FastTrack SE™

PHP cơ bản phần 5

Giảng viên :Cao Le Thanh

Nội dung bài học

- Thế nào là lập trình hướng đối tượng?
- Các đặc điểm cơ bản quả lập trình hướng đối tượng là gì?
- Hướng đối tượng được thể hiện như thế nào trong PHP?
- Sự khác biệt giữa Abstract Class và Interface.
- Thế nào là một hàm static. Phân biệt cách dùng từ khoá static::method() với self::method().

Thế nào là lập trình hướng đối tượng?

- Lập trình hướng đối tượng (tiếng Anh: Object-oriented programming, viết tắt: OOP) là kỹ thuật lập trình nhằm vào sự tương tác của các đối tượng. Mỗi đối tượng có những thuộc tính (thông tin lưu trữ), những phương thức xác định chức năng của đối tượng. Bên cạnh đó, đối tượng có khả năng phát sinh ra các sự kiện khi thay đổi thông tin, thực hiện một chức năng hay khi đối tượng khác tác động vào. Tất cả những thuộc tính, phương thức và sự kiện tạo nên cấu trúc hướng đối tượng.
- **Tư tưởng:** Tư tưởng của OOP là xây dựng một chương trình dựa trên sự phối hợp hoạt động của các đối tượng. Một đối tượng bao gồm hai thành phần chính: thông tin lưu trữ và các thao tác xử lý.

Các đặc điểm cơ bản của lập trình hướng đối tượng là gì?

FastTrack SE™

- Lập trình hướng đối tượng được biết tới với 4 tính chất cơ bản:
  - Tính trừu tượng (**Abstraction**)
  - Tính đóng gói (**Encapsulation**)
  - Tính kế thừa (**Inheritance**)
  - Tính đa hình (**Polymorphism**)

Tính trừu tượng

- **Tính trừu tượng:** Chúng ta thường nhầm lẫn giữa lớp (class) và đối tượng (object). Vì vậy chúng ta cần phải phân biệt:
- **Lớp là một tính chất trừu tượng Đối tượng là thể hiện cụ thể của lớp**
- Ví dụ: Bản thiết kế nhà là lớp, ngôi nhà được xây dựng trên bản thiết kế là đối tượng.

Tính đóng gói

- **Tính đóng gói:** Cơ chế đóng gói là phương thức tốt để thực hiện cơ chế che dấu thông tin so với các ngôn ngữ lập trình cấu trúc.
  - Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng.
  - Ngoài ra, đóng gói còn để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy.
- Các mức truy cập:
  - **Public:** Cho phép truy cập và sử dụng đến các phương thức cũng như thuộc tính ở phạm vi trong, ngoài class và nó hỗ trợ sự kế thừa.
  - **Private:** Chỉ cho phép truy cập và sử dụng đến các phương thức cũng như thuộc tính ở phạm vi trong class.
  - **Protected:** Cho phép truy cập và sử dụng đến các phương thức cũng như thuộc tính ở phạm vi trong class và nó hỗ trợ sự kế thừa.

Tính kế thừa

- **Tính kế thừa:** Là khả năng cho phép ta xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có. Lớp đã có gọi là lớp cha, lớp phát sinh gọi là lớp con và đương nhiên lớp con kế thừa tất cả các thành phần của lớp cha mở rộng các thành phần kế thừa cũng như bổ sung thêm các thành phần mới Ví dụ: Ta có lớp người là lớp cha và lớp con là lập trình viên

Tính đa hình

- **Tính đa hình:** Tính đa hình được thể hiện qua việc viết lại các method (hàm) từ class cha thông qua class kế thừa nó hoặc việc triển khai các interface. Hoặc có thể được hiểu là Lớp Con sẽ viết lại những phương thức ở lớp cha (**overwrite**). 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.

Hướng đối tượng được thể hiện như thế nào trong PHP

FastTrack SE™

Tính trừu tượng:

- Tính trừu tượng (**abstraction**) trong lập trình hướng đối tượng giúp giảm sự phức tạp thông qua việc tập trung vào các đặc điểm trọng yếu hơn là đi sâu vào chi tiết.
- Trong **PHP**: Tính trừu tượng được thể hiện qua lớp giao diện (**Interface**) và lớp trừu tượng(**abstract**).

Hướng đối tượng được thể hiện như thế nào trong PHP

FastTrack SE™

- Ví dụ:

```php
<?php

abstract class Hinh
{

    abstract protected function hienThi(){};

}

class HinhChuNhat extends Hinh
{
    public function hienThi() {
        echo "Đây là hình chữ nhật";
    }
}
$class = new HinhChuNhat();
$class->hienThi();
```

Hướng đối tượng được thể hiện như thế nào trong PHP

- **Tính đóng gói:** Trong PHP việc đóng gói được thực hiện nhờ sử dụng các từ khoá public, private và protected:
  - **Public:** Cho phép truy cập (và thay đổi giá trị) của thuộc tính và phương thức ở mọi phạm vi.
  - **Private:** Chỉ cho phép truy cập (hay thay đổi) giá trị của thuộc tính và phương thức ở phạm vi của đối tượng (hoặc lớp).
  - **Protected:** Chỉ cho phép truy cập (hay thay đổi) giá trị của thuộc tính và phương thức ở phạm vi của đối tượng con (hoặc lớp con).

Hướng đối tượng được thể hiện như thế nào trong PHP

FastTrack SE™

❖ Tính kế thừa:

Con người	Lập trình viên
Thuộc tính: Mắt, mũi, miệng,...	Thuộc tính: Mắt, mũi, miệng, **bằng cấp**, **chứng chỉ**,...
Phương thức: Ăn, ngủ, chơi game,...	Phương thức: Ăn, ngủ, chơi game,...

Hướng đối tượng được thể hiện như thế nào trong PHP

FastTrack SE™

- Ví dụ

```php
<?php

// Lớp Cha
class Person
{
    // Thuộc Tính
    var $mat = '';
    var $mui = '';
    var $mieng = '';

    // Hàm, phương thức
    function eat(){
        // lệnh
    }

    function sleep(){
        // lệnh
    }

    function playGame(){
        // lệnh
    }

}
```

```php
// Lớp Con
class Programmer extends Person {
    var $chuyenNganh = '';
    var $bangCap = '';
}
```

Hướng đối tượng được thể hiện như thế nào trong PHP

FastTrack SE™

- **Tính đa hình:** Trong PHP đa hình được hiểu là Lớp Con sẽ viết lại những phương thức ở Lớp Cha (**overwrite**). Tiếp tục ví dụ về hành động ăn của các loài động vật.

```php
// Lớp Cha
class Animal
{
    // Động Vật Ăn
    public function eat()
    {
        echo 'Động Vật Đang Ăn';
    }
}
```

```php
// Lớp Con
class Pig extends Animal
{
    public function eat()
    {
        parent::eat();
        echo 'Con Heo Đang Ăn Cám';
    }
}
```

Sự khác biệt giữa Abstract Class và Interface

- **Abstract class:** Lớp trừu tượng đơn giản được xem như một class cha cho tất cả các Class có cùng bản chất. Do đó mỗi lớp dẫn xuất (lớp con) chỉ có thể kế thừa từ một lớp trừu tượng. Bên cạnh đó nó không cho phép tạo instance, nghĩa là sẽ không thể tạo được các đối tượng thuộc lớp đó.
- **Interface:** Lớp này được xem như một mặt nạ cho tất cả các Class cùng cách thức hoạt động nhưng có thể khác nhau về bản chất. Từ đó lớp dẫn xuất có thể kế thừa từ nhiều lớp Interface để bổ sung đầy đủ cách thức hoạt động của mình (**đa kế thừa - Multiple inheritance**).

Sự khác biệt giữa Abstract Class và Interface

FastTrack SE™

❖ Ví dụ về interface:

```php
interface a{
    public function pa();
}
interface b{
    public function pb();
}
class c{
    public function pc(){
        echo 'phuong thuc pc cua lop c';
    }
}
```

```php
class d extends c implements a,b{
    public function pa(){
        echo 'phuong thuc pa duoc khai bao interface';
    }
    public function pb(){
        echo 'phuong thuc pb duoc khai bao interface';
    }
    public function pd(){
        echo 'phuong thuc pc cua lop d';
    }
}
$s = new d();
$s->pc();
```

Sự khác biệt giữa Abstract Class và Interface

FastTrack SE™

- Ví dụ về abstracct:

```php
<?php

abstract class a{
    protected $vara = 'pa2() cua lop a';
    abstract function pa();
    protected function pa2(){
        return $this->vara;
    }
}
```

```php
class b extends a{
    public function pa(){
        echo 'pa() duoc khai bao abtract o lop a';
    }
    public function pb(){
        echo $this->pa2();
    }
}
$s = new b();
$s->pb();
```

Thế nào là một hàm static

- **Hàm static** là hàm có thể được gọi mà không cần một đối tượng của class đó. Static 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. Và cũng vì điều này mà khi sử dụng static trong chương trình thì nó sẽ chiếm nhiều tài nguyên hơn các thành phần thường.
- **Phân biệt cách dùng từ khoá static::method() với self::method()**
  - self::method() tham chiếu đến hàm method tại thời điểm nó được định nghĩa.
  - static::method() tham chiếu đến hàm method tại thời điểm nó được gọi.

❖ Ví dụ:

```php
class A
{
    public static function test()
    {
        echo 'a';
    }
    public static function run()
    {
        self::test();     // always a
        static::test(); // depend on called class
    }
}
```

```php
class B extends A {
    public static function test()
    {
        echo 'b';
    }
}

A::run(); // output 'aa'
B::run(); // output 'ab'
```

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