# Fiche de révision

## Le Polymorphisme d'Interface

## 1. Qu'est-ce que c'est ?

Le polymorphisme est un concept fondamental en programmation orientée objet (POO) qui permet aux objets de différentes classes d'être traités comme des instances d'une même classe parent.

Il offre une manière flexible de travailler avec différents types d'objets qui partagent une interface commune, tout en ayant leur propre implémentation spécifique des méthodes demandées par l'interface.

**Polymorphisme d'Interface :** Cela permet à différentes classes d'implémenter la même interface et de fournir des implémentations spécifiques pour les méthodes de cette interface.

**"Some people, when confronted with a problem, think - I know, I'll use regular expressions. - Now they have two problems."**

## 2. Comment faire ?

Il faut mettre en place une interface ICrier aux deux classes « Chat » et « Chien »

### 2.1. Création de l'interface

```
interface ICrier {
    public function crier();
}
```

### 2.2. Création d'une première classe

La classe « Chat » implémente l'interface ICrier et doit implémenter la méthode « crier () ».

```
class Chat implements ICrier {
    public function crier() {
        echo "Le chat miaule.";
    }
}
```

### 2.3. Création d'une deuxième classe

La classe « Chien » implémente l'interface ICrier et doit implémenter la méthode « crier () ».

```
class Chien implements ICrier {
    public function crier() {
        echo "Le chien aboie.";
    }
}
```

### 2.4. Création d'une fonction utilisant l'interface

La fonction « faireCrier () » demande en paramètre un objet de la classe parent « Animal ».

```
<?php function faireCrier(ICrier $animal) {
    $animal->crier();
}
```

### 2.5. Utilisation de la fonction avec des objets des classes enfants

Dorénavant, vous pouvez instancier des objets d'une classe des classes implémentant ICrier et les donner en paramètres dans toute méthode ou fonction demandant un objet de le l'Interface « ICrier ».

```
$chien = new Chien();
$chat = new Chat();

faireCrier($chien); // Affiche "Le chien aboie."
faireCrier($chat);  // Affiche "Le chat miaule."
```

## 3. Bonnes Pratiques

Définir des Contrats Clairs : Les interfaces doivent définir des contrats clairs et précis. Les méthodes déclarées dans une interface doivent avoir des signatures bien définies que toutes les classes implémentantes doivent suivre.

- **Principe de Substitution de Liskov** : Toutes les classes implémentant une interface doivent être interchangeables. Un utilisateur de l'interface ne doit pas être en mesure de distinguer entre les différentes implémentations de l'interface.
- **Minimiser l'Interface** : Les interfaces ne doivent contenir que les méthodes nécessaires. Evitez les interfaces surchargées, car elles peuvent forcer les classes implémentantes à définir des méthodes dont elles n'ont pas besoin.
- **Implémentation Orientée Interface** : Programmez en fonction des interfaces plutôt que des implémentations. Cela signifie que vous devriez utiliser le type d'interface pour déclarer une variable ou un paramètre de méthode chaque fois que c'est possible, plutôt que d'utiliser le type d'une classe spécifique.

- **Réutilisation des Interfaces** : Réutilisez les interfaces existantes autant que possible plutôt que de créer de nouvelles interfaces qui dupliquent la même fonctionnalité.
- **Testabilité** : Le polymorphisme d'interface facilite le test de votre code. Vous pouvez créer des mock objects qui implémentent les interfaces pour tester le comportement de votre code sans dépendre des implémentations concrètes.
- **Documentation** : Documentez clairement chaque interface et son intention. Spécifiez ce que chaque méthode est censée faire et quelles sont les attentes en termes de valeurs d'entrée et de sortie.

En suivant ce guide, vous pourrez utiliser le polymorphisme en PHP pour créer des applications plus modulaires, flexibles et maintenables.