**Interface Segregation Principle**

**Nội dung:**

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

**Diễn giải:**

Giả như ta có 1 interface lớn, khoảng 100 methods. Việc implement sẽ rất vất vả vì các class implement interface này sẽ bắt buộc phải phải thực thi toàn bộ các method của interface, ngoài ra còn có thể dư thừa vì 1 class không cần dùng hết 100 method. Khi tách interface ra thành nhiều interface nhỏ, gồm các method liên quan tới nhau, việc implement và quản lý sẽ dễ hơn.

Hình dung cụ thể khi thiết kế 1 lớp Ultilities, sẽ có hàng trăm các method khác nhau, các method này hỗ trợ các tính toán và xử lí bổ trợ liên quan tới user, report, database… Khi method trở nên quá nhiều thì chúng ta nên tách interface lớn thành các interface nhỏ hơn, ví dụ như: UserUltilities, ReportUltilities, DBUltilities,… như vậy sẽ dễ dàng quản lí và sử dụng hơn.

Trong thực tế, bằng việc tách ra các interface nhỏ hơn, ta sẽ dễ dàng áp dụng các quy tắc khác trong SOLID hơn, điều đó làm code trở nên trong sáng và hiệu quả hơn.

**Ví dụ làm rõ:**

Xây dựng 1 interface *Phone*, các class *SmartPhone*, *PhoneBox* implement *Phone*.

interface Phone

{

public function call();

public function sms();

public function picture();

}

class SmartPhone implements Phone

{

public function call()

{

// do call

}

public function sms()

{

// send sms

}

public function picture()

{

// take picture

}

}

class PhoneBox implements Phone

{

public function call()

{

// do call

}

public function sms()

{

throw new Exception("PhoneBox can't send SMS");

}

public function picture()

{

throw new Exception("PhoneBox can't take picture");

}

}

Nếu class *Phone* có thêm các phương thức mới thì các class *SmartPhone*, *PhoneBox* sẽ phải thay đổi theo trong khi nó không cần hoặc không thế thực hiện.

Giải pháp:

* Chia nhỏ interface *Phone* thành những interface mang nhiệm vụ đặc thù.
* Các lớp *SmartPhone*, *PhoneBox* implement interface đặc thù mà nó cần.

interface Callable()

{

public function call();

}

interface Smsable()

{

public function sms();

}

interface Pictureable()

{

public function sms();

}

class SmartPhone implements Callable, Smsable, Pictureable

{

public function call()

{

// do call

}

public function sms()

{

// send sms

}

public function picture()

{

// take picture

}

}

class PhoneBox implements Callable

{

public function call()

{

// do call

}

}

Như vậy khi mở rộng ta thêm các interface đặc thù với chức năng mở rộng thì sẽ không ảnh hưởng đến các lớp dẫn xuất.

**Kết luận:**

Interface Segregation Principle (ISP) có mối liên hệ với Open - Close Principle (OCP). Vi phạm ISP có khả năng dẫn tới sự vi phạm OCP.

Những lớp trừu tượng chứa quá nhiều thuộc tính và chức năng gọi là những lớp bị ô nhiễm (polluted). Các lớp dẫn xuất phụ thuộc vào polluted interface làm tăng sự kết dính giữa các thực thể. Khi nâng cấp, sửa đổi đòi hỏi các interface này thay đổi, các lớp dẫn xuất này buộc phải thay đổi theo, điều này vi phạm nguyên lý OCP.

**Nguồn tham khảo**

<http://code4fun.vn/code4fun/page/17/>

<http://sieudaochichcode.com/2017/08/19/solid-tim-hieu-solid-de-tro-thanh-developer-gioi/>

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

<https://nhungdongcodevui.com/2017/04/13/solid-la-gi-nguyen-tac-4-chia-nho-interface-interface-segregation-principle-isp/>