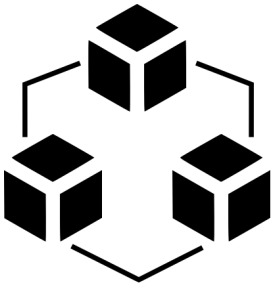
La programmation orientée objet (Poo)



La méthode classique de résolution des problème informatique passe généralement par l'analyse descendante qui consiste à décomposer un problème en sous problème jusqu'à descendre à des actions primitives.
On décompose ainsi un programme en un ensemble de sous-programmes appelés procédures qui coopèrent pour la résolution d'un problème. Cette méthode de résolution appelée programmation impérative, comporte de nombreux inconvénients. En effet les procédures et fonctions sont généralement des outils qui produisent et/ou modifient des données.
L'évolution d'une application développée suivant ce modèle n'est pas évidente car la moindre modification des structures de données d'un programme conduit à la révision de toutes les procédures manipulant ces données. De plus pour de très grosses applications, le développement peut être très long.

La POO résulte de la prise de conscience des problèmes posés par l'industrie du logiciel ces dernières années en termes de complexité accrue et de stabilité dégradée des logiciels développés. En effet la faiblesse de la programmation impérative tient essentiellement du fait que ce mode de programmation manipule une grande quantité de variables qu'il devient difficile d'identifier lorsque le programme devient très important.

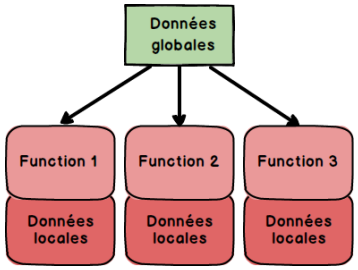


Figure 1: Programmation impérative

La Poo s'inspire du monde réel en faisant interagir des objets plus ou moins complexes à l'aide de messages. Ce paradigme de programmation laisse de côté la notion de variable au profit d'objet ayant chacun ses propres caractéristiques et dialoguant entre eux à l'aide de messages.

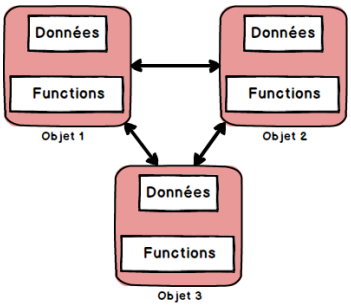


Figure 2: Programmation objet

Repères historiques
La programmation orientée objet a fait ses débuts dans les années 1960 avec le langage Lisp. Cependant elle a été été formellement définie avec le langage Simula (vers 1970) puis SmallTalk. Elle est maintenant incontournable dans les langages récents tels que Java ou Python.

1. Des objets communicants

Dans la programmation orientée objet, les différents **objets** utilisés peuvent être construits indépendamment les uns des autres (par exemple par des programmeurs différents) sans qu'il n'y ait de risque d'interférence.
Ce résultat est obtenu grâce au concept d'**encapsulation** : la fonctionnalité interne de l'objet et les variables qu'il utilise pour effectuer son travail, sont en quelque sorte enfermés dans l'objet.
Les autres objets et d'une façon général, le mode extérieur ne peuvent y accéder qu'à travers des procédures bien définies, c'est ce que l'on appelle l'**interface** de l'objet. Ces objets sont comme un iceberg qui ne laisse apparaître que son sommet au dessus de l'eau - l'interface - mais qui cache sous l'eau une partie importante et inaccessible - le traitement des données – (voir fig 3).

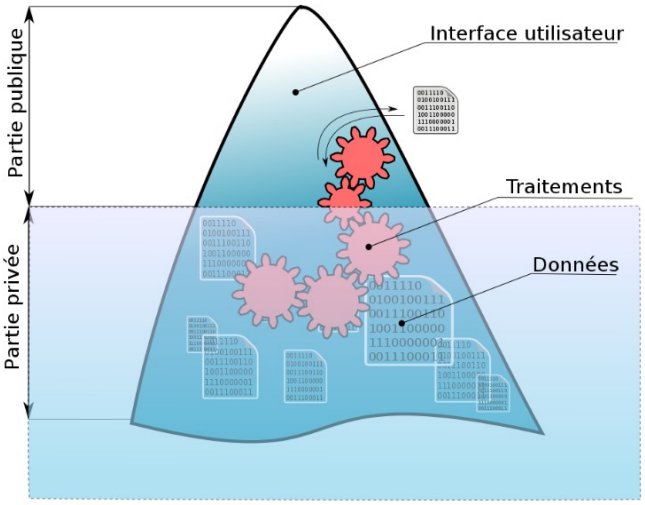


Figure 3: Métaphore de l'iceberg

Il ne faut pas opposer la programmation impérative à l'orienté objet car, au final toute programmation des traitements reste tributaire des mécanismes de programmation procédurale et structurée. On y rencontre des variables, des arguments, des boucles, des arguments de fonction, des instructions conditionnelles, tout ce que l'on trouve classiquement dans les boîtes à outils impératives.

2.Les classes

Comme dans la vie réelle, les objets d'une même famille ont des caractéristiques similaires. Par exemple toutes les voitures ont 4 roues, des portières, une peinture de finition... Dans la programmation objet, une classe est la description générale d'une famille d'objet. La classe est comme un moule à objets, elle définit les données communes (**attributs**) et les actions (**méthodes**) que les objets de la classe pourront réaliser.

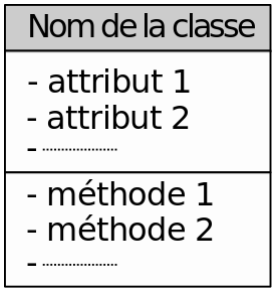


Figure 4: Représentation d'une classe

Un objet est donc un paramétrage particulier d'une classe, ainsi tous les objets d'une même classe possèdent les mêmes méthodes et attributs. Par contre les valeurs des attributs changeront d'un objet à un autre.
La représentation d'une classe peut s'effectuer en suivant le modèle de la figure 4. Ce diagramme est appelé diagramme de classe.

3.Création d'une classe

L'encapsulation désigne le principe de regrouper des données brutes dans un objet avec un ensemble de méthodes permettant de les lire ou de les manipuler. Le but étant de :

- ➔ masquer leur complexité
- ➔ Proposer une interface simple d'utilisation
- ➔ Permettre de les modifier indépendamment du reste du programme

Les attributs et les méthodes des classes peuvent être déclarés en accès :

- ➔ **Public** (**+**) : les autres objets peuvent y accéder et les modifier dans le cas des attributs.
- ➔ **Privé** (**-**) : Les autres objets n'ont pas le droit d'y accéder directement.

Pour maintenir les caractéristiques d'encapsulation, il est d'usage de déclarer les différents attributs en accès privé, ceci afin de contrôler les modification des variables de la classe. Pour accéder aux valeurs des attributs de la classe, il faut alors proposer dans l'interface des méthodes renvoyant les valeurs lorsqu'on les invoque (**accesseurs** / **getters**) et d'autres pouvant les modifier (**mutateurs** / **setters**).
Chaque classe doit définir une méthode particulière appelée **constructeur**. Ce constructeur est une méthode invoquée lors de la création d'un objet, elle permet d'initialiser l'objet.

Prenons pour exemple la numérisation d'un avare (personne proche de son argent), dans sa définition qui est présentée figure 5 nous pouvons dire que :

- ➔ la classe possède deux attributs privés (nom et fortune)
- ➔ Le constructeur permet d'initialiser les attributs de la classe
- ➔ La classe possède un accesseur à l'attribut fortune (`compter()`)

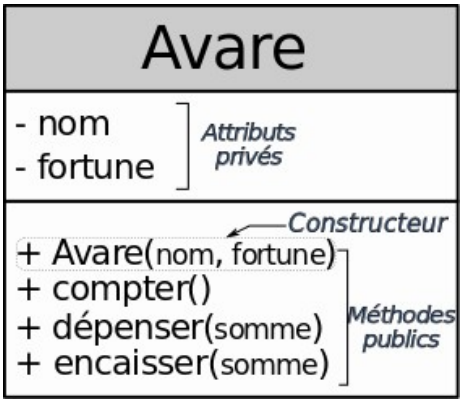


Figure 5: Classe "Avare"

4.Manipulation d'un objet

Une classe est un concept muni de caractéristiques et de propriétés. Pour bénéficier de ces caractéristiques/propriétés, il faut invoquer une instance de cette classe afin de créer un objet.

Création d'un objet

La création objet s'effectue en invoquant le constructeur de la classe. Dans l'exemple de la classe Avare précédente, la création de deux objets « Picsou » et « Gates » de fortune respective 1 000 000$ et 100 000 000 000$ s'effectue de cette façon :

```
>>> balthazar = Avare("Picsou",1000000)
>>> bill = Avare("Gates",100000000000)
>>> bill.compter()
100000000000
>>> balthazar.compter()
1000000
```

L'évolution de la fortune de ces objets s'effectue en appelant les méthodes correspondantes :

```
>>> balthazar.encaisser(100000)
>>> balthazar.compter()
1100000
>>> bill.encaisser(100000)
>>> bill.compter()
100000100000
```

5.Implémentation d'une classe sous Python

On déclare la classe à l'aide du mot clé **`class`** :

```
1  class Avare :
2      ''' Classe Avare :
3      Caractérisé par sa fortune et son nom
4      '''
```

Les lignes 2 à 5 permettent de documenter la classe à l'aide de docstring. Cette documentation est obtenue avec la commande **`__doc__`**

Le constructeur

Le constructeur permet d'initialiser l'objet. Ce constructeur est définit avec **`__init__`**

```
6      def __init__(self, name, fortune) :
7          self.__nom = name
8          self.__fortune = fortune
```

La variable **`self`**, dans les méthodes d'un objet désigne l'objet lui même auquel s'appliquera la méthodes.

Accès public/privé

La limitation d'une méthode ou d'un attribut à un accès privé est effectué en ajoutant un double underscore devant l'attribut/méthode. Par exemple l'attribut `self.__nom` est un attribut privé, pour le rendre public il faudrait écrire `self.nom`

Méthodes particulières

Le langage Python possède un certain nombre de méthodes particulières, chacune avec un nom fixé et entouré d'un double underscore comme pour le constructeur **__init__**. Ces méthodes sont appelée par certaines opérations de prédéfinies de Python.

Méthode	appel	effet
`__str__(self)`	`str(obj)`	Renvoie une chaîne de caractère décrivant l'objet
`__rep__(self)`	Invocation de l'objet dans la console	Renvoie une chaîne de caractère décrivant l'objet

L'implémentation de la classe Avare est donc :

```
1  class Avare :
2      ''' Classe Avare :
3      Caractérisé par sa fortune et son nom
4      '''
5      def __init__(self, name, fortune) :
6          self.__nom = name
7          self.__fortune = fortune
8
9      def __repr__(self) :
10         return 'Fortune de ' + self.__nom + ' : ' + str(self.__fortune) +' $'
11
12     def compter(self) :
13         return self.__fortune
14
15     def depenser(self,somme) :
16         self.__fortune -= 0.9*somme # Un avar ne depense jamais trop....
17
18     def encaisser(self,somme) :
19         self.__fortune += somme
```