Initiation à la programmation orientée objet (POO)

V 0.3 (19 octobre 2020) - IT-Akademy (Lyon, octobre 2020)

Benoît Prieur - Soarthec - CC-By-SA 4.0

Introduction

- Paradigme de programmation.
- Permet de représenter les objets et leurs relations.
- Possibilité de représentation graphique : le diagramme de classes en particulier en modélisation UML.
- Nombreux langages objets : C++, Lua, C#, Python etc.
    - On utilise ici les deux langages déjà étudiés : JavaScript et **PHP**.

L’objet : les attributs et les méthodes

- On parle d’**attributs** pour désigner les propriétés, c’est-à-dire les caractéristiques de l’objet.
- On parle de **méthodes** pour désigner les différents comportements de l’objet.

La visibilité

- Un attribut ou une méthode a une **visibilité**. C'est-à-dire que l'on indique qui peut avoir ou non accès au dit attribut.
- La visibilité par défaut dépend du langage : en PHP, la visibilité est publique (*public*).
- Ainsi on utilisera les deux mots-clés suivants relatifs à la visibilité :
  - ***public*** et ***private*** (il existe aussi ***protected***)

La visibilité (2)

- ***public*** : un attribut public est *a priori* visible de partout (la classe peut elle-même avoir une visibilité au sein du paquetage ce qui peut moduler le “partout”).
- ***private*** : un champ privé n'est visible que depuis la classe à laquelle elle appartient. Elle n'est visible nulle part ailleurs (même depuis les sous-classes).

Les attributs

```
class Personne
{
  private $annee_naissance;

  private $nom;
}
```

Les méthodes

- Décrit le comportement de la classe.
- Notion d’accesseur.
- Des méthodes particulières :
  - Le(s) constructeur(s)
  - Notion de destructeur

Les méthodes (2)

Utilisation du mot-clé ***function***.

```
public function afficher()
{
    // Implémentation de la méthode
}
```

Pointeur *this*

- Ce pointeur permet d'obtenir l'adresse de l'instance courante, c'est-à-dire de référencer l'instance courante.
- Quand on utilise ***this***, on se place en quelque sorte du point de vue de l'instance courante, pour accéder directement aux attributs et aux méthodes de l'instance.

Constructeurs et destructeur

- Il existe un **constructeur par défaut**. C'est-à-dire qu'on peut instancier une classe sans avoir décrit un constructeur explicitement.
- On peut expliciter un **constructeur par recopie**. C'est-à-dire un constructeur qui prend en paramètre une instance existante pour en produire une nouvelle, semblable.

Constructeurs

En PHP, on utilise **__construct** pour définir une méthode comme constructeur. Par exemple :

```
public function __construct($n, $a)
{
   $this->annee_naissance = $a;
   $this->nom = $n;
}
```

Les méthodes-accesseurs

- Getter/Setter : typiquement on déclare un attribut en *private* et on y accède en modification et en obtention via deux méthodes publiques.
- Méthode de lecture et écriture d'un attribut
  - Les attributs sont ***private*** ou ***protected***.
  - Les accesseurs sont ***public***.

L’instanciation

La création d’une nouvelle instance se fait par appel de l’un des constructeurs.

Par exemple :

$instance = new Personne(“Charles de Gaulle”, 1890);

L’encapsulation

L’encapsulation désigne le fait d’unifier les caractéristiques et les comportements d’un objet, réel ou non, dans une classe.

Les diverses visibilités participent à l’encapsulation : ce que l’on expose à l’extérieur, ou non.

L'héritage (et surcharge de méthodes)

L'héritage consiste à spécialiser une classe A en une classe B. Ainsi la classe B n'a pas besoin de redéfinir attributs et méthodes issues de la **classe-mère** A. B est une **classe-fille**.

Il est également possible de redéfinir une méthode héritée d'une classe-mère. On parle de surcharge.

L'héritage (2)

En PHP, on utilise le mot-clé ***extends***.

```php
<?php

class Animal
{
   public $nom;
   public function Fairedubruit()
   {
      echo 'je fais du bruit';
   }

}
```

```php
class Chat extends Animal
{
   public function Fairedubruit()
   {
      echo 'je miaule';
   }
}
```

Membres statiques

Un membre statique est un attribut ou une méthode statique ; c'est-à-dire qu'il n'est pas nécessaire d'avoir une instance pour l'utiliser. On peut ainsi appeler une méthode statique directement depuis la classe elle-même, sans recourir à une instance (qui n'existe d'ailleurs pas forcément).

Un mot sur le polymorphisme

Le **polymorphisme** désigne la possibilité qu'une classe-mère ait "plusieurs formes". Ainsi une classe *FormeGeometrique* peut avoir une classe-fille *Cercle*, une autre *Triangle*, etc.

Un mot sur l'héritage multiple

Il est possible dans certains langages de faire hériter une classe de plusieurs classes (en C++ en particulier).

Plus généralement il est possible de faire hériter une classe d'une autre classe et par ailleurs de la faire implémenter différentes interfaces (voir slide suivant pour définition).

Un mot au sujet des interfaces

- Une **interface** définit un contrat entre une classe et ladite interface qu'elle implémente.
- Une interface ne contient que des prototypes de méthodes non-implémentées.
- La classe qui implémente l'interface doit fournir une implémentation pour chacune des méthodes de l'interface.
- On parle d'implémenter une interface (et non hériter d'une interface).

Un mot au sujet des classes abstraites

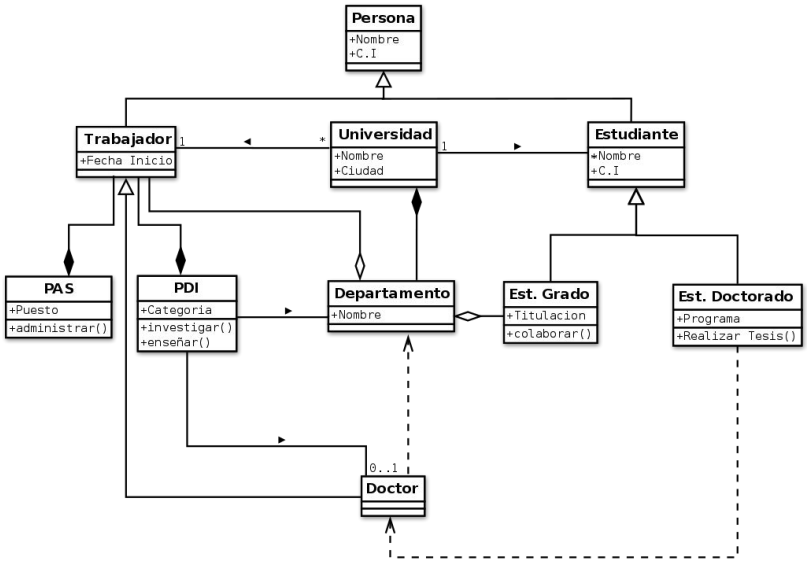
Une **classe abstraite** est une classe qui est non-instanciable et qui en général n'a pas vocation à l'être.

Soit une classe Véhicule et deux classes-filles Bateau et Avion. Il est judicieux de définir la classe Véhicule en classe abstraite.

Le diagramme de classes UML

Une représentation graphique d'un ensemble de classes peut être le diagramme de classes UML, l'un des diagrammes du langage de modélisation UML.

Diagrama de Clases



Troisième jour : sujet (1)

- Créer une classe **Combattant**
  - Il existe trois types de combattants :
    - Junior
    - Confirmé
    - Vétéran

Troisième jour : sujet (2)

- Créer une classe **Arme**
- Une instance de Combattant peut avoir une ou plusieurs armes.
- Chaque arme a des caractéristiques de combat (dégâts infligés au combattant adverse)
- Chaque combattant a une jauge d'énergie ou de vie.

Troisième jour : sujet (3)

- Mettre en place la possibilité de faire un combat entre deux combattants.
- L'interface HTML est encouragée mais facultative.
- Versionner le projet sur Git.
- Commenter et prévoir une documentation succincte (README.txt).

Une solution possible

- Pas de classe Arme pour simplifier l'ensemble.
- Solution impliquant *abstract*, *static*, *extends*, *__construct*. Noter le passage par référence (par opposition à un passage par valeur).
- Utilisation d'un bac à sable en ligne pour tester le code : https://sandbox.onlinephpfunctions.com/
- Code disponible ici : https://github.com/benprieur/IT-Ak-PHP/blob/main/app.php