Introduction à la programmation orientée objet

Programmation Orientée Object (POO) en PHP

Chris Chevalier – chevalier@chris-freelance.com

Les concepts de programmation orientée objet

Introduction à la programmation orientée objet

Chris Chevalier – chevalier@chris-freelance.com

La programmation procédurale en quelques mots…

- Jusqu'ici, vous avez toujours représenté votre code source de façon procédurale
- Elle consiste à séparer les données de leurs traitements
- Il existe néanmoins une alternative à cette pratique - Il s'agit de la programmation orientée objet
- Il ne s'agit pas nécessairement d'une meilleure pratique

La programmation orientée objet (POO) en quelques mots…

- La programmation orientée objet consiste à créer son programme de façon à faire interagir entre eux plusieurs objets
- Ainsi, en POO, tout est objet
- La **programmation orientée objet** (ou **POO**) a été introduite avec la **version 4** de **PHP**
- C'est, néanmoins, avec la **version 5** du langage qu'il gagnera ses lettres de noblesse pour devenir comparable à des **langages objets** comme **C++** ou **Java**

POO - Les avantages

- Une approche objet présente plusieurs **avantages** :
  - Une **réutilisation** de code source dans différents projets (réutilisation des classes)
  - Une approche conceptuelle et logique plus **claire** et **organisée** où chaque élément est identifié comme un **objet** ayant son **contexte**, ses **propriétés** et ses **actions**
  - Un code source **modulable** où chaque module est isolé et la création de nouveaux élément n'empiète pas sur le reste du programme
  - Une possibilité de s'adapter à des **designs patterns** pour mieux structurer son code

POO - Les inconvénients

- La **programmation orientée objet** présente cependant certains **inconvénients** :
  - Si le projet est mal conceptualisé et structuré, la **maintenabilité** et la **modularité** en seront impactées
  - Contrairement à la **programmation procédurale**, la **POO** nécessite davantage de **ressources** et donc de **temps d'exécution**

Un objet

- Un **objet** est une représentation virtuelle d'une **chose** ou d'un **concept** associé à des **caractéristiques** et des **actions**
- Un **personnage de jeu de rôle** peut être vu comme un **objet**, par exemple
- Il possédera plusieurs **caractéristiques** : **nom**, **points de vie**, **points de mana**, **points d'attaque**, etc. - Ces caractéristiques correspondent à des valeurs que nous stockerons dans des variables
- Il possédera également certaines **actions** : **attaquer**, **se défendre**, **se déplacer**, **boire une potion**, etc. - Ces actions correspondent à des fonctions

Une classe

- Une **classe** représente un **modèle de données** définissant une **structure commune** qu'auront tous les **objets dérivés** de celle-ci

- Plus concrètement, nous pouvons voir une **classe** comme un **modèle**, un **template** sur lequel nous baser pour créer nos **objets** de ce **type** et reprenant donc la **structure** imposée par cette dernière

- Pour reprendre notre exemple, nous modéliserons une classe **Personnage** dans laquelle nous définirons un certain nombre d'**attributs** et de **méthodes communes** à tous les **personnages de jeu de rôle**

- C'est sur cette **classe** que nous nous baserons pour définir tous nos **objets** de type **Personnage**

Une instance

- Lorsque l'on créé un **objet** à partir d'une **classe**, on dit qu'on **instancie** cette dernière - On créé donc une **instance de la classe**
- On se base donc sur le template définit par la **classe** pour créer un nouvel **objet** disposant de ses propres **attributs** et **méthodes** héritées de sa **classe**
- L'**objet** ainsi créé aura donc le **type de la classe** dont il est **dérivé**
- En reprenant notre exemple, on peut imaginer que les objets **Gandalf**, **Aragorn** et **Legolas** représentent des **instances** (**objets**) de la classe **Personnage**
- Vous l'aurez compris, **une classe n'est pas un objet** !

Déclarer une classe

Introduction à la programmation orientée objet

Un peu de syntaxe

- Voici la **syntaxe** à respecter pour déclarer une **classe PHP**

```php
<?php

class NomDeLaClasse {
    // Attributs

    // Méthodes
}

?>
```

Déclarer des attributs

```php
<?php
# ./Personnage.php
class Personnage {
    // Attributs
    private $nom; // Le nom du personnage
    private $pointsDeVie = 100; // Les points de vie, par défaut à 100
    private $pointsDeMagie = 50; // Les points de magie, par défaut à 50
}

?>
```

Explications

- Par convention, on nommera une **classe** en **Upper Camel Case** - Le nom du fichier qui la contient doit porter exactement le même nom - **Attention à la casse !**
- Les **attributs** représentent des caractéristiques propres à une **classe** - Tout **personnage** possédera un **nom**, un **nombre de points de vie** et un **nombre de points de magie** - Ce sont donc des caractéristiques propres à un personnage
- En **POO**, les **attributs** sont représentés par de simples **variables** - Ici, ***$nom*** est **déclarée sans valeur** - Elle est donc initialisée à ***NULL*** par défaut, contrairement aux autres **attributs** de cette classe
- Le mot-clé **private** permet de rendre l'**attribut inaccessible depuis l'extérieur de la classe**
- Notez que 2 **classes** différentes peuvent posséder des **attributs** de même nom sans générer de conflit

Déclarer des méthodes

```php
<?php
# ./Personnage.php
class Personnage {
    // Attributs
    private $nom; // Le nom du personnage
    private $pointsDeVie = 100; // Les points de vie, par défaut à 100
    private $pointsDeMagie = 50; // Les points de magie, par défaut à 50

    // Méthodes
    public function attaquer() {
        echo "Le personnage attaque !";
    }

    public function defendre() {
        echo "Le personnage se défend";
    }
}

?>
```

Explications

- Les **méthodes** représentent les **actions** applicables à un **objet**
- Il s'agit de **fonctions** standards pouvant prendre un ou plusieurs **arguments** et **retournant** ou non des **valeurs** ou des **objets**
- Comme les **attributs**, les **méthodes** sont déclarées avec un **niveau de visibilité**
- Ici, les **méthodes** sont **publiques** ; elles seront donc **accessibles en dehors de la classe** (donc sur l'**objet** lui même lorsqu'il sera **instancié**)
- Notez enfin que 2 **classes** différentes peuvent posséder des **méthodes** de même **signature** sans générer de conflit

Compléments

Introduction à la programmation orientée objet

Le principe d'encapsulation

- En POO, il est important d'interdire la modification directe des attributs d'un objet afin de garantir la validité des types et des valeurs des données des objets
- Ainsi, par respect de ce principe d'encapsulation, on manipulera nos classes en passant uniquement par leurs méthodes et non leurs attributs
- Pour faire respecter cette contrainte strictement, nous déclarons nos attributs **privés**

Visibilité des attributs et méthodes

- La visibilité des attributs et méthodes permet de définir leur accessibilité au sein du programme
- Dans ce chapitre, nous nous intéresserons à deux types de visibilités : ***public*** et ***private***
- Avec une visibilité public, un attribut ou une méthode est accessible depuis n'importe où dans le programme, que l'on se situe dans la classe ou à l'extérieur
- Au contraire, un attribut ou une méthode private est accessible uniquement au sein de sa propre classe
- Vous l'aurez compris, ces types de visibilité permettent de respecter le principe d'encapsulation

Le constructeur en quelques mots…

- Le **constructeur** représente une **méthode publique** appelée de manière implicite à la **création d'un objet** ; à son **instanciation**
- Comme une **méthode** classique, le **constructeur** peut prendre un ou plusieurs **paramètres** en entrée et lancer plusieurs **instructions**
- En **PHP**, il n'est pas possible de déclarer plusieurs fois la même **méthode** - Ici, il n'est donc pas possible de déclarer plusieurs **constructeurs** aux comportements différents
- On dit que la **surcharge** de méthode n'est pas possible en **PHP**

Le constructeur en image…

```php
<?php
# ./Personnage.php
class Personnage {
    // Attributs
    private $nom; // Le nom du personnage
    private $pointsDeVie = 100; // Les points de vie, par défaut à 100
    private $pointsDeMagie = 50; // Les points de magie, par défaut à 50

    // Méthodes
    public function __construct() {
        echo "Un objet Personnage a été créé !";
    }

    public function attaquer() {
        echo "Le personnage attaque !";
    }

    public function defendre() {
        echo "Le personnage se défend";
    }
}

?>
```

Utiliser des classes et des objets

Introduction à la programmation orientée objet

Instancier une classe

- Nous savons désormais créer le **modèle** qui pourra servir à créer les différents **objets** dérivés de cette **classe** - **Instancier** une **classe** revient à créer un **objet** issus de cette dernière

- En **PHP**, on utilise le mot-clé **new** suivi du nom d'un **classe** pour l'**instancier**

- En réalité, cette simple instruction appelle le **constructeur** (la méthode ***__construct()***) de la **classe** pour construire l'**objet** et le stocker en mémoire

- Voici comment **instancier** un **Personnage** : `$gandalf = new Personnage();`

- Un **objet** est donc représenté par une simple **variable** (ici, ***$gandalf***) dont le **type** est celui de la **classe instanciée** (ici, ***Personnage***)

Accéder aux attributs en écriture

- Voyons maintenant comment donner des **valeurs** aux **attributs** des **objets** que l'on créé

```
23    $gandalf = new Personnage();
24    $aragorn = new Personnage();
25
26    $gandalf->nom = "Gandalf";
27    $gandalf->pointsDeVie = 80;
28    $gandalf->pointsDeMagie = 120;
29
30    $aragorn->nom = "Aragorn";
31    $aragorn->pointsDeVie = 120;
32    $aragorn->pointsDeMagie = 50;
```

- Ainsi nos 2 **objets** ont maintenant des **caractéristiques** qui leur sont propres

Accéder aux attributs en lecture

- La **lecture** de nos **attributs** se fera de la même manière que son **écriture** à ceci près qu'il n'y a plus de notion d'affection de valeur

```
echo $gandalf->nom . " a " . $gandalf->pointsDeVie . " PV et " . $gandalf->pointsDeMagie . " PM.";
```

- Nous obtenons l'affichage suivant : « **Gandalf a 80 PV et 120 PM.** »

Accéder aux méthodes

- L'appel aux **méthodes** d'une **classe** se fait exactement de la même manière que pour accéder à ses **attributs**
- Dans notre exemple, nos méthodes ***attaquer()*** et ***defendre()*** se contentent d'afficher une **chaine de caractères**
- Montrons **Aragorn** se **défendre** puis **attaquer**

```
$aragorn->defendre();
$aragorn->attaquer();
```

- Bien entendu, nous aurions pu ajouter un ou plusieurs **paramètres** à ces **fonctions**

TP - La classe Chien

Introduction à la programmation orientée objet

TP - La classe Chien

- Dans un fichier ***Chien.php***, déclarez une classe ***Chien*** (**Attention à la casse !**)
- Un **chien** est caractérisé par son **nom** et sa **taille** (en **cm**)
- Un **chien** peut être **renommé** et peut **grandir** - Ces méthodes se contentent pour le moment de renvoyer une simple chaîne de caractère descriptive contenant la nouvelle valeur passée en paramètre pour le moment (Exemple : « ***Le chien mesure maintenant 60 cm*** »)
- Testez et affichez le résultat de vos fonctions dans le fichier **Chien.php**, à l'extérieur de la classe

Des questions ?