



# PHP Hướng Đối Tượng

- Các khái niệm hướng đối tượng
- Tạo các lớp (class), thuộc tính (attribute), phương thức
- Sử dụng các thuộc tính
- Gọi các phương thức
- Thừa kế
- Thiết kế lớp
- Viết mã cho các class của bạn

- Là phương pháp thiết kế và viết mã lệnh chương trình hướng thủ tục. Một chương trình lớn được phân rã thành các thủ tục, module nhỏ hơn.
- Lấy chương trình con làm nền tảng

**Chương trình = Cấu trúc dữ liệu + Thuật giải**

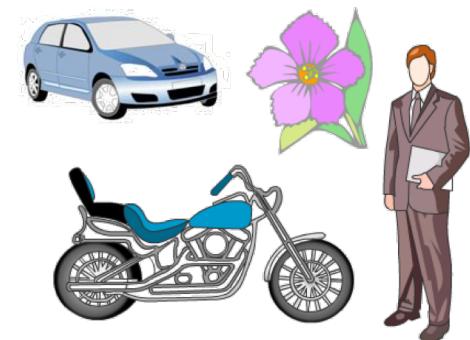
- Ứng với mỗi cấu trúc dữ liệu, thì có một thuật giải tương ứng. Nếu cấu trúc dữ liệu thay đổi thì thuật giải được viết trong các hàm xử lý dữ liệu đó cũng bị thay đổi theo => Vậy tốn nhiều thời gian, kém hiệu quả đối với các chương trình lớn

# Lập trình hướng đối tượng là gì ?

- Là phương pháp thiết kế và phát triển ứng dụng dựa trên kiến trúc lớp (class) và đối tượng (object)
- Lấy đối tượng để xây dựng thuật giải, xây dựng chương trình  
**Đối tượng = Phương thức + Dữ liệu**
- Cho phép liên kết cấu trúc dữ liệu với các thao tác.
- Cho phép xây dựng thêm các cấu trúc dữ liệu mới, thao tác mới dựa trên những cái đã có

## • **Đối tượng trong thế giới thực**

- Là những đối tượng tồn tại khách quan trong thực tế
- Ví dụ trong hệ thống đặt vé xe lửa có những thực thể như: một con tàu, một nơi đến cho chuyến đi, một đại lý bán vé,... Ngoài ra xung quanh chúng ta còn có nhiều các thực thể khác tồn tại như một chiếc ôtô, một chiếc xe máy, một bông hoa hay một con người,...
- Mỗi thực thể đều có những đặc điểm, và các khả năng thực hiện một hành vi nào đó.
  - Đặc điểm là thuộc tính hay các điểm đặc biệt mô tả thực thể.
  - Hành vi là các hoạt động hay các thao tác có liên quan đến đối tượng

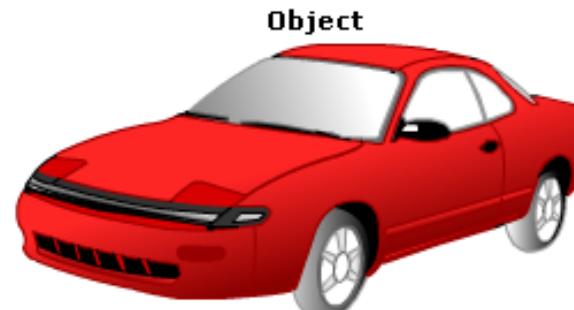
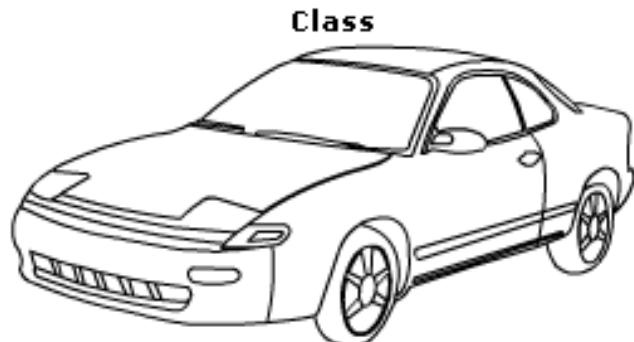


- **Đối tượng trong phần mềm ứng dụng**

- Là sự ánh xạ từ đối tượng có trong thế giới thực
- Đối tượng trong phần mềm ứng dụng có trạng thái(state) và hành động (behavior). Trạng thái của đối tượng phần mềm giống như các thuộc tính, đặc điểm còn ứng xử giống như các hành vi, thao tác của đối tượng trong thế giới thực.
- Đối tượng phần mềm lưu trữ trạng thái các trong các trường (fields: các biến trong ngôn ngữ lập trình), các hành động được thể hiện qua các phương thức
  - => Trong lập trình hướng đối tượng thì đối tượng được hiểu theo nghĩa là một thực thể. Các dữ liệu và các lệnh tác động lên dữ liệu được đóng gói lại với nhau trong một đơn vị đầy đủ tạo nên đối tượng. Một đối tượng gồm có các *phương thức (method)* và các *thuộc tính(property)*

- Là một sự trừu tượng hóa của các đối tượng có cấu trúc dữ liệu và các phương thức giống nhau (hay nói cách khác lớp là một tập các đối tượng cùng loại). Mỗi đối tượng sẽ là một thể hiện cụ thể (instance) của lớp đó. Trong lập trình, các đối tượng sẽ là các biến có kiểu lớp...
- Lớp được xem như là một kiểu dữ liệu có cấu trúc được định nghĩa trong chương trình mà các thành phần bên trong nó là các phương thức, các thuộc tính (hay các thành phần dữ liệu)

- Một lớp mô tả một tập thực thể, trong khi một đối tượng là một thực thể thật sự trong tập thực thể đó.
- Đối tượng là vật thật, trong khi lớp là một mô hình khái niệm định nghĩa tất cả các trạng thái và hành vi cần thiết của một đối tượng.
- Lớp không thay đổi, trong khi dữ liệu chứa trong một đối tượng có thể thay đổi. (Với mỗi đối tượng dữ liệu ở cùng thuộc tính có thể là khác nhau)
- Mỗi đối tượng được gọi là một thể hiện của một lớp.



- **Tính đóng gói (encapsulation):**
  - Tất cả các dữ liệu dùng để mô tả đối tượng cùng với các hàm thao tác trên dữ liệu đó được đặt trong một khối (còn gọi là lớp).
  - Với mỗi đối tượng, không cho phép truy xuất trực tiếp vào các thành phần dữ liệu của nó mà phải thông qua các thành phần chức năng (còn gọi là các phương thức) của chính đối tượng để làm việc đó.
  - Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.
- **Tính kế thừa (inheritance):** Đặc tính này cho phép xây dựng một lớp mới có thể có sẵn các đặc tính mà các lớp khác đã có thông qua kế thừa. Điều này cho phép các lớp chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại.

- **Tính đa hình (polymorphism):**

- Là định nghĩa một hàm có thể có các hành động khác nhau tùy thuộc vào ngữ cảnh. Ví dụ hàm verifyPassword() khi không có tham số sẽ thực hiện khác với khi được truyền vào hai tham số.

# Định nghĩa một class

- Cú pháp định nghĩa một lớp

```
class class_name
{
    //biến thể hiện
    var $tên_bien;

    //phương thức thể hiện
    function tên_hàm1() {

    }
    //phương thức thể hiện
    function tên_hàm2($tham_số) {

    }
    ...
}
```

- ❖ Thân lớp gồm có các khai báo thuộc tính và phương thức.
- ❖ Thân lớp phải được đặt trong 1 khối lệnh PHP duy nhất.
  - Các dữ liệu (biến) được khai báo bằng **var**
  - Các phương thức (hàm) khai báo như thông thường.

# Định nghĩa một class

- Ví dụ khai báo một class Point mô tả các đối tượng điểm trong 2D

```
class Point{  
    var $x;  
    var $y;  
    function getX() {  
        return $this->x;  
    }  
    function getY() {  
        return $this->y;  
    }  
    function setX($px) {  
        $this->x = $px;  
    }  
    function setY($py) {  
        $this->y = $py;  
    }  
}
```

# Sử dụng \$this

- \$this nghĩa là tham chiếu đến thể hiện của đối tượng hiện tại.
- Được dùng để truy xuất các thuộc tính và phương thức từ bên trong lớp (class)

```
class Point{  
    var $x;  
    var $y;  
    function setX($x) {  
        $this->x = $x; → $this->x = $x;  
    }  
    function setY($y) {  
        $this->y = $y; → $this->y = $y;  
    }  
}
```

- Sử dụng từ khóa new để tạo đối tượng
- Cú pháp:

```
$tên_đối_tượng = new tên_hàm_khởi_tạo();
```

- Ví dụ:

```
$p = new Point();
```

- Sử dụng đối tượng

- Sau đối tượng được tạo, sử dụng -> để truy xuất các thuộc tính và phương thức public.

```
$p->setX(250);  
$p->setY(200);
```

- Các từ khóa điều khiển truy xuất hay con gọi là các bổ từ(modifier) được sử dụng điều khiển việc truy xuất đến các thành viên của lớp.
- Điều khiển truy xuất giúp ngăn cản việc sử dụng sai mục đích các chi tiết của lớp, và ẩn đi các chi tiết thực thi bên trong lớp.
- Có các bổ từ truy xuất:
  - public
  - protected
  - private

- Public
  - Khi khai báo thuộc tính hay phương thức mà không chỉ rõ là private hay protected, sẽ được định là public
  - Có thể truy xuất các thành viên public từ bên trong hoặc bên ngoài lớp
- Protected
  - Nếu một thuộc tính hay phương thức được khai báo là protected, bạn chỉ có thể truy xuất từ lớp con của nó.
- Private
  - Nếu các thuộc tính hay phương thức được khai báo là private, bạn không thể truy xuất chúng từ bên ngoài của lớp. Tuy nhiên, các phương thức trong cùng lớp vẫn có thể truy xuất được.

- Là phương thức đặc biệt của lớp được dùng để khởi tạo giá trị cho các biến thành viên của lớp.
- Được gọi tự động khi tạo các thể hiện(instances) của lớp.
- Có thể có hoặc không có các tham số
- Một lớp có thể không có phương thức khởi tạo
- Trong PHP 5 có hai cách để viết một phương thức khởi tạo bên trong một lớp.
  - Tạo một phương thức khởi tạo với tên `__constructor()`
  - Tạo một phương thức khởi tạo với tên trùng với tên của lớp

- Phương thức khởi tạo `__construct()`

```
class Point{  
    var $x;  
    var $y;  
    function __construct() {  
        $this->x = 100;  
        $this->y = 100;  
    }  
    ...  
}
```

- Phương thức khởi tạo `__construct()` với tham số

```
class Point{  
    var $x;  
    var $y;  
    function __construct($px, $py) {  
        $this->x = $px;  
        $this->y = $py;  
    }  
    ...  
}
```

- Phương thức khởi tạo `__construct()` với tham số mặc định

```
class Point{  
    var $x;  
    var $y;  
    function __construct($px = 100, $py = 100) {  
        $this->x = $px;  
        $this->y = $py;  
    }  
    ...  
}
```

- Phương thức khởi tạo với tên trùng với tên của Class

```
class Point {  
    var $x;  
    var $y;  
    function Point($pX = 100, $pY = 100) {  
        $this->x = $pX;  
        $this->y = $pY;  
    }  
    ...  
}
```

- Phương thức hủy được gọi tự động bởi PHP khi kết thúc việc thực thi kịch bản bạn.
- Phương thức hủy được tạo với tên `__destruct()`.

- Cú pháp khai báo thuộc tính

```
<chỉ định truy xuất | var> $tên_thuộc_tính;
```

- Truy xuất thuộc tính

```
$tên_đối_tượng->tên_thuộc_tính;
```

- Ví dụ

```
//tạo đối tượng point
$p = new Point();
$p->x = 150;           //truy xuất thuộc tính x
$p->y = 250;           //truy xuất thuộc tính y
```

- Cú pháp khai báo phương thức

```
[chỉ định truy xuất] function tên_hàm([ds_tham_số])
{
    //thân hàm
}
```

- Ví dụ:

```
class Point{
    var $x;
    var $y;
    public function display() {
        echo "($this->x, $this->y)";
    }
}
```

- Phương thức có tham số mặc định

```
class Point{  
    var $x;  
    var $y;  
    function setX($px = 100) {  
        $this->x = $px;  
    }  
    function setY($py = 100) {  
        $this->y = $py;  
    }  
}
```

- Sử dụng phương thức có tham số mặc định

```
$p = new Point();  
$p->setX(150);           //Gọi và truyền giá trị  
$p->setY();              //Gọi nhưng không truyền giá trị
```

# Định nghĩa phương thức(method)

- Sử dụng phương thức với tham số là mảng

```
class Pheptoan{  
    function tinh tong ($arrayPara) {  
        if (is_array($arrayPara))  
        { $tong=0;  
            foreach ($arrayPara as $x)  
                $tong+=$x;  
            echo $tong;  
        }  
        else  
            echo $arrayPara;  
    }  
}
```

```
$pt = new Pheptoan();  
$mang = array(10,20,30);  
$pt->tinh tong ($mang);
```

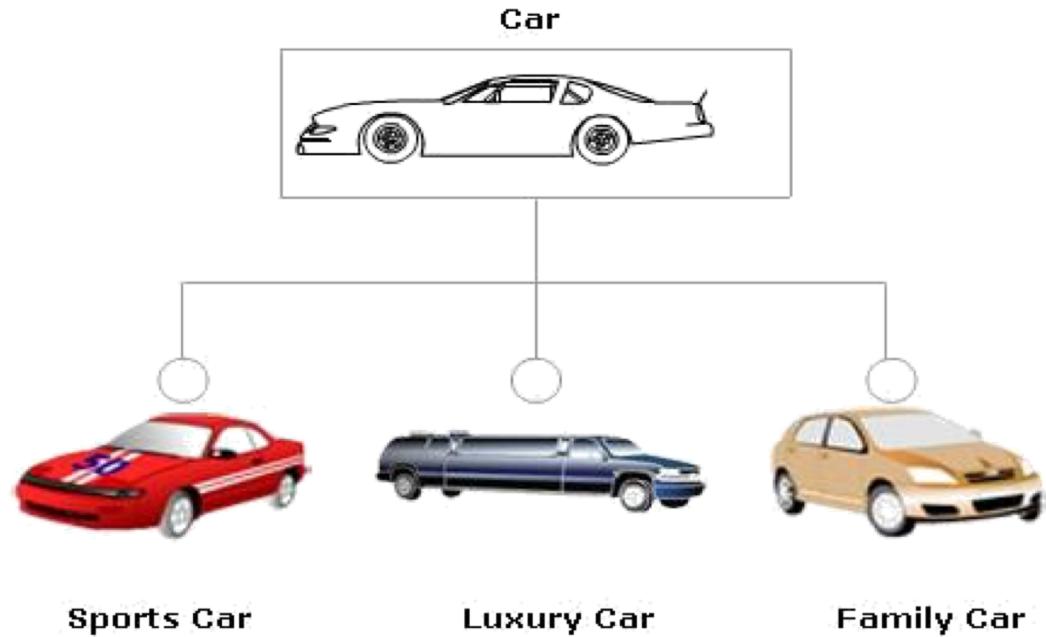
- Sử dụng từ khóa const để định nghĩa một hằng của lớp

```
class WordCounter{
    const ASC  = 1; //tên hằng không bắt đầu bằng $
    const DESC = 2;
    private $words;
    function __construct($filename) {
        $file_content = file_get_contents($filename);
        $this->words =(array_count_values(str_word_count
                                         (strtolower($file_content),1)));
    }
    public function count($order) {
        if ($order==self::ASC)
            asort($this->words);
        else if ($order==self::DESC)
            arsort($this->words);
        foreach($this->word as $key=>$val)
            echo $key."=". $val."<br/>";
    }
}
```

```
$wc = new WordCounter("word.txt");
$wc->count(WordCounter::DESC);
```

- Thùa kế trong thế giới thực

- Quá trình mà nhờ đó các đặc điểm và hành vi được truyền từ thực thể cha sang thực thể con, được gọi là thừa kế.
- Thừa kế nhằm mục đích để tránh tạo lại các đặc tính và hành vi đã có rồi, thay vào đó là sử dụng lại đã có và xây dựng thêm các đặc tính và hành vi cho các thực thể mới



- Thùa kế trong lập trình

- Trước tiên là định nghĩa một lớp tổng quát trước, sau đó định nghĩa các lớp cụ thể và thêm vào các chi tiết mới so với lớp tổng quát.
- Tiết kiệm công việc và khuyến khích tái sử dụng lại mã và có thể tùy biến
- Lớp mà thừa kế các thuộc tính và phương thức được gọi là lớp con (hay lớp dẫn xuất)
- Lớp mà được lớp con thừa kế các thuộc tính và phương thức được gọi là lớp cha

- Từ khóa extends được sử dụng để tạo lớp con
- Một số nguyên tắc khi tạo lớp con:
  - Một lớp chỉ có thể được thừa kế từ một lớp.
  - Một lớp con có thể thừa kế tất cả các thành viên protected của lớp cha.
  - Không giống như các thành viên khác, constructor không được thừa kế
- Cú pháp

```
class <tên_lớp_con> extends <tên_lớp_cha>
{
    . . . . .
}
```

- Ví dụ

Ví dụ:

```
class Car {  
    public $mileage;  
    public $color;  
    protected $make;  
    public function accelerate(){  
        echo "Car is Accelerating !";  
    }  
}  
class LuxuryCar extends Car {  
    //Luxury defines an additional feature named perks  
    public $perks;  
}
```

- Trong lớp con định nghĩa một phương thức mới có tên trùng với tên một phương thức đã có trong lớp cha thì được gọi là ghi ch同胞 (overriding).
- Mục đích của ghi ch同胞 là để cài đặt mới lại hay khác đi các hành vi của lớp cha cho phù hợp với các đối tượng của lớp con.
- Một số nguyên tắc khi tạo phương thức ghi ch同胞
  - Phương thức ghi ch同胞 trong lớp con phải có tên và số lượng tham số giống như phương thức được ghi ch同胞 ở lớp cha.
  - Lớp con có thể ghi ch同胞 các phương thức **public** hoặc **protected** ở lớp cha.
  - Phương thức được ghi ch同胞 ở lớp cha phải có **bổ tú truy xuất** mạnh hơn **bổ tú truy xuất** của phương thức ghi ch同胞.

- Sử dụng từ khóa `final` trong phần khai báo phương thức để ngăn không cho lớp con ghi đè phương thức đó.
- Phương thức nên được khai báo `final` nếu sự thay đổi thực thi là ảnh hưởng đến trạng thái nhất quán của đối tượng
- Cú pháp

```
[chỉ định truy xuất] final function tên_hàm([ds_tham_số])
{
    //thân hàm
}
```

- Ví dụ:

```
class Proposal{
    public final finalMemo() {
        echo "This is final memo";
    }
}
```

- Có thể khai báo một lớp với từ khóa final để giới hạn khả năng mở rộng (extensibility) và ngăn không cho bất kỳ một lớp nào khác có thể hiệu chỉnh(modification) định nghĩa của lớp đó.
- Cú pháp:

```
[chỉ định truy xuất] final class_name {  
    . . . . .  
}
```

- Là phương thức chỉ có phần khai báo chứ không có phần cài đặt, nghĩa là một phương thức không có câu lệnh nào trong phần thân.
- Trong PHP, phương thức trừu tượng được bắt đầu với từ khóa abstract.
- Phương thức trừu tượng chỉ là một hợp đồng (contract), các lớp con sẽ cung thực hiện cài đặt nó.
- Cú pháp:

```
abstract function  
    tên_phương_thức( [danh_sách_tham_số] );
```

- Ví dụ:

```
public abstract function drafMemo();
```

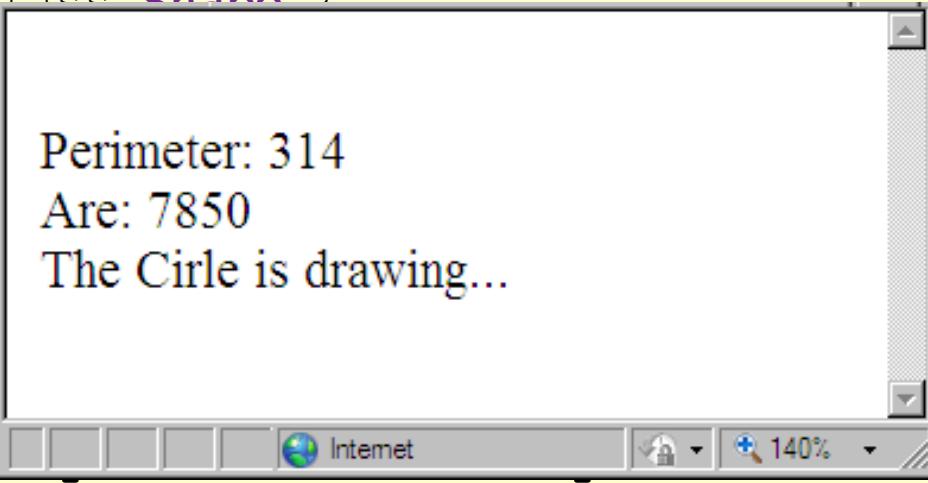
- Các lớp trừu tượng được dùng để khai báo các đặc điểm chung của các lớp con.
- Không thể tạo các đối tượng từ các lớp trừu tượng nhưng có thể tạo lớp con chúng.
- Các lớp con phải cài đặt (implement) các phương thức trừu tượng được khai báo trong lớp cha. Ngược lại, phải khai báo là phương thức trừu tượng nếu lớp con không muốn thực thi chúng.
- Cú pháp

- ✓ Framework cho các lớp khác
- ✓ Có thể không có hoặc có nhiều phương thức trừu tượng
- ✓ Không thể tạo thực thể được
- ✓ Có thể thừa kế

```
abstract class <tên_lớp>
{
    [ abstract function tên_phương_thức() ; ]
}
```

- Ví dụ

```
abstract class Shape {
    public $color;
    public function __construct($color) {
        $this->color = $color;
    }
    public function draw() {
        echo "The Circle is drawing...  
";
    }
    public function area() {
        return pi() * $this->r * $this->r;
    }
    public function perimeter() {
        return 2 * pi() * $this->r;
    }
}
```



```
Circle $c = new Circle();
$c->setR(50);
echo "<br/>Perimeter: ".$c->area();
echo "<br/>Are: ".$c->area();
$c->draw();
```

- Interface định nghĩa một tập các phương thức trừu tượng chuẩn mà lớp sẽ tuân theo.
- Một lớp cài đặt(implement) interface, thì lớp đó kế thừa các phương thức trừu tượng của interface.
- Interface không phải là một lớp, cách viết một interface cũng tương tự như viết một lớp, nhưng chúng có hai khác biệt sau:
  - Một lớp mô tả các thuộc tính và các hành vi của một đối tượng.
  - Một interface chứa các hành vi mà một lớp phải cài đặt(implements).
- Tất cả các phương thức của interface cần phải được định nghĩa bên trong lớp cài đặt(implement) interface, nếu lớp đó không là lớp trừu tượng.

- Cú pháp khai báo một interface:

```
interface Tên_Interface {  
    [public] function tên_phương_thức([tham_số]);  
    . . .  
}
```

- Ví dụ

```
interface Animal {  
    public function eat();  
    public function travel();  
}
```

- Cú pháp cài đặt interface

```
class Person implements Printable {
    private String name = new String("Bill");
    private int age = 22;
    public void printAll() {
        System.out.println("Name is " + name + ", age is " + age);
    }
}
class Stock implements Printable {
    private String tickerSymbol = new String("XYZ");
    private int shares = 100;
    private int currentPrice = 4000; // in pennies
    public void printAll() {
        System.out.println(tickerSymbol + " " + shares +
                           " shares at " + currentPrice);
        System.out.println("Value: " + currentPrice * shares);
    }
    public void sell() {
        System.out.println(tickerSymbol + " sold");
    }
}
```

- **Mở rộng(Extending) Interface:**

```
//Filename: Sports.php
interface Sports{
    public function setHomeTeam($name);
    public function setVisitingTeam($name);
}
```

```
//Filename: Football.php
interface Football extends Sports{
    public function homeTeamScored($points);
    public function visitingTeamScored($points);
    public function 
```

```
//Filename: Hockey.php
interface Hockey extends Sports {
    public function homeGoalscored();
    public function visitingGoalscored();
    public function endOfPeriod($period);
    public function overtimePeriod($ot);
}
```

- PHP cung cấp một số phương thức magic (magic method) để thực hiện một số thủ thuật trong lập trình hướng đối tượng.
- Mỗi phương thức magic đều bắt đầu bằng \_\_ (hai kí tự gạch dưới), và được PHP gọi tự động khi có một số điều kiện nào đó xảy ra như cố gắng truy cập một thuộc tính không tồn tại (hoặc không được phép truy cập).
- Chúng có thể được sử dụng để tạo thuộc tính, phương thức ảo. Điều này được gọi là overloading trong PHP
- Một số phương thức magic trong PHP

- |                  |              |                 |
|------------------|--------------|-----------------|
| ❖ __construct()  | ❖ __set()    | ❖ __toString()  |
| ❖ __destruct()   | ❖ __isset()  | ❖ __invoke()    |
| ❖ __call()       | ❖ __unset()  | ❖ __set_state() |
| ❖ __callStatic() | ❖ __sleep()  | ❖ __clone()     |
| ❖ __get()        | ❖ __wakeup() |                 |

- Được gọi khi code cố gắng truy xuất để lấy giá trị từ một thuộc tính không được định nghĩa hoặc không được truy xuất.
- Nhận một tham số - là tên của thuộc tính và trả về một giá trị được xem như là giá trị của thuộc tính đó.
- Có thể sử dụng vào việc mở rộng kiểm soát truy xuất bằng cách tạo ra các thuộc tính "read-only".

# Phương thức magic \_\_get()

- Ví dụ

```
class Customer {  
    private $name;  
    private $data = array();  
    public function __get($name) {  
        // kiểm tra xem mảng $data có tồn tại key trong $name  
        if(array_key_exists($name, $this->data)) {  
            // trả về giá trị lấy trong mảng $data  
            return $this->data[$name];  
        }  
        return null;//return "không có thuộc tính này";  
    }  
}  
$c = new Customer();  
echo $c->email; //đọc giá trị từ một thuộc tính chưa được định nghĩa
```

- Được gọi khi gán giá trị cho một thuộc tính chưa được định nghĩa hoặc không được truy xuất.
- Nhận vào hai tham số - một là tên của thuộc tính và một là giá trị gán cho thuộc tính đó.
- Có thể sử dụng kiểm tra dữ liệu khi lưu trữ hoặc tạo ra các thuộc tính "read-only".

# Phương thức magic \_\_set()

- Ví dụ dùng phương thức \_\_set() phát sinh một ngoại lệ khi gán giá trị cho thuộc tính không tồn tại

```
class Customer {  
    private $name;  
    public function __set($dt, $vl) {  
        throw new Exception('Không thể gán các giá trị  
        cho các biến chưa được định  
        nghĩa (undefined variables)', 1);  
    }  
}  
  
$c = new Customer();  
$c->email = 'email@domain.com'; //lệnh này sẽ làm phát sinh  
một ngoại lệ
```

# Phương thức \_\_call()

- Phương thức magic \_\_call() sẽ được gọi, khi đối tượng của lớp (có cài đặt \_\_call()) gọi một phương thức không hề tồn tại

```
class Caller {  
    public function __call($funcname, $args) {  
        print "Undefined method $funcname called with vars:\n";  
        print_r($args);  
    }  
}  
  
$foo = new Caller();  
$a = $foo->test(1, 2, 3);
```

# Kiểm tra sự tiến bộ



Đại học Quốc gia Hà Nội  
VIỆN CÔNG NGHỆ THÔNG TIN

# Bài tập



Đại học Quốc gia Hà Nội  
VIỆN CÔNG NGHỆ THÔNG TIN