**SOLID**

**I. SOLID Là Gì?**

**SOLID** là viết tắt của 5 chữ cái đầu trong 5 nguyên tắc thiết kế hướng đối tượng. Giúp cho lập trình viên viết ra những đoạn code dễ đọc, dễ hiểu, dễ maintain. Nó được đưa ra bởi [Robert C. Martin](http://www.goodreads.com/author/show/45372.Robert_C_Martin) và Michael Feathers. 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)

**II. Đi Sâu Vào SOLID**

1. **S**ingle responsibility priciple

- Mỗi lớp chỉ nên chịu trách nhiệm cho một nhiệm vụ cụ thể -

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.

2. **O**pen/Closed principle

- Không được sửa đổi một class có sẵn nhưng có thểmở rộng bằng kế thừa -

mỗi khi ta muốn thêm chức năng cho chương trình, chúng ta nên viết class mới mở rộng class cũ (bằng cách kế thừa hoặc sở hữu class cũ) chứ không nên sửa đổi class cũ. Việc này dẫn đến tình trạng phát sinh nhiều class, nhưng chúng ta sẽ không cần phải test lại các class cũ nữa, mà chỉ tập trung vào test các class mới, nơi chứa các chức năng mới.

3. **L**iskov substitution principe

- Trong một chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình -

4. **I**nterface segregation principle

- Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể -

Để dễ quản lý và bảo trì hơn, đỡ dư thừa vì nhiều class không dùng hết các method của interface

5. **D**ependency inversion principle

1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.

2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại.( Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

Có thể hiểu nguyên lí này như sau: những thành phần trong 1 chương trình chỉ nên phụ thuộc vào những cái trừu tượng (abstraction). Những thành phần trừu tượng không nên phụ thuộc vào các thành phần mang tính cụ thể mà nên ngược lại.

Những cái trừu tượng (abstraction) là những cái ít thay đổi và biến động, nó tập hợp những đặc tính chung nhất của những cái cụ thể. Những cái cụ thể dù khác nhau thế nào đi nữa đều tuân theo các quy tắc chung mà cái trừu tượng đã định ra. Việc phụ thuộc vào cái trừu tượng sẽ giúp chương trình linh động và thích ứng tốt với các sự thay đổi diễn ra liên tục.