

Introduction à la programmation orientée objet avec le langage PHP

Définir une classe | Créer un objet | Héritage | Opérateur et parent | Durée de vie | Les fonctions magiques | Conclusion | Bibliographie

Classe, attribut, méthode, constructeur

* PHP n'est pas un langage orienté objet mais il permet, dans une certaine mesure, la gestion des objets.
* PHP supporte la notion d'héritage et permet donc la construction d'une hiérarchie de classes et l'introduction du polymorphisme.
* PHP, dans le concept d'encapsulation, permet de définir un seul niveau de visibilité des éléments de la classe : publique.

La déclaration d'une classe se fait par l'utilisation du mot clé class, suivi du nom de la classe. A partir de ce point, le moteur PHP sait que des objets de type user (voir le code) pourront être intentés et il sera capable de leur allouer un emplacement mémoire.

```
<?php
class user {
    // propriétés

    // var Déclaration des variables
    var $name;
    var $password;
    var $last_login;

    // méthodes

class
    // constructeur
    function user ($inputName, $inputPassword) {
        $this->name = $inputName;
        $this->password = $inputPassword;
        $this->last_login = time();
    }

    // méthode d'accès
    // lire la date de la dernière connexion
    function getLastLogin() {
        return (Date("M d Y", $this->last_login));
    }
}
?>
```

Propriété (attribut ou variable)	Méthode (fonction)
<ul style="list-style-type: none">- Les attributs sont des sortes de variables internes à l'objet qui ne sont accessibles qu'à travers celui-ci. - La déclaration d'un attribut au sein d'une classe se fait par l'utilisation du mot clé var suivi du nom de l'attribu. PHP ne nécessite pas de déclarer au préalable des variables, mais, dans la cadre d'une classe, il est préférable de la faire. - Quand vous déclarez une propriété, vous ne spécifiez aucun type de donnée. Elle pourra contenir, par exemple, un entier, chaîne de caractères, un autre objet.	<ul style="list-style-type: none">- Les méthodes quant à elles, sont des fonctions internes à l'objet qui vont permettre d'agir sur les attributs de l'objet. - La déclaration d'une méthode (ou fonction) à l'intérieur d'une classe se fait de la même façon que dans un script classique. On utilise pour cela le mot réservé fonction suivi du nom de la méthode. - Si la méthode requiert des paramètres, on les indiquera entre parenthèses. En PHP il n'est pas nécessaire de spécifier le type de retour de la fonction.
<p>Une méthode particulière, le constructeur de la classe, est automatiquement appelée lors de la création d'un objet (instanciation de la classe).Cette méthode permet d'initialiser l'objet à sa création en donnant par exemple des valeurs à chacun de ses attributs. Le constructeur d'une classe est donc une méthode qui porte le même nom que la classe dont elle est le constructeur.</p>	

Les méthodes disposent d'une variable spéciale, this (suivi de l'opérateur -> qui référence une classe, une méthode ou une propriété d'une classe), qui contient l'objet ou l'instance courante. Il faut utiliser cette variable chaque fois qu'on veut nous référer à une propriété ou à une méthode de l'objet. PHP supposera qu'il s'agit d'une variable locale (voir l'usage de this au sein du constructeur de la classe user et la fonction d'affichage getLastLogin).

Instanciation, accéder aux propriétés et aux méthodes

La création d'un objet à partir d'une classe, s'appelle l'instanciation (un objet étant alors appelé instance d'une classe).
La création d'un objet permet plusieurs choses:

- La création d'un espace mémoire est réservé pour recevoir toutes les propriétés durant son utilisation
- L'attribution d'un nom à l'objet.
- En appelant le constructeur, cet objet sera initialisé (avec des valeurs par défaut ou des valeurs passées en paramètres).
- Chaque instance possède son propre jeu de propriétés, mais ses méthodes sont communes à toutes les instances de cette classe.

Pour la création d'un objet à partir d'une classe il y a quatre manière de générer l'objet:

1. Comme dans notre exemple, la variable \$currentuser contient l'objet créé avec le mot réservé new suivi du nom de la classe user. Si le constructeur de la classe ne comporte pas des arguments, il n'est pas nécessaire d'utiliser des parenthèses. En cas contraire placez-les entre parenthèses après le nom de la classe (comme pour un appel de fonction).
2. Copier un objet par un simple jeu des variables (dans l'exemple ci-dessous \$obj2 contient une copie, une nouvelle instance, susceptible de vivre sa propre vie en n'utilisant pas le constructeur de la classe user).
3. La troisième manière de générer l'objet est par inférence, c'est à dire par déduction à partir du contexte. Vue le fait que PHP permet de créer des variables \$obj3 sans une déclaration préalable on peut créer l'objet par le simple fait d'utiliser une variable dans un contexte d'objet (en appliquant l'opérateur -> à une variable suffit que cette variable contienne un objet même si jamais la classe correspondante n'a été définie (pas valable pour ajouter des méthodes).
4. La dernière manière de générer l'objet est par la transformation (casting) d'un tableau associatif. Lorsqu'un tableau devient un objet, \$obj4 , tous les éléments du tableau indexés par une chaîne de caractères deviennent des propriétés (en cas d'indexés par chaîne vide ils sont cachés pour l'objets).

Pour les objets générés par inférence (3) et casting (4) aucune classe n'a été définie. Ils appartiennent à la classe stdClass du PHP.

```
<?php
// création d'une instance par le mot new
$currentuser = new user ("Dorel", "staf2k");

// création d'une nouvelle instance par copy
$obj2 = $currentuser;
$obj2 -> commentaire = "ajoutons une propriété";

// création d'une nouvelle instance par inférence
$obj3 -> age = 25;
$obj3 -> poids= 75;

exemple1
// création d'une nouvelle instance par casting

$arr['nom'] = "Jean";
$arr['pays'] = "Suisse";
$obj4 (object) $arr;
print ($obj4->nom); // affiche la propriété nom

// imprimer la date

print ($currentuser->getLastLogin());

// imprimer le nom

print ($currentuser->name);
print ("<br />\n");
?>
```

Le propriétés d'une instance correspondent à des variables identiques à toutes celles que nous avons vues jusqu'à présent.

- Pour accéder aux propriétés il faut utiliser l'opérateur ->

- Si un propriété d'objet est elle-même un objet on peut utiliser deux fois l'opérateur -> dans la même instruction (seulement avec la version PHP4)

- À la différence des langages orientés objet (Java, C++) les membres (propriétés et méthodes) des classes PHP ne peuvent être déclarés protégé, private ou public (c'est à dire que n'importe quelle partie du script on peut y accéder pour lire ou modifier les valeurs qu'elles contiennent).

L'accès aux méthodes est semblable à l'accès aux propriétés

- L'opérateur -> est utilisé pour indiquer sur quelle instance doit porter la méthode (par exemple l'expression \$currentuser->getLasLogin()).

- Pour ce qui reste, les méthodes se comportent exactement comme des fonctions ordinaires.

```
<?php
//Objet composé d'autres objets

class chambre {
    var $nom;
    function chambre ($nom){
        //pas de conflit entre les deux "nom"
        $this->nom=$nom;
    }
}

class maison
{
    //une liste des chambres
    var $chambres;
}

exemple 2
// créer une maison vide
$home = new maison();

// ajouter des chambres
$home->chambres[] = new chambre (chambre de JJ);
$home->chambres[] = new chambre (chambre de Stek);
$home->chambres[] = new chambre (salle de bains);

// montrer le nom de la première chambre
print($home->chambres[0]); // on affiche: "chambre de JJ"
?>
```

Héritage

Le principal apport de la programmation objet est certainement de faciliter la réutilisation (reusability) de modules, en occurrence des objets. Les deux techniques utilisées sont, l'héritage et la composition (un exemple simplifié de composition dans l'exemple 2, en haut).

Ecrire une classe spécialisée à partir d'une classe générale est le meilleur moyen de réutiliser ce code déjà produit. Cette procédure s'appelle héritage et elle est introduite par le mot clé extends.

```
//
<?php

class myclass {
    var $myvar;
    function myClass($myparameter) {
        $this->myvar = $myparameter;
    }

    function afficher() {
        echo "myvar vaut : $this->myvar";
    }
}
?>

p

<?php

class myclass2 extends myclass {

    var $myvar2 ;

    function myclass2($myparameter, $myparameter2) {
        $this->myvar = $myparameter;
        $this->myvar2 = $myparameter2;
    }

    function afficher2($color) {
        echo "<font color=\"$color\">myvar vaut : $this->myvar</font><br>";
        echo "<font color=\"$color\">myvar2 vaut : $this->myvar2</font><br>";
    }
}

exemple3
?>
```

Dans cet exemple nous venons de déclarer une classe fille myclass2 qui étend la classe mère myclass précédemment créée. Dans cette classe, nous avons ajouté un attribut myvar2 et une méthode afficher2 (\$color) qui ne seront définis que dans cette classe. Nous avons également redéfini le constructeur \$parameter(\$parameter2) qui permet d'initialiser les deux attributs de l'objet lors de sa création.

//
<?php
\$myobject2 = new myclass2(10, 20);
\$arr = get_object_vars(\$obj);
\$myobject2->afficher();
?>

Ces trois lignes montrent comment utiliser un objet créé à partir d'une classe fille. Dans un premier temps, on crée une instance de la classe myclass2 que l'on stocke dans un objet nommé \$myobject2, en attribuant les valeurs 10 et 20 aux attributs \$myvar et \$myvar2. Puis on fait simplement appel à la méthode afficher2(\$color) définie dans myclass2. La troisième quant à elle est plus intéressante. En effet, à aucun moment, la méthode afficher() n'apparaît dans la classe myclass2. Et pourtant, il est tout à fait possible d'utiliser cette méthode à cet endroit car celle-ci a été héritée de la classe mère.

De la même manière, les attributs des classes mères sont également accessible dans les classes filles. Nous avons déjà pu le remarquer dans le constructeur de la classe fille myclass2 où l'on a affecté une valeur à l'attribut \$myvar qui n'est défini que dans la classe mère.

```
$this->myvar = $myparameter;

À partir de la version 4, PHP inclut un jeu de douze fonctions de gestion des classes et des instances utiles à la POO:
```

call_user_method_array -- Appelle une méthode utilisateur avec un tableau de paramètres
call_user_method -- Appelle une méthode utilisateur d'un objet
class_exists -- Vérifie qu'une classe a été définie
get_class_methods -- Retourne les noms des méthodes d'une classe
get_class_vars -- Retourne les valeurs par défaut des attributs d'une classe.
get_class -- Retourne la classe d'un objet

get_declared_classes -- Liste toutes les classes définies
get_object_vars -- Retourne un tableau associatif des propriétés d'un objet
get_parent_class -- Retourne le nom de la classe d'un objet

is_a -- Retourne TRUE si un objet est un objet est une sous-classe
is_subclass_of -- Détermine si un objet est une sous-classe
method_exists -- Vérifie que la méthode existe pour une classe.

Ces fonctions sont disponibles dans le module PHP standard, qui est toujours accessible. Il n'y pas d'installation nécessaire pour utiliser ces fonctions, elles font parties du cœur de PHP. [ch2.php.net.php]

```
// Dans ces fonctions, on définit une classe de base, et une extension.
// La classe de base définit un légume, s'il est mangeable ou pas et sa couleur.
//La sous-classe spinard ajoute une méthode pour le cuisiner, et une autre pour savoir s'il est cuisiné.
```

```
// Exemple 1. Fonctions d'objets : classes.inc
<?php
```

```
// classe de base, avec ses membres et ses méthodes
class Legume {

    var $edible;
    var $color;

    function Legume( $edible, $color="green" ) {
        $this->edible = $edible;
        $this->color = $color;
    }

    function is_edible() {
        return $this->edible;
    }

    function what_color() {
        return $this->color;
    }
}

// fin de la classe Legume // Extension la classe de base
```

```
class Epinard extends Legume {
    var $scuit = FALSE;

    function Epinard() {
        $this->Legume( TRUE, "green" );
    }

    function cuisine() {
        $this->cuit = TRUE;
    }

    function is_cooked() {
        return $this->cooked();
    }
}

// fin de la classe Epinard
?>
```

```
// Exemple 2. Fonctions d'objets : test_script.php
<?php
include "classes.inc";

// Fonctions utilitaires
```

```
function print_vars($obj) {
    $arr = get_object_vars($obj);
    while (list($prop, $val) = each($arr))
        echo "\t$prop = $val\n";
}

exemple4
```

```
function print_methods($obj) {
    $arr = get_class_methods(get_class($obj));
    foreach ($arr as $method($obj))
        echo "\tFunction $method()\n";
}

function class_parentage($obj, $class) {
    if (is_subclass_of($GLOBALS[$obj], $class)) {
        echo "L'objet $obj appartient à la classe \"$class\".get_class($obj);
        echo " est une sous-classe de $class\n";
    } else {
        echo "L'objet $obj n'est pas une sous-classe de $class\n";
    }
}

// instancie 2 objets
$vegie = new Legume(true,"blue");
$leafy = new Epinard();
```

```
// affiche les informations sur ces objets
echo "légume : CLASS ".get_class($vegie)."\n";
echo "feuilles : CLASS ".get_class($leafy);
echo ", PARENT ".get_parent_class($leafy)."\n";
```

```
// affiche les propriétés du légume
echo "\nlégumes : Propriétés\n";
print_vars($vegie);
```

```
// et les méthodes des feuillus
echo "\nfeuillus : Méthodes\n";
print_methods($leafy);
```

```
echo "\nParentés:\n";
class_parentage("leafy", "Epinard");
class_parentage("leafy", "Legume");
?>
```

Il est important de noter que dans les exemples ci-dessus, les objets \$feuille et une instance de Epinard qui est une sous-classe de Legume, donc la dernière partie du script va afficher :

```
[...]
Parentée:
L'objet feuilles n'est pas une sous classe epinard
L'objet feuilles est une sous-classe de Legume
```

Opérateur :: et parents

Nous venons de voir qu'il était possible de définir des classes à partir d'autres classes. Dans l'utilisation des objets, il pourra aussi être utile de faire appel à certaines méthodes de certaines classes sans avoir la contrainte que l'objet de cette classe soit créé auparavant.

Dans sa version 4, PHP permet d'utiliser ce genre de traitement sans des classes. On utilisera alors l'opérateur :: qui à la manière du -> pour un objet permettra l'accès aux méthodes à partir d'un nom de classe.

```
<?php
// exemple d'utilisation du l'opérateur ::
// création d'une classe simple pour afficher la date

class myDate {
    var $day;
    var $month;
    var $year;

    function myDate($day, $month, $year) {
        $this->day = $day;
        $this->month = $month;
        $this->year = $year;
    }

    function printTodayDate() {
        echo date("j/m/Y");
    }
}
?>

// Nous appelons la méthode printTodayDate() de la classe myDate sans qu'aucun objet de type myDate ne soit préalablement instancié.
<?php
myDate::printTodayDate();
?>
```

Une méthode de classe telle que celle-ci ne dois pas faire référence aux attributs qui ne seront valorisés que lors de la création effective de l'objet.

```
// exemple comment on utilise le mot réservé parent

exemple5
//De façon similaire, on peut également faire appel à la méthode de l'un objet sans pour autant être obligé de créer un objet à partir de cette classe mère. Au lieu d'utiliser le nom littéral de la classe de base dans votre code, vous pouvez utiliser le mot réservé parent , qui représente votre classe de base (celle indiquée par extends , dans la déclaration de votre classe).
```

```
<?php
class A {

    function show_one() {
        echo "Je suis A:exemple() et je fournis une fonctionnalité de base.<br>\n";
    }
}

//
class B extends A {

    function exemple() {
        echo "Je suis B:exemple() et je fournis une fonctionnalité supplémentaire.<br>\n";
        parent::exemple();
    }
}

//
$b = new B;

// Cette syntaxe va appeler B:exemple(), qui, à sont tour, va appeler A:exemple().
$b->exemple();
?>
```

// Si l'héritage change, vous n'aurez plus qu'à modifier le nom de la classe dans la déclaration extends de votre classe.

Comme toutes variables en PHP, le durée de vie d'un objet n'exécède pas la durée d'exécution du script qui le crée. Il faut transformer notre objet, en une chaîne de caractères avec la fonction serialize().

La fonction serialize retourne une chaîne représentant une valeur qui peut être stockée dans les sessions de PHP , ou une chaîne de données et la fonction unserialize() peut lire cette chaîne pour recréer la valeur originale. La fonction serialize a sauver toutes les variables d'un objet. Le nom de la classe sera sauvé mais pas les méthodes de cet objet.

Pour permettre à unserializer de lire un objet, la classe de cet objet doit être définie. C'est-à-dire, si vous avez un objet \$a de la classe A dans une page php1.php, et que vous le linéarisez avec serialize, vous obtiendrez une chaîne qui fait référence à la classe A, et contient toutes les valeurs de \$a. Pour pouvoir le relier avec la fonction unserialize dans une page page2.php, il faut que la définition de la classe A soit présente dans cette deuxième page. Cela peut se faire de manière pratique en sauvant la définition de la classe A dans un fichier séparé, et en incluant dans les deux pages page1.php et page2.php.

```
// Définir la classe A dans un fichier externe classa.inc
<?php
// classa.inc:
class A {
    var $one = 1;

    function show_one() {
        echo $this->one;
    }
}
?>
```

```
// Créer la page1.php
//page1.php:
include("classa.inc");
$a = new A;
$a = serialize($a);
// enregistrez $s où la page2.php pourra le trouver.
```

```
$fp = fopen("store", "w");
fwrite($fp, $a);
fclose($fp);
?>
```

```
// Créer la page2.php
//page2.php:
// Ceci est nécessaire pour que unserialize() fonctionne correctement
include("classa.inc");
$a = implode("", @file("store"));
unserialize($s);
// maintenant, utilisez la méthode show_one de l'objet $a.
```

\$a->show_one();
?>

Si vous utilisez les sessions et la fonction session_register pour sauver des objets, ces objets seront linéarisés automatiquement avec la fonction serialize à la fin de chaque script, et reliés avec unserialize au début du prochain script. Cela signifie que ces objets peuvent apparaître dans n'importe quelle page qui utilise vos sessions.

Il est vivement recommandé d'inclure la définition de classe dans toutes vos pages, même si vous n'utilisez pas ces classes dans toutes vos pages. Si vous l'oubliez et qu'un tel objet est présent, il perdra sa classe, et deviendra un objet de classe stdClass sans aucune fonction, et donc, plutôt inutile.

Si, dans l'exemple ci-dessus, \$a devient un objet de session avec l'utilisation de session_register("a"), vous devez pensez à inclure le fichier classa.inc dans toutes vos pages, et pas seulement page1.php et page2.php.

Les fonctions magiques

La fonction serialize() qui transforme les objets en une chaîne de caractères, le fait de façon trop "brutale" dans certains cas. En effet, il est possible que vos objets aient ouvert une connexion à une base de données et ne l'aient pas encore fermée. Dans ce cas, il est préférable (et même capital) de s'assurer que certaines opérations soit effectuées avant que l'objet ne soit effectivement linéarisé.

La fonction magique __sleep() ferme proprement toute connexion à une base de données, de valider les requêtes, de finaliser toutes les actions commencées. La fonction __wakeup() retourne obligatoirement un tableau contenant la liste des attributs à linéariser. Cette fonction est aussi pratique si vous avez de très grands objets qui n'ont pas besoin d'être sauvé entièrement. Cette fonction, doit exister dans l'objet qui sera automatiquement appelée par la fonction serialize().

La fonction __wakeup va rétablir toutes les connexions aux bases de données, et de recréer les variables qui n'ont pas été sauveées, et, si elle existe, sera automatiquement appelée par la fonction unserialize().

```
a

// Voici un exemple qui montre comment préparer les objets de la classe myclass à être linéarisés grâce à la fonction __sleep() et la fonction __wakeup().
<?php
class myclass {
    ...
    function __sleep() {
exemple7
        $this->myvar = 50;
        return array("myvar");
    }

    function __wakeup() {
        $this->myvar = 100;
    }
}
?>
```

Conclusion

Les objets sont actuellement l'un des aspects du langage PHP qui concentre le plus l'attention des développeurs du langage. On peut donc s'attendre dans les prochaines versions de PHP à une meilleure intégration et des possibilités encore plus étendues.

La partie théorique de ce cours est à présent terminée. L'essentiel qu'il aura répondu à vos attentes. Si vous observez des erreurs, des imprécisions ou d'autres fautes, merci de m'en faire part.

Bibliographie

[PHP2002] Atkinson L., Programmation en PHP, CampusPress, 2002
[PHP2003] Atkinson L., PHP 5, CampusPress, 2003
[Anthology I 2003] Fucks Harry, The PHP Anthology, Volume I: Foundations, SitePoint Pty Ltd., 2003
[Anthology II 2003] Fucks Harry, The PHP Anthology, Volume II: Applications, SitePoint Pty Ltd., 2003
[InClass] Welcome to the PHP Classes Repository[lien]
[Doc-PHP-fr] Documentation PHP 4.3.3 en Français[lien]
[Man-PHP-en] Manuals at the official PHP site[lien]
[Library-PHP] A set of PHP classes and library functions[lien]
[Tutoriel-1-fr] Cours et tutoriels, outils de développement[lien]
[SitePoint] Articles, books, forums to Site Point[lien]
[Tutoriel-2-fr] Tutoriels francophones PHP4 et MySQL[lien]
[Intro-class-en] An Introduction to Classes[lien]
[Ex-class-fr] Quelques exemples des classes[lien]