

The Open/Closed Principle (OCP)

“The software entities should be open for extension, but closed for modification.”

OCP không có nghĩa là "**code đã viết thì tuyệt đối không được sửa**". Mục tiêu là khi hệ thống cần thêm một hành vi mới mà chúng ta **đã lường trước** thì có thể thêm code mới thay vì sửa code cũ đang hoạt động ổn định.

Đặc điểm chính của một module tuân thủ OCP:

- **Open for extension:** Có thể thêm hành vi mới mà không cần sửa code cũ.
- **Closed for modification:** Việc bổ sung tính năng không gây ảnh hưởng đến module đã có.

OCP giải quyết vấn đề gì?

Trong ví dụ này chúng ta xét đến một game có nhiều loại enemy khác nhau.

```
public enum EnemyType
{
    Zombie,
    Robot
}

public void Attack(Enemy enemy)
{
    switch (enemy.Type)
    {
        case EnemyType.Zombie:
            AttackZombie(enemy);
            break;
        case EnemyType.Robot:
            AttackRobot(enemy);
            break;
    }
}
```

Trong ví dụ này, vấn đề sẽ xảy ra khi chúng ta cần thêm các loại enemy khác như Alien, Mutant, Boss... khi đó chúng ta sẽ cần quay cho trở lại `Attack()` để thêm. Nếu vẫn với thiết kế như trên thì chúng ta còn phải sửa cả `Move()`, `TakeDamage()`, `PlayAnimation()`... kéo theo là phải test lại cả enemies cũ vốn đã hoạt động ổn.

Để giải quyết vấn đề trên chúng ta có thể thử thiết kế dưới đây:

```
public interface IEnemy
{
    void Attack();
}

public class Zombie : IEnemy
{
    public void Attack()
    {
        // Zombie attack
    }
}

public class Robot : IEnemy
{
    public void Attack()
    {
        // Robot attack
    }
}
```

Với thiết kế này khi cần thêm enemy chúng ta chỉ cần triển khai interface `IEnemy`. Các enemy khác hay việc sử dụng `Attack()` không thay đổi.

```
public class Alien : IEnemy
{
    public void Attack()
    {
        // Alien attack
    }
}
```

Với thiết kế trên, **hành vi của hệ thống được mở rộng bằng cách thêm các triển khai mới trong khi phần code cũ đã ổn định không bị thay đổi.**

Abstraction và **polymorphism** là những công cụ quan trọng để đạt được điều này. Module cấp cao làm việc với abstraction ổn định, còn các hành vi khác nhau được cung cấp bởi những triển khai của abstraction đó.

Ứng dụng OCP

Với ví dụ trên rất dễ để chúng ta nhầm lẫn việc sử dụng `switch`, `if/else` là vi phạm OCP. Nhưng thực tế không phải như vậy.

Để ứng dụng đúng OCP chúng ta cần biết hệ thống cần được đóng trước loại thay đổi nào và mở với hành vi nào.

Với ví dụ trên, thiết kế như trên là rất tốt nếu enemies hoạt động riêng lẻ và độc lập với nhau. Nhưng nếu GDD yêu cầu ưu tiên Alien luôn đánh trước Robot thì thiết kế interface như trên lại không giải quyết được nhiều vấn đề.

Vậy nên **đừng cố thiết kế một module có thể đóng với mọi loại thay đổi. Điều đó là không thể.**

Áp dụng OCP cũng cần có tính chiến lược. Chúng ta cần xác định những thay đổi có khả năng xảy ra và xây dựng abstraction cho vấn đề đó. Cũng không có nghĩa là chúng ta sẽ trừu tượng hóa mọi thứ. Làm thế đồng nghĩa với việc chúng ta cần bỏ thêm thời gian, công sức và tiền bạc cho những thay đổi sẽ không bao giờ xuất hiện.

Nếu mỗi lần thêm một object, một behaviour hay một policy mới bạn phải quay lại cùng một nhóm hoặc cùng module để sửa thì đó là dấu hiệu để cân nhắc sử dụng OCP.

Tổng kết

OCP sinh ra không phải để ngăn việc sửa code mà là nguyên lý về việc **kiểm soát phạm vi của thay đổi.**

Để áp dụng OCP hãy tự hỏi: "**Loại thay đổi nào có thể thường xuyên xảy ra và làm thế nào để khi nó xảy ra chúng ta chỉ cần thêm code mới mà không phải quay lại sửa code cũ.**"

Revision #2

Created 2026-08-20 09:15:06 UTC by ThanhDV

Updated 2026-08-21 10:16:10 UTC by ThanhDV