PHP - 03 : Programmation Orientée Objet (POO)

La Programmation Orientée Objet (POO) est un paradigme de programmation basé sur la conception et la manipulation d'objets, qui regroupent des données (attributs) et des comportements (méthodes). En PHP, la POO est un puissant outil permettant de structurer et réutiliser le code de manière plus efficace et modulaire.

1. Concepts fondamentaux de la POO

1. Classe : C’est un modèle ou un plan de création d’objets. Une classe définit les attributs et les méthodes que l'objet aura.

2. Objet : Une instance d'une classe.

3. Attributs : Les variables qui appartiennent à une classe ou un objet. Elles stockent les données de l'objet.

4. Méthodes : Les fonctions qui appartiennent à une classe ou un objet. Elles définissent les comportements de l'objet.

5. Encapsulation : Limitation de l'accès direct aux données (grâce à des modificateurs comme private).

6. Héritage : Une classe peut hériter des attributs et des méthodes d'une autre classe.

7. Polymorphisme : Les objets de différentes classes peuvent être traités comme des objets de la même classe.

2. Création d'une classe et d'objets en PHP

Exemple 1 : Classe simple

```
<?php
class Voiture {
    // Attributs
    public $marque;
    public $couleur;

    // Méthodes
    public function demarrer() {
        echo "La voiture démarre.";
    }

    public function afficherDetails() {
        echo "Marque : $this->marque, Couleur : $this->couleur.";
    }
}

// Création d'objets
$voiture1 = new Voiture();
$voiture1->marque = "Toyota";
$voiture1->couleur = "Rouge";

$voiture1->demarrer(); // Affiche "La voiture démarre."
$voiture1->afficherDetails(); // Affiche "Marque : Toyota, Couleur : Rouge."
?>
```

3. Modificateurs d’accès : public, private, protected

- *public : Accessible partout.*
- *private : Accessible uniquement à l'intérieur de la classe.*
- *protected : Accessible à l'intérieur de la classe et des classes qui en héritent.*

Exemple 2 : Encapsulation avec private

```
<?php
class CompteBancaire {
    private $solde = 0;

    public function deposer($montant) {
        $this->solde += $montant;
        echo "Dépôt effectué : $montant €.";
    }

    public function afficherSolde() {
        echo "Solde actuel : $this->solde €.";
    }
}

$compte = new CompteBancaire();
$compte->deposer(100);
$compte->afficherSolde(); // Affiche "Solde actuel : 100 €."
?>
```

4. Constructeurs et destructeurs

Le constructeur est une méthode spéciale qui est appelée automatiquement lors de l'instanciation d'une classe.

Le destructeur est appelé lorsque l'objet est détruit.

Exemple 3 : Utilisation d'un constructeur

```
<?php
class Utilisateur {
    private $nom;

    // Constructeur
    public function __construct($nom) {
        $this->nom = $nom;
        echo "Utilisateur $nom créé.";
    }

    public function afficherNom() {
        echo "Nom : $this->nom";
    }
}

$utilisateur1 = new Utilisateur("Alice");
$utilisateur1->afficherNom(); // Affiche "Nom : Alice"
?>
```

5. Héritage

Une classe peut hériter d'une autre grâce au mot-clé extends.

Exemple 4 : Classe fille héritant d’une classe mère

```
<?php
class Animal {
    public function manger() {
        echo "Cet animal mange.";
    }
}

class Chien extends Animal {
    public function aboyer() {
        echo "Le chien aboie.";
    }
}

$chien = new Chien();
$chien->manger(); // Hérité de la classe Animal
$chien->aboyer(); // Méthode propre à la classe Chien
?>
```

6. Polymorphisme

Le polymorphisme permet de redéfinir les méthodes dans une classe enfant.

Exemple 5 : Redéfinir une méthode

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

class Rectangle extends Forme {
    private $longueur;
    private $largeur;

    public function __construct($longueur, $largeur) {
        $this->longueur = $longueur;
        $this->largeur = $largeur;
    }

    public function aire() {
        return $this->longueur * $this->largeur;
    }
}

$rectangle = new Rectangle(5, 3);
echo $rectangle->aire(); // Affiche 15
?>
```

7. Interfaces et classes abstraites

1. Classe abstraite : Une classe qui ne peut pas être instanciée directement. Elle peut contenir des méthodes abstraites (sans implémentation).

```
<?php
abstract class Vehicule {
    abstract public function demarrer();
}

class Moto extends Vehicule {
    public function demarrer() {
        echo "La moto démarre.";
    }
}
?>
```

2. Interface : Déclare des méthodes que les classes doivent implémenter.

```
<?php
interface Volant {
    public function voler();
}

class Avion implements Volant {
    public function voler() {
        echo "L'avion vole.";
    }
}
?>
```

Conclusion

La POO en PHP permet d'organiser le code de manière claire, modulaire et évolutive. Avec ces bases, vous pouvez construire des applications robustes et réutilisables. Pour aller plus loin, explorez les design patterns, namespace, et traits.