# Initiation à la programmation orientée objet (POO)

Par Nino SACCOMANI • Publié le 27/10/2015 à 15:56:14 • Noter cet article: ★★★★☆ (4 votes)
*Avis favorable du comité de lecture*

**Nino SACCOMANI**

## Introduction



La programmation Orientée Objet. Impossible de ne pas en avoir entendu parlé. Cette façon de programmer va même jusqu'à rentrer dans la définition de certains langages, qu'on appelle ainsi « langage orienté objet ».
Cet article s'adresse principalement aux élèves de première année, qui n'ont pas encore eu de cours là-dessus, ainsi qu'aux novices qui ne maitrisent pas ce domaine.
Il se veut clair et simple, et n'est pas un cours ni un tuto. Il a pour vocation de vous initier, de vous intéresser et de vous donner envie d'en apprendre plus, peut-être par vous-même.

## Les objets

La programmation orientée objet est un concept, et une façon d'organiser ses programmes. Le C par exemple est, par opposition à un langage orienté objet, un langage dit « procédural ». Ce qui signifie qu'il s'utilise avec des variables simples et des fonctions.
Dans un programme orienté objet, on conçoit un programme à partir d'objets. Qu'est-ce-qu'un objet ? En fait, tout en programmation peut-être un objet. Par définition, on appelle objet une variable complexe c'est-à-dire qu'elle est elle-même composée de variables et de fonctions, contrairement aux variables « scalaires » que sont les caractères, les entiers, les flottants (qui ne sont pas composés d'autres variables ni fonctions).
En soi, un objet est un peu comme une structure de C à qui on ajouterait des fonctions.
Mais vous en avez déjà vu : un exemple d'objet pourrait être une chaîne de caractères ou encore une liste en Python. La chaîne de caractère est composée de plusieurs éléments, que sont ses caractères, ou encore sa longueur, et de fonctions (différentes selon les langages de programmation). La liste en Python est composée d'éléments, et de fonctions ('append()', 'pop()'...)

## Ce qu'on appelle POO

L'intérêt de l'orienté objet est de créer ses objets définis comme on le veut, composés de variables (dites « attributs ») et de fonctions (dites « méthodes ») de notre choix. Ca commence à vous parler ? Des exemples :

| Tour |
| --- |
| ATTRIBUTS<br>-niveau<br>-degats<br>-type_de_projectiles |
| METHODES<br>-monter_de_niveau()<br>-tirer() |

- Vous voulez créer un Tower Defense ? Les tours, les ennemis, ainsi que les projectiles peuvent être des objets. La tour par exemple peut être conçue de cette manière :
- Vous voulez concevoir un programme qui exécute plusieurs processus légers (thread) en parallèle ? Un thread peut être un objet, qui possède des attributs (nécessaires à son fonctionnement) ainsi que des méthodes ('run()' par exemple)
- Vous voulez créer un blog qui contiendra des articles ? Un article peut être un objet, possédant des attributs (son texte, son titre, sa date de création)
- Vous voulez créer un éditeur de fichiers ? Vous pouvez utiliser un objet pour trouver le fichier (contenant des fonctions d'obtention du chemin) et un autre pour le modifier (ce dernier contiendra son chemin comme attribut, ainsi que des fonctions de lecture et d'écriture)

Bref, beaucoup de choses peuvent devenir des objets, et c'est bien pratique.

## De la conception à la création d'un objet

La création d'objet se fait différemment selon le type de langage qu'on utilise. Il y a par exemple les langages objets à prototype, ou ceux à classes. Nous n'aborderons ici que le cas des langages à classes, forme la plus répandue d'orienté objet.
Alors, comment ça fonctionne, en pratique ? Pour avoir un objet, il faut le concevoir, puis le créer (on dit qu'on « l'instancie »). On utilise ce qu'on appelle une classe. La classe permet de définir l'objet, ses attributs et ses méthodes. C'est un peu « le moule » qu'on utilisera pour instancier un objet. La création d'une classe varie selon les langages, voici ici quelques exemples :

```python
class Tour:

    # Attributs
    self.niveau = None
    self.degats = None
    self.type_projectiles = None

    # Méthodes
    def __init__(self):
        """Constructeur"""

    def cibler(self):
        """Méthode cibler"""

    def tirer(self, ennemi):
        """Méthode tirer"""

    def monter_niveau(self):
        """Méthode monter de niveau"""
```

PYTHON

```java
public class Tour {

    // Attributs
    private int niveau;
    private int degats;
    private String typeProjectiles;

    // Méthodes
    public Tour(){
        //Constructeur
    }
    public void cibler()  {
        //Methode cibler
    }

    public void tirer(Ennemi ennemi)  {
        //Methode Tirer
    }

    public void monterNiveau()  {
        //Methode monter de niveau
    }
}
```

JAVA

```php
class Tour
{
  // Attributs
  private $niveau;
  private $degats;
  private $typeProjectiles;

  // Méthodes
  public function __construct() {
      //Contructeur
  }

  public function cibler()
  {
    //Méthode cibler
  }

  public function tirer($ennemi)
  {
    //Méthode tirer
  }

  public function monterNiveau()
  {
    //Méthode monter de niveau
  }
}
```

PHP

Ces images sont juste là à titre d'exemple, elles n'ont pas pour but de vous apprendre comment construire une classe. Ensuite, pour utiliser un objet, une fois que l'on a la classe, il faut l'instancier. On la crée ainsi comme une variable, puis on peut utiliser toutes les méthodes et attributs en utilisant l'objet crée.

```python
tour = Tour()

tour.tirer(ennemi)
tour.monter_niveau()
```

PYTHON

```java
Tour tour = new Tour();

tour.tirer(ennemi);
tour.monterNiveau();
```

JAVA

```php
$tour = new Tour;

$tour->tirer($ennemi);
$tour->monterNiveau();
```

PHP

- L'assemblyInfo va nous permettre de gérer les versions de notre de notre application ainsi que d'autre propriété comme le copyright ou le nom de l'entreprise.
- La première chose qu'on peut voir c'est des déclarations de variables et de fonctions. En effet, on déclare et utilise les attributs comme des variables normales et les méthodes comme des fonctions normales A deux exceptions près :
  - 1) en Python vous pouvez voir un self dans les paramètres des fonctions, ainsi qu'avant un attribut. Ce « self » fait en fait référence à l'objet en cours. Il vous permettra d'accéder à la fonction ou l'attribut en dehors de la classe, et d'accéder dans la fonction aux autres fonctions et attributs déclarées « self » ( en indiquant « self » avant le nom de l'attribut ou de la fonction. Le « self » s'écrit )à la définition d'une fonction mais pas à son appel. Pour accéder aux attributs « self », on remplace le self par le nom de l'instance.
  - 2) En PHP et Java, vous pouvez voir « private » et « public » avant les méthodes/attributs. Ces mots-clés renvoient à l'accessibilité de l'attribut ou de la méthode.
- Simplement : une méthode/attribut déclarée « private » ne sera accessible que dans la classe, alors qu'un(e) déclarée « public » sera accessible partout où l 'objet est instancié. Généralement, il est recommandé de déclarer tous les attributs private et toutes les méthodes public. « Mais comment on accède aux attributs en dehors de la classe alors ? » C'est simple : on crée des méthodes qui servent à récupérer/modifier l'attribut. Ça peut paraitre compliqué, mais ca ne l'est pas vraiment. En JAVA :

```
public void getNiveau()  {
    return this.niveau
}

public void setNiveau(int niveau)  {
    this.niveau = niveau
}
```

« this.niveau » sert juste à dire au programme « prends l'attribut 'niveau' de l'objet en cours ». Le this, comme le self, pointe vers l'objet en cours. Ainsi, comme vous le voyez, la fonctions getNiveau() (pour « acceder niveau ») qui est un accesseur peut être appelée pour récuperer l'attribut niveau ; alors que setNiveau() (pour « définir niveau ») qui est un mutateur permet de le modifier. Ainsi, on peut accéder aux attributs private par cette méthode Le dernier mot-clé est « protected ». En fait, un attribut ou une methode protected sera accessible dans la classe et dans les classes héritent de la classe. On verre plus tard à quoi cela correspond.

- Le mot « constructeur » que j'utilise. Pour le Python le constructeur s'appelle « __init__ », en Java il a le même nom que la classe et en PHP il s'appelle « __construct ». En fait cette méthode va servir à initialiser l'objet. « C'est-à-dire ?». C'est-à-dire que cette fonction sera automatiquement appelle à l'instanciation de l'objet, et doit servir à initialiser les attributs nécessaires au fonctionnement de l'objet. Bien évidemment, cette méthode peut prendre des paramètres, qu'on lui passera à l'instanciation de l'objet.

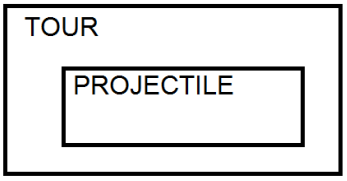
```
def __init__(self, degats):

    self.degats = degats
    self.type_projectiles = "missile"
    self.niveau = 1



tour = Tour(50)
```

Ici on passe le paramètre « 50 » à l'instanciation de l'objet, qui va devenir les degats. On initialise aussi le niveau à 1 et les projectiles à « missile »

## Les relations entre objets

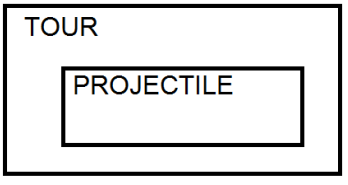
Un autre intérêt de la POO est les relations entre objets. En effet, si vous avez bien suivi vous en avez déjà imaginé un : l'appartenance. Si on instancie un objet au sein d'un autre objet, on pourra utiliser ce premier dans ce dernier. Exemple : Vous avez votre classe « Tour » qui a une fonction « tirer() ». Dans cette fonction vous pouvez très bien instancier un objet « Projectile » qui existera donc dans l'objet « Tour »

```
public function tirer($ennemi)
{
    //Bien évidemment, il faut que la classe Projectile soit déclarée quelquepart
    $projectile = new Projectile()
}
```



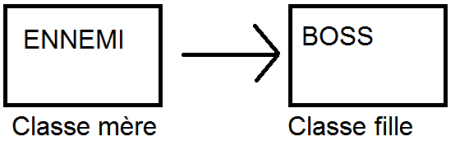

On peut même utiliser l'objet « Projectile » comme attribut de l'objet « Tour ». Ici on définit projecile comme étant un attribut de tour.

```
class Tour
{
  // Attributs
  private $niveau;
  private $degats;
  private $typeProjectile;
  private $projectile

  // Méthodes

  public function tirer($ennemi)
  {
     //Bien évidemment, il faut que la classe Projectile soit déclarée quelquepart
    $this->projectile   = new Projectile()
  }
}
```



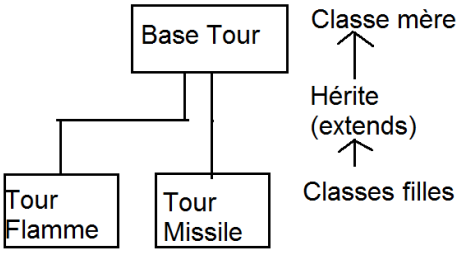

La deuxième relation principale de la POO est l'héritage. Qu'est-ce-que l'héritage ? Si une classe hérite d'une autre, elle récupère ses attributs et ses fonctions. Elle peut en ajouter d'autres, voir même redéfinir ce qu'elle a récupéré.
On dit que la classe fille hérite de la classe mère Cela signifie deux choses : Les objet crées avec la classe fille pourront accéder aux attributs et aux méthodes de la classe mère (s'ils sont public évidemment). Les fonctions de la classe fille pourront réutiliser les attributs et fonctions de la classe mère, s'ils sont public ou protected, ou bien les redéfinir. En JAVA, on utilise le mot clé « extends » :

```
public class Boss extends Ennemi{
```




Ici, on peut imaginer que la classe Ennemi définit les fonctions 'dépalacer()' et 'mourir()' : la classe Boss récupère ces fonctions pour les réutiliser. La classe Ennemi peut définir les attributs 'vie' et 'vitesseDéplacement', que la classe Boss redéfinit (par exemple en ayant plus

de répéter trop de code dans les différentes classes : on les définit dans la classe mère.




## Résumé

- Un objet est une variable complexe composée de variables, dits attributs ; et de fonctions, dites méthodes.
- Tout ou quasiment tout peut être un objet
- Un objet se définit par une classe puis se crée en appelant cette classe : on dit qu'il l'instancie. Les attributs et méthodes peuvent être privés ou publics On accède aux méthodes et attributs publics par la variable instanciée La méthode dite constructrice sert à initialiser l'objet, elle est appelée automatiquement à l'instanciation
- Un objet peut en contenir un autre en variable ou en attribut si on instancie ce dernier dans ce premier.
- Un objet est une variable complexe composée de variables, dits attributs ; et de fonctions, dites méthodes.

## Conclusion

Voilà, on a fait à peu près le tour de la programmation orientée objet. Comme dit, cet article n'a pas pour but de faire de vous des maîtres de l'orienté objet, mais de vous faire découvrir les bases et vous donner envie de découvrir plus par vous-même. J'espère que ça vous a intéresse