Introduction à la Programmation Orientée Objet en PHP

La programmation orientée objet (POO) est un paradigme de programmation qui repose sur le concept d'objets. En PHP, la POO permet de structurer et d'organiser le code pour le rendre plus modulaire, réutilisable et facile à maintenir. Voici les concepts fondamentaux :

Création de Classes et d'Objets

Une classe est un modèle ou un plan qui définit les propriétés et les méthodes associées à un type d'objet. Voici un exemple de création d'une classe et d'un objet :

```
class Voiture {
    public $marque;
    public $couleur;

    public function __construct($marque, $couleur) {
        $this->marque = $marque;
        $this->couleur = $couleur;
    }

    public function afficherDetails() {
        return "Cette voiture est une {$this->marque} de couleur {$this->couleur}.";
    }
}

// Création d'un objet
$maVoiture = new Voiture("Toyota", "Rouge");
echo $maVoiture->afficherDetails();
```

Encapsulation et Modificateurs d'Accès

L'encapsulation est un principe essentiel de la POO qui consiste à restreindre l'accès direct à certaines propriétés ou méthodes d'une classe pour protéger l'intégrité des données. En PHP, cela se fait grâce aux modificateurs d'accès :

Public, Private et Protected

- **Public** : Accès sans restriction à la propriété ou méthode.
- **Private** : Accès limité à l'intérieur de la classe uniquement.
- **Protected** : Accès à l'intérieur de la classe et de ses classes dérivées.

Exemple d'Encapsulation :

```
class CompteBancaire {
    private $solde;

    public function __construct($soldeInitial) {
        $this->solde = $soldeInitial;
    }

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

    public function retirer($montant) {
        if ($montant > $this->solde) {
            return "Fonds insuffisants.";
        }
        $this->solde -= $montant;
    }

    public function afficherSolde() {
        return "Votre solde est de {$this->solde} euros.";
    }
}

$compte = new CompteBancaire(100);
$compte->deposer(50);
echo $compte->afficherSolde();
```

Héritage et Polymorphisme

Héritage

L'héritage permet à une classe (classe enfant) de réutiliser les propriétés et méthodes d'une autre classe (classe parente). Cela favorise la réutilisation du code.

Exemple d'Héritage :

```
class Animal {
    public $nom;

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

    public function manger() {
```

```
        return "{$this->nom} mange.";
    }
}

class Chien extends Animal {
    public function aboyer() {
        return "{$this->nom} aboie.";
    }
}

$chien = new Chien("Rex");
echo $chien->manger();
echo $chien->aboyer();
```

Polymorphisme

Le polymorphisme permet à une méthode d'avoir plusieurs comportements différents en fonction du contexte ou de la classe qui l'implémente.

Exemple de Polymorphisme :

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

class Cercle extends Forme {
    private $rayon;

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

    public function aire() {
        return pi() * pow($this->rayon, 2);
    }
}

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

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

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

$formes = [new Cercle(3), new Rectangle(4, 5)];

foreach ($formes as $forme) {
    echo "L'aire est : " . $forme->aire() . "\n";
}
```

Ces concepts sont les piliers de la programmation orientée objet en PHP et permettent de créer des applications modulaires, évolutives et maintenables.