# ENCAPSULATION TRONG JAVA OOP — TOÀN DIỆN

*Phần 2: Từ mục 5 đến mục 11*

## 5. So sánh đóng gói với các đặc tính OOP khác

### 5.1. Đóng gói vs Kế thừa (Inheritance)

- Đóng gói tập trung vào việc "giấu" dữ liệu, kiểm soát truy cập, bảo vệ nội dung bên trong một đối tượng.
- Kế thừa là cơ chế cho phép một lớp con thừa hưởng thuộc tính và phương thức từ lớp cha.
- Đóng gói giúp lớp cha kiểm soát mức độ tiếp cận dữ liệu của lớp con (ví dụ, các trường private không được lớp con truy cập trực tiếp).
- Sự phối hợp: Đóng gói bảo vệ dữ liệu, kế thừa mở rộng chức năng.

### 5.2. Đóng gói vs Đa hình (Polymorphism)

- Đóng gói: Che giấu dữ liệu, cung cấp giao diện truy cập dữ liệu.
- Đa hình: Một interface hoặc lớp cha có thể đại diện cho nhiều đối tượng thuộc các lớp con khác nhau, thực thi hành vi khác nhau cùng tên phương thức.
- Liên hệ: Nhờ đóng gói, đối tượng chỉ tương tác qua giao diện chung, không cần biết chi tiết từng lớp con — tạo điều kiện cho đa hình phát huy hiệu quả.

### 5.3. Đóng gói vs Trừu tượng hóa (Abstraction)

- Trừu tượng hóa là che giấu các chi tiết không cần thiết, chỉ thể hiện những gì "cần thiết" cho bên ngoài biết.
- Đóng gói là bao bọc dữ liệu và các phương thức vào trong đối tượng, đồng thời kiểm soát quyền truy cập.
- Trừu tượng hóa thiên về ý tưởng (cái gì), đóng gói thiên về thực thi (làm như thế nào và bảo vệ ra sao).
- Trong thực tế, hai đặc tính này kết hợp chặt chẽ, bổ sung cho nhau.

## 6. Lợi ích và hạn chế của tính đóng gói

### 6.1. Lợi ích

- Bảo vệ dữ liệu: Ngăn chặn thay đổi dữ liệu ngoài ý muốn.
- Che giấu chi tiết cài đặt: Bên ngoài chỉ biết interface, không cần biết (hoặc không được

phép biết) bên trong.
- Kiểm soát hợp lệ dữ liệu: Dễ dàng chèn logic kiểm tra dữ liệu khi sử dụng setter.
- Bảo trì dễ dàng: Có thể thay đổi cách triển khai mà không ảnh hưởng code bên ngoài.
- Tăng tính mô-đun: Hạn chế phụ thuộc giữa các module/phần mềm.

### 6.2. Hạn chế

- Viết code dài dòng hơn: Cần thêm getter/setter, đôi khi làm code "cồng kềnh" nếu không sử dụng công cụ hỗ trợ.
- Lạm dụng getter/setter: Nếu lộ hết getter/setter mà không kiểm soát, bản chất đóng gói bị phá vỡ.
- Giảm tính linh hoạt trong một số trường hợp: Có thể làm việc debug phức tạp hơn, nhất là khi kiểm soát quá nhiều lớp truy cập.

## 7. Các lỗi phổ biến khi sử dụng đóng gói

### 7.1. Lỗi thường gặp

- Khai báo thuộc tính là public: Vô tình hoặc chủ ý công khai dữ liệu, dẫn tới mất kiểm soát, vi phạm đóng gói.
- Cung cấp quá nhiều setter không kiểm soát: Cho phép mọi giá trị được set mà không kiểm tra tính hợp lệ.
- Lộ quá nhiều getter/setter: Khi class có quá nhiều trường, tạo getter/setter cho tất cả sẽ khiến class "mất kiểm soát", trở thành "Java Bean anemic" (object nghèo nàn hành vi).
- Khai báo protected mà không hiểu rõ hệ quả: Lớp con có thể truy cập trực tiếp, nếu dùng không đúng dễ dẫn đến lỗi dữ liệu.
- Không dùng final cho class không muốn kế thừa: Đôi khi class nên đóng hoàn toàn (immutable).

### 7.2. Ví dụ thực tế về lỗi đóng gói

***Lỗi 1: Thuộc tính public***

```
public class Student {
    public String name; // Không nên!
    public int age;
}
```

***Bất kỳ ai cũng có thể sửa giá trị name, age mà không bị kiểm soát.***

***Lỗi 2: Setter không kiểm tra hợp lệ***

```
public void setAge(int age) {
    this.age = age; // Không kiểm soát giá trị, có thể gán age < 0
}
```

***Lỗi 3: Lộ toàn bộ getter/setter***

*// Một class với 20 trường private, tạo auto toàn bộ getter/setter bằng IDE mà không kiểm soát logic*

***Dẫn đến object bị lộ hết dữ liệu, bản chất "đóng gói" không còn ý nghĩa.***

## 8. Ứng dụng thực tiễn của đóng gói trong các dự án Java lớn

- Domain Model: Trong các hệ thống lớn (ví dụ hệ thống tài chính, ngân hàng), các class domain (Account, Customer, Transaction,...) luôn che giấu dữ liệu nhạy cảm (số dư, số tài khoản, trạng thái giao dịch) và chỉ cung cấp các phương thức (deposit, withdraw, transfer, validate,...) để thao tác một cách kiểm soát.
- API và Service Layer: Khi xây dựng RESTful API, các DTO (Data Transfer Object) đều sử dụng encapsulation để bảo vệ trường dữ liệu nhạy cảm, chỉ expose những gì client cần.
- Framework lớn như Spring, Hibernate: Đều khuyến nghị dùng private field + getter/setter để đảm bảo nguyên tắc OOP.
- Mô hình POJO (Plain Old Java Object): Bản chất POJO là encapsulated object.

Ví dụ thực tế:
- User object trong hệ thống quản lý người dùng: Trường password không bao giờ được public hoặc cho phép truy xuất thẳng, chỉ được set qua logic đã mã hóa.
- Java Collections: List, Map, Set đều encapsulate dữ liệu, chỉ expose method thao tác (add, remove, get,...) chứ không cho truy cập trực tiếp mảng bên trong.

## 9. Các mẫu thiết kế (Design Pattern) liên quan đến tính đóng gói

- Singleton Pattern: Đóng gói việc khởi tạo instance, không cho phép tạo nhiều đối tượng từ ngoài class.
- Factory Pattern: Đóng gói quá trình tạo object, che giấu chi tiết cài đặt object cụ thể.
- Builder Pattern: Đóng gói quá trình khởi tạo đối tượng phức tạp.
- Facade Pattern: Đóng gói các subsystem phức tạp, cung cấp một interface đơn giản cho bên ngoài.
- Adapter Pattern: Đóng gói một class hoặc interface cũ để phù hợp với interface mới.

- Decorator Pattern: Đóng gói đối tượng cũ để mở rộng chức năng mà không thay đổi code gốc.

Tất cả các pattern trên đều vận dụng encapsulation để che giấu chi tiết bên trong, chỉ expose những gì client cần.

## 10. Câu hỏi phỏng vấn thường gặp về đóng gói

1. 1. Đóng gói là gì? Nêu ví dụ thực tiễn trong Java.
2. 2. Sự khác biệt giữa đóng gói và trừu tượng hóa là gì?
3. 3. Tại sao lại dùng private cho các trường dữ liệu trong class?
4. 4. Getter/setter có phá vỡ đóng gói không?
5. 5. Hạn chế của tính đóng gói là gì? Có trường hợp nào không nên đóng gói không?
6. 6. So sánh encapsulation với information hiding.
7. 7. Trong Java, có thể nào truy cập biến private từ ngoài class không?
8. 8. Khi nào nên dùng protected thay cho private?
9. 9. Hãy nêu ví dụ về lỗi điển hình khi sử dụng đóng gói không đúng.
10. 10. Các design pattern nào vận dụng mạnh encapsulation?

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


- Oracle Java Tutorials — Encapsulation: https://docs.oracle.com/javase/tutorial/java/concepts/encapsulation.html
- Head First Java (O'Reilly, Kathy Sierra & Bert Bates) — Chapter về OOP và Encapsulation.
- Effective Java (Joshua Bloch) — Item về Encapsulation.
- Clean Code (Robert C. Martin) — Chương: Meaningful Encapsulation.
- Java SE 8 Documentation: https://docs.oracle.com/javase/8/docs/
- Baeldung — Guide to Encapsulation in Java: https://www.baeldung.com/java-encapsulation
- GeeksforGeeks — Encapsulation in Java: https://www.geeksforgeeks.org/encapsulation-in-java/
- TutorialsPoint — Java - Encapsulation: https://www.tutorialspoint.com/java/java_encapsulation.htm
- Design Patterns: Elements of Reusable Object-Oriented Software (GoF book)

## Kết luận

Tính đóng gói là nền tảng then chốt giúp OOP nói chung, và Java nói riêng, tạo ra các hệ thống phần mềm an toàn, dễ mở rộng, dễ bảo trì. Hiểu đúng và vận dụng hiệu quả encapsulation giúp nâng cao chất lượng code và khả năng kiểm soát hệ thống.