

14) Programmation Orientée Objet(POO)

Comme son nom l'indique nous allons parler d'objet en programmation dans ce chapitre.

Définition : La *programmation orientée objet (POO)* est un modèle de programmation informatique qui met en œuvre une conception basée sur les objets. Elle se différencie de la programmation procédurale, qui est basée sur l'utilisation de procédures, et de la programmation fonctionnelle, qui elle, repose entièrement sur le concept de *fonction*.

Qu'est-ce que c'est qu'un objet ? De quoi s'agit-il ?

On peut voir les objets comme des types de variables supplémentaires. Généralement en PHP on travaille avec 3 types de variables :

- Les nombres (entier, décimal, etc.)
- Les chaînes de caractères
- Les tableaux (simple, indexé, à plusieurs dimensions, etc.)

Le problème c'est que ces 3 types de variables sont un peu limitées. Si jamais on veut créer une sorte de réseau social et de sauvegarder des informations sur l'utilisateur pour par exemple vérifier si l'utilisateur a le droit de faire tel ou tel chose ou suit tel ou tel personne, etc. On va devoir utiliser des tableaux mais ça va vite devenir très lourd et compliqué même pour récupérer des informations, il faudra les parcourir. Avec le système d'objet vous verrez qu'on pourra s'en sortir beaucoup plus facilement.

La programmation orientée objet repose sur 5 concepts fondamentaux à savoir :

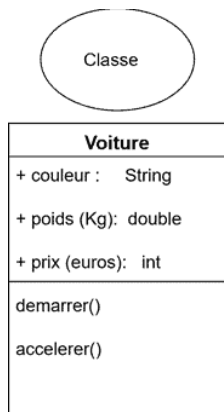
- La classe
- L'objet
- L'encapsulation
- L'héritage
- Le polymorphisme (**ATTENTION pas en PHP**)

Nous allons voir quelques un de ces concepts.

Le concept de Classe

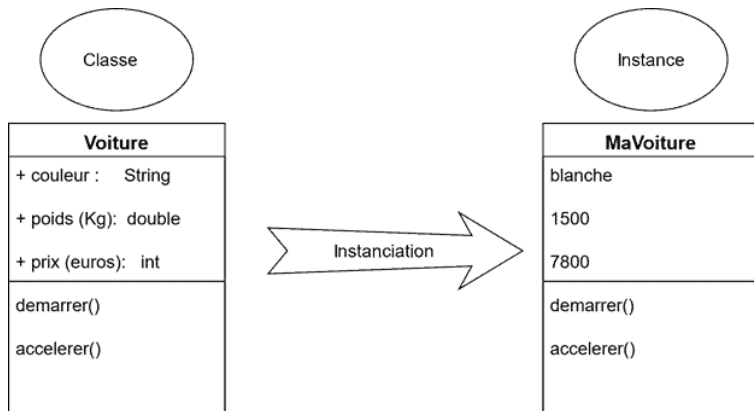
Le premier concept fondamental de l'orientée objet est la classe. Une classe est une structure abstraite qui décrit des objets du monde réel sous deux angles : ses **propriétés** (ses caractéristiques) et ses **méthodes** (les actions qu'elle peut effectuer ou son comportement).

Par exemple, la Classe Voiture représente une voiture, couleur est l'une de ses propriétés et accélérer/freiner sont deux de ses méthodes (c'est un peu comme des fonctions propres à cette classe).



Un autre exemple : on peut représenter en programmation orientée-objet les employés sous forme de classe ; auquel cas, la classe Employés représente tous les employés qui peuvent avoir pour propriétés un nom, un prénom, une adresse et une date de naissance ; les opérations qui peuvent être effectuées sur les employés peuvent être le changement de salaire, la prise de congé, la prise de retraite, etc.

La classe est finalement une sorte de moule, de modèle. Toutes les instances de classe s'appellent des **objets**. Les objets sont construits à partir de la classe, par un processus appelé **instanciation**. De ce fait, tout objet est une instance de classe.



L'instanciation

Ce que l'on appelle l'instanciation, pour être plus clair, c'est quand on crée un objet à partir de la classe.

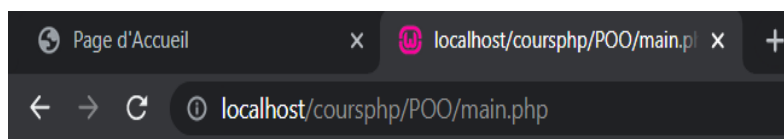
On va tout d'abord créer un dossier dans **coursphp**, que vous appellerez « **POO** » (qui je le rappelle signifie Programmation Orientée Objet). Dans ce dossier on va créer un fichier « **main.php** ». On va utiliser une Classe, qui existe déjà dans la librairie PHP, qui s'appelle

DateTime. On peut aller voir dans la documentation sur php.net. On a déjà vu ce type de date en MySQL. On va créer une première date et faire un `var_dump()` pour voir le contenu :

```
$date1 = new DateTime();  
var_dump($date1);
```

Ici on crée une variable `$date1` qui sera donc un objet qui va recevoir une instantiation de la Classe (`DateTime`) grâce au mot clé « **new** » qui en anglais veut dire nouveau. On fait par après un `var_dump()` de cette objet/variable `$date1` pour pouvoir voir ce qu'il y a dans cette variable/objet.

Lancez dans votre navigateur le chemin où se trouve le fichier `main.php` :



C:\wamp64\www\coursphp\POO\main.php:4:

```
object(DateTime)[1]  
  public 'date' => string '2022-03-23 10:32:45.427538' (length=26)  
  public 'timezone_type' => int 3  
  public 'timezone' => string 'UTC' (length=3)
```

Il a donc mis dans notre variable `$date1`, un objet de type `DateTime` avec la date et l'heure de quand j'ai fait une nouvelle instance.

Ça fonctionne un peu comme les tableaux, on avait vu qu'il y avait cette manière ci de déclarer un tableau (la syntaxe se ressemble mais ce n'est pas le même processus derrière) :

```
$tab1= array();  
$tab2= array(1,5,8);
```

Ici **\$tab1** reçoit un **nouveau tableau** qui est vide.

Et **\$tab2** reçoit un **nouveau tableau** avec 3 paramètres de type entiers.

```
$date1 = new DateTime();  
$date2 = new DateTime("12/02/1999");
```

Ici **\$date1** reçoit un **nouveau DateTime()**, sans paramètres donc prend la date d'aujourd'hui.

Et **\$date2** reçoit un **nouveau DateTime()** avec en paramètre une date (12/02/1999).

Si je fais un `var_dump()` pour chacune des variables.

```

localhost/coursphp/POO/main.php x +
localhost/coursphp/POO/main.php

C:\wamp64\www\coursphp\POO\main.php:7:
object(DateTime)[1]
  public 'date' => string '2022-03-24 07:49:08.938593' (Length=26)
  public 'timezone_type' => int 3
  public 'timezone' => string 'UTC' (Length=3)

C:\wamp64\www\coursphp\POO\main.php:8:
object(DateTime)[2]
  public 'date' => string '1999-12-02 00:00:00.000000' (Length=26)
  public 'timezone_type' => int 3
  public 'timezone' => string 'UTC' (Length=3)

C:\wamp64\www\coursphp\POO\main.php:9:
array (size=0)
  empty

C:\wamp64\www\coursphp\POO\main.php:10:
array (size=3)
  0 => int 1
  1 => int 5
  2 => int 8

```

Attention pour \$date2, c'est d'abord le mois puis le jour (12 c'est décembre).

Voici la différence : Les Classes sont la définition alors que l'objet c'est l'instanciation.

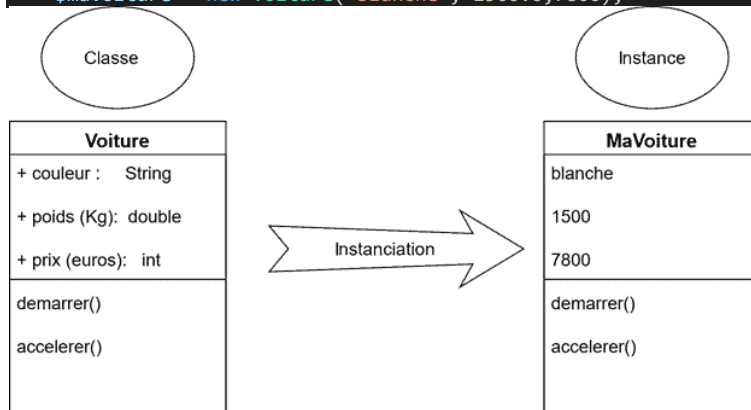
- **DateTime()** c'est le nom de la Classe qu'on utilise
- C'est grâce au new, qu'on crée une nouvelle instance de classe (objet)
- **\$date1** et **\$date2** c'est ce qu'on appelle des instances de classe ou des objets

Commenté [JD1]:

On peut reprendre le schéma avec l'exemple de la Classe Voiture et de l'instance.

En PHP ça donnerait quelques choses comme cela pour l'instanciation :

```
$maVoiture = new Voiture("blanche", 1500.0, 7800);
```

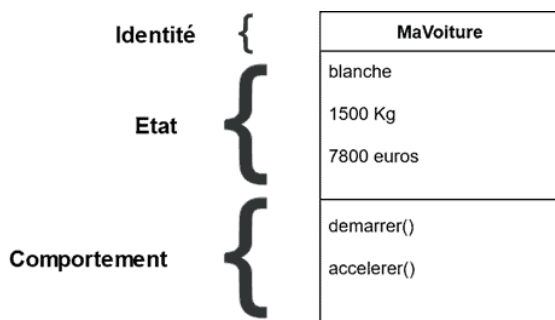


Le concept d'objet

Comme nous vous l'avons dit tout à l'heure, un objet est une instance de classe. Pour faire le parallèle avec le monde réel, l'objet c'est un peu comme une maison bâtit sur la base d'un plan particulier. Tant que les architectes se réfèrent à ce plan, ils produiront toujours les mêmes maisons.

Techniquement, un objet est caractérisé par 3 choses :

- **une identité** : l'identité doit permettre d'identifier sans ambiguïté l'objet (adresse/référence ou nom)
- **des états** : chaque objet a une valeur par défaut (lorsqu'elle est indiquée à l'instanciation) pour chacune de ses propriétés. On appelle ces valeurs, des états de l'objet.
- **des méthodes** : chaque objet est capable d'exécuter les actions ou le comportement défini dans la classe. Ces actions sont traduites en POO concrètement sous forme de **méthodes**. Les actions possibles sur un objet sont déclenchées par des appels de ces méthodes ou par des messages envoyés par d'autres objets.



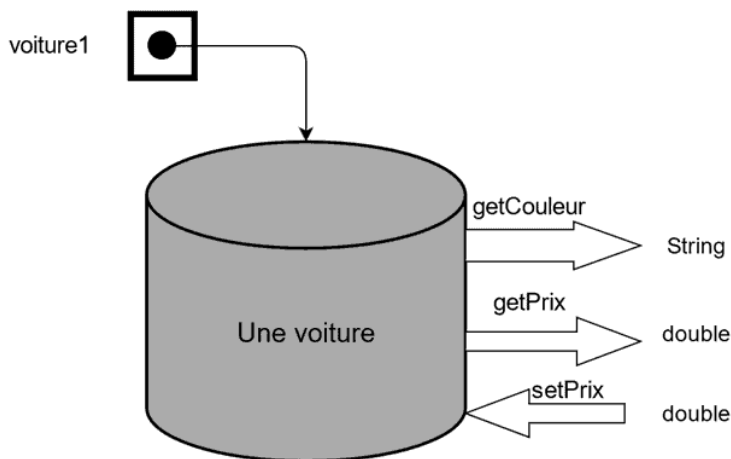
Le concept d'encapsulation

Les propriétés des objets ne peuvent être accédées que par ses méthodes. Ainsi, la classe encapsule à la fois les attributs et les méthodes qui permettent de manipuler les objets indépendamment de leurs états.

L'encapsulation permet de restreindre l'accès direct aux états et empêche la modification de l'objet hors de ses méthodes. Par exemple, si vous avez une classe *Voiture* et que vous voulez définir la valeur de sa propriété *couleur* à bleu, il vous faut passer par une méthode par exemple *definirCouleur*, implémentée par le développeur de la classe. Cette méthode peut restreindre les différentes valeurs de couleur.

Ainsi, l'encapsulation est un mécanisme qui empêche donc de modifier ou d'accéder aux objets par un autre moyen que les méthodes proposées, et de ce fait, permet de garantir l'intégrité des objets.

L'utilisateur d'une classe n'a pas forcément à savoir de quelle façon sont structurées les méthodes dans l'objet, ni le processus par lequel l'objet obtient tel ou tel état. En interdisant l'utilisateur de modifier directement les attributs, et en l'obligeant à utiliser les fonctions définies pour les modifier, on est capable de s'assurer de l'intégrité des objets.



L'encapsulation est comme un mécanisme de boîte noire qui empêche l'utilisateur d'utiliser un objet au-delà des méthodes qui lui sont proposées.

Dans le schéma précédent, la boîte noire masque les détails d'implémentation des attributs et des actions de la classe. Elle cache les attributs *couleur*, *poids*, *prix*. Le grand avantage de ce procédé est qu'en tant qu'utilisateur, on n'a plus à se préoccuper de comment est fait l'intérieur de l'objet de classe *Voiture*. On n'a plus besoin de se préoccuper sur le nombre d'attributs dans la classe *Voiture*. On se contente de connaître comment manipuler une voiture à l'aide des services offerts par la classe.

L'encapsulation permet de définir **des niveaux de visibilité** des éléments de la classe. Ces niveaux de visibilité définissent ce qu'on appelle **la portée** (ou encore **le périmètre**) de l'attribut/méthode. La portée est ainsi définie par méthode et par attribut et indique les droits à leur accès. Il existe trois niveaux de visibilité en php :

- **Public (+)**: les attributs publics sont accessibles à tous (**public**)
- **Protégée (#)**: les attributs protégés sont accessibles seulement dans la classe elle-même et aux classes dérivées(héritage). (**protected**)
- **Privée (-)**: les attributs privés sont accessibles seulement par la classe elle-même. (**private**)

On va un peu laisser la théorie et passer à la pratique. On va créer notre première classe en php. Vous comprendrez mieux lors de la pratique.

Création de notre Classe Voiture

On va créer notre première Classe. Toujours dans votre dossier « POO », créez cette fois ci un fichier que vous appellerez « **Voiture.php** », par convention la première lettre des classes c'est en majuscule.

La syntaxe pour une classe c'est :

```
class NomDeLaClasse{
    /*attributs et méthodes
    ...
    */
}
```

Créez la classe Voiture. Rien qu'avec ça on peut déjà instancier la classe Voiture pour avoir plusieurs objets de type Voiture.

On peut le tester, il suffit de retourner dans votre fichier « **main.php** » de rajouter tout en haut un « **require "Voiture.php";** » ensuite de créer une variable qui sera instancier :

```
$maVoiture = new Voiture();
var_dump($maVoiture);
```

On fait ensuite un var_dump() pour voir ce que cela donne dans notre navigateur.

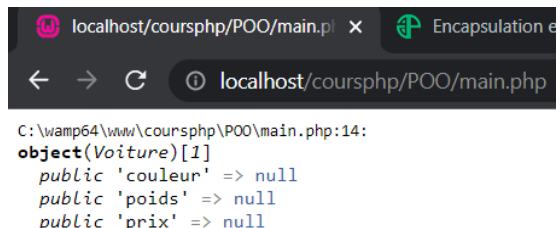
On va aussi ajouter dans notre classe les attributs/propriétés. Ce sont des variables qui vont caractériser notre objet :

- couleur (chaîne de caractères)
- poids(nombre décimal)
- prix(nombre entier)

Les attributs/propriétés, on peut les déclarer en publiques, protégées ou privées :

```
public $couleur;
public $poids;
public $prix;
```

Essayons de réexécuter notre main.php. On tombera sur ceci :



```
C:\wamp64\www\coursphp\POO\main.php:14:
object(Voiture)[1]
  public 'couleur' => null
  public 'poids' => null
  public 'prix' => null
```

Ils sont visibles et accessible par tout le monde. Si on veut les utiliser pour les afficher ou les modifier en dehors de la classe on pourra le faire grâce à leurs visibilité publiques.

(Exos)

- 1) **Modifier votre Classe Voiture en donnant cette fois ci le type adéquat des propriétés. Affichez l'objet dans le navigateur.**

Vous pouvez donner des valeurs par defaults à ces propriétés directement dans votre Classe.

- 2) **Donnez des valeurs à ces propriétés/attributs. Et affichez les dans le navigateur.**

En plus de ses propriétés, on avait vu qu'on pouvait aussi avoir des **méthodes** dans une classe. C'est-à-dire des fonctions propres à notre classe. Ici dans l'image concernant notre Classe voiture, on voit qu'il y a deux méthodes.

La syntaxe d'une méthode c'est simple :

```
public function nomDeMaMethode(): typeDeRetour{  
    //code  
}
```

On voit que c'est exactement comme ce qu'on a vu dans le chapitre des fonctions, la seule différence c'est qu'ici **il faut préciser la visibilité de la méthode**. Sachez d'avance qu'une méthode est généralement publique, par ce que c'est via les méthodes qu'on pourra accéder aux propriétés d'une classe.

- 3) **Ajoutez les deux méthodes qu'on voit dans notre image sur la classe voiture. Elles ne retourneront rien, elles afficheront chacune juste un message disant ce que fait la voiture.**

Maintenant que vous avez créé ces deux méthodes je veux y accéder dans le fichier principale « main.php ».

Pour appeler une méthode d'une classe voici la syntaxe :

```
$nomDeMonObjet->nomDeLaMethode();
```

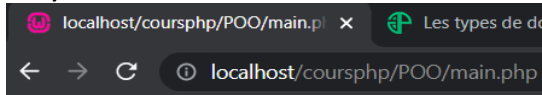
- 4) **Essayer d'appeler les deux méthodes que vous avez crée et de les afficher dans le navigateur. Je rappel que ces méthodes ne renvoi rien, elles font juste un affichage donc pas spécialement besoin de les mettre dans un var_dump().**

Pour récupérer une propriété il suffit d'utiliser la même syntaxe :

```
$nomDeMonObjet->nomDeLAttribut;
```

Après il suffira de l'afficher comme une variable normale.

5) Essayez de m'afficher ceci :

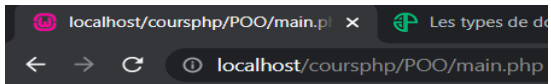


Ma voiture est de couleur : Blanc
Elle a un poids de : 510.5 Kg
Elle a un prix de : 3000 Euros

Il faut savoir qu'ici dans notre exemple on peut modifier les attributs en faisant juste ceci :

```
$maVoiture->couleur = "Noire";  
echo "Ma nouvelle couleur de voiture est : ". $maVoiture->couleur;
```

Ce qui donnera dans le navigateur



Ma voiture est de couleur : Blanc
Elle a un poids de : 510.5 Kg
Elle a un prix de : 3000 Euros
Ma nouvelle couleur de voiture est : Noire

Attention ce n'est pas du tout une bonne pratique. A partir de maintenant tous nos attributs/propriétés, on les déclarera en « **privée** ». Changez vos publics en private :

```
private string $couleur= "Blanche";  
private float $poids= 1500.0;  
private int $prix= 7800;
```

On pourra y accéder en-dehors de la classe en passant par des méthodes qui elles sont généralement « public ». Rappelez-vous, maintenant que nos attributs sont privés on ne peut plus les appeler dans « main.php » parce que la visibilité est privée, donc il n'arrive pas à accéder à ces propriétés/attributs. Dans un premier temps, on utilisera `var_dump()` pour afficher l'objet.

On va ensemble essayer de modifier la couleur par défauts de notre voiture en passant par une méthode que l'on va créer. Essayez de faire cela :

```
public function changerCouleur(String $nouvelleCouleur): void{  
    $couleur = $nouvelleCouleur;  
}
```

Ensuite afficher sur le navigateur en l'appelant :

```
$maVoiture->changerCouleur("Rouge");  
var_dump($maVoiture);
```

Que remarquez-vous ?

Vous verrez qu'il y a un problème, il affiche toujours votre couleur défini par default lors de la création de votre voiture. Il ne reconnait pas \$couleur ...mais rappelez-vous comment fonctionne les fonctions, il ne reconnait aucunes variables en dehors de la fonction. Les seules variables qu'il reconnait ce sont celles qu'on leur introduit en paramètre. Mais alors comment faire ?

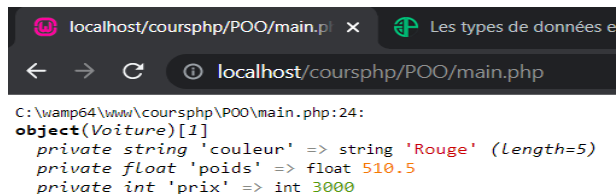
Dans notre Classe on va utiliser une variable particulière qui est « **\$this** », il fait référence à l'instance en cours. Donc si je veux la couleur de mon objet je fais :

```
$this->couleur
```

Et ainsi de suite pour les autres attributs.

Donc il faut modifier un peu notre méthode « changerCouleur » et modifier **\$couleur** en **\$this->couleur**.

Maintenant si vous réexécutez votre navigateur il affichera bien la nouvelle couleur :



```
C:\wamp64\www\coursphp\POO\main.php:24:
object(Voiture)[1]
  private string 'couleur' => string 'Rouge' (Length=5)
  private float 'poids' => float 510.5
  private int 'prix' => int 3000
```

Donc on a bien réussi à accéder à un attribut de classe qui est privé et on a même pu le modifier. Maintenant que vous savez comment modifier à votre avis comment peut-on afficher ?

- 6) Créez dans un premier temps 2 méthodes de modification pour les 2 autres attributs de classe. Et vérifiez que la modification se fait bien grâce au `var_dump()`.
- 7) Créez maintenant une méthode (`obtenirCouleur`) qui permet de récupérer la couleur et de pouvoir afficher la couleur à partir du `main.php`. Cette méthode ne reçoit rien en paramètre mais elle retourne le type de la propriété couleur. EN gros je veux accéder à couleur qui est privée en passant par la méthode. Afficher ensuite ceci :



```
Ma voiture est de couleur : Blanc
J'ai changé la couleur de ma voiture en : Rouge
```

- 8) Faites maintenant de même pour les deux autres attributs, créez 2 méthodes (`obtenirPoids` et `obtenirPrix`) ces deux méthodes retournent leur type.