PHP - Programmation orientée objet

Date de dernière mise à jour : 27/06/2007 à 19:36

Présentation de la POO (programmation orientée objet) en PHP

La POO (programmation orientée objet) est une forme particulière de programmation destinée à faciliter la maintenance et la réutilisation / adaptation de vos scripts PHP. Elle consiste à représenter des objets (du monde réel ou non) sous une forme d'entités informatiques. On représente généralement un objet global par ce que l'on appelle une classe. Une classe va regrouper un ensemble de fonctions et de propriétés pouvant agir sur l'objet. Si on prend par exemple une voiture dans le monde réel, on peut modéliser une voiture par une classe "Voiture" qui aura comme propriétés le nombre de roues, le nombre de portes, etc ...

Les classes

Une classe regroupe des fonctions et des variables (appelées cette fois "attributs", car il s'agit des attributs d'une classe) qui interragissent avec l'objet. C'est à dire que pour un objet "voiture" par exemple, vous aurez une classe nommée "Voiture" et vous pourrez avoir une fonction qui modifie le niveau de carburant (le niveau de carburant étant un attribut de la classe que l'on ne peut modifier que via une fonction (appelée "méthode") qui ira modifier cet attribut). On appelle ce principe l'encapsulation des données, le but de l'encapsulation des données étant de ne pas pouvoir accéder aux données de l'objet directement mais via des fonctions (appelées ici "méthodes"). Chaque attribut peut donc disposer de droits d'accès à l'extérieur de la classe. Nous verrons tout ceci au fur et à mesure que vous lisez cette page.

Avant toute chose, PHP 5 a amélioré le support objet de PHP par rapport à PHP 4, nous allons donc utiliser PHP 5 pour tous nos exemples. Si vous avez besoin de savoir comment PHP 4 fonctionne avec l'objet, je vous conseille d'aller jeter un oeil dans la rubrique codes sources orientés objet du site, qui vous permettra d'obtenir des sources de tous types.

Voici comment une classe peut être codée en PHP 5 :

```
<?php
class
Voiture
{
/**  * Déclaration des attributs  */
private
$niveau_carburant
;  private
$nombre_portes
;  private
$nombre_roues
;
/**  * Cette méthode un peu spéciale est le constructeur, elle est exécutée lorsque vous "créez" votre objet.  Elle doit initialiser les attributs de la classe.  */
public function
__construct
()  {
$this
->
niveau_carburant
=
50
;
$this
->
nombre_portes
```

```
=
3
;
$this
->
nombre_roues
=
4
;        }
/**         * Première méthode accessible par tous et modifiant le niveau de carburant          */
public function
modifier_carburant
(
int $niveau
)        {
$this
->
niveau_carburant
=
$niveau
;        }            /**                         * Seconde méthode accessible à tous et modifiant le nombre de portes                 */
public function
modifier_nb_portes
(
int $nb_portes
)                                    {
$this
->
nombre_portes
=
$nb_portes
;                        }
    }
?>
```

Notion d'objet

Imaginions que vous souhaitiez créer deux voitures dans votre code. Sans programmation orientée objet, on aurait pu stocker les voitures et leurs attributs dans un tableau, ce qui devient vite impossible à gérer. Avec la programmation orientée objet, vous allez pouvoir créer deux objets différents en deux lignes de code. Créer un objet se fait en "instanciant" une classe. Quand on instancie une classe, on crée une version de l'objet ayant des caractéristiques propres. Si vous créez un deuxième objet, il est indépendant du premier, bien qu'ils utilisent tous les deux la même classe (ici nos deux objets seront des voitures, ils utiliseront donc la même classe "Voiture", mais seront bien différents pour autant. Nous pourrons par exemple avoir une voiture ayant trois portes et la seconde 5 portes).

Créer un objet (instanciation d'une classe)

Voici comment on crée un objet Voiture en PHP (on supposera que vous avez inclus le fichier contenant la classe, ou alors que le code de la classe se trouve au dessus du code que vous allez voir) :

```
<?php    $objet_voiture
= new
Voiture
();
?>
```

La variable $objet_voiture représente l'objet qui est ici une voiture. Lorsque vous exécutez ce code, la méthode __construct() de la classe est exécutée. Comme il s'agit d'une fonction, elle peut prendre elle aussi des paramètres. Tout dépend comment vous souhaitez coder votre classe, mais vous pourriez très bien avoir une fonction __construct() qui initialise les attributs en fonction des paramètres que vous lui fournissez. Voici ce que ça pourrait donner :

```
<?php
class
Voiture
{
/**  * Déclaration des attributs  */
private
$niveau_carburant
;  private
$nombre_portes
;  private
$nombre_roues
;
/**
  * Cette méthode un peu spéciale est le constructeur, elle est exécutée lorsque vous "créez" votre objet. Elle doit initialiser les attributs de la clas
se.  */
public function
__construct
(
int $nb_carburant
,
int $nb_portes
,
int $nb_roues
=
4
)  {
$this
->
niveau_carburant
=
$nb_carburant
;
$this
->
nombre_portes
=
$nb_portes
;
$this
->
nombre_roues
=
$nb_roues
;  }
/**  * Première méthode accessible par tous et modifiant le niveau de carburant  */
public function
modifier_carburant
(
```

```
int $niveau
)       {
$this
->
niveau_carburant
=
$niveau
;       }
/**       * Seconde méthode accessible    à tous et modifiant le nombre de portes       */
public function
modifier_nb_portes
(
int $nb_portes
)       {
$this
->
nombre_portes
=
$nb_portes
;       }     }
?>
```

Lorsque vous créerez l'objet voiture, vous allez pouvoir sans passer par les méthodes appropriées lui fixer un niveau de carburant, un nombre de portes et un nombre de roues (par défaut 4). Voici deux façons de créer l'objet :

```
<?php     $objet_voiture
= new
Voiture
(
50
,
3
);
//50 : niveau de carburant et 3 portes, on a pas besoin de spécifier le nombre de roues car il est de 4 par défaut
$autre_voiture
= new
Voiture
(
10
,
5
,
6
);
//10 : niveau de carburant, 5 portes et 6 roues
?>
```

Il est important de signifier que les objets $objet_voiture et $autre_voiture sont deux objets différents qui peuvent avoir leurs propriétés propres. Vous commencez peut-être maintenant à comprendre avec quelle simplicité vous allez pouvoir créer autant d'objets que vous le souhaitez ;)

Constructeurs et destructeurs

Les constructeurs et destructeurs sont des méthodes particulières. D'une part, elle commencent par deux signes "underscores" accolés (touche 8 du pavé alphanumérique). D'autre part, elles sont exécutées à des moments précis.

Le constructeur est appelé automatiquement quand vous créez votre objet. Généralement, cette méthode sert à donner une valeur de départ aux différents attributs de la classe pour vous permettre de "construire" l'objet. Ce n'est cependant pas une obligation et vous pouvez très bien ne rien mettre dans cette méthode.

Le destructeur est appelé à la fin d'exécution de votre script. La méthode s'appelle cette fois-ci __destruct().

```
<?php
class
Voiture
{
/**  * Déclaration des attributs  */
private
$niveau_carburant
;  private
$nombre_portes
;  private
$nombre_roues
;
/**
  * Cette méthode un peu spéciale est le constructeur, elle est exécutée lorsque vous "créez" votre objet. Elle doit initialiser les attributs de la clas
se.  */
public function
__construct
(
int $nb_carburant
,
int $nb_portes
,
int $nb_roues
=
4
)  {
$this
->
niveau_carburant
=
$nb_carburant
;
$this
->
nombre_portes
=
$nb_portes
;
$this
->
nombre_roues
=
$nb_roues
;  }  /**  * Destructeur, appelé quand l'objet est détruit  */
public function
__destruct
(
```

```
)                    {                    echo
'L\'objet a été détruit'
;


            }            }


?>
```

Visibilité des propriétés et méthodes

PHP 5 introduit la notion de visibilité de méthodes et d'attributs. Chaques attributs et méthodes peuvent se voir attribuer un droit d'accès. Le principe de l'encapsulation voudrait que l'on mette tous les attributs uniquement modifiables et accessibles à l'intérieur de la classe, et les méthodes accessibles de l'extérieur. En pratique, ce n'est pas toujours le cas.

Les trois mots permettant de gérer les accès sont ceux-ci :

-public : n'importe qui a accès à la méthode ou à l'attribut demandé.
-protected : seule la classe ainsi que ses sous classes éventuelles (classes héritées, on verra ce que c'est plus loin).
-private : seule la classe ayant défini l'élément peut y accéder.

Pour la classe Voiture, vous pouvez constater que les attributs ne sont pas modifiables à l'extérieur de la classe, il faut passer par les méthodes appropriées.

Pour accéder à un attribut d'une classe, on utilise ce code : $objet->attribut

```
<?php
class
Voiture
{
/**          * Déclaration des attributs          */
private
$niveau_carburant
;          public
$nombre_portes
;          private
$nombre_roues
;
/**
        * Cette méthode un peu spéciale est le constructeur, elle est exécutée lorsque vous "créez" votre objet. Elle doit initialiser les attributs de la clas
se.          */
public function
__construct
(
int $nb_carburant
,
int $nb_portes
,
int $nb_roues
=
4
)          {
```

```
$this
->
niveau_carburant
=
$nb_carburant
;
$this
->
nombre_portes
=
$nb_portes
;
$this
->
nombre_roues
=
$nb_roues
;          }    }
?>
```

Maintenant, on instancie la classe :

```
<?php    $voiture
= new
Voiture
(
50
,
3
);    echo
$voiture
->
nombre_portes
;
//va afficher "3"    car l'attribut est en accès public
echo
$voiture
->
nombre_roues
;
//Erreur, on ne peut pas y accéder car    l'attribut est en accès privé !
?>
```

Notez que l'on ne met pas de signe $ pour accéder ou définir des valeurs aux attributs d'une classe. Le mot-clé $this est un peu spécial et il désigne la classe courante.

L'héritage

L'héritage consiste à utiliser une classe parente et une ou plusieurs classes filles qui héritent des propriétés de la classe parente. Par exemple, si vous avez une classe **Vehicule**, vous pouvez avoir une classe **Voiture** qui hérite de certaines propriétés de la classe **Vehicule**, ainsi qu'une autre classe **Moto** qui va hériter de certaines propriétés de la classe **Voiture** tout en ajoutant des propriétés propres. Le mot utilisé pour dire à PHP qu'une classe hérite d'une autre est le mot-clé **extends**. Prenons l'exemple d'une classe Vehicule :

```
<?php
```

```
class
Vehicule
{
/**            * Déclaration des attributs             */
protected
$prix
;
//On souhaite que les classes qui en héritent puissent y accéder             /**
        * Cette méthode un peu spéciale est le constructeur, elle est exécutée lorsque vous "créez" votre objet. Elle doit initialiser les attributs de la clas
se.           */
public function
__construct
(
int $prix_vehicule
)           {
$this
->
prix
=
$prix_vehicule
;           }         /**
        * Cette méthode        permet de modifier le prix du véhicule
*/

public function modifier_prix

;(
int $nouveau_prix
)                                 {
$this
->
prix
=
$nouveau_prix
;                         }
    }
?>
```

Comme vous pouvez le constater, on a mis dans la classe Vehicule tout ce qui sera commun aux différents véhicules que nous allons pouvoir créer. Ici, j'ai mis un attribut "prix" car une voiture ou une moto ont toutes les deux un prix. On met donc tout ce que les véhicules ont en commun dans une même classe, et cette classe sera étendue par d'autres classes plus spécifiques. Voici maintenant une classe Voiture qui étend les propriétés de la classe Vehicule :

```
<?php
class
Voiture
extends
Vehicule
{
/**            * Déclaration des attributs             */
private
$climatisation
;
```

```
/**             * Constructeur de la classe Voiture          */
public function
__construct
(
int $prix_vehicule
,
bool $clim
)              {
parent
::
__construct
(
$prix_vehicule
);
//On appelle le constructeur de la classe Vehicule en lui fournissant le prix
$this
->
climatisation
=
$clim
;          }      }
?>
```

Voici ce que ça donne au niveau de l'instanciation :

```
<?php      $voiture
= new
Voiture
(
17000
,
TRUE
);
//On crée une voiture valant 17000 euros et ayant la climatisation
$voiture
->
modifier_prix
(
15000
);
//On peut modifier le prix de la voiture
?>
```

Comme vous pouvez le constater, on peut accéder à la classe parente comme si on accédait à la classe Voiture (avec la méthode **modifier_prix()**). Si vous aviez mis la méthode **modifier_prix()** en accès privé, vous n'auriez pu effectuer la modification directement. Le niveau de protection le plus restrictif permettant cette modification est le niveau **protected**.

Ceci marque la fin du cours sur ce chapitre, qui est vraiment très loin d'être exhaustif. Je vous conseille d'aller lire la documentation de php.net pour avoir plus de précisions sur ce sujet qui mériterait bien plus qu'un chapitre.

Chapitre suivant