PHP OOP là gì?

OOP là viết tắt của Lập trình hướng đối tượng.

Lập trình thủ tục là viết các hàm hàm thực hiện các thao tác trên dữ liệu, trong khi lập trình hướng đối tượng là về việc tạo các đối tượng chứa cả dữ liệu và phương thức .

Lập trình hướng đối tượng có một số lợi thế so với lập trình thủ tục:

- OOP nhanh hơn và dễ thực hiện hơn
- OOP cung cấp một cấu trúc rõ ràng cho các chương trình
- OOP giúp giữ mã PHP DRY "Đừng lặp lại chính mình" và làm cho mã dễ dàng hơn để duy trì, sửa đổi và gỡ lỗi
- OOP cho phép tạo các ứng dụng có thể tái sử dụng đầy đủ với ít mã hơn và thời gian phát triển ngắn hơn

Mẹo: Nguyên tắc "Đừng lặp lại chính mình" (DRY) là về việc giảm sự lặp lại của mã. Bạn nên trích xuất các mã phổ biến cho ứng dụng và đặt chúng ở một nơi duy nhất và sử dụng lại chúng thay vì lặp lại nó.

PHP - Lớp và đối tượng là gì?

Các lớp và các đối tượng là hai khía cạnh chính của lập trình hướng đối tượng.

Nhìn vào hình minh họa sau để thấy sự khác biệt giữa lớp và các đối tượng:

và

Vì vậy, một lớp là một khuôn mẫu cho các đối tượng và một đối tượng là một thể hiện của một lớp.

Khi các đối tượng riêng lẻ được tạo, chúng kế thừa tất cả các thuộc tính và hành vi từ lớp, nhưng mỗi đối tượng sẽ có các giá trị khác nhau cho các thuộc tính.

PHP OOP - Các lớp và đối tượng

Một lớp là một khuôn mẫu cho các đối tượng và một đối tượng là một thể hiện của lớp.

Giả sử chúng ta có một lớp tên là Fruit. Một trái cây có thể có các thuộc tính như tên, màu sắc, trọng lượng, v.v. Chúng ta có thể định nghĩa các biến như $ name, $ color và $ weight để giữ các giá trị của các thuộc tính này.

Khi các đối tượng riêng lẻ (táo, chuối, v.v.) được tạo, chúng kế thừa tất cả các thuộc tính và hành vi từ lớp, nhưng mỗi đối tượng sẽ có các giá trị khác nhau cho các thuộc tính.

Định nghĩa một Class

Một lớp được định nghĩa bằng cách sử dụng từ khóa `class`, theo sau là tên của lớp và một cặp dấu ngoặc nhọn ({}). Tất cả các thuộc tính và phương pháp của nó đi vào bên trong {} :

Cú pháp

```php
<?php
class Fruit {
  // code goes here...
}
?>
```

Dưới đây chúng tôi khai báo một lớp có tên Fruit bao gồm hai thuộc tính ($ name và $ color) và hai phương thức set_name () và get_name () để thiết lập và nhận thuộc tính $ name:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

tạo ra 1 đối tượng

Các lớp học không là gì nếu không có đối tượng! Chúng ta có thể tạo nhiều đối tượng từ một lớp. Mỗi đối tượng có tất cả các thuộc tính và phương thức

được định nghĩa trong lớp, nhưng chúng sẽ có các giá trị thuộc tính khác nhau.

Các đối tượng của một lớp được tạo bằng cách sử dụng từ khóa `new`

Trong ví dụ dưới đây, $ apple và $ Banana là các thể hiện của lớp Fruit:

Thí dụ

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

Trong ví dụ dưới đây, chúng tôi thêm hai phương thức nữa vào lớp Fruit, để thiết lập và nhận thuộc tính $ color:

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

PHP - The $this Keyword

Từ khóa $this đề cập đến đối tượng hiện tại và chỉ có sẵn trong các phương thức.

Nhìn vào ví dụ sau:

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

Vậy, chúng ta có thể thay đổi giá trị của thuộc tính $ name ở đâu? Có hai cách:

1. Bên trong lớp (bằng cách thêm phương thức set_name () và sử dụng $ this):

```php
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

2. Bên ngoài lớp (bằng cách thay đổi trực tiếp giá trị thuộc tính):

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
$apple->name = "Apple";
?>
```

Bạn có thể sử dụng từ khóa `instanceof` để kiểm tra xem một đối tượng có thuộc về một lớp cụ thể không:

Thí dụ

```php
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

PHP - The __construct Function

Hàm tạo cho phép bạn khởi tạo các thuộc tính của đối tượng khi tạo đối tượng.

Nếu bạn tạo một hàm `__construct()`, PHP sẽ tự động gọi hàm này khi bạn tạo một đối tượng từ một lớp.

Lưu ý rằng hàm xây dựng bắt đầu bằng hai dấu gạch dưới (__)!

Chúng ta thấy trong ví dụ dưới đây, việc sử dụng hàm tạo sẽ giúp chúng ta không gọi phương thức set_name () làm giảm số lượng mã:

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Một vi dụ khac:

Thí dụ

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

PHP - The __destruct Function

Một hàm hủy được gọi khi đối tượng bị hủy hoặc tập lệnh bị dừng hoặc thoát.

Nếu bạn tạo một `__destruct()`hàm, PHP sẽ tự động gọi hàm này ở cuối đoạn script.

Lưu ý rằng hàm hủy bắt đầu bằng hai dấu gạch dưới (__)!

Ví dụ dưới đây có hàm __construct () được gọi tự động khi bạn tạo một đối tượng từ một lớp và hàm `__destruct`() được gọi tự động ở cuối tập lệnh:

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

Một vi dụ khac:

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function __destruct() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

$apple = new Fruit("Apple", "red");
?>
```

PHP OOP - Access Modifiers

Các thuộc tính và phương thức có thể có các sửa đổi truy cập kiểm soát nơi chúng có thể được truy cập.

Có ba sửa đổi truy cập:

- `public`- thuộc tính hoặc phương pháp có thể được truy cập từ khắp mọi nơi. Đây là mặc định
- `protected` - thuộc tính hoặc phương thức có thể được truy cập trong lớp và bởi các lớp xuất phát từ lớp đó
- `private` - thuộc tính hoặc phương thức CHỈ có thể được truy cập trong lớp

Trong ví dụ sau, chúng tôi đã thêm ba công cụ sửa đổi truy cập khác nhau vào ba thuộc tính. Ở đây, nếu bạn cố gắng đặt thuộc tính tên thì nó sẽ hoạt động tốt (vì thuộc tính tên là công khai). Tuy nhiên, nếu bạn cố gắng đặt thuộc tính màu hoặc trọng lượng, nó sẽ dẫn đến lỗi nghiêm trọng (vì thuộc tính màu sắc và trọng lượng được bảo vệ và riêng tư):

Thí dụ

```php
<?php
class Fruit {
  public $name;
  protected $color;
  private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

Trong ví dụ tiếp theo, chúng tôi đã thêm các sửa đổi truy cập vào hai phương thức. Ở đây, nếu bạn cố gắng gọi hàm set_color () hoặc hàm set_ weight (), nó sẽ dẫn đến một lỗi nghiêm trọng (vì hai hàm được coi là bảo vệ và riêng tư), ngay cả khi tất cả các thuộc tính đều công khai:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public $weight;

  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

PHP OOP - Inheritance

Lớp con sẽ kế thừa tất cả các thuộc tính và phương thức công khai và được bảo vệ từ lớp cha. Ngoài ra, nó có thể có các thuộc tính và phương thức riêng.

Một lớp kế thừa được xác định bằng cách sử dụng từ khóa.

`extends`

Hãy xem xét một ví dụ:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}."
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

Giải thích ví dụ

Lớp Dâu được kế thừa từ lớp Trái cây.

Điều này có nghĩa là lớp Dâu tây có thể sử dụng các thuộc tính $ name và $ color công khai cũng như các phương thức công khai __construct () và intro () từ lớp Fruit vì tính kế thừa.

Lớp Dâu cũng có phương thức riêng: message ().

PHP - Kế thừa và Công cụ sửa đổi truy cập được bảo vệ

chúng ta đã học được rằng `protected`các thuộc tính hoặc phương thức có thể được truy cập trong lớp và bởi các lớp có nguồn gốc từ lớp đó. Điều đó nghĩa là gì?

Hãy xem xét một ví dụ:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}

// Try to call all three methods from outside class
$strawberry = new Strawberry("Strawberry", "red");  // OK. __construct() is public
$strawberry->message(); // OK. message() is public
$strawberry->intro(); // ERROR. intro() is protected
?>
```

Trong ví dụ trên, chúng ta thấy rằng nếu chúng ta cố gắng gọi một `protected` phương thức (intro ()) từ bên ngoài lớp, chúng ta sẽ nhận được một lỗi. `public` phương pháp sẽ hoạt động tốt!

Hãy xem xét một ví dụ khác:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
    // Call protected method from within derived class - OK
    $this -> intro();
  }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() is public
$strawberry->message(); // OK. message() is public and it calls intro() (which is protected) from within the derived class
?>
```

Trong ví dụ trên chúng ta thấy rằng tất cả đều hoạt động tốt! Đó là bởi vì chúng ta gọi `protected` phương thức (intro ()) từ bên trong lớp dẫn xuất.

PHP - Các phương thức được ghi đè

Các phương thức được kế thừa có thể được ghi đè bằng cách xác định lại các phương thức (sử dụng cùng tên) trong lớp con.

Nhìn vào ví dụ dưới đây. Các phương thức __construct () và intro () trong lớp con (Strawberry) sẽ ghi đè các phương thức __construct () và intro () trong lớp cha (Fruit):

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public $weight;
  public function __construct($name, $color, $weight) {
    $this->name = $name;
    $this->color = $color;
    $this->weight = $weight;
  }
  public function intro() {
    echo "The fruit is {$this->name}, the color is {$this->color}, and the weight is {$this->weight} gram.";
  }
}

$strawberry = new Strawberry("Strawberry", "red", 50);
$strawberry->intro();
?>
```

PHP - The final Keyword

Các `final` từ khóa có thể được sử dụng để ngăn chặn lớp thừa kế hoặc để ngăn chặn phương pháp trọng.

Ví dụ sau đây cho thấy cách ngăn chặn kế thừa lớp:

```php
<?php
final class Fruit {
  // some code
}

// will result in error
class Strawberry extends Fruit {
  // some code
}
?>
```

Ví dụ sau đây cho thấy cách ngăn chặn phương thức ghi đè:

```php
<?php
class Fruit {
  final public function intro() {
    // some code
  }
}

class Strawberry extends Fruit {
  // will result in error
  public function intro() {
    // some code
  }
}
?>
```

PHP - Hằng số lớp

Hằng số không thể thay đổi một khi nó được khai báo.

Các hằng số lớp có thể hữu ích nếu bạn cần xác định một số dữ liệu không đổi trong một lớp.

Một hằng số lớp được khai báo bên trong một lớp với `const` từ khóa.

Các hằng số lớp là trường hợp nhạy cảm. Tuy nhiên, nên đặt tên các hằng số trong tất cả các chữ cái viết hoa.

Chúng ta có thể truy cập một hằng số từ bên ngoài lớp bằng cách sử dụng tên lớp theo sau là toán tử phân giải phạm vi (`::`) theo sau là tên hằng, như ở đây:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Hoặc, chúng ta có thể truy cập một hằng số từ bên trong lớp bằng cách sử dụng `self`từ khóa theo sau là toán tử phân giải phạm vi (`::`) theo sau là tên hằng, như ở đây:

Thí dụ

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```