Programmez en orienté objet en PHP

Introduction à la Prog Orienté Objet (POO) en PHP

1. Pourquoi la POO ?
    - Prob avec le code procédural : Ajoutez du code en augmentant sans fin, qui peut mener à un code difficile à maintenir
    - Solution : La POO aide à structurer le code de manière organisé et évolutive, comme la construction d'un pont solide plutôt que de simplement empiler les briques

2. Choix de la structure :
    - Comme pour un pont, il est crucial de bien choisir la structure en POO. trop petit, et vous devez tous refaire. Trop grand, et vous pourriez gaspiller des ressources ou avoir des probs de maintenance.

3. En résumé :
    - **Avantages de la POO** : Structure le code de manière organisée et évolutive, facilitant la maintenance et l'extension des projets.
    - Importance d'une **structure appropriée** : Choisir la bonne structure est crucial pour la pérennité et la facilité d'entretien du code.
    - La POO comme solution aux limitations du code procédural, offrant une meilleure organisation et évolutivité.
    - **Concepts fondamentaux** : Objets (instances de classes) et Classes (modèles pour créer des objets), regroupant propriétés et méthodes.
    - L'instanciation comme processus de création d'objets à partir de classes définies.

Découverte des Objets et Classe en PHP

1. Objet et Classes :
   - Objet = Instance d'une classe. Regroupe des propriétés (données) et des méthodes (fonctions)
   - Classes = Modèle pour créer des objets, Définit les propriétés et les méthodes dispo.

2. Instanciation :
   - Créer un objet : Utilise le mot-clé "new" pour créer une instance d'une classe.

     Exemple :

     ```
     $date = new DateTime();
     ```

   - Instance : Un objet créé à partir d'une classe. Chaque instance est un objet même si elles sont issues de la même classe.

3. Référence et Modif :
   - Référence : les Objets PHP sont passés par référence, ce qui signifie que toutes les modifs de l'objet affecte l'original.

     Exemple :

     ```
     $date->modify('+1 day'); //Ici on modifie l'objet Date
     ```

   - Passage par référence : pour modifier une variable dans une fonction, utilisez le symbole '&'

     exemple :

     ```
     function foo(&$var) {
         $var = 2;
     }
     ```

   - Comparaison : Utilisez ' ==' pour comparer les valeurs et '===' pour comparer les instances (si elles sont le même objet en mémoire)

4. Accès aux Propriétés et Méthode $

- Accès : Utilisez ‘ - >’ pour accéder aux propriétés et méthodes d’un objet.

  Exemple :

  ```
  $date->format('d/m/Y');
  ```

- Chaînage (Chaining) : Appel plusieurs méthodes successivement sur le même objet

  Exemple :

  ```
  $formatedDate = $date->modify('+1 day')->format('d/m/Y'
  ```

5. stdClass et JSON

- stadClass : Classe générique utilisé par PHP pour les objets JSON non typé.

  Exemple :

  ```
  $s = '{"date":"2021-03-23","timezone":"Europe/Paris"}';
  $obj = json_decode($s);
  ```

- JSON Décodé : PHP utilise stdClass lorsque la classe spécifique n’est pas définie.

6. Méthode Retour :

- Retour d’Objet : Les méthodes peuvent retourner l’instance actuelle ou une nouvelle instance, en facilitant le chainage.

  Exemple :

  ```
  $newDate = $date->modify('+1 day'); // $newDate est l'o
  ```

Résumé :

- **Classe :** Modèle avec propriétés et méthodes.
- **Objet :** Instance d'une classe, créée avec `new`.
- **Instance :** Objet spécifique créé à partir d'une classe.
- **Accès et Manipulation :** Utilisez `>` pour accéder aux propriétés et méthodes. Les objets sont passés par référence.
- **Chaînage :** Appeler plusieurs méthodes sur le même objet en série.
- **stdClass :** Utilisée pour les objets JSON sans classe définie.

Création de nos classes en PHP

1. Déclaration de classe :
   - syntaxe de base :

```
<?php
class NomDeLaClasse {
    // Propriétés et méthodes ici
}
```

   - Mot-clé 'class' suivi du nom de la classe en PascalCase, et une paire d'accolade où l'on définira les propriétés et méthodes.
   - Nom pertinent : Le nom de la classe doit être descriptif pour faciliter la compréhension du code, surtout dans des projets complexes.
2. Instanciation de Classes :
   - Instanciation :

```
$pont = new Pont;
```

   - Mot clé 'new' permet de créer une instance de classe, c.-à-d. un objet qui pourra utiliser les propriétés et les méthodes définies dans la classe.
3. Ajout de propriétés :
   - Déclaration et Typage :

```
declare(strict_types=1); // En haut du fichier pour un typ

class Pont {
    public float $longueur = 0;
}
```

- Propriétés : Ce sont des variables appartenant a la classe, définies avec ‘public’, un type (‘float’), et un nom précédé du symbole ‘$’
- Typage strict : Avec ‘declare(strict_types=1);’, PHP impose un typage strict, réduisant le risque d’erreurs dues à des types inatendue.

4. Utilisation des propriétés
    - Accès et Modif :

```
$pont->longueur = 263.0;
var_dump($pont);
```

    - Propriétés d’un objet sont accessible et modifiable via l’opérateur ‘→’. Il est possible de définir des valeurs spécifiques pour chaque instance de la classe.
5. Conversion Automatique de types :

```
// declare(strict_types=1); // Commenté

$pont->longueur = '263.0'; // PHP convertira automatiqueme
```

    - Sl ‘declare(strict_types=1);’ est désactiver, PHP tente de convertir les types automatiquement, ce qui peut mener à des résultats inattendu.
6. Ajout de Méthodes :
    - Déclaration de méthodes :

```
class Pont {
    public float $longueur;
    public float $largeur;

    public function getSurface(): float {
```

```
        return $this->longueur * $this->largeur;
    }
}
```

- Méthode : fonction définie dans une classe, qui permet d'effectuer des actions spécifiques en utilisant les propriétés de l'objet.
- Visibilité 'public' : Indique que la méthode est accessible en dehors de la classe. 'getSurface' calcule et renvoie la surface du pont.
- Utilisation de '$this' : '$this' fait référence à l'instance actuelle de la classe, permettant d'accéder à ses propriétés

7. Utilisation de méthodes :
    - Appel de Méthode :

```
$surface = $pont->getSurface();
var_dump($surface);
```

    - Une méthode est appelée sur l'objet via opérateur '→', sui vie du nom de la méthode et des parenthèses.
8. Gestion de multiple instance :
    - Exemple avec plusieurs instances :

```
$towerBridge = new Pont;
$towerBridge->longueur = 286.0;
$towerBridge->largeur = 15.0;

$manhattanBridge = new Pont;
$manhattanBridge->longueur = 2089.0;
$manhattanBridge->largeur = 36.6;

$towerBridgeSurface = $towerBridge->getSurface();
$manhattanBridgeSurface = $manhattanBridge->getSurface();

var_dump($towerBridgeSurface, $manhattanBridgeSurface);
```

    - Plusieurs Instances : Chaque instance d'une classe peut avoir des valeurs de propriétés différentes, ce qui permet de manipuler plusieurs

objets similaires de manières indépendantes.

Concevez l'API publique de vos objets

1. Définissez le domaine public de vos classes :
   - Principe de base :
     - Toutes méthodes et propriétés d'une classe que vous souhaitez rendre accessible depuis l'extérieur de la classe doivent être préfixées avec le mot-clé 'public'.
     - Exemple :

```
class Pont {
    public float $longueur;
    public float $largeur;
    public function getSurface(): float {
        return $this->longueur * $this->largeur;
    }
}
```

     - Si une méthode ou une propriété n'est pas censée être utilisée à l'extérieur de la classe, elle doit être déclarée avec le mot clé 'private'.
   - Illustration avec une Propriété Privée (Private) :
     - Exemple où l'on souhaite renvoyer la surface avec une unité (ex : m²), en utilisant une Propriété Privée :

```
class Pont {
    private string $unite = 'm²';
    public float $longueur;
    public float $largeur;

    public function getSurface(): string {
        return ($this->longueur * $this->largeur) . $th
    }
}
```

- Résultat : La propriété ‘$unite’ est privée et inaccessible depuis l’extérieur, simplifiant l’usage de la classe pour les autres développeurs.

2. Masquez la complexité de nos Objets :
   - Encapsulation des propriétés :
     - Encapsulation consiste à protéger en les rendant privées, empêchant ainsi toute modif directe depuis l’extérieur.
     - Exemple :

```
class Pont {
    private string $unite = 'm²';
    private float $longueur;
    private float $largeur;
}
```

   - Problème : Si vous essayez d’accéder à une propriété privée depuis l’extérieur, PHP renverra une erreur fatale, arrêtant l’exécution du script.
3. Accédez aux Propriétés Privées avec des Getters et Setters :
   - Création de méthode publiques :
     - Pour accéder et modifier des propriétés privées, on utilise des méthodes appelées ‘setters’ (ou ‘mutateurs’) et ‘getters’ (ou ‘accesseurs’).
     - Exemple Setters :

```
class Pont {
    private float $longueur;
    private float $largeur;

    public function setLongueur(float $longueur): void
        if ($longueur < 0) {
            trigger_error('La longueur est trop courte.
        }
        $this->longueur = $longueur;
```

```
    }
}
```

- Exemple de Getters :

```
class Pont {
    private float $longueur;

    public function getLongueur(): float {
        return $this->longueur;
    }
}
```

- Avantages : ‘setters’ permettent de contrôler la valeur avant de l’assigner à la propriété (ex : vérifie que la longueur n'est pas négative). ‘Getters’ permettent d’accéder à la valeur encapsulée.

En résumé :

- *Simplifie l’utilisation* : en privatisant méthode et propriété non essentielles, on simplifie l’utilisation des classes pour les autres devs. Limite également risque d’erreur.
- *Encapsulation* : permet via ‘getters’ et ‘setters’ de contrôler l’accès aux données sensibles, garantissant ainsi la cohérence des valeurs dans les objets.

Utilisez les Propriétés et Méthodes Statiques en PHP

1. Propriété et Méthodes statiques
   - contexte :
     - Propriétés et Méthode vue jusqu’ici était liée à des objets spécifiques. Propriété et Méthode statique sont associées à la classe elle-même et non à ses instances.
   - Concept de base :
     - Une méthode statique peut être appeler sans qu’une instance de la classe soit créée. Les propriétés statiques, elles, partagent leurs valeurs entre toutes les instances de la classe.

2. Déclaration et utilisation des méthodes statique :
    - Déclaration :
        - Avec le mot clé ‘static’

```
class Pont {
    public static function validerTaille(float $taille)
        if ($taille < 50.0) {
            trigger_error('La longueur est trop courte.
        }
        return true;
    }
}

var_dump(Pont::validerTaille(150.0)); // Renvoie true
var_dump(Pont::validerTaille(20.0));  // Provoque une e
```

    - Accès aux méthodes static :
        - Méthode Static appeler via ‘: :’ (ex : (Pont::validerTaille(150.0)), car elles sont liées à la classe et non à une instance.

```
class Pont {
    public static function validerTaille(float $taille)
        if ($taille < 50.0) {
            trigger_error('La taille est trop courte. (
        }
        return true;
    }

    public function setLongueur(float $longueur): void
        self::validerTaille($longueur);
        $this->longueur = $longueur;
    }
}
```

3. Propriété statique :

- Définition et utilisation : Est partagé entre toutes les instances d'une classe. À pour mot clé 'static'.

```
class Pont {
    public static int $nombrePietons = 0;

    public function nouveauPieton() {
        self::$nombrePietons++;
    }
}

$pontLondres = new Pont;
$pontLondres->nouveauPieton();

$pontManhattan = new Pont;
$pontManhattan->nouveauPieton();

echo Pont::$nombrePietons; // Affiche 2
```

- Fonctionnement : '$nombrePietons' est partagée par toutes les instances de la classe 'Pont'. Chaque appel à '$nouveauPieton()' incrémente la même valeur quel que soit l'instance utilisée.

4. Propriétés Immuables avec les Constantes de Classe :
    - Déclaration : Une propriété qui ne change jamais de valeur doit être déclaré comme constante avec mot clé 'const'

```
class Pont {
    private const UNITE = 'm²';
    private float $longueur;
    private float $largeur;

    public function getSurface(): string {
        return ($this->longueur * $this->largeur) . self::
    }
}
```

    - Particularité constante :

    - Ne sont pas préfixés par '$' et sont ecrite 'UPPER_SNAKE_CASE'.
    - Accessible via 'self::CONSTANTE_NAME'
- utilisation des constantes : permettent de centraliser les valeurs immuables (ex : unité de mesure, coef...) au début de la classe, facilitant ainsi la gestion et la modif des configs de classe.

Exploitez les Méthodes Communes à Tous les Objets en PHP

1. Méthode magique en PHP :
    - Appeler a différent moment du cycle de vie d'un objet. Sont Identifiables par leurs préfixes '__'(_ x2). Pas besoin de les appeler explicitement, PHP le fait en arrière-plan.
    - Permettent intervenir dans des processus tel que la création, la modif et destruction d'objet. Essentielles pour personnaliser le comportement de vos classes.
2. Constructeur ('__construct') :
    - Fonction : Est appelé automatiquement lors de la création d'une nouvelle instance de classe. Sert principalement à initier les propriétés de l'objet.
    - Utilisation classique :

```
class Pont {
    private float $longueur;
    private float $largeur;

    public function __construct(float $longueur, float $la
        $this->longueur = $longueur;
        $this->largeur = $largeur;
    }
}

$towerBridge = new Pont(286.0, 62.0);
```

    Ici valeur de Long et Large sont définis dès la création de l'objet 'Pont'.

    - Forme courte (PHP 8+)

```
class Pont {
    public function __construct(private float $longueur, p
    }
}

$towerBridge = new Pont(286.0 , 62.0);
var_dump($towerBridge); //Affiche propriété Longueur et La
```

PHP 8 permet une syntaxe plus concise où les propriétés sont définies directement dans les paramètres du constructeur.

3. Destructeur (’__destruct’) :
    - Automatiquement appeler lorsqu’un objet est détruit, c.-à-d. Lorsqu’il est plus référencé ou a la fin du script. Cette méthode est utile pour libérer des ressources, fermer des connexions ou effectuer d’autre action de nettoyage.

```
class Pont {
public function __destruct() {
echo "L'objet Pont est détruit";
    }
}

$towerBridge = new Pont();
unset($towerBridge);
```

Ici, `__destruct` est appelé explicitement en détruisant l'objet avec `unset`.

4. Autre Méthode magique importante :
    - ‘__clone’ : Appelé lorsqu’un objet est cloné
    - ‘__toString’ : Définit objet sous forme de chaine de caractère
    - ‘__invoke’ : permet de rendre Objet appelable comme une Function.
    - ‘__get’ et ‘__set’ : gerent acces au propriété non définie ou innaccessible.

- ‘__isset’ et ‘__unset’ : gèrent verif et suppressions de propriétés non définie.
- ‘__call’ : Intercepte des appels à des méthodes inaccessible ou non définie.
- ‘__serialize’ et ‘__unserialize’ : personalise la sérialisation et désérialisation d’un objet.

5. Conclusion et Pratique

Les méthodes magiques de PHP sont puissantes et vous permettent de contrôler finement le comportement de vos objets. Toutefois, elles doivent être utilisées avec précaution pour éviter de rendre votre code complexe ou difficile à comprendre.

Conseils :

- **Expérimentez :** Mettez en pratique ces méthodes en créant vos propres classes et observez comment PHP les gère automatiquement.
- **Testez :** Explorez les limites et les possibilités offertes par ces méthodes pour mieux les comprendre.