La programmation objet en PHP

Introduction

- La programmation orientée objet est une façon différente de coder qui va suivre des règles différentes de celles que vous connaissez dans la programmation procédurale.
- La programmation orientée objet c'est tout simplement faire de son site un ensemble d'objets qui interagissent entre eux. En d'autres termes : tout est objet.
- La logique de la programmation orientée objet est presque la même dans tout les langages de programmation, donc cela va ressembler à la POO en python avec des différence de syntaxe.
- La POO est basée sur la notion **d'objet** et **classe**.

C'est quoi un objet?

- Notre entourage est pleine d'objets, l'ordinateur est un objet, un stylo est un objet, un téléphone, ...etc
- Un objet on peut le définir comme étant un élément qui possède un ensemble de caractéristiques et des fonctionnalités.
- Par exemple, un stylo a des caractéristiques comme; la **forme**, la **couleur**, la **taille**,...et possède également une fonctionnalité, qui est **ECRIRE**

C'est quoi un objet?

- Les caractéristiques d'un objet, on les appelle en PHP les **attributs**

- Les fonctionnalités d'un objets on les appelles les **méthodes**.

C'est quoi une classe?

- Une classe c'est la définition de l'objet qu'on va utiliser.
- À partir d'une classe, on peut créer une infinité d'objets.
- Par exemple, on peut se servir d'un **moule** pour créer des gâteaux. Le **moule** est **unique**, par contre on peut créer une **infinités** de **gâteaux** à partir de ce moule.(le moule c'est l'équivalent de classe et les gâteaux sont des équivalents d'objets)

Comment créer une classe ?

- En PHP, on crée une nouvelle classe avec le mot clef **class**.
- On placera généralement chaque nouvelle classe créée dans un fichier à part qu'on nomme généralement de cette manière: **NomDeClasse.class.php** par exemple **Voiture.class.php**

```
class Voiture{
  // Déclaration des attributs et méthodes ici.
}
```

- Une classe peut regrouper plusieurs éléments:
  - Des attributs
  - Des méthodes (entre autres accesseurs et mutateurs)
  - Un constructeur

Déclaration des attributs

- Un attribut est une variable qui va stocker une information concernant une caractéristique d'un objet.

- Puisqu'en PHP on ne déclare pas le type d'une variable, donc pour déclarer un attribut, il faut le nom précédé par sa **visibilité (les modificateur d'accès)**.

```
class Voiture
{
  private $_matricule ;
  private $_modele ;
  private $_marque ;
  private $_prix ;
}
```

Visibilité d'un attribut (ou méthode)

- La visibilité d'un attribut ou d'une méthode indique à partir d'où on peut y avoir accès. On utilise pour cela des mots clés qu'on appelle les modificateurs d'accès.

  1. **Public**: le mot clef public signifie ici qu'on va pouvoir accéder à notre attribut depuis **l'intérieur et l'extérieur de notre classe.**
  2. **Private**: le mot clef private signifie que notre attribut est accessibles uniquement à l'intérieur de la classe. C'est-à-dire qu'il n'est plus accessible de l'extérieur de la classe.

- **Rq**:
  - La visibilité est **par défaut public** en PHP.

Principe d'encapsulation

- Le principe de l'encapsulation consiste à masquer le code écrit de manière qu'il ne soit pas visible à l'utilisateur.
- Le concepteur de la classe a englobé dans celle-ci un code qui peut être assez complexe et il est donc inutile voire dangereux de laisser l'utilisateur manipuler ces objets sans aucune restriction. Ainsi, il est important d'interdire à l'utilisateur de modifier directement les attributs d'un objet c'est pour cela on les mets **private**.

Création des méthodes

- Une méthode représente une fonctionnalité d'un objet.
- Une méthode est une fonction donc elle est introduite par le mot **function**
- La visibilité des méthodes est souvent **public.**

```
public  function Rouler()
{
   echo "Je suis une voiture et je roule !";
 }
```

Les accesseurs (getters)

- Un accesseur est une méthode qui **retourne la valeur d'un attribut.**
- On peut créer autant d'accesseurs que d'attributs.
- Par convention, le nom de l'accesseur commence par **Get**

```
Public function  GetMatricule(){

return  $this ->_matricule;
}
```

- Le mot clef **$this** sert à faire référence à **l'objet couramment utilisé**.
- L'opérateur -> sert à accéder à un attribut ou méthode depuis notre objet (équivalent de point en javascript)
- **Remarque**: l'attribut matricule est sans l'opérateur $. (_matricule)

Les mutateurs (setters)

- Un mutateur est une méthode qui permet d'affecter une valeur à un attribut c'est-à-dire permet de **modifier la valeur d'un attribut.**
- Par convention le nom d'un mutateur est précédé par le mot **Set**

```
Public function  SetMatricule ($maValeur){

 $this ->_matricule=$maValeur;
}
```

Créer un objet

- Pour créer un nouvel objet, on doit faire précéder le nom de la classe par le mot-clé **new**, comme ceci :

```
$MaVoiture= new Voiture();
```

- La création d'un nouveau objet à partir d'une classe s'appelle **l'instanciation**.
- Un objet est une **instance** de classe.
- Le mot **new** est utilisé pour une nouvelle instance d'une classe.

Appeler une méthode d'un objet

- Pour appeler une méthode d'un objet, il va falloir utiliser un opérateur ->

```
class Voiture
{
  private $_matricule;
  private $_modele;
  private $_marque;
  private $_prix;
 public   function Rouler()
 {
    echo "Je suis une voiture et je roule !";
  }
....
}
$MaVoiture= new Voiture();

$MaVoiture -> Rouler();
```

Appeler une classe dans un autre fichier

- Généralement la classe est définie dans un fichier à part, et les objets se trouvent dans l'autres fichiers.
- Question: si la classe et l'objet se trouvent dans des fichiers distincts, comment est ce que la classe est reconnue?
  - Dans ce cas il faudra **appeler le fichier** qui **contient** la définition de la **classe** à l'intérieur du fichier où est définit l'objet. On va utiliser pour ce faire le mot **include** ou **require**

```
require "Voitures.class.php";
Ou
include "Voitures.class.php";
```

Appeler une classe dans un autre fichier

Voiture.class.php

```php
class Voiture
{
  private $_matricule;
  private $_modle;
  private $_marque;
  private $_prix;

 public   function Rouler()
 {
    echo "Je suis une voiture et je roule !";
  }

}
```

Programme.php

```php
include "Voiture.class.php";

$MaVoiture= new Voiture();

$MaVoiture -> SetMatricule("AB564243");
echo $MaVoiture -> GetMatricule();
$MaVoiture -> Rouler();
```

Le constructeur

- Un constructeur est une méthode spéciale qui sert à ***construire*** un objet.

- Le rôle d'un constructeur est **d'initialiser** les attributs d'un objet dès sa création.

- Pour créer un constructeur on écrit: **__construct**
- ***Syntaxe***:

```
public function __construct(les parametres){
//des initialisations
}
```

La méthode constructeur

- Exemple:

```
public function  __construct ($m , $mdl , $mar , $pr){
$this ->_matricule =$m;
$this ->_modele=$mdl;
$this ->_marque=$mar;
$this ->_prix=$pr;
}
```

- **Remarques**:
  - si aucun constructeur n'a été définit dans la classe php utilise le constructeur par défaut (sans paramètres)
  - En php, un seul constructeur peut être définit dans une classe.
  - Si un constructeur est définit dans une classe, le constructeur par défaut n'est plus utilisable.

Le destructeur

- Le destructeur et une méthode publique identifiée par le nom **__destruct()**. Il est appelé automatiquement par le compilateur lorsqu'il n'y a plus aucune référence à l'objet en cours. Autrement dit, le destructeur est appelé quand il n'y a plus aucun appel d'un membre quelconque de l'objet.

```
public function  __destruct (){
                echo "destruction de la voiture ". $this->_matricule ;
}
```

La méthode __tostring()

- La méthode magique __toString() va être appelée dès que l'on va traiter un objet comme une chaine de caractères (par exemple lorsqu'on tente d'echo un objet).
- Attention, cette méthode doit obligatoirement retourner une chaine ou une erreur sera levée par le PHP.

```
public function  __toString(){
        return 'Matricule de la voiture: ' . $this->_Matricule. '<br>
        Le prix est: ' . $this->_prix. '<br><br>';
    }
```

Les attributs statiques

- Un attribut statique est un **attribut qui appartient à la classe** et non pas à l'objet.
- Ainsi, **tous les objets auront accès** à cet attribut et **cet attribut aura la même valeur pour tous les objets.**
- Pour définir un attribut statique on doit ajouter le mot **static** après la visibilité de l'attribut.

```
private static $NomAttribut ;
```

Méthodes statiques

- Une méthode **statique** est une méthode qui ne va pas appartenir à une instance de classe ou à un objet mais **qui va appartenir à la classe** dans laquelle elle a été définie.
- On va pouvoir définir une méthode statique à l'aide du mot clef **static**.

```
public static function MaMethode()
 {
   //code
 }
```

Accéder à une variable statique à l'intérieur de la classe

- Pour accéder à une **variable statique à l'intérieur d'une méthode de la classe**, on ne va pas utiliser le mot **$this**, celui-ci est réservé au objets, nous allons utiliser le mot **self** suivi de l'opérateur :: de la manière suivante:

```
self :: nomVariable Statique ;
```

- self fait référence à la classe

Accéder à une méthode statique

- Pour accéder à une méthode statique, **en dehors de la classe,** on dois le faire à partir de la classe et non pas à partir d'un objet. L'accès se fait avec l'opérateur ::

- ***Syntaxe:***

```
NomClasse :: methodeStatique()
```