Chapitre 2: Introduction à la conception orientée objet

1. Introduction

La conception propose une solution au problème spécifié lors de l'analyse, il existe deux grandes approches de conception, l'approche orientée fonction (approche fonctionnelle) et l'approche orientée objet[1].

1.1. L'approche fonctionnelle

- Raisonnement en termes de fonctions du système ;
- Séparation des données et du code de traitement ;
- Décomposition fonctionnelle descendante.

Les limites de l'approche fonctionnelle

- Un programme est conçu comme un ensemble de modules fonctionnels (procédures ou fonctions) qui manipulent des données ;
- Communication entre fonctions par passage de paramètres ou par variables globales ;
- Accès libre aux données par n'importe quelle fonction ;
- Difficulté de réutiliser du code déjà écrit et testé.

1.2. L'approche orientée objet

- Regroupement données-traitements ;
- Diminution de l'écart entre le monde réel et sa représentation informatique ;
- Décomposition par identification des relations entre objets.

2. Concepts de l'approche orientée objet

2.1. Objet : Un objet est une entité représentée par un état et un ensemble d'opérations qui manipulent cet état.

Un **objet** est caractérisé par :

- un **identificateur** (nom de l'objet) ;
- un **état** (sous forme d'un ensemble d'attributs) ;
- un **comportement** (un ensemble d'opérations).

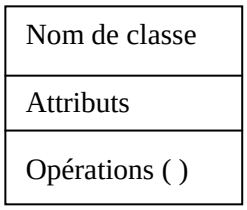
[1]L'approche de conception orientée objet est basée sur la notion d'objet qui représente les entités du monde réel.

2.2. Classe : Une classe correspond à la description d'une famille d'objet ayant une même structure et un même comportement.

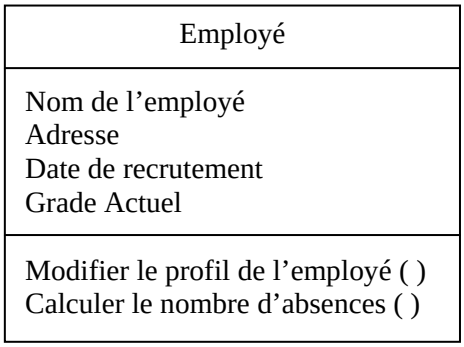
Une classe définit, une partie statique et une partie dynamique :

- La partie statique est représentée par un ensemble **d'attributs** pouvant posséder une valeur (ces attributs caractérisent l'état des objets durant leurs exécutions).
- La partie dynamique est représentée par l'ensemble des **opérations** qui définissent le comportement commun aux objets de la même classe.

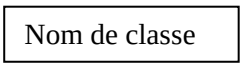
Notation graphique[2] :

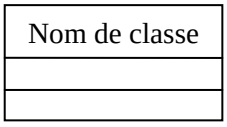


Exemple de classe :



Remarque : Une classe peut être citée avec uniquement son nom, sans en préciser les détails.



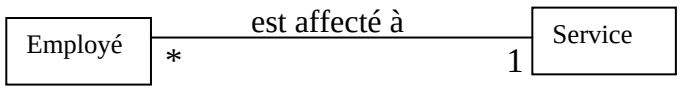


Instance d'une classe: Une instance est un objet créé à partir d'une classe. La création d'un objet à partir d'une classe s'appelle l'instanciation.

[2]Les notations graphiques utilisées dans ce cours sont inespérées d'UML (Unified Modeling Language).

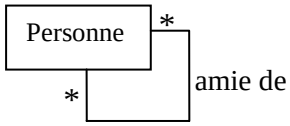
2.2.1. Association : L'association représente une relation entre plusieurs classes. Elle correspond à l'abstraction[3] des liens qui existent entre les objets dans le monde réel. Une association peut être identifiée par son nom. Il est possible d'exprimer les multiplicités[4] (cardinalités) sur le lien d'association.

Exemple :



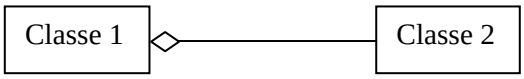
Dans cet exemple, on représente le fait qu'un employé est affecté à un seul service mais que ce dernier peut accueillir plusieurs employés à la fois.

Remarque : Lorsque le lien existe entre des objets de la même classe, on parle d'association réflexive.
Exemple : Deux personnes peuvent être amies (on considère que l'amitié est réciproque).

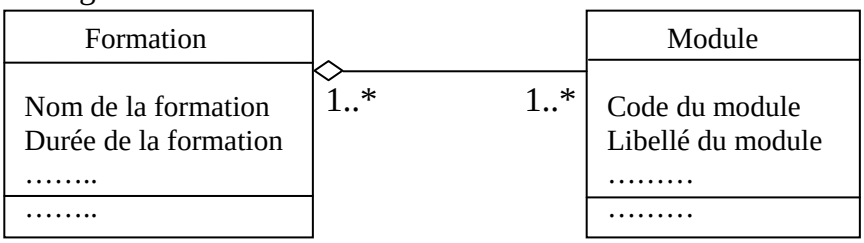


2.2.1.1. Agrégation et Composition:
Il existe des liens d'association particuliers, tel que l'agrégation et la composition :
1) Agrégation : L'agrégation est une association qui décrit une relation d'inclusion entre une partie et un tout (l'agrégat). L'agrégation se représente par un petit losange blanc du côté de l'agrégat.

Notation graphique :



Exemple :
Dans le cadre d'une formation, le cursus de cette dernière est une agrégation de modules à enseigner.



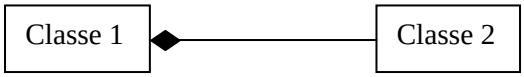
Remarquons dans cet exemple, que la suppression d'une formation ne conduit pas automatiquement à la suppression des modules étant donné que ces derniers peuvent très bien être enseignés dans d'autres formations.

---

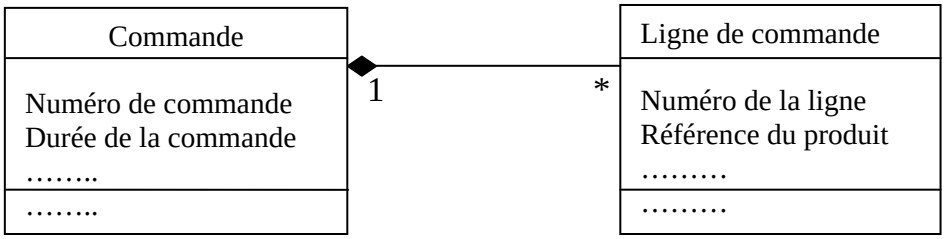
[3]L'abstraction consiste à identifier les caractéristiques intéressantes d'une entité en vue d'une utilisation précise.
[4]D'autres multiplicités sont présentées dans le chapitre 3. Les multiplicités utilisées dans ce chapitre sont :
1 un et un seul
0..1 zéro ou un
* de zéro à plusieurs
1..* de un à plusieurs

2) Composition : Une composition est une forme forte d'agrégation. C'est-à-dire que la suppression de l'objet agrégat mène à la suppression des objets agrégés. La cardinalité du côté composite ne doit pas être supérieure à 1 (1 ou 0..1). La composition se représente par un petit losange de couleur noire.

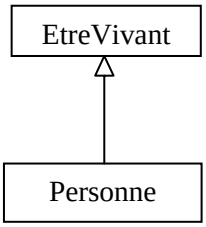
Notation graphique :



Exemple :
Une commande est composée d'un ensemble de lignes de commande décrivant les produits commandés. La suppression d'une commande conduira obligatoirement à la suppression de toutes ses lignes.



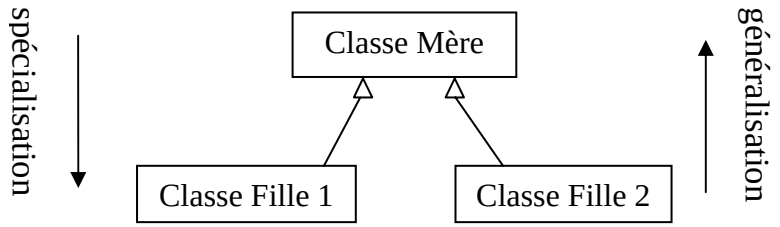
2.3. Héritage : L'héritage permet un partage hiérarchique de propriété (attributs et opérations).
La relation d'héritage entre deux classes est représentée par une flèche à la tête en forme de triangle blanc.
Exemple : L'exemple suivant représente la classe Personne qui hérite de la classe EtreVivant.



L'héritage est mis en œuvre grâce à deux propriétés qui sont : la **généralisation** et la **spécialisation**.

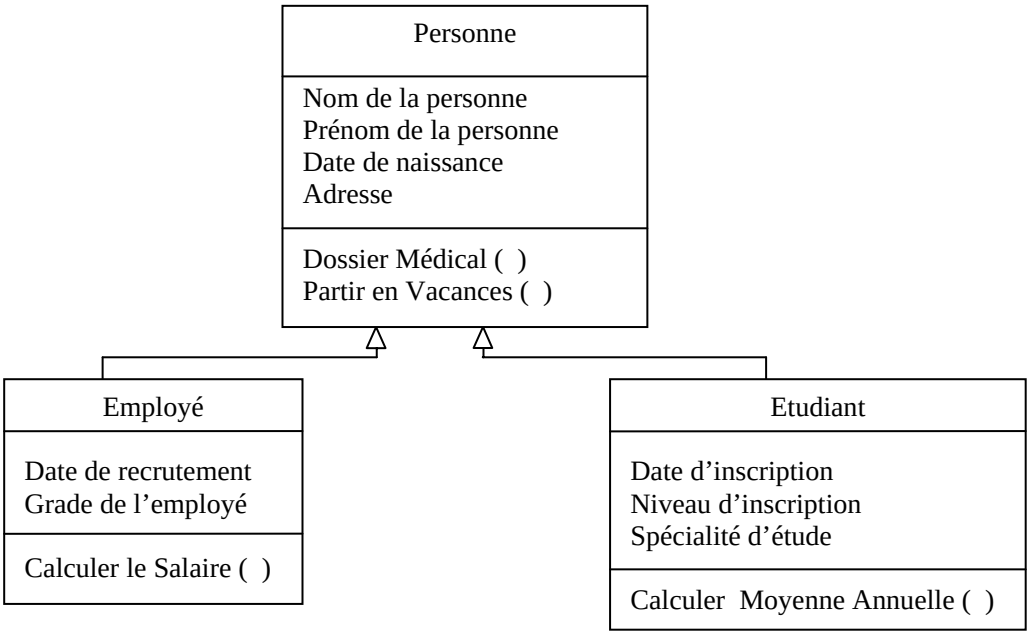
- La généralisation décrit le fait de pouvoir regrouper un ensemble de classes partageant des éléments en commun en une seule superclasse (ou classe mère)
- La spécialisation représente le phénomène inverse, c'est-à-dire, pouvoir dériver à partir d'une classe ou superclasse des sous classes (ou classes filles) ayant des propriétés spécifiques les distinguant les une des autres.

Exemple :



* L’héritage est simple lorsque une classe hérite d’une super classe

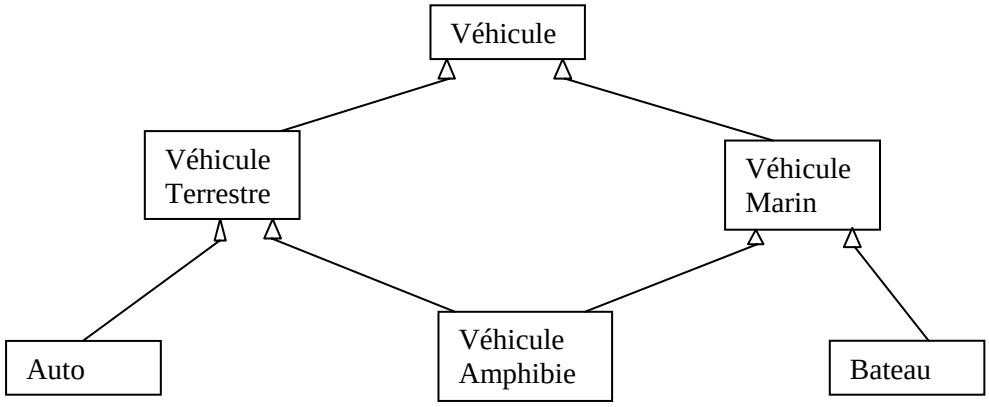
Exemple d’héritage simple:



Les classes « Employé » et « Etudiant » sont des dérivés de la classe « Personne » et héritent des propriétés de cette dernière. Cependant, « Employé» se distingue par d’autres attributs tels que la date de recrutement, le grade et le salaire alors que « Etudiant » possède une date d’inscription, un niveau d’étude, une spécialité et une moyenne annuelle.

* L’héritage est multiple quand il y a deux ou plusieurs super-classes pour une même sous-classe.

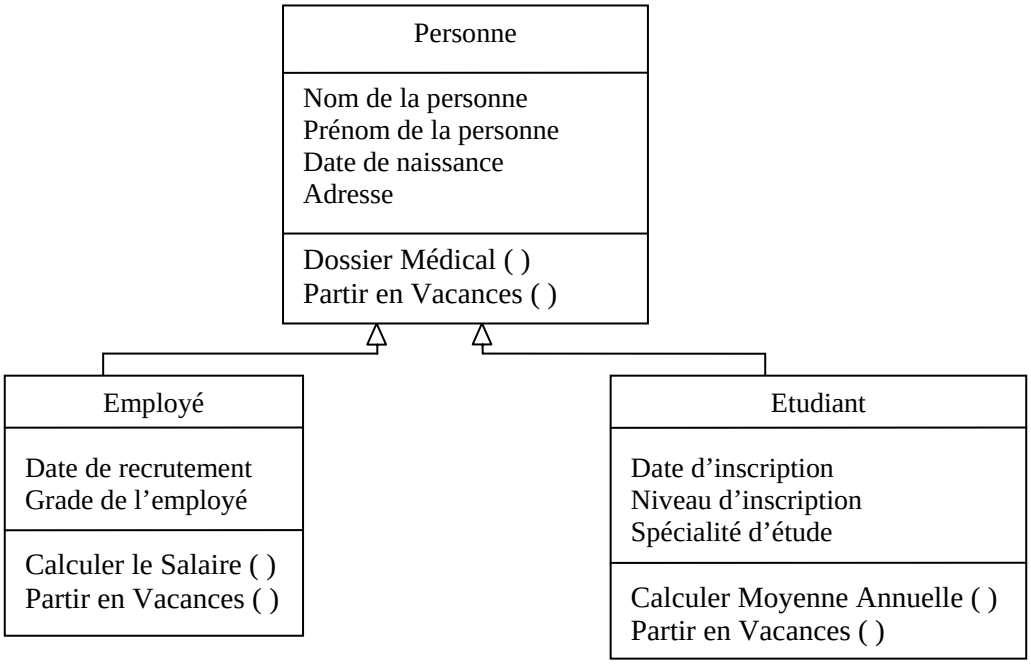
Exemple d’héritage multiple:



La classe « Véhicule Amphibie » hérite à la fois de la classe « Véhicule Terrestre » et à la fois de la classe « Véhicule Marin ».

2.4. Polymorphisme : Il consiste, tout en gardant le même nom pour une méthode héritée, à associer un code spécifique qui vient ainsi se substituer à celui de la méthode héritée.

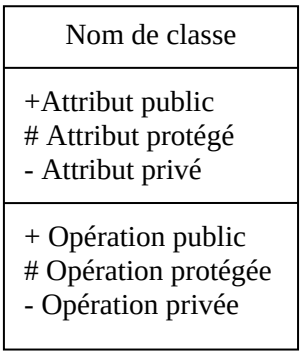
Exemple :
Si l'on reprend l'exemple de l'héritage simple, l'opération « Partir en Vacances ( ) » dont héritent les classes « Employé » et « Etudiant » pourrait avoir des implémentations différentes pour ces deux sous classes. D'où, il serait préférable d'implémenter cette opération par deux méthodes différentes : une pour la sous classe « Employé » et une pour la sous classe « Etudiant ».



2.5. Encapsulation : L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet. L'encapsulation permet de définir des niveaux de visibilité des éléments de la classe.
On distingue trois niveaux différents de visibilité: public, protégé et privé. L'encapsulation est représentée par un signe plus « + » dans le cas de pubic, un signe moins « - » dans le cas de privé et un dièse « # » dans le cas de protégé. Le tableau suivant détaille la signification de ces signes :

public	+	élément non encapsulé visible par tous
protégé	#	élément encapsulé visible dans les sous classe de la classe
privé	-	élément encapsulé visible seulement dans la classe

Notation graphique :



Exemple :