

The Liskov Substitution Principle (LSP)

“Subtypes must be substitutable for their base types.”

Đơn gì thì nếu code của bạn đang hoạt động đúng với object `Base` thì việc truyền vào một object `Derived` $\therefore$ `Base` không được làm đoạn code đó gặp vấn đề.

Ví dụ kinh điển Square và Rectangle

Về mặt toán học chúng ta đều biết hình vuông cũng là một hình chữ nhật. Thế nên rất dễ dẫn tới thiết kế như sau:

```
public class Rectangle
{
    public virtual int Width { get; set; }
    public virtual int Height { get; set; }

    public int Area()
    {
        return Width * Height;
    }
}
public class Square : Rectangle
{
    public override int Width
    {
        set
        {
            base.Width = value;
            base.Height = value;
        }
    }

    public override int Height
    {
        set
        {
            base.Width = value;
            base.Height = value;
        }
    }
}
```

Nhìn qua thì thiết kế này vẫn đảm bảo `width == Height`. Nhìn riêng thì có vẻ như 2 class này hoàn toàn hợp lý. Nhưng hãy xem cách sử dụng dưới đây:

```
public int CalculateArea(Rectangle rectangle)
{
    rectangle.Width = 5;
    rectangle.Height = 4;

    return rectangle.Area();
}

public void Main()
{
    Console.WriteLine(CalculateArea(new Rectangle())); // 20
    Console.WriteLine(CalculateArea(new Square())); // 16
}
```

Như ví dụ trên chúng ta có thể thấy kết quả khác nhau và không đúng với `Square`. Hay nói cách khác `Square` không thể thay thế cho `Rectangle`.

Điểm đáng chú ý là nếu chỉ xét riêng `Square` thì nó vẫn là một hình vuông hợp lệ với `width == Height`.

Vấn đề nằm ở **contract** của `Rectangle`. Client sử dụng `Rectangle` có quyền kỳ vọng rằng `rectangle.Width = 5`; chỉ thay đổi `Width` chứ không âm thầm thay đổi cả `Height`. `Square` phá vỡ kỳ vọng đó.

Do đó trong thiết kế hướng đối tượng, quan hệ "Square **IS-A** Rectangle" không nên chỉ được quyết định dựa trên quan hệ ngoài đời thực.

“ LSP thực sự quan tâm tới **IS-A về behavior** về mặt behavior. Nếu behavior của subtype không tương thích với những kỳ vọng hợp lý của client với base type thì tức là inheritance đó không phù hợp.

Contract quan trọng hơn inheritance

Có thể xem mỗi public API là một contract giữa object và client.

Contract thường gồm:

- **Precondition:** điều kiện client phải đáp ứng trước khi gọi.
- **Postcondition:** điều object đảm bảo sau khi thực hiện.

Nói đơn giản **Derived class không được đòi hỏi nhiều hơn và không được cung cấp ít hơn base class**.

Một ví dụ cho việc subtype đòi hỏi precondition mạnh hơn base type:

```
public class Account
{
    public virtual void Withdraw(int amount)
    {
        if (balance < amount) return;
    }
}
```

```

        // Withdraw
    }
}

public class ChildAccount : Account
{
    public override void Withdraw(int amount)
    {
        if (balance < amount || amount > 100) return;

        // Withdraw
    }
}

```

Trong ví dụ này code vốn chạy đúng với `Account` nhưng có thể thất bại với `ChildAccount`.

Một ví dụ khác, chúng ta có `Bird.Fly()` client có quyền kỳ vọng object đó có thể bay nhưng `Penguin`: `Bird` và `Penguin.Fly()` lại không thực hiện được kỳ vọng đó thì tức là hierarchy đang có vấn đề.

LSP liên quan trực tiếp tới OCP

Một dấu hiệu rất phổ biến khi vi phạm LSP là client bắt đầu phải kiểm tra subtype

```

public void Process(Rectangle rectangle)
{
    if (rectangle is Square)
    {
        // Special case for Square
    }
    else
    {
        // Normal Rectangle behavior
    }
}

```

Ban đầu abstraction này được tạo ra để client chỉ cần biết `Rectangle` nhưng giờ client cần phải đọc lại để biết thêm về `Square`. Với mỗi subtype lại cần thêm một nhánh để xử lý riêng. Khi đó abstraction gần như không còn ý nghĩa.

Đây cũng là lý do mà LSP có quan hệ chặt chẽ với OCP. Khi subtype không thể thay thế base type, client thường phải thêm type checking để xử lý ngoại lệ. Mỗi subtype mới buộc client phải sửa và khiến code không còn **closed for modification**.

Tổng kết

LSP không đơn giản là quy tắc về inheritance. Nó là quy tắc về đảm bảo **abstraction thực sự đáng tin cậy**.

Câu hỏi hữu ích khi đánh giá một quan hệ inheritance là: "**Nếu thay Base bằng Derived, những đoạn code đang sử dụng Base có còn hoạt động đúng theo kỳ vọng ban đầu không?**"

Nếu câu trả lời là không thì chưa chắc subtype đã đã thỏa mãn **IS-A về mặt behavior** và đó là điều mà LSP muốn chúng ta tránh.

Revision #2

Created 2026-08-21 09:10:17 UTC by ThanhDV

Updated 2026-08-24 03:56:41 UTC by ThanhDV