POO : Programmation Orientée Objet en PHP

POO : Programmation Orientée Objet

Pourquoi utiliser la POO

- Réutilisation du code
- Simplification de la syntaxe
- Faciliter la maintenance
- Faciliter la mise à jour

POO : Programmation Orientée Objet

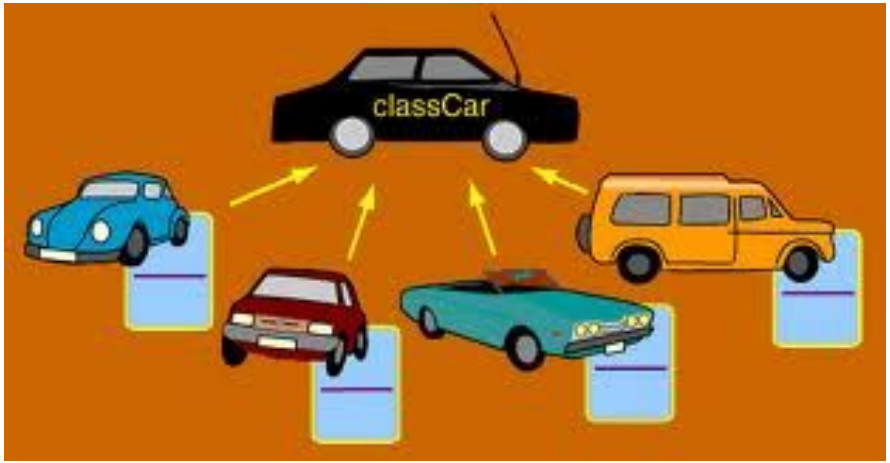
Les mots clés de la POO

- Classe
- Objet
- Attributs
- Méthode
- Constructeur
- Héritage
- Abstraction
- Encapsulation

POO : Programmation Orientée Objet

Classe

Une classe est définition d'un objet. Une sorte de moule permettant de créer plusieurs instances de la classe (**les objets**).
Chaque classe comporte des **variables** et des **méthodes**.



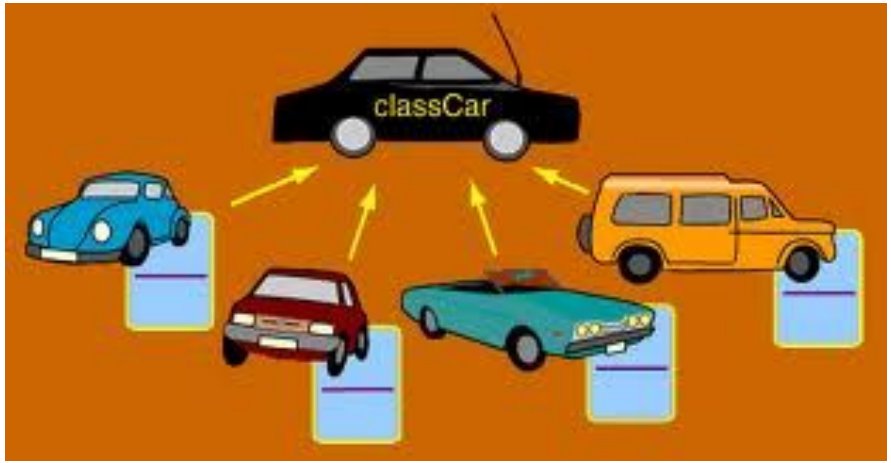

```php
/*
 * Déclaration de la classe Car
 */
class Car{

}
```

POO : Programmation Orientée Objet

Objet

Un objet est une **instance** d'une classe.




```php
/*
 * Objet cox
 * Instance de la class Car
 */

$cox = new Car();
```

POO : Programmation Orientée Objet

Attribut / Propriétés

Ce sont les variables qui sont définies dans la classe et dont les valeurs sont propres à chaque objet.

Par exemple pour la classe Car on peut définir un attribut portes

classe

```php
/*
 * Déclaration de la classe Car
 */
class Car{
   // Attribut portes - Valeur 0 par defaut
   public $porte = 0;
}
```

objet

```php
/*
* Objet cox
* Instance de la class Car
*/

$cox = new Car();
// Affectation d'une valeur
$cox->porte = 4;
// Affichage de la valeur ici 4
echo $cox->porte
```

POO : Programmation Orientée Objet

Méthodes

Ce sont les fonctions sont définies dans la classe et s'appliquent indépendamment à chaque instance d'objet

Par exemple pour la classe Car on peut définir un klaxon

classe

```php
/*
 * Déclaration de la classe Car
 */
class Car{
    // Attribut portes - Valeur 0 par defaut
    public $porte = 0;

    public function klaxon(){
        echo 'tuuuuuut tuuuuut';
    }
}
```

objet

```php
/*
* Objet cox
* Instance de la class Car
*/

$cox = new Car();
$cox->klaxon(); // Affiche tuuuuuut tuuuuut
```

POO : Programmation Orientée Objet

Visibilité d'un attribut ou d'une méthode

indique à partir de quel endroit on peut accéder à une méthode ou un attribut.

public : on peut y accéder depuis l'objet ou depuis la classe

private : on peut y accéder uniquement depuis la classe

classe

```php
/*
 * Déclaration de la classe Car
 */
class Car{
    // Attribut portes - Valeur 0 par defaut
    public $porte = 0;

    public function klaxon(){
        echo 'tuuuuuut tuuuuut';
    }
}
```

objet

```php
/*
* Objet cox
* Instance de la class Car
*/

$cox = new Car();
$cox->klaxon(); // Affiche tuuuuuut tuuuuut
```

POO : Programmation Orientée Objet

```php
/*
 * Déclaration de la classe Car
 */
class Car{
    // Attribut portes - Valeur 0 par defaut
    public  $porte = 0;
    private $codeSecretDemarage = "eaeearezer6546546";

    /*
     * Methode qui verifie la clee et demarre la voiture
     */
    public function demarre($clee){
        if($clee == $this->codeSecretDemarage){
            echo 'la voiture demarre';
        }else{
            echo 'la voiture ne demare pas'
        }
    }
}
```

classe

```php
/*
 * Objet cox
 * Instance de la class Car
 */

$cox = new Car();
// La clée ne s'affiche pas elle est bien cachée
$cox->codeSecretDemarage;
// La cox ne demarre pas
$cox->demarre('321651+65162162615');
// La cox ne demarre
$cox->demarre('eaeearezer6546546');
```

objet

POO : Programmation Orientée Objet

Le Constructeur

Le constructeur est une méthode qui est automatiquement appelée à l'instanciation d'une classe.

classe

```php
/*
 * Déclaration de la classe Car
 */
class Car{

    // Constructeur
    public function __construct(){
        echo 'Bonjour';
    }

}
```

objet

```php
/*
 * Objet cox
 * Instance de la class Car
 */

// Affiche 'Bonjour'
$cox = new Car();
```

POO : Programmation Orientée Objet

Démo

Dans cet exemple nous allons créer une classe Personnage qui comme dans un jeu de rôle peuvent s'attaquer et être soignés, les personnages disposent de points de vie et d'une force d'attaque

Attributs :

- nom
- vie
- atk

Méthodes :

- parler
- attaquer
- regenerer *
- mort *

POO : Programmation Orientée Objet

Documenter ses classes

- Un code bien documenté est un code facile à comprendre
- Permet de créer une documentation avec des outil de génération automatique de doc tels que phpDocumentor ou doxygen
- Les IDE modernes tels que Net Bean permettent une auto-complétion du code
- PHP est un langage faiblement typé, les commentaires donnent une indication sur les type de données attendues et retournés

POO : Programmation Orientée Objet

Documenter ses classes

Descriptif de la classe

L'attribut attendu pour le $nom est une chaîne de caractère

L'attribut attendu pour la $vie est un entier

```php
/**
 * Class personnage
 * Permet de jouer avec des personnages
 */
Class Personnage{

    /**
     *  @var string nom du personnage
     */
    public  $nom = 'Jon Doe';

    /**
     *  @var integer niveau de vie du personnage
     */
    public  $vie = 100;
```

POO : Programmation Orientée Objet

Documenter ses classes

le paramètre $cible attendu est un objet de type personnage

La méthode retourne une valeur de type chaîne de caractère

```php
/**
 * @param $cible object type Personnage
 * @return string
 * Soustrait la valeur de l'attaque au personnage cible
 */ public function attaquer($cible){
    $cible->vie -= $this->atk;
}
```

POO : Programmation Orientée Objet

TP Création d'une classe Formulaire

Dans cet exemple nous allons créer une classe Formulaire qui permet de générer le code HTML des champs de formulaire de type text et submit. Le formulaire pourra être pré-remplis avec des un tableau de valeurs

Vous devez documenter correctement la classe et générer la documentation avec l'outil de votre choix.

Tuto : utilisation Doxygen
ftp://ftp-developpez.com/cyberzoide/java/doxygen.pdf

POO : Programmation Orientée Objet

Propriétés & méthodes statiques

Les propriétés et les méthodes statiques peuvent être utilisées sans avoir besoin d'instancier la classe.

On peut y accéder directement en utilisant le nom de la classe

POO : Programmation Orientée Objet

```php
// Lecture d'un attribut statique
echo Texte::$hello;

// Execution d'une methode statique
echo Texte::formatZero(4);

// Execution d'une méthode statique
// faisant appel à une prppriété statique
Texte::parler();
```

```php
Class Texte{

    public static $hello = "Hello Word";

    /**
     * @var $chiffre integer
     * @return sting
     */
    public static function formatZero($chiffre){
        if($chiffre < 10){
            return '0'.$chiffre;
        }else{
            return $chiffre;
        }
    }


    public static function parler(){
        echo self::$hello;
    }

}
```

POO : Programmation Orientée Objet

Héritage

- L’héritage est un des pilier de la POO, il permet une meilleure organisation du code

- Quand on parle **d'héritage**, c'est qu'on dit qu'une classe B hérite d'une classe A. La classe A est donc considérée comme la classe **mère** et la classe B est considérée comme la classe **fille**.

- Lorsqu'on dit que la classe B **hérite** de la classe A, c'est que la classe B **hérite** de tous les attributs et méthodes de la classe A. Si l'on déclare des méthodes dans la classe A, et qu'on crée une instance de la classe B, alors on pourra appeler n'importe quelle méthode déclarée dans la classe A **du moment qu'elle est publique**.

POO : Programmation Orientée Objet

Un archer dispose de 2 fois moins de vie qu'un personnage et un pouvoir t'attaque 2 fois supérieur au personnage.

Un archer ajoute le texte "Est un archer" à son nom.

```php
Class Archer extends Personnage{

    public function __construct($nom){
        parent::__construct($nom .' Est un Archer');
        $this->vie = $this->vie/2;
    }


    public function attaque($cible){
        $cible->vie -= $this->atk*2;
    }

}
```

POO : Programmation Orientée Objet

Autoloader de Classes

Il est conseillé de séparer chaque classe dans un fichier, or lorsque l'on à besoin de nombreuses classes pour un projet on se retrouve avec un grand nombre de require en tête du code.

L'autoloading viens palier à ce problème en chargeant automatiquement les classe dés que l'on en à besoin.

POO : Programmation Orientée Objet


```php
function __autoload($class_name){
    require('class/' . $class_name . '.php');
}
```

POO : Programmation Orientée Objet

Les design pattern

En développement logiciel, un patron de conception (plus souvent appelé design pattern) est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel.

Il décrit une solution standard, utilisable dans la conception de différents logiciel.

POO : Programmation Orientée Objet

Le pattern Singleton

Le Singleton, permet d'avoir une classe qui sera instanciée qu'une seule fois tout au long de notre application.

Par exemple, dans le cadre d’une application nous aurons une seule et unique configuration. On va donc chercher à instancier cette objet une seule fois pour pouvoir ensuite récupérer l'instance à tout moment de notre application

POO : Programmation Orientée Objet

```php
require('Config.php');

$config = config::getInstance();
$config->dbName = 'autre nom';

$config2 = config::getInstance();
echo $config2->dbName;
```

```php
class Config{

public $dbName  = 'gretaDb';
public $dbPassW = 'gretaPw';


// L'attribut qui stockera l'instance unique
private static $_instance;

/**
 * La méthode statique qui permet d'instancier
 * ou de récupérer l'instance unique
 **/
public static function getInstance()
{
    if (is_null(self::$_instance)) {
        self::$_instance = new Config();
    }
    return self::$_instance;
}

/**
 * Le constrcuteur avec sa logique est privé pour émpêcher
 * l'instanciation en dehors de la classe
 **/
private function __construct(){}
```