DÉCOUVERTE DE LA POO

Programmation Orientée Objet en PHP5

I - INTRODUCTION

POO = PHP 5+

- Pourquoi ne pas parler de POO en PHP4 ? **Revient à dire** Au fait, quelle est la différence essentielle entre PHP4 et PHP5 ?
- En PHP5, le modèle objet à été réécrit pour permettre de meilleures performances et offrir plus de fonctionnalitées (essentielles pour la plupart)
- En résumé, la réecriture du modèle objet est la différence majeure entre PHP5 et 4
- PHP5 a donc un modèle objet complet c'est pourquoi on ne parlera pas de POO en PHP4

PROCÉDURAL **VS** ORIENTÉ OBJET

- Centré procédure (code éxécuté ligne après ligne)
- Les fonctions se balladent dans un espace commun (aucune séparation)
- Presque aucune réutilisation du code
- Courbe d'apprentissage très rapide

- Centré objet
- Les fonctions (méthodes) ne se balladent pas dans un espace commun, elles appartiennent aux objets
- Réutilisation et partage du code
- Courbe d'apprentissage lente

PROCÉDURAL **VS** ORIENTÉ OBJET

- function addArticle() {...}
- function updateArticle() {...}
- function deleteArticle() {...}
- function findArticle() {...}
- function findAllArticle() {...}
- function addCategory() {...}

- Article->add()
- Article->update()
- Article->delete()
- Article->find()
- Article->findAll()
- Category->add()

PROCÉDURAL **VS** ORIENTÉ OBJET

- function addArticle() {...}
- function addCategory() {...}
- Les fonctions appelées portent un nom différent car elles cohabitent dans un même espace commun. Si chacune s'appelait add(), l'une écraserait l'autre.

- Article->add()
- Category->add()
- Les fonctions (méthodes) appelées portent le même nom car chacune appartient à un objet et n'existe donc que dans l'environnement de cet objet.

II - COMPRENDRE LA BASE

CLASSE

- Structure d'une classe :
  - Des variables "dynamiques" => **Attributs** (aussi appelés Propriétés)
  - Des variables "fixes" => **Constantes** (pour l'instant on met cette partie de côté)
  - Des fonctions => **Méthodes**

EXEMPLE D'UNE CLASSE

```php
<?php

class Article
{
    /*
     * Contenu de ma classe
     */
}
```

OBJET

- Structure d'un objet :
  - Des variables => **Attributs** (aussi appelés Propriétés)
  - Des fonctions => **Méthodes**

EXEMPLE D'UN OBJET

```php
<?php

class Article
{
    /*
     * Contenu de ma classe
     */
}

$article = new Article();
```

CLASSE & OBJET

- Une Classe est unique et définit des attributs et des méthodes (c'est en quelque sorte un plan de fabrication)
- Un Objet est le résultat de l'intanciation d'une classe et intègre donc les attributs et méthodes de la classe instanciée (c'est l'objet crée à partir du plan de fabrication)

INSTANCIATION

- J'ai une classe appelée "Article"
- La classe "Article" dispose du "plan de construction" d'un article
- Pour avoir un objet qui contienne l'article (titre + contenu) il faut que j'instancie la classe "Article" : pour l'exemple, je stocke l'object (l'article) dans la variable $article
- $article est donc un objet qui contient mon article, il résulte donc de l'instanciation de la classe "Article". On dit que $article est une instance de la classe "Article"
- Pour faire grossier : **une instance est un objet**

ENCAPSULATION

- Ajouter à la classe des attributs / méthodes disposant de **visibilité**
- Permettre à l'utilisateur de la classe de se concentrer sur l'utilisation de celle-ci et non son fonctionnement interne
- On doit donc interdire l'accès à des attributs / méthodes depuis l'extérieure de la classe
- Pour cela, on utilisera les mots-clefs **public** et **private**

VISIBILITÉ

- La visibilité s'applique aux attributs et aux méthodes
- Deux types de visibilité : **public** et **private**
- Un attribut ou une méthode dite **publique** peut-être appelée **à partir de l'objet instancié**
- Un attribut ou une méthode dite **privée** ne peut-être appelée **que par l'objet lui-même**

VISIBILITÉ - PUBLIC

```php
<?php

class Article
{
    public $id;
}

$article = new Article();
$article->id;
```

L'objet instancié $article peut accéder à l'attribut $id

VISIBILITÉ - PRIVATE

```php
<?php

class Article
{
    private $id;
}

$article = new Article();
$article->id;
```

Member has private access

- L'objet instancié $article ne peut pas accéder à l'attribut $id

SETTERS / GETTERS

- La création de setters et de getters est implicitement lié au principe d'encapsulation
- L'utilisateur d'une classe ne doit pas avoir accès aux attributs de la classe
- Pour pouvoir **définir** une valeur à un **attribut privé $id**, il faut créer un **setter** public qui définira la valeur de l'attribut $id
- Pour pouvoir **récupérer** la valeur de l'**attribut privé $id**, il faut crée un **getter** public qui récupérera la valeur de l'attribut $id

EXEMPLE D'UTILISATION DE SETTERS / GETTERS

```php
<?php

class Article
{
    private $id;

    public function setId($id)
    {
        $this->id = (int) $id;
    }

    public function getId()
    {
        return $this->id;
    }
}
```

```php
$article = new Article();
$article->setId(5);
$article->getId();
$article->id; // Error
```

RECAPITULATIF

Classe

Instanciation

Private

Objet

Visibilité

Setters

Public

Encapsulation

Getters