https://www.bgmp.fr

LA PROGRAMMATION ORIENTÉE OBJET EN PHP

La *programmation orientée objet* (POO) en PHP est un des cœurs principaux de ce langage. J'ai déjà évoqué *ce sujet* lors de mon approche de Python. La logique va donc être la même, seul le langage change !

Voyons plus en détail comment PHP utilise la POO.

LES CLASSES

Une classe est une représentation de quelque chose (donnée, container, etc.), c'est un « objet commun ». On peut lui définir un patron de représentation. Et c'est avec ce patron que l'on pourra ensuite générer des objets.

Pour définir une classe, on prépare des attributs (= variables). On peut aussi lui ajouter des méthodes (= fonctions) qui donnent des fonctionnalités par défaut.
On peut ensuite créer une nouvelle instance de la classe (= objet).
À cet objet, on peut ainsi définir des valeurs aux attributs (on utilise la flèche pour l'attribution des valeurs et on ne met pas de $ lors de l'appel de l'attribut).

Dans une classe, pour accéder aux attributs eux-mêmes, on utilise $this. Lors de l'utilisation de la classe, $this sera remplacé alors par le nom de l'objet.
Attention : $this réfère toujours et uniquement à la classe elle-même !

```php
class Table {
    /**
    * Attributs
    */
    public $nom;
    public $plateau;
    public $pieds;

    /**
    * Méthodes
    */
    public function fold() {
        echo "la table " . $this->nom . " est pliée.";
    }
}

$tableKlak = new Table;
$tableKlak->nom = "Klak";
$tableKlak->plateau = "marbre";
$tableKlak->pieds = "acier";

echo "La table " . $tableKlak->nom . " a des pieds en " . $tableKlak->pieds . " et un plateau en " . $tableKlak->plateau
// affichera : La table Klak a des pieds en acier et un plateau en marbre

$tableKlak->fold();
// affichera : La table Klak est pliée.
```

LES OBJETS

On peut voir les objets comme étant des array surpuissants, car on peut leur affecter des attributs pour être sûrs de ce qu'il y a dedans.
Il y a cependant une grosse différence entre les objets et les arrays : dans un array, on peut rajouter à la volée une nouvelle clé et sa valeur, ou ne pas remplir une clé déjà existante. Avec un objet, on est obligé de prévoir en avance les données qui vont exister, et on sera obligé de ne remplir que celles-ci, et pas une de plus.

Il est aussi possible de mettre des valeurs par défaut aux attributs :

```php
class Table {
    public $nom;
    public $plateau = "plastique";
    public $pieds = 'plastique';
}
```

On peut définir des attributs 'obligatoires'. Et pour cela on, va utiliser des « méthodes magiques ». La plus importante est le constructeur car elle est systématiquement appelée lors de la création de l'objet. Cela permet de mettre une surcouche de protection (pour être sûr d'avoir les données nécessaires).

```php
class Table {
    public $nom;
    public $plateau = "plastique";
    public $pieds = 'plastique';

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

$tableKlak = newTable("Klak");
```

À ce moment il n'y a plus besoin d'instancier l'attribut nom car il le fait automatiquement lors de la création de l'objet.

De manière générale, on ne crée pas une nouvelle variable. On se sert de celles existantes. Dans l'exemple précédent, on écrirait le constructeur de la façon suivante :

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

LES MÉTHODES

Les méthodes sont des fonctions (qui existent dans une classe). Pour créer une méthode, on la déclare dans la classe avec public function :

```php
class Ship {
    public $name;

    public function sayHello() {
        echo "Hello !";
    }
}
```

Pour appeler une méthode, on utilise la même flèche -> que pour les attributs :

```php
$myShip->sayHello();
```

Les méthodes permettent donc d'avoir des classes qui sont des petits packs de données, comme un array, mais qui peuvent aussi effectuer des actions, ce qu'un array ne peut absolument pas faire !

Et comme pour une fonction classique, on peut aussi ajouter des arguments à nos méthodes de classe.

Attention, il ne faut pas oublier qu'une méthode doit absolument retourner quelque chose (donc ne pas oublier de mettre un return ou echo) !

LA DOCUMENTATION

Lorsque l'on code en programmation orientée objet en PHP, il est de convention chez tous les éditeurs de documenter son code avec phpdoc. Il permet lors de l'autocomplétion dans VSCode (ou tout type d'IDE) d'avoir des informations sur les fonctions, les classes, les variables, etc.

En PHP, on utilise un bloc de commentaires (/** */) avant la déclaration de la class. Par exemple pour un constructeur :

```php
/**
* @param string $name Nom du personnage
* @param integer $power Puissance du personnage
*/
```

Il suffit donc de déclarer : @param (annotation paramètre) suivi du type de données, du nom du paramètre, et éventuellement de ce qu'il représente.

Attention : phpdoc est uniquement de la déclaration et n'a aucune valeur de vérifier les bugs. C'est une aide pour les utilisateurs du code (voir même pour permettre de générer automatiquement une doc)

Pour les attributs, on utilise @var suivi du type de variable (integer, string, boolean).

GESTION DES CLASSES

Pour limiter les bugs, l'accès aux données des classes et objets doit être contrôlé. Dans un premier temps, la déclaration des attributs ne sera plus public mais private :

```php
class Table {
    private $nom;
    private $plateau = "plastique";
    private $pieds = 'plastique';
}
```

On ne peut alors plus accéder à l'attribut ->nom car il est privé (private). Cela permet donc de ne plus avoir d'erreurs de saisie dans nos objets, par contre on ne peut plus rien mettre dedans du tout.

Pour pallier ce problème, on va utiliser des getter et des setter. Mais qu'est-ce que c'est ?

GETTER

Un attribut private, n'est accessible ni en lecture, ni en écriture.
Nous allons donc utiliser des getters, des méthodes qui nous permettent d'accéder aux données de ces attributs.

L'intérêt est d'avoir plus de possibilités, pour formater les données ou pour traduire ce qui vient de la base de données, etc.
Il transforme donc la donnée pour lui permettre de s'afficher.

Par exemple, on peut modifier la casse d'un attribut lors de son affichage :

```php
class Table {
    // ...
    public function getNameUppercase() {
        return strtoupper($this->nom);
    }
}

$maTable = new Table();
echo "TABLE : " . $maTable->getNameUppercase();

// Le nom de la nouvelle table sera écrit en majuscule
```

SETTER

Ce sont des méthodes qui ne servent qu'à une chose : changer la valeur d'un des attributs de notre classe. Le setter prépare donc la donnée.

En fait, quand on met un attribut sur private, on n'y a plus accès depuis l'extérieur de la classe. En revanche, il reste accessible à l'intérieur de la classe ! Un setter permet alors de retourner l'objet lui-même avec `return $this ;`

```php
class Table {
    private $nom;
    private $plateau = "plastique";
    private $pieds = 'plastique';

    public function setName($nom) {
        $this->nom = $nom;
    }
}
```

Le gros avantage des setters, c'est qu'il y a des validations dans nos données. Dans les méthodes set***(), on peut valider les données avant de réellement les donner à nos objets.

TYPAGE

Pour avoir une levée d'erreur, il faut bien penser à typer les paramètres. Par exemple pour le constructeur :

```php
public function __construct(string $newName, int $newStrength = null, int $newPower = null)
{
    // instructions
}
```

Et pour typer une valeur retour dans une fonction, on l'ajoute au moment de la déclaration :

```php
function maFonction () : int
{
    // instructions
}
```

Nous avons vu dans les grandes lignes la programmation orientée objet en PHP.
Il nous reste à voir comment bien organiser notre code grâce à la MVC… Ça sera donc le thème du prochain post !!!