Fiche de révision

L’héritage en PHP

1. Qu’est-ce que c’est ?

L'héritage en programmation orientée objet (POO) est un principe fondamental qui permet à une classe d’hériter des attributs (propriétés) et des méthodes d'une autre classe. Ce concept est crucial pour la réutilisation du code et la création d'une hiérarchie de classes organisée et maintenable.

Voici les points clés de l'héritage en POO :

- **Classe de base** (ou classe parent) : C'est la classe dont les propriétés et méthodes seront héritées. On l'appelle parfois la superclasse.
- **Classe dérivée** (ou classe enfant) : C'est la classe qui hérite d'une autre classe. Elle peut utiliser toutes les propriétés et méthodes publiques et protégées de la classe parent, en fonction des règles d'accès définies. On l'appelle parfois la sous-classe.
- **Réutilisation du code** : L'héritage permet aux développeurs de réutiliser le code existant, réduisant ainsi la redondance et améliorant la maintenabilité du code.
- **Spécialisation** : Les classes enfant peuvent étendre ou spécialiser le comportement des classes parent en surchargeant les méthodes existantes ou en ajoutant de nouvelles propriétés et méthodes.
- **Polymorphisme** : L'héritage est étroitement lié au polymorphisme, un principe de la POO qui permet aux objets de différentes classes liées par héritage d'être traités de manière interchangeable.
- **Constructeurs et destructeurs** : Les classes dérivées héritent des constructeurs et destructeurs de la classe parent, mais peuvent également définir les leurs. Si un constructeur est défini dans la classe enfant, il est courant d'appeler explicitement le constructeur de la classe parent à l'intérieur de celui-ci.

En PHP, l'héritage est implémenté en utilisant le mot-clé **extends** lorsque vous déclarez une classe. L'héritage permet de structurer les applications de manière hiérarchique et logique, facilitant la gestion de la complexité dans les grands systèmes.

“A good programmer is someone who always looks both ways before crossing a one-way street.” - Doug Linder

2. Comment faire ?

2.1. Définir la classe Parent

La première étape consiste à définir une classe parent qui contiendra les propriétés et méthodes communes que les classes enfants vont hériter.

```
class Vehicule {
    private $_marque;
    private $_modele;

    public function __construct($marque, $modele) {
        $this->_marque = $marque;
        $this->_modele = $modele;
    }

    public function afficherDetails() {
        echo "Marque: " . $this->_marque . ", Modèle: " . $this->_modele;
    }
}
```

2.2. Créer la classe Enfant

Ensuite, définissez une classe enfant qui hérite de la classe parent à l'aide du mot-clé **extends**.

```
class Voiture extends Vehicule {
    private $_nombreDePortes;

    public function __construct($marque, $modele, $nombreDePortes) {
        parent::__construct($marque, $modele); // Appel du constructeur de la
classe parent
        $this->_nombreDePortes = $nombreDePortes;
    }

    public function afficherDetailsVoiture() {
        // Appel de la méthode de la classe parent
        parent::afficherDetails();
        echo ", Nombre de portes: " . $this->_nombreDePortes;
    }
}
```

2.3. Utiliser l'héritage

Vous pouvez maintenant créer des instances de la classe enfant et utiliser à la fois ses méthodes et celles héritées de la classe parent.

```
$maVoiture = new Voiture("Toyota", "Corolla", 4);

$maVoiture->afficherDetails(); // Affiche "Marque: Toyota, Modèle: Corolla

$maVoiture->afficherDetailsVoiture(); // Affiche "Marque: Toyota, Modèle: Corolla, Nombre de portes: 4"
```

3. Bon à savoir

Quelques astuces supplémentaires !

- **Surcharge** (Overriding) : Une classe enfant peut surcharger les méthodes de la classe parent pour modifier ou étendre leur comportement.
- **Visibilité** : Utilisez les mots-clés public, protected, et private pour contrôler l'accès aux propriétés et méthodes dans le contexte de l'héritage.
- **Méthodes et propriétés statiques** : Les méthodes et propriétés statiques peuvent également être héritées.
- **Instanceof** : Utilisez l'opérateur instanceof pour vérifier si un objet est une instance d'une classe spécifique ou de ses classes parent.

4. Bonnes Pratiques

L'héritage en programmation orientée objet permet à une classe de reprendre (ou d'"hériter") des propriétés et méthodes d'une autre classe. En PHP, l'héritage est un outil puissant qui aide à organiser et structurer le code de manière logique, favorisant la réutilisation et la maintenance. Voici quelques bonnes pratiques à suivre lors de l'utilisation de l'héritage en PHP :

- **Principe de Responsabilité Unique :** Chaque classe doit avoir une seule raison de changer, ce qui signifie qu'elle doit avoir une seule tâche ou responsabilité. Lors de la conception de vos classes, assurez-vous que l'héritage est utilisé pour étendre la fonctionnalité d'une classe de manière logique et cohérente.
- **Préférer la Composition à l'Héritage :** Parfois, il est plus logique d'utiliser la composition plutôt que l'héritage. Si une classe a besoin de certaines fonctionnalités d'une autre classe, envisagez de l'inclure comme propriété plutôt que d'hériter de toute la classe. Cela augmente la modularité et la flexibilité du code.
- **Utiliser les Modificateurs d'Accès :** Protégez les propriétés et les méthodes en utilisant les modificateurs d'accès `private` et `protected` pour empêcher leur accès direct de l'extérieur de la classe ou en dehors des classes enfants. Utilisez `public` uniquement lorsque l'accès externe est nécessaire.
- **Surcharge de Méthodes** : Lorsque vous redéfinissez (ou "surchargez") une méthode dans une classe enfant, assurez-vous que son comportement reste cohérent avec celui de la classe parent. Le principe de substitution de Liskov doit être respecté, ce qui signifie qu'une instance de la classe parent doit pouvoir être remplacée par une instance de la classe enfant sans affecter le bon fonctionnement du programme.

- **Chainage des Constructeurs :** Lorsque vous définissez des constructeurs dans les classes enfants, n'oubliez pas d'appeler le constructeur de la classe parent (si nécessaire) en utilisant `parent::__construct()`. Cela garantit que toute l'initialisation nécessaire dans la classe parent est également effectuée pour la classe enfant.
- **Éviter l'Héritage Multiple :** PHP ne supporte pas l'héritage multiple (une classe ne peut pas hériter directement de plusieurs classes). Pour obtenir une fonctionnalité similaire, utilisez les traits ou l'implémentation d'interfaces.
- **Documentation :** Documentez clairement les relations d'héritage et expliquez pourquoi l'héritage est utilisé. Cela aidera les autres développeurs (ou vous-même ultérieurement) à comprendre rapidement la structure et le but de votre code.

En suivant ces bonnes pratiques, vous pouvez vous assurer que votre utilisation de l'héritage en PHP est efficace, maintenable et logique, contribuant à la clarté et à la qualité globale de votre code.