Introduction à la Programmation Orientée Objet en PHP

La programmation orientée objet (POO) est un paradigme de programmation qui repose sur l'utilisation de classes et d'objets pour structurer le code. En PHP, la POO permet de créer des applications modulaires, réutilisables et maintenables.

Une classe est un plan ou un modèle qui définit les propriétés (attributs) et les comportements (méthodes) d'un objet. Un objet, quant à lui, est une instance concrète de cette classe.

Voici un exemple simple pour illustrer ces concepts :

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

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

$voiture1 = new Voiture();
$voiture1->marque = "Toyota";
$voiture1->couleur = "Rouge";
$voiture1->demarrer();
```

Dans cet exemple, la classe `Voiture` définit des attributs et une méthode. L'objet `$voiture1` est une instance de la classe `Voiture`.

Encapsulation et Modificateurs d'Accès

L'encapsulation est l'un des piliers de la POO. Elle consiste à restreindre l'accès direct aux données sensibles d'une classe en utilisant des modificateurs d'accès.

Les principaux modificateurs d'accès en PHP sont :

- public : L'attribut ou la méthode est accessible depuis n'importe où.
- private : L'attribut ou la méthode est accessible uniquement au sein de la classe où il est défini.
- protected : L'attribut ou la méthode est accessible uniquement au sein de la classe où il est défini et des classes qui en héritent.

Exemple d'encapsulation :

```
class Banque {
    private $solde;

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

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

```
    public function getSolde() {
        return $this->solde;
    }
}

$compte = new Banque(1000);
$compte->deposer(500);
echo $compte->getSolde(); // Affiche 1500
```

Dans cet exemple, l'attribut `$solde` est privé et ne peut être accédé directement. Des méthodes publiques permettent d'interagir avec cet attribut de manière contrôlée.

Héritage et Polymorphisme

L'héritage permet à une classe d'hériter des propriétés et méthodes d'une autre classe. Cela favorise la réutilisation du code.

Exemple d'héritage :

```
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(); // Cet animal mange.
$chien->aboyer(); // Le chien aboie.
```

Le polymorphisme, quant à lui, permet à des objets de classes différentes de réagir différemment à la même méthode.

Exemple de polymorphisme :

```
class Forme {
    public function dessiner() {
        echo "Dessiner une forme.";
    }
}

class Cercle extends Forme {
    public function dessiner() {
        echo "Dessiner un cercle.";
    }
}

class Rectangle extends Forme {
    public function dessiner() {
        echo "Dessiner un rectangle.";
    }
```

```
}

$formes = [new Cercle(), new Rectangle()];
foreach ($formes as $forme) {
    $forme->dessiner();
}
```

Dans cet exemple, la méthode `dessiner` se comporte différemment en fonction de la classe de l'objet.

Ce document met en évidence les bases de la programmation orientée objet en PHP, en insistant sur l'importance de l'encapsulation, de l'héritage et du polymorphisme pour créer un code à la fois robuste et facile à maintenir.