# La Programmation Orientée Objet

## 1. Qu'est-ce que c'est ?

La programmation orientée objet, ou POO, est une façon d'organiser son code en **classes** afin de pouvoir le faire fonctionner avec des **instances** de ces **classes**, que l'on appelle des **objets**. Ces objets possèdent des **propriétés** (ou attributs), et des **méthodes** (fonctions), que l'on peut appeler depuis un objet.

La POO est entièrement dépendante de la notion de **classe**, alors voyons en quoi cela consiste.

## 2. Par où commencer ?

Une **classe** est un fichier qui permet de décrire l'ensemble des **propriétés** et **méthodes** de notre ou de nos futur(s) **objet(s)**.

Sa structure se présente ainsi :

```php
// Déclaration de la classe
class Personnage {
    // Liste des propriétés de la classe
    // Les propriétés ont deux informations :
    // Leur visibilité : Est-ce que j'y ai accès depuis une instance de mon objet ou non ?
    // Leur type : je ne pourrais pas y mettre de valeur d'un autre type
    private string $nom;
    private int $atk;
    private int $def;
    private int $magie;
    private bool $isArcher;
    // Fin de la liste
    // Liste des méthodes

    // Constructeur
    // Il permet d'assigner des valeurs aux propriétés de mon objet
    // Dès que je crée une nouvelle instance
    // Il est appelé par défaut sans avoir besoin de préciser son nom
    // et se note toujours __construct
    public function __construct($monNom, $monAtk, $maDef, $maMagie, $monPersonnageEstUnArcher = false) {
        $this->nom = $monNom;
        $this->atk = $monAtk;
        $this->def = $maDef;
        $this->magie = $maMagie;
        $this->isArcher = $monPersonnageEstUnArcher;
    }
```

```php
    // Ici on a une méthode publique, donc que l'on peut appeler directement
    // Sur l'instance de notre objet. Cette méthode affiche une fiche de stats
    // Correspondant aux valeurs assignées à notre personnage.
    public function afficherFichePersonnage() {

        $string = "Nom du personnage : " . $this->nom . "<br>";
        $string .= "Attaque : " . $this->atk . "<br>";
        $string .= "Défense : " . $this->def . "<br>";
        $string .= "Magie : " . $this->magie . "<br>";
        $ouiOuNon = ($this->isArcher) ? "Oui" : "Non";
        $string .= "Sait manier un arc : " . $ouiOuNon . "<br>";

        return $string;
    }
    // Fin de la liste des méthodes
}
// Fin de la classe
```

Une fois notre classe décrite, on va pouvoir **créer des objets à partir d'une classe**. C'est ça, la programmation orientée objet : on crée des objets à partir d'une ou de plusieurs classes pour les faire interagir entre eux.

Pour créer ces instances, on va utiliser la syntaxe **new** suivie du nom de la classe :

```php
// Instanciation de mes objets. J'en ai 3
$monPerso = new Personnage("Gandalf", 100, 1, 500);
$monPerso2 = new Personnage("Bilbo", 10, 100, 0);
$monPerso3 = new Personnage("Legolas", 80, 30, 50, true);
```

Les valeurs que je passe entre parenthèses correspondent aux paramètres qu'attend le **constructeur**, et elles seront assignées aux **propriétés** de mon objet.

Enfin, pour faire appel aux **méthodes** ou aux **propriétés** de mon objet, je peux utiliser l'opérateur **->**, comme ici :

Attention cependant, je n'ai accès de cette manière qu'aux propriétés / méthodes **publiques** de mon objet.

```php
// J'appelle ma méthode publique afficherFichePersonnage() sur chaque instance
// de ma classe.
echo $monPerso->afficherFichePersonnage();
echo "<br>";
echo $monPerso2->afficherFichePersonnage();
echo "<br>";
echo $monPerso3->afficherFichePersonnage();
```

## 3. Applications

Lors des exemples au-dessus, nous avons créé plusieurs personnages avec des propriétés définies. Ici, ce sont des personnages avec des **statistiques**, semblable à un jeu de rôles, mais dans la vraie vie nous allons surtout nous servir de ces classes pour créer des objets correspondants à des **entités**, c'est-à-dire des tables en base de données.

Chaque **propriété** de la classe correspondra à un **champ** dans la table, et chaque **objet** sera en réalité un **enregistrement** de la classe.

Attention cependant, si des rapprochements peuvent être faits entre la P.O.O. et la base de données, les données que nous définissons dans nos objets ne sont pour l'instant sauvegardées nulle part ! Elles ne sont pas **persistantes**.

## 4. Getters et Setters

C'est comme ça que l'on appelle l'ensemble des **méthodes** de la classe qui vont nous permettre d'accéder ou de définir ses propriétés. Une propriété n'a pas besoin de getter ou de setter si elle est publique, mais pour des raisons de sécurité on préférera les mettre en privée par défaut et définir des méthodes d'accès comme des getters et des setters.

Exemple pour une classe qui aurait les propriétés nom, effet, et degats :

```php
// Getter pour le nom
    public function getNom() {
        return $this->nom;
    }
    // Setter pour le nom
    public function setNom($value) {
        $this->nom = $value;
    }
    // Getter pour l'effet
    public function getEffet() {
        return $this->effet;
    }
    // Setter pour l'effet
    public function setEffet($value) {
        $this->effet = $value;
    }
    // Getter pour les dégats
    public function getDegats() {
        return $this->degats;
    }
    // Setter pour les dégats
    public function setDegats($value) {
        $this->degats = $value;
    }
```

On n'est absolument pas obligé d'avoir des getters et des setters pour chaque propriété. Il est possible qu'une propriété doive pouvoir être lue, mais pas définie. Dans ce cas elle n'aurait qu'un getter. Et si une donnée doit pouvoir être définie mais pas visible du tout depuis l'extérieur, alors elle aura un setter mais pas de getter.

## 5. Visibilité des propriétés

Mettre des propriétés de notre classe en privé ou en public n'est pas sans conséquence. En effet, si les propriétés sont toutes **publiques**, alors elles pourront être appelées depuis n'importe où sur mon objet avec la syntaxe **->**, comme ceci :

```php
$sorcier1 = new Sorcier(100, "Harry Potter", 1);
echo $sorcier1->nom;
```

Cela peut poser des problèmes de sécurité, ou non selon le besoin. Si l'on souhaite s'assurer que les propriétés de notre objet ne sont modifiées que lorsque nous en faisons la demande, on va les passer en visibilité **privée**. Cela veut dire que depuis l'extérieur, je n'aurais accès à la valeur de ces propriétés qu'en appelant le **getter** correspondant, qui lui doit être **public**, comme ceci :

```php
$sorcier1 = new Sorcier(100, "Harry Potter", 1);
echo $sorcier1->getNom();
```

Et surtout cela veut dire que si je décide de ne **pas** créer de getter correspondant à cette propriété, alors elle ne sera pas du tout accessible en dehors de la classe. Ça peut être un choix à faire.

À noter cependant que dans la classe, on a toujours accès aux propriétés de notre objet en utilisant la syntaxe **->** et surtout l'objet **$this**, comme ceci :

```php
echo $this->nom;
```

Il existe enfin une troisième visibilité que l'on peut choisir, c'est la visibilité **protected**. Les propriétés que l'on a défini comme **protected** ne seront pas accessibles ailleurs que dans la classe en question, en utilisant **$this**. Si on veut pouvoir les définir, on peut toujours créer un **setter**, mais un getter ne fonctionnera pas car la donnée ne **doit pas** être lue en dehors de la classe.

Exemple : une connexion à la BDD.

```php
protected $db;
```

## 6. Processus de création d'une classe

Avec ce que nous venons de voir, on peut définir une méthodologie assez efficace pour être sûr de ne rien oublier. Nous allons effectuer ces actions dans l'ordre :

1- Rédiger la liste des **propriétés** de la classe. Je dois savoir de quelles informations la classe va être le réceptacle, quel va être leur **type**, ainsi que leur **visibilité**.
2- Rédiger le **constructeur** de la Classe. Il doit permettre de bâtir un objet en assignant les valeurs passées avec la syntaxe **new MaClasse($propriete1, $propriete2)** aux propriétés de la classe.
3- Rédiger les **getters** et les **setters**, si besoin. À vous de trancher si vous allez avoir besoin d'accéder à tout, ou non.
4- **Facultatif** : Rédiger d'autres méthodes, selon le besoin. Peut-être que vous n'aurez rien d'autre à écrire, mais quoi qu'il en soit il faut le faire en dernier.
5- **Facultatif** : Rédiger une méthode **__toString()** . Cette méthode sera appelée par défaut lorsque vous traiterez votre objet comme une chaîne de caractère, par exemple en en faisant simplement un **echo**. À vous de prévoir ce qui doit s'afficher dans ce cas-là, par exemple dans le cas d'une classe Personne, on peut vouloir afficher son nom et prénom…