



# PHP ORIENTE OBJET

## PRÉSENTATION GÉNÉRALE





Un objet

Une classe

Développement orienté objet

Pourquoi utiliser la POO

Programmation objet vs programmation procédural

Les apports de la POO

Abstraction / Encapsulation / Héritage





Un objet



Chaise longue



Souris d'ordinateur



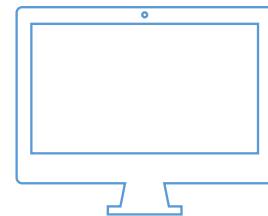
Ordinateur



Un site web / Statique

## Attributs d'un ordinateur

- Marque
- Prix
- Couleur
- Taille de l'écran



## Fonctions d'un ordinateur

- Additionner
- Multiplier
- Soustraire

## Attributs d'une chaise longue

- Couleur
- Taille
- Matière
- Position



## Fonctions une chaise longue

- Changer de position
- Mettre en position allongée
- Mettre en position assise



Pourquoi utiliser la POO

Gagner du temps

Réutiliser mon code source

Organiser mon code source

Utiliser une méthode standard



## Programmation orienté Objet VS Programmation procédurale

Programmation orienté objet	Programmation procédurale
Architecture de projet standardisée.	Architecture de projet propre au développeur.
Ensemble d'objet interagissant entre eux	Code source exécuté ligne par ligne.
Maintenance simplifiée	Faible organisation du code source. Travail en équipe difficile
Simplifie la réutilisation de code source	
Simplifie la documentation du projet	

Les langages de programmation utilisant la POO



Et bien plus encore !  
De nombreux framework utilisent la programmation orienté objet.

## Principe de la POO

Ensemble d'objets interagissant ensemble.

Possibilité de limiter les accès à certaines méthodes ou attributs de l'objet.



Une classe

# Moule qui va servir à créer l'objet

Décrit les fonctions et les attributs d'un objet

Sert à créer (instancier) les objets grâce au mot clé « new »



```
<?php  
class Ordinateur  
{  
}  
$ordi = new Ordinateur();
```

```
<?php  
class ChaiseLongue  
{  
}  
$chaiseLongue = new ChaiseLongue();
```



L'encapsulation

## Objectif : Faciliter l'utilisation par le consommateur



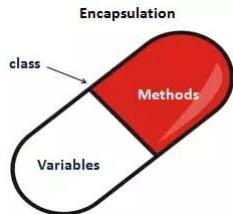
Objet avec une logique complexe



L'utilisateur n'agit pas directement sur l'objet pour simplifier et sécurisé l'utilisation



## Création d'attributs, de méthodes et visibilité



La visibilité d'un **attribut** ou d'une **méthode** permet de définir la manière dont un attribut ou une méthode sera accessible dans le programme

- **public** : l'attribut ou la méthode sera accessible dans tous le programma
- **private** : l'attribut sera utilisable seulement par l'objet lui-même, aucun autre accès ne sera autorisé

```
<?php
class Ordinateur
{
    private $_marque;
    private $_model;
    private $_result;
    public $displaySize = '16 pouces';
}

$ordi = new Ordinateur();
echo($ordi->displaySize);
echo($ordi->_marque);
```

16 pouces

(!) Fatal error: Uncaught Error: Cannot access private property Ordinateur::\$\_marque in /var/www/html/helloworld.php on line 12

(!) Error: Cannot access private property Ordinateur::\$\_marque in /var/www/html/helloworld.php on line 12

Call Stack

#	Time	Memory	Function	Location
1	0.0027	416584	{main}()	.../helloworld.php:0

```
<?php
class Ordinateur
{
    private $_marque;
    private $_model;
    private $_result;
    public $displaySize = '16 pouces';

    public function sayHello(){
        echo('Hello World !');
    }

    private function sayGoodBye(){
        echo('Goodbye !');
    }
}

$ordi = new Ordinateur();
$ordi->sayHello();
$ordi->sayGoodbye();
```

Hello World !

(!) Fatal error: Uncaught Error: Call to private method Ordinateur::sayGoodbye() from global scope in /var/www/html/helloworld.php on line 20

(!) Error: Call to private method Ordinateur::sayGoodbye() from global scope in /var/www/html/helloworld.php on line 20

Call Stack

#	Time	Memory	Function	Location
1	0.0030	417216	{main}()	.../helloworld.php:0



## Une méthode magique : Le constructeur

```
<?php  
class Ordinateur  
{  
    private $_marque;  
    private $_model;  
    private $_result;  
  
    public function __construct($marque, $model){  
        echo('Un nouvel objet Ordinateur est créé !');  
        $this->_marque = $marque;  
        $this->_model = $model;  
    }  
  
}  
  
$ordi = new Ordinateur( marque: "Macbook pro", model: "16 Pouces");  
var_dump($ordi);
```

Le constructeur est une méthode magique.

Le constructeur aura toujours une visibilité public (logique, sinon comment pourrait-on créer de nouveaux objets ?)

Cela signifie qu'il sera appelé implicitement (automatiquement). Il est appelé lorsque l'on instancie un nouvel objet avec le mot clé « new ».

Nous remplissons le moule pour obtenir notre objet.

Un nouvel objet Ordinateur est créé !

```
/var/www/html/helloworld.php:17:  
object(Ordinateur)[1]  
    private '_marque' => string 'Macbook pro' (length=11)  
    private '_model' => string '16 Pouces' (length=9)  
    private '_result' => null
```

## Une méthode magique : Le destructeur

Permet de faire un traitement quand un objet est détruit

Permet de faire un traitement quand un objet est détruit

Est appelé implicitement à la fin du script

```
class Ordinateur
{
    private $_marque;
    private $_model;
    private $_result;
    public $displaySize = '16 pouces';

    public function sayHello(){
        echo('Hello World !');
    }

    private function sayGoodBye(){
        echo('Goodbye !');
    }

    public function __destruct(){
        echo('Suppression de mon objet');
    }
}

$ordi = new Ordinateur();
$ordi->sayHello();
```

---

Hello World ! Suppression de mon objet

## L'héritage

D'autres classes pourront hériter de Ordinateur.

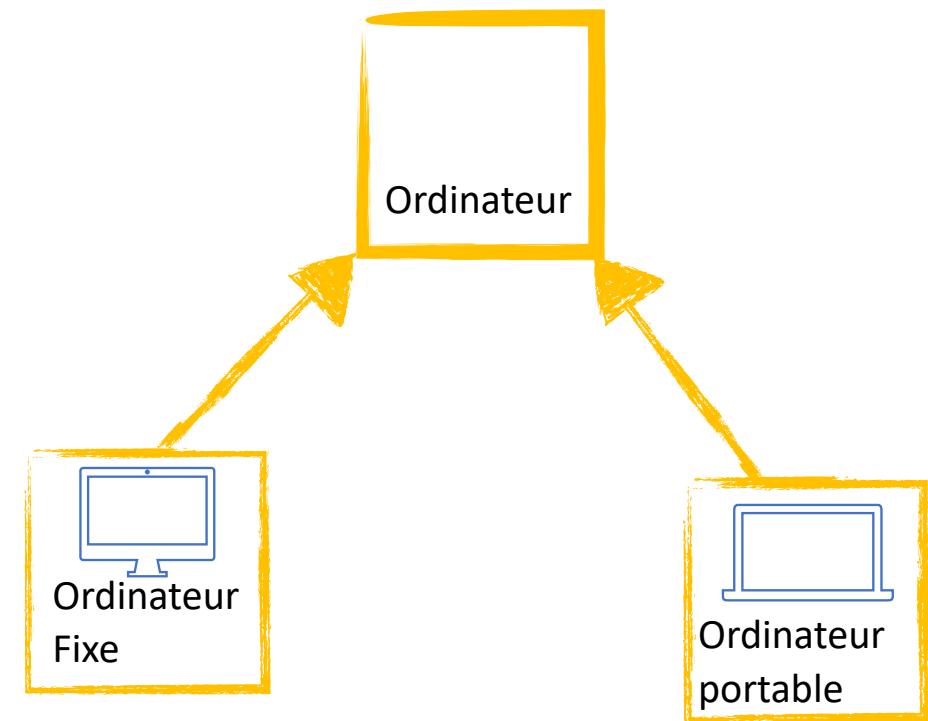
Une autre classe pourra hériter de ordinateur fixe ou portable

Une classe fille à une seule classe parente

Une classe mère à une infinité de classes filles

Les classes enfants comprendront toutes les fonctions et attributs de la classe parent.

Les règles de visibilité seront toujours valables





## L'héritage

## Utilisation du mot clé « extends »

```
<?php
class Ordinateur
{
    private $marque;
    private $model;
    private $result;
}

class OrdinateurFixe extends Ordinateur {
    private $displaySize = '16 pouces';
}
```

Si les classes sont créés dans des fichiers séparés, il ne faut pas oublier de les insérer.

Ordinateur.php

```
<?php
class Ordinateur{
    private $marque;
    private $model;
    private $result;
}

?>
```

index.php  
Ordinateur.php  
OrdinateurPortable.php

OrdinateurPortable.php

```
<?php
class OrdinateurPortable extends Ordinateur{
    private $displaySize = "16 pouces";
}
```

index.php

```
<?php
require "Ordinateur.php";
require "OrdinateurPortable.php";

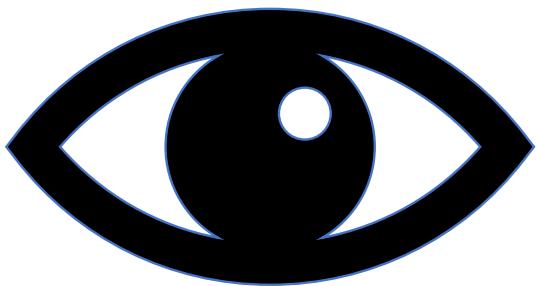
$ordiPortable = new OrdinateurPortable();
```

## L'héritage / visibilité protected



Si l'on créé une fonction ayant le même nom, elle sera écrasée !

En revanche, on peut appeler des fonctions de la classe parent avec le sélecteur parent::nomDeLaFonction

**Protected**

Les méthodes et attributs ayant la visibilité protected seront accessible à l'intérieur des classes mais aussi dans les classes qui en héritent. Ils ne seront pas accessible dans l'application



## L'héritage

index.php

```
<?php
require "Ordinateur.php";
require "OrdinateurPortable.php";

$ordiPortable = new OrdinateurPortable( marque: 'Apple', model: 'Macbook Pro', displaySize: '16 pouces');

var_dump($ordiPortable);
```

```
/var/www/html/php-backpack/cours_poo/index.php:13:
object(OrdinateurPortable)[1]
  private '_displaySize' => string '16 pouces' (length=9)
  private '_marque' (Ordinateur) => string 'Apple' (length=5)
  private '_model' (Ordinateur) => string 'Macbook Pro' (length=11)
  private '_result' (Ordinateur) => null
```

OrdinateurPortable.php

```
<?php
class OrdinateurPortable extends Ordinateur{
    private $_displaySize;

    public function __construct($marque, $model, $displaySize){
        parent::__construct($marque, $model);
        $this->displaySize = $displaySize;
    }
?>
```

Ordinateur.php

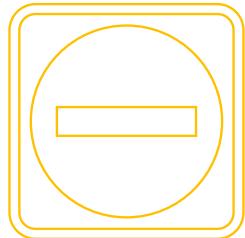
```
<?php
class Ordinateur{
    private $_marque;
    private $_model;
    private $_result;

    public function __construct($marque, $model){
        $this->_model = $model;
        $this->_marque = $marque;
    }
?>
```



## Classe abstraite

## Classe abstraite



Empêche de créer une nouvelle instance objet.

Des classes pourront en hériter

Fournit un plan dans la création de futures classe

« En programmation orientée objet (**POO**), une **classe abstraite** est une **classe** dont l'implémentation n'est pas complète et qui n'est pas instanciable. Elle sert de base à d'autres **classes** dérivées (héritées). »  
[Wikipedia](#)

Exemple je souhaite que l'on puisse créer des ordinateurs portable, des ordinateurs fixe, mais PAS des ordinateurs

```
<?php
abstract class Ordinateur{
    private $_marque;
    private $_model;
    private $_result;

    public function __construct($marque, $model){
        $this->_model = $model;
        $this->_marque = $marque;
    }
?>
```

```
<?php
class OrdinateurPortable extends Ordinateur{
    private $displaySize;

    public function __construct($marque, $model, $displaySize){
        parent::__construct($marque, $model);
        $this->displaySize = $displaySize;
    }
?>
```

```
6
7     <?php
8     require "Ordinateur.php";
9     require "OrdinateurPortable.php";
10
11     $ordinateurPortable = new OrdinateurPortable( 'Apple', 'Macbook Pro', '16 pouces');
12
13     $ordinateur = new Ordinateur("Apple", "Macbook Pro");
```

(!)	Fatal error: Uncaught Error: Cannot instantiate abstract class Ordinateur in /var/www/html/php-backpack/cours_poo/index.php on line 13			
(!)	Error: Cannot instantiate abstract class Ordinateur in /var/www/html/php-backpack/cours_poo/index.php on line 13			
Call Stack				
#	Time	Memory	Function	Location
1	0.0021	415520	{main}()	./index.php:0

## Attributs et méthodes statiques

- Appartiennent à la classe et non à l'objet
- Peut être appelé sans créer une nouvel objet
- Attributs et fonctions statiques
- Toutes les fonctions et méthodes statiques devront avoir une visibilité public

 Ordinateur.php index.php

```
<?php
abstract class Ordinateur{
    public static $allowedType = ['Ordinateur Portable', 'Ordinateur Fixe'];

    private $_marque;
    private $_model;
    private $_result;

    public function __construct($marque, $model){
        $this->_model = $model;
        $this->_marque = $marque;
    }
}
?>
```

```
<?php
require "Ordinateur.php";
require "OrdinateurPortable.php";

var_dump(Ordinateur::$allowedType);
.
```

```
array (size=2)
  0 => string 'Ordinateur Portable' (length=19)
  1 => string 'Ordinateur Fixe' (length=15)
```



## Attributs et méthodes statiques

```
<?php
require "Ordinateur.php";
require "OrdinateurPortable.php";

$ordi = new OrdinateurPortable( marque: "Apple", model: "Macbook pro", displaySize: "16 pouces");

var_dump($ordi->getAllowedType());
```

```
<?php
abstract class Ordinateur{
    public static $allowedType = ['Ordinateur Portable', 'Ordinateur Fixe'];

    private $_marque;
    private $_model;
    private $_result;

    public function __construct($marque, $model){
        $this->_model = $model;
        $this->_marque = $marque;
    }

    public function getAllowedType(){
        return self::$allowedType;
    }
}
```

Les méthodes de la classe pourront accéder  
grâce au mot clé « self »

```
/var/www/html/php-backpack/cours_poo/index.php:13:
array (size=2)
  0 => string 'Ordinateur Portable' (length=19)
  1 => string 'Ordinateur Fixe' (length=15)
```



# Echanges / Questions

