PHP : OBJET

CLASSE EN PHP

Une classe est une représentation informatique d'un objet de la vie courante, comme une voiture, elle contient des attributs et des méthodes.

Mais quelle est l'intérêt ? On parle de **P**rogrammation **O**rienté **O**bjet ou **POO**.

Une classe est peu une « boîte à outils » permettant de créer des objets, on dit alors qu'un objet est une instance de classe.

C'est-à-dire que l'on créé des objets qui vont contenir les attributs et méthodes de notre classe, et **chaque instance a ses propres attributs et méthodes**, tel que déclaré dans la classe.

CRÉER UNE CLASSE

Par convention, lorsque l'on créé une classe on créé un fichier « **NomDeMaClasse.php** » en « PascalCase ».

Afin de créer une classe on utilise le mot-clé « class » avec le nom de celle-ci, il s'agit du nom du fichier sans le « .php » :

Nom de la classe

Attributs

Constructeur de la classe **(facultatif !)**

Méthodes (getter)

```php
class Marque
{
    private string $nom;

    public function __construct(string $nom)
    {
        $this->nom = $nom;
    }

    public function getNom(): string {
        return $this->nom;
    }

}
```

CRÉER UNE CLASSE

A partir de PHP 8.0, on peut écrire notre classe de cette manière :

Attributs

Nom de la classe

Constructeur de la classe

Méthodes (getter)

```php
class Marque
{

    public function __construct(private string $nom) { }

    public function getNom(): string {
        return $this->nom;
    }

}
```

CLASSE : LE $THIS

Le « **$this** » représente « **cette instance** » de la classe, on ne peut s'en servir qu'à l'intérieur d'une classe.

Ainsi le contenu de la méthode « **getNom()** », renvoie la valeur de l'attribut « **nom** » de la classe courante.

```php
public function getNom(): string {
    return $this->nom;
}
```

UTILISATION DES CLASSES

C'est à ce moment là que l'on va parler d'instanciation, pour cela on va utiliser le mot-clé « **new** » :

```php
$renault = new Marque( nom: 'renault');
```

Déclaration d'une variable

Mot clé « new »
Il représente le fait d'instancier d'une classe

Nom de la classe à instancier

Paramètre du constructeur de la classe

(PS : il est très important de comprendre que lorsque l'on créé une classe, on ajoute un nouveau type dans notre application)

UTILISATION DES CLASSES

Lorsque l'on instancie une classe, la variable devient un objet du type de la classe créée, autrement dit la variable « **$renault** » est du type « **Marque** ».

On peut maintenant utiliser les méthodes déclarées dans la classe Marque depuis la variable « **$renault** » :

```php
$name = $renault->getNom();
```

Une variable, qui est une instance de la classe en question

On utilise une flèche pour appeler les méthodes de la classe

La méthode appelée

UTILISATION DES CLASSES

On a déclaré un « **getter** » pour le nom, afin de récupérer le nom de la marque, si l'on souhaite la modifier en dehors, on doit faire un « **setter** ».

Tous les setter fonctionnent de la même manière : par convention on met toujours « **set** » devant suivi du nom de la propriété.

Le contenu de celle-ci doit venir affecter la valeur du nom au sein de la classe.

```php
class Marque
{

    public function __construct(private string $nom) { }

    public function getNom(): string {
        return $this->nom;
    }

    public function setNom(string $nom): void {
        $this->nom = $nom;
    }

}
```

LES TRAITS

Un trait est une « partial » d'attributs et méthodes à réutiliser dans les classes.
L'intérêt est de ne pas réécrire plusieurs fois ces attributs/méthodes.

On utilise un « Trait » dans une classe via le mot-clé « use » :

Ainsi, les attributs « id » et « name » existeront dans la classe « Model ».

On peut utiliser le mot clé « **self** », dans ces Traits, afin d'indiquer que la méthode renvoie un objet du type qui utilise le trait. (Ex : « fluent setter »)

```php
trait TraitName
{

    private int $id;

    private string $name;

    /**
     * @return int
     */
    public function getId(): int
```

```php
class Model
{

    use TraitName;

public function setId(int $id): self
{
    $this->id = $id;
    return $this;
}
```

HÉRITAGE

A quoi sert l'héritage ?
C'est de lier une classe enfant à une classe parente.

Le but est de donner des comportements et attributs de la classe mère aux classes enfants.
Ainsi, les attributs et méthodes de la classe mère seront hérités dans la classe fille, elle pourra donc y accéder même si ceux-ci ne sont pas déclaré au sein de sa classe.

On déclare une classe fille qui étends d'une classe mère avec le mot clés « **extends** », ici « **Piece** » est la classe mère et « **Rook** » la classe fille. On peut dire qu'une **« Rook » est une « Piece »**, c'est le même type.

```
class Rook extends Piece
{
```

Dans ce cas de figure, il est préférable que les attributs de la mère soient en « **protected** ».

HÉRITAGE : OVERRIDE

C'est quoi l'Override ?

Il s'agit de la réécriture de comportements (méthodes) de la classe mère dans la classe enfant.

Prenons ce cas de figure, j'ai une méthode « **toMove** » dans la classe mère.
Tant que celle-ci n'a pas été réécrite dans la classe fille, elle sera accessible depuis une instance de la classe fille et aura le comportement de la classe mère.

```php
public function toMove(): void {
    echo 'Move dans la classe mère';
}
```

Seulement, une méthode de la classe mère peut être réécrite dans la fille, et ainsi avoir son propre comportement, qui peut donc différer de celui de la mère.

```php
public function toMove(): void
{
    echo 'Move dans la classe fille';
}
```

HÉRITAGE : OVERRIDE (PARENT)

Même si l'enfant, peut vouloir réécrire un comportement de la mère, on peut cependant continuer d'appeler celui de la mère avec le mot clé : « **parent** » suivi de deux deux-points et le nom de la méthode dans la classe mère.

```php
public function toMove(): void
{
    echo 'Move dans la classe fille';
    parent::toMove();
}
```

J'appelle la méthode « **move** » depuis notre objet :

- L'enfant n'a pas réécrit la méthode, alors celle de la mère sera appelée

- L'enfant a réécrit la méthode, alors celle de l'enfant sera appelée

HÉRITAGE : ABSTRACT

« **abstract** » est un mot clé en programmation objet, qui peut se placer sur :

- Une classe, la classe deviendra « classe abstraite ». On ne pourra plus l'instancier (autrement dit, ne plus la créer directement avec un **new**).
On ne pourra donc l'instancier que via ses enfants.

```
abstract class Piece
{
```

- Une méthode, une méthode abstraite n'a pas de contenu, on ne fait que la déclaration de celle-ci et de ce qu'elle doit renvoyer.
Ainsi, les classes filles auront **OBLIGATION** de l'implémenter.

(cf, ci-joint dans la classe mère)

(cf, ci-joint dans la classe fille)

```
public abstract function move(): void;

public function move(): void
{
    // TODO: Implement move() method.
}
```