Programmation Orientée Objet (POO) en PHP

Par Robert DIASSÉ

Introduction

La programmation orientée objet (POO) est un paradigme qui permet d'organiser et de structurer le code de manière plus efficace, en modélisant des entités sous forme d'objets. Contrairement à la programmation procédurale, qui repose sur une séquence d'instructions, la POO permet de **structurer le code autour d'objets** possédant des propriétés et des comportements.

Pourquoi utiliser la POO ?

- **Réutilisation du code** : grâce à l'héritage, on peut éviter les répétitions.
- **Organisation et maintenance** : le code est plus structuré et facile à comprendre.
- **Encapsulation** : protège les données des accès non désirés.
- **Extensibilité** : facilite l'ajout de nouvelles fonctionnalités.

1. Classes et Objets

Le Modèle du Monde Réel

Une classe est un modèle qui définit des objets ayant des caractéristiques et des comportements. Imagine un plan d'architecte pour une maison : il définit la structure, mais ce n'est qu'une fois construite que la maison devient réelle. De la même manière, une classe en POO est un modèle, et un objet est une instance de ce modèle.

Exemple : Une recette de cuisine est une classe (modèle) et chaque plat préparé selon cette recette est un objet.

Dans un jeu vidéo, un personnage possède des caractéristiques (**points de vie, force, vitesse**) et des comportements (**attaquer, courir, sauter**). On peut modéliser cela en POO :

```php
class Personnage {
    public $nom;
    public $vie = 100;
```

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

    public function attaquer() {
        echo "$this->nom attaque !";
    }
}

$joueur1 = new Personnage("Héros");
$joueur1->attaquer();
```

Explication :

- La classe `Personnage` définit une structure pour nos personnages.
- Le `__construct()` est un **constructeur**, une méthode spéciale qui est exécutée automatiquement lors de la création de l'objet.
- `$this->nom` permet de faire référence à l'attribut `nom` de l'objet en cours.
- `new Personnage("Héros")` crée une instance de `Personnage`.
- La méthode `attaquer()` affiche un message d'attaque.

Exercice 1 :

Créez une classe `Voiture` avec :

- Un attribut `marque`
- Un constructeur qui initialise `marque`
- Une méthode `rouler()` qui affiche un message indiquant que la voiture roule.

---

2. Encapsulation et visibilité (`public`, `private`, `protected`)

La Protection des Données

L'encapsulation protège l'accès aux données d'un objet en contrôlant leur visibilité.
Elle restreint l'accès direct aux attributs d'un objet pour éviter toute modification accidentelle. C'est comme une boîte avec un cadenas : seules les personnes ayant la clé (les méthodes publiques) peuvent y accéder.
Exemple : Un distributeur de billets ne permet pas d'accéder directement à l'argent. L'utilisateur doit suivre un processus (entrer un code) pour obtenir l'argent, tout comme les méthodes d'un objet permettent de modifier des attributs privés en toute sécurité.

```
class CompteBancaire {
    private $solde = 0; // Accessible uniquement dans la classe

    public function deposer($montant) {
        $this->solde += $montant;
    }

    public function voirSolde() {
```

```
        return $this->solde;
    }
}

$compte = new CompteBancaire();
$compte->deposer(500);
echo $compte->voirSolde();
```

Explication :

- `private $solde` empêche l'accès direct au solde depuis l'extérieur de la classe.
- `deposer()` modifie la valeur de `$solde`.
- `voirSolde()` retourne le solde actuel.

Exercice 2 :

Ajoutez une méthode `retirer($montant)` qui permet de retirer de l'argent du compte, en s'assurant que le solde est suffisant.

---

3. Héritage

Réutilisation et Extension

L'héritage permet de créer des classes dérivées d'une classe existante.
Elle permet de créer une nouvelle classe basée sur une autre, évitant ainsi de réécrire du code redondant.
Exemple : Une voiture de sport et une voiture électrique sont toutes deux des types de voitures (classe parent), mais elles ont leurs propres spécificités (classes filles). Une voiture de sport peut avoir un moteur puissant, tandis qu'une voiture électrique a une batterie spécifique, mais elles partagent toutes deux des attributs communs comme les roues et le volant.

```
class Animal {
    protected $nom;
    public function __construct($nom) {
        $this->nom = $nom;
    }
    public function parler() {
        echo "$this->nom fait un bruit";
    }
}

class Chien extends Animal {
    public function parler() {
        echo "$this->nom aboie";
    }
}

$rex = new Chien("Rex");
$rex->parler();
```

Explication :

- `Chien` hérite de `Animal`, il récupère les attributs et méthodes.
- `parler()` est redéfini pour changer son comportement.

Exercice 3 :

Créez une classe `Chat` qui hérite de `Animal` et affiche "miaule".

---

4. Polymorphisme

Un Comportement Adaptable

Le polymorphisme permet d'avoir plusieurs méthodes portant le même nom mais ayant des comportements différents.
Il permet d'utiliser une même méthode pour différents types d'objets. Cela permet d'écrire du code plus flexible.
Exemple : Un interrupteur allume différentes lampes, ventilateurs ou appareils électroniques selon l'endroit où il est utilisé. Tous ces objets ont un bouton "ON/OFF", mais l'effet est différent selon l'appareil.

```php
class Forme {
    public function aire() {
        return 0;
    }
}

class Carre extends Forme {
    private $cote;
    public function __construct($cote) {
        $this->cote = $cote;
    }
    public function aire() {
        return $this->cote * $this->cote;
    }
}
```

Exercice 4 :

Ajoutez une classe `Cercle` avec une méthode `aire()` qui utilise `pi() * rayon²`.

---

5. Gestion des exceptions

La Sécurité du Code

Les exceptions permettent de gérer les erreurs et d'empêcher les crashs.
Exemple : Un ascenseur ne doit pas fonctionner si les portes sont ouvertes. Si quelqu'un tente d'appuyer sur un bouton pour monter alors que les portes sont ouvertes, un message d'erreur s'affiche plutôt que de démarrer le moteur.

```php
class Division {
    public function diviser($a, $b) {
        if ($b == 0) {
            throw new Exception("Division par zéro interdite !");
        }
        return $a / $b;
    }
}

try {
    $calcul = new Division();
    echo $calcul->diviser(10, 0);
} catch (Exception $e) {
    echo "Erreur : " . $e->getMessage();
}
```

Exercice 5 :

Créez une exception pour empêcher un solde bancaire négatif.

---

6. Comparaison avec le procédural

La programmation procédurale consiste à exécuter des instructions ligne par ligne, tandis que la POO regroupe les données et leur traitement dans des objets.
Un restaurant géré en procédural serait une cuisine où chaque employé doit se rappeler des recettes par cœur. En POO, on a un chef qui gère les recettes sous forme de fiches bien organisées, permettant une meilleure gestion et un service plus efficace.

```php
// Procédural
$nom = "Rex";
function parler($nom) {
    echo "$nom aboie";
}
parler($nom);

// Objet
$rex = new Chien("Rex");
$rex->parler();
```

Projet: Gestion d'une Bibliothèque

1. Créer la classe `Livre` avec `titre`, `auteur`, `disponible`.
2. Créer la classe `Membre` avec `nom`, `prenom` et une méthode pour emprunter un livre.
3. Créer la classe `Bibliothèque` qui stocke une collection de livres.
4. Gérer l'ajout, le retrait et l'affichage des livres.

5. Utiliser la gestion des exceptions pour éviter les erreurs d'emprunt.