính linh hoạt khi lập trình.

5.3. Đa hình vs Trừu tượng hóa (Abstraction)

- Trừu tượng hóa ẩn bớt chi tiết, chỉ cung cấp interface cần thiết.
- Đa hình là khả năng sử dụng interface này với nhiều cài đặt khác nhau.
- Ví dụ: Abstract class/Interface + đa hình = khả năng gọi chung, hành vi khác nhau.

6. Lợi ích và hạn chế của tính đa hình

6.1. Lợi ích

- Linh hoạt, mở rộng hệ thống dễ dàng: Thêm class mới không ảnh hưởng code cũ.
- Tối ưu hóa thiết kế: Code tổng quát, dễ bảo trì.
- Dễ kiểm thử: Có thể thay thế implement khác nhau (mock, fake,...).
- Hỗ trợ cho các mẫu thiết kế OOP hiện đại (Strategy, Factory, Command, ...).

6.2. Hạn chế

- Khó truy vết lỗi: Hành vi thực tế quyết định ở runtime.
- Có thể lạm dụng: Quá nhiều tầng kế thừa/đa hình dẫn tới code khó hiểu.
- Hiệu suất: Runtime polymorphism có thể chậm hơn so với static binding (overloading).

7. Các lỗi phổ biến và ví dụ thực tế về sử dụng chưa đúng đa hình

7.1. Lỗi phổ biến

- Không override đúng method: Quên hoặc sai chữ ký method, dẫn tới không có đa hình thực sự.
- Lạm dụng ép kiểu (downcasting): Thay vì sử dụng đa hình đúng cách, lại ép kiểu xuống lớp con.
- Tạo method overload quá nhiều: Làm code khó đọc, khó bảo trì.
- Dùng biến kiểu con thay vì interface/lớp cha: Mất tính mở rộng của đa hình.

7.2. Ví dụ thực tế

Lỗi 1: Không ghi đè đúng

```
class Animal {
    void sound() { System.out.println("..."); }
}
class Dog extends Animal {
    void Sound() { System.out.println("Woof"); } // Sai tên method, không override!
}
// Gọi a.sound() vẫn chỉ ra "...", không phải "Woof".
```

Lỗi 2: Lạm dụng ép kiểu

```
Animal a = new Dog();
((Dog)a).bark(); // Dùng downcast thay vì gọi qua đa hình
// Nên dùng method chung thay vì ép kiểu.
```

8. Ứng dụng thực tiễn của đa hình trong các dự án Java lớn

- Xây dựng API/Framework: Spring, Hibernate, các thư viện Java đều thiết kế interface/abstract class, client code làm việc với interface, runtime chọn implement phù hợp.
- UI Event Listener: Java Swing, Android, ... — mọi event đều là interface, mỗi loại component implement khác nhau.
- Plugin Architecture: Hệ thống hỗ trợ plugin, extension dùng đa hình interface để kết nối nhiều thành phần mở rộng.
- Xử lý danh sách object hỗn hợp: Ví dụ, mảng Shape[] có thể chứa Circle, Rectangle, Triangle..., for-each gọi draw() và mỗi object tự xử lý.

9. Các mẫu thiết kế (Design Pattern) liên quan đến đa hình

- Strategy Pattern: Định nghĩa interface cho một thuật toán, cho phép đổi thuật toán runtime.
- Factory Pattern: Dùng interface/abstract class để client code không phụ thuộc vào class cụ thể.
- Command Pattern: Lưu trữ và thực thi các object command implement interface chung.
- Observer Pattern: Các observer implement cùng interface, subject gọi notify() cho tất cả.
- Template Method Pattern: Phương thức khung trong abstract class, subclass override step cụ thể.

10. Câu hỏi phỏng vấn thường gặp về đa hình

1. 1. Polymorphism là gì? Đưa ví dụ thực tiễn trong Java.
2. 2. Phân biệt compile-time và runtime polymorphism.
3. 3. Sự khác biệt giữa overriding và overloading?
4. 4. Tại sao nên lập trình dựa trên interface, không dựa trên implementation cụ thể?
5. 5. Khi nào cần dùng downcasting? Nguy hiểm gì khi dùng?
6. 6. Có thể overload static method hoặc constructor không?
7. 7. Interface giúp đa hình như thế nào?
8. 8. Làm sao Java quyết định method nào sẽ được gọi ở runtime?
9. 9. Có thể override method private không?
10. 10. Cho ví dụ thực tế ứng dụng đa hình trong framework phổ biến.

11. Tài liệu, nguồn tham khảo uy tín

- Oracle Java Tutorials – Polymorphism: https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
- Head First Java (O'Reilly, Kathy Sierra & Bert Bates) – Chương về Đa hình.
- Effective Java (Joshua Bloch) – Item về Polymorphism.
- Clean Code (Robert C. Martin) – Các chương về interface, polymorphism.
- Java SE 8 Documentation: https://docs.oracle.com/javase/8/docs/
- Baeldung – Polymorphism in Java: https://www.baeldung.com/java-polymorphism
- GeeksforGeeks – Polymorphism in Java: https://www.geeksforgeeks.org/polymorphism-in-java/
- TutorialsPoint – Java - Polymorphism: https://www.tutorialspoint.com/java/java_polymorphism.htm
- Design Patterns: Elements of Reusable Object-Oriented Software (GoF book)

Kết luận

Tính đa hình làm cho Java OOP cực kỳ linh hoạt và mạnh mẽ, tăng tính mở rộng, bảo trì, xây dựng hệ thống có khả năng "thay thế" thành phần dễ dàng mà không sửa đổi code cũ. Nhưng muốn tận dụng hiệu quả, phải hiểu bản chất, tránh lạm dụng hoặc sử dụng sai cách.