Table des matières

1. Les classes

- Une **classe** représente un objet, c'est-à-dire une représentation en langage de programmation d'un objet
- Une classe permet d'enrichir votre application et ainsi ajouter un nouveau **type** utilisable (en plus des types de bases, int, string etc)
- On peut voir une classe comme un tableau associatif
- Par exemple, une voiture peut se représenter par :
    - Sa couleur, string
    - Sa marque, string
    - Son modèle, string
    - Ses chevaux, int
    - Etc…
- Ces informations sont appelées des **attributs** et décrivent votre objet
- Une classe a besoin d'un constructeur pour l'instancier, c'est-à-dire permettre la création d'objet de ce type.
- Une classe peut aussi voir des **fonctions** (ou **méthodes**), elles représentent souvent un comportement de la classe, de notre objet. Par exemple, on peut imaginer qu'une voiture ai une fonction « rouler() », qui représente le fait qu'elle avance.
- Les **attributs**, et les **fonctions** d'une classe ont ce que l'on appelle un indice de « **visibilité** », c'est-à-dire un mot clé pour indiquer qui à le droit d'accéder à l'attribut ou la méthode.
- Il existe 3 indices de visibilité :
    - Public : accessible depuis partout, aussi bien dans la classe courante que dans les autres
    - Protected : accessible uniquement par la classe courante et ses éventuelles classes filles (**héritage**)
    - Private : accessible uniquement par la classe courante
- Il est recommandé que les attributs d'une classe soient toujours « **private** » ou « **protected** » dans le cas d'une classe mère (héritage). Car c'est le rôle de la classe de modifier ses attributs.
- Ainsi, pour modifier les attributs d'une classe, on utilise ce que l'on appelle des « **getter** » et « **setter** », ils ont là pour permettre de modifier (setter) et récupérer (getter) les attributs de notre objet.
- Au sein d'une classe, on utilise le mot clé « **$this** » pour accéder aux fonctions ou attributs de celle-ci.
    - Ce mot clé représente « cette classe », « cet objet »

1.1 Exemple de classe

```
class Voiture {

	private string $model;

	private string $brand ;

	private string $color ;

	private int $power ;

	/*
	 * Constructeur de la voiture
	 * @param string $color, la couleur de la voiture
	 */
	public function __construct(string $color) {
		// On utilise le mot clé $this pour accéder à l'attribut color de la voiture
		// Attention à ne pas confondre : $color représente le paramètre du
		// constructeur, et $this->color l'attribut de la voiture !
		$this->color = $color ;
	}

	/*
	 * Getter de l'attribut couleur
	 * @return l'attribut couleur (de type string)
	 */
	public getColor() : string {
		// On utilise le mot clé $this pour accéder à l'attribut color de la voiture
		return $this->color ;
	}

	/*
	 * Setter de l'attribut couleur
	 * @param $color, la couleur à set à l'attribut
	 * @return l'attribut couleur (de type string)
	 */
	public setColor(string $color) : void {
		// On utilise le mot clé $this pour accéder à l'attribut color de la voiture
		// Attention à ne pas confondre : $color représente le paramètre du
		// constructeur, et $this->color l'attribut de la voiture !
		$this->color = $color ;
	}

}
```

1.2 Utilisation d'une classe

- On instancie un objet, c'est-à-dire que l'on créé une variable qui sera du type de notre objet, via le mot clé « **new** »
- Une fois l'objet créé on peut accéder aux attributs et méthode de notre objet, comme on le faisait avec le $this : $this->nomFonction

```
// Créer un objet de type voiture que l'on assigne à la variable voiture
$voiture = new Voiture('bleue') ;
// Affiche le retour de la fonction getColor de notre objet
echo $voiture->getColor() ;
// Appelle la fonction setBrand de la classe
$voiture->setBrand('Renault') ;
```

1.3 Mot clé « static »

- Suit les mêmes règles de visibilité habituelles
- **Représente un attribut ou méthode commune à toutes les instances de la classe**
- On accède aux attributs static via le mot clé « **self** », en dehors de la classe on y accède via : LeNomDeLaClasse ::NomAttributMethodeStatic
- Exemple :

```
class Voiture {

	public static int $nbRoues = 4 ;

	public static function getNbRoues() : int {
		echo self ::$type ;
	}

}
```

2. Héritage

- On utilise le mot clé « **extends** » pour dire qu'une classe étend une autre, c'est-à-dire qu'une classe devient du même type que l'autre classe. On parle de classe mère et classe fille.

```
class A { }

class B extends A { }
```

Ici on dit que B est A, B étant de la classe A.

- En héritage, les attributs ou fonctions sont communs aux classes filles, on recommandera l'usage du mot clé « **protected** » pour la visibilité
- On peut effectuer de la redéfinition de constructeur ou de fonction, dans une classe fille. C'est-à-dire que ma classe fille peut définir un autre comportement, pour une même fonction.

Prenons en exemple, le contructeur d'une pièce de jeu d'échec (classe mère) :

```php
public function __construct(
    string $color, string $name, string $pathImage
) {
    $this->color = $color;
    $this->name = $name;
    $this->pathImage = $pathImage;
}
```

Il peut se redéfinir dans la classe fille de cette manière :

```php
public function __construct(string $color)
{
    parent::__construct($color, 'Pion', 'pion.png');
}
```

On utilise le mot clé « **parent :: nomFonction** » dans la classe fille

- On peut aussi définir des fonctions dans la classe mère, qui seront donc utilisables par les classes filles. (ici code de la classe mère)

```php
public function echoName(): void {
    echo 'Bonjour de la classe mère';
}
```

Cependant il est autorisé aux classes filles de pouvoir les redéfinir, en gardant ou en surchargeant le comportement de la classe mère.
Dans une classe fille, on effectue ce code :

```php
public function echoName(): void
{
    echo 'Je suis un Pawn, classe fille de ChessPiece';
}
```

Lorsque j'appellerai la fonction echoName sur ma classe, fille, elle appellera sa propre fonction redéfinit, et non « Bonjour de la classe mère ».

- Le mot clé « **abstract** » représente une classe abstraite, autrement dit une classe qui ne s'instancie pas.
- On peut aussi utiliser le mot clé « **abstract** » sur une fonction, alors elle n'aura aucun traitement et on se contentera de lui donner un nom, des potentiels arguments et un type de retour.

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

L'intérêt est d'obliger les classes filles à lui définir un comportement (lui ajouter un traitement).