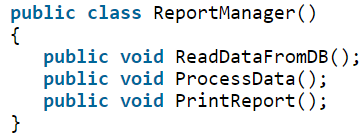
Nguyên lý thiết kế OOP SOLID giúp code dễ refactor, maintain hơn, tuy nhiên lại viết dài hơn

1. **Single responsibility**

Một class chỉ nên giữ 1 trách nhiệm duy nhất

* Chỉ có thể sửa đổi class với 1 lý do duy nhất

VD:



Class này giữ 3 trách nhiệm: Đọc dữ liệu từ DB, xử lý dữ liệu, in kết quả.

Nếu thay đổi 1 trong 3 thì sẽ thay đổi class này 🡺 càng về sau thì sẽ càng phình to ra

* Theo nguyên lý thì phải tách ra làm 3 class riêng

1. **Open/closed**

Có thể thoải mái mở rộng 1 class, nhưng không được sửa đổi bên trong class đó

VD:

Khi muốn thêm chức năng cho project thì nên viết class mới là mở rộng của class cũ (extend hoặc has-a)

1. **Liskov substitution**

Trong program, các object của SubClass có thể thay thế SuperClass mà không làm thay đổi tính đúng đắn của chương trình

VD:

Các class VịtBầu, VịtXiêm có thể kế thừa class Vịt, chương trình chạy bình thường.

Tuy nhiên nếu ta viết class VịtChạyPin, cần pin mới chạy được. Khi class này kế thừa class Vịt, vì không có pin không chạy được, sẽ gây lỗi.

Đó là 1 trường hợp vi phạm nguyên lý này.

1. **Interface segregation**

Chia Interface lớn ra thành nhiều Interface nhỏ với các mục đích cụ thể

1. **Dependency inversion**

Các module cấp cao không nên phụ thuộc vào các modules cấp thấp (mối quan hệ has-a).

* Cả 2 nên phụ thuộc vào abstraction/interface.

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.



* **Reference**

<https://toidicodedao.com/2015/03/24/solid-la-gi-ap-dung-cac-nguyen-ly-solid-de-tro-thanh-lap-trinh-vien-code-cung/>

<https://blogs.msdn.microsoft.com/cdndevs/2009/07/15/the-solid-principles-explained-with-motivational-posters/>