Mémo - Programmation Orientée Objet - PHP

Campus Numérique 2018 - Véronique ROUAULT

Résumé sur LogicBig.com.

Cours complet de Grafikart

Qu'est-ce que la POO

Consiste en la définition et l'interaction de briques logicielles appelées objets.

Quelques définitions et syntaxes de base

Classe (Class)

- Une classe est un ensemble de variables et de fonctions (attributs et méthodes).

En général on crée un fichier par classe.

Instance

Instancier une classe, c'est se servir d'une classe afin qu'elle crée un objet. Une instance est donc un objet.

Attribut (ou propriétés)

- Un attribut est une variable

Méthodes

- Une méthode est une fonction

Principe d'encapsulation

- Seul le créateur de la classe peut la modifier. Avec des attributs privés l'accès à la classe est réservée.

Syntaxes de base

Création d'une classe

On déclare une classe avec le mot-clé *class* suivi du nom de la classe en minuscules et deux accolades ouvrantes et fermantes qui encercleront la liste des attributs et méthodes.

Syntaxe

```
<?php
```

```
class MyClass {
   // defining a property
   public $myProperty = 'a property value';

   // defining a method
   public function display() {
     echo $this->myProperty;
   }
}

$var = new MyClass();
$var->display();
?>
```

Renvoie

```
a property value
```

Le mot clé "New" : Permet de créer une nouvelle instance de classe

Un objet, c'est une instance de la classe.

```
$var = new MyClass();
```

pseudo-variable $this

Fait référence à l'objet courant. Il est utilisé uniquement dans la classe.

Object operator ->

Appel d'une méthode ou de l'accès à une propriété sur une instance d'objet. Utilisé aussi avec $this

```
$var->display();
```

Visibilité d'un attribut ou d'une méthode

Les mots clés de visibilité (public, protected, private) déterminent la façon dont les propriétés / méthodes d'un objet sont accessibles:

- *public* : peut être consulté n'importe où.
- *protected* : peut être consulté par la classe et les sous-classes uniquement.

- *private* : peut être consulté par la classe seulement.

Une propriété doit être définie avec l'un des mots clés de visibilité ci-dessus. Une méthode définie sans aucun d'eux aura une visibilité publique par défaut. Pour les méthodes, peu importe leur visibilité. C'est ce qu'on appelle le principe d'encapsulation.

Syntaxe

```php
<?php
class Personne {
  public $var1 = 'public var';
  protected $var2 = 'protected var';
  private $var3 = 'private var';

  function printHello() {
    echo $this->var1 . '<br>';
    echo $this->var2 . '<br>';
    echo $this->var3 . '<br>';
  }
}

$obj = new MyClass();
echo $obj->var1 . '<br>'; // prints public var
$obj->printHello(); // prints all
```

Renvoie

```
public var
public var
protected var
private var
```

Création d'attributs

```php
<?php
class Personnage
{
  private $_force = 50;        // La force du personnage, par défaut à 50.
  private $_localisation = 'Lyon'; // Sa localisation, par défaut à Lyon.
  private $_experience = 1;        // Son expérience, par défaut à 1.
  private $_degats = 0;            // Ses dégâts, par défaut à 0.
}
```

Chaque attribut est précédé d'un underscore (« _ »). Notation à respecter (il s'agit de la notation PEAR)

Création de méthodes

Les méthodes ne sont généralement pas masquées à l'utilisateur sauf si on le souhaite vraiement.

```
<?php
class Personnage
{
  private $_force;        // La force du personnage
  private $_localisation; // Sa localisation
  private $_experience;   // Son expérience
  private $_degats;       // Ses dégâts

  public function deplacer() // Une methode qui deplacera le personnage{
  }

  public function frapper() // Une methode qui frappera un personnage{
  }

  public function gagnerExperience() // Une methode augmentant l'attribut
$experience {
  }
}
```

Constantes de classe

Définies avec le mot clé 'const'.

Peut pêtre défini dans une classe. La visibilité par défaut des constantes de classe est publique.

Les constantes sont allouées une fois par classe, et non pour chaque instance de classe.

Opérateur de résolution de portée (:😀

Au lieu d'utiliser ->, le double-point permet l'accès à la statique et à la constante. Cet opérateur est également utilisé pour accéder aux fonctionnalités de super classe.

Utiliser 'self'

Au lieu d'utiliser *$ this*, le mot-clé *self* est utilisé pour accéder aux constantes de la classe. Généralement, pour tous les accès de niveau classe, self doit être utilisé et pour tous les accès aux instances d'objet en cours $ ceci doit être utilisé dans la classe.

Syntaxe

```
<?php
class MyClass {
   const PI = 3.14159;
```

```
  function showPI() {
       echo self::PI . "<br>";
  }
}

echo MyClass::PI . "<br>";
$class = new MyClass();
$class->showPI();
echo $class::PI . "<br>";
?>
```

Renvoie

```
3.14159
3.14159
3.14159
```

Créer et manipuler un objet

Créer un objet

Pour créer un nouvel objet, faire précéder le nom de la classe à instancier du mot-clé *new*

```
<?php
$perso = new Personnage;
```

Appeler les méthodes de l'objet

```
<?php
// Nous créons une classe « Personnage ».
class Personnage
{
  private $_force;
  private $_localisation;
  private $_experience;
  private $_degats;

  // Nous déclarons une méthode dont le seul but est d'afficher un texte.
  public function parler()
  {
    echo 'Je suis un personnage !';
  }
}
```

```
$perso = new Personnage;
$perso->parler();
```

Accéder à un élément depuis la classe

Si on essaye d'accéder à un attribut privé, php retournera une erreur :

Pour accéder à un attribut privé hors de la classe on soit utiliser la pseudo-variable *$this*

A noter : *$this* est une variable représentant l'objet à partir duquel on a appelé la méthode :

```
<?php
class Personnage
{
  private $_experience = 50;

  public function afficherExperience()
  {
    echo $this->_experience;
  }

  public function gagnerExperience()
  {
    // On ajoute 1 à notre attribut $_experience.
    $this->_experience = $this->_experience + 1;
  }
}
$perso = new Personnage;
$perso->gagnerExperience();   // On gagne de l'expérience.
$perso->afficherExperience(); // On affiche la nouvelle valeur de l'attribut.
```

Implémenter d'autres méthodes

```
<?php

class Personnage
{
  private $_degats; // Les dégâts du personnage.
  private $_experience; // L'expérience du personnage.
  private $_force; // La force du personnage (plus elle est grande, plus
l'attaque est puissante).

  public function frapper($persoAFrapper)
  {
    $persoAFrapper->_degats += $this->_force;
  }
```

```
  public function gagnerExperience()
  {
    // On ajoute 1 à notre attribut $_experience.
    $this->_experience = $this->_experience + 1;
  }
}

$perso1 = new Personnage; // Un premier personnage
$perso2 = new Personnage; // Un second personnage

$perso1->frapper($perso2); // $perso1 frappe $perso2
$perso1->gagnerExperience(); // $perso1 gagne de l'expérience

$perso2->frapper($perso1); // $perso2 frappe $perso1
$perso2->gagnerExperience(); // $perso2 gagne de l'expérience

?>
```

Résumé :

```
la méthodefrapper()demande un argument : le personnage à frapper ;

cette méthode augmente les dégâts du personnage à frapper en fonction de la
force du personnage qui frappe.
```

Exiger des objets en paramètre

```
<?php
class Personnage
{
  // …

  public function frapper(Personnage $persoAFrapper)
  {
    // …
  }
}
```

En ajoutant le nom de la classe au paramètre on s'assure que la méthode *frapper()* ne sera exécutée que si le paramètre passé est de type *Personnage*, sinon PHP interrompt tout le script.

Accesseurs et mutateurs

Méthodes magiques :

- __get() : getters ou accesseur

- __set() : setters ou mutateur

Ces méthodes portent par convention le même nom que l'attribut dont elles renvoient la valeur. Il yn a autant de getter et de setter que d'attributs.

Accéder à un attribut : l'accesseur ou getter

```php
<?php
class Personnage
{
  private $_force;
  private $_experience;
  private $_degats;

  public function frapper(Personnage $persoAFrapper)
  {
    $persoAFrapper->_degats += $this->_force;
  }

  public function gagnerExperience()
  {
    // Ceci est un raccourci qui équivaut à écrire « $this->_experience = $this->_experience + 1 »
    // On aurait aussi pu écrire « $this->_experience += 1 »
    $this->_experience++;
  }

  // Ceci est la méthode degats() : elle se charge de renvoyer le contenu de l'attribut $_degats. C'est un getter
  public function degats()
  {
    return $this->_degats;
  }

  // Ceci est la méthode force() : elle se charge de renvoyer le contenu de l'attribut $_force. C'est un getter
  public function force()
  {
    return $this->_force;
  }

  // Ceci est la méthode experience() : elle se charge de renvoyer le contenu de l'attribut $_experience. C'est un getter
  public function experience()
  {
    return $this->_experience;
  }
}
```

Modifier la valeur d'un attribut : les mutateurs ou setters

Ces méthodes sont de la forme *setNomDeLAttribut()*

Voici la liste des mutateurs (ajoutée à la liste des accesseurs) de notre classePersonnage:

```
<?php
```

Constructeurs et destructeurs

Une fonction de constructeur __construct () est une fonction spéciale (magique) qui est automatiquement appelée quand une instance d'objet est créée avec le mot-clé 'new'.

Un constructeur peut avoir n'importe quel nombre de paramètres définis par l'utilisateur. Les constructeurs devraient idéalement être utilisés pour initialiser l'objet Le return est automatique avec un constructeur.

Un *destructeur* est une fonction spéciale __destruct () qui est appelée automatiquement lorsque l'objet est supprimé de la mémoire.

Les destructeurs sont généralement utilisés pour le nettoyage, la persistance de l'état, etc.

Syntaxe

```
<?php
class MyClass {
   private $prop;

   function __construct($var) {
     echo 'In constructor of ' . __CLASS__ . '<br>';
     $this->prop = $var;
   }

   public function displayProp() {
     echo $this->prop .'<br>';
   }

   function __destruct() {
     echo 'destroying ' . __CLASS__;
   }
}

$a = new MyClass('the property value');
$a->displayProp();
?>
```

Renvoie

```
In constructor of MyClass
the property value
destroying MyClass
```

L'auto-chargement de classes

Propriétés statiques et méthodes

Les propriétés de classe et les méthodes peuvent être déclarées avec un mot-clé *static* qui en fait des fonctionnalités de niveau classe et nous n'avons pas besoin d'une instance de classe pour accéder à ces fonctionnalités.

Comme les constantes de classe, nous accédons aux propriétés/méthodes statiques avec deux points (:😃 et pour y accéder dans la classe, nous utilisons le mot-clé *self*.

$this n'est pas disponible dans une méthode statique.

Par défaut, les fonctionnalités statiques ont une accessibilité «publique».

Pour l'initialisation de variable statique, les mêmes règles s'appliquent en tant qu'expressions const.

Syntaxe

```php
<?php
class MyClass {
  static $var = 'a static property';

  static function aMethod() {
    return self::$var;
  }
}

echo MyClass::$var . '<br>';
echo MyClass::aMethod();
?>
```

Renvoie

```
a static property
a static property
```

Héritage

L'héritage est le processus d'extension d'une classe existante (classe parente) par une nouvelle classe (sous-classe fille) utilisant des mots-clés *extends*.

Une sous-classe hérite de toutes les propriétés et méthodes de sa classe parente sauf pour les classes privées.

L'héritage est utilisé pour la réutilisation du code.

PHP permet l'héritage unique (au plus une super classe) Syntaxe

```
<?php
class MyParentClass {
  protected $var = 'a super call property';

  public function display() {
     echo $this->var . '<br>';
  }
}
class MySubclass extends MyParentClass {
  public function getVar() {
     return 'returning ' . $this->var;
  }
}

$a = new MySubClass();
$a->display();
echo $a->getVar();
?>
```

Renvoie

```
a super call property
returning a super call property
```