php

```
 * @var boolean
 */
define('PSI_INTERNAL_XML', false);

if (version_compare("5.2", PHP_V
    die("PHP 5.2 or greater is r

}
if (!extension_loaded("pcre"))
    die("phpSysInfo requires th
        properly.");

}

require_once APP_ROOT.'/includes/autoloader.inc.p

// Load configuration
require_once APP_ROOT.'/config.php';

if (!defined('PSI_CONFIG_FILE') || !defined('PSI_DE
    $tpl = new Template("/templates/html/error_confi
    echo $tpl->fetch();
    die();
```

php POO

Messmer Lucie

Dans ce cours :

- Introduction au POO
- Objets et Classes
- Instance de classe
- Méthodes
- Visibilité

Exercice POO1

Introduction : Programmation Orienté Objet

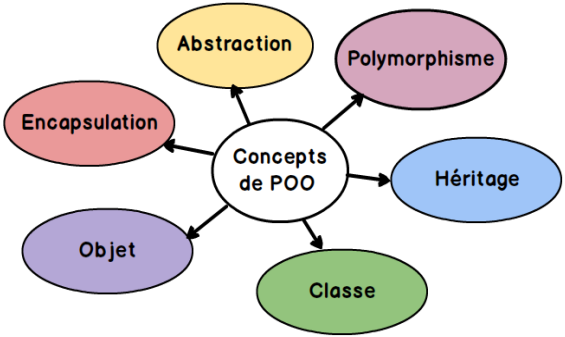
Définition

POO

La programmation orientée objet est un modèle de langage de programmation qui s'articule autour d'objets et de données, plutôt que d'actions et de logique. Par le passé, un programme était une procédure logique qui récupérait des données en entrée, les traitait puis produisait des données en sortie.

L'enjeu de la programmation était d'écrire la logique, pas de définir les données. La programmation orientée objet a changé la perspective : l'important, ce sont les objets à manipuler plutôt que la logique nécessaire à cette manipulation.

Les objets sont très divers : des personnes (définies par leurs nom, adresse, etc.) aux bâtiments et aux étages (aux propriétés descriptibles et gérables) jusqu'aux petits widgets placés sur le bureau de votre ordinateur (comme les



Concept

Les concepts et les règles utilisés en programmation orientée objet procurent les bénéfices non négligeables :

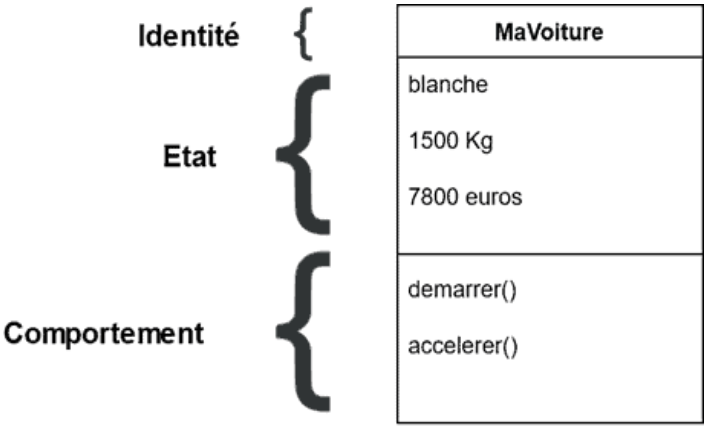
- Le concept de classe de données permet de définir des sous-classes d'objets de données qui partagent certaines voire toutes les caractéristiques de la classe principale. Cette propriété dite « d'héritage » contraint à une analyse poussée des données, accélère le développement et produit un code plus précis.

- Comme une classe définit uniquement les données dont elle doit s'occuper, quand une instance de cette classe (un objet) s'exécute, le code ne peut pas accéder par erreur à d'autres données du programme. Masquer les données est une spécificité qui renforce la sécurité du système et évite de corrompre les données par accident.

- La définition d'une classe est réutilisable par le programme pour lequel on l'a initialement créée, mais aussi par d'autres programmes orientés objet. Elle est donc plus facile à distribuer pour une utilisation en réseau.

- Le concept de classes de données permet à un programme de créer n'importe quel type de données encore indéfini dans le langage lui-même.

Exemple d'objets :




```
class Voiture {
        //Propriétés
        var $couleur ;
        var $masse ;
        var $prix ;

        //Constructeur
        function __construct($couleur, $masse, $prix){
           $this-> couleur = $couleur ;
           $this-> masse = $masse ;
           $this-> prix = $prix ;
        }

        //Méthode
        function demarrer(){

        }

        function accelerer(){

        }
    }

$maVoiture = new Voiture("blanche", 1500, 1800) ;
```

Objet et Classes

Définition

Un **objet**, en programmation, est le nom que l'on donne à l'élément :

- qui se trouve dans une variable ;
- dans lequel on peut stocker des valeurs ;
- et auquel on peut demander de faire des actions.

Pour pouvoir manipuler cet objet, il faut lui donner un nom qu'on va appeler **Classe**.

Une **classe** est une définition qui contient le nom des propriétés qu'on pourra manipuler ainsi que des méthodes. Les **propriétés**, ce sont des variables internes à cette définition dans lesquelles on stocke des valeurs. Une classe a aussi des méthodes : il s'agit de fonctions internes à la classe. La classe détermine ce qu'il sera possible de faire avec l'objet.

Le fait de créer un objet à partir d'une classe s'appelle **instancier une classe**.

La classe serait l'objet en général (stylo), et un objet de cette classe serait un objet spécifique (stylo à plume Mont Blanc).

Créer un obet

On créé un objet comme on créé une variable, mais cet objet, au lieu d'avoir une valeur en aura généralement plusieurs, chacune liée à une propriété différente.

Syntaxe :

$nomDeMonObjet = {

propriété : valeur ,

propriété2 : valeur2 ,

propriété3 : valeur3

};

exemple :

```
$monStylo = {
        "marque" : "Mont Blanc" ,
        "encre" : "noire" ,
        "type" : "stylo à plume" ,
        "prix" : 250
}
```

Créer une classe

Pour définir une classe en PHP, on le fait de la manière suivante :

Syntaxe :

class NomDeLaClasse {

}

NB : Par conventions, le nom des classes prennent une majuscule

Instancier une classe

Instancier une classe signifie créer un nouvel objet à partir de son modèle. Cela est possible grâce au mot-clef **new**.

exemple :

```
class Voiture { }

$maVoiture = new Voiture ;
```

Déclarer le contenu de la classe.

La classe est le modèle sur lequel on va baser nos différents objets. Il lui fait donc définir des propriétés.

Par exemple, définissons la classe Identité :

```
class Identite {
        var $nom ;
        var $prenom ;
        var $age ;
        var $sexe ;
        function __construct($nom, $prenom, $age, $sexe){
                $this->nom = $nom ;
                $this->prenom = $prenom ;
                $this->age = $age ;
                $this->sexe = $sexe ;
      }

}
```

Cette classe est construite sur 4 variables, $nom, $prenom, $age, $sexe. Nous pouvons donc définir son **constructor.** Ce dernier est une méthode appelée lorsque l'on veut instancier un nouvel objet. Il se place dans la classe.

La mot-clef **$this** est disponible lorsqu'une méthode est appelée depuis un contexte objet.
$this est la valeur de l'objet appelant.

Typer nos propriétés

declare(strict_types=1) ;

Cette instruction exige à PHP un certain type en résultat. Elle n'est pas oblgatoire mais est conseillé pour éviter des comportements inatendus;

Faisons le test :

```
declare(strict_types=1);

class Regle
{
  public float $taille = 0;
}

$regle = new Regle;
$regle->taille = 33.0;

var_dump($regle);
```

```
//declare(strict_types=1);

class Regle
{
  public float $taille = 0;
}

$regle = new Regle;
$regle->taille = '33.0';

var_dump($regle);
```

2nd test :

```
var_dump('2Bcarré' + '4Bc');
```

Puis ajoutons l'instruction **declare(strict_types=1);**

Déchiffrons le code précédent :

```
class Regle
{
  public float $taille = 0;
}

$regle = new Regle;
$regle->taille = 33.0;
```

→ définit l'accessibilité de la propriété

→ type de la propriété (bonne pratique)

→ nom de la propriété, précédé de $

→ On assigne la valeur 33 pour notre instance $regle

Les méthodes

Une classe peut définir des fonctions qu'elle seule peut exécuter. Ces fonctions sont appellées méthodes de classes.

Prenons l'exemple d'un pont, nous pouvons par exemple calculer sa surface à l'aide d'une méthode.

```
class Pont
{
  public float $longueur;
  public float $largeur;
}

$pont = new Pont;
$pont->longueur = 286.0;
$pont->largeur = 15.0;

$surface = $pont->longueur * $pont->largeur;

var_dump($surface);
```

Il pourrait être judicieux de créer une fonction dédiée à cette tâche de calcul de surface, pour éviter de faire le calcul à plusieurs reprises.
Cette fonction s'écrierait dans la classe, et serait donc appelée méthode.

```
public function getSurface(): float
{
   return $this->longueur * $this->largeur;
}
```

Appeler une méthode

Pour utiliser une méthode sur une classe il faut continuer à utiliser l'écriture "→" sur l'instance de classe sur laquelle nous voulons l'utiliser.

Pour le pont :

```
$pont = new Pont;
$pont → longueur = 286.0;
$pont → largeur = 15.0;
$surface = $pont → getSurface();
var_dump($surface);
```

Méthode statique

Les classes ne servent pas uniquement à manipuler des propriétés. Elles permettent également d'effectuer des calculs, des actions, ou encore de retourner des informations.

Créer une instance n'est donc pas toujours nécessaire. Hors, si l'on veut utiliser des méthodes sans instance, il faut les déclarer comme **statiques**. Il faut donc faire attention à ne manipuler que des propriétés statiques.

Pour déclarer une méthode statique, il faut utiliser le mot-clef **static :**

```
public static function abonnementValide(int $abonnement): bool { //la méthode à utiliser }
```

exemple :

```
public static function abonnementValide(int $abonnement): bool {
        if ($abonnement == 0) {
                trigger_error(
                        "L'utilisateur n'est pas abonné et ne peut donc pas accéder au site",
                        E_USER_ERROR
                );
        }
        return true ;
}
var_dump(Abo::abonnementValide(1));
```

NB : Ici nous n'avons pas besoin d'utiliser → pour accéder à la méthode, tout simplement car → permet d'accéder aux éléments d'un objet (donc d'une instance), or ici nous souhaitons accéder aux éléments de la classe directement.

Utilisation de : :

Les constantes de classes

Les constantes dans une classe sont définies à l'aide du mot-clef **const**

Syntaxe :

const MACONSTANTE = "ma Valeur" ;

Une constante de classe est par définition statique, elle ne sera jamais modifier et sa valeur sera partagée pour toutes les instances. Elles sont dites valeurs immuables.

Mot-clef self :

self permet de cibler une méthode statique de classe.

Dans l'exemple précédent, nous aurions pu utiliser le code suivant pour vérification :

```
public static function abonnementValide(int $abonnement): bool
  {
     self::abonnementValide($abonnement);

   $this->abonnement = $abonnement;
  }
```

Visibilité

La visibilité d'une propriété, d'une méthode ou 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.

Propriétés :

Les propriétés des classes peuvent être définies comme publiques, privées ou protégées. Les propriétés déclarées sans explicitement utiliser un mot-clef de visibilité seront automatiquement définies comme publiques.

Méthodes :

Les méthodes des classes peuvent être définies comme publiques, privées ou protégées. Les méthodes déclarées sans explicitement utiliser un mot-clef de visibilité seront automatiquement définies comme publiques.

Constantes :

À partir de PHP 7.1.0, les constantes de classes peuvent être définies comme public, privée ou protégée. Les constantes déclarées sans mot clé de visibilité explicite sont définies en tant que public.

Méthode communes aux objets

Dès la définition d'une classe, nous avons accès à un jeu de méthode particulières.

Elles sont prédéfinies par PHP, et sont automatiquement appelées dans la plupart des cas.

Ce jeu de méthodes est un moyen d'intervenir dans le processus de création, modification et suppression des objets de PHP, afin d'y ajouter des comportements personnalisés.

Syntaxe :

Ces méthodes sont reconnaissables puisqu'elles sont précédées d'un underscore **_ .**

_construct et **_destruct**.

_construct :

Cette méthode est appelée automatiquement lors de la création de nouvelle instance de classe à l'aide du mot-clef new.

Le constructeur sert à initialiser des données de départ pour notre objet.

_destruct :

Le destructeur est appelée automatiquement lorsque l'objet est supprimé de la mémoire : en utilisant la fonction unset() ou en remplaçant les données.