PHP - 12 : Les bases de la programmation orientée objet (P.O.O.)

Introduction

Il existe plusieurs types de programmation ou paradigmes (un concept), dont la programmation orientée objet qui consiste à modéliser le code d'une application en tant qu'objet.

Définitions

Objet

Tout d'abord, qu'est-ce qu'**un objet** ?

C'est évident pour les objets physiques (une chaise, un smartphone, une voiture, un stylo... sont des objets) - dits tangibles (= qu'on peut toucher) - mais en P.O.O. il faut se représenter tout élément d'une application comme un étant un objet, c'est-à-dire que même des choses plus "abstraites" (ou intangibles) deviennent des objets : un utilisateur d'une application, un compte en banque, un fichier à manipuler ou même une connexion à la base de données.

Un objet possède :

- des **attributs** (ou propriétés ou encore caractéristiques), par exemple un numéro, un nom, une dimension... qui en définissent un état de l'objet.
- des **fonctionnalités**, par exemple : avancer, calculer, enregistrer en base de données...

Ces attributs et fonctionnalités permettent de définir l'objet et de le distinguer des autres objets (qui ont des attributs et des fonctionnalités différentes). L'objet possède ainsi un état (= les attributs) et un comportement (= les méthodes).

Prenons pour exemple un objet *voiture* :

- ses caractéristiques sont de posséder un nom de marque, un nom de modèle, une puissance, un nombre de roues, un type de carrosserie, une couleur, un kilométrage etc.
- ses fonctionnalités sont de rouler, freiner etc.

> Souvenez-vous (vu en Javascript), les langages informatiques orientés objet possèdent des objets natifs, directement utilisables. C'est le cas aussi en PHP.

Classe

La base de la P.O.O. est la classe, qui permet de définir l'ensemble des attributs et fonctionnalités d'un objet : c'est le moule qui va pouvoir fabriquer les exemplaires d'un objet.

Dans la classe, les caractéristiques de l'objet se nomment attributs et les fonctionnalités se nomment **méthodes** (concrètement : des fonctions).

Les attributs et méthodes sont appelés **membres** de la classe.

On déclare une classe par le mot clé `class`, suivi du nom qu'on souhaite lui donner. Puis tout le code de la classe sera compris dans un bloc d'accolades :

```php
<?php
class vehicule
{

    [ CODE ]

} // -- fin de la classe
?>
```

On enregistre ce fichier avec l'extension `.php`. Une bonne pratique consiste à nommer le fichier avec le nom de la classe.

> Une autre bonne pratique est d'ajouter le suffixe `.class` dans le nom de fichier ce qui permet d'identifier immédiatement qu'il s'agit d'une classe, ce qui donne au final, par exemple, `vehicule.class.php` et, aussi, de placer ces fichiers dans un répertoires nommé `classes`.

Attributs

A l'intérieur de la classe, on trouve en premier lieu des variables : ce sont les attributs de la classe.

Exemple

```php
class voiture
{
   public $_marque;
   public $_modele;
   public $_version;
   protected $_immatriculation;
   private $_nbKm = 0;
```

Les attributs peuvent être directement affectés d'une valeur (mais pas d'une expression : calcul, concaténation, résultat d'appel de fonction).

Comme on peut le voir dans l'exemple, les attributs sont précédés d'un mot-clé définissant leur visibilité.

Visibilité

La **visibilité** est un mécanisme de sûreté qui permet de mettre en place un contrôle d'accès aux variables.

Il existe 3 types de visibilité :

- Publique (mot-clé `public`) : tout le code de l'application peut avoir accès à la variable et la manipuler.
- Privée (mot-clé `private`) : seule la classe dans laquelle la variable est définie peut y avoir accès.
- Protégée (mot-clé `protected`) : la classe parente (dans laquelle est définie la variable) et les classes filles peuvent avoir accès (nous verrons plus loin sur les notions de classes parente/fille).

Exemples

```php
class vehicule
{
   public $_marque;
   public $_modele;
   public $_version;
   protected $_immatriculation;
   private $_nbKm = 0;
```

Ces 4 types peuvent ensuite être manipulés dans les méthodes de la classe ou les méthodes des classes filles selon leur visibilité.

> A noter que la visibilité d'un attribut dans une classe parente peut être redéfinie dans une classe fille, toutefois dans ce cas la visibilité ne peut être abaissée (si on a par exemple `protected` dans la classe mère, on ne pourra pas redéfinir à `private` dans la classe fille.

Mutateurs et accesseurs

L'encapsulation et les principes de visibilité édictent que les attributs d'uen classe ne devraient jamais être manipulés directement à l'extérieur de la classe (pour éviter des accididents tels que suppression, mauvaise valeur etc.).

¨Par conséquent, les attributs devraient toujours être déclarés comme privés, avec le mot-clé `private`. et être manipulés ensuite par des méthodes appelées accesseurs et mutateurs (*getters* et *setters* en anglais).

Mutateur

La méthode qui sert de mutateur permet de définir la valeur d'un attribut. Elle reçoit en argument la variable qui contient cette valeur.

Par convention, les mutateurs ont le préfixe `set` et il est pratique de leur donner le nom de l'attribut concerné :

```php
<?php
// vehicule.class.php

// Mutateur : définit/modifie la valeur passée en argument à l'attribut
public function setMarque($sMarque)
{
   return $this->_marque = $sMarque;
}
```

> Il faut écrire une méthode mutateur **pour chaque attribut**.

Accesseur

```php
 <?php
// vehicule.class.php

// Accesseur : renvoie la valeur d'un attribut
public function getMarque()
{
    return $this->_marque;
}
```

Méthodes

Après les déclarations d'attributs et de constantes, on trouve des fonctions, qui, dans une classe, sont appelées **méthodes**. Celles-ci ont un comportement similaire aux fonctions standard (traitement d'opération, réception d'arguments et retour de variables).

Exemple

```php
class vehicule
{
   public $_marque;
   public $_modele;
   protected $_immat;
   private $_nbKm = 0;

   // Méthode
   public function avancer(int $nbKm2)
   {
      $this->$_nbKm = $this->$_nbKm + $nbKm2;
      return $this->_nbKm;
   }
} // -- fin de la classe
?>
```

Comme on peut le voir, l'appel à un attribut de la classe au sein d'une méthode n'a pas besoin de recevoir l'attribut en argument.

> Comme pour les attributs, les méthodes ont une visibilité qui ont les mêmes valeurs `public`, `private` et `protected` et le même fonctionnement que pour les attributs. En général, les méthodes sont publiques sinon elles ne pourraient pas être appellées par le reste de l'application, mais certains cas peuvent nécessiter une visibilité privée ou protégée.

Instance et utilisation d'une classe

L'**instance** (ou occurrence) est un exemplaire de la classe (*fabriqué par le même moule*). Par abus de langage, on dit souvent objet, mais cela n'est donc pas tout à fait exact.

Exemple :

- Paul possède une Renault Clio, version RS, couleur rouge, achetée en 2018, qui a un numéro de série unique (ce qui est le cas d'une grande majorité de produits manufacturés), par exemple *1234*.
- Pierre possède exactement le même modèle que Paul, de couleur rouge et achetée en 2018 aussi; ce qui la distingue de celle de Paul ce sera un numéro de série différent (par exemple *7856*).

Les voitures de Pierre et de Paul seront donc 2 instances différentes de la classe `voiture`. Elles ont été fabriquées dans un même "moule" (la chaîne d'assemblage d'une même usine) = ce moule est **la classe**.

Pour utiliser une classe, Il faut au préalable que le fichier contenant la classe ait été chargé avec `include` (ou `require`) :

```php
<?php
include("classes/vehicule.class.php");
```

puis on va l'appeler et créer un objet : cet objet sera **une instance** (ou **instanciation**) de la classe. Pour cela, on utilise dans le code de l'application l'instruction `new` suivie du nom de la classe :

```php
$oVehicule = new vehicule();
```

Une fois une instance initialisée, on va pouvoir faire appel aux méthodes de la classe, via le signe `->` :

```php
$oVehicule->avancer(5);
```

Les principes de la P.O.O.

Encapsulation

Un objet rassemble en lui-même (au sein de la classe) ses données (les attributs, représentant l'état de l'objet), et le code capable d'agir dessus (les méthodes) : on dit que les attributs et les méthodes sont **encapsulées**, on a des variables avec une visibilité et il faut appeler la classe pour pouvoir accéder à ces variables (elles ne peuvent pas l'être directement).

Héritage

La P.O.O. présente un mécanisme dit d'héritage :

- Une classe B peut utiliser les membres d'une classe A; on dit alors que B hérite de A (ou : dérive, étend),
- La classe A est alors appelée classe parente (ou : mère, super-classe),
- La classe B est alors appelée classe fille (ou : enfant, héritée, dérivée, sous-classe)

L'héritage peut être résumé par la formule *la classe B est une sorte (sous-espèce) de la classe A*, la classe B ayant des caractéristiques communes avec la classe parente A.

Par exemple : voiture, camion, bus, moto, bateau sont des types de véhicules :

- ils ont en commun des attributs (un nom de marque, de modèle, une puissance, une couleur etc.) et des méthodes (avancer, freiner...)
- mais ils ont certains points de différences : un camion possède un poids maxi, un bus un nombre de passagers etc.

Possibilités et limites de l'héritage :

- Une classe fille ne peut posséder qu'une seule classe parente mais avoir une infinité de classes filles.
- Une classe mère peut posséder une infinité de classes filles (ex : la classe `vehicules` pourraient avoir pour classes filles `voiture`, `moto`, `camion`, `bus`, `velo`, `bateau` etc.).

L'héritage peut se faire faire sur plusieurs niveaux hiérarchiques : par exemple une classe C hérite de la classe B qui hérite elle-même de la classe A (donc ici sur 3 niveaux).

Pour implémenter l'héritage, la classe fille doit utiliser le mot-clé `extends` suivi du nom de la classe parente (ou classe mère) dans la définition de la classe :

```php
<?php
class voiture extends vehicule
{
    [ CODE ]
} // -- fin de la classe
```

- Ici, la classe `voiture` hérite de la classe `vehicule` : la classe `voiture` est donc la classe fille (ou classe dérivée) et la classe `vehicule` la classe parente.
- La classe `voiture` aura accès (selon la visibilité des membres) aux attributs et méthodes de la classe `vehicule` parente. Le mot clé `parent::` devra être utilisé pour appleer un attribut/méthode de la classe parente. (par exemple `parent::avancer()` appelle la méthode `avancer()` de la classe parente).

Polymorphisme

Le polymorphisme permet à une méthode d'adopter plusieurs "formes" dans des classes différentes, c'est-à-dire d'être redéfinie dans des classes filles : une méthode d'une classe fille ne contiendra pas le même code que la méthode portant le même nom dans la classe parente.

Exemple

Dans la classe parente `vehicule`, la fonction `avancer()` gèrera des kilomètres pour les classes filles `voiture`, `bus` ou `camion`, tandis qu'elle retournera des milles nautiques (*miles* en anglais) pour la classe fille `bateau` car elle y est codée de la sorte :

Classe fille `voiture` :

```php
<?php
class voiture extends vehicule
{
   public function avancer(int $nbKm2)
   {
      $this->$_nbKm = $this->$_nbKm + $nbKm2;
      return $this->_nbKm;
   }
}
?>
```

Classe fille `bateau` :

```php
<?php
class bateau extends vehicule
{
   private $_nbMiles = 0;

   public function avancer(int $nbMiles)
   {
      $this->$_nbMiles = $this->$_nbMiles + $nbMiles2;
      return $this->_nbMiles;
   }
}
```

> Attention, cet exemple est volontairmeent très simpliste; dans la réalité c'est plus complexe : il existe en fait plusieurs types de polymorphisme qu'on rencontre selon le langage implémenté. En PHP, le polymorphisme se limite à la redéfinition (principe de surcharge) d'une méthode dans une classe fille.

Points particuliers

Constructeur

Le constructeur est une méthode qui permet de lancer des opérations automatiquement lors d'une instanciation

Le constructeur doit être déclaré avec la méthode suivante : `__construct()` et peut recevoir des arguments.

> Le constructeur est facultatif en PHP (mais obligatoire dans certains langages).

Exemple

Dans cet exemple, on initiale le nombre de roues

```php
class voiture
{
   public $_marque;
   public $_modele;
   private $_roues;

   // Définition du constructeur de la classe
   function __construct($nbroues)
   {
        $this->_roues = 4;
   }

   [ AUTRES METHODES ]

} // -- fin de la classe
```

Appel de la classe : on doit alors passer les valeurs des arguments à initialiser :

```php
$oVoiture = new voiture('Renault', 'Clio', 2018);
```

Destructeur

L'inverse du constructeur est le destructeur : `__destruct()` qui permet de déréférencer des valeurs (attributs de l'objet).

Elle reste optionnelle et peu utilisée. Elle revêt une utilité dans certains cas, par exemple pour fermer une ressource, par exemple un fichier ouvert avec la fonction `fopen()`.

Constructeurs et destructeurs PHP

Conclusion

Il faut bien noter que nous avons abordé dans cette séquence uniquement les bases. La programmation orientée objet présente en effet des notions beaucoup plus avancées (clonage, itérateurs, classes abstraites, interfaces, traits, design patterns, introspection, héritage multiple dans certains langages...).

Exemple

Le code complet de la classe `produits` permettant de gérer les produits du projet *Jarditou*.

Remarquez que les méthodes sont celles qui effectuent les opérations de CRUD sur la table `produits`.

```php
class Produits
{
   private $_pro_id;
   private $_pro_cat_id;
   private $_pro_ref;
   private $_pro_libelle;
   private $_pro_description;
   private $_pro_prix;
   private $_pro_stock;
   private $_pro_couleur;
   private $_pro_photo;
   private $_pro_d_ajout;
   private $_pro_d_modif;
   private $_pro_bloque;

   // Accesseur : retourne la valeur courante de l'attribut
   public function getPro_id()
   {
        return $this->_pro_id;
   }

   // Mutateur : fixe/modifie la valeur passée en argument à l'attribut
   public function setPro_id($id)
   {
       return $this->_pro_id = $id;
   }

   // Afficher l'enregistrement
   public function afficher($pro_id)
   {
       // "SELECT * FROM produits WHERE pro_id = ".$pro_id";
   }
} // -- fin de la classe
```

Utilisation de la classe dans un script PHP :

```php
// Chargement de la classe
require("classes/produits.class.php");

// Création d'une instance
$oProduit = new Produits();

// Définition du produit à utiliser, ici le produit 7 en base de données
$oProduits->setPro_id(7);

// On veut afficher l'id du produit :
echo $oProduit->getPro_id();
```

Exercices

- Réalisez l'exercice suivant.

Ressources

- Pierre Giraud
- OpenClassrooms