

# PHP

## CM3-1 : Programmation orientée objet en PHP

Mickaël Martin Nevot

V1.0.0



Cette œuvre de [Mickaël Martin Nevot](#) est mise à disposition selon les termes de la [licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage à l'Identique 3.0 non transposé](#).

# PHP

- I. Présentation
- II. PHP I
- III. XML
- IV. Regexp
- V. PHP II
- VI. MySQL
- VII. **POO**
- VIII. PDO
- IX. Hacking
- X. PHP « avancé »

# Rappel : POO

- **Paradigme** ( $\neq$  méthodologie)
- Concept :
  - Objet :
    - **État** (attributs)
    - **Comportement** (méthodes)
    - Identité
  - **Encapsulation**
  - **Héritage**
  - **Polymorphisme**
- Langage de programmation :
  - Java, C++, Ada, **PHP**, C#, Objective C, Python, etc.

# Rappel : objet et classe

- **Objet** : un concept, une idée/entité du monde physique
  - Exemples : voiture, étudiant, chat, fenêtre, forme, etc.
- **Classe** : regroupe les objets de mêmes comportements
- **Instancier** : fabriquer un exemplaire d'un élément à partir d'un modèle qui lui sert en quelque sorte de moule
- Un objet est une instance de **classe**
- **Réification** : permet de transformer ou à transposer un concept en une entité informatique

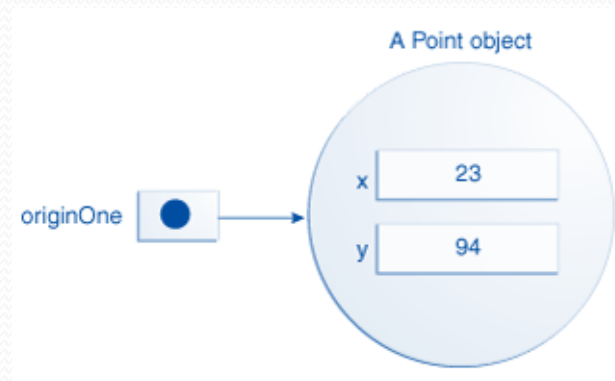
Une classe représente une responsabilité

# Rappel : état et comportement

- État :
  - Défini par l'ensemble des attributs
  - **Attribut**, variable d'instance, donnée membre :
    - Variable spécifique à l'objet
- Comportement :
  - Défini par l'ensemble de méthodes
  - **Méthode** : fonctions spécifique à l'objet
  - Méthodes :
    - **Constructeur** : appelé à la création de l'objet
    - **Destructeur** : appelé à la destruction d'un objet
    - Méthode abstraite : méthode sans code
    - Accesseurs et mutateurs : sert de mandataire d'accès à l'état de l'objet depuis l'extérieur de celui-ci

# Variable/méthode d'instance

- Variable d'instance (attribut) :
  - **Varie d'une instance (objet) à l'autre**
  - Initialisation (et même déclaration) facultative
- Fonction/méthode (passage par référence, depuis PHP 5+) :
  - Type primitif : ne modifie pas la valeur d'une variable
  - Objet : référence vers le même objet



Opérateur & implicite à partir de PHP5+

# Notation fléchée / this

- Notation fléchée :

- Pour accéder à un attribut :

```
$myObj->att1; // Accès à un attribut.
```

- Pour accéder à une méthode :

```
$myObj->meth1(); // Accès à une méthode.
```

```
$myObj->meth2(1, 2); // Accès à une méthode.
```

- Mot clef `$this` (lecture seule) :

- Désigne l'**objet courant** (celui dans lequel on code) :

```
$this->att1; // Accès à un attribut.
```

```
$this->meth1(); // Accès à une méthode.
```

# Constante de classe

- **Non modifiable** ← `≠ define(...)`
- Ne commence pas par un `$`
- Par convention : écrite en majuscule
- **Locale** à la classe
- Valeur scalaire uniquement
- Mot clef : `const`

```
class MyClass
{
    const CONSTANT = 'val';
    function showConstant() { echo self::CONSTANT; }
}
...
echo MyClass::CONSTANT;
```



# Statique

- **Variable de classe :**

- Donnée commune à tous les objets d'une même classe
- Existe même s'il n'y a aucune instance de la classe :

```
public static $att1; // Définition.
```

```
...
```

```
MyClass::$att1 = 3;
```

```
$myObj::$att1 = 3; // Fortement déconseillé !
```

- **Méthode de classe :**

- Ne s'intéresse pas à un objet particulier :

```
public static function meth1() { ... }; // Définition.
```

```
...
```

```
MyClass::meth1();
```

```
$myObj::meth1(); // Fortement déconseillé !
```

Fortement déconseillé avec `->` !

Fortement déconseillé avec `$this` !

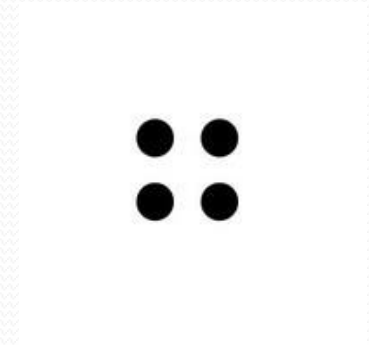
# Paamayim Nekudotayim /self

- Paamayim Nekudotayim

- **Opérateur de résolution de portée ::**

- Permet l'accès aux :

- Constantes
    - Variables de classe
    - Méthodes redéfinies (d'instance ou de classe)



- Mot clef `self` (lecture seule) :

- Désigne la **classe courante** (celle dans laquelle on code) :

```
self::$att1; // Accès à une variable de classe.
```

```
self::meth1(); // Accès à une méthode de classe.
```

# Méthodes magiques

- **Appels automatiques** : pas d'appel direct
- Constructeur : `__construct()`
- Destructeur : `__destruct()`
- Personnalisation du clonage : `__clone()`
  - Recopie de contenus d'objets :  
`$obj2 = clone $obj1;`
- Convertit l'objet en chaîne de caractères : `__toString()`
- Appel d'une méthode virtuelle : `__call($method, $args)`
- Sérialisation (et désérialisation) : `__sleep()`, `__wakeup()`

# Méthodes magiques

- **Surcharge magique :**

- Permet de **créer dynamiquement** des **attributs/méthodes**
- Méthodes appelées lors d'utilisations **non déclarées/visibles**
- À la lecture d'attribut : `__get()`
- À l'écriture d'attribut : `__set()`
- À l'appel de `isset()` ou `empty()` : `__isset()`
- À l'appel de `unset()` : `__unset()`

- **Autres méthodes magiques :**

- `__callStatic()` , `__debugInfo()` , `__set_state()` ,  
`__invoke()`

# Constructeur

- Mot clef : `__construct()`
- Méthode magique appelée à l'**instanciation d'un objet** pour initialiser son état
- Si aucun constructeur n'existe, un **constructeur par défaut** (sans paramètre et qui ne fait rien) est défini
- Mot clef `new` (création et allocation mémoire)

```
class MyClass
{
    function __construct($p) { ... }
    ...
}
...
// Création et allocation mémoire : instanciation.
$myObj1 = new \MyClass(5);
```

# Destructeur

- Mot clef : `__destruct()`
- Méthode magique appelée à la **destruction d'un objet**



L'usage d'un destructeur est assez rare

# Encapsulation

- Concerne : constructeur, attribut et méthode
- **Accessibilité** (visibilité) :
  - public (par défaut) : accessible de partout et sans aucune restriction
  - protected : accessible au sein de sa classe et à ses classes filles
  - private : accessible uniquement au sein de sa classe

- **Accesseurs/mutateurs** :

```
private $cpt;  
public function getCpt(){ return $this->cpt; }  
public function setCpt ($p){ $this->cpt = $p; }
```

Il est impossible d'affecter une visibilité à une classe

# Const., accesseurs, mutateurs

```
class Point
```

```
{  
    public function __construct(private double $x, private double $y)  
    {  
        ...  
    }  
}
```

Version moderne (PHP 8+)

La déclaration et l'initialisation des attributs sont faites automatiquement sans code supplémentaire

```
$point1 = new \Point(3.14, 42.0);  
var_dump($point1);
```

PHP OOP



# Héritage

- Héritage simple (une seule super-classe et unidirectionnelle)
- Mot clef `extends`
- Redéfinition :
  - Même signature de méthode (ne peut pas être moins accessible) :
    - Ajout/modification de paramètres possible s'ils sont optionnels
  - Réécriture du code

```
class MySuperClass {  
    function meth1($a) { instruction1 }  
}  
...  
class MyClass extends MySuperClass {  
    function meth1($a) { instruction2 }  
}
```

Il n'y a pas de surcharge en PHP

# Parent

- Permet de **réutiliser le code de la méthode de la super-classe**
- L'appel au constructeur de la super-classe n'est pas implicite, il faut le spécifier :

```
class MySuperClass {  
    public function __construct($a) { instruction1 }  
}  
...  
class MyClass extends MySuperClass {  
    public function __construct($a) {  
        parent::__construct($a - 1);  
        ...  
    }  
}
```

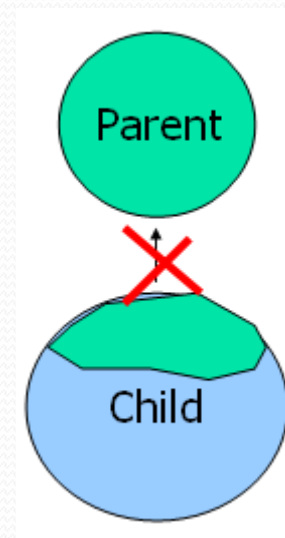
# Mot clef final

- Méthode (pour interdire toute redéfinition) :

```
public final function meth1() { ... }
```

- Classe (pour interdire tout héritage) :

```
final class MyClass { ... }
```



# Erreur et exception

- Signaux indiquant un cas exceptionnel :
  - Erreur : irrécupérable (arrêt de l'application)
  - Exception : récupérable (traitable)
- Exception :
  - Interrompt le flot d'exécutions normales
  - Traitement d'erreur :
    - Séparation du code normal/exceptionnel (lisibilité)
    - Récupération à un autre niveau (propagation dans la pile)
  - Si propagée jusqu'en haut de la pile : arrêt du script



# Gestion des erreurs

- Fixe le niveau de rapport d'erreurs : `error_reporting(...)`
- Spécifie une fonction utilisateur comme gestionnaire d'erreurs : `set_error_handler($handler, ...)`
- Déclenche une erreur utilisateur : `trigger_error($msg, ...)`
- Constantes prédéfinies :
  - Remarque : `E_NOTICE`
  - Avertissement : `E_WARNING`
  - Erreur : `E_ERROR`
  - Erreur d'analyse (fatales ?) : `E_PARSE`
  - Suggestions d'optimisation : `E_STRICT`
  - Alerte obsolète : `E_DEPRECATED`

# Gestion des exceptions

## • Mot clef

Pas de throws

- try : délimitation « d'usabilité » d'exceptions
- catch : capturer l'exception (traitement)
- finally : toujours exécuté après try ... catch ...
- throw : lancer l'exception (signalement)

```
function inverse(bool $x)
{
    if (!$x) { throw new \Exception('Div. par zero'); }
    else { return 1 / $x; }
}
try
{
    echo inverse(0);
}
catch (Exception $e)
{
    echo 'Exception : ' . $e->getMessage();
}
```

- [LogicException](#) (extends [Exception](#))
  - [BadFunctionCallException](#)
    - [BadMethodCallException](#)
  - [DomainException](#)
  - [InvalidArgumentException](#)
  - [LengthException](#)
  - [OutOfRangeException](#)
- [RuntimeException](#) (extends [Exception](#))
  - [OutOfBoundsException](#)
  - [OverflowException](#)
  - [RangeException](#)
  - [UnderflowException](#)
  - [UnexpectedValueException](#)

# Type et polymorphisme

- **Type :**

- Défini les valeurs qu'une donnée peut prendre
- Défini les opérateurs qui peuvent lui être appliqués
- Défini la syntaxe : « comment l'appeler ? »
- Défini la sémantique : « qu'est ce qu'il fait ? »
- Une classe est un type (composé), une interface aussi...

- **Polymorphisme :**

- Capacité d'un objet à avoir plusieurs types
- Permet d'utiliser une classe héritière comme une classe héritée

# Classe abstraite

- Ne peut pas être instanciée (mais constructeur possible)
- **Si une seule méthode est abstraite, la classe doit l'être**
- Abstraction possible à plusieurs niveaux d'héritage
- Méthodes : accessibilité `private` impossible
- Mot clef `abstract` :

- **Classe :**

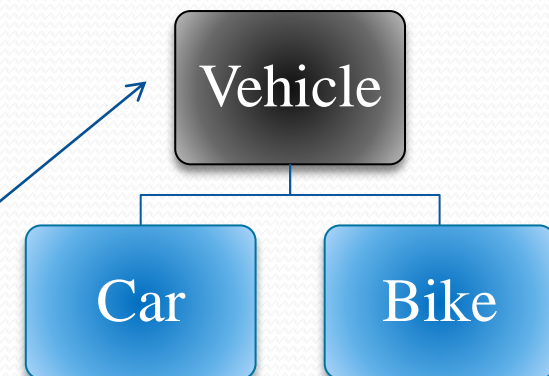
```
abstract class MyClass { ... }
```

- **Méthode :**

```
abstract function meth1( ... );
```

Pas de corps

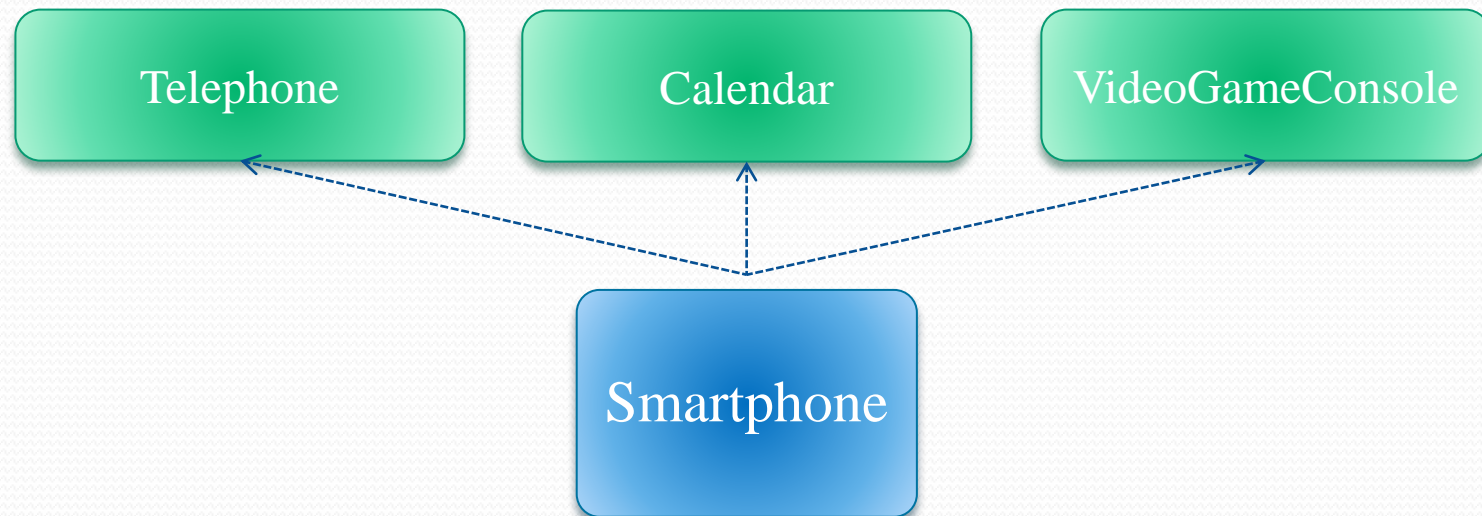
La classe `Vehicle` est abstraite :  
il n'y a pas d'instance de `Vehicle` mais  
des instances de `Car` ou de `Bike`





# Interface

- Modèle pour une classe
- **Classe totalement abstraite** sans attribut (non constant)
- Toutes les méthodes d'une interface doivent être **publique**
- Une classe implémentant une interface doit implanter (***implémenter***) toutes les méthodes déclarées par l'interface



# Interface

- Une interface **donne son type** aux classes l'implémentant
- Mot clef interface (pas abstract) :

```
interface MyInterface { ... };
```

- Mot clef implements :

```
class MyClass implements MyInterface1 { ... }  
class MyClass1 implements MyInterface1, MyInterface2 ... { ... }  
class MyClass2 extends MySuperClass implements MyInterface1 ... { ... }
```

- Les interfaces peuvent se dériver (mot clef extends)

```
interface MyInterface3 extends MyInterface1 { ... };  
interface MyInterface4 extends MyInterface1, MyInterface2 { ... };
```

# Chargement auto. des classes

- **Enregistre une fonction** de chargement auto. :

```
spl_autoload_register($autoload, ...)
```

- **Appelée automatiquement** lorsqu'une classe/interface doit être chargée

- Crée une **file d'attente** de fonctions de chargement auto., les exécute les unes après les autres, dans l'ordre de définition

```
function my_autoload ($class) {  
    include __DIR__ . '/' . $class . '.php';  
}  
spl_autoload_register('my_autoload');
```

```
$obj = new Class();
```

`__autoload()`

# Bonnes pratiques

- Penser à l'initialisation
- Penser à construire les objets avant de les utiliser
- Attention à l'encapsulation
- Pas d'utilisation de variable d'instance dans une méthode de classe
- Pas d'héritage multiple



# Crédits

## Auteur

Mickaël Martin Nevot

[mmartin.nevot@gmail.com](mailto:mmartin.nevot@gmail.com)



Carte de visite électronique

## Relecteur

- Christophe Delagarde  
([christophe.delagarde@univ-amu.fr](mailto:christophe.delagarde@univ-amu.fr))
- Pierre-Alexis de Solminihac ([pa@solminihac.fr](mailto:pa@solminihac.fr))

Cours en ligne sur : [www.mickael-martin-nevot.com](http://www.mickael-martin-nevot.com)

