PHP

-

Programmation orientée objet (POO)

La programmation orientée objet c'est quoi?

Dans le monde de la programmation, il existe deux grands concepts à savoir, la **programmation procédurale** et la **programmation orientée objet**.

Programmation procédurale

Dans la programmation procédurale (ou fonctionnelle), un programme est divisés en plusieurs étapes qui s'exécutent d'une manière séquentielle. Durant ces étapes, l'ensembles des variables mises en jeu sont modifiées par le biais de procédures ou fonctions qui se succèdent.

Malgré la simplicité que l'on peut toucher lors de la programmation procédurale, celle ci a plusieurs inconvénients que je vais résumer dans les points suivants:

- **Entretien du programme complexe**: puisque le programme s'exécute séquentiellement du début à la fin, le fait de vouloir modifier son comportement (même légèrement) revient à faires de modifications (parfois majeures) dans plusieurs endroits.
- **Avancement lent dans les grandes applications**: le fait de coder de grandes applications à l'aide d'un langage de programmation procédurale requière beaucoup de temps. Chose qui n'est pas toujours à portée de main lorsqu'il s'agit de clients exigeants.
- **Travail en groupe non favorable**: les grands projets sont généralement réalisés par une équipe constituée de plusieurs programmeurs, chacun s'attribut une tâche (ou une partie du projet). Le fait de vouloir reconstituer l'application finale n'est pas toujours chose aisée, ce qui demande généralement de repasser en vue certains bouts de codes.

Les langages de programmation procédurale les plus connus sont: C, Pascal, Perl, PHP4, PL/SQL...

Programmation orientée objet (POO)

Vers les années 1970, le concept de l'objet est né. La programmation orientée objet se base sur la manipulation d'entités appelés **objets**. Les objets sont plus concrets et plus perceptibles par les humains. Ils peuvent représenter des entités physiques du quotidien comme une personne, une maison, une entreprise...

Un objet du POO possède une structure interne (qui définit ses propriétés) et ses propres comportements et peut interagir avec d'autres objets.

Si la programmation orientée objet gagne du terrain c'est principalement grâce à ses nombreux avantages qu'on peut résumer dans ces points:

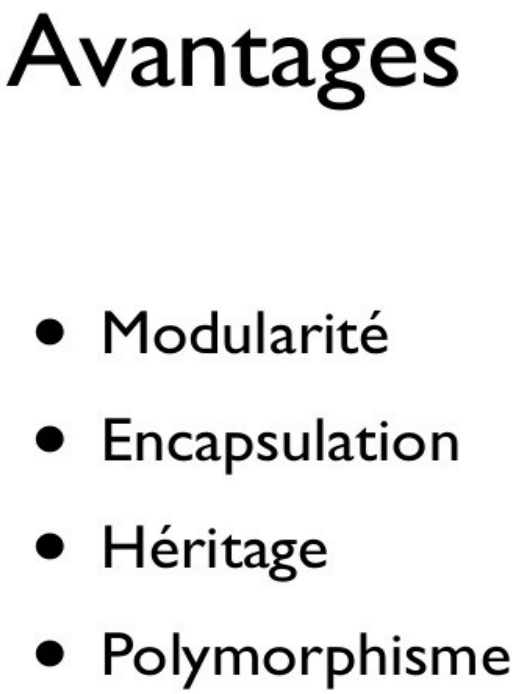
- **Plus intuitive**: La POO permet de manipuler des objets qui peuvent interagir entre eux. Ce concept est plus proche du monde physique où nous vivons.
- **Plus organisé**: Les objets étant indépendants les uns des autres (même s'ils peuvent interagir entre eux), le code source devient plus organisé et plus claire.
- **Plus évolutif**: Un code organisé est plus facile à maintenir.
- **Code réutilisable**: En POO le code peut être divisé en modules plus faciles à réutiliser, non pas seulement dans le projet courant, mais dans d'autres projets.
- **Favorise le travail en équipe**: A l'inverse de la programmation procédurale, la POO favorise le travail en équipe en découpant les parties du projet faciles à regrouper après.

Les langages de programmation orientés objet les plus connus sont: C++, Java, PHP5 (et PHP7), C#, Ruby, Python...

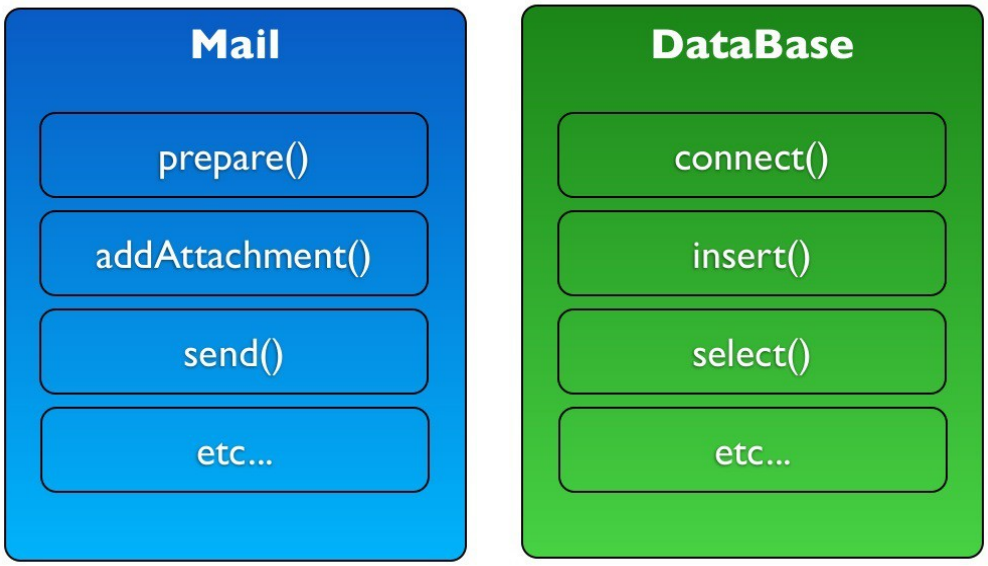
Programmation orientée objet en PHP

La programmation orientée objet a fait son apparition en PHP depuis sa version 4, mais le concept n'était pas assez mature comme c'est le cas pour des langages comme Java ou C++. Avec l'apparition de PHP5 beaucoup de fonctionnalités ont été ajoutées au langage. Cependant, l'ultime avancée technologique consiste à la prise en charge de la POO comme elle se doit.

Dès qu'on parle de PHP5, on pense implicitement au PHP Orienté Objet. Bien qu'on peut continuer à programmer en procédural même avec les nouvelles fonctionnalités de PHP5, il est évident que la POO a pris le dessus surtout avec l'intégration de ses véritables concepts, comme l'héritage, les interfaces, les classes abstraites etc...

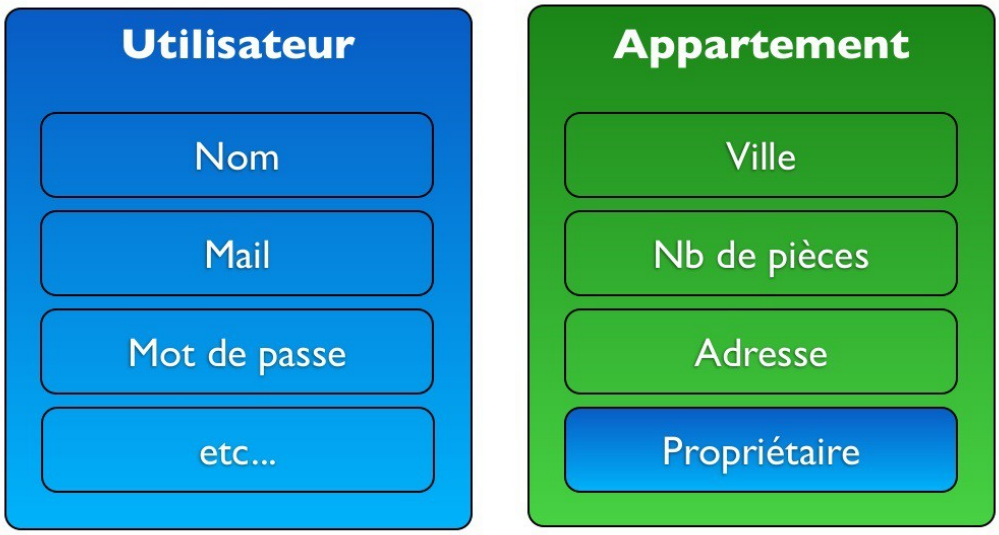


Modularité



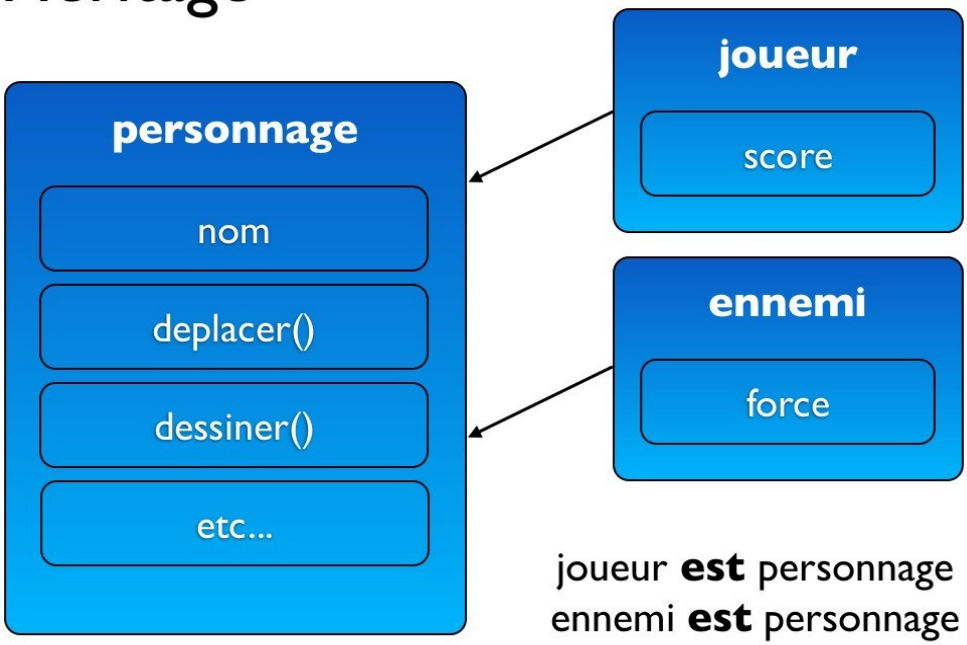
Séparation des données et des services par entité logique.

Encapsulation



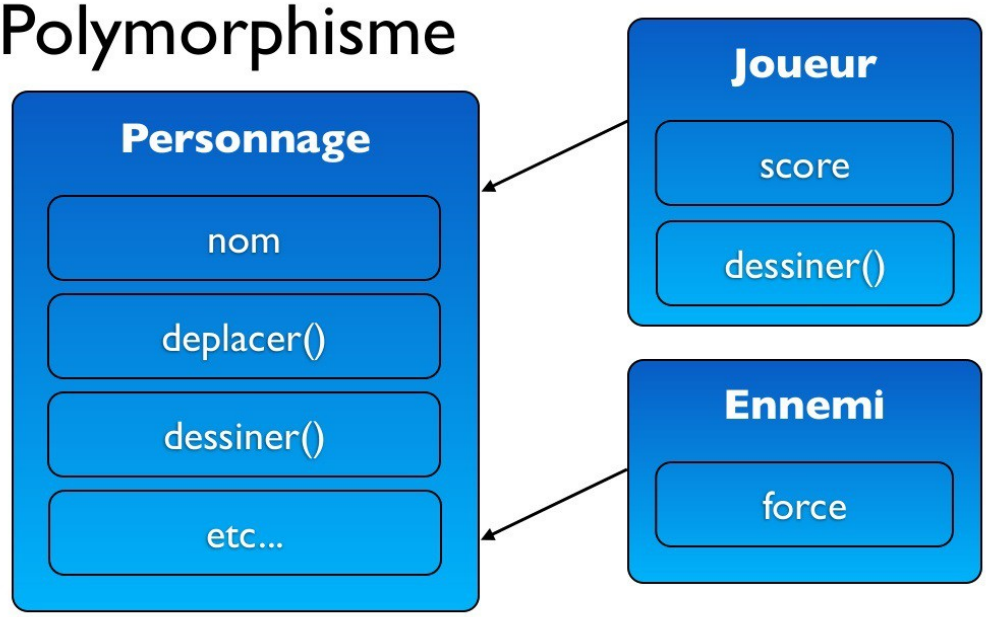
Propriétaire est une donnée de type Utilisateur
associée à un Appartement

Héritage



joueur **est** personnage
ennemi **est** personnage

Polymorphisme



Service **dessiner()** :
générique pour un Ennemi, spécifique pour un Joueur

Les bases de la POO en PHP

Généralités

Qu'est ce qu'un objet?

La programmation orientée objet est plus naturelle donc plus intuitive. Si c'est le cas, c'est parce qu'elle utilise des entités appelés **objets**. Un objet possède sa propre structure interne qui définit ses propriétés et son comportement. Si on compare avec le monde réel, les objets sont partout autour de nous. Une personne est un objet, une voiture en est un autre, une maison, une école une télé, un ballon... tous sont des objets.

Prenons l'objet "voiture". Bien qu'il en existe plusieurs modèles, le fait d'évoquer le mot "voiture" fait penser à des points comme:

•couleur

•options

•puissance du moteur

•vitesse

•nombre de rapports de vitesse

•...

Ces points représentent les caractéristiques (ou propriétés) de l'objet voiture. Dans le jargon de la POO on les appelles **des attributs**.

Cependant, une voiture peut aussi entamer des actions, par exemple:

•Accélérer

•Ralentir

•Tourner à droite

•Tourner à gauche

•Reculer

•...

Vous avez sans doute remarqué que j'ai utilisé des verbes pour spécifier les actions que peut entreprendre une voiture. L'ensemble de ces actions constitue le comportement de celle-ci. En POO on les appelles **des méthodes**.

Une classe c'est quoi?

Les objets de la POO doivent être créés d'abord pour pouvoir être manipulés après. C'est la **classe** qui se charge de donner vie aux objets.

Une classe est une structure cohérente de propriétés (attributs) et de comportements (méthodes). C'est elle qui contient la définition des objets qui vont être créés après. En général on considère une classe comme un moule à objets. Avec un seul moule on peut créer autant d'objets que l'on souhaite.

Techniquement parlant, une classe est une structure qui contient des attributs (appelés aussi variables membres) et des méthodes (connus également par fonctions membres). Les méthodes agissent sur les attributs de cette structure. Par exemple, le fait d'**accélérer** augmente la **vitesse** de la voiture.

Vocabulaire

Classe
Un nouveau type de données,
comportant des *attributs* et des *méthodes*

Attribut
Une donnée propre à une classe

Méthode
Une fonction propre à une classe

L'instanciation

L'instanciation est le fait de créer une **instance**. Pour être précis, on parle d'une **instance de classe**. La classe étant le moule qui sert à fabriquer les objets, alors chaque objet créé correspond à une instance de la classe qui lui a donné vie.

Vocabulaire

Instancier une classe
Allouer la mémoire nécessaire pour une classe

Objet
Une instance de classe

```php
$pierre = new Joueur;
$paul   = new Joueur;

$pierre->score = 10;

$paul->dessiner();
```

Vocabulaire

classe = type

Types en PHP :

- Avant : NULL, bool, int, float, string, array.
- Maintenant : Mail, Produit, Utilisateur, Voiture...

objet = instance en mémoire

En première approx. : objet = variable de type classe

Mais deux variables peuvent en fait pointer sur le même objet.

Créer une classe

Déclaration de la classe

La classe renferme l'ensemble des propriétés et de méthodes qui servent à définir l'identité de l'objet qui en découlera (l'instance de classe).

Supposons que l'on souhaite créer une classe "voiture". Celle ci aura certaines propriétés (attributs) comme: couleur, puissance et vitesse. Elle aura aussi des méthodes comme accélérer et ralentir. Notre classe "voiture" renfermera tous ces éléments (membres).

Pour créer la classe "voiture" on écrira le code suivant:

```
<?php
   class Voiture{
      // Déclaration des membres
   }
?>
```

le mot clé **class** sert à indiquer que l'on veut créer une classe. Il est suivi du nom de celle ci, **Voiture** dans ce cas, suivi de deux accolades (ouvrante et fermante) qui contiendront la déclaration des éléments membres (attributs et méthodes).

> Vous avez remarqué que le nom de la classe commence par une majuscule. En fait, il est pour coutume de capitaliser les noms des classes en PHP. Cependant, si vous ne faites pas de même, la classe fonctionnera aussi.

Déclaration des attributs - Principe d'encapsulation

Les attributs sont les variables membres de la classe. Ils constituent les propriétés ou les caractéristiques de l'objet (l'instance de classe) qui en sera né.

Pour déclarer un attribut il faut le précéder par sa **visibilité**. La visibilité d'un attribut indique à partir d'où on peut en avoir accès. Il existe trois types de visibilité:

•**public**: dans ce cas, l'attribut est accessible de partout (de l'intérieur de la classe dont il est membre comme de l'extérieur).

•**private**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre.

•**protected**: dans ce cas, l'attribut est accessible seulement de l'intérieur de la classe dont il est membre ainsi que de l'intérieur des classes fille qui héritent de cette classe (Nous verrons l'héritage plus loin dans ce cous).

Cependant, c'est la visibilité **private** (ou **protected** qui est considérée comme extension de **private**) qui est recommandée. On parle alors du **principe d'encapsulation.**

Principe d'encapsulation:

L'encapsulation est un principe fondamental de la POO. Il vise à masquer les attributs aux utilisateurs du code (les programmeurs qui se serviront de la classe par la suite). En fait, ce qui est important dans une classe ce sont les attributs. Les méthodes ne font qu'agir sur ceux ci. Le fait d'exposer les attributs aux utilisateur peut compromettre le bon fonctionnement de la classe. Il faut donc les masquer et leur limiter l'accès uniquement de l'intérieur de la classe par le biais des méthodes prévues à cet effet.

Pour mieux comprendre comment la manipulation directe des attributs peut compromettre le bon fonctionnement de la classe, il n'y a pas mieux qu'un exemple.

Imaginons que nous conduisons une voiture et que nous voulons accélérer pour atteindre une vitesse élevée. On sait que tout est question de mécanique sous le capot. Il suffirait alors d'actionner quelques engrenages et quelques courroies pour augmenter la vitesse. Cependant, il est inconcevable de faire cela en manipulant les pièces mécaniques directement, d'abord parce que c'est dangereux, et aussi parce qu'on peut abîmer la mécanique de la voiture en procédant à des fausses manœuvres. Or, l'habitacle de la voiture est déjà équipé de pédales qui permettent d'accélérer ou ralentir sans danger et sans même être connaisseur en mécanique.

Dans cet exemple, on peut assimiler les pièces mécaniques sous le capot aux attributs et les pédales aux méthodes.

Définissons maintenant les attributs de la classe "Voiture":

```
<?php
   class Voiture{
      private $couleur="Rouge";
      private $puissance=130;
      private $vitesse=0;
   }
?>
```

Vous avez remarqué que les attributs sont des variables PHP. Or, ce ne sont pas des variables classiques car pour les manipuler il faut passer par l'instance de classe (ou parfois, par la classe elle même).

Dans cet exemple nous avons déclaré les attributs en les initialisant. Cependant, c'est une pratique peu courante, car l'initialisation des attributs est prévue ailleurs (on verra ça plus loin).

Le code précédent peut donc être écrit simplement comme ceci:

```
<?php
   class Voiture{
      private $couleur;
      private $puissance;
      private $vitesse;
   }
?>
```

Déclaration des méthodes

Les méthodes sont des fonctions membres. Ce sont elles qui se chargent de manipuler les attributs et dotent ainsi la classe de son comportement.

Pour déclarer une méthode il suffit de procéder comme si on déclarait une fonction en PHP en la précédant par la visibilité qui peut être soit **public** ou **private**. Une méthode publique est accessible de partout (de l'intérieur comme de l'extérieur de la classe). Une méthode privée est accessible seulement de l'intérieur de la classe. Elle est généralement sollicitée par une autre méthode publique.

Supposons que je veux maintenant ajouter les méthodes **accelerer()** et **ralentir()** à la classe **Voiture**, le code ressemblerait à ceci:

```
<?php
   class Voiture{
      private $couleur;
      private $puissance;
      private $vitesse;
      public function accelerer(){
         // Corps de la méthode accelerer()
      }
      public function ralentir(){
         // Corps de la méthode ralentir()
      }
   }
?>
```

Instanciation de classes

et

manipulation d'objets

Instancier une classe

La classe est le moule qui sert à fabriquer les objets. Le nombre d'objet que l'on peut créer à l'aide d'une classe est illimité. On appelle l'opération qui consiste à créer un objet **instanciation** et l'objet ainsi créé peut aussi être appelé **instance de classe**.

Reprenons l'exemple de la page précédente:

```
<?php
   class Voiture{
      private $couleur="Rouge";
      private $puissance;
      private $vitesse;
      public function accelerer(){
         echo "Appel de la méthode accelerer()";
      }
      public function ralentir(){
         echo "Appel de la méthode ralentir()";
      }
   }
?>
```

J'ai pris le soin d'initialiser l'attribut $couleur et mettre quelque chose dans les deux méthodes accelerer() et ralentir().

Supposons qu'on veut créer un objet du nom de **$citadine** à partir de la classe **Voiture**. L'instanciation de la classe ressemblerait à ceci:

```
<?php
   $citadine = new Voiture();
?>
```

N'oubliez pas de placer le dollars avant l'identifiant de l'objet.

Nous avons désormais créé l'objet **$citadine** qui dispose de ses propres attributs $couleur, $puissance, et $vitesse et les méthodes accelerer() et relentir(). Autrement dit, si nous créons un autre objet à partir de la même classe, il disposera également de ses propres attributs et méthodes du même nom. Cependant, même si les identifiants des membres des deux objets créés sont similaires, ils n'ont rien à voir les uns des autres.

Appel d'attributs et méthodes

Après avoir créé l'objet, on peut désormais accéder à ses membres (attributs et méthodes) par le biais de l'opérateur **->** (un tiret suivi d'un chevron fermant).

Voici un exemple:

```
<?php
   class Voiture{
      private $couleur="Rouge";
      private $puissance;
      private $vitesse;
      public function accelerer(){
         echo "Appel de la méthode accelerer()";
      }
      public function ralentir(){
         echo "Appel de la méthode ralentir()";
      }
   }

   $citadine = new Voiture();

   $citadine -> accelerer();
   echo $citadine -> couleur;
?>
```

Ce qui donne:

> Appel de la méthode accelerer()
> **Fatal error**: Cannot access private property Voiture::$couleur in **index.php** on line **17**

La première ligne de l’exécution est logique. On a pu appeler la méthode accelerer() depuis l'extérieur de la classe car elle est publique. Ce qui affiche le message "Appel de la méthode accelerer()". Nous avons utilisé l'opérateur **->** pour désigner la méthode accelerer() de l'objet $citadine (instance de la classe Voiture).

La deuxième ligne par contre affiche l'erreur "Cannot access private property" qui signifie que l'on peut pas accéder à un attribut privé de l'extérieur de la classe, ce qui est tout à fait logique. Si l'attribut $couleur était public, alors le message "Rouge" serait affiché à la place de l'erreur.

Notez que quand on veut accéder à un attribut par le biais de l'opérateur **->**, le dollars de celui-ci est exclu.