La programmation par objet (POO) en PHP 7 - 8

Partie 1

Les classes et objets

La programmation par objet (POO) a été intégrée au langage PHP dans sa version 4. Mais à cette époque, le modèle objet de PHP était beaucoup trop sommaire. Nous ne pouvions réellement parler de programmation orientée objet.

Les développeurs de PHP se sont alors penchés sur la question et ont amélioré ce modèle objet qui, depuis la version 5, n'a plus rien à envier aux autres langages objets comme Java ou C++.

PHP 7 a encore amélioré le concept, des changements mineurs mais cependant très utiles sont présents dans les versions de PHP 8.* .

Quels sont les avantages et inconvénients d'une approche objet ?

La programmation orientée objet offre de nombreux avantages :

La possibilité de réutiliser le code dans différents projets. Les classes ainsi créées pourront avoir une nouvelle vie dans une application tierce.

Une conception de l'algorithme plus claire et organisée. Le programmeur identifie chaque élément de son programme comme un objet ayant son contexte, ses propriétés et des actions qui lui sont propres.

Un code modulaire. Chaque type d'objet possède son propre contexte et ne peut agir avec d'autres qu'avec des interfaces bien précises. Cela permet d'isoler chaque module et d'en créer séparément de nouveaux qui viendront s'ajouter à l'application.Cette approche est particulièrement employée dans le cas de projets répartis entre plusieurs développeurs.

Possibilité de s'adapter aux design patterns (motifs de conception) pour une meilleure structuration du code (par exemple MVC).

Il existe cependant quelques inconvénients à l'utilisation de la programmation par objet :

Une application orientée objet mal conceptualisée sera difficilement maintenable et modulaire.
La programmation par objet nécessite généralement plus de ressources et de temps d'exécution qu'un code procédural.

Remarque : la programmation par objet n'est pas synonyme de bonne programmation ! Il existe d'excellents développeurs capables de produire du bon code procédural, maintenable, structuré et modulaire.

Mais il existe aussi des développeurs en programmation objet qui produisent du mauvais code suite à un manque de conceptualisation, ou qui en font "trop".

Qu'est-ce qu'un objet ?

Définition

Un « objet » est une représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés et des actions.

Par exemple : une voiture, une personne, un animal, un nombre ou bien un compte bancaire peuvent être vus comme des objets.

Attributs

Les « attributs » (aussi appelés « données membres ») sont les caractères propres à un objet.

Une personne, par exemple, possèdent différents attributs qui lui sont propres comme le nom, le prénom, la couleur des yeux, le sexe, la couleur des cheveux, la taille...

Méthodes

Les « méthodes » sont les actions applicables à un objet.

Un objet personne, par exemple, dispose des actions suivantes : manger, dormir, boire, marcher, courir...

Qu'est-ce qu'une classe ?

Une « classe » est un modèle de données définissant la structure commune à tous les objets qui seront créés à partir d'elle. Plus concrètement, nous pouvons percevoir une classe comme un moule grâce auquel nous allons créer autant d'objets de même type et de même structure qu'on le désire.

Par exemple, pour modéliser n'importe quelle personne, nous pourrions écrire une classe Personne dans laquelle nous définissons les attributs (couleurs des yeux, couleurs des cheveux, taille, sexe...) et méthodes (marcher, courir, manger, boire...) communs à tout être humain.

Qu'est-ce qu'une instance ?

Une instance est une représentation particulière d'une classe.

Lorsque l'on crée un objet, on réalise ce que l'on appelle une « instance de la classe ». C'est à dire que du moule, on en extrait un nouvel objet qui dispose de ses attributs et de ses méthodes. L'objet ainsi créé aura pour type le nom de la classe.

Par exemple, les objets Hugo, Romain, Nicolas, Daniel sont des instances (objets) de la classe Personne.

Remarque : une classe n'est pas un objet. C'est un abus de langage de dire qu'une classe et un objet sont identiques.

Déclaration d'une classe

Nous venons de définir le vocabulaire propre à la programmation orientée objet. Entrons à présent dans le vif du sujet, c'est-à-dire la déclaration et l'instanciation d'une classe. Nous allons déclarer une classe Personne qui nous permettra ensuite de créer autant d'instances (objets) de cette classe que nous le souhaitons.

Syntaxe de déclaration d'une classe

Le code suivant présente une manière de déclarer et de structurer correctement une classe.

Déclaration d'une classe PHP 7

```
<?php
  class NomDeMaClasse
  {
    // Attributs
    // Constantes
    // Méthodes
  }
?>
```

Voici par exemple ce que cela donne avec notre exemple :

Exemple de la classe Personne

```
<?php
  class Personne
  {
    // Attributs
    public $nom;
    public $prenom;
    public $dateDeNaissance;
    public $taille;
    public $sexe;

    // Constantes
    const NOMBRE_DE_BRAS = 2;
    const NOMBRE_DE_JAMBES = 2;
    const NOMBRE_DE_YEUX = 2;
    const NOMBRE_DE_PIEDS = 2;
    const NOMBRE_DE_MAINS = 2;

    // Méthodes
    public function __construct() { }

    public function boire()
    {
   echo 'La personne boit<br/>';
    }

    public function manger()
    {
   echo 'La personne mange<br/>';
    }
  } ?>
```

Remarque : par convention, on écrit le nom d'une classe en « Upper Camel Case », c'est-à-dire que tous les mots sont accrochés et chaque première lettre de chaque mot est écrit en capital.

Comme vous pouvez le constater, nous avons déclaré 5 attributs publics, 5 constantes, 1 méthode constructeur et 2 méthodes classiques. Détaillons chaque élément.

Les attributs

Les attributs sont les caractéristiques propres d'un objet. Toute personne possède un nom, un prénom, une date de naissance, une taille, un sexe... Tous ces éléments caractérisent un être humain.

Par cet exemple, nous déclarons les attributs de notre classe public. Il existe trois niveaux de visibilité (public, private et protected) qui peuvent être appliqués à un attribut.

Le mot-clé public permet de rendre l'attribut accessible depuis l'extérieur de la classe. Ce n'est pas une bonne pratique à adopter mais nous l'utiliserons tel quel pour faciliter la compréhension.

En programmation orientée objet, un attribut n'est ni plus ni moins qu'une variable.

Notez également que nous avons juste déclaré les attributs. En revanche, aucun type ni aucune valeur ne leur ont été attribués. Ils sont donc par défaut initialisés à la valeur NULL.

Remarque : deux classes différentes peuvent avoir les mêmes attributs sans risque de conflit.

Les constantes

Il est aussi possible de déclarer des constantes propres à la classe. Contrairement au mode procédural de programmation, une constante est déclarée avec le mot-clé const.

Remarque : une constante doit être déclarée et initialisée avec sa valeur en même temps.

Le constructeur

Le constructeur est une méthode particulière. C'est elle qui est appelée implicitement à la création de l'objet (**instanciation**).

Dans notre exemple, le constructeur n'a ni paramètre ni instruction. Le programmeur est libre de définir des paramètres obligatoires à passer au constructeur ainsi qu'un groupe d'instructions à exécuter à l'instanciation de la classe. Nous nous en passerons pour simplifier notre exemple.

Remarque :
L'interprétation PHP de la surcharge est différente de celle de la plupart des langages orientés objet. La surcharge, habituellement, fournit la possibilité d'avoir plusieurs méthodes portant le même nom mais avec une quantité et des types différents d'arguments.

Les méthodes

Les méthodes sont les actions que l'on peut appliquer à un objet. Il s'agit en fait de fonctions qui peuvent prendre ou non des paramètres et retourner ou non des valeurs / objets. S'agissant d'actions, nous vous conseillons de les nommer avec un verbe à l'infinitif.

Elles se déclarent de la même manière que des fonctions traditionnelles.

Au même titre que les attributs, on déclare une méthode avec un niveau de visibilité.

Le mot-clé public indique que l'on pourra appliquer la méthode en dehors de la classe, c'est à dire sur l'objet lui même.

Remarque : deux classes différentes peuvent avoir les mêmes méthodes sans risque de conflit.

Utilisation des classes et des objets

Notre classe est désormais prête mais ne sert à rien toute seule. Comme nous l'avons expliqué plus haut, une classe est perçue comme un moule capable de réaliser autant d'objets de même type et de même structure qu'on le souhaite. Nous allons donc présenter maintenant la phase de concrétisation d'une classe.

Instanciation d'une classe

L'instanciation d'une classe est la phase de création des objets issus de cette classe. Lorsque l'on instancie une classe, on utilise le mot-clé **new** suivant du nom de la classe. Cette instruction appelle la méthode constructeur (**__construct()**) qui construit l'objet et le place en mémoire. Voici un exemple qui illustre 4 instances différentes de la classes Personne.

Création d'objets de type Personne

```
<?php
  $personne1 = new Personne();
  $personne2 = new Personne();
  $personne3 = new Personne();
  $personne4 = new Personne();
?>
```

Un objet est en fait une variable dont le type est celui de la classe qui est instanciée.

Remarque : si nous avions défini des paramètres dans la méthode constructeur de notre classe, nous aurions dû les indiquer entre les parenthèses au moment de l'instance. Par exemple : $personne1 = new Personne('Hamon','Hugo');

Accès aux attributs

Accès en écriture

Nous venons de créer 4 objets de même type et de même structure. Dans l'état actuel, ce sont des clones car leurs attributs respectifs sont tous déclarés mais ne sont pas initialisés. Nous affectons à présent des valeurs à chacun des attributs de chaque objet.

Utilisation des attributs d'un objet

```
<?php
  // Définition des attributs de la personne 1
  $personne1->nom = 'Hamon';
  $personne1->prenom = 'Hugo';
  $personne1->dateDeNaissance = '02-07-1987';
  $personne1->taille = '180';
  $personne1->sexe = 'M';
  // Définition des attributs de la personne 2
  $personne2->nom = 'Dubois';
  $personne2->prenom = 'Michelle';
  $personne2->dateDeNaissance = '18-11-1968';
  $personne2->taille = '166';
```

```
  $personne2->sexe = 'F';
  // Définition des attributs de la personne 3
  $personne3->nom = 'Durand';
  $personne3->prenom = 'Béatrice';
  $personne3->dateDeNaissance = '02-08-1975';
  $personne3->taille = '160';
  $personne3->sexe = 'F';
  // Définition des attributs de la personne 4
  $personne4->nom = 'Martin';
  $personne4->prenom = 'Pierre';
  $personne4->dateDeNaissance = '23-05-1993';
  $personne4->taille = '155';
  $personne4->sexe = 'M';
?>
```

Nous avons maintenant des objets ayant chacun des caractéristiques différentes.

Dans notre exemple, nous accédons directement à la valeur de l'attribut. Cela est possible car nous avons défini l'attribut comme étant public. Si nous avions déclaré l'attribut avec les mots-clés private ou protected, nous aurions dû utiliser un autre mécanisme pour accéder à sa valeur ou bien la mettre à jour.

Accès en lecture

La lecture de la valeur d'un attribut d'un objet se fait exactement de la même manière que pour une variable traditionnelle. Le code suivant présente comment afficher le nom et le prénom de la personne 1.

Affichage du nom et du prénom de la personne 1

```
<?php
 echo 'Personne 1 :<br/><br/>';
 echo 'Nom : ', $personne1->nom ,'<br/>';
 echo 'Prénom : ', $personne1->prenom;
?>
```

L'exécution de ce programme produit le résultat suivant sur la sortie standard :

Résultat d'exécution du code

```
Personne 1 :
Nom : Hamon
Prénom : Hugo
```

Accès aux constantes

L'accès aux constantes ne peut se faire qu'en lecture via l'opérateur :: . L'exemple suivant illustre la lecture d'une constante de la classe Personne.

Accès à une constante

```
<?php
 echo 'Chaque personne a ', Personne::NOMBRE_DE_YEUX ,' yeux.';
?>
```

L'exécution de ce code affiche la chaîne suivante sur la sortie standard :

Résultat de l'exécution du code
Chaque personne a 2 yeux.

Remarque : si l'on tente de redéfinir la valeur d'une constante, PHP génèrera une erreur de ce type :

Erreur générée en cas de redéfinition de constante

```
Parse error: syntax error, unexpected '=' in
/Users/Emacs/Sites/Demo/Autres/Personne.php on line 36
```

Accès aux méthodes

Leur utilisation est exactement la même que pour les attributs. Rappelez-vous, nous avions défini deux méthodes pour notre classe Personne. Il s'agit des méthodes boire() et manger(). Toutes deux affichent une chaîne de caractères sur la sortie standard lorsqu'elles sont exécutées.

Reprenons nos 4 objets précédents. Nous simulons que ces 4 personnes sont à table et qu'elles dînent. Simultanément les deux premières personnes boivent le contenu d'un verre pendant que les deux autres mangent. Cela se représente donc par l'appel de la méthode boire() sur les objetspersonne1 et personne2, puis par l'appel de la méthode manger() sur les objets personne3 etpersonne4.

Appel de méthode sur des objets

```
<?php
  $personne1->boire();
  $personne2->boire();
  $personne3->manger();
  $personne4->manger();
?>
```

Après exécution, le résultat est le suivant :

La personne boit
La personne boit
La personne mange
La personne mange

Remarque : comme pour le constructeur, si nos méthodes avaient eu besoin de paramètres en entrée, nous les aurions indiqués au moment de l'appel entre les parenthèses de la fonction.