La programmation par objet (POO) en PHP 7 - 8

Partie 2

Visibilité des propriétés et des méthodes

La visibilité des propriétés et des méthodes d'un objet constitue une des particularités élémentaires de la programmation orientée objet. Ce tutoriel a pour objectif de présenter les différents niveaux de visibilité que propose le modèle objet de PHP 8. Nous les passerons en revue un par un au travers d'exemples pratiques et nous apporterons quelques bonnes pratiques à adopter lorsqu'on les utilise.

Qu'est-ce que la visibilité des propriétés et méthodes d'un objet ?

La visibilité permet de définir de quelle manière un attribut ou une méthode d'un objet sera accessible dans le programme. Comme Java, C++ ou bien ActionScript 3, PHP introduit 3 niveaux différents de visibilité applicables aux propriétés ou méthodes de l'objet.

Il s'agit des visibilités publiques, privées ou protégées qui sont respectivement définies dans la classe au moyen des mots-clés public, private et protected.

L'exemple suivant illustre la syntaxe de déclaration de données membres (attributs) et de méthodes ayant des visibilités différentes. Nous expliquerons juste après chaque particularité des niveaux de visibilité.

Utilisation des mots-clés public, private et protected

```
<?php
class Vehicule
{
  // Attributs
  public $marque;
  private $_volumeCarburant;
  protected $_estRepare;
  // Méthodes
  public function __construct()
  {
    $this->_volumeCarburant = 40;
    $this->_estRepare = false;
  }
  // Démarre la voiture si le réservoir
  // n'est pas vide
  public function demarrer()
  {
    if ($this->_controlerVolumeCarburant())
    {
   echo 'Le véhicule démarre';
      return true;
    }
  echo 'Le réservoir est vide...';
```

```
    return false;
  }
  // Vérifie s'il y'a du carburant dans le réservoir
  private function _controlerVolumeCarburant()
  {
    return ($this->_volumeCarburant > 0); // renvoi true ou false
  }
  // Met le véhicule en maintenance
  protected function reparer()
  {
    $this->_estRepare = true;
  echo 'Le véhicule est en réparation';
  }
}
```

Nous remarquons ici que nous utilisions déjà les attributs et méthodes publiques dans les exemples du premier tutoriel sur la programmation orientée objet. Présentons à présent chacun des niveaux de visibilité.

L'accès public

C'est l'accès par défaut de PHP si l'on ne précise aucune visibilité. Tous les attributs et méthodes qui sont déclarés sans l'un de ces trois mots-clés sera considéré automatiquement par l'interpréteur comme publique.

Le mot-clé public indique que les propriétés et méthodes d'un objet seront accessibles depuis n'importe où dans le programme principal ou bien dans les classes mères héritées ou classes filles dérivées. Retenez ces termes dans un coin de votre mémoire car nous présenterons les notions d'héritage au prochain tutoriel.

Concrètement, c'est ce que nous faisions dans le tutoriel précédent. Ce qui donne avec notre exemple :

Utilisation de la visibilité publique

```
<?php
  // Instanciation de l'objet : appel implicite à la méthode
__construct()
  $monVehicule = new Vehicule();
  // Mise à jour de la marque du véhicule
  $monVehicule->marque = 'Peugeot';
  // Affichage de la marque du véhicule
 echo $monVehicule->marque;
?>
```

Dans cet exemple, nous remarquons que nous pouvons lire et modifier directement la valeur de l'attribut publique marque en l'appelant directement de cette manière : $monVehicule->marque(sans le dollars).

Nous verrons juste après que cette même utilisation sera impossible avec les attributs privés et protégés.

Note : lorsque l'on instancie la classe pour créer un nouvel objet, le mot-clé "new" se charge d'appeler la méthode constructeur de la classe. Cette dernière doit donc obligatoirement être déclarée publique car elle est appelée depuis l'extérieur de la classe.

L'accès private

Le mot-clé private permet de déclarer des attributs et des méthodes qui ne seront visibles et accessibles directement que depuis l'intérieur même de la classe. C'est à dire qu'il devient impossible de lire ou d'écrire la valeur d'un attribut privé directement en faisant $monVehicule->_volumeCarburant. De même pour utiliser la méthode _controlerVolumeCarburant() via$monVehicule->_controlerVolumeCarburant(). V ous remarquerez en testant, qu'une erreur fatale est générée à l'exécution.

La valeur de l'attribut privé $_volumeCarburant est initialisée dans le constructeur de la classe. C'est-à-dire quand on construit l'objet.

Par exemple, lorsque notre véhicule est enfin assemblé et prêt à être utilisé, il sera vendu au client avec le réservoir d'essence rempli,soit 40L.

Une méthode est déclarée private lorsqu'elle n'a pas vocation à être utilisée en dehors de la classe. Les méthodes privées servent notamment à valider des données dans un objet ou bien à effectuer différents traitements internes. C'est par exemple la tâche que remplit la méthode privée_controlerVolumeCarburant() lorsqu'elle est appellée par la méthode publique demarrer(). Cette méthode privée vérifie qu'il y'a bien du carburant dans le véhicule en testant la valeur de l'attribut privé $_volumeCarburant. Si tel est le cas, la méthode retourne TRUE et donc la voiture démarre. Dans le cas contraire, la valeur retournée est FALSE et la voiture ne peut démarrer.

Appel de la méthode demarrer() sur l'objet $monVehicule

```
<?php
  // Affiche : "Le véhicule démarre"
  $monVehicule->demarrer();
?>
```

Note : par convention, on déclare les attributs privés et protégés avec un underscore afin de les identifier plus facilement lorsque l'on relit le code. Cette règle d'usage n'est pas suivie par tous les développeurs mais nous vous recommandons vivement de l'adopter dans vos développements.

L'accès protected

L'accès protégé (protected) est un intermédiaire entre l'accès public et l'accès privé. Il permet d'utiliser des attributs et méthodes communs dans une classe parente et ses classes dérivées (héritantes).

Le chapitre concernant l'héritage n'est pas au programme de ce tutoriel. Nous l'étudierons dans le prochain cours. Toutefois, nous allons devoir l'anticiper afin de comprendre comment fonctionne l'accès protégé. Pour cela, nous allons ajouter une classe Voiture qui héritera de notre classe actuelleVéhicule. Comme une voiture est une sorte de véhicule, il nous est donc possible de dériver la classeVéhicule avec une classe Voiture.

La classe Vehicule contiendra les attributs et méthodes communs à tout type de véhicule tandis que la classe Voiture encapsulera les propriétés et méthodes propres aux voitures. Par exemple, lamarque ou le booléen estRepare peuvent très bien être des attributs

partagés. Par contre, le volume de carburant dans le réservoir n'est pas commun à tous les véhicules.

En effet, un vélo, une trottinette ou bien encore un aviron peuvent être considérés comme des véhicules non motorisés. Ils n'ont donc pas de réservoir et ne fonctionnent pas avec du carburant.

Utilisation des accès protégés

```
<?php
class Vehicule
{
  // Attributs
  protected $_marque;
  protected $_estRepare;
  // Méthodes
  public function __construct($marque)
  {
    $this->_marque = $marque;
    $this->_estRepare = false;
  }
  // Met le véhicule en maintenance
  public function reparer()
  {
    $this->_estRepare = true;
  echo 'Le véhicule est en réparation';
  }
}
class Voiture extends Vehicule
{
  // Attributs
  private $_volumeCarburant;
  // Constructeur
  public function __construct($marque)
  {
    // Appel du constructeur de la classe parente
    parent::__construct($marque);
    $this->_volumeCarburant = 40;
  }
  // Démarre la voiture si le réservoir
  // n'est pas vide
  public function demarrer()
  {
    if ($this->_controlerVolumeCarburant())
    {
   echo 'Le véhicule démarre';
      return true;
    }
  echo 'Le réservoir est vide...';
    return false;
  }
  // Vérifie qu'il y'a du carburant dans le réservoir
  private function _controlerVolumeCarburant()
  {
    return ($this->_volumeCarburant > 0);
```

```
  }
}
```

Nous venons de déclarer deux attributs protégés dans la classe parente Vehicule et nous avons redescendu l'attribut _volumeCarburant dans la classe Voiture. De même nous avons redescendu les méthodes demarrer() et _controlerVolumeCarburant() dans la classe Voiture. Ainsi, nous pouvons désormais instancier un nouvel objet de type Voiture et profiter des attributs et des méthodes de la classe Vehicule en plus. Enfin, notez que la méthode reparer() est passée du modeprotected à public afin de pouvoir l'appeler depuis l'extérieur de la classe. Ce qui donne :

Utilisation des attributs et méthodes protégés

```
<?php
  $monVehicule = new Voiture('Peugeot');
  $monVehicule->demarrer();
  $monVehicule->reparer();
?>
```

Mise à jour d'un attribut privé ou protégé : rôle du mutator

Nous avons évoqué plus haut qu'il était impossible d'accéder à la valeur d'un attribut privé ou protégé en utilisant la syntaxe suivante : $objet->attribut. Une erreur fatale est générée. Comment faire pour mettre à jour la valeur de l'attribut dans ce cas ? C'est là qu'intervient le mutator.

Le mutator n'est ni plus ni moins qu'une méthode publique à laquelle on passe en paramètre la nouvelle valeur à affecter à l'attribut. Par convention, on nomme ces méthodes avec le préfixe set. Par exemple : setMarque(), setVolumeCarburant()... C'est pourquoi on entendra plus souvent parler de setter que de mutator.

Ajoutons un mutator à notre classe Voiture qui permet de modifier la valeur du volume de carburant. Cela donne :

Ajout du mutator (setter) setVolumeCarburant à la classe Voiture

```
<?php
class Voiture extends Vehicule
{
  // ...
  public function setVolumeCarburant($dVolume)
  {
    $this->_volumeCarburant = $dVolume;
  }
}
```

Pour respecter les conventions de développement informatique, nous utilisons dans notre exemple l'écriture au moyen du préfixe set. Mais ce nom est complètement arbitraire et notre méthode aurait très bien pu s'appeler faireLePlein() ou remplirLeReservoir(). Il ne tient qu' à vous de donner des noms explicites à vos méthodes.

Quant à l'utilisation de cette méthode, il s'agit tout simplement d'un appel de méthode traditionnel.

Mise à jour de la valeur de l'attribut privé $_volumeCarburant

```php
<?php
  // Je remplis mon réservoir de 50 L d'essence
  $monVehicule->setVolumeCarburant(50);
?>
```

Obtenir la valeur d'un attribut privé ou protégé : rôle de l'accessor (getter)

Nous venons d'étudier comment mettre à jour (écriture) la valeur d'un attribut privé ou protégé. Il ne nous reste plus qu' à aborder le cas de la lecture de cette valeur. Comment restitue-t-on cette valeur dans un programme sachant que l'accès direct à l'attribut est interdit ? De la même manière que pour le mutator, nous devons déclarer une méthode qui va se charger de retourner la valeur de l'attribut. Ce n'est donc ni plus ni moins qu'une fonction avec une instruction return.

On appelle ce type de méthode un accessor ou bien plus communément un getter. En effet, par convention, ces méthodes sont préfixées du mot get qui se traduit en français par « obtenir, récupérer ». Au même titre que les mutators, il est possible d'appliquer n'importe quel nom à un accessor.

Reprenons notre exemple précédent. Le getter ci-après explique le principe de retour de la valeur d'un attribut privé. Nous renvoyons ici la valeur de la variable $_volumeCarburant.

Déclaration d'un accessor pour l'attribut privé $_volumeCarburant de la classe Voiture

```php
<?php
class Voiture extends Vehicule
{
  // ...
  public function getVolumeCarburant()
  {
    return $this->_volumeCarburant;
  }
}
```

Voici un exemple d'utilisation de la méthode getVolumeCarburant :

Utilisation de l'accessor getVolumeCarburant() sur l'objet $monVehicule

```php
<?php
echo sprintf("Il me reste %u L d'essence", $monVehicule->getVolumeCarburant());
?>
```

C'est tout ce qu'il faut savoir d'essentiel sur la visibilité des méthodes et des attributs ainsi que sur les accès aux valeurs de ces derniers. Avant de conclure ce tutoriel, faisons un court rappel de bonnes pratiques de développement orienté objet histoire que vous puissiez démarrer immédiatement avec des bases solides.

Quelques bonnes pratiques...

Voici une petite série récapitulative de conseils et de réflexes à mettre en œuvre en développement orienté objet afin d'uniformiser et de standardiser le code. Ces astuces syntaxiques ne sont pas des techniques personnelles puisque ce sont des conventions éprouvées par des spécialistes des langages orientés objet. Malgré tout, ne le prenez pas comme des paroles d'évangile. Vous êtes libres d'adopter les conventions et règles syntaxiques de votre choix.

- Préfixer les noms de méthodes et d'attributs privés avec un underscore afin de les distinguer plus rapidement à la relecture du code.
- Placer les attributs et méthodes en accès privé ou bien en accès protégé si l'on souhaite dériver la classe dans le futur.
- Utiliser autant que possible les conventions de nommage pour les accessors et les mutators (getNomAttribut() et setNomAttribut()).