**Liskov Substitution Principle**

**Nội dung:**

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.

**Ví dụ làm rõ:**

**Ví dụ 1:**

Hình vuông là hình chữ nhật đặc biệt có chiều cao bằng chiều rộng. Vậy class *Square* có phải con của class *Rectangle*?

Xem xét mối quan hệ kế thừa giữa hình vuông và hình chữ nhật như sau:

class Rectangle

{

protected $m\_width;

protected $m\_height;

public function setWidth(int $width) {

$this->m\_width = $width;

}

public function setHeight(int $height) {

$this->m\_height = $height;

}

public function getWidth() {

return $this->m\_width;

}

public function getHeight() {

return $this->m\_height;

}

public function getArea() {

return $this->m\_width \* $this->m\_height;

}

}

class Square extends Rectangle

{

public function setWidth(int $width) {

$this->m\_width = $width;

$this->m\_height = $width;

}

public function setHeight(int $height) {

$this->m\_height = $height;

$this->m\_width = $height;

}

}

Theo Liskov Substitution Principle thì class *Square* có thể thay thế class *Rectangle* nên ta thực hiện việc test như sau:

class Test

{

public function checkArea(Rectangle $r)

{

$r->setWidth(10);

$r->setHeight(5);

if($r->getArea() == 50) {

return 'true';

}

return 'false';

}

}

$test = new Test;

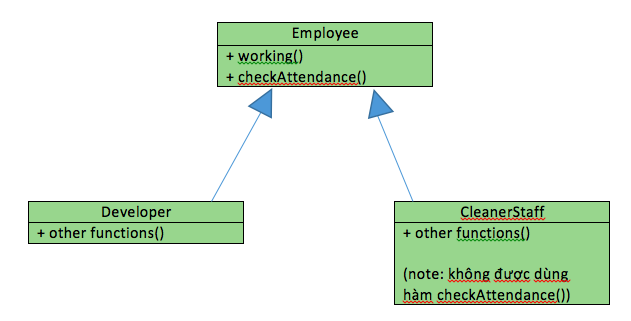
echo 'Test with Rectangle: '.$test->checkArea(new Rectangle).PHP\_EOL;// Test with Rectangle: true

echo 'Test with Square: '.$test->checkArea(new Square).PHP\_EOL;// Test with Square: false

Phương thức *Test* hoạt động đúng (50) khi $r là một thể hiện của *Rectangle* nhưng khi thay thế $r là thể hiện của *Square* thì kết quả là false (25). Lớp Square đã làm mất đi tính đúng đắn của chương trình.

Vậy hình vuông không phải là 1 hình chữ nhật? Xét về mặt hành vi thì hình vuông không phải là hình chữ nhật vì hành vi của hình vuông không thỏa mãn yêu cầu của hàm checkArea.

**Ví dụ 2:**



*Trường hợp vi phạm Liskov Substitution Principle*

Giả sử có công ty sẽ điểm danh vào mỗi buổi sáng, và chỉ có các nhân viên thuộc biên chế chính thức mới được phép điểm danh. Ta bổ sung phương thức *checkAttendance()* vào class *Employee*.

Hình dung có một trường hợp sau: công ty thuê một nhân viên lao công để làm vệ sinh văn phòng, mặc dù là một người làm việc cho công ty nhưng do không được cấp số ID nên không được xem là một nhân viên bình thường, mà chỉ là một nhân viên thời vụ, do đó sẽ không được điểm danh.

Nguyên tắc này nói rằng: Nếu chúng ta tạo ra một class *cleanerStaff* kế thừa từ class *Employee*, và implement hàm *working()* cho class này, thì mọi thứ đều ổn. Tuy nhiên class này cũng vẫn sẽ có hàm *checkAttendance()* để điểm danh, mà như thế là sai quy định dẫn đến chương trình bị lỗi. Như vậy, thiết kế class *cleanerStaff* kế thừa từ class *Employee* là không được phép. Có nhiều cách để giải quyết tình huống này ví dụ như tách hàm *checkAttendance()* ra một interface riêng và chỉ cho các class *Developer*, *Tester* và *Salesman* implement.

**Kết luận:**

Trong đời sống, A là B (hình vuông là hình chữ nhật) không có nghĩa là class A nên kế thừa class B.

Trong lập trình chỉ cho class A kế thừa class B khi class A thay thế được cho class B. Cần xem xét các đối tượng về mặt hành vi chứ không nên sử dụng những mối quan hệ trong đời thật.

**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/2017/04/08/solid-la-gi-nguyen-tac-3-tinh-kha-di-thay-the-liskov-substitution-principle-lsp/>