BÀI 7: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG - OOP

I. Tổng quan lớp đối tượng

- Đối tượng (object) là những thực thể tồn tại trong thế giới thực
- Tiếp cận hướng đối tượng là kỹ thuật cho phép biểu diễn tự nhiên các đối tượng trong thực tế với các đối tượng bên trong chương trình

II. Thành phần của lớp đối tượng: Thuộc tính – Phương thức

a. Thuộc tính:

Thuộc tính của lớp chính là thuộc tính của đối tượng mà lớp đó mô tả

```
class tenLop{
    var $tenthuoctinh;
}
```

b. Phương thức:

Phương thức của lớp chính là phương thức của đối tượng, đó là những hành động (hành vi) của đối tượng. Các phương thức nó rất giống với hàm nên nó cũng có các tính chất như hàm đó chỉ khác là phương thức nằm trong một lớp đối tượng và muốn gọi đến nó thì phải thông qua lớp đối tượng này

```
class ClassName {
    function tenPhuongThuc($bien){
        // code
    }
}
```

III. Xây dựng lớp đối tượng (demo)

IV. Sử dụng lớp đối tượng

Cú pháp: *$ten_bien = **new** ClassName();*

Để truy xuất đến các thuộc tính của đối tượng ta dùng toán tử (->) để trỏ đến.

Cú pháp: *$ten_bien->ten_phuong_thuc();*

V. Tính kế thừa trong OOP:

Cú pháp:

```
class classCon extends classCha
{

}
```

Gọi các phương thức và thuộc tính của lớp cha:

Như khái niệm tính kế thừa, lớp con kế thừa từ lớp cha nên tất cả các phương thức và thuộc tính đều **coi như** là của nó nên cách gọi cũng như cách nó gọi các phương thức thuộc tính của nó, đó là từ khóa `$this>thuoctinh`, `$this->phuong_thuc()`.

Tuy nhiên để phân biệt hàm nào của cha, hàm nào của con người ta hay gọi bằng từ khóa: `parent::thuoctinh`, `parent::phuong_thuc()`.

VI. Các mức truy cập private protected và public:

a. Private:

Đây là thành phần chỉ dành riêng cho nội bộ của lớp, nghĩa là ta không thể truy xuất tới thành phần private ở lớp con hoặc ở bên ngoài lớp.

```
class KhoaHoc
{
    private $name; //khai báo thuộc tính
    function setKhoaHoc($tenkhoahoc){
            $this->name = $tenkhoahoc;
    }
    function getKhoaHoc(){
         return $this->name;
    }
}
// Khởi tạo một lớp đối tượng khóa học
$khoahoc = new KhoaHoc();
// Gán giá trị cho thuộc tính name
$khoahoc->setKhoaHoc('PHP');
// Lấy giá trị thuộc tính tên khóa học
echo $khoahoc->getKhoaHoc();   // kết quả xuất ra PHP
```

b. Protected:

Mức truy cập **protected** chỉ cho phép truy xuất nội bộ trong lớp đó và lớp kế thừa, riêng ở bên ngoài lớp sẽ không truy xuất đc. Mức

protected thường được dùng cho những phương thức và thuộc tính có khả năng bị lớp con định nghĩa lại (overwrite).

c. Public:

Đây là mức truy cập thoáng nhất bởi vì bạn có thể truy cập tới các phương thức và thuộc tính ở bất cứ đâu, dù trong nộ bộ của lớp hay ở lớp con hay cả bên ngoài lớp đều được.

Khi khai báo thuộc tính là public ta có thể dùng từ khóa **var** để thay thế cho public

Khi khai báo với hàm là public nếu ta không đặt từ public ở đầu thì php sẽ tự hiểu hàm này là public.

VII. Hàm khởi tạo và hàm hủy trong OOP:

a. Constructor:

Hàm khởi tạo cũng là một hàm bình thường nhưng có điểm đặc biệt là nó luôn luôn được gọi tới khi ta khởi tạo một đối tượng. Hàm khởi tạo có thẻ có tham số hoặc không có tham số, có thể có giá trị trả về hoặc không.

Trong PHP có hai cách khai báo tên hàm khởi tạo. Cách thứ nhất là khai báo tên trùng với tên lớp:

```
class SinhVien
{
    function SinhVien() {
        echo 'Lớp Sinh Viên vừa được
khởi tạo';
    }
}
// khởi tạo lớp SinhVien
$sinhvien = new SinhVien();
```

Cách thứ hai là khai báo với __construct()

```
class SinhVien
   {
       function __construct() {
           echo 'Lớp Sinh Viên vừa
   được khởi tạo';
       }
   }

   // khởi tạo lớp SinhVien
   $sinhvien = new SinhVien();
```

Hàm khởi tạo trong kế thừa:

Khi lớp con kế thừa từ lớp cha thì khi ta tạo một đối tượng thuộc lớp con thì sẽ xảy ra một trong các trường hợp sau đây:

Trường hợp 1: Nếu lớp Con có hàm khởi tạo và lớp cha cũng có hàm khởi tạo

Trường hợp này hàm khởi tạo của lớp con sẽ được chạy, còn hàm khởi tạo ở lớp cha không được chạy.

```
class A {
    function __construct() {
        echo 'Lớp A được khởi tạo';
```

```
    }
}

class B extends A {
    function __construct() {
        echo 'Lớp B được khởi tạo';
    }
}
$a = new B(); // Kết quả là Lớp B được khởi tạo
```

Trường hợp 2: Nếu lớp con không có hàm khởi tạo, lớp Cha có hàm khởi tạo. Trường hợp này hàm khởi tạo ở lớp cha sẽ được chạy.

```
// Lớp A
class A {
    function __construct() {
        echo 'Lớp A được khởi tạo';
    }
}

// Lớp B
class B extends A {
```

```
}
    // Khởi Tạo Lớp B
$a = new B(); // Kết quả là Lớp A Chạy
```

Trường hợp 3: Lớp Con có hàm khởi tạo, lớp cha không có hàm khởi tạo Trường hợp này hàm khởi tạo lớp con sẽ được chạy.

```
// Lớp A
class A {

}
   // Lớp B
class B extends A
{
     function __construct() {
           echo 'Lớp B được khởi tạo';
     }
}
// Khởi Tạo Lớp B
$a = new B(); // Kết quả là Lớp B Chạy
```

Trường hợp 4: Cả 2 lớp chà và lớp con đều không có hàm khởi tạo
Trường hợp này sẽ không có hàm nào được chạy

b. Destructor:

Hàm hủy là hàm tự động gọi sau khi đối tượng bị hủy, nó thường được sử dụng để giải phóng bộ nhớ chương trình. Trong đối tượng hàm hủy có thể có hoặc không.

```
      // Lớp A
class A {
    function __construct() {
         echo 'Lớp A được khởi tạo <br/>';
    }
    function show()
    {
        echo 'Lớp A đang được sử dụng <br/>';
    }
    function __destruct() {
         echo 'Lớp A bị hủy  <br/>';
    }
}
// Chương trình
$a = new A();
   $a->show();
```

Hàm hủy trong kế thừa:

Tương tự như hàm khởi tạo trong kế thừa. Nếu lớp Con có hàm hủy thì được ưu tiên chạy, còn nếu lớp Con không có hàm hủy thì sẽ chạy ở lớp Cha, còn nếu cả 2 đều không có thì sẽ không chạy hàm nào.

VII. Thuộc tính và phương thức tĩnh trong OOP – static

Biến được khai báo **static** thì có giá trị toàn cục, không thay đổi giá trị

Khai báo:

a. Thuộc tính: [private|public|protected] static $name,
   ví dụ **public static $name**
b. Phương thức: [private|public|protected] static function functionName(){},
   ví dụ **public static function setName()**

Truy xuất (trực tiếp không cần khởi tạo object**)**

Ví dụ:

```
class Subject
{
    protected static $_name = 'unknown';

    public static function setName($name){
        Subject::$_name = $name;
    }

    public static function getName(){
```

```
        return Subject::$_name;
    }
}

Subject::setName('PHP');
echo Subject::getName(); //kq: PHP
```

Lưu ý:

- Gọi đến thuộc tính tĩnh thì thông thường ta dùng $this->name,

nhưng trong thành viên tĩnh ta dùng thêm dấu $,

ví dụ ***Subject::$name***

- Không sử dụng từ khóa ***$this*** (Vì các thuộc tính và phương thức tĩnh ở dạng toàn cục)
- Ngoài cách gọi trực tiếp tên class ta có thể dùng từ khóa **self** để thay thế, ví dụ ***self::$_name***

Gọi các hàm tĩnh bên trong class: sử dụng cú pháp hai chấm (::)

VD:

```
// Lớp động vật
class Subject
{
    protected static $_name = 'Unknown';

    public static function setName($name){
        Subject::$_name = $name;
    }
```

```
    public static function getName(){
        return Subject::$_name;
    }



    public static function all($name){
        Subject::setName($name);
        echo Subject::getName();
    }
}

Subject::all('PHP-OOP');// Kết quả: PHP-OOP
```

Kế thừa khi sử dụng static:

Không có gì khác biệt so với class thông thường, nhưng dùng hai dấu chấm (::) để truy xuất đến hàm tĩnh của lớp cha.

IX. Abstract Class và Object Interface:

1. Abstract Class (Lớp trừu tượng):

a. *Đặc điểm:*

- Các lớp được định nghĩa là abstract có thể không cần khởi tạo
- Lớp Abstract sẽ định nghĩa các phương thức mà từ đó các lớp con sẽ kế thừa nó và Overwrite lại (tính đa hình).
- Bất kỳ Class nào chứa ít nhất một phương thức trừu tượng cũng phải được trừu tượng
- Các phương thức được định nghĩa là trừu tượng chỉ khai báo tên phương thức - không thể định nghĩa chức năng.
- Tất cả các phương thức của lớp abstract đều phải được khai báo là abstract và phải ở mức protected và public, không được ở mức private.
- Lớp Abstract có thể có thuộc tính nhưng thuộc tính không được khai báo là abstract, và bạn không thể khởi tạo một biến của lớp Abstract được.
- Function thuộc lớp con có thể chứa tham số tùy chọn không được khai báo trong Function của lớp cha

b. Ví dụ:

```php
<?php
abstract class AbstractClass
{
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    public function printOut() {
        print $this->getValue() . "\n";
```

```
    }
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
```

```php
echo $class1->prefixValue('FOO_') ."\n";


$class2 = new ConcreteClass2;
$class2->getValue();
echo $class2->prefixValue('FOO_') ."\n";
?>
```

3. **Object Interface**:

a. Đặc điểm:

- **Object Interface** cho phép bạn tạo ra phương thức mà một lớp phải thực hiện, mà không cần phải định nghĩa cách xử lý những phương thức này.
- **Object Interface** được định nghĩa giống như một Class, nhưng với từ khóa **interface** thay thế từ khóa **class** và không có bất kỳ phương thức nào có nội dung được định nghĩa.
- Tất cả các phương thức khai báo trong một **Object Interface** phải là **public**; đây là bản chất của một **Interface**.
- Để implement một interface, dùng từ khóa **implement**. Tất cả các phương thức trong interface phải được thực hiện trong một lớp; nếu ko sẽ gây ra lỗi nghiêm trọng.
- Lớp implement phải sử dụng mức truy cập phương thức như được định nghĩa trong interface, nếu ko sẽ gây ra lỗi nghiêm trọng

b. Ví dụ:

1.

```
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

class Template implements iTemplate
{
    private $vars = array();
```

```
    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $
value, $template);
        }

        return $template;
    }
}
```

2.

```
<?php
interface a
{
    public function foo();
}

interface b
{
```

```
    public function bar();
}

interface c extends a, b
{
    public function baz();
}

class d implements c
{
    public function foo()
    {
    }

    public function bar()
    {
    }

    public function baz()
    {
    }
}
?>
```