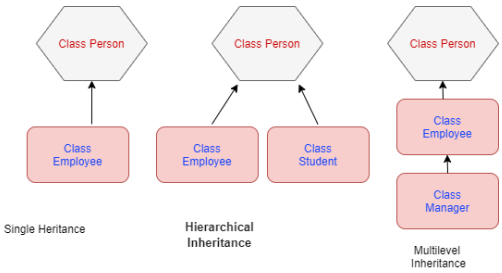
La programmation orientée objet (POO) en PHP (à partir de la version 5)

En programmation orientée objet (POO), les concepts d'héritage, d'encapsulation, d'interface et d'abstraction sont fondamentaux pour structurer et organiser le code. Voici une explication de chacun de ces concepts avec des exemples en PHP :

1. Héritage



L'héritage permet à une classe de dériver d'une autre classe. La classe dérivée (ou sous-classe) hérite des propriétés et des méthodes de la classe parente (ou super-classe). Cela permet de réutiliser le code existant et d'ajouter ou de modifier des fonctionnalités.

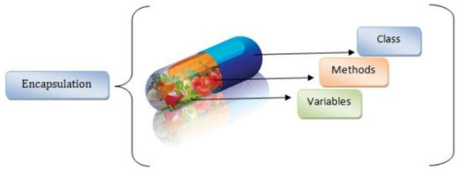
Exemple :

```php
class Animal {
    public $name;

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

    public function speak() {
        return "Sound";
    }
}
```

```php
class Dog extends Animal {
    public function speak() {
        return "Woof!";
    }
}

$dog = new Dog("Buddy");
echo $dog->name; // Buddy
echo $dog->speak(); // Woof!
```

Dans cet exemple, la classe `Dog` hérite de la classe `Animal`. Elle peut accéder aux propriétés et méthodes de `Animal`, mais elle peut aussi redéfinir la méthode `speak()`.

2. Encapsulation



L'encapsulation consiste à restreindre l'accès direct à certaines propriétés ou méthodes d'un objet. Cela se fait en définissant des niveaux de visibilité (`public`, `protected`, `private`) qui déterminent comment ces éléments peuvent être utilisés.

- `public` : Accessible de partout.
- `protected` : Accessible seulement depuis la classe elle-même et ses sous-classes.
- `private` : Accessible seulement depuis la classe elle-même.

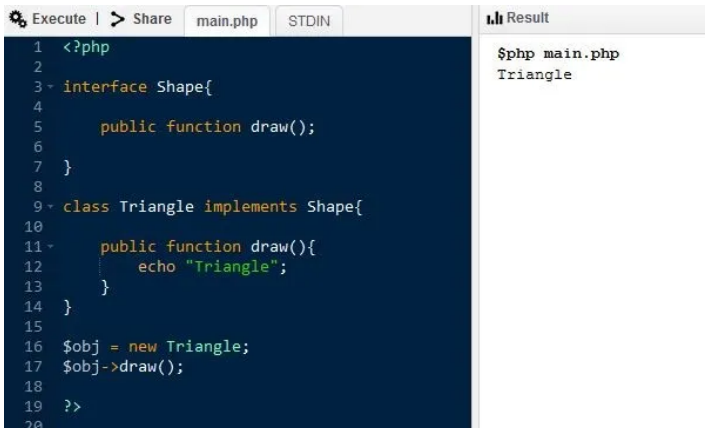
Exemple :

```php
class Car {
    private $engine;

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

```
    public function startEngine() {
        return "Engine started: " . $this->engine;
    }

    private function checkOil() {
        return "Oil level is good";
    }
}

$car = new Car("V8");
echo $car->startEngine(); // Engine started: V8
// echo $car->checkOil(); // Erreur: méthode privée inaccessible
```

Ici, la méthode `checkOil()` est privée et ne peut être appelée qu'à l'intérieur de la classe `Car`. La propriété `$engine` est également privée, ce qui empêche son accès direct en dehors de la classe.

3. Interface



Une interface définit un ensemble de méthodes qu'une classe doit implémenter. Elle ne contient que les signatures des méthodes, sans implémentation. Une classe qui implémente une interface doit définir toutes les méthodes de l'interface.

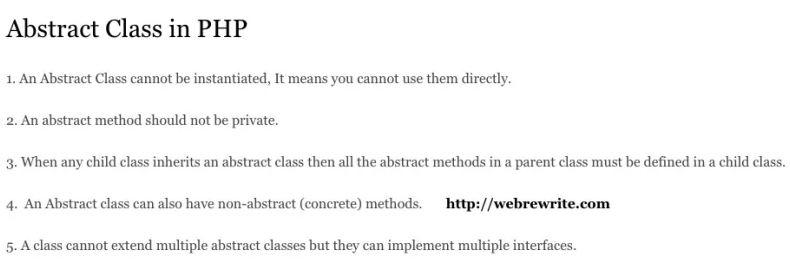
Exemple :

```
interface Movable {
    public function move();
}
```

```php
class Car implements Movable {
    public function move() {
        return "The car moves forward";
    }
}

class Bicycle implements Movable {
    public function move() {
        return "The bicycle pedals forward";
    }
}

$car = new Car();
echo $car->move(); // The car moves forward

$bicycle = new Bicycle();
echo $bicycle->move(); // The bicycle pedals forward
```

Dans cet exemple, l'interface `Movable` définit une méthode `move()`. Les classes `Car` et `Bicycle` implémentent cette interface et définissent leur propre version de `move()`.

4. Abstraction



L'abstraction consiste à définir des classes abstraites qui ne peuvent pas être instanciées directement. Elles peuvent contenir des méthodes abstraites (sans implémentation) que les sous-classes doivent implémenter, ainsi que des méthodes concrètes (avec implémentation).

Exemple :

```php
abstract class Shape {
    protected $color;

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

    abstract protected function area();

    public function getColor() {
        return $this->color;
    }
}

class Circle extends Shape {
    private $radius;

    public function __construct($radius, $color) {
        $this->radius = $radius;
        parent::__construct($color);
    }

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

$circle = new Circle(5, "red");
echo $circle->area(); // 78.539816339745
echo $circle->getColor(); // red
```

Ici, `Shape` est une classe abstraite avec une méthode abstraite `area()` que toutes les sous-classes doivent implémenter. La classe `Circle` implémente cette méthode et fournit une fonctionnalité spécifique.

Conclusion

- **Héritage** : Permet à une classe de dériver d'une autre et de réutiliser ou d'étendre son comportement.
- **Encapsulation** : Protège les données internes d'une classe en con-

trôlant l'accès à ces données.

- **Interface** : Définit un contrat que les classes doivent suivre, en spécifiant des méthodes sans implémentation.
- **Abstraction** : Fournit une classe de base avec des méthodes abstraites que les sous-classes doivent implémenter, et éventuellement des méthodes concrètes.

Ces concepts sont essentiels pour écrire un code modulaire, maintenable et réutilisable en PHP.