

PHP : La couche Objet

L2 MPCIE - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr



Janvier 2018

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 cadres 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 surcharge 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

```
1 class Action {
2     const PARIS="Palais Brognard";
3     public $nom;
4     public $scours;
5     public $bourse="bourse de Paris ";
6     public function info() {
7         echo "Informations du ",date("d/m/Y H:i:s"), "<br/>";
8         $now=getdate();
9         $heure= $now["hours"];
10        $jour= $now["wday"];
11        echo "<h3>Horaires des cotations</h3>";
12        if(($heure>=9 && $heure <=17)&& ($jour!=0 && $jour!=6)) {
13            echo "La Bourse de Paris est ouverte <br/>";
14        } else {
15            echo "La Bourse de Paris est fermée <br/>";
16        }
17        if(($heure>=16 && $heure <=23)&& ($jour!=0 && $jour!=6) ) {
18            echo "La Bourse de New York est ouverte <hr/>";
19        } else {
20            echo "La Bourse de New York est fermée <hr/>";
21        }
22    }
23 }
```

Création et utilisation d'objets

- Instanciation avec mot-clé `new` suivi du nom de la classe (littéral 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

```
1 require ("exemple9-2.php");
2 $action= new Action();
3 $action->nom = "Mortendi";
4 $action->cours = 1.15;
5 echo "<b>L'action $action->nom cotée à la $action->bourse vaut $action->cours
&euro; </b><hr/>";
6 $action->info();
7 echo "La structure de l'objet \$action est : <br/>";
8 var_dump($action);
9 echo "<h4>Descriptif de l'action</h4>";
10 foreach ($action as $prop=>$valeur) {
11     echo "$prop = $valeur <br/>";
12 }
13 if ($action instanceof Action) {
14     echo "<hr/>L'objet \$action est du type Action";
15 }
```

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

```
1 require ('exemple9-5.php');
2 $client="Geelsen";
3 $mortendi = new Action();
4 $mortendi->nom = "Mortendi";
5 $mortendi->cours="1.76";
6 $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

```
1 class Acces {
2     public $varpub ="Propriété publique";
3     protected $varpro ="Propriété protégée";
4     private $varpriv ="Propriété privée";
5     function lireprop () {
6         echo "Lecture publique : $this->varpub", "<br/>";
7         echo "Lecture protégée : $this->varpro", "<br/>";
8         echo "Lecture privée : $this->varpriv", "<hr/>";
9     }
10 }
11 $objet=new Acces ();
12 $objet->lireprop();
13 echo $objet->varpub;
14 // echo $objet->varpriv; Erreur fatale
15 // 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

```
1 class Accesmeth {
2     private $code="Mon code privé";
3     private function lirepriv() {
4         echo "Lire privée ", $this->code, "<br/>";
5     }
6     protected function lirepro() {
7         echo "Lire protégée ", $this->code, "<br/>";
8     }
9     public function lirepub() {
10        echo "Lire publique : ", $this->code, "<br/>";
11        $this->lirepro();
12        $this->lirepriv();
13    }
14 }
15 $objet=new accesmeth();
16 $objet->lirepub();
17 // $objet-> lirepro();//Erreur fatale
18 // $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

```
1 class Info {
2     public static $bourse="Bourse de Paris";
3     public static function getheure() { return date("h : m : s"); }
4     public static function afficheinfo() {
5         return Info::$bourse . ", il est " . self::getheure();
6     }
7 }
8 echo Info::$bourse, "<br/>"; //Bourse de Paris
9 echo Info::getheure(), "<br/>"; //02 : 01 : 41
10 echo Info::afficheinfo(), "<hr/>"; //Bourse de Paris, il est 02 : 01 : 41
11 $objet=new Info();
12 /*
13  * $objet->bourse="New York";//Notice: Accessing static property Info::$bourse as non static
14  * echo "\$objet->bourse : ",$objet->bourse,"<hr />";//Notice ... New York
15  */
16 /* Ajouter de préférence un setter :
17  * public function setbourse($val) {info::$bourse=$val};
18  */
```

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

```
1 class Action {
2     private $propnom;
3     private $propcours;
4     protected $propbourse;
5     function __construct ($nom, $cours, $bourse="Paris") {
6         $this->propnom=$nom;
7         $this->propcours=$cours;
8         $this->propbourse=$bourse;
9     }
10    function __destruct () {
11        echo "L'action $this->propnom n'existe plus!<br />";
12    }
13 }
14 $alcotel = new Action("Alcotel",10.21);
15 $bouch = new Action("Bouch",9.11,"New York");
16 $bim = new Action("BIM",34.50,"New York");
17 $ref=&$bim;
18 unset($alcotel) ;//Appel au destructeur
19 unset($bim) ;//Pas d'appel au destructeur
20 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

```
1 class Varchar {
2     private $chaine;
3     function __construct($a) {
4         $this->chaine= (string)$a;
5     }
6     function add($addch) {
7         $this->chaine.=$addch;
8         return $this;
9     }
10    function getch() {
11        return $this->chaine;
12    }
13 }
14 $texte=new Varchar("Apache ");
15 echo $texte->getch();//Apache
16 echo $texte->add( " PHP 7 ")->getch();//Apache PHP 7
17 echo $texte->add(" MySQL ")->add("SQLite ")->getch();//Apache PHP 7 MySQL SQLite
```

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.

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

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.

<code>__construct</code>	Instantiation avec <code>new</code> .
<code>__destruct</code>	Destruction de la dernière référence.
<code>__clone</code>	Clonage avec <code>clone</code> .
<code>__set</code>	Accès en écriture à propriétés <i>inaccessibles</i> .
<code>__get</code>	Accès en lecture à propriétés <i>inaccessibles</i> .
<code>__isset</code>	Vérification avec <code>isset()</code> ou <code>empty()</code> de propriétés <i>inaccessibles</i> .
<code>__unset</code>	Suppression avec <code>unset()</code> de propriétés <i>inaccessibles</i> .
<code>__call</code>	Appel à méthodes <i>inaccessibles</i> .
<code>__callStatic</code>	Appel à méthodes statiques <i>inaccessibles</i> .
<code>__sleep</code>	Appel avant sérialisation.
<code>__wakeup</code>	Appel avant récupération des données sérialisées.
<code>__toString</code>	Appel en contexte <code>string</code> .

Méthodes magiques

exemple9-26.php

```
1 class A {
2     private $code ;
3     public function __construct($v) {
4         $this->code = $v;
5     }
6     public function __set($p, $v) {
7         echo "Affectation de la valeur $v à la propriété $p\n ";
8         $this->$p = $v;
9     }
10    public function __get($p) {
11        return $this->$p;
12    }
13    public function __isset($p) {
14        if (isset($this->$p))
15            echo "La propriété $p est définie\n";
16        else
17            echo "La propriété $p n'est pas définie\n";
18    }
19    public function __unset($p) {
20        echo "Effacement de la propriété $p\n";
21        unset($this->$p);
22    }
23 }
24 $obj = new A('AZERTY');
25 echo isset($obj->code); //La propriété code est définie
26 echo "code = ", $obj->code, "\n"; //code = AZERTY
27 $obj->code="QWERTY"; //Affectation de la valeur QWERTY à la propriété code
28 echo "code = ", $obj->code, "\n"; //code = QWERTY
29 unset($obj->code);
30 echo "code = ", $obj->code, "\n"; //PHP Notice:  Undefined property: A::$code
```

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

```
1 class A {}
2 class B extends A {}
3 $p=new A();
4 $e=new B();
5 if($p instanceof A)
6     echo var_dump($p), "is a A\n";//object(A)#1 (0) {} is a A
7 if($e instanceof A)
8     echo var_dump($e), "is a A\n";//object(B)#2 (0) {} is a A
9 if(is_subclass_of($e,'A'))
10     echo var_dump($e), "instance of subclass of A\n";
11     //object(B)#2 (0) {} instance of subclass of A
12 if(is_subclass_of('B','A'))
13     echo "B subclass of A\n";//B subclass of A
```