Notes du cours d'analyse (ANA2)

Nathan Furnal

15 mai 2021

Table des matières

1 Qu'est-ce que l'analyse ?

Tout d'abord, il faut se rappeler du vocabulaire des cours précédents. On va parler de :

projet informatique Un projet informatique est tout ce qui touche à la réalisation d'un système informatique par une équipe de développement. Il y a un début et une fin, avec des délivrables. Il est constitué d'un Système d'information (SI), lui même composé d'un Système d'Information Automatisé (SIA).

Système d'information Un système d'information contient toutes les informations collectées sur le projet. Les besoins de l'utilisateur, l'organisation de l'entreprise, l'esprit de l'entreprise, les règles internes.

Système d'information automatisé Un système d'information automatisé est la partie automatisée du système d'information, tout ce qui tourne sur un processeur. On peut aussi l'appeler système informatique.

Le projet commence par une pré-étude pour établir des buts et des besoins, la faisabilité du projet. Puis, le projet évolue dans sa planification et dans le même temps on développe l'application et l'infrastructure nécessaire. Une fois que le projet est mis en production, on en fera l'exploitation, la maintenance ; corrective comme évolutive. Enfin, le projet évolue ou atteint sa fin de vie utile.

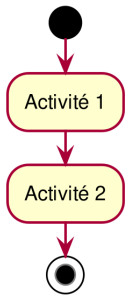
Maintenant que le cadre est posé on peut comprendre le rôle de l'analyse. C'est de modéliser les différentes étapes du projet et plus particulièrement du développement pour aider à comprendre et expliquer, grâce à une interaction avec le client. Les diagrammes permettent de partager le contenu de l'analyse à un public plus large que juste les développeurs. Il y a donc un rôle de **modélisation**, c'est-à-dire de simplification synthétique pour représenter un besoin du projet ou une partie de système.

2 Diagramme d'activités

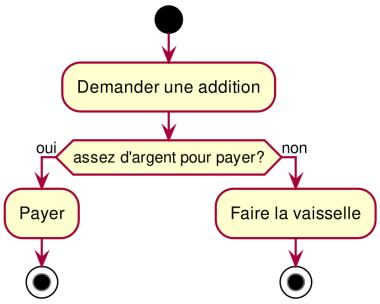
Le diagramme d'activités représente la dynamique du système d'information automatisé (SIA), ainsi que l'enchaînement d'activités. On peut l'utiliser à plusieurs niveaux de modélisation, que ce soit pour les programmes, les méthodes ou les instructions. Le diagramme d'activités décrit un **comportement**.

2.1 Types de diagrammes

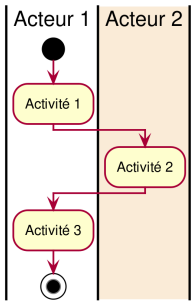
On va utiliser le langage UML (*Unified Modeling Language*) pour représenter les activités. Un diagramme d'activité possède un et un seul début mais, via des conditions, peut avoir plusieurs fins. On notera aussi que la forme des composantes a une signification. Les activités sont un rectangles arrondis, les conditions des losanges, les débuts et fins des cercles pleins ou vides, etc. Voici un exemple :



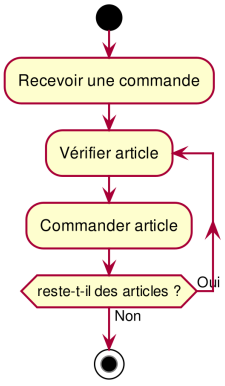
Et voici un autre exemple avec des conditions :



Un exemple de swimlanes, une interaction entre acteurs (machines, utilisateurs, développeurs,...) qui sont parties prenantes du SIA. L'utilisation de couloirs permet de montrer qui est responsable de quoi.



Et finalement un exemple d'itération.



2.2 Exercices

La correction des exercices de la section diagrammes d'activités est disponible sur le Drive.

De manière générale, un diagramme d'activité a un **et un seul** point de départ car on doit pouvoir toujours commencer de la même manière. Aussi, passer d'une activité à une autre doit être **déterministe** c'est-à-dire qu'une même activité doit toujours se comporter de la même manière ; de ce fait, une transition sortante seulement existe par activité. Enfin, une activité ne peut pas être un cul-de-sac, elle doit mener à un nœud final qui résout le diagramme. À noter qu'il peut y avoir plusieurs nœuds finaux et qu'une activité peut avoir plusieurs transitions **entrantes**.

3 Les classes et objets

Pour cette section, nous nous concentrons sur une description de la **structure** et non plus du comportement. En décrivant les **classes** et les **objets**.

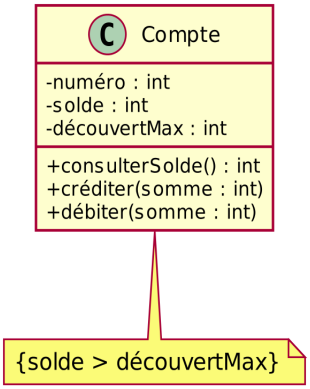
3.1 Classe

Une classe décrit un concept général dans le sens du code, comme une `Vidéo`, un `Point`, etc. De nombreux exemples sont repris dans les slides et dans le cours de DEV2.

Ce qui nous intéresse est qu'une classe est une déclaration d'état et via des **attributs** et de comportement via des **méthodes**, on les sépare par une ligne. Attention, on ne décrit pas le comportement de la méthode elle-même, ce n'est pas le rôle du diagramme de classe. Pour tous les attributs et méthodes, on y attachera un **type** qui permet de les décrire. Aussi, la **visibilité** est importante.

Les membres (attribut ou méthode) publics sont notés par un `+`, ceux qui sont protégés par un `#`, privés par un `-` et package par un `~`.

Voici un exemple avec une classe `Compte` avec une note précisant une contrainte, on peut aussi utiliser des traits.

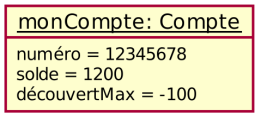


3.2 Objet

Un objet est un élément du monde réel, là où une classe est plus proche d'un plan ou d'un concept. De ce fait, les objets ont des valeurs attribuées à leur état, ce qui n'est pas le cas des classes.

De manière générale, un objet sera une *occurrence* ou une *instance* d'une classe. Là où la classe est l'idée générale, un objet est une réalisation de cette idée avec des valeurs qu'on lui attribue. La classe spécifie la structure et l'objet est une création qui suit cette structure.

Voici un exemple de classe `Compte`, instanciée via un objet `monCompte`.

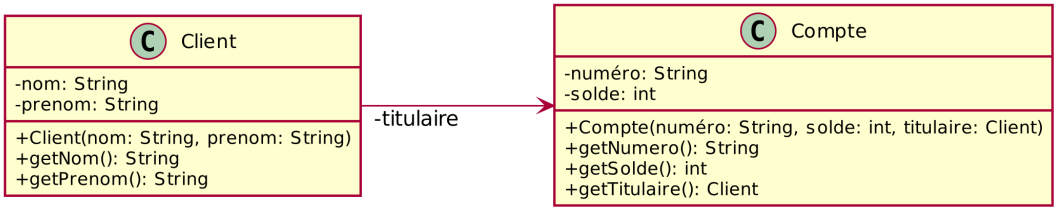


Ici, on remarque que les valeurs des attributs sont précisées mais qu'on ne donne pas les méthodes, en effet le comportement de l'objet est défini par sa classe, il n'est pas précisé à nouveau dans l'UML de l'objet.

4 Les associations $1-1$ et $1-N$

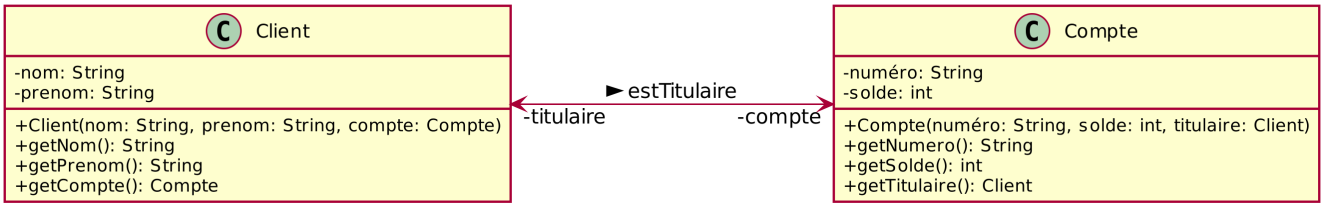
Une association exprime une connexion durable entre classe. Plus précisément, elle permet de noter que des classes sont liées, par logique ou par choix. De plus, on veut pouvoir montrer cette liaison dans le graphe UML.

Prenons par exemple, un compte bancaire appartenant à un client. On va vouloir montrer l'association qui existe entre un client et son compte ou bien entre le compte et le client à qui il appartient. L'association doit avoir un nom descriptif compréhensible et les rôles de chaque classe dans l'association doivent être clairement affichés.



Ici, il y a plusieurs informations intéressantes, le **constructeur** de `Compte` nécessite un client mais il n'est pas repris dans les attributs de la classe. C'est parce que c'est une information redondante. En effet, en notant dans l'association qu'un compte nécessite un client, on a déjà dénoté cet attribut, on ne doit pas l'afficher à nouveau. De plus, la flèche est unidirectionnelle et signifie qu'on peut seulement accéder au client depuis le compte, mais pas l'inverse.

Voici une seconde solution, bidirectionnelle.



Encore une fois, les attributs qui apparaissent par l'association ne sont pas repris dans les attributs du diagramme, on évite la répétition.

Ce n'est pas tout, les objets aussi peuvent être liés. On doit donc préciser le vocabulaire. Un objet et une **instance** d'un classe et un **lien** entre objet est une **instance** d'une association entre classes.

Voici un court résumé :

Diagramme d'objets	**Diagramme de classes**
Instances de classes	*Classes*
Liens entre les instances	*Associations* entre les classes

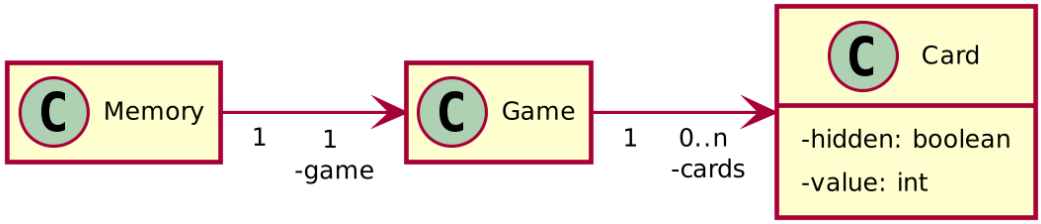
5 Les associations $N - N$

Une association n'est pas toujours de type 1 pour 1 comme montré au-dessus, on peut avoir des associations plus nombreuses. La **multiplicité** désigne le nombre d'objets qui peuvent participer à une association.

Les multiplicités dépendent du contexte et doivent couvrir tous les cas possibles, elles sont notées en général :

- Syntaxe : {min}..{max}
  - 0..1
  - 1..1 ou 1
  - 0..n ou 0..* ou *
  - 1..n ou 1..*

Voici un exemple avec le jeu *Memory* réalisé en DEV2. Un *Memory* a bien un et un jeu (*Game*), un jeu comporte autant de cartes qu'on veut.

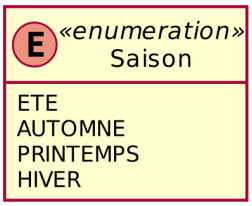


6 Les compositions et énumérations

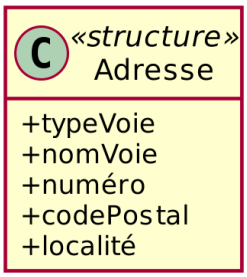
On peut utiliser d'autres types de classes, en UML, qui sont notées avec le symbole suivant «». On précise le type de données entre les chevrons. On peut soit créer une **énumération** qui contient une liste connue et limitée d'instance ou bien une **structure** qui est la modélisation d'un attribut décomposable ou composé.

Ces classes ne peuvent pas avoir d'associations.

Voici par exemple les quatre saisons sous forme d'énumération, un exemple plus complet et repris dans les slides.



Voici un exemple de classe **structure** :



6.1 Association réflexive

Il existe des cas de figure où les associations font référence à elles-mêmes, lorsqu'il y a des liens entre objets d'une classe.

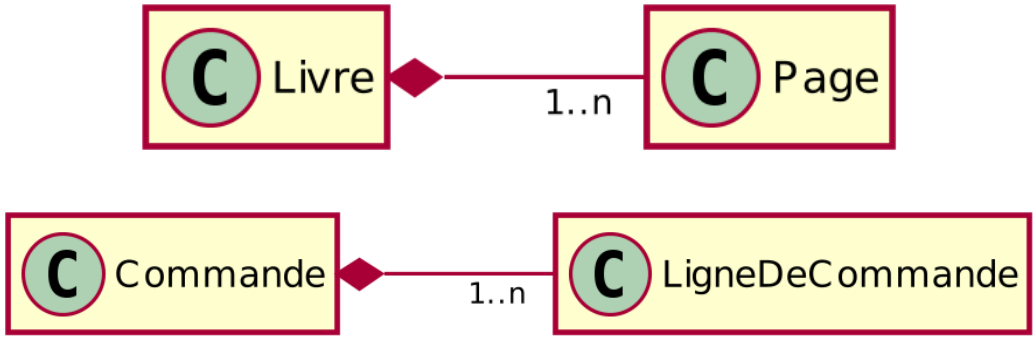
Un exemple est celui des cours, certains cours sont pré-requis d'autres, même si ce sont tous des cours.

6.2 Composition

Une composition ajoute une sémantique pour pouvoir signifier qu'un objet est composé d'un autre ou bien fait partie d'un autre.

- Un objet *composant* ne peut être que dans 1 seul objet *composite*
- Un objet *composant* n'existe pas sans son objet composite
- Si un objet *composite* est détruit, ses composants le sont aussi

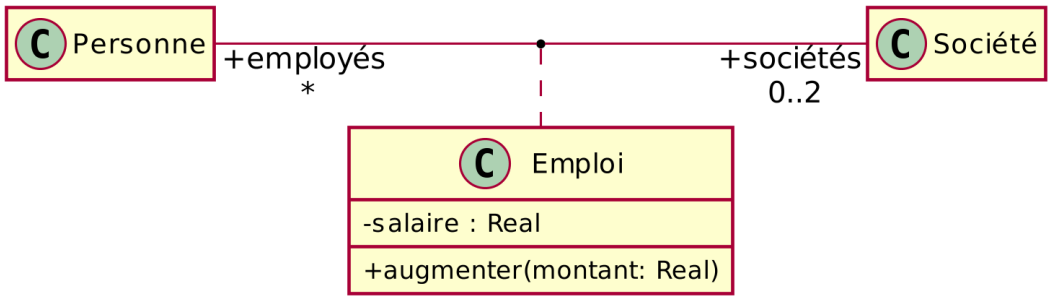
Voici des exemples de composition :



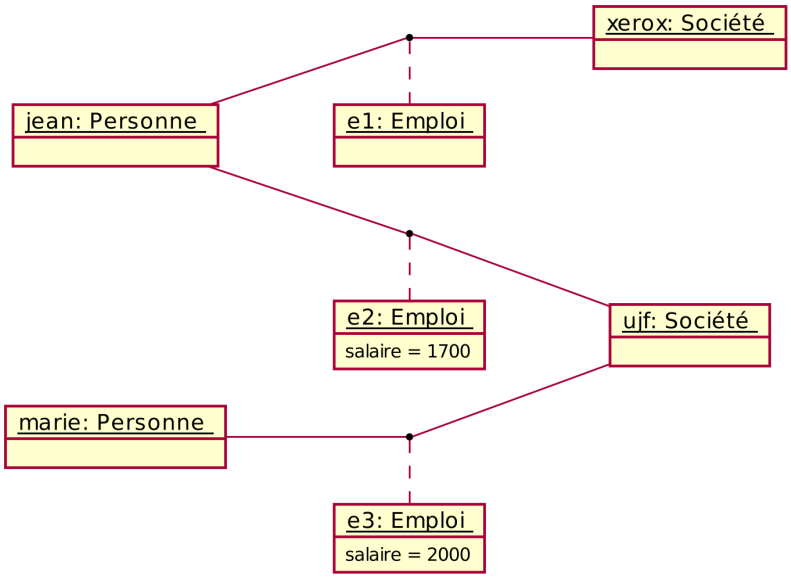
7 Les classes d'associations

Une classe d'association (ou classe-association ou classe associative) est un élément de modélisation UML qui a les propriétés d'une classe et d'une association. Ce qui signifie qu'une connexion durable entre deux classes, est une classe.

Aussi, il faut continuer à respecter l'unicité d'association entre couple d'objets. C'est-à-dire qu'un couple d'objets ne peut être connecté que par un seul lien pour une **association précise**.



Quand ce type de relation est instanciée par des objets, on peut imaginer plusieurs liens tant que chaque instance de la classe d'association est unique à un couple d'objets.

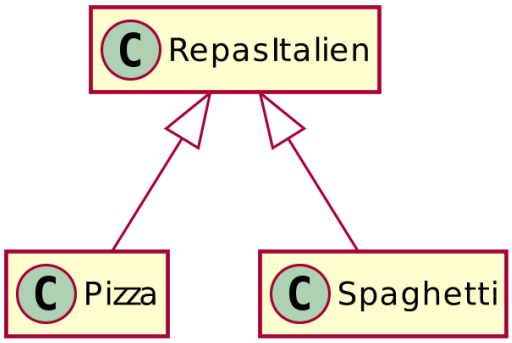


Attention, les exemples ci-dessus ne sont que ça, des exemples. On peut aussi imaginer des classes d'associations plus complexes avec des relations à plusieurs classes.

8 L'héritage

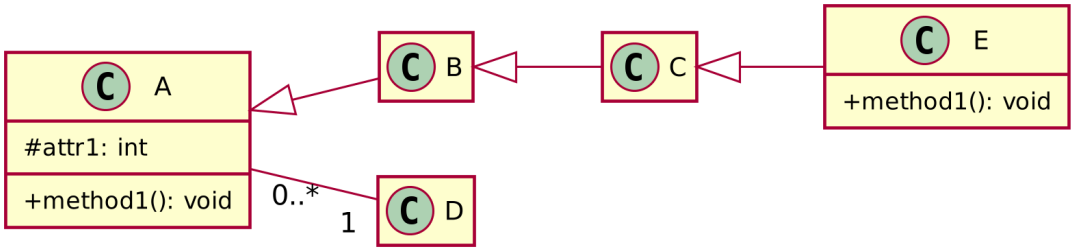
L'héritage est un principe de la programmation orientée objet où une classe (enfant) étend une autre classe (parent), en héritant de ses membres. On peut ainsi créer des arbres de liaisons entre classes. La classification entre des classes pour savoir qui hérite de qui n'est pas toujours évidente, il faut donc être explicite sur les choix qu'on fait.

L'association d'héritage se réalise entre classes et ne porte pas de nom ni n'a de multiplicité, elle est notée par une flèche, de l'enfant vers le parent. Une proposition équivalente à l'héritage est de dire qu'on "est" la classe dont on hérite. Par exemple, une classe `Pizza` qui hérite de `RepasItalien` permet de dire qu'on est une pizza et aussi un repas italien. Une classe `Spaghetti` héritant elle aussi de `RepasItalien` est un plat italien, mais pas une pizza. On peut le voir sous la forme suivante.



Comme les enfants héritent des comportements des parents, on ne doit pas réécrire les membres du parent, chez l'enfant. Cependant, si on décide de redéfinir un membre dans une sous-classe, alors on doit réécrire ce membre. Ceci permet de montrer qu'une même méthode se comporte différemment suivant la classe à laquelle elle appartient, on parlera de **polymorphisme**. Attention, une classe ne peut **pas** hériter d'elle-même et l'héritage ne peut pas créer de cycle.

Dans l'exemple suivant, les classes enfants ont les mêmes membres que la classe parent, la classe `E` par contre redéfini l'unique méthode de la super-classe `A`. En pratique, ce diagramme signifie que des instances des sous-classes de `A` peuvent avoir un lien avec des instances de `D`. En effet, `B`, `C` et `E` sont tous des enfants de `A` et donc en hérite la capacité à créer une association avec `D`.

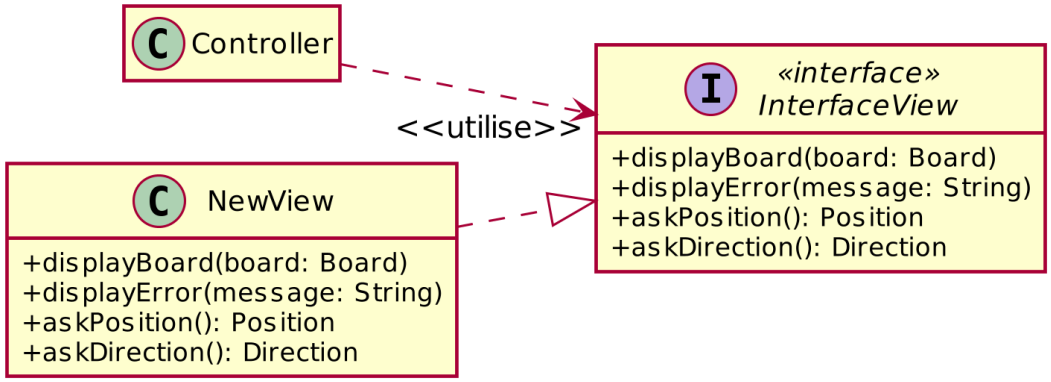


Aussi, il est important de remarquer que l'héritage n'apparaît que dans les diagrammes de classes. Les objets n'ont pas de liens explicites pour démontrer l'héritage, un lien est bien une instance d'une association, pas d'un héritage. Donc, au moment de la représentation, aucun lien n'apparaît pour désigner l'héritage des objets.

9 Les interfaces

Enfin, le dernier chapitre traite des **interfaces**. Une interface est une collection de méthodes qui décrit le service d'une classe ou d'un composant.

Un exemple d'interface, avec son implémentation, peut être le suivant (à noter qu'on implémente des méthodes et qu'il faut donc les noter à nouveau dans l'UML). On utilise la flèche pointillée pleine pour montrer une implémentation de l'interface et une flèche pointillée simple pour l'usage.



10 Les paquets

Un paquet est un élément d'organisation qui permet de regrouper des éléments, en général on regroupe des classes, interfaces, objets, etc ; liés dans un sens logique, logiciel ou business. Un paquet donne aussi un espace de nom et touche donc à la **visibilité**. En effet, en Java par exemple, la visibilité par défaut est `package`, ce qui signifie qu'un élément n'est pas visible en dehors du paquet.

Dans un diagramme, un élément ne peut appartenir qu'un à seul paquet, mais un paquet peut en englober d'autres. Le paquet le plus englobant, c'est-à-dire au plus haut de la hiérarchie, est la racine de l'arbre des paquets.

Voici un exemple qui force un peu le trait pour faire comprendre les différents cas de figure :