Programmation Orientée Objet (POO) en PHP - Rappel des bases

La programmation orientée objet (POO) est un paradigme de programmation qui repose sur les concepts de **classes** et **objets**. En PHP, elle permet d'organiser le code, de le rendre plus modulaire, réutilisable et maintenable.

---

1. Concepts clés de la POO

1.1. Classe et Objet

- Une **classe** est un modèle ou un plan qui définit les propriétés (variables) et les méthodes (fonctions) qu'un objet peut avoir.
- Un **objet** est une instance d'une classe.

Exemple :

```php
<?php
// Définir une classe
class Person {
    public $name; // Propriété
    public $age;  // Propriété

    // Méthode
    public function greet() {
        return "Bonjour, je m'appelle " . $this->name;
    }
}

// Créer un objet
$person = new Person();
$person->name = "Alice";
$person->age = 30;

echo $person->greet(); // Affiche : Bonjour, je m'appelle Alice
?>
```

1.2. Propriétés et Méthodes

- **Propriétés** : Variables définies dans une classe.
- **Méthodes** : Fonctions définies dans une classe.

Exemple :

```php
<?php
class Calculator {
```

```php
    public $number1;
    public $number2;

    public function add() {
        return $this->number1 + $this->number2;
    }
}

$calc = new Calculator();
$calc->number1 = 5;
$calc->number2 = 10;

echo $calc->add(); // Affiche : 15
?>
```

1.3. Modificateurs d'accès

Les modificateurs d'accès contrôlent la visibilité des propriétés et méthodes :

- **public** : Accessible partout.
- **private** : Accessible uniquement dans la classe.
- **protected** : Accessible dans la classe et ses sous-classes.

Exemple :

```php
<?php
class BankAccount {
    private $balance = 0; // Propriété privée

    public function deposit($amount) {
        $this->balance += $amount;
    }

    public function getBalance() {
        return $this->balance;
    }
}

$account = new BankAccount();
$account->deposit(100);
// echo $account->balance; // Erreur : balance est private
echo $account->getBalance(); // Affiche : 100
?>
```

1.4. Constructeurs

Un **constructeur** est une méthode spéciale appelée automatiquement lors de l'instanciation d'un objet.

Exemple :

```php
<?php
class Product {
    public $name;
    public $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

    public function display() {
        return "$this->name coûte $this->price €.";
    }
}

$product = new Product("Ordinateur", 1500);
echo $product->display(); // Affiche : Ordinateur coûte 1500 €.
?>
```

---

2. Concepts avancés de base

2.1. Héritage

L'héritage permet à une classe d'hériter des propriétés et méthodes d'une autre classe.

Exemple :

```php
<?php
class Animal {
    public function eat() {
        return "Je mange.";
    }
}

class Dog extends Animal {
    public function bark() {
        return "Wouf! Wouf!";
    }
}

$dog = new Dog();
echo $dog->eat(); // Hérité de Animal
echo $dog->bark(); // Défini dans Dog
?>
```

2.2. Interfaces

Une interface définit un ensemble de méthodes que les classes implémentantes doivent obligatoirement inclure.

Exemple :

```php
<?php
interface Flyable {
    public function fly();
}

class Bird implements Flyable {
    public function fly() {
        return "Je vole.";
    }
}

$bird = new Bird();
echo $bird->fly(); // Affiche : Je vole.
?>
```

2.3. Encapsulation

L'encapsulation consiste à restreindre l'accès direct aux propriétés en utilisant des **getters** et **setters**.

Exemple :

```php
<?php
class User {
    private $password;

    public function setPassword($password) {
        $this->password = md5($password); // Encapsulation du traitement
    }

    public function getPassword() {
        return $this->password;
    }
}

$user = new User();
$user->setPassword("secret");
echo $user->getPassword(); // Affiche le mot de passe hashé
?>
```

2.4. Polymorphisme

Le polymorphisme permet d'utiliser une méthode de différentes manières via l'héritage ou les interfaces.

Exemple :

```php
<?php
class Shape {
    public function draw() {
        return "Je dessine une forme.";
    }
}

class Circle extends Shape {
    public function draw() {
        return "Je dessine un cercle.";
    }
}

$shape = new Shape();
echo $shape->draw(); // Affiche : Je dessine une forme.

$circle = new Circle();
echo $circle->draw(); // Affiche : Je dessine un cercle.
?>
```

3. Avantages de la POO

1. **Réutilisabilité** : Le code peut être réutilisé sous forme de classes ou modules.
2. **Modularité** : Le code est organisé en petites parties indépendantes.
3. **Maintenance** : Les changements dans une partie du code n'affectent pas les autres parties.
4. **Extensibilité** : Les nouvelles fonctionnalités peuvent être ajoutées facilement.

Si tu souhaites approfondir un concept ou explorer d'autres exemples pratiques, fais-le-moi savoir ! 😊