Partie 4 — Programmation Orientée-Objet

Syntaxe

- Une classe s'écrit au moyen du mot `class` suivi du nom de la classe et d'accolades (pas de point virgule).
- Disponibilité des mots clefs `private`, `protected` et `public`.
- Disponibilité de la variable `$this` à l'intérieur d'un classe pour référencer l'objet lui-même
- Résolution de portée (*Paamayim Nekudotayim*) : `parent::` et `self::`, pour accéder aux membres statiques, constants, ou à ce qui a été redéfini (cf. exemple à suivre).
- Recommandation de convention : utiliser l'underscore pour marquer l'héritage :

```
class Arme{}
class Arme_Epee extends Arme{}
class Arme_Epee_Claymore extends Arme_Epee{}
```

Un exemple

```php
<?php
class Insecte {
    protected function faireDuBruit() {
        return 'Frrr';
    }
}
class Insecte_Bourdon extends Insecte {
    protected function faireDuBruit() {
        return 'Bzzz';
    }
    public function identifierParent() {
        return parent::faireDuBruit();
    }
    public function identifierSelf() {
        return self::faireDuBruit();
    }
}

$leon = new Insecte_Bourdon();
echo $leon->identifierParent(); // Frrr
echo $leon->identifierSelf();   // Bzzz
```

- `self` cherche en premier lieu dans la classe courante, avant d'aller chercher dans la classe parent.
- Lorsqu'une classe étendue redéfinit une méthode de la classe parente, PHP n'appelera pas la méthode d'origine. Il appartient à la méthode dérivée d'appeler la méthode d'origine en cas de besoin (voir exemple précédent).
- Un appel à une méthode ou un attribut `static` s'écrit : `NomDeClasse::$attribut` (voir exemple suivant).

Variables et méthodes statiques (ou de classe)

```php
<?php
class Sanglier{
    public static $sangliers = 0;
    public function __construct(){
        ++self::$sangliers;
    }
}

$blip = new Sanglier();
$blop = new Sanglier();
echo Sanglier::$sangliers;    //affiche "2"
```

Interface

Définition

Ensemble de méthodes que les classes souhaitant l'implanter doivent définir.

- `print_r(get_declared_interfaces())` : obtenir la liste des interfaces déclarées.
- Ne pas implanter toutes les méthodes de l'interface cause une erreur fatale :

```
<?php
interface Arme{
    // une classe qui veut implanter l'interface Arme doit définir la méthode faireMal()
    public function faireMal();
}

class Hache implements Arme{} //illégal car pas de définition de faireMal()
$h = new Hache();
$h->faireMal();  // Fatal error: Class Arme contains 1 abstract method and must
                    therefore be declared abstract or implement the remaining methods
```

Classes abstraites et finales

- Une classe *abstraite* ne peut être instanciée, et doit donc être dérivée pour être utilisée.
- Contrairement à une interface, la classe abstraite peut contenir du code (et non seulement des signatures de méthodes).

```
abstract class muche{ ... }
class truc extends muche { ... }
```

- Inversement, une classe *finale* est quant à elle instanciable, mais non extensible.
- De manière similaire, une méthode peut être déclaré finale ce qui empêche toute surcharge.

```
class parentClass {
      final private function someMethod() { }
}
class childClass extends parentClass {
   private function someMethod() { }
}
// Fatal error, cannot override final method, or class
```

Polymorphisme d'héritage

- Le polymorphisme d'héritage (ou *spécialisation/overriding*) consiste à redéfinir les méthodes existantes (adaptation comportementale).
- Par contre, la *surcharge* d'opérateur ou la définition de méthodes de même nom mais de prototypes différents ne sont pas supportées par PHP 5.

```
class animal {
    function parler(){
        echo 'Frrr';
    }
}

class sanglier extends animal {
    function parler() {
        echo 'Gruiiik';
    }
}

$s = new Sanglier();
$s->parler();          // Gruiiik
```

Références

- Chaque utilisation de l'opérateur d'affectation (=) sur un objet créée une nouvelle référence vers l'adresse en mémoire de l'objet.

```
<?php
class Truc{
    public function __destruct(){
        echo 'Objet détruit<br />';
    }
}

$obj_1 = new Truc();
$obj_2 = $obj_1; // on créé une nouvelle référence vers l'objet
$obj_3 = $obj_2; // on créé une nouvelle référence vers l'objet

echo 'I<br />';
unset($obj_1);  //il reste deux références, l'objet persiste donc en mémoire
echo 'II<br />';
unset($obj_2); //il reste une référence, l'objet persiste donc en mémoire
echo 'III<br />';
unset($obj_3);
?>
I
II
III
Objet détruit // destruction de la troisième et dernière référence
```

Références et clônage

- Le mot clé `clone` permet de créer une copie de l'objet original et non pas une référence. Chaque copie est un objet indépendant des autres, sa destruction ne concerne donc que lui.

```
<?php
class Truc{
    public function __destruct(){
        echo 'Objet détruit<br />';
    }
}

$obj_1 = new Truc();
$obj_2 = clone $obj_1;
$obj_3 = clone $obj_2;

echo 'I<br/>';
unset($obj_1);
echo 'II<br/>';
unset($obj_2);
echo 'III<br/>';
?>
I
Objet détruit
II
Objet détruit
III
Objet détruit
```

Exceptions

```
try {
    throw new Exception('incident');
}
catch(Exception $e) {
    echo $e->getMessage(); //affiche «incident»
}
```

Mots clefs réservés

`class`	Déclaration de classe
`const`	Déclaration de constante de classe
`function`	Déclaration d'une méthode
`public/protected/private`	Accès (par défaut `public` si aucun accès n'est explicitement défini)
`new`	Création d'objet
`self`	Résolution de portée (la classe elle-même)
`parent`	Résolution de portée (la classe `parent`)
`static`	Résolution de portée (appel statique) disponible depuis PHP 5.3 et 6.0
`extends`	Héritage de classe
`implements`	Implantation d'une interface (dont il faut redéclarer toutes les méthodes)

Méthodes magiques I

`__construct()` Constructeur de la classe

`__destruct()` Destructeur de la classe

`__set()` Déclenchée lors de l'accès en écriture à une propriété de l'objet

`__get()` Déclenchée lors de l'accès en lecture à une propriété de l'objet

`__call()` Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel non statique)

`__callstatic()` Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel statique) : disponible depuis PHP 5.3 et 6.0

`__isset()` Déclenchée si on applique `isset()` à une propriété de l'objet

Méthodes magiques II

`__unset()` Déclenchée si on applique `unset()` à une propriété de l'objet

`__sleep()` Exécutée si la fonction `serialize()` est appliquée à l'objet

`__wakeup()` Exécutée si la fonction `unserialize()` est appliquée à l'objet

`__toString()` Appelée lorsque l'on essaie d'afficher directement l'objet : `echo $object`

`__set_state()` Méthode statique lancée lorsque l'on applique la fonction `var_export()` à l'objet

`__clone()` Appelé lorsque l'on essaie de cloner l'objet

`__autoload()` Cette fonction n'est pas une méthode, elle est déclarée dans le scope global et permet d'automatiser les `include/require` de classes PHP

Fonctions utiles I

`class_parents()` Retourne un tableau de la classe parent et de tous ses parents

`class_implements()` Retourne un tableau de toutes les interfaces implantées par la classe et par tous ses parents

`get_class()` Retourne la classe de l'objet passé en paramètre

`get_called_class()` À utiliser dans une classe, retourne la classe appelée explicitement dans le code PHP et non au sein de la classe

`class_exists()` Vérifie qu'une classe a été définie

`get_class()` Retourne la classe d'un objet

`get_declared_classes()` Liste des classes définies

`get_class_methods()` Liste des méthodes d'une classe

`get_class_vars()` Liste des propriétés d'une classe

Constantes magiques

`__CLASS__` Donne le nom de la classe en cours

`__METHOD__` Donne le nom de la méthode en cours (équivalent de `get_class($this)`)

POO et PDO (exemple : sélection d'utilsateurs

- L'opération de sélection mobilise toujours deux méthodes à la suite : `execute()` et `fetchAll()`.
- Ces comportements peuvent être réunis dans une méthode d'une nouvelle classe.
- De plus saisir à chaque fois les informations de connexion est inutile.
- C'est le principe *DRY : Don't Repeat Yourself* : se répéter est 1) embêtant 2) source d'erreurs. La POO sert à en partie à ça : la « mise en facteur du code ».

Classe dérivant de PDO

```
class MyPDO extends PDO {

    public function __construct($dsn, $user=NULL, $password=NULL) {
        parent::__construct($dsn, $user, $password);

        # Envoyer des exceptions plutôt que des erreurs PHP
        $this->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    }

    public function prepare($sql, $options=NULL) {
        $statement = parent::prepare($sql);

        // En cas de requête "SELECT"
        if(strpos(strtoupper($sql), 'SELECT') === 0) {
            $statement->setFetchMode(PDO::FETCH_ASSOC);
        }

        return $statement;
    }
}
```

Relation entre tables et classes I

```php
<?php
class UserTable {

    /* Requêtes préparées */
    private $selectById;
    private $selectByName;
    private $selectAll;
    private $insert;

    /* Constructeur */
    public function __construct($db) {

        $this->selectAll = $db->prepare(
            "SELECT id, name FROM user ORDER BY name, id");

        $this->selectById = $db->prepare(
            "SELECT id, name FROM user WHERE id = ?");

        $this->selectByName = $db->prepare(
            "SELECT id, name FROM user WHERE name LIKE ? ORDER BY name, id");

        $this->insert = $db->prepare(
            "INSERT INTO user (name, password) VALUES (:name, :password)");
```

Relation entre tables et classes II

```
    public function insert($name, $password) {
        $this->insert->execute(
        array(':name' => $name, ':password' => $password));
        return $this->insert->rowCount();
    }

    public function selectAll() {
       $this->selectAll->execute();
        return $this->selectAll->fetchAll();
    }

    public function selectById($id) {
        $this->selectById->execute(array($id));
        return $this->selectById->fetch();
    }

    public function selectByName($name) {
        $this->selectByName->execute(array('%'.$name.'%'));
        return $this->selectByName->fetchAll();
    }
}
```

POO et PDO I

Imaginons quelque chose comme cela

```
$db = new MyPDO(
'mysql:host=localhost;dbname=mabase', 'utilisateur', 'motdepasse'
);

$userTable = new UserTable($db);
$userTable->truncate();
$userTable->insert('Ulric', '4321');
$userTable->insert('Verena', '4321');
$userTable->insert('Manann', '4321');
```

```
print_r($userTable->selectById(1));

Array ( [id] => 1 [name] => Ulric )
```

POO et PDO II

```
print_r($userTable->selectByName('r'));
Array (
    [0] => Array ( [id] => 2 [name] => Verena )
    [1] => Array ( [id] => 1 [name] => Ulric )
)
```

```
print_r($userTable->selectAll());

Array (
    ...
)
```

Un pas vers les *frameworks*

- Des *frameworks* tels que *Symfony* ou *Zend Framework* reposent sur de tels concepts.
- À chaque table sont associées deux classes, l'une contenant les méthodes d'accès aux données au niveau de la table, l'autre représentant un tuple de la table.