Fiche de révision

Les Associations en PHP

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

Une association en langage objet est un concept clé de la programmation orientée objet (POO) qui décrit une relation entre deux ou plusieurs classes. Elle indique que les entités (ou objets) de ces classes ont une relation ou une interaction les unes avec les autres d'une manière spécifique.

Voici les points principaux pour comprendre les associations en POO :

- **Relation Binaire ou N-aire** : Une association peut être binaire (entre deux classes) ou n-aire (impliquant plus de deux classes). Elle illustre comment les instances de ces classes peuvent être connectées.
- **Multiplicité** : La multiplicité spécifie le nombre d'instances d'une classe qui peuvent être associées à une seule instance de l'autre classe. Par exemple, "un à un", "un à plusieurs", "plusieurs à un", ou "plusieurs à plusieurs".
- **Navigation** : L'association peut être unidirectionnelle ou bidirectionnelle. Si elle est unidirectionnelle, vous pouvez naviguer d'une classe à l'autre, mais pas forcément dans le sens inverse. Si elle est bidirectionnelle, les deux classes peuvent se référer l'une à l'autre.
- **Rôle** : Les rôles dans une association décrivent la fonction d'une classe dans une relation. Par exemple, dans une association entre une classe `Professeur` et une classe `Cours`, le professeur "enseigne" le cours et le cours est "enseigné par" le professeur.
- **Agrégation et Composition** : Ce sont des types spéciaux d'associations. L'agrégation représente une relation "tout/partie" où les parties peuvent exister indépendamment du tout. La composition est une forme plus forte d'agrégation où les parties ne peuvent pas exister sans le tout.

En implémentant des associations dans la programmation orientée objet, les développeurs peuvent concevoir des applications qui reflètent de manière plus précise les interactions et les relations complexes du monde réel. Les associations aident à structurer le code de manière logique, facilitant ainsi la maintenance et l'extension des applications.

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight." – Bill Gates

2. Comment faire ?

L'association en programmation orientée objet (POO) est un terme qui décrit une relation entre deux classes où une instance d'une classe utilise ou interagit avec une instance d'une autre classe.

2.1. Définir les classes

Commencez par définir les deux classes qui seront associées. Par exemple, si nous avons une classe Auteur et une classe Livre, nous définirons ces deux classes avec leurs propriétés et méthodes respectives.

Définissons la classe « Auteur »

```
class Auteur {
    private string $_nom;
    private string $_prenom;

    public function __construct(string $nom, string $prenom) {
        $this->_nom = $nom;
        $this->_prenom = $prenom;
    }
  public function getNom():string
    {
        return $this->_nom;
    }
    public function gePrenom():string
    {
        return $this->_prenom;
    }
}
```

Ensuite nous pouvons définir la classe « Livre »

```
class Livre {
    private string $_titre;

    public function __construct(string $titre) {
        $this->_titre = $titre;
    }
    public function getTitre():string
    {
```

```
        return $this->_titre;
    }
}
```

Ici, il n'y a pas encore de relation mais comme chacun sait, tout livre à un auteur. Nous simplifions ici en considérant qu'il n'y a qu'un seul auteur.

2.2. Implémenter l'association

Nous avons décidé qu'il n'y a qu'un auteur par livre. Modifions le code pour réaliser l'association.

```
class Livre {
    private string $_titre;
    private Auteur $_auteur;

    public function __construct(string $titre, Auteur $auteur) {
        $this->_titre = $titre;
        $this->_auteur = $auteur; // L'association est faite ici
    }
    public function getTitre():string
    {
        return $this->_titre;
    }
    public function getAuteur():Auteur
    {
        return $this->_auteur;
    }

}
```

2.3. Créer des instances

Dans cet exemple, chaque Livre est associé à un Auteur. L'association est représentée par la propriété $_auteur dans la classe Livre, qui contiendra une instance de la classe Auteur.

```
$auteur = new Auteur("Hugo", "Victor");
$livre = new Livre("Les Misérables", $auteur);
```

2.4. Utiliser l'association

Une fois l'association établie, vous pouvez accéder aux propriétés et méthodes des objets associés en utilisant ces références.

Par exemple, pour afficher le nom de l'auteur d'un livre :

```
echo $livre->getAuteur()->getPrenom(); // Affiche "Victor"
```

3. Naviguer dans l'autre sens ?

Les différents auteurs peuvent avoir écrit des livres. Avec le code déjà réalisé, vous ne pouvez pas accéder depuis un auteur aux livres écrit.

3.1. Implémenter l'association

Il faut ajouter un attribut privé pour contenir la liste de livre. Par facilité, vous pouvez utiliser la classe **ArrayObject**. C'est un tableau qui est géré sous forme d'un objet, cela permet de rester dans le paradigme Objet. D'autres structures peuvent être mobilisées ou créées.

```
class Auteur {
    private string $_nom;
    private string $_prenom;
    private ArrayObject $_livres;

    public function __construct(string $nom, string $prenom) {
        $this->_nom = $nom;
        $this->_prenom = $prenom;
        $this->_livres = new ArrayObject();
    }
  public function getNom():string
    {
        return $this->_nom;
    }
    public function gePrenom():string
    {
        return $this->_prenom;
    }
}
```

3.2. Mettre en place les getters et les setters

Le tableau de livre étant privé, il faut mettre en place les méthodes pour ajouter un livre et lire la liste de livres dans la classe Auteur.

```
    public function ajouterLivre(Livre $livre):void
    {
        $this->_livres->append($livre);
    }
    public function getLivres():ArrayObject
    {
        return $this->_livres;
    }
```

On pourrait ajouter une méthode pour supprimer un livre de la liste également.

4. Bonnes Pratiques

Les associations en programmation orientée objet (POO) en PHP sont cruciales pour représenter les relations entre différentes entités ou classes. Voici quelques bonnes pratiques pour gérer les associations en PHP objet :

- **Choisir le Type d'Association Approprié** : Comprenez la nature de la relation entre les objets avant de décider du type d'association (un-à-un, un-à-plusieurs, plusieurs-à-plusieurs). Utilisez des objets ou des tableaux pour représenter ces relations de manière appropriée dans votre code.
- **Utiliser des Accesseurs et des Mutateurs** : Encapsulez l'accès aux propriétés associées via des méthodes d'accès (getters) et de modification (setters). Cela permet de contrôler et de valider les modifications apportées aux associations.
- **Gérer la Cohérence des Données** : Assurez-vous que les modifications apportées à un côté de l'association sont reflétées de l'autre côté. Par exemple, si vous supprimez un objet dans une relation un-à-plusieurs, assurez-vous de mettre à jour les références dans les objets associés.
- **Utiliser des Constructeurs pour Initialiser les Associations** : Les constructeurs peuvent être utilisés pour initialiser les objets associés, ce qui peut garantir que les instances de vos classes commencent leur vie dans un état cohérent.
- **Lazy Loading** : Pour les associations complexes ou les relations un-à-plusieurs/plusieurs-à-plusieurs, envisagez d'utiliser le chargement paresseux (lazy loading) pour ne charger les objets associés que lorsqu'ils sont réellement nécessaires.
- **Éviter les Dépendances Circulaires** : Faites attention à ne pas créer de dépendances circulaires où deux objets sont mutuellement dépendants. Cela peut conduire à des difficultés lors de la destruction des objets et à des fuites de mémoire.
- **Documenter les Associations** : Documentez clairement les associations dans le code, en particulier dans les classes où les relations ne sont pas évidentes. Les commentaires peuvent aider les autres développeurs à comprendre le modèle d'objet et les relations entre les classes.
- **Utiliser des Identifiants pour Représenter les Associations** : Dans certains cas, en particulier lors du travail avec des bases de données, il est pratique d'utiliser des identifiants (clés étrangères) pour représenter les associations plutôt que des références d'objet directes.
- **Préférer la Composition à l'Héritage** : Lorsque c'est possible, utilisez la composition plutôt que l'héritage pour réutiliser le code entre les classes, ce qui augmente la flexibilité et la réutilisabilité de votre code.
- **Tester les Associations** : Assurez-vous de tester les associations entre vos classes, en particulier la logique qui gère la création, la modification et la suppression des relations.

En suivant ces bonnes pratiques, vous pouvez vous assurer que les associations dans vos applications PHP sont robustes, maintenables et compréhensibles.