Accesseurs, mutateurs et méthodes magiques

Les accesseurs et mutateurs

Nous avons vu dans la leçon précédente, que lorsqu'une propriété était privée, seule la classe pouvait la modifier.

Pour l'initialisation, aucun problème, nous avons le constructeur qui est une méthode, donc `public` par défaut et qui peut donc initialiser des propriétés privées :

```
<?php
class User {

  function __construct(private string $prenom, private string $nom) {}
}

$user1 = new User('Jean', 'Dupont');
```

Mais nous ne pouvons ni accéder aux propriétés, ni les modifier à l'extérieur de la classe.

Pourquoi dès lors, mettre des propriétés ou méthodes en visibilité privée ou protégée ? Il s'agit d'une indication à vous-même et aux autres développeurs qu'il faut modifier les objets d'une classe d'une certaine façon pour ne pas entraîner des bugs, par exemple à cause d'inconsistances.

Prenons l'exemple de la vidéo :

```
<?php
class User {

  public string $nomComplet;

  function __construct(public string $prenom, public string $nom) {
    $this-> nomComplet = $prenom . ' ' . $nom;
  }
}

$user1 = new User('Jean', 'Dupont');
```

```
echo $user1->nomComplet, '<br>'; // Jean Dupont
$user1->prenom = 'Paul';
echo $user1->nomComplet; // Jean Dupont
```

Ici nous avons créé une inconsistance. Le nom complet n'est plus correct, et si l'exemple est ici simple, il faut s'imaginer des classes plus complexes.

Nous allons pouvoir empêcher de nombreux bugs en utilisant des accesseurs et des mutateurs : ce sont des méthodes qui ont pour objectif d'accéder ou de modifier des propriétés privées proprement sans endommager le fonctionnement de la classe.

Voici l'exemple modifié :

```
<?php
class User {

  private string $nomComplet;

  function __construct(private string $prenom, private string $nom) {
    $this-> nomComplet = $prenom . ' ' . $nom;
  }

  function setPrenom(string $nouveauPrenom) {
    $this->prenom = $nouveauPrenom;
    $this->nomComplet = $nouveauPrenom . ' ' . $this->nom;
  }

  function setNom(string $nouveauNom) {
    $this->nom = $nouveauNom;
    $this->nomComplet = $this->prenom . ' ' . $nouveauNom;
  }

  function getPrenom() {
    return $this->prenom;
  }

  function getNom() {
    return $this->nom;
  }

  function getNomComplet() {
```

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

$user1 = new User('Jean', 'Dupont');
echo $user1->getNomComplet(), '<br>'; // Jean Dupont
$user1->setPrenom('Marie');
$user1->setNom('Dupuis');
echo $user1->getNomComplet(); // Marie Dupuis
```

Ici, nous empêchons toute inconsistance : grâce au mutateurs (`setters`) et aux accesseurs (`getters`), nous définissons exactement comment modifier les objets de notre classe sans compromettre son fonctionnement.

Utiliser ce fonctionnement revient à créer une interface, appelée `API` , pour manipuler vos classes.

Il ne faut pas forcément adopter ce `pattern` partout, il faut réfléchir lors de l'écriture de la classe : est-ce que la modification de propriétés non désirée peut "casser" ma classe ? Est-ce que certaines propriétés ne sont utiles que dans ma classe ?

Les méthodes magiques

Les méthodes magiques sont des méthodes spéciales qui permettent d'exécuter du code lorsque certaines actions sont réalisées sur un objet.

Nous en avons déjà vu deux : `__construct()` appelée lors de la création d'un objet, et `__destruct()` appelée lorsqu'il n'y a plus aucune référence à un objet.

Nous allons voir quelques autres méthodes magiques.

`__get()` est appelée pour lire des données depuis des propriétés protégées ou privées ou inexistantes :

```
<?php
class User {

  private string $nomComplet;

  function __construct(private string $prenom, private string $nom) {
    $this-> nomComplet = $prenom . ' ' . $nom;
  }
```

```
  function __get($prop) {
    if (isset($this->$prop)) {
      return $this->$prop;
    } else {
      echo "La propriété n'existe pas !";
    }
  }
}

$user1 = new User('Jean', 'Dupont');
echo $user1->nomComplet, '<br>'; // Jean Dupont
echo $user1->prenom, '<br>'; // Jean
echo $user1->nom, '<br>'; // Dupont
echo $user1->age, '<br>'; // La propriété n'existe pas !
```

C'est souvent utile pour ne pas avoir à écrire de nombreux accesseurs si vous mettez toutes les propriétés en visibilité privée.

`__set()` est appelée pour écrire des données depuis des propriétés protégées ou privées ou inexistantes :

```
<?php
class User {

  private string $nomComplet;

  function __construct(private string $prenom, private string $nom) {
    $this-> nomComplet = $prenom . ' ' . $nom;
  }

  function __get(string $prop) {
    if (isset($this->$prop)) {
      return $this->$prop;
    } else {
      echo "La propriété n'existe pas !";
    }
  }

  function __set(string $prop, string $val) {
    $this->$prop = $val;
```

```
    if ($prop === 'prenom' || $prop === 'nom') {
      $this->nomComplet = $this->prenom . ' ' . $this->nom;
    }
  }
}

$user1 = new User('Jean', 'Dupont');
echo $user1->nomComplet, '<br>'; // Jean Dupont
$user1->nom = 'Marie';
echo $user1->nomComplet, '<br>'; // Jean Marie
```

`__isset()` est appelée lorsque qu'on appelle la fonction `isset()` sur des propriétés protégées ou privées ou inexistantes :

```
<?php
class User {

  private string $nomComplet;

  function __construct(private string $prenom, private string $nom) {
    $this-> nomComplet = $prenom . ' ' . $nom;
  }

  function __isset($prop) {
    return isset($this->$prop);
  }

  function __get(string $prop) {
    if (isset($this->$prop)) {
      return $this->$prop;
    } else {
      echo "La propriété n'existe pas !";
    }
  }

  function __set(string $prop, string $val) {
    $this->$prop = $val;
    if ($prop === 'prenom' || $prop === 'nom') {
      $this->nomComplet = $this->prenom . ' ' . $this->nom;
    }
```

```
    }
}

$user1 = new User('Jean', 'Dupont');
echo isset($user1->nom), '<br>'; // 1
```

`__unset()` est appelée lorsque qu'on appelle la fonction `unset()` sur des propriétés protégées ou privées ou inexistantes.

`__toString()` détermine comment l'objet est affiché en chaîne de caractères. Par exemple, en faisant `echo $obj;` .