ENCAPSULATION TRONG JAVA OOP — TOÀN DIỆN

MỤC LỤC

1. Định nghĩa, nguồn gốc và ý nghĩa lý thuyết của đóng gói

1.1. Định nghĩa

Đóng gói (Encapsulation) là một trong bốn đặc tính cơ bản của lập trình hướng đối tượng (OOP), bên cạnh kế thừa, đa hình và trừu tượng hóa. Đóng gói là kỹ thuật gói gọn dữ liệu (biến, thuộc tính) và các phương thức thao tác với dữ liệu đó bên trong một lớp, đồng thời che giấu chi tiết thực thi với thế giới bên ngoài, chỉ cho phép truy cập thông qua các giao diện (method) đã được kiểm soát. Nói cách khác, đóng gói giúp "ẩn" các thành phần bên trong đối tượng, chỉ cung cấp những gì cần thiết cho bên ngoài sử dụng.

> ***Tài liệu tham khảo: Oracle Java Documentation: What is Encapsulation? https://docs.oracle.com/javase/tutorial/java/concepts/encapsulation.html***

1.2. Nguồn gốc lịch sử

Ý tưởng đóng gói bắt nguồn từ lý thuyết lập trình hướng đối tượng, lần đầu tiên giới thiệu trong ngôn ngữ Simula (1967), rồi phổ biến rộng rãi qua C++ và Java. Alan Kay, một trong những người sáng lập khái niệm OOP, nhấn mạnh việc "trao đổi thông tin thông qua thông điệp, chứ không thao túng trực tiếp dữ liệu".

1.3. Ý nghĩa trong lập trình hướng đối tượng

- Tách biệt giao diện và triển khai.
- Giảm phụ thuộc giữa các module.
- Tăng cường an toàn dữ liệu, kiểm soát quyền truy cập.
- Hỗ trợ nguyên lý thiết kế information hiding — "giấu thông tin".

2. Vai trò và tác dụng của tính đóng gói trong phát triển phần mềm

- Bảo vệ dữ liệu: Ngăn chặn truy cập trực tiếp đến dữ liệu nội bộ, tránh các thay đổi không mong muốn từ bên ngoài.
- Che giấu chi tiết cài đặt: Người dùng chỉ cần quan tâm đến "cái gì" đối tượng làm, không cần biết "làm như thế nào".
- Dễ bảo trì, mở rộng: Khi muốn thay đổi logic bên trong, chỉ cần sửa lớp mà không ảnh hưởng đến mã bên ngoài đang sử dụng đối tượng.
- Tăng tính linh hoạt và an toàn: Có thể bổ sung kiểm tra hợp lệ (validation) dữ liệu trong setter. Dễ dàng thay đổi cách lưu trữ/triển khai bên trong đối tượng.

Tóm lại: Đóng gói là nền tảng cho sự an toàn, mở rộng và bảo trì phần mềm lớn.

3. Hiện thực hóa đóng gói trong Java

3.1. Cách triển khai

- Khai báo các biến thành viên là private hoặc protected.
- Cung cấp phương thức truy cập (getter) và thay đổi giá trị (setter) là public hoặc theo nhu cầu.

3.2. Phân tích các mức độ truy cập (Access Modifier)

Modifier	Ý nghĩa (Java)	Phạm vi truy cập
private	Chỉ truy cập trong nội bộ lớp	Nội bộ class
default (package-private)	Truy cập trong cùng package	Package
protected	Truy cập trong cùng package và các lớp con	Package + Subclass
public	Truy cập mọi nơi	Everywhere

Ví dụ minh họa:

```
public class Example {
    private int a;          // chỉ trong class Example
    int b;                  // default: cùng package
    protected int c;        // cùng package + subclass
    public int d;           // bất kỳ đâu
}
```

3.3. Getter và Setter

- Getter: Phương thức để đọc giá trị biến private
- Setter: Phương thức để gán giá trị mới, có thể thêm kiểm tra hợp lệ

4. Ví dụ mã nguồn minh họa

4.1. Ví dụ đơn giản

Encapsulation với getter/setter:

```
public class Person {
    private String name;
    private int age;

    // Getter
    public String getName() {
        return name;
    }

    // Setter
    public void setName(String name) {
        this.name = name;
    }

    // Getter
    public int getAge() {
        return age;
    }

    // Setter có kiểm tra hợp lệ
    public void setAge(int age) {
        if(age >= 0) {
            this.age = age;
        } else {
            throw new IllegalArgumentException("Tuổi không hợp lệ");
        }
    }
}
```

Sử dụng:

```
Person p = new Person();
p.setName("Minh");
p.setAge(25);
```

```
System.out.println(p.getName()); // Minh
```

4.2. Ví dụ nâng cao: Kết hợp với Kế thừa và Đa hình

```
// Lớp cha
public class Account {
    private double balance;
    protected String accountNumber;

    public Account(String accountNumber) {
        this.accountNumber = accountNumber;
        this.balance = 0.0;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if(amount > 0) balance += amount;
    }

    public boolean withdraw(double amount) {
        if(amount > 0 && amount <= balance) {
            balance -= amount;
            return true;
        }
        return false;
    }
}

// Lớp con
public class SavingsAccount extends Account {
    private double interestRate;

    public SavingsAccount(String accountNumber, double interestRate) {
        super(accountNumber);
        this.interestRate = interestRate;
    }

    public void applyInterest() {
```

```
        deposit(getBalance() * interestRate / 100);
    }
}
```

4.3. Phân tích code

- Dữ liệu nhạy cảm được bảo vệ, tránh truy cập và thay đổi trái phép.
- Dễ kiểm soát logic nghiệp vụ qua các phương thức.
- Thay đổi triển khai không ảnh hưởng phía sử dụng.