### **PHP OPP**

### **Giới thiệu về OOP**

* OOP viết tắt của Object-Oriented Programming – Lập trình hướng đối tượng ra đời giải quyết các vấn đề mà lập trình truyền thống gặp phải.
* Trong lập trình hướng đối tượng OOP, có hai thuật ngữ rất quan trọng là lớp (class) và đối tượng (object).
* Class là cái tạo ra nhiều Object có cùng thuộc tính, tính chất của Class đó.
* OBject: được tạo ra từ Class và mang những đặc tính kế thừa từ Class

### **1. Tính chất của lập trình hướng đối tượng**

* Tính trừu tượng (abstraction).
* Tính kế thừa (inheritance).
* Tính đóng gói (encapsulation).
* Tính đa hình (polymorphism).

### **2. Ưu điểm của lập trình hướng đối tượng**

* Dễ dàng quản lý code khi có sự thay đổi chương trình.
* Dễ mở rộng dự án.
* Tiết kiệm được tài nguyên đáng kể cho hệ thống.
* Có tính bảo mật cao.
* Có tính tái sử dụng cao.

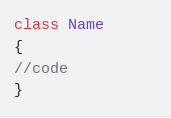
### **3. *Đối tượng (object):***

* Trong lập trình hướng đối tượng, đối tượng được hiểu như là 1 thực thể: người, vật hoặc 1 bảng dữ liệu, . . .
* Một đối tượng bao gồm 2 thông tin: thuộc tính và phương thức:
  + Thuộc tính chính là những thông tin, đặc điểm của đối tượng. Ví dụ: một người sẽ có họ tên, ngày sinh, màu da, kiểu tóc, . . .
  + Phương thức là những thao tác, hành động mà đối tượng đó có thể thực hiện. Ví dụ: một người sẽ có thể thực hiện hành động nói, đi, ăn, uống, . . .

### **4. *Lớp (class):***

* Các đối tượng có các đặc tính tương tự nhau được gom lại thành 1 lớp đối tượng.
* Bên trong lớp cũng có 2 thành phần chính đó là thuộc tính và phương thức.
* Ngoài ra, lớp còn được dùng để định nghĩa ra kiểu dữ liệu mới.

Và để khai báo nó trong PHP thì chúng ta sử dụng cú pháp sau:



Lớp là một khuôn mẫu còn đối tượng là một thể hiện cụ thể dựa trên khuôn mẫu đó.

* Để dễ hiểu hơn mình sẽ lấy một ví dụ thực tế:
  + Các thông tin, đặc điểm như 4 chân, 2 mắt, có đuôi, có chiều cao, có cân nặng, màu lông . . .
  + Các hành động như: kêu meo meo, đi, ăn, ngủ, . . .
* Như vậy mọi động vật thuộc loài mèo sẽ có những đặc điểm như trên.
* Đối tượng chính là một con mèo cụ thể nào đó.

### **5. Các tính chất của lập trình hướng đối tượng**

### **5.1. *Tính trừu tượng (abstraction)***

Trừu tượng hóa là quá trình đơn giản hóa một đối tượng mà trong đó chỉ bao gồm những đặc điểm quan tâm và bỏ qua những đặc điểm chi tiết nhỏ. Quá trình trừu tượng hóa dữ liệu giúp ta xác định được những thuộc tính, hành động nào của đối tượng cần thiết sử dụng cho chương trình.

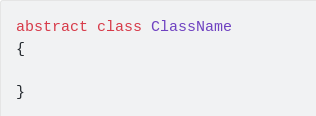
***5.1.1. Abstract class***

Lớp Abstract sẽ định nghĩa các phương thức (hàm) mà từ đó các lớp con sẽ kế thừa nó và Overwrite lại (tính đa hình).

Đối với lớp này thì nó sẽ có các điểm khác sau:

* Các phương thức ( hàm ) khi được khai báo là abstract thì chỉ được định nghĩa chứ không được phép viết code xử lý trong phương thức.
* Trong abstract class nếu không phải là phương thức abstract thì vẫn khai báo và viết code được như bình thường.
* Phương thức abstract chỉ có thể khai báo trong abstract class.
* Các thuộc tính trong abstract class thì không được khai báo là abstract.
* Không thể khởi tạo một abstract class.
* Vì không thể khởi tạo được abstract class nên các phương thức được khai báo là abstract chỉ được khai báo ở mức độ protected và public.
* Các lớp kế thừa một abstract class phải định nghĩa lại tất cả các phương thức trong abstract class đó ( nếu không sẽ bị lỗi).

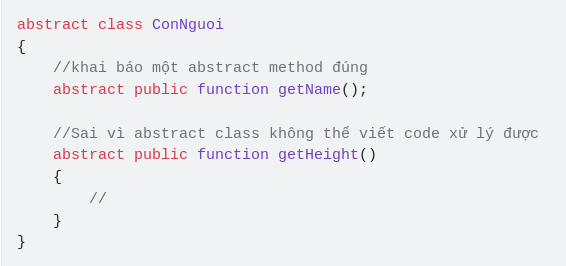
Và để khai báo một abstract class ta dùng cú pháp sau:

****

Cú pháp khai báo một phương thức abstract:

**abstract visibility function methodName();**

Trong đó: visibility là một trong 2 từ khóa public, protected hoặc có thể bỏ trống (bỏ trống thì là public), methodName là tên của phương thức chúng ta cần khai báo.



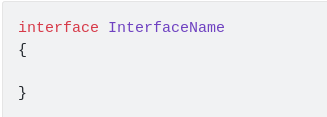
*Ví dụ 2: Phải định nghĩa lại các phương thức abstract của abstract class đó khi kế thừa.*

******

***5.1.2. Interface***

*Interface trong hướng đối tượng là một khuôn mẫu, giúp cho chúng ta tạo ra bộ khung cho một hoặc nhiều đối tượng và nhìn vào interface thì chúng ta hoàn toàn có thể xác định được các phương thức và các thuộc tính cố định (hay còn gọi là hằng) sẽ có trong đối tượng implement nó.*

*Để khai báo interface trong PHP chúng ta dùng cú pháp:*

******

*Tính chất của interface*

* *Interface không phải là một đối tượng.*
* *Trong interface chúng ta chỉ được khai báo phương thức chứ không được định nghĩa chúng.*
* *Trong interface chúng ta có thể khai báo được hằng nhưng không thể khai báo biến.*
* *Một interface không thể khởi tạo được (vì nó không phải là một đối tượng).*
* *Các lớp implement interface thì phải khai báo và định nghĩa lại các phương thức có trong interface đó.*
* *Một class có thể implements nhiều interface.*
* *Các interface có thể kế thừa lẫn nhau.*

***5.1.3. So sánh giữa interface và abstract class***

***Những điểm giống nhau giữa interface và abstract class:***

* *Đều không thể khởi tạo đối tượng.*
* *Đều có thể chứa phương thức abstract*

***Những điểm khác nhau:***

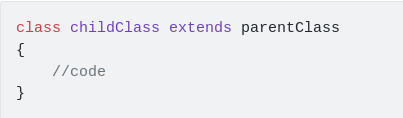
| *Interface* | *Abstract class* |
| --- | --- |
| *Chỉ có thể kế thừa nhiều interface khác.* | *Có thể kế thừa từ 1 lớp và nhiều interface.* |
| *Chỉ chứa các khai báo và không có phần nội dung* | *Có thể chứa các thuộc tính thường và các phương thức bình thường bên trong.* |
| *Không có constructor và cũng không có destructor.* | *Có constructor và destructor.* |
| *Phạm vi truy cập mặc định là public* | *Có thể xác định modifier.* |
| *Dùng để định nghĩa 1 khuôn mẫu hoặc quy tắc chung.* | *Dùng để định nghĩa cốt lõi của lớp, thành phần chung của lớp và sử dụng cho nhiều đối tượng cùng kiểu.* |
| *Cần thời gian để tìm phương thức thực tế tương ứng với lớp dẫn đến thời gian chậm hơn 1 chút.* | *Nhanh hơn so với interface.* |
| *Khi ta thêm mới 1 khai báo. Ta phải tìm hết tất cả những lớp có thực thi interface này để định nghĩa nội dung cho phương thức mới.* | *Đối với abstract class, khi đĩnh nghĩa 1 phương thức mới ta hoàn toàn có thể định nghĩa nội dung phương thức là rỗng hoặc những thực thi mặc định nào đó. Vì thế toàn bộ hệ thống vẫn chạy bình thường.* |

### **5.2. *Tính kế thừa (inheritance)***

*Tính kế thừa trong lập trình hướng đối tượng cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa. Lớp được kế thừa còn được gọi là lớp cha và lớp kế thừa được gọi là lớp con.*

*Điều này cho phép các đối tượng có thể tái sử dụng hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại.*

*Trong PHP để khai báo kế thừa từ một lớp cha chúng ta sử dụng từ khóa extends theo cú pháp:*

******

### **5.3. *Tính đóng gói (encapsulation)***

*Tính đóng gói là tính chất không cho phép người dùng hay đối tượng khác thay đổi dữ liệu thành viên của đối tượng nội tại. Chỉ có các hàm thành viên của đối tượng đó mới có quyền thay đổi trạng thái nội tại của nó mà thôi. Các đối tượng khác muốn thay đổi thuộc tính thành viên của đối tượng nội tại, thì chúng cần truyền thông điệp cho đối tượng, và việc quyết định thay đổi hay không vẫn do đối tượng nội tại quyết định. Trong PHP việc đóng gói được thực hiện nhờ sử dụng các từ khoá public, private và protected:*

* *Private là giới hạn hẹp nhất của thuộc tính và phương thức trong hướng đối tượng. Khi các thuộc tính và phương thức khai báo với giới hạn này thì các thuộc tính phương thức đó chỉ có thể sử dụng được trong class đó, bên ngoài class không thể nào có thể sử dụng được nó kể cả lớp kế thừa nó cũng không sử dụng được, nếu muốn lấy giá trị hoặc gán giá trị ở bên ngoài class thì chúng ta phải thông qua hai hàm SET và GET.*
* *Khác với private một chút thì các phương thức và thuộc tính khi khai vào với visibility là protected thì chúng ngoài được sử dụng trong class đó ra thì class con kết thừa từ nó cũng có thể sử dụng được, như bên ngoài class không có thể sử dụng được.*
* *Đây là visibility có mức độ truy cập rộng nhất trong hướng đối tượng, khi một thuộc tính hay phương thức sử dụng visibility này thì chúng ta có thể tác động vào thuộc tính hay phương thức đó từ cả trong lẫn ngoài class. Thông thường khi không khai báo visibility thì chương trình dịch tự nhận nó là public nhưng để cho đúng chuẩn thì mọi người lên khai báo từ khóa này vào thay vì bỏ trống.*

### **5.4. *Tính đa hình (polymorphism)***

*Tính đa hình trong lập trình hướng đối tượng là sự đa hình của mỗi hành động cụ thể ở những đối tượng khác nhau. Ví dụ hành động ăn ở các loài động vật hoàn toàn khác nhau như: con heo ăn cám, hổ ăn thịt, con người thì ... ăn hết =)).*

*Đó là sự đa hình trong thực tế, còn đa hình trong lập trình thì được hiểu là Lớp Con sẽ viết lại những phương thức ở lớp cha (ovewrite).*

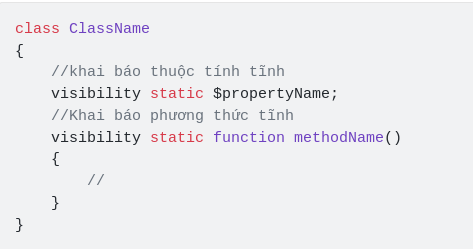
*Các class cùng implement một interface nhưng chúng có cách thức thực hiện khác nhau cho các method của interface đó.*

### ***6. Thế nào là một hàm static. Phân biệt cách dùng từ khoá static::method() với self::method()***

***6.1. Static là gì?***

*Static trong lập trình hướng đối tượng là một thành phần tĩnh (có thể là thuộc tính hoặc phương thức) mà nó hoạt động như một biến toàn cục, dù cho nó có được xử lý ở trong bất kỳ một file nào đi nữa (trong cùng một chương trình) thì nó đều lưu lại giá trị cuối cùng mà nó được thực hiện vào trong lớp.*

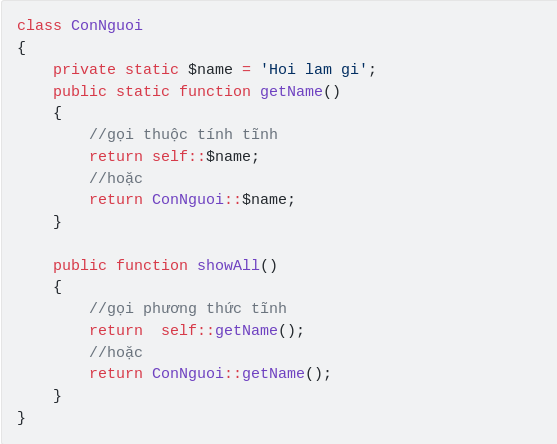
*Để khai báo một thuộc tính hay một phương thức là static thì chúng ta chỉ việc thêm từ khóa static sau vibsility.*

******

*Khi bạn khai báo một thuộc tính hay một phương thức ở dạng static thì bạn sẽ không thể gọi bằng cách thông thường là dùng từ khóa this được nữa mà sẽ có các cách gọi khác như sau:*

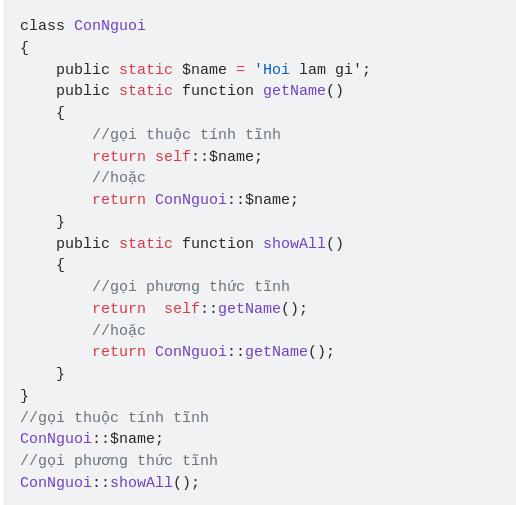
*Gọi phương thức và thuộc tính tĩnh trong class*

*Để gọi phương thức và thuộc tính tĩnh trong class thì chúng ta có thể sử dụng cú pháp selft::ten hoặc ClassName::ten hoặc static::ten.*

******

*Gọi phương thức và thuộc tính tĩnh ngoài class*

*Để gọi phương thức tĩnh ở bên ngoài class thì chúng ta gọi theo cú pháp ClassName::tenPhuongThuc(), ClassName::$tenthuoctinh.*

******

***6.2. Phân biệt cách dùng từ khoá static::method() với self::method()***

* *Self: Truy xuất đến class khai báo nó.*
* *Static: Truy xuất đến đối tượng hiện tại.*

### **7.Trait**

**7.1. Trait là gì???**

PHP hay Ruby đều là những ngôn ngữ hướng đối tượng chỉ hỗ trợ single inheritance(đơn kế thừa). Để khắc phục những giới hạn của đơn kế thừa trong việc sử dụng lại source code, từ PHP 5.4 trở đi PHP hỗ trợ Traits là cơ chế giúp cho lập trình viên có thể sử dụng lại các phương thức từ các class khác nhau một cách dễ dàng hơn. Một trait tương tự như là 1 class nhưng chỉ nhằm mục đích nhóm chức năng lại. Và trait không thể khởi tạo giống class và trait sinh ra để bổ sung cho kế thừa truyền thống. Thay vì phải kế thừa 1 class hay interface để sử dụng lại 1 nhóm chức năng, thì với trait bạn không cần phải kế thừa vẫn có thể sử dụng được

***Các đặc điểm của Traits:***

* Traits có chức năng gom lại các phương thức và thuộc tính mà chúng ta muốn sử dụng lại nhiều lần.
* Traits như một abstract class ( đều không thể khởi tạo được) nhưng không hoàn toàn giống nhau.
* Các phương thức trong Traits có thể bị override lại trong class sử dụng nó.

***Ưu điểm của Traits:***

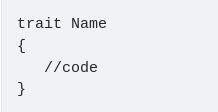
* Giảm việc lặp code đáp ứng được nguyên tắc(DRY - Don't Repeat Yoursefl).
* Khắc phục được điểm yếu đơn kế thừa của PHP.

***Nhược điểm của Traits:***

* Traits đó là sẽ gây khó khăn có chúng ta đọc được các phương thức từ một class có sử dụng traits.
* Traits tạo ra việc bạn có thể dễ viết các classes cồng kềnh và có quá nhiều chức năng. Một Trait bản chất là một cách để copy và paste code giữa các classes. Bởi có một cách đơn giản để thêm một nhóm các methods vào một class.

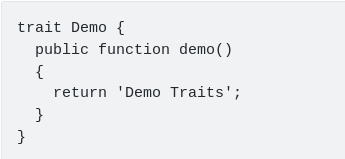
**7.2. Khai báo Trait**

Để khai báo một trait trong PHP ta sử dụng cú pháp:



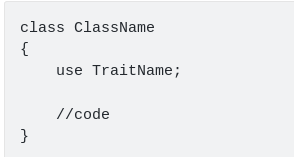
Trong đó: Name là tên của trait các bạn muốn đặt.

**Ví dụ: Khai báo trait demo:**

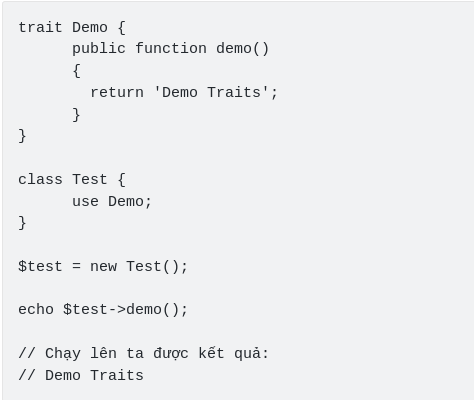


**7.3. Sử dụng Trait trong class**

Để sử dụng trait trong class thì các bạn chỉ cần sử dụng cú pháp:

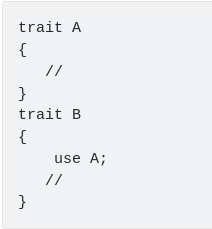


Ví dụ khai báo một trait và sử dụng trong class



**7.4. Trait lồng nhau**

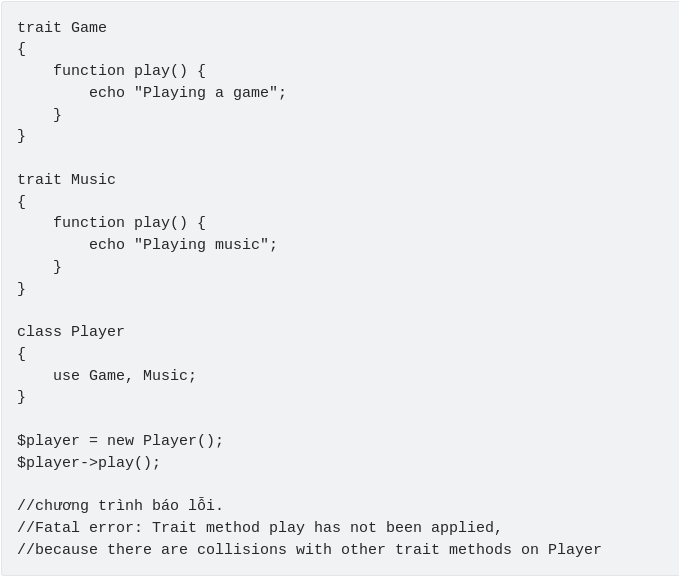
Cũng như các cấu trúc điều khiển hay vòng lặp thì bạn cũng có thể sử dụng trait lồng nhau.



Lúc này khi bạn gọi trait B là bạn cũng có thể sử dụng được chức năng của trait A.

**7.5. Ưu tiên phương thức trong Trait**

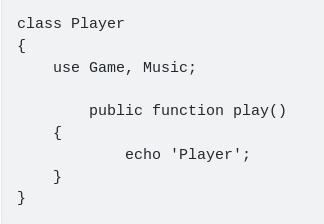
Giả sử như bạn có hai trait như sau:



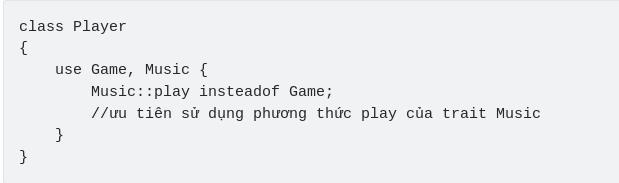
Bây giờ nếu như bạn gọi 2 trait trên vào trong một class và sử dụng thì lập tức chương trình lỗi ngay.

Giờ đây sẽ có 2 giải pháp để sử lỗi trên như sau:

**Cách 1:** Là bạn sẽ phải override lại phương thức trùng tên đó ở trong class sử dụng.



**Cách 2:** Xử dụng insteadof để xét độ ưu tiên cho phương thức bạn muốn sử dụng.

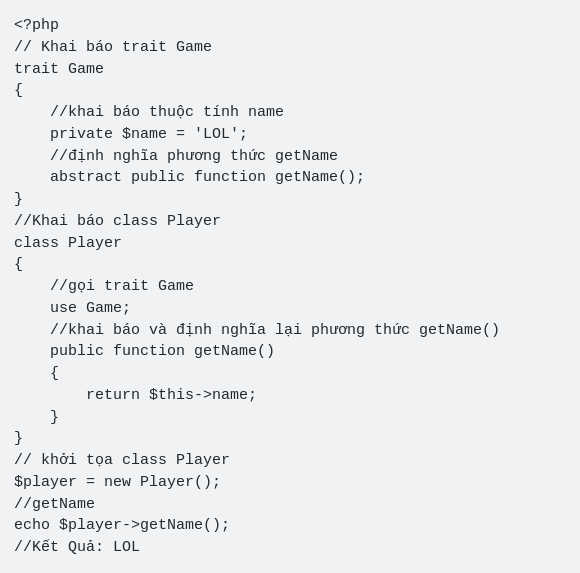


**7.6. Phương thức tĩnh và thuộc tính tĩnh trong trait**

**Kết Luận:** Như vậy Thuộc tính tĩnh trong Trait chỉ ảnh hưởng bên trong một class thôi còn khi gọi nó ở trong một class khác thì hoàn toàn bình thường.

**7.8. Phương thức trừu tượng trong trait.**

Trait cũng hỗ trợ chúng ta sử dụng phương thức trừu tượng



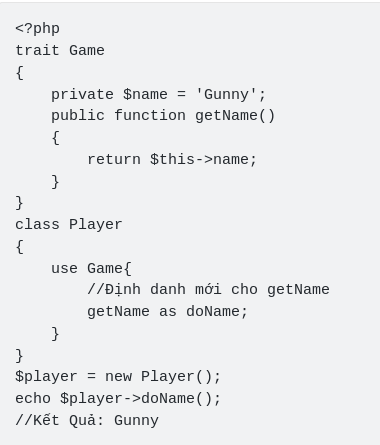
**7.9. Thay đổi visibility của phương thức trait.**

Trong PHP cũng có hỗ trợ chúng ta thay đổi visibility của phương thức trong traits với cú pháp như sau:

**use methodName as visibility;**



**7.10. Tạo định danh mới cho Traits.**



**7.11. So sánh Traits với abstract và interface**

***a) Trait với Interface***

**Giống nhau**: Đều không có thể khởi tạo được mà chỉ có thể khai báo.

**Khác nhau:**

| **Trait** | **Interface** |
| --- | --- |
| Có thể khai báo thuộc tính. | Chỉ có thể khai báo thuộc tính cố định (hằng). |
| Có thể định nghĩa và khởi tạo phương thức. | Chỉ có thể khai báo không thể định nghĩa. |

***b) Trait với Abstract class***

**Giống nhau:** Đều không có thể khởi tạo được mà chỉ có thể khai báo.

**Khác nhau:**

| **Trait** | **Abstract class** |
| --- | --- |
| Có thể lồng nhau một cách đơn giản | Khi muốn sử dụng lẫn nhau phải kế thừa hết sức lằng nhằng. |

### **8. Namespace**

**8.1 Namespace là gì???**

Namespace giúp tạo ra một không gian tên cho hàm và lớp trong lập trình nói chung và trong PHP nói riêng.

**8.2 Khai báo namespace**

Để khai báo namespace ta sử dụng cú pháp sau:

**namespace name**

Trong đó: name là tên của namespace mà bạn muốn đặt.

Chú ý: khi khai báo namespace thì chúng ta phải đặt nó ở phía trên cùng của file.

**8.3 Gọi namespace**

Khi mà một class đã được một namespace định danh thì bạn sẽ không thể gọi theo cách thông thường được nữa mà phải gọi với cú pháp:

**new tenNamespace\tenClass();**

**8.4 Nạp namespace bằng use**

Nếu như bạn không muốn gọi namespace theo cách trên thì bạn sử dụng từ khóa use để nạp theo cú pháp sau:

**use tenNamespace\tenClass;**

****

**8.5 Định danh cho namespace Để định đanh cho namespace ta sử dụng cú pháp sau:**

**use tenNamespace as tenMoi;**

### **9. Magic methods**

**9.1 Magic methods là gì**

Magic methods là các phương thức đặc biệt được tạo ra nhằm giải quyết các vấn đề về sự kiện trong chương trình (cụ thể là với class), và đối với PHP cũng thế.

**9.2 Một số Magic methods trong PHP**

* \_\_construct(): gọi khi khởi tạo đối tượng.
* \_\_destruct(): gọi khi hủy đối tượng.
* \_\_set(): gọi khi ta truyền dữ liệu cho một thuộc tính không được phép truy cập.  
  \_\_get(): khi đọc dữ liệu từ một thuộc tính không được phép truy cập.
* \_\_isset(): được gọi khi gọi hàm isset() hoặc empty() trên một thuộc tính không được phép truy cập.
* \_\_unset(): được gọi khi hàm unset() được sử dụng trong một thuộc tính không được phép truy cập.
* \_\_call():được gọi khi ta gọi một phương thức không được phép truy cập trong phạm vi của một đối tượng.
* \_\_callstatic(): được kích hoạt khi ta gọi một phương thức không được phép truy cập trong phạm vi của một phương thức tĩnh.
* \_\_toString(): phương thức này giúp class chỉ định xem sẽ in ra cái gì khi nó được dùng.
* \_\_invoke():phương thức này được gọi khi một lệnh cố gắng gọi một đối tượng như một hàm.
* \_\_sleep(): được gọi khi serialize() một đối tượng.
* \_\_wakeup: được gọi khi unserialize() đối tượng.
* \_\_set\_state(): được gọi khi chúng ta var\_export() đối tượng.
* \_\_clone(): được sử dụng khi chúng ta clone một object.
* \_\_debugInfo(): được gọi khi chúng ta sử dụng hàm vardump().

### **CÁC PHƯƠNG PHÁP THIẾT KẾ HƯỚNG ĐỐI TƯỢNG (SOLID)**

SOLID là gì? SOLID dịch ra là cứng. Nghe vẫn chả liên quan gì lắm đúng không =))). Nhưng sau đây khi mình tách nó ra và phân tích thì sẽ thấy nó rất quan trọng và không thể thiếu được trên con đường trở thành một lập trình viên "cứng".

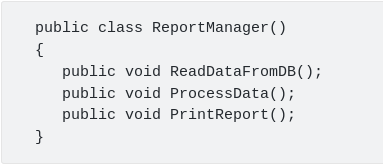
**Nguyên lý đầu tiên:**

#### **S : Single responsibility principle**

Nội dung của nguyên lý này là :

***Một class chỉ nên giữ 1 trách nhiệm duy nhất (Chỉ có thể sửa đổi class với 1 lý do duy nhất)***

*Bạn nhìn thấy bộ dao kia chứ. Tiện lợi nhỉ. Ta có thể cắt với nhiều loại hình vật khác nhau. Nhưng... Nếu nó hỏng đúng một con dao mà ta cần thì sao. Ta sẽ phải tháo cả bộ ra để sửa. Điều này rất phức tạp, có thể ảnh hưởng tới nhiều bộ phận khác nhau. Trong IT cũng vậy, việc các requirement rất hay thay đổi, dẫn tới sự thay đổi code. Nếu một class có quá nhiều chức năng, quá cồng kềnh, việc thay đổi code sẽ rất khó khăn, mất nhiều thời gian, còn dễ gây ảnh hưởng tới các module đang hoạt động khác Ta sẽ làm một ví dụ về sự vi phạm nguyên lý này.*

******

*Class này đang chịu trách nhiệm cho 3 chức năng đó là ReadDataFromDB, ProcessData, PrintReport. Điều này vi phạm nguyên lý đầu tiên của chúng ta. Ta cần phải tách class này ra thành 3 class khác. Tuy là số class tăng lên nhưng việc bảo trì sẽ đơn giản và dễ dàng hơn. Giống như hình ảnh về bộ dao ở trên. Khi ta tách nhỏ các con dao theo mỗi chức năng riêng của nó, thay vì việc sửa một con dao liên quan đến cả bộ, ta sẽ chỉ sửa một mình nó và không làm ảnh hưởng đến các con dao khác.*

***Nguyên lý thứ 2:***

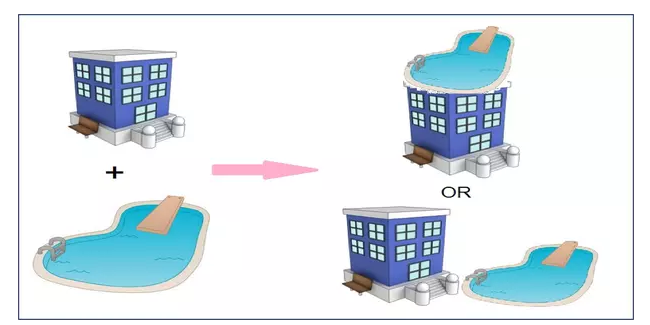
#### ***O : Open/Closed principle***

*Nội dung của nguyên lý này:*

***Có thể thoải mái mở rộng 1 class, nhưng không được sửa đổi bên trong class đó (open for extension but closed for modification)***

*Theo nguyên lý này, 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) không nên sửa đổi class cũ.*

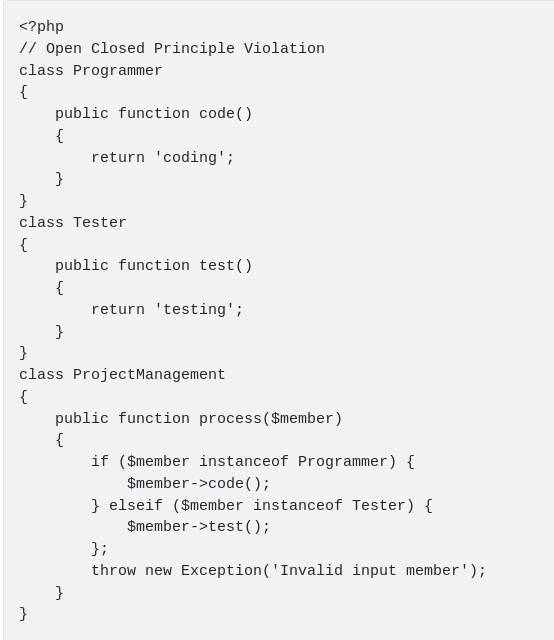
*Thử hình dung rằng “tiện nghi sống” của bạn đang là 1 căn nhà, bây giờ bạn muốn có thêm 1 tính năng là “hồ bơi” để thư giãn. Bạn có 2 cách để làm điều này:*

**

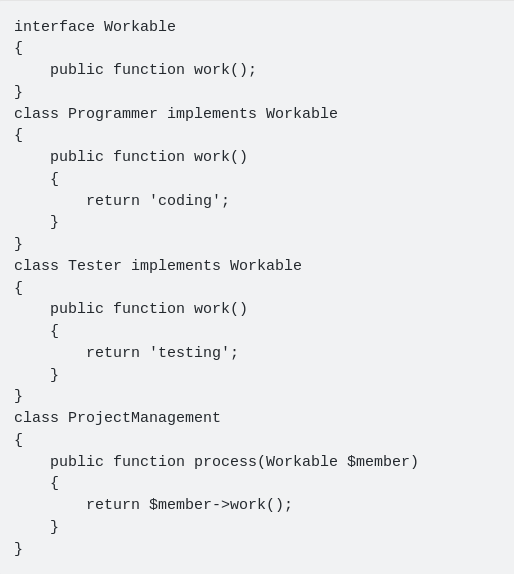
*Cách 1: thay đổi hiện trạng của căn nhà, xây thêm 1 tầng nữa để làm hồ bơi.*

*Cách 2: không làm thay đổi căn nhà đang có, mua thêm 1 mảnh đất cạnh nhà bạn và xây hồ bơi ở đó.*

*Mặc dù cả 2 cách đều giải quyết được vấn đề nhưng cách đầu tiên có vẻ rất thiếu tự nhiên và kì cục. Cách này làm thay đổi hiện trạng của căn nhà, và nếu không cẩn thận có thể làm hư luôn những thứ đang có. Cách thứ 2 an toàn hơn rất nhiều và đáp ứng tốt được nhu cầu muốn có hồ bơi của bạn. Nguyên tắc này có ý rằng: không được thay đổi hiện trạng của các lớp có sẵn, nếu muốn thêm tính năng mới, thì hãy mở rộng bằng cách kế thừa để xây dựng class mới. Làm như vậy sẽ tránh được các tình huống làm hỏng tính ổn định của chương trình đang có. Ví dụ:*

******

*Nhìn vào đây các bạn sẽ thấy code này chạy đúng. Nhưng mấu chốt là ở đoạn hàm xử lý ifvàelsekia. Nếu$member` thuộc một class khác thì sao. Ta sẽ phải xử lý thêm ở trong function process và như thế, mọi thứ càng ngày sẽ càng cồng kềnh. Vì thế, cách giải quyết là tạo một interface, mình sẽ gọi là Workable, sau đó chúng ta sẽ dùng 2 class Tester và Programmer implements interface này.*

******

*Nhìn xem, bây giờ nếu bạn muốn mở rộng class ProjectManagement, bạn chỉ việc tạo thêm các class khác implements từ class Workable mà không cần phải xử lý if else như trên nữa. Cảm giác mọi thứ đã mượt mà hơn rất nhiều .*

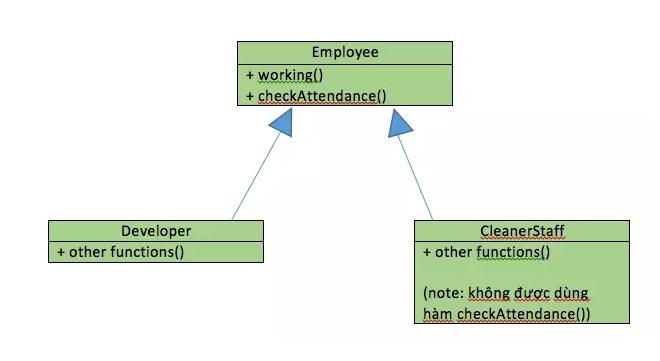
***Nguyên lý thứ 3:***

#### ***L: Liskov substitution principle***

*Nội dung :*

***Các đối tượng kiểu class con có thể thay thế các đối tượng kiểu class cha mà không gây ra lỗi.***

*Ví dụ:*

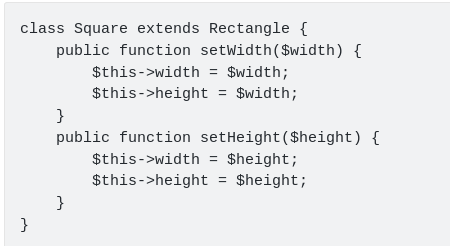
**

*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 lớp 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. Hình ảnh này mô tả một sự vi phạm đến nguyên lý thứ 3. Nếu chúng ta tạo ra một lớp cleanerStaff kế thừa từ lớp Employee, và implement hàm “working()” cho lớp này, thì mọi thứ đều ổn, tuy nhiên lớp 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 lớp cleanerStaff kế thừa từ lớp 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 lớp Developer, Tester và Salesman implements.*

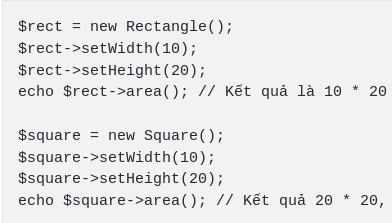
*Ví dụ: Chúng ta tạo một class Rectangle dưới đây:*

******

*Như chúng ta biết, hình vuông là một hình chữ nhật có chiều dài bằng chiều rộng, nên ta sẽ kế thừa từ hình chữ nhật:*

******

*Tính diện tích:*

******

*Ở đây, sau khi $square->setHeight có giá trị. Lập tức nó sẽ set giá trị cho width cũng bằng giá trị của height. Tại sao lại như vậy?*

*Trong hình học, hình vuông là hình chữ nhật, nó là trường hợp đặc biệt của hình chữ nhật. Các phương thức setWidth và setHeight trong Rectangle, nó đúng trong Rectangle nhưng nếu tham chiếu sang Square, hai phương thức này không có ý nghĩa bởi nó được sử dụng để thiết lập cho một đối tượng khác không phải là hình vuông. Trong trường hợp này, Square không tuân theo nguyên lý thay thế Liskov và sự trìu tượng trong kế thừa từ Rectangle là không ổn.*

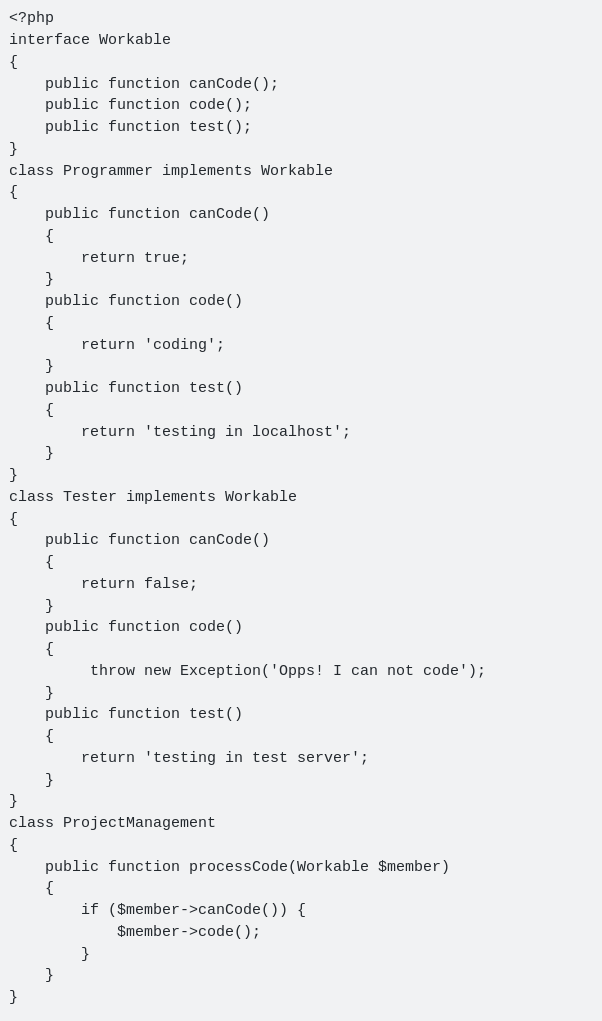
***Nguyên lý thứ 4 :***

#### ***I : Interface segregation principle***

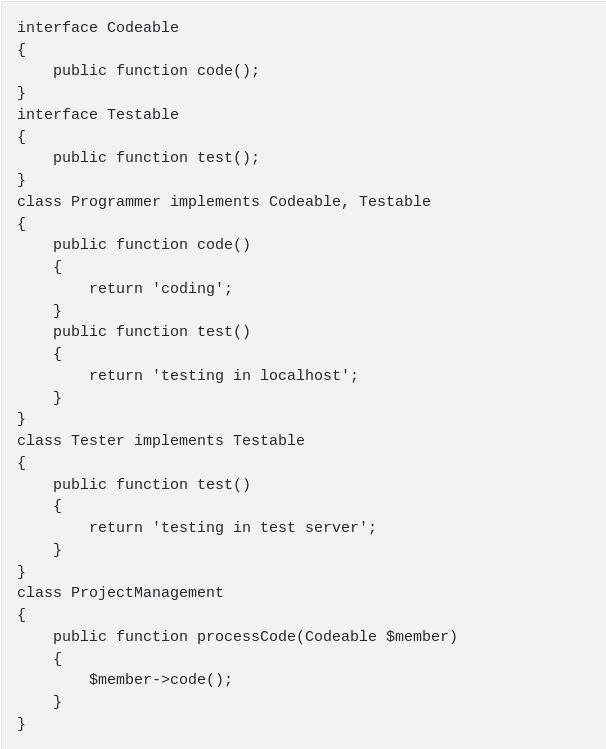
*Nội dung :*

***Một class không nên thực hiện một interface mà nó không dùng đến hoặc không nên phụ thuộc vào một phương thức mà nó không sử dụng. Để làm điều này, thay vì một interface lớn bao trùm chúng ta tách thành nhiều interface khác nhau.***

*Như các bạn đã biết. Một class implements từ một interface sẽ phải thực hiện việc override lại tất cả các phương thức của interface này. Một interface thì có thể có nhiều class implements , và có thể có những phương thức trong interface mà class này không dùng đến. Điều này dẫn dến sự dư thừa và không tối ưu. Cùng xem ví dụ sau:*

******

*Sự dư thừa đã hiện lên, và chúng ta sẽ tối ưu lại bằng cách tách interface tổng thành các interface nhỏ hơn:*

******

***Nguyên lý thứ 5:***

#### ***D : Dependency Inversion Principle***

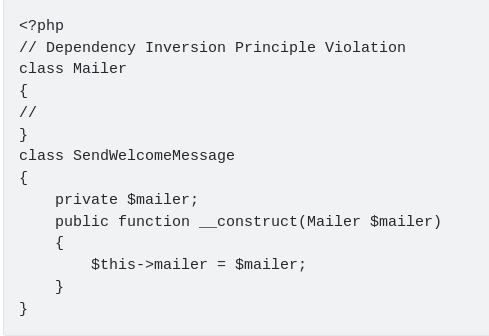
*Nội dung:*

***Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại (Các class giao tiếp với nhau thông qua interface (abstraction), không phải thông qua implementation).***

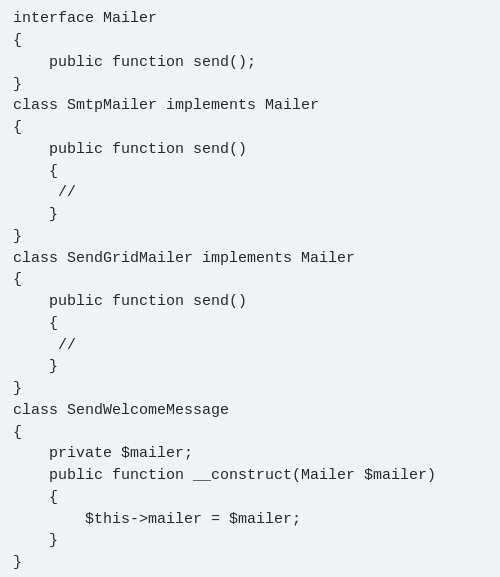
***Giải thích:***

* *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.*

***Ví dụ:***

******

*Trong ví dụ này, chương trình phụ thuộc vào cái cụ thể, chính là $mailer. Đặt vấn đề, nếu chúng ta muốn gửi mail theo nhiều dạng, như Smtp, hay SendGrid thì sao. Đây chính là cái trừu tượng mà mình đã giải thích ở trên. Nó luôn luôn thay đổi nhưng bản chất vẫn là send. Ta sẽ sửa lại một chút để theo nguyên lý này:*

******

*Giờ bạn muốn gửi mail theo loại gì thì sẽ phân chia trong class đó. bạn chỉ việc thêm 1 class mới implements từ interface Mailer và xử lý trong nó. Ngoài ra code cũng rõ ràng và đẹp hơn rất nhiều.*