SOLID là gì

SOLID là viết tắt của 5 chữ cái đầu trong 5 nguyên tắc thiết kế hướng đối tượng, giúp cho developer viết ra những đoạn code dễ đọc, dễ hiểu, dễ maintain, được đưa ra bởi Bob Martin và Michael Feathers. Việc theo sát 5 nguyên tắc này nói thì để đáp ứng cả 5 nguyên tắc e là điều không đơn giản. 5 nguyên tắc đó bao gồm:

- **S**ingle responsibility priciple (SRP)
- **O**pen/Closed principle (OCP)
- **L**iskov substitution principe (LSP)
- **I**nterface segregation principle (ISP)
- **D**ependency inversion principle (DIP) Trong bài viết này mình sẽ giới thiệu từng nguyên tắc trong 5 nguyên tắc trên cũng như cách áp dụng nó làm tăng chất lượng code trong Ruby

Nguyên lý đầu tiên ứng với chữ `S` trong `SOLID`, có ý nghĩa là một class chỉ nên giữ một trách nhiệm duy nhất. Một class có quá nhiều chức năng sẽ trở nên cồng kềnh và trở nên khó đọc, khó maintain. Mà đối với ngành IT việc requirement thay đổi, cần thêm sửa chức năng là rất bình thường, nên việc code trong sáng, dễ đọc dễ hiểu là rất cần thiết. Để hiểu rõ hơn, ta cũng soi vào đoạn code vi phạm nguyên tắc này

Nguyên lý thứ 2 ứng với chữ `O` trong `SOLID`. Nội dung `Có thể thoải mái mở rộng 1 class nhưng không được sửa đổi bên trong class đó (open for extension but closed for modification)`

Nguyên tắc thứ 3, ứng với chữ `L` trong `SOLID`. Nội dung nguyên tắc này được phát biểu như sau: `Bất cứ instance nào của class cha cũng có thể được thay thế bởi instance của class con của nó mà không làm thay đổi tính đúng đắn của chương trình`

`N`guyên tắc thứ 4 ứng với chữ `I` trong `SOLID`, nội dung nguyên tắc này như sau: `Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface`

nhỏ, với nhiều mục đích cụ thể Client không nên phụ thuộc vào interface mà nó không sử dụng.

Nguyên tắc thứ 5 ứng với chữ D trong SOLID, nội dung nguyên tắc này như sau: 1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction. 2. Abstraction không nên phụ thuộc vào chi tiết, mà ngược lại