



# POO - PHP - Partie 1 : Manipulation d'objets

▼ Cours	PHP
▼ Type	Guide
Supports	<a href="https://openclassrooms.com/fr/courses/1665806-programmez-en-orienté-objet-en-php">https://openclassrooms.com/fr/courses/1665806-programmez-en-orienté-objet-en-php</a>
▼ Difficulté	Facile

La programmation orientée objet se définit par sa capacité à construire des modèles que l'on appelle des classes. Ainsi, chaque objet instancié de cette classe peut hériter des propriétés de sa classe mère et se différencier avec de nouvelles variables ou méthodes.

## Créer une classe

```
<?php  
class NomDeLaClasse {  
    // Propriétés, méthodes de la classe  
}
```

Le nommage de la classe se fera toujours en PascalCase.  
On instancie un objet de cette classe avec le mot clé `new`.

## Instancier un objet en PHP

```
<?php  
class NomDeLaClasse {
```

```
}
```

```
$objet = new NomDeLaClasse;
```

Une instance est un objet créé à partir d'une classe. Chaque instance récupère les propriétés et les méthodes définies dans la classe Mère.

**i** Créer une classe sert à englober un ensemble de paramètres au même endroit pour y faire référence plus tard dans le code et éviter les répétitions. On peut aussi s'en servir pour construire des modèles qui serviront à créer d'autres classes.

## Raccourci “`→`” pour un objet

Vous pouvez accéder à une variable ou une propriété d'un objet avec la fonction raccourcie “`→`”.

```
<?php
```

```
$date = new DateTime;
```

```
echo $date->format('d/m/Y');
```

Ici, on accède à la méthode `format()` de l'objet `$date`, instancié depuis la classe `DateTime` (classe par défaut de PHP).

Grâce à ce raccourci, on peut effectuer du chaînage : C'est à dire, appeler plusieurs méthodes à la suite :

```
<?php
```

```
$formatDate = $date->modify('+1 day')->format('d/m/Y');
```

**!** Ici on manipule directement un objet, ce qui veut dire qu'on ne renvoie AUCUNE copie de cette objet, mais bien l'objet lui même modifié. Pour renvoyer une copie, il faudrait d'abord copier l'objet dans une nouvelle variable avec la méthode “`clone`”.

Certaines classes possèdent des instances que l'on peut appeler pour lancer une copie par défaut. Ainsi `DateTime` possède la sous-classe `DateTimeImmutable`, qui permet de cloner la classe `DateTime` par défaut.

## Attribution de `stdClass` par défaut

PHP possède la capacité d'assigner dynamiquement une classe à un objet qui n'a pas de nom spécifié. Ainsi, en récupérant un objet au format .JSON par exemple, PHP l'interprétera automatiquement dans une classe par défaut nommée `stdClass`.

**!** Ce n'est pas forcément une bonne pratique.

## Manipuler une variable dans une fonction

```
<?php
```

```
function calcul(&$valeur) {
```

```
    $valeur .= 2;
```

```
}
```

```
$valeur = 1;
calcul($valeur);
// $valeur vaudra donc 3
```

Ici on utilise le symbole `&` pour affecter une nouvelle valeur à notre variable `$valeur`. Sans ce symbole, la manipulation dans la fonction resterait dans la fonction. (Avec le `&`, le résultat est égal à 3, sans le `&`, il serait égal à 1).

Lors de l'utilisation d'un objet, il n'y a pas besoin d'écrire ce `&`. PHP comprends directement que l'on souhaite modifier l'objet en question et le modifiera.

## Ajouter des propriétés, des méthodes et les manipuler

### I - Propriétés.

```
<?php
class Voiture {
    public string $marque = "peugeot";
}

$renault = new Voiture;
$renault->marque = "renault";
```

Ici, on a créer une classe Voiture en lui attribuant une `$marque` qui par défaut sera peugeot. Le nouvel objet `$renault` quand à lui utilise la classe Voiture mais modifie la valeur de `$marque` par renault.

Le mot clé `public` permet de définir une accessibilité à la propriété `$marque`. En dehors de la classe, cette propriété est utilisable.

**i** On peut forcer le PHP à respecter le typage des propriétés d'une classe (ici String pour la marque) avec la fonction `declare` au début du script.

```
<?php
declare( strict_types = 1);
```

### II - Méthodes.

```
<?php
declare(strict_types = 1);

class Voiture {
    public string $marque;
    public float $numeroDeSerie;

    public function getMarque(): string {
        return $this->marque;
    }
}
```

Ici on utilise une méthode `getMarque()` pour obtenir la marque d'un véhicule. Quand on instancie un nouvel objet, on pourra retourner sa marque actuelle à tout moment avec cette méthode.

Le mot clé `$this` fait référence ici à l'objet instancié lui-même.

### III - Manipulation avancées.

```
<?php
declare(strict_types = 1);

class Peugeot {
    public float $hauteur;
    public float $longueur;
    private string $marque = "Peugeot";

    public function getSize(): String{
        return ($this->hauteur * $this->largeur) . "est la taille de la voiture" . $this->marque;
    }
}
```

On utilise une `private string` car on souhaite que `$marque` ne soit pas modifiée en dehors de sa classe. Elle peut être retournée telle quelle, mais elle ne changera jamais.

On peut aussi utiliser un setter pour définir une valeur d'objet et la limiter. On parle d'encapsulation lorsque l'accès est verrouillé au sein d'une classe.

```
<?php
declare(strict_types = 1);

class Peugeot {
    public float $hauteur;
    public float $longueur;
    private string $marque = "Peugeot";

    public function setHauteur(float $hauteur): void{
        if ($hauteur < 0){
            trigger_error(
                "La hauteur ne peut être négative",
                E_USER_ERROR
            );
        }
        $this->hauteur = $hauteur;
    }
}
```

`setHauteur()` attend un argument `$hauteur` de type float.

On appelle ça un mutateur en français (setter en anglais).

On l'accompagne souvent d'un getter (accesseur en français).

```
public function getHauteur(): float {
    return $this->hauteur;
}
```

## Méthodes statiques

Il est possible d'appeler des propriétés ou des méthodes sans instancier d'objet à une classe. Ainsi, on peut regrouper un ensemble de fonction dans une classe et leur attribuer le mot clé `static`.

```
<?php
declare(strict_types=1);

class Calculs{
    public static function addition(float $a, float $b): float{
        return $a + $b;
    }
}

var_dump(Calculs::addition(150.0, 200.0));
```

Ici, on fait référence à la classe et non pas à un objet. On utilise donc “`self`” à la place de “`this`”. Ce qui transforme “`-`” en “`::`”.

Le mot `self` s'utilise au sein de la classe. Imaginons que l'on possède une variable `$taille`, une fonction `tailleMinimale()` et une fonction `set_Taille()`. Dans cette dernière, on utilisera `self::tailleMinimale($taille)` pour s'assurer que la taille minimale est conforme en faisant référence à notre fonction de classe.

Le mot clé “`static`” s'utilise aussi plus généralement dans des cas comme ci-dessous :

```
<?php
declare(strict_types=1);

class Pont
{
    // Définition de la propriété statique. Elle sera partagée entre toutes les instances.
    public static int $nombrePietons = 0;

    // Je laisse volontairement la méthode non statique. On appelle cette méthode au sein de chaque instance.
    // Seule la référence à la propriété est importante.
    public function nouveauPieton()
    {
        // Mise à jour de la propriété statique de la classe générale.
        self::$nombrePietons++;
    }
}

$pontLondres = new Pont;
$pontLondres->nouveauPieton();

$pontManhattan = new Pont;
$pontManhattan->nouveauPieton();

echo Pont::$nombrePietons;
```

On peut définir des valeurs immuables (inchangeables) avec les constantes :

```
<?php
class MaClasse {
    private const UNITE = "mètres"; // On y fait appel dans une fonction de cette manière : self::UNITE;
}
```

## CONSTRUCT, SETTER, GETTER...

Lors de la création d'une classe, PHP attribue des méthodes par défaut à la classe qui sont par défaut "cachée".

Ces méthodes permettent d'intervenir dans le processus de création, modification et suppression des objets.

On les appelle des méthodes magiques dont voici la liste:

### \_\_construct()

S'utilise pour déclarer les constructeurs dans une classe. C'est ce qui initialise les variables dans l'instance d'un objet.

```
class BaseClass {
    function __construct(int $id, string $text) {
        $this->id = $id;
        $this->text = $text;
    }
}

$object = new BaseClass(0, "bonjour");
```

### \_\_destruct()

Cette méthode est appelée dès qu'il n'y a plus de référence sur un objet donné. C'est effectué donc à chaque fois que le script se termine. On peut aussi l'appeler manuellement avec le mot clé "unset".

### \_\_call()

Cette méthode est appelée lorsqu'on invoque des méthodes inaccessibles dans un contexte objet. C'est ce qui permet d'effectuer \$monObjet->methode

### \_\_callStatic()

Même chose mais dans un contexte statique.

### \_\_get(), \_\_set(), \_\_unset(), \_\_isset()...

méthodes de manipulation et de récupération. set pour affecter des données, get pour les récupérer, isset pour vérifier si elles existent, unset pour les défaire. Ces méthodes agissent sur les propriétés private ou protected. Les publiques étant accessibles par défaut.

### \_\_sleep() et \_\_wakeup()

\_\_sleep() permet de linéariser une série de méthodes publiques en un tableau de données.  
\_\_wakeup() à l'inverse permet de retourner ce tableau en une série de propriétés. Ces méthodes

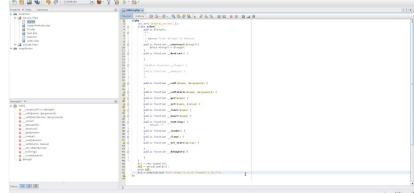
sont étroitement lié à `__serialize()` et `__unserialize()`.

Il est un peu difficile de comprendre ces méthodes alors voici une petite vidéo pouvant aider :

PHP magic methods: sleep and wakeup

This is a learning video in PHP magic methods. Sleep is called when serialized and wakeup when unserialized.

<https://www.youtube.com/watch?v=NI1kRzWTL5U>



### **\_\_toString()**

Détermine comment l'objet doit réagir lorsqu'il est traité comme une chaîne de caractères. Par exemple, ce que 'echo \$objet' affichera.

### **\_\_invoke()**

cette méthode permet de reconnaître l'objet comme une fonction. On peut donc faire un `$objet("argument")` et définir ce qui se passera à ce moment là dans la méthode invoke justement.

### **\_\_set\_state()**

Permet de définir la manière dont sera retournée la propriétés de classe ciblée lors de l'utilisation de `var_export()` (Une sorte de formatage du format)

### **\_\_clone()**

Permet de copier un objet existant en affectant une nouvelle instance. On peut aussi paramétriser des paramètres supplémentaires à l'intérieur de cette méthode (par exemple incrémenter une variable de +1 pour un compteur de clone etc...)

### **\_\_debugInfo()**

Méthode appelé lors qu'un `var_dump()` de l'objet. On peut configurer les propriétés qui seront retournées ou non.