



# **POO – PROGRAMATION ORIENTEE OBJET**

Copyright @iknsa

[contact@iknsa.com](mailto:contact@iknsa.com)

# PROGRAMMEZ EN ORIENTEE OBJET



*Programmation  
Orientée Objet*



# OBJECTIF DE LA FORMATION

- Développement en POO
- Utilisation de PDO
- Etudes des design-pattern (MVC, SINGLETON,...)



# POO - HISTORIQUE

- ▶ La programmation orientée objet (POO) a fait son apparition dans la version 3 de PHP. C'était alors simplement un moyen d'autoriser la syntaxe OO (par opposition au procédural), mais pas réellement un moyen de programmer efficacement avec des objets.
- ▶ PHP4 a continué dans la lancée, proposant de nouveaux mots clefs, mais toujours sans proposer une syntaxe proche de langages ayant une plus grande maturité comme C++ ou Java. C'est ce qui a facilité le passage des applications et des hébergements de PHP3 à PHP4, et c'est ce qui retarde la mise en place massive de PHP5 à travers le Web.
- ▶ PHP5, en revanche, introduit de véritables concepts OO : le constructeur est plus clairement identifié, le destructeur fait son apparition, les objets sont tous pris en charge comme des références, de nouveaux mots clefs font leur apparition (public, protected et private) ainsi que des interfaces et des classes abstraites...


# POO - C'EST QUOI ?



- Pour permettre à une entreprise de concevoir et développer un logiciel sur une durée de plusieurs années, il est nécessaire de structurer les différentes parties du logiciel de manière efficace. Pour cela, des règles précises de programmation sont établies, afin que les différents groupes de personnes intervenant sur un même logiciel puissent échanger leurs informations. En pratique, on a constaté que malgré l'instauration de règles, l'adaptation ou la réutilisation d'une partie programme nécessitait souvent une modification très importante d'une grande partie du code. La programmation orientée objet est donc apparue avec, pour **objectifs principaux** :
  - de concevoir l'organisation de grands projets informatiques autour d'entités précisément structurés, mêlant données et fonctions (les objets) facilitant la modélisation de concepts sophistiqués .
  - d'améliorer la sûreté des logiciels en proposant un mécanisme simple et flexible des données sensibles de chaque objet en ne les rendant accessibles que par le truchement de certaines fonctions associées à l'objet (encapsulation) afin que celles-ci ne soient pas accessibles à un programmeur inattentif ou malveillant.
  - de simplifier la réutilisation de code en permettant l'extensibilité des objets existants (héritage) qui peuvent alors être manipulés avec les même fonctions (polymorphisme).



# POO – CONCEPTS FONDAMENTAUX

- Objet, classe et instance
  - les membres de classe et d'instance
  - envoi de message et méthode
  - héritage, encapsulation et polymorphisme
  - Constructeur et destructeur
  - Classe abstraite / concrète
- 





# POO - TERMINOLOGIE

➤ Qu'est ce qu'un objet ?

➤ **Définition simplifiée:**

➤ C'est une boîte dans laquelle il y a des informations stockées et qui a des boutons sur lesquels on appuie pour obtenir ses informations.

➤ **Dans la vraie vie:**

➤ C'est une super-variable php qui contient des variables simples (attributs) et des fonctions (méthodes) qui permettent de récupérer ses attributs.

un objet est un conteneur symbolique et autonome qui contient des informations et des mécanismes concernant un sujet, manipulés dans un programme. C'est le concept central de la programmation orientée objet (POO).

# POO - OBJET

- un objet est créé à partir d'un modèle appelé classe ou prototype, duquel il hérite les comportements et les caractéristiques. Les comportements et les caractéristiques sont typiquement basés sur celles propres aux choses qui ont inspiré l'objet : une personne (avec son état civil), un dossier, un produit.

Prenons un exemple concret...

Article = Objet



Titre
Auteur
Date de création
Visuel
Corps de texte





# POO - CLASSE

- ▶ La classe est la structure d'un objet. Il s'agit du plan de tout ce qui compose l'objet. La classe est composée de deux parties:

les attributs ( ou propriétés )  
les méthodes

Les attributs sont les données associées à l'objet et les méthodes sont des fonctions qui sont associées à la classe.

# POO - CLASSE

Prenons un exemple concret...

Patron = Classe



titre
auteur
date
image
message

# POO – INSTANCIER UNE CLASSE

Le mot-clé class sert simplement à créer la classe. Il est suivi du nom de la classe ainsi que du bloc d'instruction dans lequel sera spécifié l'ensemble des attributs et méthodes.

Exemple de classe:

```
<?php  
class Personne{  
}  
?>
```

et si je veux instancier une personne:

```
<?php  
    $personne = new personne();  
?>
```

# POO – INSTANCIER UNE CLASSE

- ▶ Une instance, c'est tout simplement le résultat d'une instanciation. Une instanciation, c'est le fait d'instancier une classe. Instancier une classe, c'est se servir d'une classe afin qu'elle nous crée un objet. En gros, une instance est un objet.
- ▶ L'instanciation d'une classe est la phase de création des objets issus de cette classe. Lorsque l'on instancie une classe, on utilise le mot-clé *new* suivant du nom de la classe. Cette instruction appelle la méthode constructeur ( `__construct()` ) qui construit l'objet et le place en mémoire.
- ▶ EXEMPLE: **`$article = new Article();`**

# POO – INSTANCIER UNE CLASSE

php

## Instanciation

```
class voiture
{
    const ROUES_MOTRICES = 2;
    public $marque;
    function freiner($force_de_freinage)
    {
        //code qui freine
    }
}
```

```
$MaVoiture = new voiture();
```

- Les parenthèses sont optionnelles si le constructeur ne nécessite pas de paramètre
- En PHP5 toute classe doit être déclarée avant d'être utilisée
  - Non obligatoire en PHP4



# POO – SYNTAXE D'UNE CLASSE

```
<?php  
class NomDeMaClasse  
{  
    // Attributs  
  
    // Constantes  
  
    // Méthodes  
}  
?>
```





# POO – ATTRIBUTS METHODES

- La classe est composée de deux parties: les attributs ( ou propriétés ) les méthodes.
- Les « attributs » (aussi appelés « données membres ») sont les caractères propres à un objet.

Une personne, par exemple, possède différents attributs qui lui sont propres comme le nom, le prénom, la couleur des yeux, le sexe, la couleur des cheveux, la taille...

- Les « méthodes » sont les actions applicables à un objet.

Un objet personne, par exemple, dispose des actions suivantes : manger, dormir, boire, marcher, courir..

# POO – LES CONSTANTES

- **Les constantes**

- Il est aussi possible de déclarer des constantes propres à la classe. Contrairement au mode procédural de programmation, une constante est déclarée avec le mot-clé *const*.
- **Remarque** : *une constante doit être déclarée et initialisée avec sa valeur en même temps.*

- ❖ `const NOMBRE_DE_BRAS = 2;`
- ❖ `const NOMBRE_DE_JAMBES = 2;`
- ❖ `const NOMBRE_DE_YEUX = 2;`

# POO - Affichage attributs

```
<?php
class Article {
    public $_titre = "J'aime les objets";
    public $_auteur = "Moussa camara";
}
```

```
$article = new Article();
echo 'Titre : '. $article->_titre . '<br/>';
echo 'Auteur : '. $article->_auteur;
?>
```

**Affichage:**

**Titre : J'aime les objets**  
**Auteur : Moussa camara**

# POO – Affichage constante

- L'accès aux constantes ne peut se faire qu'en lecture via l'opérateur ::

```
<?php
```

```
class Article {
```

```
    public $_titre = "J'aime les objets";
```

```
    public $_auteur = "Moussa camara";
```

```
    const NOMBRE_DE_PAGE = 200;
```

```
}
```

```
    echo 'Mon article fait ', Article::NOMBRE_DE_PAGE , ' pages';
```

```
?>
```

**Affichage:** Mon article fait 200 pages

# POO - Affichage des méthodes

```
<?php
class Article {
    public $_titre = "J'aime les objets";
    public $_auteur = "Moussa camara";

    public function afficheNomAuteur()
    {
        return $this->_auteur;
    }
}

$article = new Article();
echo $article->afficheNomAuteur();
?>
```

**Affichage : Moussa camara**

# POO – DECLARATION CLASSE

## Remarque :

*par convention, on écrit le nom d'une classe en « **Upper Camel Case** », c'est à dire que tous les mots sont accrochés et chaque première lettre de chaque mot est écrit en capital.*

***/\* NomDeMaClasse \*/***

```
<?php
```

```
class Personne
```

```
{
```

```
    // Attributs
```

```
    public $nom;
```

```
    public $prenom;
```

```
    public $dateDeNaissance;
```

```
    // Constantes
```

```
    const NOMBRE_DE_BRAS = 2;
```

```
    const NOMBRE_DE_JAMBES = 2;
```

```
    const NOMBRE_DE_YEUX = 2;
```

```
    // Méthodes
```

```
    public function __construct() { }
```

```
    public function courrir()
```

```
    {
```

```
        echo 'Je cours plus vite que Bolt';
```

```
    }
```

```
    public function manger()
```

```
    {
```

```
        echo 'Je mange plus vite que Hulk <br/>';
```

```
    }
```

```
}
```



# POO - CLASS

Prenons un exemple concret...

Classe

Attributs	Méthodes
\$titre	donneMoiTitre()
\$auteur	donneMoiAuteur()
\$date	donneMoiDate()
\$image	donneMoiImage()
\$message	donneMoiMessage()

# POO – CLASS EXAMPLE

Classe

```
<?php
```

```
class Article {  
    private $_titre;  
    private $_auteur;  
    private $_date;  
    private $_image;  
    private $_message;  
  
    public function donneMoiTitre() { }  
    public function donneMoiAuteur() { }  
    public function donneMoiDate() { }  
    public function donneMoiImage() { }  
    public function donneMoiMessage() { }  
}
```

```
?>
```

# POO – CLASS EXAMPLE

## Création de notre 1er objet

```
<?php

class Article {
    private $_titre = "J'aime les objets";
    private $_auteur = "Philippe Stark";
    ...

    public function donneMoiTitre() {
        return $this->_titre;
    }
    ...
}

$article = new Article();

?>
```



# POO - La variable courante *\$this*

- ▶ Le mot clé *\$this* permet de désigner l'objet dans lequel on se trouve, c'est-à-dire que lorsque l'on désire faire référence dans une fonction membre à l'objet dans lequel elle se trouve, on utilise *this*.
- ▶ Grâce à cette variable spéciale, il est possible dans une fonction membre de faire référence aux propriétés situées dans le même objet que la fonction membre.
- ▶ Ainsi, lorsque l'on désire accéder à une propriété d'un objet à partir d'une méthode du même objet, il suffit de faire précéder le nom de la donnée membre par *\$this->*.

# POO – THIS EXAMPLE

```
<?php
class Article {
    private $_titre = "J'aime les objets";
    private $_auteur = "Moussa camara";

    public function donneMoiTitre() {
        return $this->_titre;
    }
}

$article = new Article();
echo ($article->donneMoiTitre());
?>
```

**Affichage: J'aime les objets**

# POO - EXAMPLES

## Quelques essais...

```
class Article {
    $_titre = "J'aime les objets";
}
---
class Article {
    public $_titre = "J'aime les objets";
}
$article = new Article();
print $article->_titre;
---
class Article {
    private $_titre = "J'aime les objets";
}
$article = new Article();
print $article->_titre;
```

```
class Article {
    private $_titre = 'J'aime les objets';
    public function donneMoiTitre() {
        print 'Je veux pas';
    }
}
$article = new Article();
$article->donneMoiTitre();
---
class Article {
    private $_titre = 'J'aime les objets';
    public function donneMoiTitre() {
        return $this->_titre;
    }
}
$article = new Article();
print $article->donneMoiTitre();
```





# POO – LA Visibilité ou La portée des variables

- La visibilité d'un attribut ou d'une méthode peut être définie en prefixant sa déclaration avec un mots clé: public, protected ou private.
- Les "**public**" peuvent être appelés à n'importe quelle partie du programme.
- Les "**protected**" ne peuvent être appelés que par la classe elle même ou les classes parents/enfants. (Héritantes)
- Les "**private**" sont disponibles que pour la classe en elle même.

# POO – La portée des variables

```
<?php
class Article {
    private $_private = "Je suis private";
    protected $_protected = "je suis protected";
    public $_public = "je suis public";
}

$article = new Article();

print_r($article->_public);
print_r($article->_private);
print_r($article->_protected);
?>
```

je suis public

 Fatal error: Cannot access private property Article::\$\_private in C:\wamp64\www\formation-poo\index.php on line 11

Call Stack

#	Time	Memory	Function	Location
1	0.0005	237760	{main}()	...\index.php:0

# POO - getter et setter

- On peut accéder et modifier les attributs d'une classe grace aux getters et setters.

- 1ere méthode**

```
<?php
class Article {
    public $_titre = "J'aime les objets";
    public $_auteur = "Moussa camara";

    public function donneMoiTitre() {
        return $this->_titre;
    }
}
```

```
$article = new Article();
```

```
$article->_titre = 'je change le titre';
echo($article->_titre);
?>
```

- Affichage: je change le titre**



# POO - getter et setter

- Il est préférable de passer par des méthodes qui feront la modification.
- On appelle ce genre de méthode un setter ; et on récupère la valeur avec un getter.

- **2ème méthode**

« Best practice »

# POO - getter et setter

```
<?php
class Article {
    public $_titre = "J'aime les objets";
    public $_auteur = "Moussa camara";

    public function setTitre($_titre)
    {
        return $this->_titre = $_titre;
    }
    public function getTitre()
    {
        return $this->_titre;
    }
}

$article = new Article();
$article->setTitre('J'aime beaucoup beaucoup beaucoup les objets :-');
print $article->getTitre();
?>
```

**Affichage: J'aime beaucoup beaucoup beaucoup les objets :-)**



# MINI TP

- ▶ Mini Tp – Exo1

- ▶ Créer une classe personne avec:

- ▶ les attributs nom,prenom,age

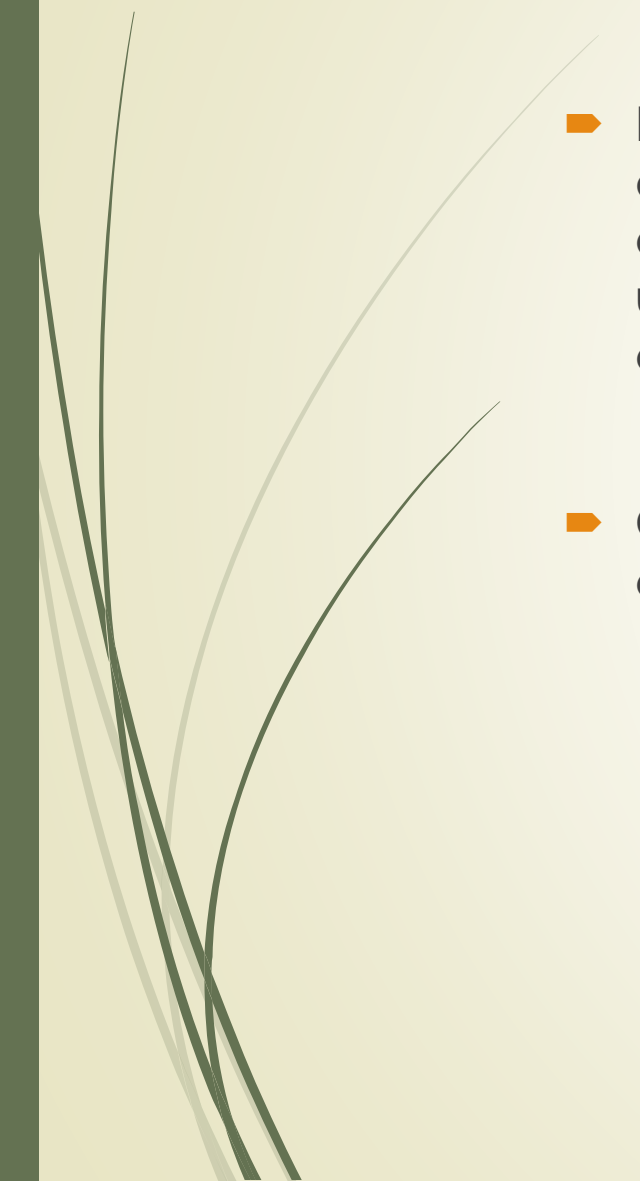
- ▶ Les méthodes: getPrenom, getNom, getAge, setprenom, setNom,setAge

- ▶ Afficher le prenom, nom et age de la personne





# POO – LES ANNOTATIONS

- ▶ En programmation, une annotation est un élément permettant d'ajouter des méta-données à un code source. Selon le langage de programmation et ce qu'a choisi le programmeur, elles peuvent être accessibles uniquement lors de la compilation, présentes uniquement dans le fichier compilé, voire accessibles à l'exécution.
  - ▶ C'est utilisé par les IDEs pour décrire les méthodes et peut être utilisé comme dans symfony pour définir les types
- 

# POO – LES ANNOTATIONS - EXEMPLE

```
class Personne{  
    /**  
     * @var string  
     */  
    private $nom;  
    /**  
     * @var string  
     */  
    private $prenom;  
    /**  
     * @var int  
     */  
    private $age;
```



# POO – LES NAMESPACES

- ▶ On ne peut pas créer une classe, une fonction avec le même. Si toutefois on s'en amène à le faire, PHP nous l'autorise en utilisant les namespaces.
- ▶ Le namespace représente le répertoire dans lequel se trouve le fichier. C'est une sorte de chemin virtuel donné à php faciliter lors de l'importation de fichier.
- ▶ namespace, permet de dire **"je travaille dans ce dossier"**
- ▶ use, permet d'importer une classe d'un **autre "dossier"**


# POO – LES NAMESPACES

**Fichier: namespace1**

```
<?php  
namespace monEspace1;  
class Personne  
{  
}  
?>
```

**Fichier: namespace2**

```
<?php  
namespace monEspace2;  
class Personne  
{  
}  
?>
```



# POO - Exo

- Exercice
- Créer deux classes personnes.
  - Personne1: nom prenom age avec getter et setter
  - Personne2: nom prenom age taille avec getter et setter
- <https://github.com/mouskito/formation-poo/tree/used-namespaces>



# POO – METHODE MAGIQUE - CONSTRUCTEUR

- ▶ Un constructeur est une méthode qui va nous permettre de rendre notre classe immédiatement opérationnelle et utilisable en définissant des propriétés dès qu'un utilisateur va vouloir créer une nouvelle instance de cette classe.
- ▶ Un **constructeur** va donc se charger de mettre en place les données, d'associer les méthodes avec les champs et de créer le *diagramme d'héritage* de l'objet, autrement dit de mettre en place toutes les liaisons entre les ancêtres et les descendants.

# POO – METHODES PREDEFINIES

## Principales méthodes magiques

37

`__construct()`

`__destruct()`

`__get()`

`__isset()`

`__set()`

`__unset()`

`__call()`

`__callStatic()`

`__toString()`

`__clone()`



# POO – CONSTRUCTEUR EXEMPLE

```
<?php
class Article {
    public $_titre;
    public function __construct() {
        $this->setTitre('Cool le poo');
    }
    public function getTitre() {
        return $this->_titre;
    }
    public function setTitre($new_titre)
    {
        $this->_titre = $new_titre;
    }
}
$article = new Article();
print_r ($article->getTitre());
?>
```

Affichage: Cool le poo



# POO - CONSTRUCTEUR - EXEMPLE

- Utilisation des constructeurs avec ou sans parametres
- <https://github.com/mouskito/formation-poo/tree/used-conconstruct>




# POO – DESTRUCTEUR

- Le **destructeur** : il se charge de **détruire l'instance de l'objet**. La mémoire allouée pour le *diagramme d'héritage* est libérée. Certains compilateurs peuvent également se servir des destructeurs pour éliminer de la mémoire le code correspondant aux méthodes d'un type d'objet si plus aucune instance de cet objet ne réside en mémoire.

Là encore, différentes remarques doivent être gardées à l'esprit.

- Tout comme pour les constructeurs, un objet peut **ne pas avoir de destructeur**. Une fois encore, c'est le compilateur qui se chargera de la destruction **statique** de l'objet.




```
<?php
class ExempleConstructeur
{
    /**
     * @var bool
     */
    public $couleur;

    function __construct()
    {
        $this->couleur = "rouge";
    }

    function __destruct() {
        print "Destruction de " . $this->couleur . "\n";
    }
}
```



# POO – méthode `__set`

- Surchage : `__set()`.
  - Cette méthode est appelée au moment de la création d'une propriété dynamique de l'objet
- 



# POO – METHODE `__get`

- Surchage : `__get()`.
- Cette méthode est appelée au moment de la consultation d'une propriété créée dynamiquement sur l'objet



# POO – Méthode `__sleep`

- ▶ `__sleep()`.
- ▶ Cette méthode est appelée avant toute linéarisation (conversion de données en chaîne avec la fonction `serialize()`). Son rôle est de retourner un tableau avec les noms de toutes les variables de l'objet qui doivent être linéarisées.



- Conversion : `__toString()`.
- Cette méthode est appelée lorsque l'objet est traité comme une chaîne (echo `$maVoiture`, par exemple)



```
class Soda{  
    public $marque;  
    private $prop = [];  
    function __construct($marque) {  
        $this->marque = $marque;  
    }  
    function __toString() {  
        return $this->marque;  
    }  
}  
$mabouteille = new Soda('coca');  
echo $mabouteille;
```

Affichage:  
coca



# POO – HERITAGE

- L'héritage est un mécanisme qui facilite la réutilisation du code et la gestion de son évolution. Grâce à l'héritage, les objets d'une classe ont accès aux données et aux méthodes de la classe parent et peuvent les étendre. Les sous classes peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent. Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge. L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de super classes et de sous classes.
- Une classe qui hérite d'une autre est une sous classe et celle dont elle hérite est une `super_classe`.
- Une classe peut avoir plusieurs sous classes. Une classe ne peut avoir qu'une seule classe mère

- 
- 
- ▶ On utilise le mot clé **extends** pour indiquer qu'une classe hérite d'une autre. En l'absence de ce mot réservé associé à une classe, le compilateur considère la classe Object comme classe parent.
  - ▶ En résumé:
    - ▶ L'héritage permet de spécialiser le code et/ou réutiliser du code déjà existant pour l'adapter à ses besoins. En d'autres termes, il permet, à partir d'une classe déjà existante, d'en créer une nouvelle qui reprendra ses caractéristiques ce qui nous permettra de les adapter à d'autres besoins sans modifier la classe de base.



# POO - SYNTHASE

```
<?php  
    class Mere{  
    }  
    class Fille extends Mere{  
    }  
?>
```

# POO - EXEMPLE HERITAGE

```
<?php
class Mere{
    public function parent(){
        $mere=" Je suis la classe Mère.";
        return $mere;
    }
}

class Fille extends Mere{
    public function enfant(){
        $fille=" Je suis la classe Fille, j'hérite de ma mère";
        return $fille;
    }
}

$objet=new Fille();
echo $objet->parent(). "<br />";
echo $objet->enfant();
?>
```

## Affichage:

Je suis la classe Mère.

Je suis la classe Fille, j'hérite de ma mère



# POO - HERITAGE

- ▶ Mini tp

- ▶ Créer des types d'animaux (chat, chien, lapin, lion, ...) qui héritent de la classe animal

- ▶ <https://github.com/mouskito/formation-poo/tree/used-heritage>



# POO – SURCHARGE DE METHODE

- ▶ On a créé une classe fille qui hérite de la classe mère avec la méthode parent(). Si on crée une autre méthode parent() dans la classe fille celle-ci va écraser la methode parent qui est appelée dans la fille. Pour utiliser ces deux methodes avec le même nom, on peut surcharger les méthodes.
- ▶ Ainsi la médthode parent de la classe mère est combibée dans la classe fille à l'aide de l'opérateur de résolution de portée ::



# POO - SURCHARGE

```
<?php
class Mere{
    public function parent(){
        $mere=" Je suis la classe Mère.";
        return $mere;
    }
}
class Fille extends Mere{
    public function parent(){
        /* Surcharger la methode parent*/
        $mere = Mere::parent();
        /* Fin surcharger*/
        $mere.=" Je suis la classe Fille, j'hérite de ma mère";
        return $mere;
    }
}
$objet=new Fille();
echo $objet->parent(). "<br />";
?>
```

Affichage:  
Je suis la classe Mère. Je suis la  
classe Fille, j'hérite de ma mère



# POO – ENCAPSULATION

- ▶ Le principe d'encapsulation correspond tout simplement à la façon dont on définit la visibilité et l'accessibilité de nos propriétés et méthodes de classe.
- ▶ L'encapsulation est, avec l'héritage, l'un des concepts fondamentaux de la programmation orientée objet PHP.
- ▶ On va pouvoir définir trois niveaux d'accessibilité différents pour nos propriétés et méthodes grâce aux trois mots clefs suivants :
  - ▶ public
  - ▶ private
  - ▶ protected

# POO - Méthodes statiques

- La méthode statique est une méthode qui n'a pas besoin d'être appelée depuis un objet. Sa syntaxe est celle ci: CLASSE::METHODE()

```
<?php
```

```
class Article {  
    public static function afficherMonNom(){  
        return "Moussa Camara";  
    }  
}
```

```
print_r( Article::afficherMonNom() );
```

```
?>
```

- Affichage: Moussa Camara**

# POO - HYDRATION

- Hydrater un objet, c'est tout simplement fournir des valeurs à ses attributs.
- Remplir les variables d'un objet avec celles d'un visiteur, à travers un formulaire.

```
public function hydrate(array $table)
{
    var_dump($table);die;
    foreach ($table as $property => $value) {
        $method = 'set' . ucfirst($property);

        /* fonction de verification si la methode existe ou pas*/
        if (method_exists($this, $method)) {
            $this->$method($value);
        }
    }
}
```

# POO - HYDRATION

## Hydratation

```
public function __construct(array $data) {  
    $this->hydrate($data);  
}  
  
public function hydrate(array $data) {  
    foreach ($data as $key => $value) {  
        $method = 'set'.ucfirst($key);  
        if (method_exists($this, $method)) {  
            $this->$method($value);  
        }  
    }  
}
```

# POO - DAO (Data Access Object)





# POO - DAO (Database Access Object)

- ▶ Le DAO est un moyen de scinder la couche métier et la couche de stockage de données.
- ▶ Couche métier: connections à la base de donnée
- ▶ Couche stockage: la création, la lecture, la modification et la suppression. Ces quatre tâches basiques sont souvent raccourcies à l'anglaise en *CRUD*.



# PDO - C'EST QUOI ?

- ▶ PDO ( pour [PHP](#) Data Object ) est une interface qui peut accéder à une base de données ( [MySQL](#) par exemple ) depuis PHP. La classe PDO est orientée objet et existe depuis PHP 5.1.0
- ▶ Activer PDO
  - ▶ Pour activer PDO assurez-vous que la ligne suivante est décommentée dans le fichier php.ini
  - ▶ `extension=php_pdo_mysql.dll`

# POO - CONNECTION A LA BDD

## Connexion à la BDD : Les Managers - Partie 1

```
try {  
    $DB = new PDO('mysql:host=localhost;dbname=aston', 'root', '');  
}  
catch(Exception $e) {  
    die('Erreur : ' . $e->getMessage());  
}
```

# POO - CONNECTION A LA BDD

```
<?php
define("PDO_HOST", "localhost");
define("PDO_DBBASE", "aston");
define("PDO_USER", "root");
define("PDO_PW", "");

try{
    $connection = new PDO("mysql:host=" . PDO_HOST . " dbname=" . PDO_DBBASE,
PDO_USER, PDO_PW,array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8") );
}
catch (PDOException $e){
print "Erreur !: " . $e->getMessage() . "<br/>";
die();
}
?>
```



# POO - Préparer une requête SQL

- La méthode **prepare()** de l'objet PDO permet de préparer une requête. Elle accepte comme paramètre une chaîne de caractères qui correspond à la requête souhaitée. Les marqueurs interrogatifs sont là pour désigner les valeurs à passer à la requête. Ces valeurs seront ensuite passées lors de l'exécution de la requête.
- La méthode **execute()** est appelée par l'objet qui correspond à la requête préparée. Elle accepte comme paramètre une variable de type tableau qui contient les valeurs qui remplaceront, dans l'ordre, les marqueurs interrogatif. Première entrée pour le premier marqueur, deuxième entrée pour le deuxième marqueur et ainsi de suite.

# POO – Insert into BDD

## Connexion à la BDD : Les Managers - Partie 2

```
class ArticleManager {  
    ...  
    public function addArticle(Article $article) {  
  
        $ADD_ARTICLE = $this->_db->prepare('INSERT INTO article SET titre=:  
titre, auteur=:auteur, date=NOW(), image=:image, message=:message');  
        $ADD_ARTICLE->bindParam(':titre', $article->getTitre());  
        ...  
        $ADD_ARTICLE->execute();  
        $ADD_ARTICLE->closeCursor();  
    }  
}
```

# POO – SHOW LIST

## Connexion à la BDD : Les Managers - Partie 3

```
class ArticleManager {  
    ...  
    public function getAllArticle() {  
  
        $LIST = $this->_db->query('SELECT *, DATE_FORMAT(date, \'%d/%m/%y\')  
AS date FROM article ORDER BY date DESC');  
  
        while ($list_fetch = $LIST->fetchAll()) {  
            $article[] = $list_fetch;  
        }  
        return $article;  
        $LIST->closeCursor();  
    }  
}
```

# POO – UPDATE

```
class ArticleManager {  
    ...  
    public function updateArticle(Article $article) {  
  
        $UP_ARTICLE = $this->_db->prepare('UPDATE article SET titre=:titre,  
auteur=:auteur, image=:image, message=:message WHERE id=:id');  
        $UP_ARTICLE->bindValue(':id', $article->getId());  
        ...  
        $UP_ARTICLE->execute();  
        $UP_ARTICLE->closeCursor();  
    }  
}
```

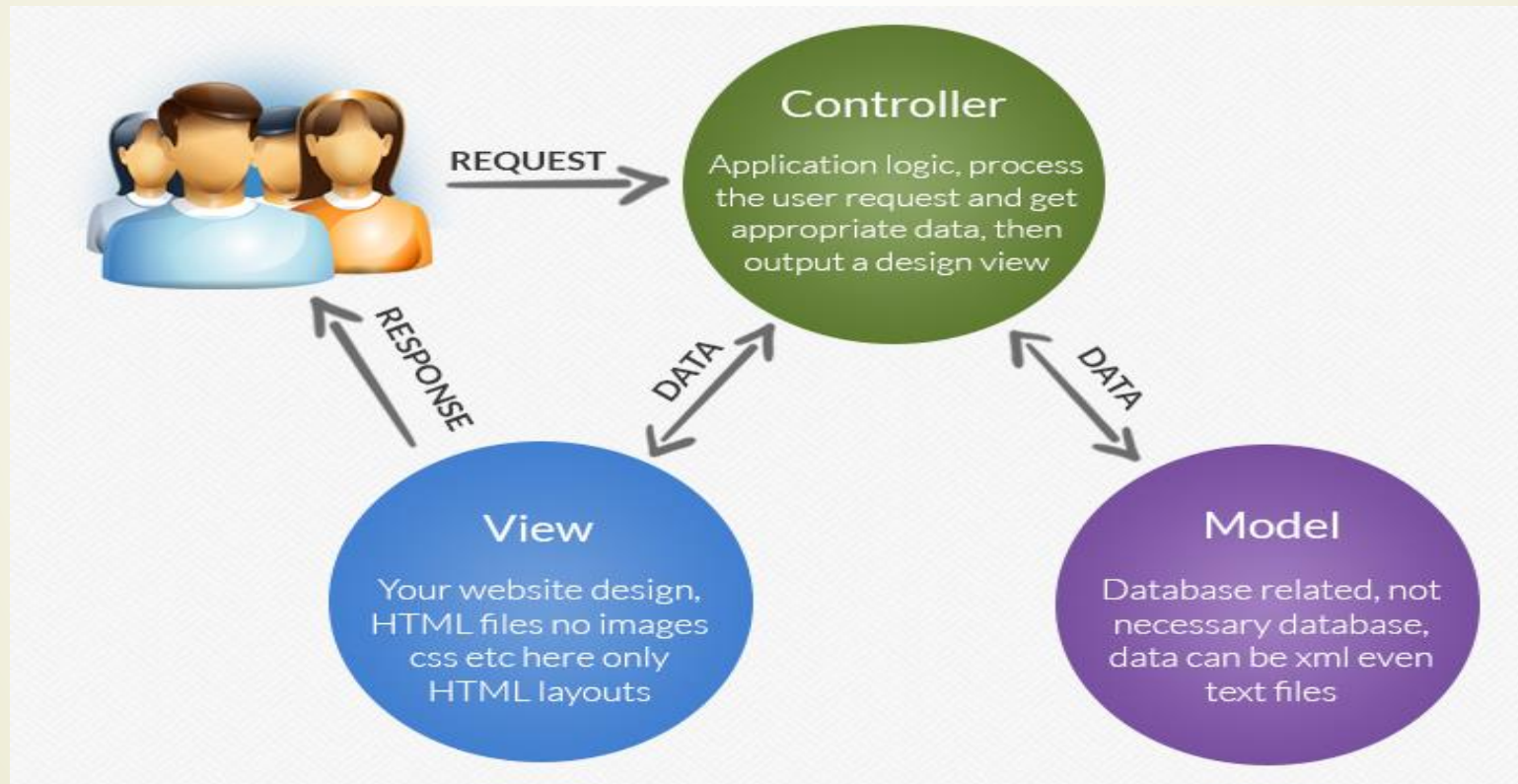




# POO - DELETE

```
class ArticleManager {  
    ...  
    public function deleteArticle($article_id) {  
        $article_id = (int) $article_id;  
        $this->_db->exec('DELETE FROM article WHERE  
id='.$article_id);  
    }  
}
```

# POO – MVC (Modele, View, controleur)





# Modele, View, controleur

- ▶ L'architecture MVC signifie Modele, View, controleur. C'est une façon de travailler qui consiste à séparer les taches d'accès des données (Modèle), de leurs récupération (controleur) et de leurs présentation (Vue)
- ▶ Par exemple, si vous voulez enregistrer un nouvel article :
  - ▶ 1) le controleur (C) va récupérer les informations, via un formulaire, les traiter
  - ▶ 2) lancer l'enregistrement dans la bases de données via le modèle (M). Le modèle permet en effet d'accéder aux données et de les conserver.
  - ▶ 3) le controleur lance ensuite l'affichage via la vue (V). La vue affichera tout le code HTML de la page en récupérant éventuellement des variables fournies par le controleur.



# MVC - CONTROLEUR

## Le modèle MVC

### Contrôleur :

- Implémente les actions.
- Autant de classes que nécessaire, regroupés par entités logiques (Articles, utilisateurs...)
- Et aussi pour les erreurs (404, 403, ...).
- Chaque contrôleur a ses fonctions (Lister, ajouter, modifier, supprimer,...)



# MVC - MODELE

## Le modèle MVC

### Modèle :

- Gère les échanges avec la BDD.
- Une classe par entité dans la BDD (utilisateur, article, produit, catégorie, etc..).



# MVC – LA VUE

## Le modèle MVC

### Vue :

- Affichages HTML.
- Injecte des parties variables (provenant du modèle par exemple).
- On peut utilisé un moteur de gabarit (Smarty, Twig).



# MVC – DESIGN PATTERNS

## Problématique :

Un code pompeux, mal organisé et difficilement maintenable.

## Solution :

Une organisation structurée en fichiers simples et en trois fonctionnalités :  
Modèle, vue et contrôleur .