

Victor Thuillier

PROGRAMMEZ EN ORIENTÉ OBJET EN

PHP

2^e 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'OPENCCLASSROOMS

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...



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, 7^e 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.

Retrouvez nos bundles (livres papier + e-book) et livres numériques sur
<http://izibook.eyrolles.com>

Victor Thuillier

PROGRAMMEZ EN ORIENTÉ OBJET EN

PHP

2^e édition

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

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.

ISBN : 978-2-212-14472-7

© OpenClassrooms, 2017
© Groupe Eyrolles, 2017, pour la présente édition

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

Théorie : les bases de la POO	1
1 Introduction à la POO	3
Qu'est-ce que la POO ?	3
<i>Il était une fois le procédural</i>	3
<i>Naissance de la programmation orientée objet</i>	4
<i>Définition d'un objet</i>	4
<i>Définition d'une classe</i>	4
<i>Définition d'une instance</i>	5
<i>Exemple : création d'une classe</i>	5
<i>Le principe d'encapsulation</i>	6
Créer une classe	6
<i>Syntaxe de base</i>	6
<i>Visibilité d'un attribut ou d'une méthode</i>	6
<i>Création d'attributs</i>	7
<i>Création de méthodes</i>	8
En résumé	9
2 L'utilisation des classes	11
Créer et manipuler un objet	11
<i>Créer un objet</i>	11
<i>Appeler les méthodes de l'objet</i>	11
<i>Accéder à un élément depuis la classe</i>	12
<i>Implémenter d'autres méthodes</i>	14
<i>Exiger des objets en paramètres</i>	17
Les accesseurs et mutateurs	18
<i>Accéder à un attribut : l'accesseur</i>	18

<i>Modifier la valeur d'un attribut : les mutateurs</i>	19
<i>Retour sur notre script de combat</i>	21
Le constructeur	22
L'autochargement de classes	25
En résumé	27
3 L'opérateur de résolution de portée	29
Les constantes de classe	29
Les attributs et méthodes statiques	32
<i>Les méthodes statiques</i>	32
<i>Les attributs statiques</i>	34
En résumé	36
4 La manipulation des données stockées	37
Une entité, un objet	37
<i>Rappels sur la structure d'une BDD</i>	37
<i>Travailler avec des objets</i>	38
L'hydratation	42
<i>La théorie de l'hydratation</i>	42
<i>L'hydratation en pratique</i>	43
Gérer sa BDD correctement	49
<i>Une classe, un rôle</i>	49
<i>Les caractéristiques d'un manager</i>	50
<i>Les fonctionnalités d'un manager</i>	50
<i>Essayons tout ça !</i>	53
En résumé	54
5 TP : minijeu de combat	55
Ce que nous allons faire	55
<i>Cahier des charges</i>	55
<i>Notions utilisées</i>	56
<i>Préconception</i>	56
Première étape : le personnage	57
<i>Les caractéristiques du personnage</i>	57
<i>Les fonctionnalités d'un personnage</i>	58
<i>Les getters et setters</i>	60
<i>L'hydratation des objets</i>	61
<i>Codons le tout !</i>	62
Deuxième étape : le stockage en base de données	64
<i>Les caractéristiques d'un manager</i>	65
<i>Les fonctionnalités d'un manager</i>	65
<i>Codons le tout !</i>	67

Troisième étape : l'utilisation des classes.	69
Améliorations possibles	81
6 L'héritage	83
La notion d'héritage.	83
<i>Définition</i>	83
<i>Procéder à un héritage</i>	84
<i>Redéfinir les méthodes</i>	86
<i>L'héritage à l'infini !</i>	88
Un nouveau type de visibilité : <i>protected</i>	89
Imposer des contraintes	90
<i>L'abstraction</i>	91
<i>La finalisation</i>	92
La résolution statique à la volée	94
<i>Cas complexes</i>	97
<i>L'utilisation de <code>static::</code> dans un contexte non statique</i>	102
En résumé	103
7 TP : personnages spécialisés	105
Ce que nous allons faire.	105
<i>Cahier des charges</i>	105
<i>Des nouvelles fonctionnalités pour chaque personnage</i>	106
<i>La base de données</i>	106
<i>Le coup de pouce du démarrage</i>	106
<i>Le personnage</i>	107
<i>Le magicien</i>	108
<i>Le guerrier</i>	108
Correction	109
Améliorations possibles	121
8 Les méthodes magiques	123
Le principe	123
La surcharge magique des attributs et méthodes	124
<i>__set</i> et <i>__get</i>	124
<i>__isset</i> et <i>__unset</i>	127
<i>__call</i> et <i>__callStatic</i>	131
La linéarisation des objets.	132
<i>Posons le problème</i>	133
<i>serialize</i> et <i>__sleep</i>	134
<i>unserialize</i> et <i>__wakeup</i>	135
Les autres méthodes magiques.	136
<i>__toString</i>	136

__set_state	137
__invoke	138
__debugInfo	139
En résumé	140
Théorie : techniques avancées	141
9 Les objets en profondeur	143
Un objet, un identifiant	143
Comparer des objets	146
Parcourir des objets	149
En résumé	151
10 Les interfaces	153
Présentation et création d'interfaces	153
<i>Le rôle d'une interface</i>	153
<i>Créer une interface</i>	153
<i>Implémenter une interface</i>	154
<i>Les constantes d'interfaces</i>	156
Hériter ses interfaces	156
Les interfaces prédéfinies	157
<i>Définition d'un itérateur</i>	157
<i>L'interface Iterator</i>	158
<i>L'interface SeekableIterator</i>	159
<i>L'interface ArrayAccess</i>	161
<i>L'interface Countable</i>	165
<i>Bonus : la classe ArrayIterator</i>	168
En résumé	169
11 Les exceptions	171
Une différente gestion des erreurs	171
<i>Lancer une exception</i>	171
<i>Attraper une exception</i>	173
Des exceptions spécialisées	175
<i>Hériter la classe Exception</i>	175
<i>Emboîter plusieurs blocs catch</i>	177
<i>Exemple concret : la classe PDOException</i>	179
<i>Exceptions prédéfinies</i>	179
<i>Exécuter un code même si l'exception n'est pas attrapée</i>	180
Gérer les erreurs facilement	181
<i>Convertir les erreurs en exceptions</i>	181

<i>Personnaliser les exceptions non attrapées</i>	183
En résumé	184
12 Les traits	185
Le principe des traits	185
<i>Le problème</i>	185
<i>Résoudre le problème grâce aux traits</i>	186
<i>Utiliser plusieurs traits</i>	188
<i>La résolution des conflits</i>	189
<i>Les méthodes de traits et méthodes de classes</i>	189
Aller plus loin avec les traits	191
<i>Définition d'attributs</i>	191
<i>Conflit entre attributs</i>	191
<i>Traits composés d'autres traits</i>	192
<i>Changer la visibilité et le nom des méthodes</i>	192
<i>Les méthodes abstraites dans les traits</i>	194
En résumé	195
13 L'API de réflexivité	197
Obtenir des informations sur ses classes.	197
<i>Les informations propres à la classe</i>	198
<i>Les relations entre classes</i>	199
Obtenir des informations sur les attributs des classes.	202
<i>Instanciation directe</i>	202
<i>Récupération des attributs d'une classe</i>	202
<i>Le nom et la valeur des attributs</i>	203
<i>La portée de l'attribut</i>	204
<i>Les attributs statiques</i>	205
Obtenir des informations sur les méthodes des classes.	206
<i>Création d'une instance de ReflectionMethod</i>	206
<i>Publique, protégée ou privée ?</i>	207
<i>Abstraite ou finale ?</i>	208
<i>Constructeur ou destructeur ?</i>	209
<i>Appeler la méthode sur un objet</i>	209
Utiliser des annotations	210
<i>Présentation d'addendum</i>	211
<i>Récupérer une annotation</i>	212
<i>Savoir si une classe possède telle annotation</i>	213
<i>Une annotation à multiples valeurs</i>	214
<i>Des annotations pour les attributs et méthodes</i>	215
<i>Imposer une annotation à une cible précise</i>	216
En résumé	217

14 UML : présentation	219
UML, kézako ?	219
Modéliser une classe	221
<i>Première approche</i>	221
<i>Exercices</i>	223
Modéliser les interactions	223
<i>L'héritage</i>	224
<i>Les interfaces</i>	224
<i>L'association</i>	225
<i>L'agrégation</i>	226
<i>La composition</i>	226
En résumé	227
 15 UML : modélisation des classes	 229
Les bons outils	229
<i>Installation</i>	229
Modéliser une classe	231
<i>Créer une classe</i>	231
<i>Modifier une classe</i>	232
<i>Gestion des options de la classe</i>	232
<i>La gestion des attributs</i>	233
<i>La gestion des constantes</i>	234
<i>La gestion des méthodes</i>	235
<i>La gestion du style de la classe</i>	238
Modéliser les interactions	239
<i>Création des liaisons</i>	239
<i>Exercice</i>	243
Exploiter son diagramme	243
<i>Enregistrer son diagramme</i>	243
<i>Exporter son diagramme</i>	245
En résumé	246
 16 Les design patterns	 247
Laisser une classe créant les objets : le pattern Factory	247
<i>Le problème</i>	247
Écouter les objets : le pattern Observer	249
<i>Le problème</i>	249
<i>Exemple concret</i>	252
<i>ErrorHandler : une classe gérant les erreurs</i>	253
<i>MailSender : une classe s'occupant d'envoyer les e-mails</i>	253
<i>BDDWriter : une classe s'occupant de l'enregistrement en BDD</i>	254
<i>Testons le code !</i>	254
<i>Des classes anonymes pour les observateurs</i>	255

Séparer les algorithmes : le pattern Strategy	256
<i>Le problème</i>	256
<i>Exemple concret</i>	257
<i>Allégeons le code avec les classes anonymes</i>	260
Une classe, une instance : le pattern Singleton	261
<i>Le problème</i>	261
L'injection de dépendances	263
<i>Pour conclure</i>	266
En résumé	267
17 TP : un système de news	269
Ce que nous allons faire	269
<i>Cahier des charges</i>	269
<i>Retour sur le traitement des résultats</i>	270
<i>Gestion efficace des dates</i>	272
Correction	273
<i>Le diagramme UML</i>	273
<i>Le code du système</i>	273
18 Les générateurs	287
Les notions de base	287
<i>Étude de cas</i>	287
<i>Les générateurs</i>	289
Zoom sur les valeurs retournées	292
<i>Retourner des clés avec les valeurs</i>	292
<i>Retourner une référence</i>	294
Les coroutines	297
<i>La méthode send</i>	297
<i>La méthode throw</i>	302
En résumé	307
19 Les closures	309
Création de closures	309
<i>La syntaxe</i>	309
<i>Un exemple d'utilisation</i>	310
<i>L'utilisation de variables extérieures</i>	311
Lier une closure	312
<i>Lier une closure à un objet</i>	312
<i>Lier temporairement une closure à un objet</i>	314
<i>Lier une closure à une classe</i>	315
<i>Les liaisons automatiques</i>	317
<i>Implémentation du pattern Observer à l'aide de closures</i>	318
En résumé	320

Pratique : réalisation d'un site web 321

20 Description de l'application 323

Une application, qu'est ce que c'est ?	323
<i>Le déroulement d'une application</i>	323
<i>Un peu d'organisation</i>	326
Les entrailles de l'application	328
<i>Retour sur les modules</i>	328
<i>Le back controller de base</i>	329
<i>La page</i>	330
<i>L'autoload</i>	332
Résumé du déroulement de l'application	336

21 Développement de la bibliothèque 337

L'application	337
<i>Présentation</i>	337
<i>La requête du client</i>	338
<i>La réponse envoyée au client</i>	339
<i>Retour sur notre application</i>	341
<i>Les composants de l'application</i>	343
Le routeur	344
<i>Réfléchissons, schématisons</i>	344
<i>Codons</i>	347
Le back controller	351
<i>Réfléchissons, schématisons</i>	351
<i>Codons</i>	352
<i>Accéder aux managers depuis le contrôleur</i>	354
<i>Petit rappel sur la structure d'un manager</i>	354
<i>La classe Managers</i>	354
<i>À propos des managers</i>	356
La page	358
<i>Réfléchissons, schématisons</i>	359
<i>Codons</i>	360
<i>Retour sur la classe BackController</i>	361
<i>Retour sur la méthode HTTPResponse::redirect404()</i>	362
Bonus 1 : l'utilisateur	362
<i>Réfléchissons, schématisons</i>	362
<i>Codons</i>	363
Bonus 2 : la configuration	365
<i>Réfléchissons, schématisons</i>	365
<i>Un format pour le fichier</i>	365
<i>L'emplacement du fichier</i>	365
<i>Le fonctionnement de la classe</i>	365
<i>Codons</i>	366

22 Le front-end	369
L'application	369
<i>La classe FrontendApplication</i>	369
<i>Le layout</i>	370
<i>Les deux fichiers de configuration</i>	372
<i>L'instanciation de FrontendApplication</i>	373
<i>Réécrire toutes les URL</i>	374
Le module de news	374
<i>Les fonctionnalités</i>	374
<i>La structure de la table news</i>	375
<i>L'action index</i>	377
<i>L'action show</i>	380
L'ajout de commentaires	383
<i>Le cahier des charges</i>	383
<i>La structure de la table comments</i>	383
<i>L'action insertComment</i>	385
<i>L'affichage des commentaires</i>	388
23 Le back-end	393
L'application	393
<i>La classe BackendApplication</i>	393
<i>Le layout</i>	395
<i>Les deux fichiers de configuration</i>	395
Le module de connexion	396
<i>La vue</i>	396
<i>Le contrôleur</i>	397
Le module de news	397
<i>Les fonctionnalités</i>	398
Les commentaires	407
<i>Les fonctionnalités</i>	407
24 Gérer les formulaires	417
Le formulaire	417
<i>La conception</i>	417
<i>Le développement de l'API</i>	421
<i>Testons nos nouvelles classes</i>	426
Les validateurs	428
Le constructeur de formulaires	433
<i>La conception des classes</i>	434
<i>Le développement des classes</i>	435
<i>L'ajout de l'autoload</i>	437
<i>La modification des contrôleurs</i>	438
Le gestionnaire de formulaires	442
<i>La conception du gestionnaire de formulaires</i>	442

<i>Le développement du gestionnaire de formulaires</i>	<i>443</i>
<i>La modification des contrôleurs.</i>	<i>444</i>

Annexe	445
---------------	------------

L'opérateur instanceof	447
-------------------------------	------------

Présentation de l'opérateur.	447
L'opérateur instanceof et l'héritage	449
L'opérateur instanceof et les interfaces.	450
En résumé	452

Index	453
--------------	------------

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 ?

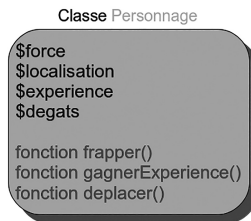
Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité. Dans le cas du personnage, nous aurons une fonction `frapper()`. Cette fonction devra simplement modifier la variable `$degats` du personnage en fonction de la variable `$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 créez 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 `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.