PHP POO

S.Gounane

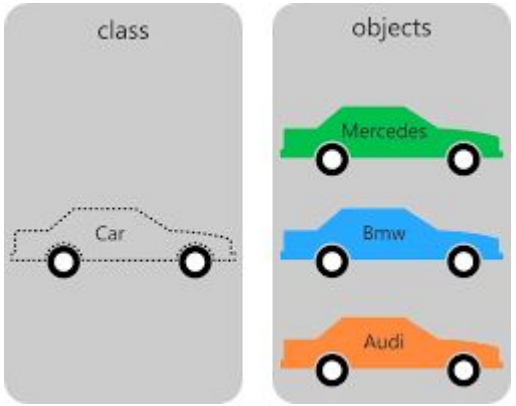
GI 2019-2020

C’est quoi POO

1. La programmation procédurale consiste à écrire des procédures ou des fonctions qui effectuent des opérations sur les données,
2. la POO consiste à créer des objets qui contiennent à la fois des données et des fonctions.
3. La POO présente plusieurs avantages par rapport à la programmation procédurale :
    - La POO est plus rapide et plus facile à exécuter
    - La POO fournit une structure claire pour les programmes
    - La POO facilite la maintenance, la modification et le débogage du code
    - La POO permet de créer des applications entièrement réutilisables avec moins de code et un temps de développement plus court

Class vs Object

1. une classe est un modèle pour les objets
2. Un objet est une instance d'une classe.
3. Lorsque les objets individuels sont créés, ils héritent de toutes les propriétés et comportements de la classe
4. Chaque objet aura des valeurs différentes pour les propriétés.



Définir une classe et créer de objets

Une classe est définie en utilisant le mot - clé ***class***, suivi du ***nom*** de la classe et d'une paire d'accolades (***{}***). Toutes ses propriétés et méthodes vont à l'intérieur des accolades :

fruit.php

```php
<?php
        class Fruit {
                public $name;   // Properté
                public $color; // Properté
                function set_name($name) {           // Methode
                        $this->name = $name;
                 }
                function get_name() {           // Methode
                        return $this->name;
                }
        }
 ?>
```

home.php

```php
<?php
        include(“fruit.php”)
        $apple = new Fruit();
        $banana = new Fruit();
        $apple->set_name('Apple');
        $banana->set_name('Banana');
        echo $apple->get_name();
        echo "<br>";
        echo $banana->get_name();
        var_dump($apple instanceof Fruit);
 ?>
```

Le constructeur d'une class

1. Un constructeur permet d'initialiser les propriétés d'un objet lors de la création de l'objet.
2. PHP appellera automatiquement la fonction ***__construct()*** (deux traits de soulignement (__))

```php
<?php
	class Fruit {
		public $name;
		public $color;
		function __construct($name) {
			$this->name = $name;
		}
		function get_name() {
			return $this->name;
		 }
	 }
	$apple = new Fruit("Apple");
	echo $apple->get_name();
?>
```

Le destructeur

1. Un destructeur est appelé lorsque l'objet est détruit ou que le script est arrêté ou quitté.
2. PHP appelle __destruct() automatiquement à la fin du script.

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
        $this->name = $name;
  }
  function __destruct() {
        echo "The fruit is {$this->name}.";
  }
}
```

Les modificateurs d'accès

- Les propriétés et les méthodes peuvent avoir des modificateurs d'accès qui contrôlent l'endroit où elles sont accessibles.
- Il existe trois modificateurs d'accès :
  - ***public***: la propriété ou la méthode est accessible de ***partout***. ***C'est par défaut***
  - ***protected***: la propriété ou la méthode est accessible au sein de la ***classe*** et par ***les classes dérivées*** de cette classe
  - ***private:*** la propriété ou la méthode est ***UNIQUEMENT*** accessible dans la classe

L’Héritage

Héritage en POO = Lorsqu'une classe dérive d'une autre classe.

La classe enfant héritera de toutes les propriétés et méthodes publiques et protégées de la classe parent.

De plus, il peut avoir ses propres propriétés et méthodes.

Une classe héritée est définie à l'aide du mot - clé extends.

Regardons un exemple : Exemple <?php class Fruit { public $name; public $color; public function __construct($name, $color) { $this->name = $name; $this->color = $color; } public function intro() { echo "The fruit is {$this->name} and the color is {$this->color}."; } } // Strawberry is inherited from Fruit class Strawberry extends Fruit { public function message() { echo "Am I a fruit or a berry? "; } } $strawberry = new Strawberry("Strawberry", "red"); $strawberry->message(); $strawberry->intro(); ?>

L'Héritage

fruit.php

```
Exemple
<?php
class Fruit {
        public $name;
        public $color;
        function __construct($name, $color) {
                $this->name = $name;
                $this->color = $color;
        }
        function intro() {
                echo "The fruit is {$this->name}
        and the color is {$this->color}.";
         }
 }
?>
```

strawberry.php

```
<?php
include("fruit.php");
Strawberry extends Fruit {
        function message() {
                echo "Am I a fruit or a berry? ";
        }
}
$strawberry = new Strawberry("Strawberry",
"red"); $strawberry->message();
$strawberry->intro();
?>
```

Le mot-clé final

Le mot-clé final peut être utilisé pour:

- Empêcher l'héritage de classe

```
<?php

final class Fruit {
// some code

 }  // will result in error

class Strawberry extends Fruit {

// some code }

?>
```

- Empêcher la redéfinition de méthode.

```
<?php
class Fruit {
        final public function intro() {
        // some code
         }
 }
class Strawberry extends Fruit {
         // will result in error
        public function intro() {
                // some code
        }
 ?>
```

Les Constantes

- Les constantes ne peuvent pas être modifiées une fois déclarées.
- Les constantes de classe peuvent être utiles si vous devez définir des données constantes au sein d'une classe.
- Une constante de classe est déclarée à l'intérieur d'une classe avec le mot - clé const.
- Les constantes de classe sont sensibles à la casse.
- Il est recommandé de nommer les constantes en lettres majuscules.

Les Constantes

On peut accéder à une constante

- depuis l'extérieur de la classe en utilisant le nom de la classe suivi de l'opérateur de résolution de portée (**::**) suivi du nom de la constante
- depuis l'intérieur de la classe en utilisant le mot-clé **self** suivi de l'opérateur de résolution de portée (**::)** suivi du nom de la constante, comme ici :

Exemple:

```
<?php
class MathConst {
	const PI = 3.14;
	const E=0.434;
	function showConsts(){
		echo self::PI, self::E
	}
}
echo MathConst::PI;
echo MathConst::E;
?>
```

Classes et méthodes Abstraites

1. Une méthode abstraite est une méthode déclarée, mais non implémentée dans le code.
2. Une classe abstraite est une classe qui contient au moins une méthode abstraite.
3. Une classe ou une méthode abstraite est définie avec le mot - clé ***abstract***

Classes et méthodes Abstraites

Exemple

```php
<?php

    abstract class ParentClass {

        abstract public function someMethod1();

        abstract public function someMethod2($name, $color);

        abstract public function someMethod3() : string;

    }

?>
```