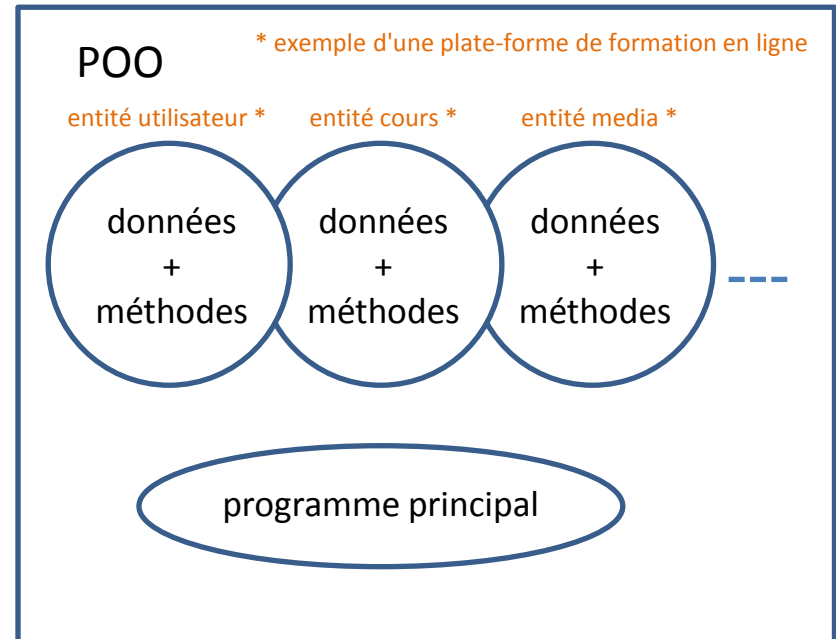
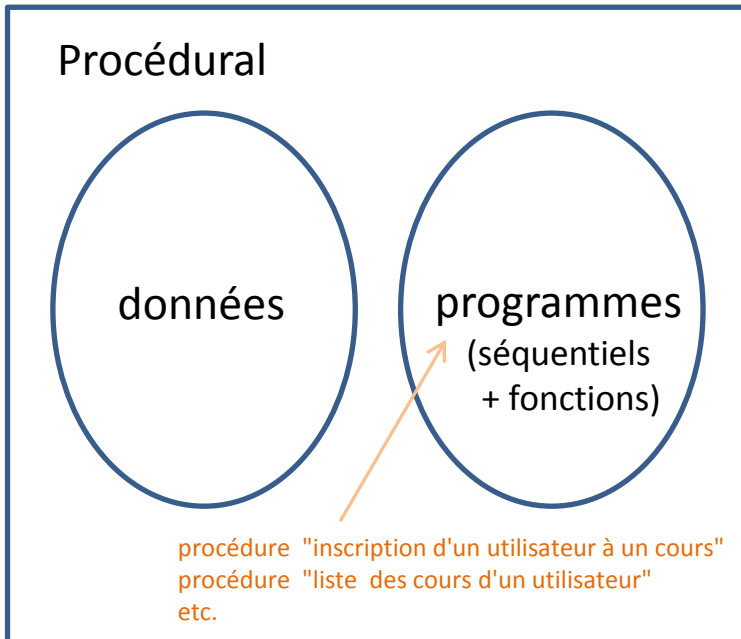


Programmation Orientée Objet en PHP

Introduction

Qu'est ce que la POO ?

Créée par opposition à la programmation procédurale



Dans le modèle POO, il s'agit de créer des entités de codes qui contiennent des données (variables), mais aussi toutes les méthodes (fonctions des entités) associées à l'entité pour exploiter et manipuler ces données.

Pourquoi la POO ?

Quels avantages ?

- Factorisation du code
- Modularité
- Lisibilité du code
- Sécurité du code

Les principes de la POO

- **Encapsulation** : on regroupe dans l'entité toutes les variables et méthodes associées à l'entité. Certaines pourront être protégées du reste du code.

- **Héritage** : une entité peut "hériter" des variables et méthodes communes d'une autre entité.

Par exemple, les entités Professeur et Etudiant héritent des variables et méthodes de l'entité Personne.

- **Polymorphisme** : l'entité qui hérite des méthodes ou variables peut les modifier ou les redéfinir.

Par exemple, l'entité Cylindre qui hérite de l'entité Cercle, possède la même méthode calculerAire() que l'entité Cercle, mais le code de cette méthode est bien sûr modifié par rapport à celui de l'entité Cercle.

Les termes de la POO

- **Classe** : Il s'agit du modèle.
Elle permet de déclarer les comportements et les données que possèdera l'objet créé à partir de celle-ci.
- **Objet** : C'est l'instance d'une classe, une de ses représentations.
Un objet a des valeurs qui lui sont propre.
- **Méthode** : C'est une fonction qui appartient à l'objet.
Cette fonction ne peut être appelée qu'à travers l'objet, si la classe le permet.
- **Propriété** : C'est une variable qui appartient à l'objet.
Elle permet de caractériser l'objet.

Déclaration d'une classe

fichier MaClasse.class.php

```
<?php  
  
class MaClasse  
{  
  
}
```

- Normes d'écriture :
 - Majuscule pour le nom de la classe
 - On déclare une classe par fichier (on peut en déclarer plusieurs)
 - Le nom du fichier indique la classe (ou le dossier)

Les propriétés et leur portée

On déclare les propriétés en début de classe :

```
<?php

class MaClasse
{
    public $maVariable1;
    private $maVariable2 = 0;
}
```

La portée des variables est leur disponibilité pour le reste du code.

3 types de portée :

<i>public</i>	accès universel
<i>protected</i>	accès dans cette classe et ses classes dérivées
<i>private</i>	accès uniquement dans cette classe

Déclaration de fonction

fichier MaClasse.class.php

```
<?php

class MaClasse
{
    public $maVariable1;
    private $maVariable2 = 0;

    public function maMethode1() {
        return true;
    }
}
```

Comme pour les propriétés, 3 types de portée :

- public* méthode utilisable par tous les objets de la classe et de ses classes dérivées
- protected* méthode utilisable dans la classe et ses classes dérivées, mais par aucun objet
- private* méthode utilisable que dans la classe qui la contient

Comment accéder aux éléments ?

fichier MaClasse.class.php

```
<?php

class MaClasse
{
    public $maVariable1;
    private $maVariable2 = 0;

    public function maMethode1() {
        return true;
    }
}
```

L'accès aux propriétés et aux méthodes d'un objet instancié se fait en utilisant le symbole de la flèche représenté par un tiret suivi du signe supérieur : " -> ".

Le nom de la propriété est celui de la variable sans le préfixe \$.

fichier test-MaClasse.php

```
...

$monObjet = new MaClasse; // instantiation d'un objet à partir de la classe
$monObjet->maVariable1;   // accès à cette propriété de cet objet
$monObjet->maMethode1();   // exécution de cette méthode de cet objet
```

```
$monObjet->maVariable2;
```

➡ **Fatal error:** Cannot access private property MaClasse::\$maVariable2

Mot-clé **\$this**

Pour qu'une méthode accède aux variables déclarées dans la classe, elle doit utiliser la syntaxe suivante :

`$this->maVar`

quel que soit le type de portée de `maVar`, qu'elle soit donc déclarée :

```
public    $maVar;  
ou protected $maVar;  
ou private  $maVar;
```

La pseudo-variable `$this` fait référence à l'objet en cours.

Utilisation d'une classe

fichier MaClasse.class.php

```
<?php  
  
class MaClasse  
{  
  
}  

```



fichier test-MaClasse.php

```
<?php  
  
require_once 'MaClasse.class.php';  
  
$monObjet = new MaClasse;  

```

appel d'un fichier dans un autre fichier:

- require
- require_once
- include
- include_once

Exercice 1

Écrivez une classe représentant une personne.

Elle doit avoir les propriétés nom, prénom et date de naissance ('JJ/MM/AAAA').

Une méthode `getPersonne()` doit afficher le texte d'information suivant :

"La personne <prénom> <nom> est née le <date de naissance>"

Créez des objets à partir de cette classe et utilisez la méthode `getPersonne()`.

Le constructeur `__construct($variable1, $variable2, ...)`

Le constructeur est une méthode spéciale de la classe qui va être appelée au moment de la création d'un nouvel objet, instance de cette classe.

Les paramètres de cette méthode sont les valeurs que vous voulez attribuer aux propriétés de l'objet.

fichier MaClasse.class.php

```
<?php
class MaClasse
{
    private $maVariable1;
    private $maVariable2;

    public function __construct($maVariable1, $maVariable2) {
        $this->maVariable1 = $maVariable1;
        $this->maVariable2 = $maVariable2;
    }
}
```

fichier test-MaClasse.php

```
$monObjet = new MaClasse('valeur1', 'valeur2');
echo '<pre>'.print_r($monObjet, true).'
```

```
MaClasse Object
(
    [maVariable1:MaClasse:private] => valeur1
    [maVariable2:MaClasse:private] => valeur2
)
```

Exercice 2

Reprenez l'exercice 1 en le dotant d'un constructeur.

Ne permettre l'initialisation des propriétés qu'à travers le constructeur.

Réalisez les mêmes opérations de création d'objets et d'affichage.

Le destructeur `__destruct()`

La destruction d'un objet est implicite, dès qu'il n'y a plus de références sur cet objet.

Le destructeur est une fonction spéciale sans paramètres qui ne retourne aucune valeur.

Si cette fonction existe, elle est appelée
soit après la destruction explicite de l'objet avec la fonction `unset()`,
soit après la fin du script et la disparition de toutes les références à l'objet.

Peu utilisée, elle permet par exemple d'ajouter une trace dans un fichier log,
ou encore de fermer la connexion à une base de donnée,...

Exercice 3

Reprenez l'exercice 2 en le dotant d'un destructeur.

Ajoutez l'affichage d'un message dans ce destructeur.

Vérifiez son exécution lorsqu'il y a un `unset()` ou après la fin du script .

Accesseurs (getters)

Ce sont des méthodes qui permettent d'accéder aux propriétés de l'objet via celles-ci et non plus directement, lorsque ces propriétés sont 'private' ou 'protected'.

```
class MaClasse
{
    private $maVariable1;
    private $maVariable2;

    public function __construct($maVariable1, $maVariable2) {
        $this->maVariable1 = $maVariable1;
        $this->maVariable2 = $maVariable2;
    }

    public function getMaVariable1() {
        return $this->maVariable1;
    }

    public function getMaVariable2() {
        return $this->maVariable2;
    }
}
```

```
$monObjet = new MaClasse('valeur1', 'valeur2');
echo $monObjet->getMaVariable1();
echo $monObjet->getMaVariable2();
```

Exercice 4

Reprenez l'exercice 3 en ajoutant les accesseurs.

Utilisez ces accesseurs pour afficher les propriétés des objets que vous créez.

Mutateurs (setters)

Ce sont des méthodes qui permettent de mettre à jour les propriétés de l'objet via celles-ci et non plus directement, lorsque ces propriétés sont 'private' ou 'protected'.

Elles sont indispensables pour contrôler l'intégrité de l'information avant de la modifier, c'est le **principe de l'encapsulation**.

```
private $courriel;

public function __construct($maVariable1, $maVariable2, $courriel) {
    $this->setMaVariable1($maVariable1);
    $this->setMaVariable2($maVariable2);
    $this->setCourriel($courriel);
}

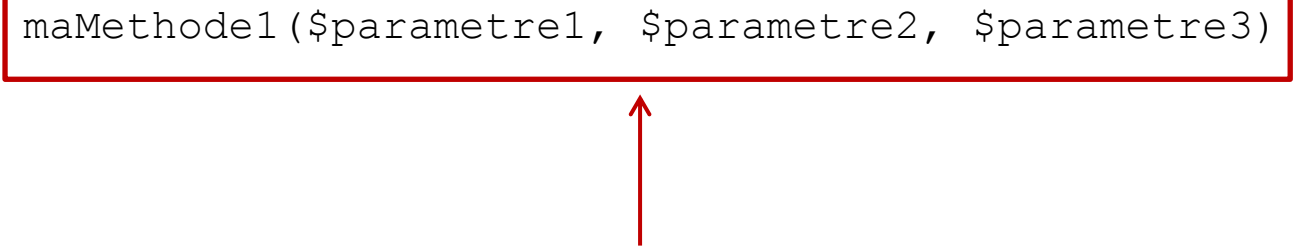
public function setCourriel($courriel = null) {
    $courriel = trim(strtolower($courriel));
    if (filter_var($courriel, FILTER_VALIDATE_EMAIL)) {
        $this->courriel = $courriel;
        return true;
    } else {
        return false;
    }
}
```

Vocabulaire : signature d'une méthode

Dans le codage d'une méthode, **la signature** désigne le nom de la méthode et les paramètres qui sont passés à cette méthode.

Par exemple :

```
function maMethode1($parametre1, $parametre2, $parametre3) {  
    ...  
    ...  
    ...  
}
```



La signature de cette méthode est maMethode1 suivi des 3 paramètres \$parametre1, \$parametre2 et \$parametre3.

Exercice 5

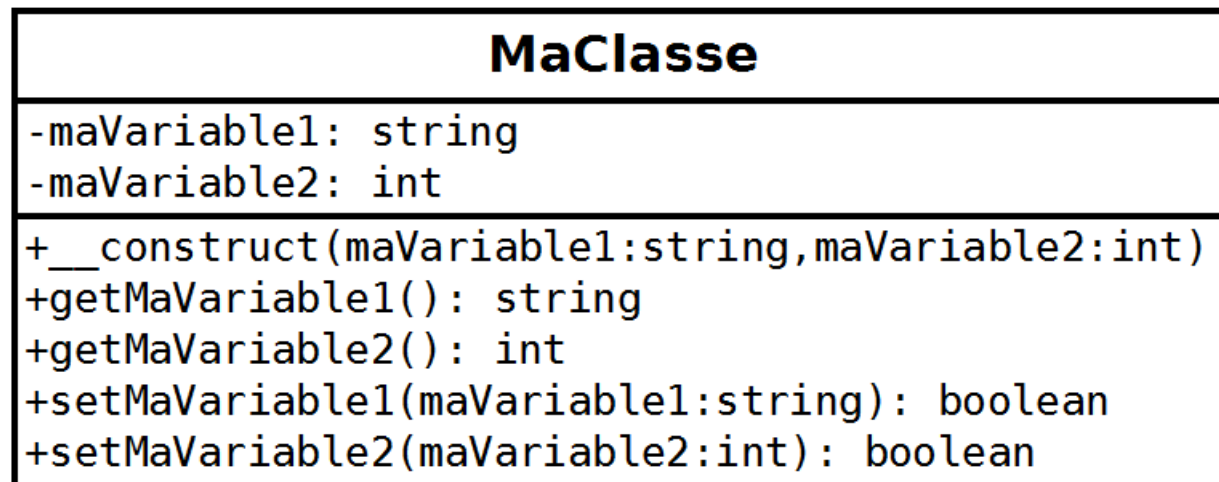
Reprenez l'exercice 4 en ajoutant les mutateurs.

Tester ces mutateurs avec des valeurs correctes et des valeurs erronées.

Représentation graphique

Unified Modeling Language: langage de modélisation graphique

→ **Diagramme de classes** (c'est le diagramme UML le + important)



Éditeur de diagramme DIA: <http://dia-installer.de/index.html.fr>

Éditeur en ligne: <https://www.genmymodel.com>