POO Exemples

Inspirez vous de ces exemples et tester par vous-mêmes les possibilités de la POO

La plupart des exemples proviennent de la documentation officielle de PHP. N’hésitez pas à la consulter : <https://www.php.net/manual/fr/language.oop5.php>

Si vous vous ennuyiez : <https://www.codingame.com/playgrounds/32339/exercices-de-php-pour-debutant>

Créez les classes suivantes et implémenter les notions vues en cours au travers de celles-ci. Ajoutez toutes les méthodes et attributs que vous souhaitez.

* Classe Calculatrice (méthodes pour additionner et soustraire)
* Classe Voiture (Accélérer, Freiner, Marque, Couleur, vitesse, vitesse maximum, année)
* Classe Rectangle (méthodes pour donner le périmètre et la surface)

# Visibilité des attributs

A faire (au moins 😊 ) :

* Créer une classe qui contient 3 attributs de type texte. Un public, un protected, un private.
* Ajouter une fonction qui affiche les 3 attributs.

La visibilité d'une propriété, d'une méthode ou (à partir de PHP 7.1.0) une constante peut être définie en préfixant sa déclaration avec un mot-clé : public, protected, ou private. Les éléments déclarés comme publics sont accessibles partout. L'accès aux éléments protégés est limité à la classe elle-même, ainsi qu'aux classes qui en héritent et parente. L'accès aux éléments privés est uniquement réservé à la classe qui les a définis.

<?php  
/\*\*  
 \* Définition de MyClass  
 \*/  
class MyClass  
{  
    public $public = 'Public';  
    protected $protected = 'Protected';  
    private $private = 'Private';  
  
    function printHello()  
    {  
        echo $this->public;  
        echo $this->protected;  
        echo $this->private;  
    }  
}  
  
$obj = new MyClass();  
echo $obj->public; // Fonctionne  
echo $obj->protected; // Erreur fatale  
echo $obj->private; // Erreur fatale  
$obj->printHello(); // Affiche Public, Protected et Private

# Visibilité des méthodes

<?php  
/\*\*  
 \* Définition de MyClass  
 \*/  
class MyClass  
{  
    // Déclare un constructeur public  
    public function \_\_construct() { }  
  
    // Déclare une méthode publique  
    public function MyPublic() { }  
  
    // Déclare une méthode protégée  
    protected function MyProtected() { }  
  
    // Déclare une méthode privée  
    private function MyPrivate() { }  
  
    // Celle-ci sera publique  
    function Foo()  
    {  
        $this->MyPublic();  
        $this->MyProtected();  
        $this->MyPrivate();  
    }  
}  
  
$myclass = new MyClass;  
$myclass->MyPublic(); // Fonctionne  
$myclass->MyProtected(); // Erreur fatale  
$myclass->MyPrivate(); // Erreur fatale  
$myclass->Foo(); // Public, Protected et Private fonctionnent

# Constructeur

<?php  
class Point {  
    protected int $x;  
    protected int $y;  
  
    public function \_\_construct(int $x, int $y = 0) {  
        $this->x = $x;  
        $this->y = $y;  
    }  
}  
  
// Pass both parameters.  
$p1 = new Point(4, 5);  
// Pass only the required parameter. $y will take its default value of 0.  
$p2 = new Point(4);  
// With named parameters (as of PHP 8.0):  
$p3 = new Point(y: 5, x: 4);  
?>

# Opérateur de portée

L'opérateur de résolution de portée (aussi appelé Paamayim Nekudotayim) ou, en termes plus simples, le symbole "double deux-points" (::), fournit un moyen d'accéder aux membres [static](https://www.php.net/manual/fr/language.oop5.static.php) ou [constant](https://www.php.net/manual/fr/language.oop5.constants.php), ainsi qu'aux propriétés ou méthodes surchargées d'une classe.

<?php  
class MyClass {  
    const CONST\_VALUE = 'Une valeur constante';  
}  
  
$classname = 'MyClass';  
echo $classname::CONST\_VALUE;  
  
echo MyClass::CONST\_VALUE;  
?>

# Constantes de classe

Il est possible de définir des constantes par classes qui restent identiques et non modifiables. La visibilité par défaut des constantes de classe est public.

Note:

Les constantes de classes peuvent être redéfinies par une classe enfant. À partir de PHP 8.1.0, les constantes de classes ne peuvent pas être redéfinies par une classe enfant si elle a été définit comme finale.

<?php  
class MyClass  
{  
  const CONSTANT = 'valeur constante';  
  
  function showConstant() {  
    echo  self::CONSTANT . "\n";  
  }  
}  
  
echo MyClass::CONSTANT . "\n";  
  
$classname = "MyClass";  
echo $classname::CONSTANT . "\n";  
  
$class = new MyClass();  
$class->showConstant();  
  
echo $class::CONSTANT."\n";  
?>

# Statique

Le fait de déclarer des propriétés ou des méthodes comme statiques vous permet d'y accéder sans avoir besoin d'instancier la classe. Celles-ci peuvent alors être accédées statiquement depuis une instance d'objet.

Méthodes statiques ¶

Comme les méthodes statiques peuvent être appelées sans qu'une instance d'objet n'ait été créée, la pseudo-variable $this n'est pas disponible dans les méthodes déclarées comme statiques.

<?php  
class Foo  
{  
    public static function aStaticMethod() {  
        // ...  
    }  
}  
  
Foo::aStaticMethod();  
$classname = 'Foo';  
$classname::aStaticMethod();  
?>

# Static & Const

Dans le contexte d'une classe, les variables statiques sont sur la portée de la classe (pas l'objet), mais contrairement à un const, leurs valeurs peuvent être modifiées.

class ClassName {

static $my\_var = 10; // defaults to public unless otherwise specified

const MY\_CONST = 5;

}

echo ClassName::$my\_var; // returns 10

echo ClassName::MY\_CONST; // returns 5

ClassName::$my\_var = 20; // now equals 20

ClassName::MY\_CONST = 20; // error! won't work.