Victor Thuillier

PROGRAMMEZ EN ORIENTÉ OBJET EN

PHP

2e édition

OPENCLASSROOMS

EYROLLES

PROGRAMMEZ EN ORIENTÉ OBJET EN

PHP

2e édition

Difficile aujourd'hui de passer à côté de la programmation orientée objet ! La quasi-totalité des outils créés actuellement pour les développeurs PHP utilise cette façon de programmer. Il est donc indispensable de savoir ce que c'est et comment s'en servir. Que vous ayez déjà quelques notions sur ce concept ou que vous soyez complètement novice en la matière, cet ouvrage est fait pour vous !

QU'ALLEZ-VOUS APPRENDRE ?

Théorie : les bases de la POO

- Introduction à la POO
- L'utilisation des classes
- L'opérateur de résolution de portée
- La manipulation des données stockées
- TP : minijeu de combat
- L'héritage
- TP : personnages spécialisés
- Les méthodes magiques

Théorie : techniques avancées

- Les objets en profondeur
- Les interfaces
- Les exceptions
- Les traits
- L'API de réflexivité
- UML : présentation
- UML : modélisation des classes
- Les design patterns
- TP : un système de news
- Les générateurs
- Les closures

Pratique : réalisation d'un site web

- Description de l'application
- Développement de la bibliothèque
- Le front-end
- Le back-end
- Gérer les formulaires

À PROPOS DE L'AUTEUR

Passionné par le Web, Victor Thuillier apprend grâce au site OpenClassrooms à créer son premier site à l'âge de 12 ans. Voulant aller plus loin, il décide d'approfondir ses connaissances dans le domaine, et plus particulièrement sur le langage PHP. Il découvre la programmation orientée objet à l'âge de 14 ans et s'en sert pour réaliser de nombreux sites Internet. Fort de son expérience, il rédige ensuite ce cours dans le but d'aider ceux qui, comme lui auparavant, souhaitent découvrir cette façon de programmer.

L'ESPRIT D'OPENCLASSROOMS

Des cours ouverts, riches et vivants, conçus pour tous les niveaux et accessibles à tous gratuitement sur notre plate-forme d'e-éducation : www.openclassrooms.com. Vous y vivrez une véritable expérience communautaire de l'apprentissage, permettant à chacun d'apprendre avec le soutien et l'aide des autres étudiants sur les forums. Vous profiterez des cours disponibles partout, tout le temps : sur le Web, en PDF, en eBook, en vidéo…

www.editions-eyrolles.com

PROGRAMMEZ EN ORIENTÉ OBJET EN

PHP

DANS LA MÊME COLLECTION

J. Pardanaud, S. de la Marck. – **Découvrez le langage JavaScript.**
N°14399, 2017, 478 pages.

A. Bacco. – **Développez votre site web avec le framework Symfony3.**
N°14403, 2016, 536 pages.

M. Chavelli. – **Découvrez le framework PHP Laravel.**
N°14398, 2016, 336 pages.

R. De Visscher. – **Découvrez le langage Swift.**
N°14397, 2016, 128 pages.

M. Lorant. – **Développez votre site web avec le framework Django.**
N°21626, 2015, 285 pages.

E. Lalitte. – **Apprenez le fonctionnement des réseaux TCP/IP.**
N°21623, 2015, 300 pages.

M. Nebra, M. Schaller. – **Programmez avec le langage C++.**
N°21622, 2015, 674 pages.

SUR LE MÊME THÈME

H. Bersini. – **La programmation orientée objet.**
N°67399, 7e édition, 2017, 696 pages.

P. Martin, J. Pauli, C. Pierre de Geyer, É. Daspet. – **PHP 7 avancé.**
N°14357, 2016, 732 pages.

E. Biernat, M. Lutz. – **Data science : fondamentaux et études de cas.**
N°14243, 2015, 312 pages.

B. Philibert. – **Bootstrap 3 : le framework 100 % web design.**
N°14132, 2015, 318 pages.

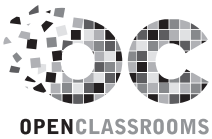
C. Delannoy. – **Le guide complet du langage C.**
N°14012, 2014, 844 pages.

Victor Thuillier

PROGRAMMEZ EN ORIENTÉ OBJET EN

PHP

2e édition



EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

ISBN : 978-2-212-14472-7

Introduction

Difficile aujourd'hui de passer à côté de la programmation orientée objet ! Peut-être n'en avez-vous d'ailleurs jamais entendu parler ; si c'est le cas, nous allons remédier à cela immédiatement ! La quasi-totalité des outils créés actuellement pour les développeurs PHP utilise cette façon de programmer, il est donc indispensable de savoir ce que c'est et comment s'en servir. Que vous ayez déjà quelques notions sur ce concept ou que vous soyez complètement novice en la matière, cet ouvrage est fait pour vous ! Nous partirons de zéro et découvrirons ensemble toute la puissance de cette nouvelle façon de développer.

Cet ouvrage a été pensé pour être suivi par quiconque possédant des connaissances en PHP. Nous commencerons par poser les bases de ce nouveau concept avec quelques exercices et travaux pratiques pour assimiler ces notions. Après avoir découvert des notions plus avancées, nous réaliserons ensemble un site web complet de A à Z ! Vous vous rendrez compte au fil de la lecture de cet ouvrage de la façon dont la programmation orientée objet vous permettra d'organiser vos projets de manière plus cohérente, assurera une maintenance facilitée et vous permettra de partager vos codes de façon plus aisée !

Mais avant de vous lancer dans ce (très) vaste domaine, il est indispensable, voire obligatoire :

- d'être à l'aise avec PHP et sa syntaxe ; si ce n'est pas le cas, suivez le cours « Concevez votre site web avec PHP et MySQL » *(http://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql)* ;
- d'avoir bien pratiqué ;
- d'être patient ;
- d'avoir au moins PHP 5.4 sur son serveur.

Si vous avez déjà pratiqué d'autres langages permettant la programmation orientée objet, c'est un gros plus, surtout si vous savez programmer en Java, car PHP a principalement tiré son modèle objet de ce langage.

On y va ?

Table des matières

Pratique : réalisation d'un site web 321

Première partie

Théorie : les bases de la POO

1 Introduction à la POO

Alors ça y est, vous avez décidé de vous lancer dans la POO en PHP ? Sage décision ! Nous allons donc plonger dans ce vaste domaine par une introduction à cette nouvelle façon de penser : qu'est-ce que la POO ? En quoi ça consiste ? En quoi est-ce si différent de la méthode que vous employez pour développer votre site web ? Tant de questions auxquelles je vais répondre.

Cependant, puisque je sais que vous avez hâte de commencer, nous allons entamer sérieusement les choses en créant notre première **classe** dès la fin de ce chapitre. Vous commencerez ainsi vos premiers pas dans la POO en PHP !

Qu'est-ce que la POO ?

Il était une fois le procédural

Commençons cet ouvrage par une question : comment est représenté votre code ? La réponse est unique : vous avez utilisé la « représentation procédurale » qui consiste à séparer le traitement des données des données elles-mêmes. Imaginons par exemple que vous avez un système de news sur votre site. D'un côté, vous avez les données (les news, une liste d'erreurs, une connexion à la BDD, etc.), et de l'autre côté, vous avez une suite d'instructions qui viennent modifier ces données. Si je ne me trompe pas, c'est de cette manière que vous codez.

Cette façon de se représenter votre application vous semble sans doute la meilleure, puisque c'est la seule que vous connaissez. D'ailleurs, vous ne voyez pas trop comment votre code pourrait être représenté de manière différente. Eh bien cette époque d'ignorance est révolue : voici maintenant la programmation orientée objet !

Naissance de la programmation orientée objet

Alors, qu'est-ce donc que cette façon de représenter son code ? La POO, c'est tout simplement faire de son site un ensemble d'objets qui interagissent entre eux. En d'autres termes, tout est objet.

Définition d'un objet

Je suis sûr que vous savez ce que c'est. D'ailleurs, vous en avez pas mal à côté de vous : je suis sûr que vous avez un ordinateur, une lampe, une chaise, un bureau, ou que sais-je encore. Ce sont tous des objets. En programmation, les objets sont sensiblement la même chose.

L'exemple le plus pertinent quand on fait un livre sur la POO est d'utiliser l'exemple du personnage dans un jeu de combat. Ainsi, imaginons que nous ayons un objet `Personnage` dans notre application. Un personnage a des caractéristiques :

- une force ;
- une localisation ;
- une certaine expérience ;
- et enfin des dégâts.

Toutes ces caractéristiques correspondent à des valeurs. Comme vous le savez sûrement, les valeurs sont stockées dans des variables. C'est toujours le cas en POO. Ce sont des variables un peu spéciales, mais nous y reviendrons plus tard.

Mis à part ces caractéristiques, un personnage a aussi des capacités. Il peut :

- frapper un autre personnage ;
- acquérir de l'expérience ;
- se déplacer.

Ces capacités correspondent à des fonctions. Comme pour les variables, ce sont des fonctions un peu spéciales et on y reviendra en temps voulu. En tout cas, le principe est là.

Vous savez désormais qu'on peut avoir des objets dans une application. Mais d'où sortent-ils ? Dans la vie réelle, un objet ne sort pas de nulle part. En effet, chaque objet est défini selon des caractéristiques et un plan bien précis. En POO, ces informations sont contenues dans ce qu'on appelle des **classes**.

Définition d'une classe

Comme je viens de le dire, les classes contiennent la définition des objets qu'on va créer par la suite. Prenons l'exemple le plus simple du monde : les gâteaux et leur moule. Le moule est unique. Il peut produire une quantité infinie de gâteaux. Dans ce cas, les gâteaux sont les *objets* et le moule est la *classe* : le moule va définir la forme du gâteau. La classe contient donc le plan de fabrication d'un objet, et on peut s'en servir autant qu'on veut afin d'obtenir une infinité d'objets.

Concrètement, une classe, c'est quoi ?

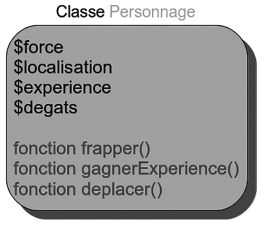
Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité. Dans le cas du personnage, nous aurons une fonction `frapper()`. Cette fonction devra simplement modifier la variable `$degats` du personnage en fonction de la variable `$force`. Une classe est donc un regroupement logique de variables et fonctions que tout objet issu de cette classe possédera.

Définition d'une instance

Une instance, c'est tout simplement le résultat d'une *instanciation*. Une *instanciation*, c'est le fait d'*instancier* une classe. *Instancier* une classe, c'est se servir d'une classe afin qu'elle crée un objet. En gros, une instance est un objet.

Exemple : création d'une classe

Nous allons créer une classe `Personnage` (sous forme de schéma, bien entendu). Celle-ci doit contenir la liste des variables et des fonctions citées précédemment : c'est la base de tout objet `Personnage`. Chaque instance de cette classe possédera ainsi toutes ces variables et fonctions (figure suivante).



Le schéma de notre classe

Vous voyez donc les variables et fonctions stockées dans la classe `Personnage`. Sachez qu'en réalité, on ne les appelle pas comme ça : il s'agit d'**attributs** (ou propriétés) et de **méthodes**. Un attribut désigne une variable et une méthode désigne une fonction.

Ainsi, tout objet `Personnage` aura ces attributs et méthodes. On pourra modifier ces attributs et invoquer ces méthodes sur notre objet, afin de modifier ses caractéristiques ou son comportement.

Le principe d'encapsulation

L'un des gros avantages de la POO est qu'on peut masquer le code à l'utilisateur (l'utilisateur est ici celui qui se servira de la classe, pas celui qui chargera la page depuis son navigateur). Le concepteur de la classe a englobé dans celle-ci un code qui peut être assez complexe et il est donc inutile, voire dangereux, de laisser l'utilisateur manipuler ces objets sans aucune restriction. Ainsi, il est important d'interdire à l'utilisateur de modifier directement les attributs d'un objet.

Prenons l'exemple d'un avion où sont disponibles des centaines de boutons. Chacun de ces boutons permet d'effectuer des actions sur l'avion. C'est l'*interface* de l'avion. Le pilote se moque de savoir de quoi est composé l'avion : son rôle est de le piloter. Pour cela, il va se servir des boutons afin de manipuler les composants de l'avion. Le pilote ne doit pas se charger de modifier manuellement ces composants : il pourrait faire de grosses bêtises.

Le principe est exactement le même pour la POO : l'utilisateur de la classe doit se contenter d'invoquer les méthodes en ignorant les attributs. Comme le pilote de l'avion, il n'a pas à trifouiller dedans. Pour instaurer une telle contrainte, on dit que les attributs sont **privés**. Pour l'instant, ceci peut sans doute vous paraître abstrait, mais nous y reviendrons.

Il est temps à présent de créer notre première classe !

Créer une classe

Syntaxe de base

Le but de cette section va être de traduire la figure précédente en code PHP. Mais auparavant, je vais vous donner la syntaxe de base de toute classe en PHP :

```
<?php
class Personnage // Présence du mot-clé class suivi du nom de la classe
{
  // Déclaration des attributs et méthodes ici
}
```

Cette syntaxe est à retenir absolument. Heureusement, elle est simple.

Ce qu'on vient de faire est donc de créer le moule, le plan qui définira nos objets. On verra dans le prochain chapitre comment utiliser ce plan afin de créer un objet. Pour l'instant, contentons-nous de construire ce plan et de lui ajouter des fonctionnalités.

La déclaration d'attributs dans une classe se fait en écrivant le nom de l'attribut à créer, précédé de sa **visibilité**.

Visibilité d'un attribut ou d'une méthode

La visibilité d'un attribut ou d'une méthode indique à partir d'où on peut y avoir accès. Nous allons voir ici deux types de visibilité : `public` et `private`.

Le premier, `public`, est le plus simple. Si un attribut ou une méthode est `public`, alors on pourra y avoir accès depuis n'importe où, depuis l'intérieur de l'objet (dans les méthodes qu'on a créées), comme depuis l'extérieur. Je m'explique. Quand on crée un objet, c'est principalement pour pouvoir exploiter ses attributs et méthodes. L'extérieur de l'objet, c'est tout le code qui n'est pas *dans* votre classe. En effet, quand vous créerez un objet, celui-ci sera représenté par une variable, et c'est à partir de celle-ci qu'on pourra modifier l'objet, appeler des méthodes, etc. Vous allez donc dire à PHP « dans cet objet, donne-moi cet attribut » ou « dans cet objet, appelle cette méthode » : c'est ça, appeler des attributs ou méthodes depuis l'extérieur de l'objet.

Le second, `private`, impose quelques restrictions. On aura accès aux attributs et méthodes *seulement* depuis l'intérieur de la classe, c'est-à-dire que seul le code voulant accéder à un attribut privé ou une méthode privée écrit(e) *à l'intérieur* de la classe fonctionnera. Sinon, une jolie erreur fatale s'affichera, disant que vous ne pouvez pas accéder à telle méthode ou tel attribut parce qu'il ou elle est privé(e).

Là, ça devrait faire *tilt* dans votre tête : le principe d'encapsulation ! C'est de cette manière qu'on peut interdire l'accès à nos attributs.

Création d'attributs

Pour déclarer des attributs, on va donc les écrire entre les accolades, les uns à la suite des autres, en faisant précéder leurs noms du mot-clé `private`, comme ceci :

```
<?php
class Personnage
{
  private $_force;         // La force du personnage
  private $_localisation; // Sa localisation
  private $_experience;    // Son expérience
  private $_degats;        // Ses dégâts
}
```

Vous pouvez constater que chaque attribut est précédé d'un underscore (_). Ceci est une notation qu'il est préférable de respecter (il s'agit de la notation PEAR), qui dit que chaque nom d'élément privé (ici il s'agit d'attributs, mais nous verrons plus tard qu'il peut aussi s'agir de méthodes) doit être précédé d'un underscore.

Vous pouvez initialiser les attributs lorsque vous les déclarez (par exemple, leur mettre une valeur de 0 ou autre). Exemple :

```
<?php
class Personnage
{
  private $_force = 50;             // La force du personnage, par défaut à 50.
  private $_localisation = 'Lyon'; // Sa localisation, par défaut à Lyon.
  private $_experience = 1;         // Son expérience, par défaut à 1.
  private $_degats = 0;             // Ses dégâts, par défaut à 0.
}
```

La valeur que vous leur donnez par défaut doit être une expression scalaire statique. Par conséquent, leur valeur ne peut, par exemple, pas être issue d'un appel à une fonction (`private $_attribut = strlen('azerty')`) ou d'une variable, superglobale ou non (`private $_attribut = $_SERVER['REQUEST_URI']`). Si votre version de PHP est antérieure à la 5.6, vous ne pouvez spécifier que des valeurs statiques, ce qui rend impossible l'assignation du résultat d'une opération. Par exemple, vous ne pouvez pas écrire de `private $_attribut = 1 + 1` ou bien `private $_attribut = 'Hello ' . 'world !'`.

Création de méthodes

Pour la déclaration de méthodes, il suffit de faire précéder la visibilité de la méthode du mot-clé `function`. Les types de visibilité des méthodes sont les mêmes que ceux des attributs. Les méthodes n'ont en général pas besoin d'être masquées à l'utilisateur, vous les mettrez souvent en `public` (à moins que vous teniez absolument à ce que l'utilisateur ne puisse pas appeler cette méthode, par exemple s'il s'agit d'une fonction qui simplifie certaines tâches sur l'objet, mais qui ne doit pas être appelée n'importe comment).

```
<?php
class Personnage
{
  private $_force;        // La force du personnage
  private $_localisation; // Sa localisation
  private $_experience;   // Son expérience
  private $_degats;       // Ses dégâts

  public function deplacer() // Une méthode qui déplacera le personnage
                             // modifiera sa localisation.
  {

  }

  public function frapper() // Une méthode qui frappera un personnage
                            // suivant la force qu'il a.
  {

  }

  public function gagnerExperience() // Une méthode augmentant l'attribut
                                     // $experience du personnage.
  {

  }
}
```

Et voilà !

De même que l'underscore précède les noms d'éléments privés, vous pouvez remarquer que le nom des classes commence par une majuscule. Il s'agit aussi du respect de la notation PEAR.

En résumé

- Une classe, c'est un ensemble de variables et de fonctions (attributs et méthodes).
- Un objet, c'est une *instance* de la classe pour pouvoir l'utiliser.
- Tous vos attributs doivent être privés. Pour les méthodes, peu importe leur visibilité. C'est ce qu'on appelle le principe d'encapsulation.
- On déclare une classe avec le mot-clé `class` suivi du nom de la classe, et enfin deux accolades ouvrantes et fermantes qui encercleront la liste des attributs et méthodes.