PHP : La couche Objet

L2 MPCIE - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr

Janvier 2019

Développement Web et programmation orientée objet

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

etudiant.php

```php
class Etudiant
{
    public $nom;
    public function __construct($n)
    {
        $this->nom = $n;
    }
    public function getNom()
    {
        return $this->nom;
    }
    public function setNom($n)
    {
        $this->nom = $n;
    }
}
```

Instanciation et accès

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).

Accès aux variables dans les méthodes

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

Instanciation et accès

etudiant.php

```php
class Etudiant
{
    public $nom;
    public function __construct($n)
    {
        $this->nom = $n;
    }
    public function getNom()
    {
        return $this->nom;
    }
    public function setNom($n)
    {
        $this->nom = $n;
    }
}
```

instantiation-acces.php

```php
require 'etudiant.php';
$e = new Etudiant("Turing"); // appel implicite au constructeur magique __construct
echo gettype($e) . "<br/>"; // object
($e instanceof Etudiant) ? "\$e est de classe Etudiant<br/>" : "";
echo $e->getNom() . "<br/>"; // accès à propriété via accesseur ("getter")
$e->setNom("von Neumann"); // modification de propriété via mutateur ("setter")
echo $e->getNom() . "<br/>";
$e->nom = "Knuth"; // accès direct à propriété publique : OK
echo $e->getNom();
```

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.

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.

Spécificateurs d'accessibilité

private.php

```php
class Etudiant
{
    private $nom;
    public function __construct($n)
    {
        $this->nom = $n;
    }
    public function getNom()
    {
        $this->format();
        return $this->nom;
    }
    public function setNom($n)
    {
        $this->nom = $n;
    }
    public function __toString()
    {
        return " Nom : " . $this->nom;
    }

    private function format() {
        $this-> nom = ucwords($this->nom);
    }
}
$etudiant = new Etudiant("Turing");
//$etudiant->nom = "von Neumann"; // accès à propriété privée : KO
$etudiant->setNom("von Neumann");  // modification de propriété privée via mutateur
echo $etudiant;
// $etudiant->format(); // accès à méthode privée : KO
```

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`.

static.php

```php
class Etudiant {
    public static $effectif = 0;
    private $nom;
    public function __construct($n) {
        $this->nom = $n;
        self::$effectif += 1;
    }
    public function __toString() {
        return "Effectif : " . self::$effectif . " Nom : " . $this->nom;
    }
    public static function getEffectif() {
        return self::$effectif;
    }
}
$etudiant = new Etudiant("Turing");
echo $etudiant."<br/>";
echo  Etudiant::getEffectif(); // accès à méthode statique publique
```

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

destruct.php

```php
class Etudiant {
    private $nom;
    function __construct($nom = "tutdinea") {
        $this->nom = $nom;
    }
    function __destruct() {
        echo "L'étudiant $this->nom n'existe plus!<br />";
    }
}
$e1 = new Etudiant("E1");
$e2 = new Etudiant("E2");
$e3 = new Etudiant("E3");
$refsure2 = & $e2;
unset($e1); // Appel au destructeur
unset($e2); // Pas d'appel au destructeur : retse la référence $refsure2
echo "Fin du script"; // suivi d'appel au destructeur sur $2 et $e3
```

Déréférencement de méthodes

Chaînage d'appels de méthodes retournant des objets

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

dereferencement.php

```php
class A {
   private $chaine;
   function __construct($a) {
      $this->chaine= (string) $a[0];
   }
   function add($ch) {
      $this->chaine .= $ch[0];
      return $this;
   }
   function __toString() {
      return   $this->chaine;
   }
}
$lamp=new A("Linux");
echo $lamp->add("Apache")->add("MySQL")->add("PHP");//LAMP
```

Héritage

Création de sous-classe

- Avec le mot clé `extends`.

Référence aux membres de la classe parente en cas de surcharge

- Avec `parent::...`

Constructeur et destructeur parent à appeler explicitement, si nécessaire

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

Héritage multiple interdit

- Utiliser traits et/ou interfaces multiples.

Héritage

heritage.php

```php
class Etudiant {
    private $nom;
    protected function __construct($n) {
        $this->nom = $n;
    }
    protected function getNom() {
        return $this->nom;
    }
    public function __toString() { // doit être publique
        return " Nom : " . $this->nom;
    }
}
class Alternant extends Etudiant {
    private $entreprise = "ACME";
    public function __construct($n, $e) {
        parent::__construct($n);
        $this->entreprise = $e;
    }
    public function getNom() {
        return parent::getNom();
    }
    public function getEntreprise() {
        return $this->entreprise;
    }
    public function __toString() {
        return parent::__toString() . " Entreprise : {$this->entreprise}";
    }
}
$alt = new Alternant("Jobs", "Apple");
echo $alt->getNom()."<br/>";
echo $alt;
```

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

```
1  class A {
2        public $a;
3        function __construct($a) {
4              $this->a = $a;
5        }
6  }
7  $a1 = new A("a1");
8  $a2 = $a1;
9  echo $a2->a; // a1
10 $a2->a = "a2";
11 echo $a2->a; // a2
12 echo $a1->a; // a2
```

Clonage d'objets

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 propriétés inaccessibles.
`__get`	Accès en lecture à propriétés inaccessibles.
`__isset`	Vérification avec `isset()` ou `empty()` de propriétés inaccessibles.
`__unset`	Supression avec `unset()` de propriétés inaccessibles.
`__call`	Appel à méthodes inaccessibles.
`__callStatic`	Appel à méthodes statiques inaccessibles.
`__sleep`	Appel avant sérialisation.
`__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
```

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
```