PHP 5

Programmation Orientée Objet

Introduction

« La programmation orientée objet consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle domaine) en un ensemble d'entités informatiques. Ces entités informatiques sont appelées objets. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, couleur, ...). »

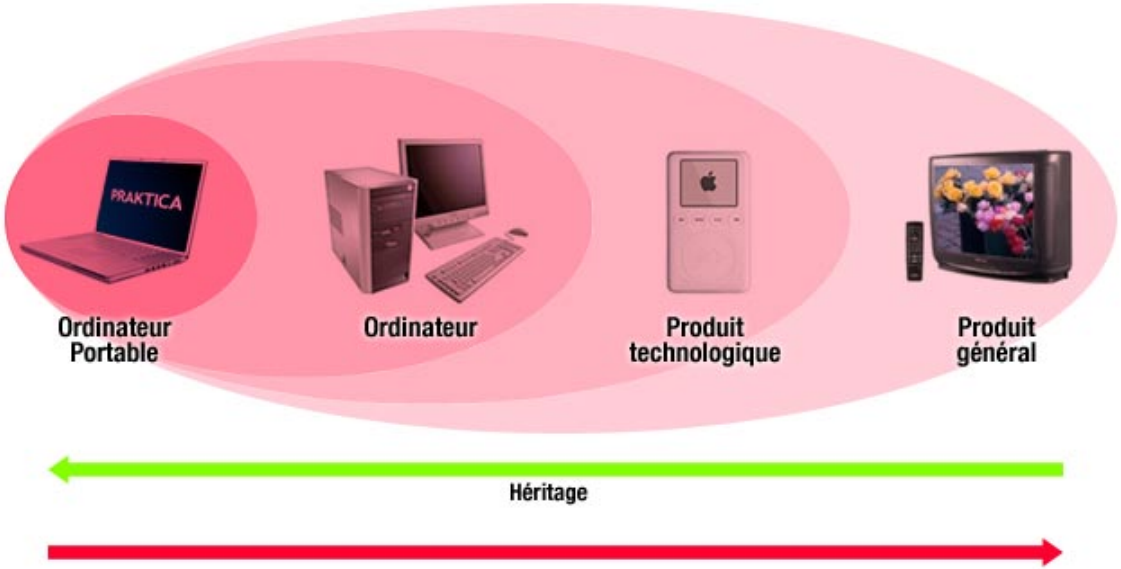
La programmation objet, qu'on appelle POO pour Programmation Orienté Objet, s'est développée à la fin des années 80 avec l'apparition du C++.

Aujourd'hui, presque tous les langages de programmation sont basés sur une représentation objet, comme le C# (prononcé Sharp) le C++ de Microsoft pour le point net (Dot Net), le JAVA de SunMicrosystem, l'Objectif C le C++ de Mac OS X, ou bien sur les premiers langages grand public tels que le Visual BASIC sur Windows ou Real BASIC sur Mac.

Les grands principes : classifier et décomposer

La représentation objet se calque le plus possible sur notre perception de la réalité. Comme nous pourrions qualifier notre environnement et comme nous l'avons toujours fait, en classifiant et décomposant des processus complexes en une multitude de processus simples et abordables.

De prime abord, les baleines n'ont rien avoir avec les vaches, pourtant ces deux espèces ont en commun d'être des mammifères vertébrés, elles partagent les mêmes caractéristiques (respiration pulmonaire, système nerveux central développé, reproduction vivipare, etc...), ce sont des mammifères.
Donc toutes espèces, objets, éléments qui nous entourent peuvent être "classés" dans différentes catégories. Catégories qui peuvent faire partie elles même d'autres catégories...



L'ordinateur portable fait partie de la catégorie des ordinateurs, qui font partie des produits technologiques, qui eux mêmes font partie des produits: catégorie général.
On pourrait continuer plus loin dans ce classement, mais il faut bien s'arrêter à un moment.
Par contre, on ne peut pas remonter, un ordinateur n'est pas forcément portable et inéluctablement un produit n'est pas forcement un ordinateur portable. Le classification se fait de manière descendante uniquement.

Les grands principes : Décomposition

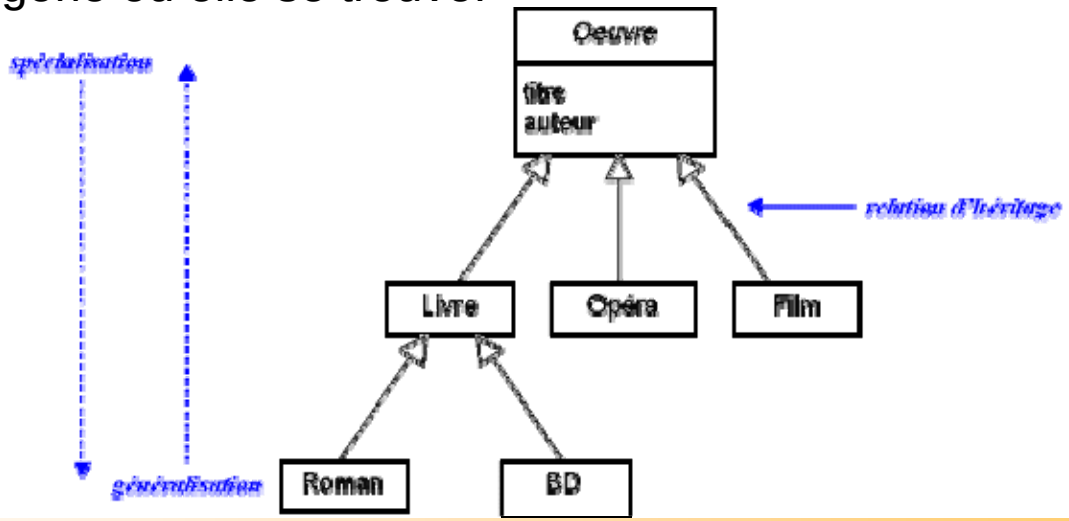
C'est simple, exemple les animaux :

Une sous-catégorie (une vache) fait partie d'une catégorie (mammifère): une vache est un mammifère.
Une catégorie (mammifère) ne peut faire partie d'une sous-catégorie (une vache): un mammifère n'est pas forcément une vache.

On peut dire également de l'ordinateur portable qu'il a une multitude d'éléments communs avec les ordinateurs de bureau, il hérite du processeur, de la mémoire, du CD-Rom etc.... Que les ordinateurs ont également de nombreux point commun avec une console de jeux, un DVD de salon, un lecteur MP3 : La mémoire, le processeur... C'est moins systématique puisqu'ils ne font pas partie de la même catégorie et qu'on descend dans la hiérarchisation des produits listés.

Alors que la classification se fait de manière descendante, l'héritage des fonctionnalités se fait de manière montante. Une sous-catégorie hérite donc des caractéristiques de la catégorie où elle se trouve.

Notation UML pour l'héritage



Les grands principes : Décomposition

Si on prend une formule 1 par exemple, une voiture extrêmement complexe à construire, et bien Ferrari n'est que le centralisateur de centaines de fabricants qui conçoivent le système de freinage, les pneus, le volant, l'informatique embarquée... Prestataires qui font eux aussi appel à d'autres prestataires pour élaborer une ou des parties de leur commande pour Ferrari.

Chaque intervenant est libre de faire son travail comme il l'entend, pour autant qu'il connaisse les attentes de son client et les contraintes, comme par exemple comment intégrer son travail dans le produit final: la voiture.

L'intérêt.
Donc la POO permet non seulement de classifier en catégorie, sous-catégorie, les données dont tous programmes a besoin, d'utiliser un système d'héritage, mais aussi de faire en sorte de diviser un programme complexe en une multitude de briques élémentaires qui font un processus bien précis, en connaissant ce qu'elles doivent traiter, comment le traiter et ce qu'elles doivent renvoyer comme résultat, en d'autre terme ce qu'on attend d'elles. Par contre, à l'intérieur elles sont autonomes. Grâce à ces possibilités, elles peuvent être reprises presque n'importe où dans un autre programme.

Cette division des tâches est pratique pour le travail en équipe, puisque chaque équipe peut travailler sur sa partie, sans avoir à connaître toutes les autres tâches. L'équipe ne connaît que les variables qu'elle devra traiter et le résultat à renvoyer.

Exemple avec notre F1:
La F1 (instance ou objet) est une voiture (catégorie) de sport (sous-catégorie)) qui roule (héritage des caractéristiques d'une voiture) fabriquée par Ferrari (une classe), cette F1 de Ferrari (instance ou objet de la classe) a demandée la participation de différents prestataires (autres classes) comme par exemple l'équipe en charge de faire tourner le moteur (autre objet) de 700 chevaux (propriété ou attibut de l'objet) en faisant les réglages adéquates- 2 tours de tournevis à gauche- (méthodes ou prototypes) se concentrant uniquement (variable) sur la partie moteur.

De la réalité à la programmation objet

La Classe : La classe est la description d'une catégorie. Comment l'identifier, quels sont ses principes. L'ordinateur portable est une classe, mais un PowerBook n'est pas une classe, c'est un Objet, ou plutôt une instance. La classe "Ordinateur portable" contiendra en faite, par exemple, processeur, mémoire, type de lecteur, type d'entrée, type de sortie...

On voit que la classe n'est en faite qu'un moule pour identifier comment l'objet peut être et va être créé.

Instance : L'objet en lui même. Le fait d'instancier une classe fait la création d'un objet. Le PowerBook d'Apple, est une instance de la classe Ordinateur portable.

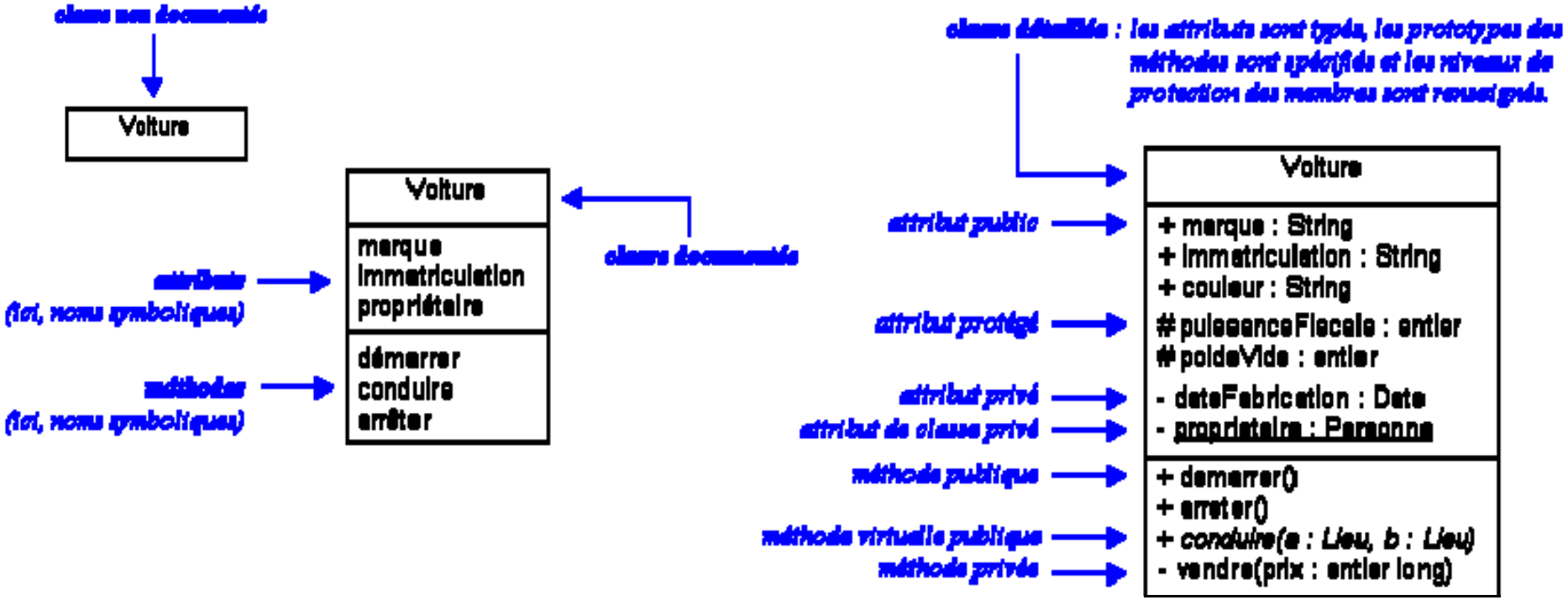
Propriété : Qu'on appelle aussi l'Attribut. Les propriétés de l'objet sont en fait les valeurs associées à cet objet. Par exemple pour l'objet PowerBook, on peut préciser sa taille mémoire, sa fréquence du processeur, sa couleur, son écran...
On peut voir les propriétés de l'objet comme les variables qui y sont exclusivement rattachées.

Les Méthodes : Qu'on appelle aussi prototype ou opérations. Les méthodes de l'objet sont les procédures de fonctionnement de l'objet. Comment interagir avec lui, lui ordonner telle ou telle action. Pour l'objet PowerBook, la méthode de cet objet 'Allumer' allumera l'ordinateur portable PowerBook.

De la réalité à la programmation objet

Notation UML :Diagramme de classes

Une classe d'objets est représentée par un rectangle comprenant trois parties :



nom de la classe, attributs et méthodes.

Les listes des attributs et des opérations sont toutefois optionnelles suivant le degré de détail recherché dans un diagramme : ces parties peuvent être vides ou même absentes. Les attributs et les opérations possèdent une visibilité (notamment publique ou protégée) qui est indiquée par un symbole précédent leur nom.

Php et la POO : Créer une classe

Toutes les notions POO nécessaires, ne sont présentes dans PHP que depuis la version 5, avant cela il manquait beaucoup d'éléments qui permettaient à prétendre de faire de la programmation Objet, notamment la notion de constructeur et de destructeur mais passons…

Dans la suite du document je vous donne les éléments nécessaire à la POO dans les version 5 et 4 de PHP quand cela est possible.

Quelques remarques :

Vous **NE POUVEZ PAS** couper la définition d'une classe en plusieurs fichiers.

De la même façon, vous **NE POUVEZ PAS** couper la définition d'une classe en de multiples blocs. Ce qui suit ne fonctionnera pas :

```
<?php
class test {
?>
<?php
    function test() {
        echo 'OK';
    }
}
?>
```

Impossible !

Le nom stdClass est utilisé en interne par Zend et ne doit pas être utilisé.

Les noms de fonctions __sleep et __wakeup sont magiques en PHP. Vous ne pouvez pas utiliser ces noms de fonctions dans vos classes.

En PHP 4, seuls les initialiseurs constants pour les variables var sont autorisés. Utilisez les constructeurs pour les initialisations variables, ou utilisant des expressions.

Php et la POO : Créer une classe

PHP 4

```
<?php
class Panier {
// Eléments de notre panier
   var $items;
// constructeur
function panier(){

}
// Ajout de $num articles
function add_item ($artnr, $num) {
       $this->items[$artnr] += $num;
 }

// suppression
function remove_item($artnr, $num) {
 if ($this->items[$artnr] > $num) {
    $this->items[$artnr] -= $num;
    return true;
 }
elseif($this->items[$artnr] == $num)
{unset($this->items[$artnr]);
 return true;
}
else { return false;
}
}
}
?>
```

PHP 5

```
<?php

class Panier {

function __construct() {
        print « dans constructeur\n";
     $this->Name = "MyClass";
 }

function __destruct() {
 print "Destruction de ".$this->name."\n";
 }

 // Ajout de $num articles
function add_item ($artnr, $num) {
       $this->items[$artnr] += $num;
 }

// suppression
function remove_item($artnr, $num) {


}

}
?>
```

NB : Les 2 syntaxes sont acceptés avec PHP5

Php et la POO : Créer une classe

Initialisation de membres de classe

```
<?php
/* Aucune de ces syntaxes ne fonctionnera en PHP 4  ou 5 */
class Panier {
    var $date_du_jour = date("d/m/Y");
    var $name = $firstname;
    var $owner = 'Fred ' . 'Jones';
    var $items = array("DVD", "Télé", "Magnétoscope");
}

/* Voici comment cela doit se faire. */
class Panier {
    var $date_du_jour;
    var $name;
    var $owner;
    var $items;
    function Panier() {
        $this->date_du_jour = date("d/m/Y");
        $this->name = $GLOBALS['firstname'];
        /* etc. */
    }
}
?>
```

Php et la POO : Créer une classe

Création d'un objet de classe = instancier un objet

```
<?php

$cart = new Panier;
$cart->add_item("10", 1);

$another_cart = new Panier;
$another_cart->add_item("0815", 3);

?>
```

L'instruction ci-dessus crée l'objet $cart de la classe Panier .

La fonction add_idem() est appelée afin d'ajouter l'article numéro 10 dans le panier $cart.

3 articles numéro 0815 sont ajoutés au panier $another_cart .

Php et la POO : Créer une classe

Accès aux membres d'une classe

```
<?php

// correct, le signe $ est unique
$cart->items  = array("10" => 1);

// incorrect, car $cart->$items devient $cart->""
$cart->$items = array("10" => 1);

// correct, mais risque de ne pas se comporter comme
prévu
// $cart->$myvar devient $cart->items
$myvar = 'items';
$cart->$myvar = array("10" => 1);

?>
```

Php et la POO : Créer une classe

Portée des classes, variables membres et méthodes :

Il est désormais possible de définir la portée des variables ou méthodes membres d'une classe.

Public : possibilité d'accès dans tous les contextes

Protected : les variables ou méthodes membres déclarées en *protected* ne sont accessibles qu'à l'intérieur de la classe ou d'une classe dérivée

Private : accès uniquement depuis la classe

Final : ce mot clé s'applique uniquement aux méthodes, et permet d'empêcher sa réécriture dans une classe dérivée

Abstract : ce mot clé s'applique aux classes et aux méthodes. Une classe déclarée en *abstract* ne peut être instanciée. Quand aux méthodes, elles devront être réécrites dans une classe dérivée.

Php et la POO : Créer une classe

Les constantes

Les constantes sont des variables qui devront rester inchangées tout au long de la vie de l' objet. Toute tentative de réécriture provoquera une erreur.
exemple :

```
<?php
class MaClasse {

    const COUCOU = 'salut';

    function __construct() {
        echo 'Hello World';
    }
}


// affichera : salut
echo MaClasse::COUCOU;
?>
```

Php et la POO : Créer une classe

```
<?php
class Panier {
    // Eléments de notre panier
    var $items;

    // Ajout de $num articles de type $artnr au panier

    function add_item ($artnr, $num) {
        $this->items[$artnr] += $num;
    }

    // Suppression de $num articles du type $artnr du panier

    function remove_item($artnr, $num) {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } elseif ($this->items[$artnr] == $num) {
            unset($this->items[$artnr]);
            return true;
        } else {
            return false;
        }
    }
}
?>
```

Php et la POO : Créer une classe

Héritage/Généralisation

Une classe ne peut hériter que d'une seule autre classe, et l'héritage multiple n'est pas supporté. Les héritages se font avec le mot clé ' extends '.

Héritage de classes

```
<?php

class Panier_nomme extends Panier {
    var $owner;

    function set_owner ($name) {
        $this->owner = $name;
    }

}
?>
```

Attention :PHP 4 n'appelle pas automatiquement le constructeur de la classe supérieure depuis le constructeur de la classe dérivée. Il est de votre responsabilité de propager l'appel des constructeurs.

Php et la POO : Créer une classe

Parfois, il est pratique de faire référence aux fonctions et variables d'une classe de base, ou bien d'utiliser des méthodes de classes qui n'ont pas encore d'objets créés.

L'opérateur « :: » est là pour ça!

Exemples avec l'opérateur ::

```
<?php
class A {
  function example() {
    echo "Je suis la fonction originale A::example().<br />\n";
  }
}
class B extends A {
  function example() {
    echo "Je suis la fonction redéfinie B::example().<br />\n";
    A::example();
  }
}
// Il n'y a pas d'objets de classe A.
// L'affichage est :
//   Je suis la fonction originale A::example().<br />
A::example();
// Création d'un objet de la classe B.
$b = new B;
$b->example();
// L'affichage est :
//   Je suis la fonction redéfinie B::example().<br />
//   Je suis la fonction originale A::example().<br />
?>
```

Php et la POO : Créer une classe

Parfois, il est pratique de faire référence aux fonctions et variables d'une classe de base, ou bien d'utiliser des méthodes de classes qui n'ont pas encore d'objets créés.

Au lieu d'utiliser le nom littéral de votre classe de base dans votre code, vous pouvez utiliser le mot réservé parent , qui représente votre classe de base (celle indiqué par extends , dans la déclaration de votre classe).

```
<?php
class A {
  function example() {
    echo "Je suis la fonction originale A::example().<br />\n";
  }
}
class B extends A {
  function example() {
    echo "Je suis la fonction redéfinie B::example().<br />\n";
    parent::example();
  }
}
// Création d'un objet de la classe B.
$b = new B;
$b->example();

// L'affichage est :
//   Je suis la fonction redéfinie B::example().<br />
//   Je suis la fonction originale A::example().<br />
?>
```

Php et la POO : Sauvegarde d'objets

serialize retourne une chaîne représentant une valeur qui peut être stockée dans les sessions de PHP, ou une base de données. **unserialize** peut relire cette chaîne pour recréer la valeur originale. serialize va sauver toutes les variables d'un objet. Le nom de la classe sera sauvé mais par les méthodes de cet objet. Ce qui veut dire que devrait rappeler la définition de votre classe, avant la commande unserialize pour que votre objet soit utilisable.

```
<?php
classa.inc:
  class A {
    var $one = 1;
    function show_one() {
      echo $this->one;
    }
  }
?>
page1.php:
<?php
  include("classa.inc");
  $a = new A;
  $s = serialize($a);
  // enregistrez $s où la page2.php pourra le trouver.
  $fp = fopen("store", "w");
  fputs($fp, $s);
  fclose($fp);
?>
page2.php:
<?php
  // Ceci est nécessaire pour que unserialize() fonctionne correctement
  include("classa.inc");
  $s = implode("", @file("store"));
  unserialize($s);
  // maintenant, utilisez la méthode show_one de l'objet $a.
  $a->show_one();
?>
```

Php et la POO : Les fonctions magiques __sleep et __wakeup

serialize s'assure que votre classe a une méthode avec le nom magique __sleep . Si c'est le cas, cette fonction est appelée avant toute linéarisation. Elle peut alors nettoyer l'objet et on s'attend à ce qu'elle retourne un tableau avec la liste des noms de variables qui doivent être sauvées.

Le but de cette fonction __sleep est de fermer proprement toute connexion à une base de données, de valider les requêtes, de finaliser toutes les actions commencées. Cette fonction est aussi pratique si vous avez de très grands objets qui n'ont pas besoin d'être sauvé entièrement.

A l'inverse, unserialize s'assure de la présence de la fonction magique __wakeup . Si elle existe, cette fonction reconstruit toutes les ressources d'un objet.

Le but de cette fonction __wakeup est de rétablit toutes les connexions aux bases de données, et de recréer les variables qui n'ont pas été sauvées.

```
<?php
class Connection {
   protected $link;
   private $server, $username, $password, $db;
   public function __construct($server, $username, $password, $db)
   {
       $this->server = $server;
       $this->username = $username;
       $this->password = $password;
       $this->db = $db;
       $this->connect();
   }
   private function connect()
   {
       $this->link = mysql_connect($this->server, $this->username, $this->password);
       mysql_select_db($this->db, $this->link);
   }
   public function __sleep()
   {
       mysql_close($this->link);
   }
   public function __wakeup()
   {
       $this->connect();
   }
}
?>
```

Php et la POO : Classe abstraite et interface

Classe abstraite

Une classe abstraite est une classe dont toutes les méthodes ne sont pas définies. Elle ne peut être instanciée. Elle permet de *prototyper* les classes. Lors de l'héritage depuis une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parent doivent être définies par l'enfant ; de plus, ces méthodes doivent être définies avec la même (ou plus faible) visibilité . Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie en tant que protégée ou publique.

Interface

Une interface est le prototype d'une classe, elle n'implémente pas les méthodes. Une classe peut implémenter plusieurs interfaces. Pour implémenter une interface, l'opérateur « implements » est utilisé. Toutes les méthodes de l'interface doivent être implémentées dans une classe ; si ce n'est pas le cas, une erreur fatale sera émise. Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule.

Php et la POO : Classe abstraite et interface

```
<?
interface Vehicule {
(int) nombrepassager;
(int) nombreroue;
function demarrer();
function arreter();
}

abstract class vehiculemarin implements Vehicule {
    (int) nombreroue = 0;
}

abstract class vehiculeterre implements Vehicule {
    (int) nombreroue = 4;
}

class bateau extends véhiculemarin {
(int) nombrepassager = 5;
function demarrer() { ... };
function arreter() { ... };
}

class voiture extends vehiculeterre {
(int) nombrepassager = 6;
function demarrer() { ... };
function arreter() { ... };
}
?>
```

Php et la POO : Clonage de classe

Etant donné que les objets sont passés par référence et non plus par copie, il devient parfois nécessaire de pouvoir copier un objet explicitement, c'est le rôle du mot-clé clone.

```
class Dog { public $nick; }
 $a = new Dog();
 $a->nick = 'Raoul le pitbull';
 $b = clone $a;
 $b->nick = 'Hubert le cocker';
 echo $a->nick."\n";
 echo $b->nick."\n";
 echo "--------------------\n";

class Paper {
     public $type = 'original';
     function __clone()
     { $this->type = 'copie'; }
}
$c = new Paper();
$d = clone $c;
echo $c->type."\n";
echo $d->type."\n";
```

```
Raoul le pitbull
Hubert le cocker
--------------------
original
copie
```

Php et la POO : Objets retournés

En PHP4, lorsqu'une fonction retourne un objet, il est impossible de directement l'exploiter et d'appeler ses méthodes. Ceci est résolu en PHP5 et fonctionne correctement.

```
class Dog {
        public $nick;
        function __construct($nick) {
        $this->nick = $nick;
        }
}
class Master {
        private $dog;
        function __construct() {
        $this->dog = new Dog('Raoul le pitbull');
        }
        function getDog() {
        return $this->dog; }
        }
        $m = new Master();

echo $m->getDog()->nick;
```

Php et la POO :

PHP 5 introduit le mot-clé " final " qui empêche les classes filles de surcharger une méthode en en préfixant la définition par le mot-clé " final ". Si la classe elle-même est définie comme finale, elle ne pourra pas être étendue.

Lors de l'utilisation de l'opérateur de comparaison « == » , les objets sont comparées de manière simple, à savoir : deux objets sont égaux s'ils ont les mêmes attributs et valeurs, et qu'ils sont des instances de la même classe.

D'un autre coté, lors de l'utilisation de l'opérateur d'identité (===), les objets sont identiques uniquement s'ils font référence à la même instance de la même classe.

Auto-chargement de classes

Vous pouvez définir la fonction __autoload qui va automatiquement être appelée si une classe n'est pas encore définie au moment de son utilisation. Grâce à elle, vous avez une dernière chance pour inclure une définition de classe, avant que PHP ne déclare une erreur.

```
<?php
function _autoload($class_name) {
    require_once $class_name . '.php';
}

?>
```

Php et la POO :

Parcours d'objet simple

```
<?php
class MyClass
{
  public $var1 = 'valeur 1';
  public $var2 = 'valeur 2';
  public $var3 = 'valeur 3';

  protected $protected = 'variable protégée';
  private   $private   = 'variable privée';

  function iterateVisible() {
     echo "MyClass::iterateVisible:\n";
     foreach($this as $key => $value) {
         print "$key => $value\n";
     }
  }
}
$class = new MyClass();
foreach($class as $key => $value) {
    print "$key => $value\n";
}
echo "\n";
$class->iterateVisible(); L'exemple ci-dessus va afficher :
var1 => valeur 1
var2 => valeur 2
var3 => valeur 3

MyClass::iterateVisible:
var1 => valeur 1
var2 => valeur 2
var3 => valeur 3
protected => variable protégée
private => variable privée
```

Php et la POO : Exceptions

Les exceptions permettent d'arrêter le fonctionnement normal d'un traitement afin d'en signaler les erreurs. Cependant, le fonctionnement des exceptions n'est pas idyllique comparé aux autres langages. N'importe quelle classe peut être envoyée par le mécanisme des exceptions, pas seulement celle qui hérite de la classe Exception.

Le mot-clé finaly n'existe pas.

Les erreurs classiques (division par zéro, fichier inexistant, etc..) ne génèrent aucune exception.

Il peut y avoir plusieurs blocs catch. Si l'exception n'est pas récupérée, elle remonte jusqu'à l'arrêt complet du script.

```
try {
        $error = "Toujours lancer cette erreur";
        throw new Exception($error);
        echo "Jamais exécuté\n";
}
catch (Exception $e) {
        echo "Capture de l'exception : ".$e->getMessage()."\n";
 }

echo "Reprise du fonctionnement normal\n";
```

Php et la POO : Exceptions étendues

Une classe Exception définie par l'utilisateur peut être définie en étendant la classe Exception interne. Les membres et les propriétés suivantes montrent ce qui est accessible dans la classe enfant qui est dérivée de la classe exception interne.

```
<?php
class Exception
{
  protected $message = 'exception inconnu'; // message de l'exception
  protected $code = 0;    //code de l'exception défini par l'utilisateur
  protected $file;              // nom du fichier source de l'exception
  protected $line;                 // ligne de la source de l'exception

  function __construct(string $message=NULL, int code=0);

  final function getMessage();              // message de l'exception
  final function getCode();                 // code de l'exception
  final function getFile();                 // nom du fichier source
  final function getLine();                 // ligne du fichier source
  final function getTrace();                // un tableau de backtrace()
  final function getTraceAsString();        // chaîne formattée de trace

  /* Remplacable */
  function _toString();
  // chaîne formatée pour l'affichage
}
?>
```

Php et la POO : Exceptions étendues

Si une classe étend la classe Exception interne et redéfinit le constructeur , il est vivement recommandé qu'elle appelle aussi parent::__construct() pour s'assurer que toutes les données disponibles ont été proprement assignées. La méthode __toString() peut être réécrite pour fournir un affichage personnalisé lorsque l'objet est présenté comme une chaîne.

```
<?php
/**Définition d'une classe d'exception personnalisée**/
class MyException extends Exception
{
  // Redéfinissez l'exception ainsi le message n'est pas facultatif
  public function __construct($message, $code = 0) {
  // traitement personnalisé que vous voulez réaliser ...
 // assurez-vous que tout a été assigné proprement
  parent::__construct($message, $code);
}
  // chaîne personnalisé représentant l'objet
  public function __toString() {
    return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
  }
  public function customFunction() {
    echo "Une fonction personnalisée pour ce type d'exception\n";
  }
}
```

Php et la POO : Exceptions étendues

```
/*** Création d'une classe pour tester l'exception***/
class TestException
{  public $var;
 const THROW_NONE    = 0;
  const THROW_CUSTOM  = 1;
  const THROW_DEFAULT = 2;
 function __construct($avalue = self::THROW_NONE) {
    switch ($avalue) {
      case self::THROW_CUSTOM:
        // jète une exception personnalisé
        throw new MyException('1 est un paramètre invalide', 5);
        break;
      case self::THROW_DEFAULT:
        // jète l'exception par défaut.
        throw new Exception('2 n\'est pas autorisé en tant que paramètre', 6);
        break;
      default:
        // Aucune exception, l'objet sera créé.
        $this->var = $avalue;
        break;
    }
  }
}
```

Php et la POO : Exceptions étendues

```
// Exemple 1
try {  $o = new TestException(TestException::THROW_CUSTOM); }
catch (MyException $e) {            // Devrait être attrapée
  echo "Capture mon exception\n", $e;
  $e->customFunction();
}
catch (Exception $e) {              // Sauté
  echo "Capture l'exception par défaut\n", $e;
}
// Continue l'exécution
var_dump($o);
echo "\n\n";
```

```
//Exemple 2
try { $o = new TestException(TestException::THROW_DEFAULT); }
catch (MyException $e) {            // Ne correspond pas à ce type
  echo "Capture mon exception\n", $e;
  $e->customFunction();
}
catch (Exception $e) {              // Devrait être attrapée
  echo "Capture l'exception par défaut\n", $e;
}
// Continue l'exécution
var_dump($o);
echo "\n\n";
```

Php et la POO : Exceptions étendues

```
// Exemple 3
try { $o = new TestException(TestException::THROW_CUSTOM); }
catch (Exception $e) {                  // Devrait être attrapée
  echo "Capture l'exception par défaut\n", $e;
}

// Continue l'exécution
var_dump($o);
echo "\n\n";
```

```
// Exemple 4
try {  $o = new TestException(); }
catch (Exception $e) {                  // sauté, aucune exception
  echo "Capture l'exception par défaut\n", $e;
}

// Continue l'exécution
var_dump($o);
echo "\n\n";
?>
```