PHP : La couche Objet

L3 Informatique - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr

Janvier 2019

L'adoption de la programmation orientée objet (POO) est une tendance lourde en développement web et notamment en PHP

- Intérêts du paradigme pour les gros projets logiciels : modularisation, réutilisation, ...
- Les applications Web intègrent de multiples patrons de conception.
- Les cadriciels PHP sont orientés objet.
- De nombreuses extensions PHP (ne) proposent (que) des interfaces de programmation d'application (API) objet.

La couche Objet en PHP

Proche de Java (refonte totale en PHP 5)

- Accessibilité publique, protégée ou privée des propriétés et méthodes.
- Propriétés et méthodes statiques.
- Déréférencement.
- Héritage simple avec surchage implicite des méthodes et liaison statique tardive.
- Méthodes et classes finales.
- Traits, classes abstraites, interfaces.
- Classes anonymes.
- Clonage d'objets.

Particularités

- Méthodes magiques.

Création de classe

Recommandation

Définir chaque classe dans un fichier dédié à inclure par les scripts y faisant appel.

Création de classe

- Avec mot-clé `class`.
- Constantes, variables et méthodes déclarées avec la syntaxe usuelle.
- Valeurs par défaut autorisées pour les variables.
- Typage des méthodes autorisé.
- Spécificateur d'accès optionnel (`public` par défaut).

Création de classe

exemple9-2.php

```php
class Action {
    const PARIS="Palais Brognard";
    public $nom;
    public $cours;
    public $bourse="bourse de Paris ";
    public function info() {
        echo "Informations du ",date("d/m/Y H:i:s"),"<br/>";
        $now=getdate();
        $heure= $now["hours"];
        $jour= $now["wday"];
        echo "<h3>Horaires des cotations</h3>";
        if(($heure>=9 && $heure <=17)&& ($jour!=0 && $jour!=6)) {
            echo "La Bourse de Paris est ouverte <br/>";
        } else {
            echo "La Bourse de Paris est fermée <br/>";
        }
        if(($heure>=16 && $heure <=23)&& ($jour!=0 && $jour!=6) ) {
            echo "La Bourse de New York est ouverte <hr/>";
        } else {
            echo "La Bourse de New York est fermée <hr/>";
        }
    }
}
```

Création et utilisation d'objets

- Instanciation avec mot-clé `new` suivi du nom de la classe (litéral ou variable `string`).
- Accès en lecture/écriture aux propriétés et invocation de méthodes avec mot-clé `$this` et notation fléchée `->` (sans symbole `$` pour les variables).

exemple9-4.php

```php
require("exemple9-2.php");
$action= new Action();
$action->nom = "Mortendi";
$action->cours = 1.15;
echo "<b>L'action $action->nom cotée à la $action->bourse vaut $action->cours &euro;</b><hr/>";
$action->info();
echo "La structure de l'objet \$action est : <br/>";
var_dump($action);
echo "<h4>Descriptif de l'action</h4>";
foreach($action as $prop=>$valeur) {
  echo "$prop = $valeur <br/>";
}
if($action instanceof Action) {
      echo "<hr/>L'objet \$action est du type Action";
}
```

Accès aux variables dans les méthodes

- Avec `$this->` pour les variables propres.
- Littéralement pour les superglobales ou globales déclarées avec `global`.

exemple9-5.php

exemple9-6.php

```php
require('exemple9-5.php');
$client="Geelsen";
$mortendi = new Action();
$mortendi->nom ="Mortendi";
$mortendi->cours="1.76";
$mortendi->info();
```

Spécificateurs d'accessibilité

Trois niveaux d'accès aux propriétés

- `public` : accès universel (par défaut).
- `protected` : accès limité à la classe et ses dérivées.
- `private` : accès limité à la classe.

exemple9-6B.php

```php
class Acces {
  public $varpub ="Propriété publique";
  protected $varpro="Propriété protégée";
  private $varpriv="Propriété privée";
  function lireprop() {
    echo "Lecture publique : $this->varpub","<br/>";
    echo "Lecture protégée : $this->varpro","<br/>";
    echo "Lecture privée : $this->varpriv","<hr/>";
  }
}
$objet=new Acces();
$objet->lireprop();
echo $objet->varpub;
// echo $objet->varpriv; Erreur fatale
// echo $objet->varpro; Erreur fatale
```

Spécificateurs d'accessibilité

Trois niveaux d'accès aux méthodes

- `public` : à tout objet et toute instance (par défaut).
- `protected` : réservé aux instances de la classe et ses dérivées.
- `private` : réservé aux instances de la classe.

exemple9-7.php

```php
class Accesmeth {
   private $code="Mon code privé";
   private function lirepriv() {
        echo "Lire privée ",$this->code,"<br/>";
   }
   protected function lirepro() {
        echo "Lire protégée ",$this->code,"<br/>";
   }
   public function lirepub() {
        echo "Lire publique : ",$this->code,"<br/>";
        $this->lirepro();
        $this->lirepriv();
   }
}
$objet=new accesmeth();
$objet->lirepub();
//$objet->lirepro();//Erreur fatale
//$objet->lirepriv();//Erreur fatale
```

Propriétés et méthodes statiques

- Déclaration avec le mot-clé `static`.
- Référence avec `C::$p` et `C::m()` en dehors de la classe et `self::$p` et `self::m()` en dedans pour classe `C` de propriété `p` et méthode `m`.

exemple9-8.php

```php
class Info {
   public static $bourse="Bourse de Paris";
   public static function getheure() { return date("h : m : s"); }
   public static function afficheinfo() {
      return Info::$bourse.", il est ".self::getheure();
   }
}
echo Info::$bourse,"<br/>"; //Bourse de Paris
echo Info::getheure(),"<br/>";//02 : 01 : 41
echo Info::afficheinfo(),"<hr/>";//Bourse de Paris, il est 02 : 01 : 41
$objet=new Info();
/*
 * $objet->bourse="New York";//Notice: Accessing static property Info::$bourse as non static
 * echo "\$objet->bourse : ",$objet->bourse,"<hr />";//Notice ... New York
 */
/* Ajouter de préférence un setter :
 * public function setbourse($val) {info::$bourse=$val};
 */
```

Constructeur et destructeur

Constructeur

- Créé avec la méthode magique

  `void __construct(...)`
- Appelé automatiquement à l'instanciation avec `new`.
- Un constructeur au plus par classe.

Destructeur

- Créé avec la méthode magique (sans paramètres)

  `void __destruct()`
- Appelé automatiquement en fin de script ou par appel explicite à `unset(...)` sur l'instance s'il ne reste qu'une référence.
- Utile pour libérer des ressources (fermeture de fichier, connexion à BDD)
- Un destructeur au plus par classe.

Constructeur et destructeur

exemple9-9.php

```php
class Action {
   private $propnom;
   private $propcours;
   protected $propbourse;
   function __construct($nom,$cours,$bourse="Paris") {
      $this->propnom=$nom;
      $this->propcours=$cours;
      $this->propbourse=$bourse;
   }
   function __destruct() {
      echo "L'action $this->propnom n'existe plus!<br />";
   }
}
$alcotel = new Action("Alcotel",10.21);
$bouch = new Action("Bouch",9.11,"New York");
$bim = new Action("BIM",34.50,"New York");
$ref=&$bim;
unset($alcotel);//Appel au destructeur
unset($bim);//Pas d'appel au destructeur
echo "<hr/><h4> FIN du script </h4><hr/>";
```

Déréférencement de méthodes

Chaînage d'appels de méthodes retournant des objets

Syntaxe : `$x->m1()->m2()->...`

exemple9-10.php

```php
class Varchar {
   private $chaine;
   function __construct($a) {
      $this->chaine= (string)$a;
   }
   function add($addch) {
      $this->chaine.=$addch;
      return $this;
   }
   function getch() {
      return   $this->chaine;
   }
}
$texte=new Varchar("Apache ");
echo $texte->getch();//Apache
echo $texte->add( " PHP 7 ")->getch();//Apache  PHP 7
echo $texte->add(" MySQL ")->add("SQLite ")->getch();//Apache  PHP 7  MySQL SQLite
```

Ajout dynamique de propriétés à objets

Possibilité d'ajouter de nouvelles propriétés à une instance donnée

- Viole le principe d'encapsulation.
- Se protéger avec "surcharge" (overriding) de la méthode magique `__set`.

exemple9-11.php

```php
class Action {
   public $propnom;
   public $propcours;
   public $propbourse;
   function __construct($nom,$cours,$bourse) {
        $this->propnom=$nom;
        $this->propcours=$cours;
        $this->propbourse=$bourse;
   }
}
$bim = new Action("BIM",9.45,"New York");
var_dump($bim);
$bim->date="2017";
var_dump($bim);
echo "Propriété date : ",$bim->date;//2017
```

Héritage

Création de sous-classe

- Avec le mot-clé `extends`.

Référence aux membres de la classe parente en cas de surcharge

- Avec `parent::...`

exemple9-12.php

Constructeur et destructeur parent à appeler explicitement, si nécessaire

- Avec `parent::__construct(...)`, `parent::__destruct()`.

Héritage multiple interdit

- Utiliser traits et/ou interfaces multiples.

Traits

Alternative flexible aux classes abstraites

Un trait

- Peut contenir propriétés et méthodes.
- Ne peut pas être instancié.
- Création avec `trait nom-trait {...}`
- Peut être utilisé par n'importe quelle classe avec `use nom-trait;`
- Peut utiliser d'autres traits.

Une classe peut mixer plusieurs traits, tout en héritant d'une classe et en implémentant des interfaces ...

Traits

exemple9-13.php

```php
trait Marcher {
 public $pattes;
 function marche() { echo "Je marche sur ". $this->pattes." pattes<br/>"; }
}
trait Voler {
 public $ailes;
 function vole() { echo "Je vole avec ". $this->ailes." ailes <br/>"; }
}
trait Nager {
 function nage() { echo "Moi je sais nager<br/>"; }
}
class Cheval { use Marcher,Nager; }
class Oiseau { use Marcher,Voler; }
class Pegase { use Marcher,Nager,Voler; }
// Un aigle
$aigle=new Oiseau(); $aigle->pattes=2; $aigle->ailes=2;
echo "<h3>L'aigle: </h3>"; $aigle->marche(); $aigle->vole();
// Un cheval
$dada=new Cheval(); $dada->pattes=4;
echo "<h3>Le cheval : </h3>";
$dada->marche(); $dada->nage();
// Pégase, le cheval ailé
$chevalaile=new Pegase(); $chevalaile->ailes=2; $chevalaile->pattes=4;
echo "<h3>Pégase : </h3>";
$chevalaile->marche(); $chevalaile->vole(); $chevalaile->nage();
```

Traits - Collision de noms

Collision de noms

Si classe enfant, trait et classe parente contiennent des méthodes homonymes, la 1ère a priorité sur la 2nde, elle-même prioritaire sur la 3ème.

Collision de noms

Si une classe utilise deux traits contenant des méthodes homonymes, résolution par

- Hiérarchisation avec `trait1::nom-commun insteadof trait2;`
- Renommage (aliasing) avec `nom-trait::nom-commun as nom-alias;`

Traits : Hiérarchisation et renommage

exemple9-15.php

```php
trait UN {
 public function small($text) {
  echo "<small>trait UN : $text</small>"; }
 public function big($text) {
  echo "<h4>trait UN : $text</h4>";
 }
}
trait DEUX {
 public function small($text) {
  echo "<i>trait DEUX : $text </i>";
 }
 public function big($text) {
  echo "<h2>trait DEUX : $text </h2>";
 }
}
class Texte {
 use UN, DEUX {
  DEUX::small insteadof UN;
  UN::big insteadof DEUX;
  DEUX::big as gros;
  UN::small as petit;
 }
}
//test des traits
$a=new Texte();
$a->small("Méthode small");//trait DEUX : Méthode small
$a->big("Méthode big");//trait UN : Méthode big
$a->gros("Méthode gros");//trait DEUX : Méthode gros
$a->petit("Méthode petit");//trait UN : Méthode petit
```

Classe abstraite

Classe qui

- Ne peut pas être instanciée.
- Ne contient que des membres `public` ou `protected`.
- Peut contenir des méthodes abstraites (méthodes sans implémentation).
- Se crée avec `abstract class nom-classe {...}`
- S'hérite avec possibilité de relâcher les spécificateurs d'accès.
- S'hérite avec `class enfant extends parent {...}`

exemple9-16.php

Interface

Notion plus restrictive que les classes abstraites

Une interface :

- Ne contient aucune propriété.
- Ne définit que des méthodes abstraites publiques.
- Se crée avec `interface nom-interface {...}`
- S'implémente avec `class nom-classe implements nom-interface {...}`

Une classe peut implémenter plusieurs interfaces.

exemple9-17.php

Méthodes et classe finales, classes anonymes

Pour empêcher toute modification par héritage

Préfixer définition de méthode/classe par le mot-clé `final`.

Classe anonyme

A l'instar des fonctions anonymes, définition/affectation avec `$obj=new class(...) {...}`

exemple9-18.php

Résolution statique à la volée (late static binding)

En complément de la résolution dynamique (dynamic binding/dispatch) pour sélectionner l'implémentation d'une méthode statique polymorphe (e.g., méthode surchargée dans une classe enfant).

exemple9-15b.php

```php
class Pere {
   static public function info($nom) {
      static::affiche($nom);
   }
   static public function affiche($nom) {
      echo "<h3>Je suis le père $nom </h3>";
   }
}
class Fils extends Pere
{
   static public function affiche($nom) {
      echo "<h3>Je suis le fils $nom </h3>";
   }
}
Fils::info('Matthieu');
```

Clonage d'objets

Les objets sont passés par référence (objets-affectation.php)

```
1 class A { public $a; function __construct($a) { $this->a=$a; }
}
2 $a1 = new A("a1"); $a2 = $a1; echo $a2->a;//a1
3 $a2->a="a2"; echo $a2->a;//a2
4 echo $a1->a;//a2
```

Copie superficielle d'objet (shallow copy) par clonage

- Avec `$objetclone = clone $objet;`
- Les références internes de l'objet sont préservées.
- Possibilité d'ajuster le clonage (eg. réaliser une copie profonde) en définissant la méthode magique `__clone` appelée automatiquement via `clone`.

exemple9-19.php

Méthodes magiques

Méthodes appelées automatiquement selon le contexte et qui peuvent être redéfinies

Membre *inaccessible* = non déclaré ou non visible (i.e. `protected` ou `private`) dans le contexte courant.

`__construct`	Instantiation avec `new`.
`__destruct`	Destruction de la dernière référence.
`__clone`	Clonage avec `clone`.
`__set`	Accès en écriture à propriété inaccessible.
`__get`	Accès en lecture à propriété inaccessible.
`__isset`	Vérification avec `isset()` ou `empty()` de propriété inaccessible.
`__unset`	Suppression avec `unset()` de propriété inaccessible.
`__call`	Appel à méthode inaccessible.
`__callStatic`	Appel à méthode statique inaccessible.
`__sleep`	Appel avant sérialisation de l'objet.
`__wakeup`	Appel avant récupération des données sérialisées.
`__toString`	Appel en contexte `string`.

Méthodes magiques

exemple9-26.php

```php
class A {
    private $code ;
    public function __construct($v) {
        $this->code = $v;
    }
    public function __set($p, $v) {
        echo "Affectation de la valeur $v à la propriété $p\n ";
        $this->$p = $v;
    }
    public function __get($p) {
        return $this->$p;
    }
    public function __isset($p) {
        if(isset($this->$p))
            echo "La propriété $p est définie\n";
        else
            echo "La propriété $p n'est pas définie\n";
    }
    public function __unset($p) {
        echo "Effacement de la propriété $p\n";
        unset($this->$p);
    }
}
$obj = new A('AZERTY');
echo isset($obj->code); //La propriété code est définie
echo "code = ", $obj->code, "\n"; //code = AZERTY
$obj->code="QWERTY"; //Affectation de la valeur QWERTY à la propriété code
echo "code = ", $obj->code, "\n"; //code = QWERTY
unset($obj->code);
echo "code = ", $obj->code, "\n"; //PHP Notice:  Undefined property: A::$code
```

Veiller à interdire la définition de nouvelle propriété via __set

set.php

```php
class A {
   protected $a;
   public function __construct() {
      $this->a=1;
   }
   public function __set($name, $value) {
      // attention on utilise ->$name donc
      // création possible de nouvelles propriétés
      $this->$name=$value;
   }
   public function __toString() {
      $t=get_object_vars($this);
      $s="";
      foreach($t as $key => $value) $s.="$key = $value\n";
      return $s;
   }
}
$a1=new A();
$a1->a=2;
$a1->b=3;  // création attribut $b pour $a1 !
echo $a1."\n";  // a=2, b=3
```

Test de type et d'héritage

Vérifier si une variable est l'instance d'une classe

- Avec mot-clé `instanceof`.

Vérifier si une variable est l'instance d'une sous-classe ou si une relation d'héritage existe entre classes

- Avec mot-clé `is_subclass_of`.

instanceof.php

```php
class A {}
class B extends A {}
$p=new A();
$e=new B();
if ($p instanceof A)
    echo var_dump($p), "is a A\n";//object(A)#1 (0) {} is a A
if ($e instanceof A)
    echo var_dump($e), "is a A\n";//object(B)#2 (0) {} is a A
if (is_subclass_of($e,'A'))
    echo var_dump($e), "instance of subclass of A\n";
    //object(B)#2 (0) {} instance of subclass of A
if (is_subclass_of('B','A'))
    echo "B subclass of A\n";//B subclass of A
```