Ingénierie avancée du logiciel

Introduction générale:

UML (Unified Modeling Language) : est un langage de modélisation graphique destiné au développement logiciel orienté objet.

L'UML est issu de la fusion des premiers langages orientés objet (Booch, OMT, OOSE).

La première version UML 1.0 a été normalisée par l'OMG[1] (Object Management Group).

La dernière version d'UML est UML 2.5.1 (revue en 2017). Elle comporte 14 types de diagrammes : ***7 structurels (statiques)*** et ***7 comportementaux (dynamiques)***. Ces diagrammes sont utilisés avec des méthodes de GDP (gestion de projets) pour spécifier et concevoir des systèmes logiciels.

Les diagrammes statiques :

- Diagramme de classes
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement
- Diagramme de paquetage
- Diagramme de structure composite (représentation sous forme de boites blanches des relations entre composants)
- Diagramme de profils (utilisation d'un méta-modèle de référence UML pour le spécialiser pour un domaine particulier)

Les diagrammes dynamiques :

- Diagramme de cas d'utilisation
- Diagramme de séquences
- Diagramme d'activités
- Diagramme d'états-transitions
- Diagramme de communication (représentation simplifiée d'un diagramme de séquence qui montre uniquement les échanges de messages entre objets)
- Diagramme global d'interaction (représentation des enchainements possibles entre les scénarios sous forme de diagramme de séquence). Il s'agit d'une variante du diagramme d'activités
- Diagramme de temps (représentation des variations d'une donnée au cours du temps)

[1] OMG : Organisme américain qui standardise tous ce qui se rapporte à l'objet

Remarque :

L'UML n'est pas une méthode ou un processus, il s'agit d'un langage pseudo-formel de modélisation orientée objet. Il est utilisé dans le développement logiciel avec une méthode ou méthodologie de gestion de projets informatiques.

Plan du semestre :

Partie 1 : UML

I. Diagramme de cas d'utilisation

II. Diagramme de classes et Design patterns

III. Diagramme de séquences

IV. Diagramme d'activités

V. Diagramme d'état-transition

Partie 2 : Gestion de projets informatiques

I. Diagramme de cas d'utilisation (UCD : Use Case Diagram)

Un UCD expose les fonctionnalités d'un système (les actions qu'on peut réaliser).

Il identifie également les possibilités d'interactions entre le système et les acteurs qui vont utiliser le système.

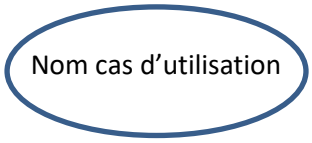
Un UCD est composé de :

- **Cas d'utilisation**
- **Acteurs**
- **Relations**

1. Cas d'utilisation :

Il s'agit d'une fonctionnalité simple d'un système modélisé. Exemple : Ajouter fiche client, rechercher un client, supprimer,…

Notation :



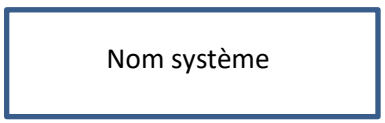
La fonctionnalité doit être décrite d'un point de vue utilisateur.

2. Acteurs :

Un acteur est un utilisateur du système. Il existe 2 types d'acteurs :

- Acteur principal : il interagit directement avec le système et réalise le cas d'utilisation. Il s'agit souvent d'un acteur humain (peut être dans de rare cas un système).
  Notation :

- Acteur secondaire : ne fait que recevoir l'information à l'issue de la réalisation du cas d'utilisation. Il s'agit souvent d'un système.
  Notation :

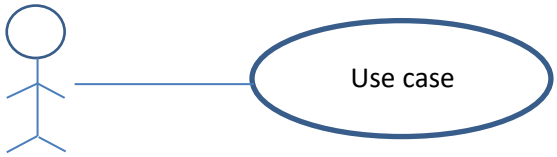


3. Relations :

3.1. Relation d'association :

3.2. Il s'agit de la relation entre un acteur et un cas d'utilisation (use case).

Notation :



3.3. Relation entre acteurs :

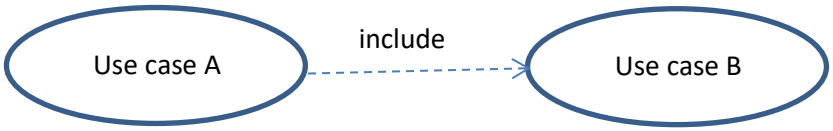
3.4. Un acteur peut hériter des fonctionnalités d'un autre acteur.

Notation :

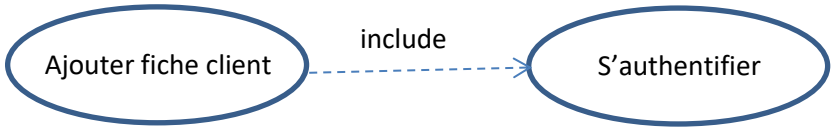
3.5. Relations entre cas d'utilisation :

- **Include** : si un use case A inclut le use case B, cela implique que la réalisation de A nécessite d'abord la réalisation de B (obligatoire)

  Notation :

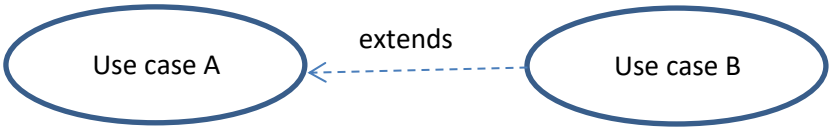


  Exemple :

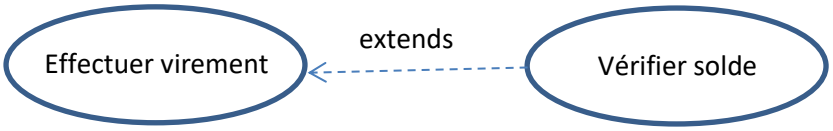


- **Extend** : Si le use cas B étend le use case A cela implique que B peut être appelé pendant la réalisation de A (l'extension est une réalisation optionnelle)

  Notation :



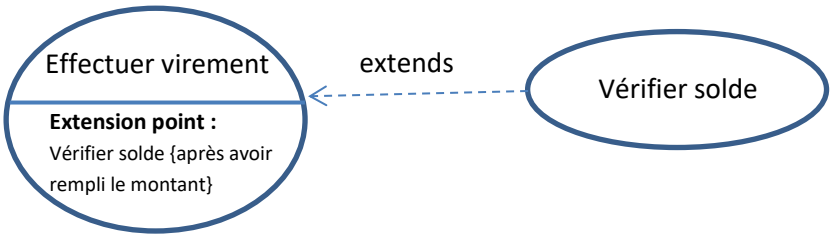
  Exemple :



Remarques :

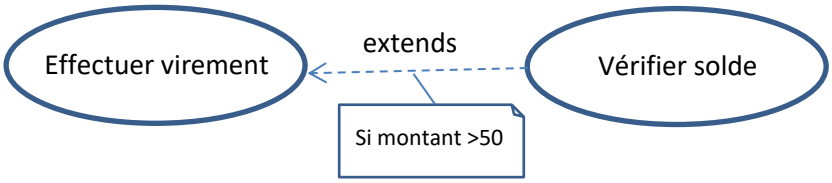
- Il est possible de préciser à quel moment l'extension est appelée, en utilisant un « extension point ».

Exemple :



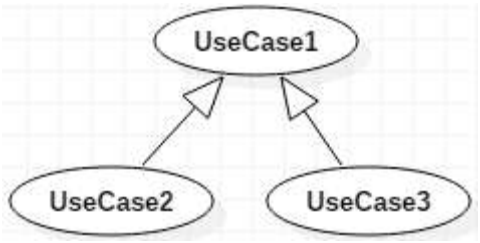
- On peut aussi ajouter une condition à l'extension sous forme de note.

Exemple :

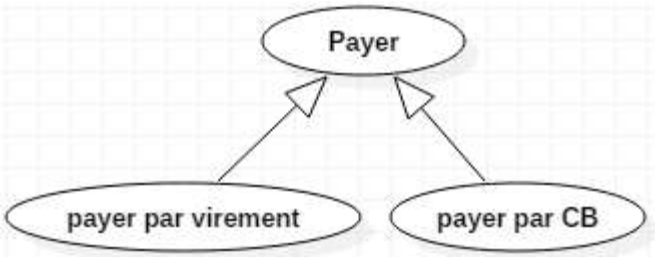


- **Relation de généralisation/spécialisation** : il s'agit d'une forme d'héritage sémantique entre cas d'utilisation

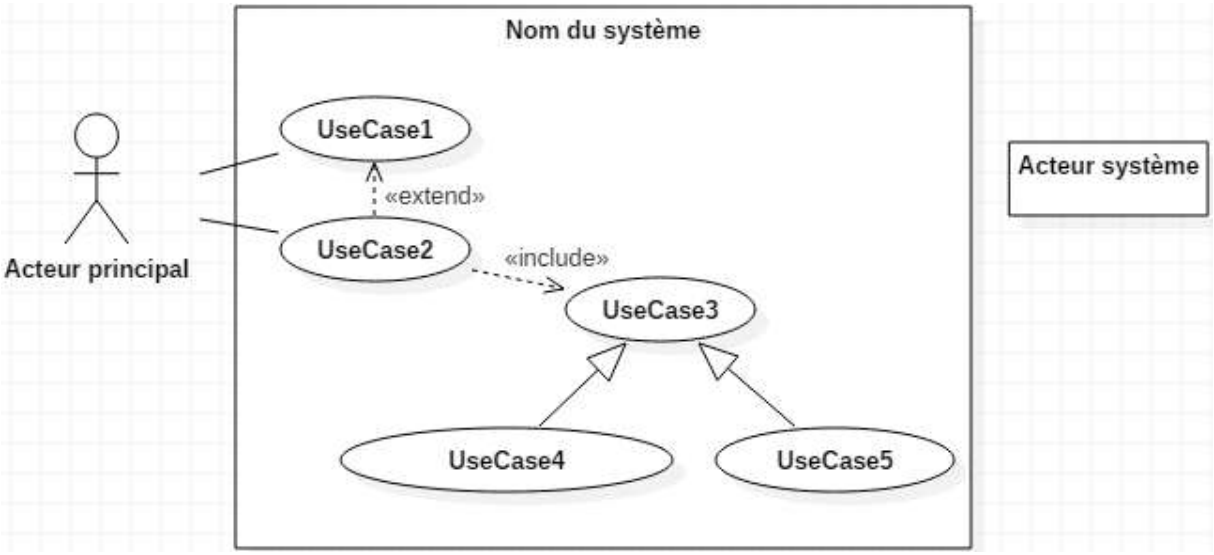
Notation :



Exemple :



4. Vue générale d'un diagramme de cas d'utilisation :



Exercice d'application : voir exercice 1 du TD1

5. Description des cas d'utilisation :

Un cas d'utilisation peut être décrit à l'aide d'un scénario. Chaque cas contient des séquences d'actions pour sa réalisation, on décrit ces actions à l'aide d'un scénario.

Il existe 3 types de scénarios :

- **Scénario nominal :** il s'agit du scénario de déroulement normal des actions (obligatoire)
- **Scénario alternatif :** il décrit les éventuelles étapes différentes liées au choix de l'utilisateur (optionnel)
- **Scénario d'exception :** il décrit les cas d'erreurs et autres exceptions qui peuvent se produire et arrêter le déroulement normal d'un scénario (optionnel)

Fiche descriptive d'un scénario :

Cas d'utilisation	Nom du cas d'utilisation
Acteur(s)	Les acteurs qui peuvent réaliser le cas
Description	Description succincte du cas tel que son objectif
Pré-condition(s)	Les conditions obligatoires au bon déroulement du cas ou qui sont à l'origine de son exécution
Scénario nominal	1. 2. 3. … (On décrit ici l'interaction entre le système et le user)
Scénario(s) alternatif(s)	2.1.a. b. 2.2. 3.1.a.
Scénario(s) d'exception(s)	3. 1…. a. 3.2. ..
Post-condition(s)	Il s'agit d'un ou des résultats tangibles obtenus à la fin de l'exécution du use case (*exemples* : fiche client enregistrée dans la BD, email envoyé,…)

Exemple : description du cas d'utilisation « publier une annonce » de l'exercice 1 du TD1.

Cas d'utilisation	Publier une annonce
Acteur(s)	Admin ENSA
Description	L'administrateur publie une annonce en ajoutant une description textuelle. Il a aussi la possibilité d'ajouter une image.
Pré-condition(s)	L'administrateur doit être authentifié
Scénario nominal	1. L'utilisateur clique sur la fonctionnalité « publier une annonce » 2. Le système affiche la page demandée 3. L'utilisateur saisit le titre et une description textuelle de l'annonce puis valide l'action 4. Le système lui notifie que l'annonce a été publiée
Scénario(s) alternatif(s)	3.1.a. L'utilisateur saisit le titre, la description textuelle de l'annonce puis clique sur ajouter image et ajoute l'image souhaitée. Il valide ensuite son action. 3.2 . L'utilisateur annule la saisie
Scénario(s) d'exception(s)	3. l'utilisateur ne saisit pas les données obligatoires (titre et description) et valide l'action 3.1. le système affiche un message d'erreur « vous devez saisir le titre et la description »
Post-condition(s)	Annonce enregistrée dans la BD et publiée

Remarque :

Le cas d'utilisation « ajouter image » est un cas simple, c'est pour cela qu'il a été décrit directement dans le scénario alternatif. S'il s'agissait d'un scénario complexe (avec des alternatives et exceptions), on aurait mis :

3.1.a l'utilisateur fait appel au cas d'utilisation interne « ajouter image »

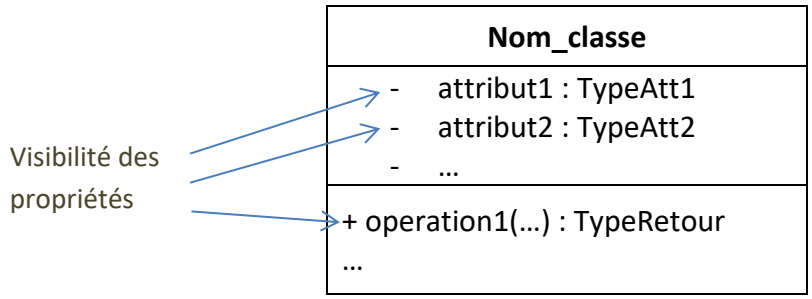
On créera ensuite une autre fiche descriptive pour le cas d'utilisation *ajouter image*.

II. Diagramme de classes :

Un diagramme de classes permet de modéliser les classes d'un système et les relations entre elles. Il s'agit précisément de concevoir les classes d'un domaine d'application métier.

Un diagramme de classes est indépendant d'un langage de POO.

1. Représentation d'une classe en UML :



Types de visibilité :

\+ : public (visible partout)

\- : privé (visible dans la classe uniquement)

\# : protected (visible dans la classe et ses sous classes)

~ : package (visible dans le package)

2. Relations entre classes :

2.1. L'association :

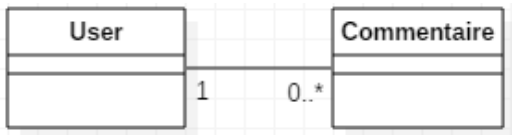
Notation :

Il s'agit d'association binaire (entre 2 classes).

Les multiplicités :

- 1 ou 1..1 : exactement 1
- * ou 0..* : plusieurs
- 1..* : au moins 1 (1 à plusieurs)
- On peut préciser un nombre min et max. *Exemple :* 1..3, 0..5, …

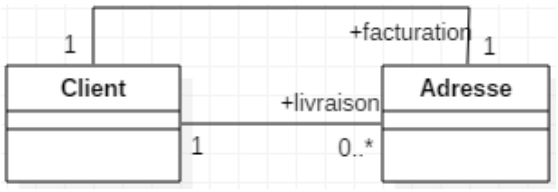
Exemple :



Les rôles :

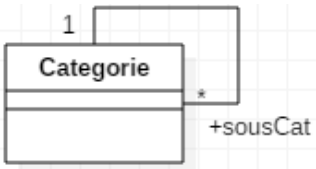
On peut également préciser les rôles dans une association, surtout quand plusieurs associations existent entre deux mêmes classes.

Exemple :



- Cas de la réflexivité : (auto-association)

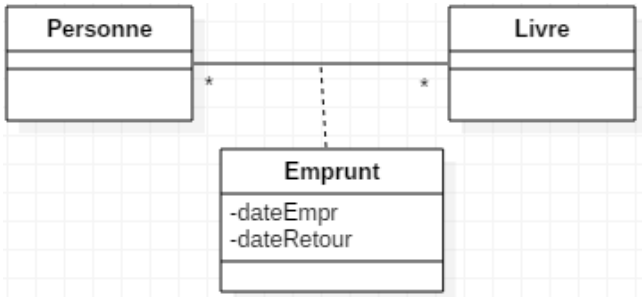
Exemple :



- Cas de la classe-association :

Quand la relation entre deux classes donne lieu à d'autres propriétés, on utilise dans ce cas une « classe-association ».

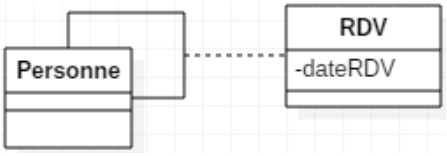
Exemple :



Remarque :

Il est possible d'avoir une classe-association avec une auto-association.

Exemple :



- Navigabilité :

La navigabilité entre 2 classes est par défaut dans les 2 sens dans une association. Il est possible que la navigation soit unidirectionnelle.

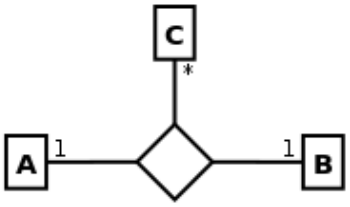
Exemple :

Dans l'exemple, la classe article est navigable (à partir de commande on peut accéder à Article) mais pas réciproquement.

- Association n-aire :

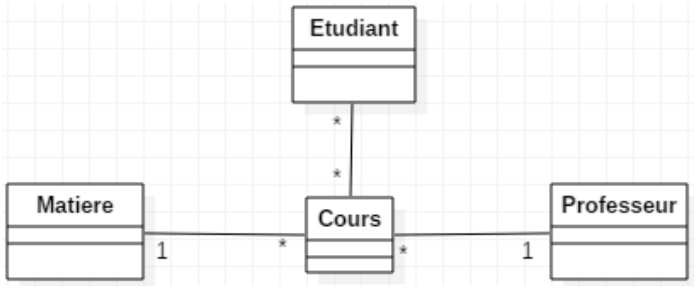
Il est possible d'avoir une association n-aire (entre plus de 2 classes). L'exemple le plus connu est l'association ternaire.

Notation :



Ce type d'association est souvent imprécis et ambigüe pour les choix d'implémentation. Il est préférable de ne pas l'utiliser et de le remplacer tout simplement par l'association binaire comme dans l'exemple suivant:

Exemple :



2.2. La composition

Il s'agit d'un cas spécial de l'association. La composition permet de modéliser « tout ou partie de » (quand un élément est composé d'éléments).

Notation :

Légende :

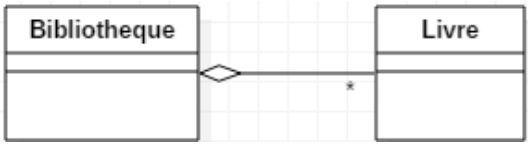
- Classe 1 : composite
- Classe2 : composant
- La destruction du composite implique la destruction du composant (relation forte)

Exemple :

2.3. L'agrégation :

Même principe que la composition, sauf que la relation entre composite et composant n'est pas forte : les composants peuvent exister en dehors du composite.

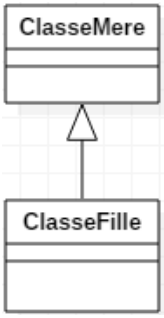
Exemple :



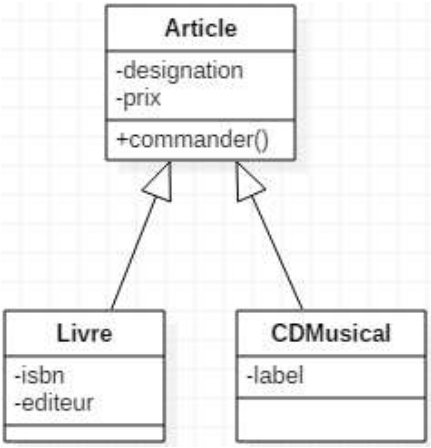
2.4. L'héritage :

Permet de factoriser des concepts métier en mettant en relation des classes ayant des caractéristiques communes (la factorisation doit être logique).

Notation :



Exemple :

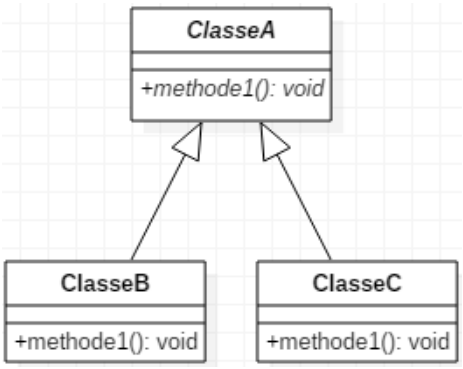


Il est possible de faire un choix de modélisation sans héritage mais le choix de l'héritage est plus judicieux pour des questions d'évolutivité du système.

2.5. Classe abstraite

Contient au moins une méthode abstraite. L'implémentation des méthodes se fait dans les classes filles.

Notation :



2.6. Interface

Quand toutes les méthodes sont abstraites, on parle d'interface. Une interface s'implémente par une classe concrète ou une classe abstraite.

Notation :

N.B : Il peut y avoir de l'héritage entre interfaces

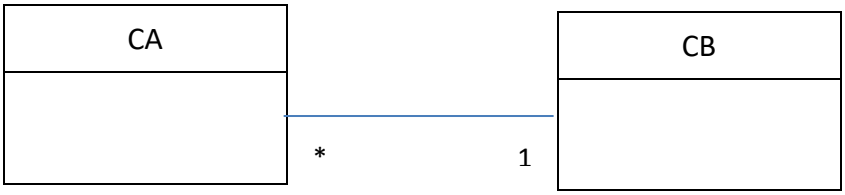
3) Implémentation : *Exemples en Java*

3.1. Association bidirectionnelle (1 à 1)


```
Public Class CA {
   private String att1 ;
   private CB attcb ;
…
}
```

```
Public Class CB {
   private String att2 ;
   private CA attca;
…
}
```

3.2. Association bidirectionnelle 1 à *

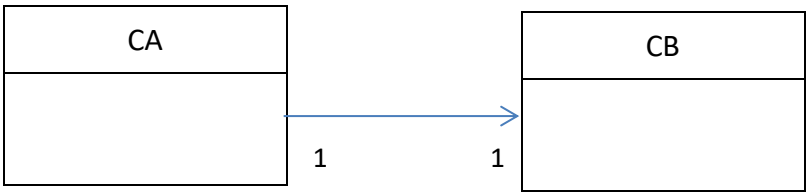


La multiplicité * prend la forme d'un tableau ou d'une collection :

```
Public Class CA {

        private CB attcb ;

…

}
```

```
Public Class CB {

    private ArrayList<CA> attca;

…

}
```

3.3. Association unidirectionnelle :



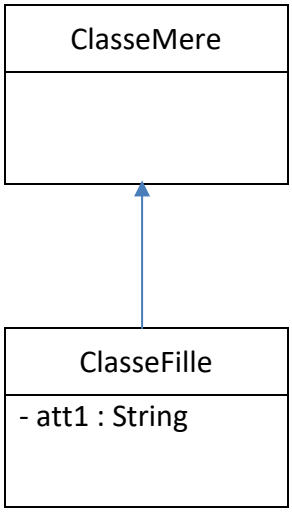

```
Public Class CA {

        private CB attcb ;

…

}
```

```
Public Class CB {

// il n'y a pas d'attributs de type
CA ici car CB ne voit pas CA

…

`
```

Remarque :

La classe association, l'agrégation et la composition sont implémentés comme les associations binaires.

3.4. L'héritage :




```
Public Class ClasseMere {
…
}
```

```
Public Class ClasseFille extends ClasseMere {
…
}
```

3.5. Cas de l'interface :


```
Public Interface Inter {
…
}
```

```
Public Class CA implements Inter {
…
//implementation des methodes de
l'interface
}
```