Université Hassan II de Casablanca
Ecole supérieure de Technologie
Département Génie Informatique

Support de Cours

2éme Année

Ingénierie des Systèmes d'Information

-

Unified Modeling Language (UML)

Elaboré par : Pr. Nadia AFIFI

AVANT-PROPOS

L'information se présente sous trois formes: les données, les connaissances et les messages. On a l'habitude de désigner par «système d'information» l'ensemble des moyens techniques et humains qui permet de stocker, de traiter ou de transmettre l'information. On dira donc qu'un Système d'Information est «tout moyen dont le fonctionnement fait appel d'une façon ou d'une autre à l'électricité et qui est destiné à élaborer, traiter, stocker, acheminer, présenter ou détruire l'information» (AGI n°900/SGDN). Ainsi, l'objectif du Système d'Information(SI) est de rendre possible la mise en commun d'informations appartenant à des sources séparées.

L'élaboration des SI fait appel à des méthodes et des langages de modélisation afin de les concevoir et de pouvoir les réaliser par la suite. La conception classique qui se base sur la décomposition fonctionnelle des systèmes (devenus de plus en plus complexes) n'est plus capable de suivre les évolutions incessantes des technologies et des besoins applicatifs. Donc l'approche objet est devenue incontournable dans le cadre du développement des systèmes logiciels complexes. L'approche objet requiert de modéliser avant de concevoir.

Dans ce support on présente de façon simplifiée les différents concepts du langage de modélisation UML. Le Langage propose uniquement une notation dont l'interprétation est définie par un standard, mais pas une méthodologie complète ; Alors qu'une méthode de développement définit à la fois un langage de modélisation et la marche à suivre lors de la conception. C'est un langage de modélisation fondé sur des événements et ou messages. Ce n'est pas un langage formel ni un langage de programmation.

L'objectif de ce cours est de fournir une référence des concepts de base d'UML 2.x. Il est composé de sept chapitres qui peuvent être lu séparément, Chaque chapitre est divisé en deux ou trois sections qui présentent les différents diagrammes d'UML.

Le support s'adresse aux étudiants de premier cycle (DUT, BTS). Il pourrait également servir de référence aux étudiants de second cycle (universités et écoles d'ingénieurs) désireux d'apprendre les concepts de base du langage UML.

Table des matières

CHAPITRE I

INTRODUCTION

-

Les méthodes objet et la genèse d'UML

"L'apprentissage est la seule chose que l'esprit n'épuise jamais, ne craint jamais et ne regrette jamais "

Léonard de Vinci

I.1 Historique des Méthodes

- **Les premières méthodes d'analyse (années 70)**

  Découpe cartésienne (fonctionnelle et hiérarchique) d'un système.

- **L'approche systémique (années 80)**

  Modélisation des données + modélisation des traitements (Merise, Axial, IE, etc.).

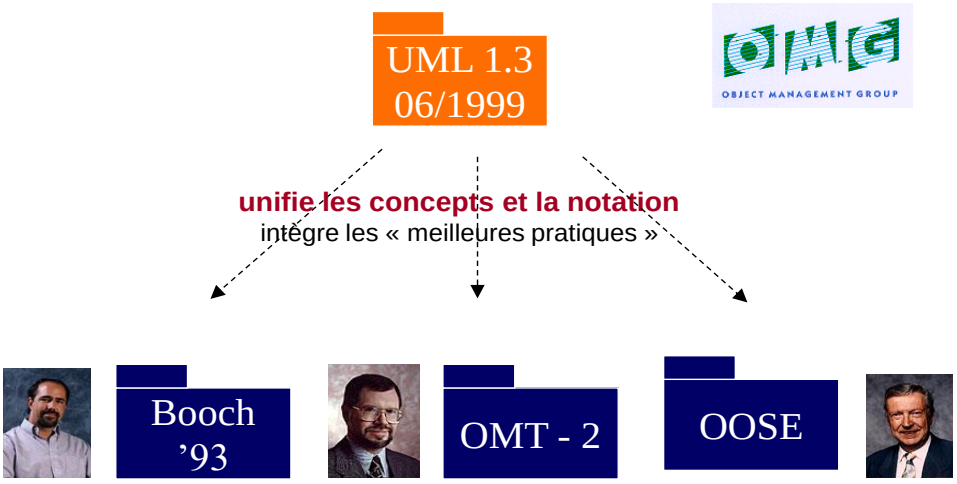
- **L'émergence des méthodes objet (1990-1995)**

  - ✓ Prise de conscience de l'importance d'une méthode spécifiquement objet : comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux)?
  - ✓ Plus de 50 méthodes objet sont apparues durant cette période (Booch, Classe-Relation, Fusion, OOD, OMT, OOA, OOM, OOSE, etc.) !
  - ✓ Aucune méthode ne s'est réellement imposée.

- **Les premiers consensus (1995)**

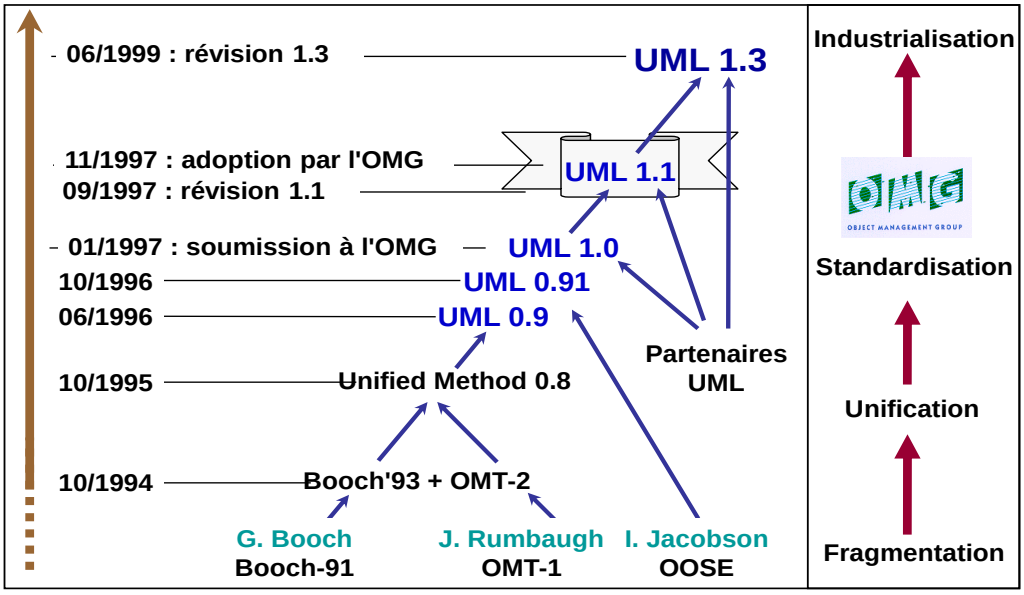
  - ✓ **OMT** (James Rumbaugh) : vues statiques, dynamiques et fonctionnelles d'un système.
    - Issue du centre de R&D de General Electric.
    - Notation graphique riche et lisible.
  - ✓ **OOD** (Grady Booch) : vues logiques et physiques du système
    - Définie pour le DOD, afin de rationaliser le développement d'applications ADA, puis C++.
    - Ne couvre pas la phase d'analyse dans ses 1ères versions (préconise SADT).
    - Introduit le concept de package (élément d'organisation des modèles).
  - ✓ **OOSE** (IV.Ar Jacobson) : couvre tout le cycle de développement
    - Issue d'un centre de développement d'Ericsson, en Suède.
    - La méthodologie repose sur l'analyse des besoins des utilisateurs.

UML est un standard OMG



- **UML est un langage public et standardisé**
  - ✓ L'Object Management Group (OMG) a retenu UML comme le langage de modélisation objet standard, pour la spécification et la conception
  - ✓ L 'OMG est désormais responsable de ses évolutions
  - ✓ www.omg.org

I.1.1 Historique d'UML



2.5.1	December 2017	https://www.omg.org/spec/UML/2.5.1
2.4.1	July 2011	https://www.omg.org/spec/UML/2.4.1
2.3	May 2010	https://www.omg.org/spec/UML/2.3
2.2	January 2009	https://www.omg.org/spec/UML/2.2
2.1.2	October 2007	https://www.omg.org/spec/UML/2.1.2
2.0	July 2005	https://www.omg.org/spec/UML/2.0
1.5	March 2003	https://www.omg.org/spec/UML/1.5
1.4	September 2001	https://www.omg.org/spec/UML/1.4
1.3	February 2000	https://www.omg.org/spec/UML/1.3
1.2	July 1999	https://www.omg.org/spec/UML/1.2
1.1	December 1997	https://www.omg.org/spec/UML/1.1

I.2 Introduction au Langage de Modélisation UML

Ce qu'est et n'est pas UML

- UML = Langage de modélisation
  graphique & textuel

- UML ≠ Méthode !

I.2.1 Définition d'UML

- UML est un langage de modélisation pour :
  - ✓ comprendre et décrire des besoins
  - ✓ spécifier des systèmes, simples ou complexes
  - ✓ concevoir et construire des solutions
  - ✓ documenter (textes et graphiques) un système
  - ✓ communiquer entre les membres de l'équipe de projet
- UML est un langage à usage général, quel que soit :
  - ✓ le type de système - logiciel, matériel, organisation, etc.
  - ✓ le domaine métier - gestion, ingénierie, finance, etc.
  - ✓ le processus de développement - cascade, itératif, RAD, etc.

I.3 Modélisation avec UML

I.3.1 Pourquoi modéliser

- Un modèle est une simplification de la réalité qui permet de mieux comprendre le système à développer.
- Il permet :

- ✓ De visualiser le système comme il est ou comme il devrait l'être.
- ✓ De valider le modèle vis à vis des clients
- ✓ De spécifier les structures de données et le comportement du système.
- ✓ De fournir un guide pour la construction du système.
- ✓ De documenter le système et les décisions prises.

I.3.2 Les principes de la modélisation

1) Le modèle doit être connecté au monde réel.

2) Un modèle peut être exprimé avec différents niveaux de précision.

3) Un simple modèle n'est pas suffisant, il y a plusieurs façons de voir un système.

- ✓ plan de masse
- ✓ vue de face, de coté, …
- ✓ plan des niveaux
- ✓ plan électrique
- ✓ plan de plomberie
- ✓ plan des calculs de construction

I.3.3 Qu'apporte la modélisation objet

- ➢ Plus grande indépendance du modèle par rapport aux fonctionnalités demandées.
- ➢ Des fonctionnalités peuvent être rajoutées ou modifiées, le modèle objet ne change pas.
- ➢ Plus proche du monde réel.

I.3.4 Les objectifs d'UML

- ➢ Représenter des systèmes entiers
- ➢ Prendre en compte les facteurs d'échelle
- ➢ Créer un langage de modélisation utilisable à la fois par les humains et les machines
- ➢ Recherche d'un langage commun :

- Utilisable par toutes les méthodes
- Adapté à toutes les phases du développement
- Compatible avec toutes les techniques de réalisation

➢ Un processus de développement logiciel universel est une utopie :

- Impossible de prendre en compte toutes les organisations et cultures d'entreprises.
- Un processus est adapté (donc très lié) au domaine d'activité de l'entreprise.
- Même si un processus constitue un cadre général, il faut l'adapter de manière précise au contexte de l'entreprise.

UML un langage

➢ UML n'est pas une méthode

➢ UML est un langage de modélisation objet

➢ UML a été adopté par toutes les méthodes objet

➢ UML est dans le domaine public, c'est une norme

UML un langage pour :

➢ **visualiser**

chaque symbole graphique a une sémantique

➢ **spécifier**

de manière précise et complète, sans ambiguïté,

➢ **construire**

les classes, les relations SQL peuvent être générées automatiquement

➢ **documenter**

les différents diagrammes, notes, contraintes, exigences seront présentés dans un document.

I.3.5 UML et les domaines d'utilisation

➢ Systèmes d'information des entreprises

➢ Les Banques et les services financiers

- Télécommunications
- Transport
- Défense et aérospatiale
- Scientifique
- Applications distribuées par le WEB

I.4 Les Composants d'UML

UML se décompose en plusieurs sous ensembles :

- Les vues : Les vues sont les observables du système. Elles décrivent le système d'un point de vue donné, qui peut être organisationnel, dynamique, temporel, architectural, géographique, logique, etc. En combinant ces vues, il est possible de définir (ou retrouver) le système complet.
- Les diagrammes : ce sont des éléments graphiques. Ceux-ci décrivent le contenu des vues, qui sont des notions abstraites. Les diagrammes peuvent faire partie de plusieurs vues
- Les modèles d'élément : ce sont les briques des diagrammes UML, ces modèles sont utilisés dans plusieurs types de diagrammes. Ex: cas d'utilisation, classe, association, etc.

I.4.1 Les trois éléments de base en UML

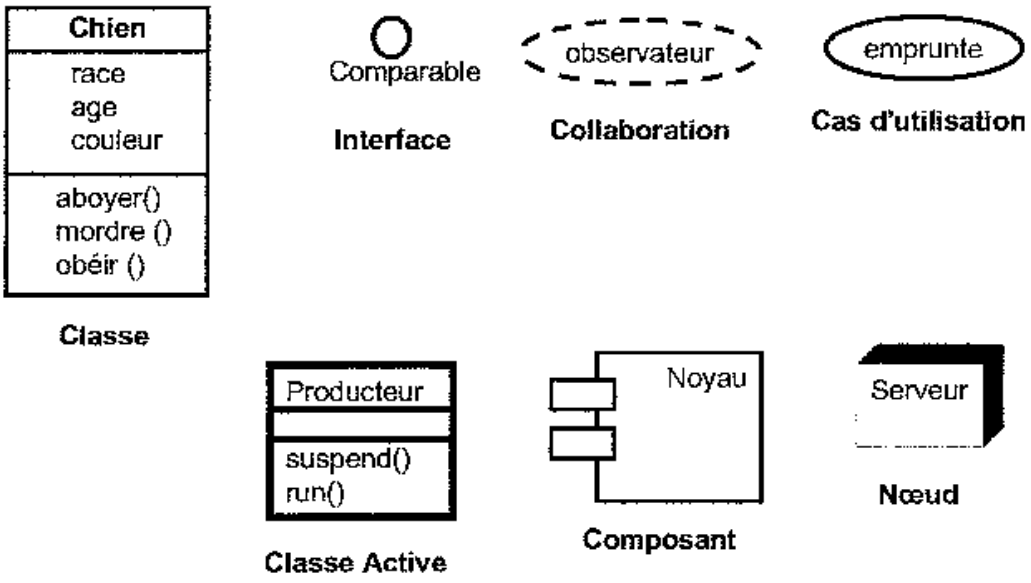
1) Les blocs de base pour construire
   - Les entités utilisées : Entités structurelles
     - Entités de comportement
     - Entités de regroupement
     - Entité d'annotation
   - La notion de relation
   - Les diagrammes

2) Les règles à observer pour utiliser ces blocs de base
   - Règles sémantiques
   - Règles de présentation

3) Les mécanismes communs
   - Spécification

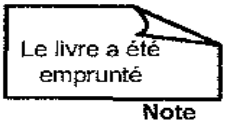
- Présentation
- Extension des modèles

Les entités structurelles



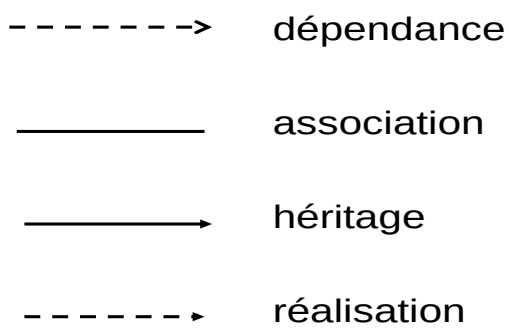
Les entités de comportement

Les entités de groupement



Les entités de notation

Les relations

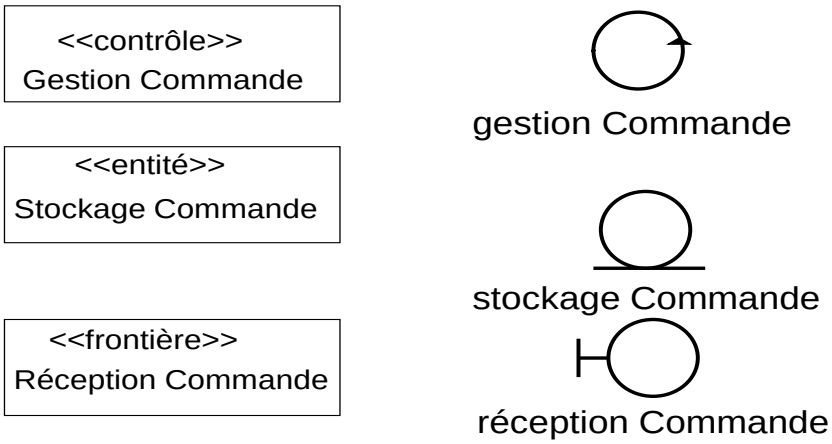


En langage UML, une relation est une connexion entre des éléments de modèle. Une relation UML est un type d'élément de modèle qui ajoute une sémantique à un modèle en définissant la structure et le comportement entre les éléments de modèle.

Les relations UML sont regroupées dans les catégories suivantes :

Catégorie	**Fonction & Définition**
Bords d'activité	Représentent le flux entre des activités
Associations	Indiquent que des instances d'un élément de modèle sont connectées aux instances d'un autre élément de modèle. Relation entre 2 classificateurs ou plusieurs qui implique des connections entre leurs instances.
Dépendances	Indiquent qu'un changement apporté à un élément de modèle peut affecter un autre élément de modèle. Relations entre 2 éléments; dans laquelle un changement sur un élément (indépendant) peut affecter l'autre (dépendant)
Généralisations	Indiquent qu'un élément de modèle est une spécialisation d'un autre élément de modèle
Réalisations	Indiquent qu'un élément de modèle fournit une spécification qu'un autre élément de modèle implémente Relation entre spécification et implémentation.
Transitions	Représentent des changements dans un état

Stéréotypes et icônes associées



I.4.2 Les Modèles d'UML

- Le modèle des classes qui capture la structure statique
- Le modèle des états qui exprime le comportement dynamique des objets
- Le modèle des cas d'utilisation qui décrit les besoins des utilisateurs
- Le modèle d'interaction qui représente les scénarios et les flots de messages
- Le modèle de réalisation qui montre les unités de travail
- Le modèle de déploiement qui précise la répartition des processus

I.4.3 Les Diagrammes D'UML

- UML 2.5 définit 14 sortes de diagrammes (9 en UML 1.3) pour représenter les différents points de vue de modélisation.
- Les diagrammes peuvent montrer tout ou partie des caractéristiques des éléments de modélisation, selon le niveau de détail utile dans le contexte d'un diagramme donné.
- **Diagrammes structurelles ou statiques**
  - ✓ Les diagrammes de classes
  - ✓ Les diagrammes d'objets
  - ✓ Les diagrammes de composants
  - ✓ Les diagrammes de déploiement
  - ✓ Les diagrammes de paquetages

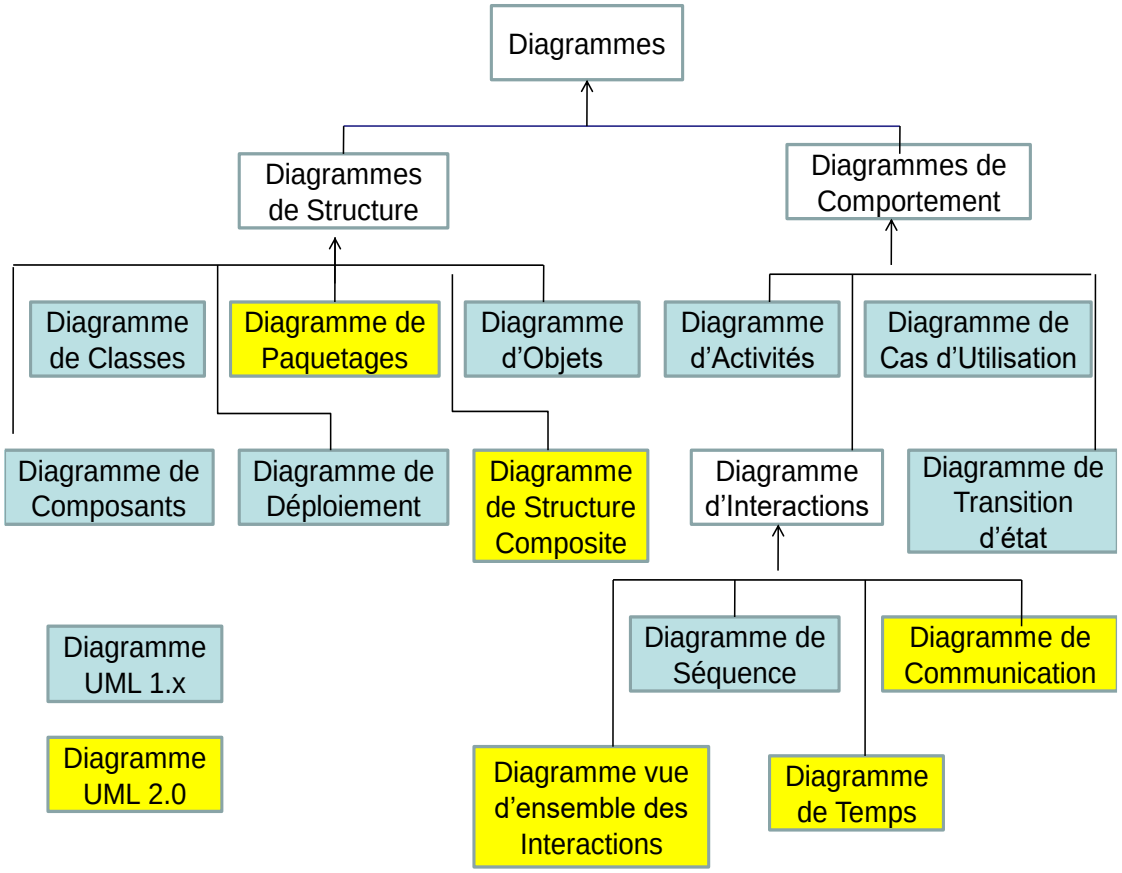
- ✓ Les diagrammes de structure composite (collaboration)
- ✓ Les diagrammes de profils

➢ **Diagrammes comportementaux**

- ✓ Les diagrammes d'états-transition
- ✓ Les diagrammes d'activités
- ✓ Les diagrammes de cas d'utilisation

➢ **Diagrammes d'interactions ou dynamiques**

- ✓ Les diagrammes de séquence
- ✓ Les diagrammes de communication
- ✓ Les diagrammes global d'interaction
- ✓ Les diagrammes de temps



Les 14 diagrammes d'UML2

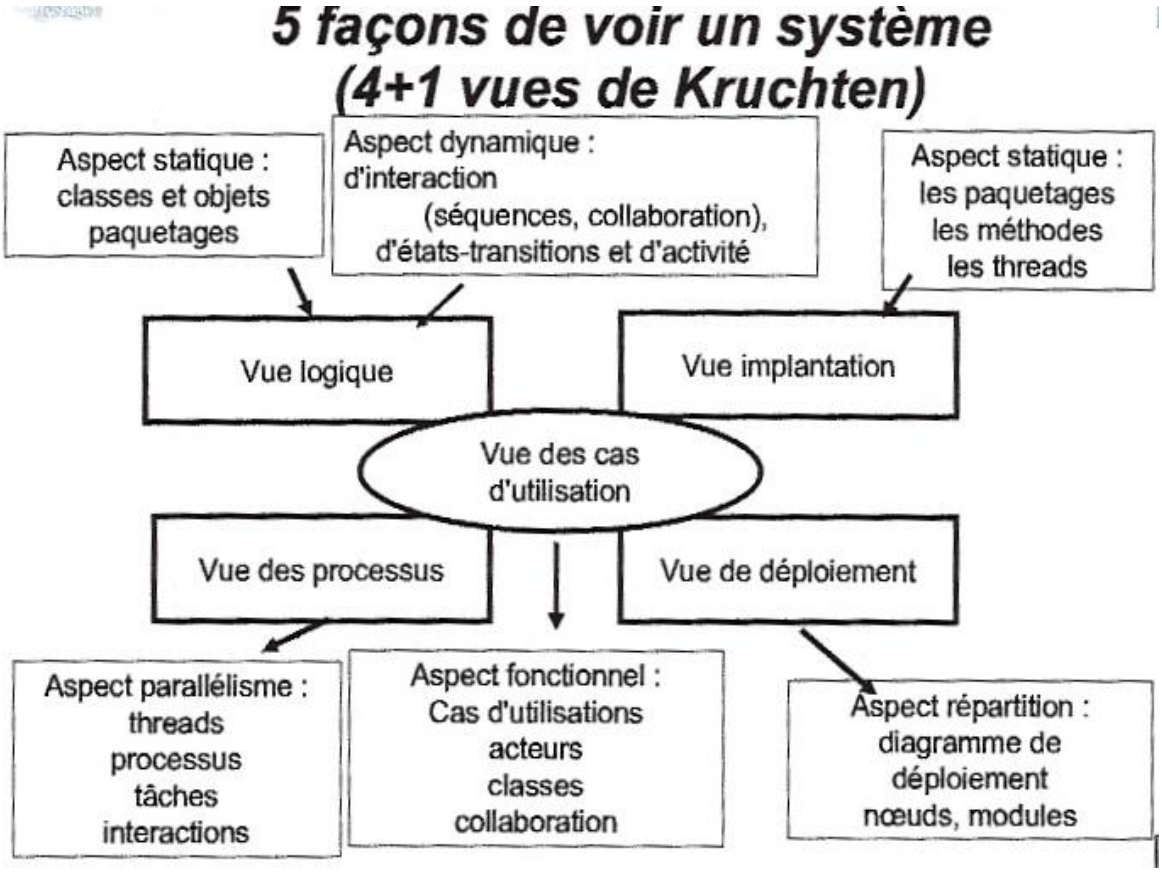
1. Les diagrammes de classes *(Class Diagram) :* représentent la structure statique en terme de classes et de relations.
2. Les diagrammes d'objets *(Object Diagram) :* représentent les objets et

leurs relations et correspondent à des diagrammes de collaboration simplifiés, sans représentation des envois de messages.

3. Les diagrammes de composants *(Component Diagram) :* représentent les composants physiques d'une application.
4. Les diagrammes de déploiement *(Deployment Diagram) :* montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.
5. Les diagrammes de structures composites *(Composite Structure Diagram) :* représentent la structure interne d'un objet complexe lors de son exécution, dont ses points d'interaction avec le reste du système.
6. Les diagrammes de cas d'utilisation *(Use-cases ou Use Case Diagram) :* représentent les fonctions du système du point de vue de l'utilisateur.
7. Les diagrammes d'activité *(Activity Diagram) :* représentent le comportement d'une opération, d'un cas d'utilisation ou un processus métier.
8. Les diagrammes de séquence *(Sequence Diagram)* : sont une représentation temporelle des objets et de leurs interactions.
9. Les diagrammes d'états-transition *(State Machine Diagram) :* représentent le comportement d'un classificateur ou d'une opération en terme d'états.
10. Les diagrammes de communication *(Communication Diagram) :* sont une représentation simplifiée des diagrammes de séquence se concentrant sur les échanges de messages entre les objets; équIV.Alents au diagrammes de collaboration (UML 1) qui sont une représentation spatiale des objets, des liens et des interactions.
11. Les diagrammes de temps *(Timing Diagram) :* permettent de décrire les variations d'une donnée au cours du temps.
12. Les diagrammes global interaction *(Interaction Overview Diagram) :* permettent de décrire les enchainements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité)
13. Les diagrammes des paquetages *(Package Diagram) :* servent à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions.

14. Les diagrammes de profiles *(Profil Diagram):* permettent de spécialiser, de personnaliser pour un domaine particulier un méta-modéle de référence d'UML

I.4.4 Les Vues d'UML



I.4.1 Point de vue des cas d'utilisation

- Description du modèle vue par ses utilisateurs finaux : correspond aux besoins attendus par chaque acteur –QUOI et QUI ;
- Regroupe le comportement du système selon
  - ✓ Priorité: critique, important, accessoire ;
  - ✓ Risques à circonscrire ;
  - ✓ Options disponibles ;
  - ✓ Couverture de l'architecture ;
  - ✓ Autres objectifs tactiques et contraintes.

I.4.2 Point de vue logique

- Définition du système vu de l'intérieur : COMMENT peuvent être satisfaits les besoins des acteurs ;
- Décomposition orientée-objet
  - ✓ Décomposition en objets et classes ;
  - ✓ Regroupement en paquetages ;
  - ✓ Connexions par héritage, association, etc.
  - ✓ Accent sur l'abstraction, l'encapsulation, l'uniformité ;
  - ✓ Réalisation des scénarios des cas d'utilisation.

I.4.3 Point de vue processus

- Vue temporelle et technique : met en œuvre les notions de taches concurrentes, contrôle, synchronisation, etc.
- Décomposition en tâches et processus ;
- Regroupement des groupes de processus ;
- Communication ;
- Information sur les caractéristiques suivantes :
  - ✓ Disponibilité, fiabilité ;
  - ✓ Intégrité, performance ;
  - ✓ Contrôle.

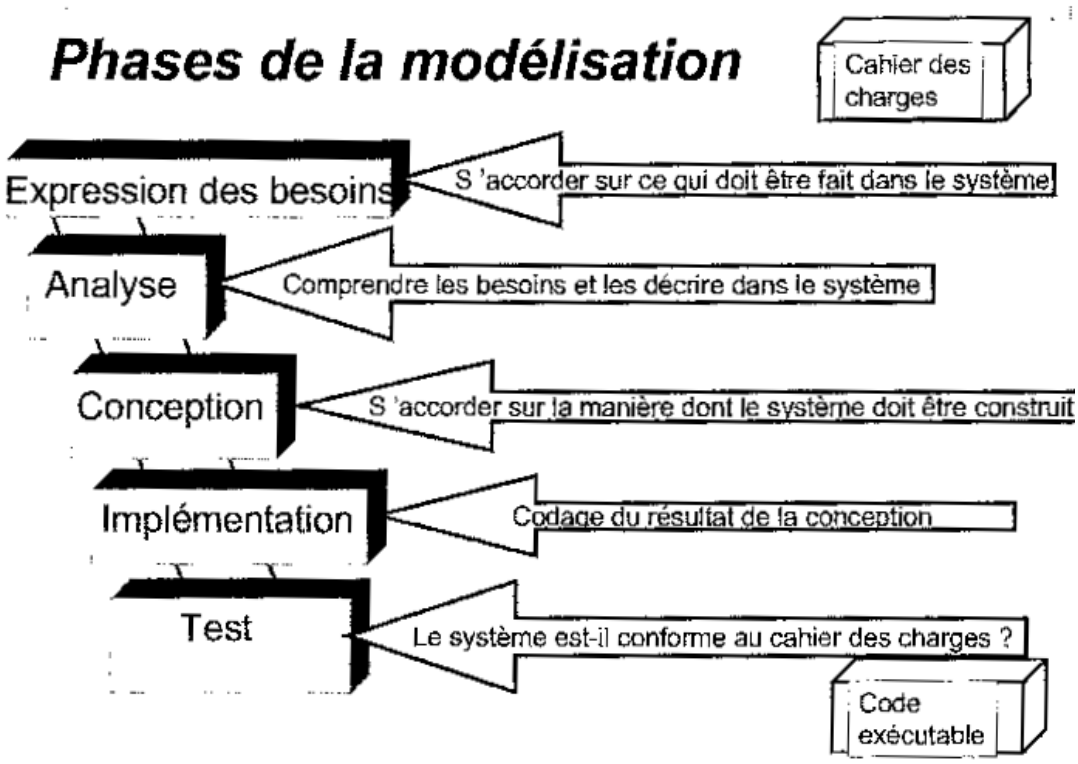
I.4.4 Point de vue implantation

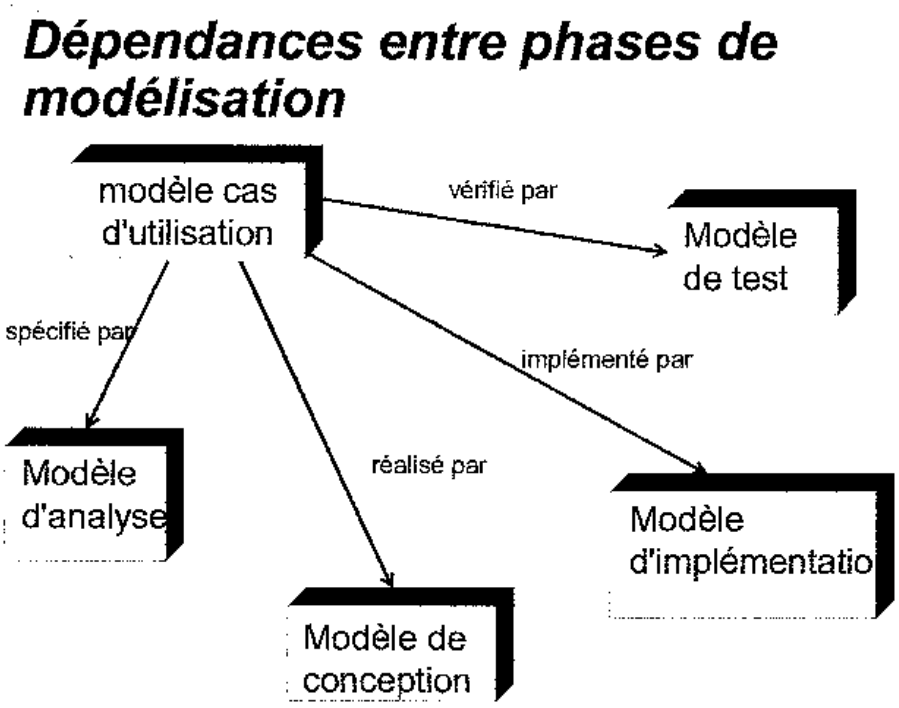
- Définit les dépendances entre les modules ;
- Décomposition en modules et niveaux ;
- Regroupement de modules en paquetages ;
- Organisation des sous-systèmes en niveaux pour :
  - ✓ Réduire le couplage et la visibilité ;
  - ✓ Augmenter la robustesse ;
- Information sur les caractéristiques suivantes :
  - ✓ Facilité de développement ;
  - ✓ Potentiel de réutilisation ;
  - ✓ Gestion de configuration ;

I.4.5 Point de vue déploiement

- Décrit la position géographique et l'architecture physique de chaque élément du système : OU ;
- Décomposition en nœuds d'exécution ;
- Rôle d'un nœud ;
- Inter-connectivité, topologie ;
- Information sur les caractéristiques suivantes :
  - ✓ Performance, disponibilité ;
  - ✓ Installation, maintenance.

I.5 Les Phases de la modélisation





Les différentes étapes de la modélisation

1. Expression des besoins ;
2. Spécification ;
3. Analyse ;
4. Conception ;
5. Implémentation ;
6. Tests de vérification ;
7. Validation ;
8. Maintenance et évolution.

I.5.1 Expression des besoins

ROLE :

- Permettre une meilleure compréhension du système
- Comprendre et structurer les besoins du client :
  - ✓ Clarifier, filtrer et organiser les besoins, ne pas chercher l'exhaustivité
- Une fois identifiés et structurés, ces besoins :
  - ✓ Définissent le contour du système à modéliser (ils précisent le but à

atteindre),

✓ Permettent d'identifier les fonctionnalités principales (critiques) du système,

✓ Comprendre le contexte du système en définissant un modèle du domaine et du métier,

✓ Recenser les besoins fonctionnels et les définir par des cas d'utilisations

✓ Noter les contraintes, exigences non fonctionnelles.

Le modèle du domaine regroupe les objets qui se situent dans le contexte du système : entités existantes ou événements qui s'y produisent.

➢ *Exigences non fonctionnelles:*

✓ Contraintes de concurrence ;

✓ Contraintes de temps de réponse ;

✓ Contraintes de distribution ;

✓ Contraintes de performance ;

✓ Contraintes de répartition.

I.5.1 Spécification

➢ Ce que le système doit être et comment il peut être utilisé.

I.5.2 Analyse

L'objectif est de déterminer les éléments intervenant dans le système à construire, ainsi que leur structure et leurs relations.

Elle doit décrire chaque objet selon 3 axes :

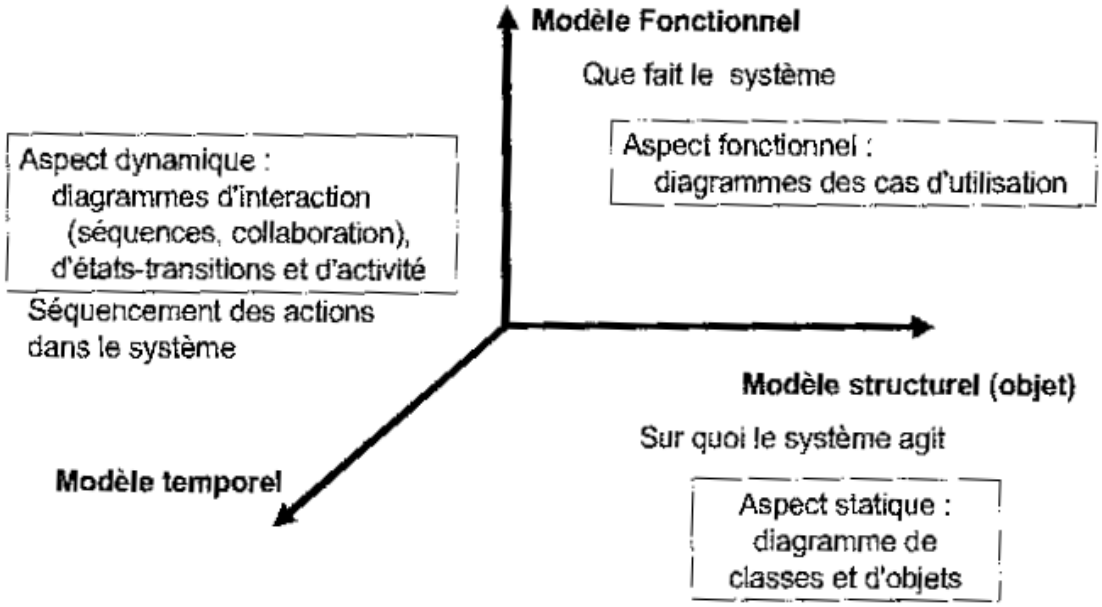
- Axe fonctionnel : savoir-faire de l'objet.

- Axe statique : structure de l'objet

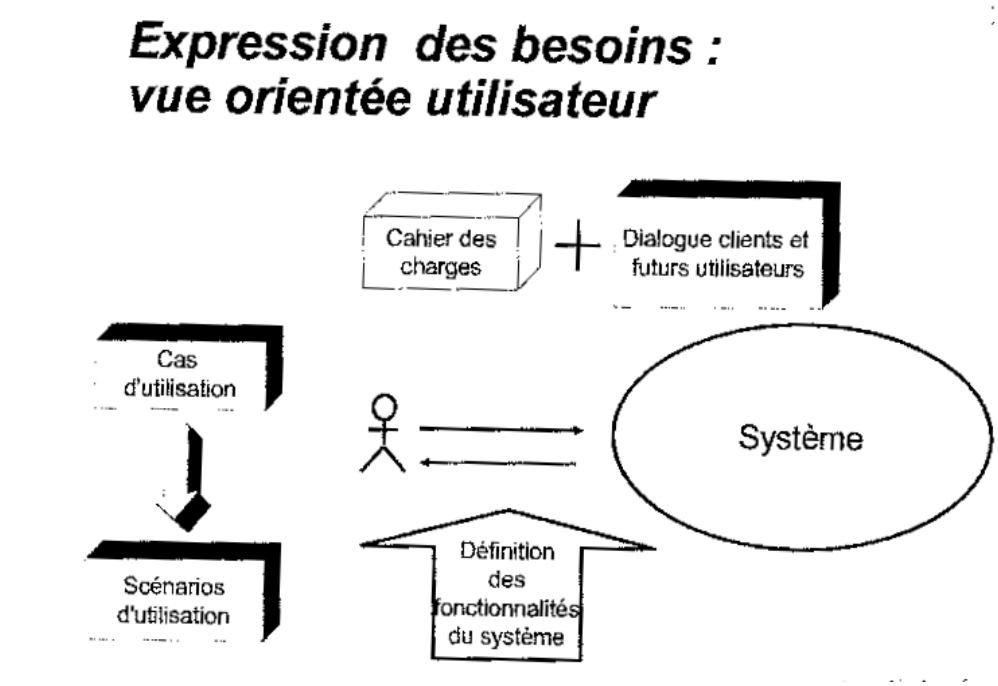
- Axe dynamique : cycle de vie de l'objet au cours de l'application (Etats et messages de l'objet).

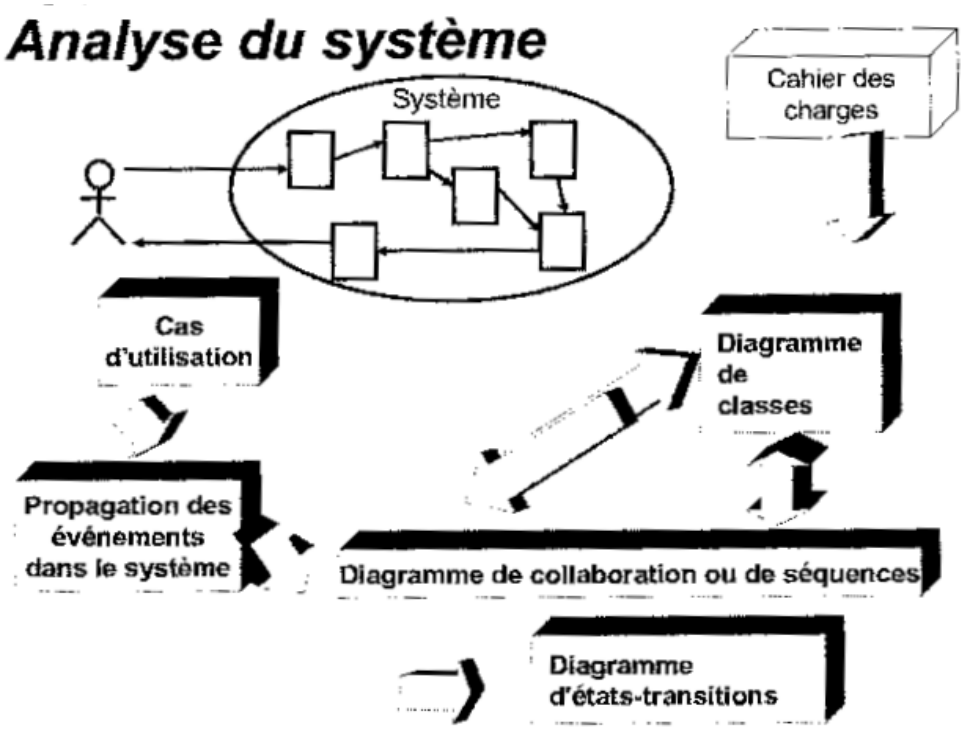
Ces descriptions ne tiennent pas compte des contraintes techniques (informatique).

Les trois composantes d'une modélisation



- Le but de l'analyse est de traduire dans un langage proche de celui des informaticiens les modèles exprimés dans l'expression des besoins.
- Cependant pour rester compréhensible par les clients ou utilisateurs, il n'intervient que des entités du domaine (métier) considéré.
- Elle sert d'interface, avec l'expression des besoins, aux dialogues et aux contrats avec les clients et les utilisateurs.
- L'analyse doit servir de support pour la conception, l'implémentation et la maintenance.





La notation graphique a pour but :

- Modéliser les objets, les relations entre objets, les interactions avec le système.
- Servir de support de communication entre l'analyste, le client, et les utilisateurs.

I.5.4 Conception

- Elle consiste à apporter des solutions techniques aux descriptions définies lors de l'analyse : architecture technique ; performances et optimisation ; stratégie de programmation.
- On y définit les structures et les algorithmes.
- Cette phase sera validée lors des tests.

I.5.5 Implémentation

- C'est la réalisation de la programmation.

I.5.6 Tests de vérification

- Ils permettent de réaliser des contrôles pour la qualité technique du système.
- Il s'agit de relever les éventuels défauts de conception et de programmation (revue de code, tests des composants,...).
- Il faut instaurer ces tests tout au long du cycle de développement et non à la fin pour éviter des reprises conséquentes du travail (programmes de tests robustes ; Logiciels de tests).

I.5.7 Validation

- Le développement d'une application doit être lié à un contrat ayant une forme de cahier de charges, où doivent se trouver tous les besoins de l'utilisateur. Ce cahier de charge doit être rédigé avec la collaboration de l'utilisateur et peut être par ailleurs complété par la suite.
- Tout au long des ces étapes, il doit y avoir des validations en collaboration également avec l'utilisateur.
- Une autre validation doit aussi être envisagée lors de l'achèvement du travail de développement, une fois que la qualité technique du système est démontrée. Elle permettra de garantir la logique et la complétude du système.

I.5.8 Maintenance et évolution :

- Deux sortes de maintenances sont à considérer :
  - Une maintenance corrective, qui consiste à traiter les "buggs ".
  - Une maintenance évolutive, qui permet au système d'intégrer de nouveaux besoins ou des changements technologiques.

CHAPITRE II

DIAGRAMMES

DES CAS D'UTILISATIONS
-
USE CASES DIAGRAM

Ce que doit faire le système sans spécifier comment il le fait

" L'art de la connaissance, c'est de savoir ce qui doit être ignoré "

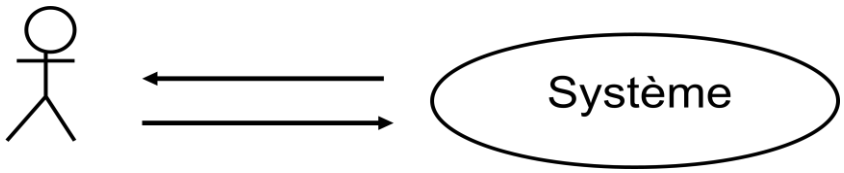
Djalâl-ad-Dîn Rûmî

II.1 Cahier des Charges : Gestion de Bibliothèque

- Un gérant de bibliothèque désire automatiser la gestion des prêts.
- Il commande un logiciel permettant aux utilisateurs de connaître les livres présents, d'en réserver jusqu'à 2. L'adhérent peut connaître la liste des livres qu'il a empruntés ou réservés.
- L'adhérent possède un mot de passe qui lui est donné à son inscription.
- L'emprunt est toujours réalisé par les employés qui travaillent à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre).
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts réalisés dans la bibliothèque.

II.2 But des Use Cases

- Les cas d'utilisation représentent les fonctionnalités que le système doit savoir faire.
- Chaque cas d'utilisation peut être complété par un ensemble d'interactions successives d'une entité en dehors du système (l'utilisateur) avec le système lui même.



Les Uses Cases permettent :

- De connaître le comportement du système sans spécifier comment ce comportement sera réalisé.
- De définir les limites précises du système
- Au développeur de bien comprendre l'attente des utilisateurs et les experts du domaine.

De plus les Use Cases sont :

- Des instruments de validation et de test du système en cours et en fin de construction.

II.3 Modèle des Cas d'Utilisations

Un diagramme de cas d'utilisation définit :

1. Les acteurs ;
2. Les cas d'utilisations ;
3. Le système ;
4. Les liens entre acteurs et cas d'utilisations.

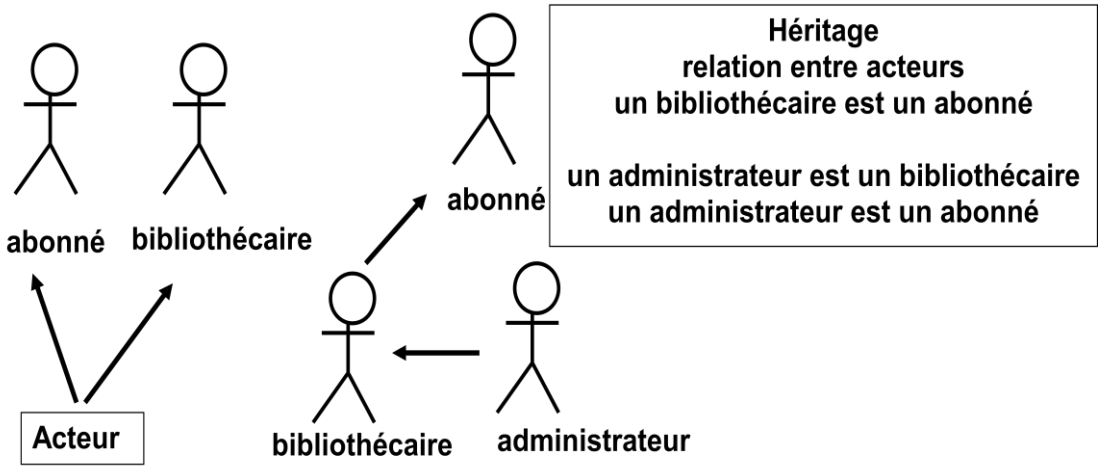
Un modèle de cas d'utilisation se définit par :

- Des diagrammes de cas d'utilisation ;
- Une description textuelle des scénarios d'utilisation ;
- Une description de ces scénarios par :
  - ✓ Les diagrammes de séquences ;
  - ✓ Les diagrammes de communication (collaboration).

II.4 Concepts de base du Diagramme des UC

II.4.1 Acteurs

- Un acteur représente une personne ou un périphérique qui joue un rôle (interagit) avec le système.
- Relation entre acteurs : généralisation (héritage)

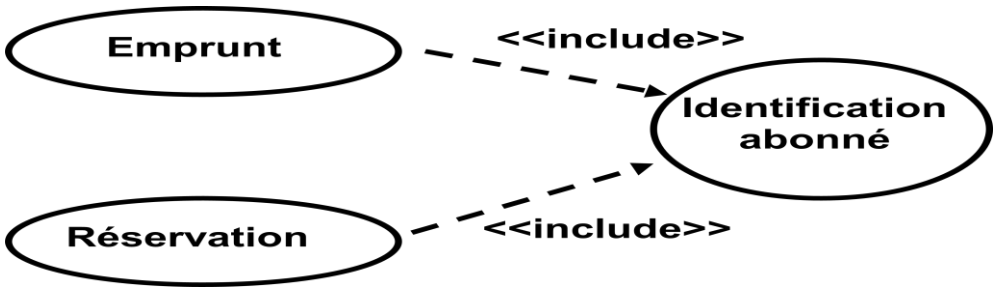


II.4.2 Cas d'utilisation (Use-Case)

- Un cas d'utilisation est un moyen de représenter les différentes possibilités d'utiliser un système.
- Il exprime toujours une suite d'interactions entre un acteur et l'application.
- Il définit une fonctionnalité utilisable par un acteur.

II.4.2.1 Organisation des Use Cases : «include»

- La relation **«include»** précise qu'un cas d'utilisation contient le comportement défini dans un autre cas d'utilisation.
- Cette relation permet de mettre en commun des comportements communs à plusieurs cas d'utilisation.

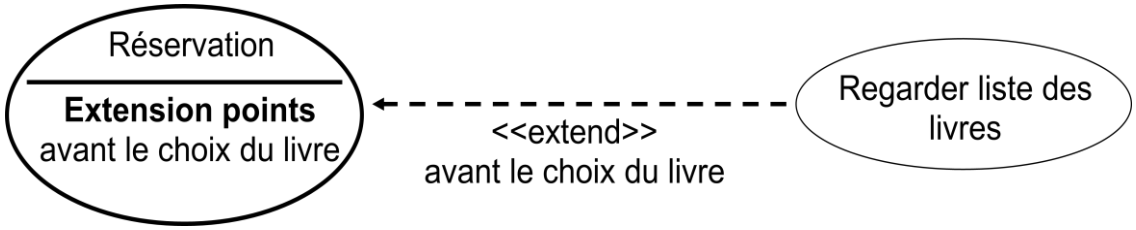


Un cas A est inclus dans un cas B si le comportement décrit par le cas A est inclus dans le comportement du cas B: on dit alors que le cas B dépend de A.

II.4.2.2 Organisation des Use Cases: «extend»

- La relation ***«extend»*** précise qu'un cas d'utilisation peut dans certains cas augmenter le comportement d'un autre cas d'utilisation.
- Une condition devra valider cette augmentation.
- Le point d'utilisation de cette augmentation peut être défini dans un "point d'extension".

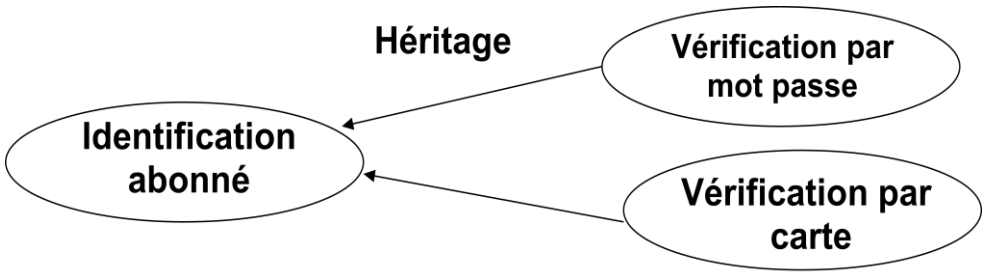
- Si le comportement de B peut être étendu par le comportement de A on dit que A étend B.
- Un point d'extension doit spécifier le ou les points du cas d'utilisation au niveau desquels le comportement d'extension entre en jeu.



- Dans cet exemple, le cas d'utilisation « Regarder la liste des livres » augmente le cas d'utilisation d'une réservation, avant le choix du livre, si l'utilisateur en fait la demande.

II.4.2.3 Organisation des Use Cases : Généralisation

- Cette relation **« *est un* »** introduit la notion d'héritage.
- Les cas d'utilisation "Vérification par mot passe" et "Vérification par carte" sont des spécialisations du cas d'utilisation "Identification abonné".
- Autrement dit : si l'on dit que l'on fait une "Identification abonné", ce peut être une "Vérification par carte" ou une "Vérification par mot passe".



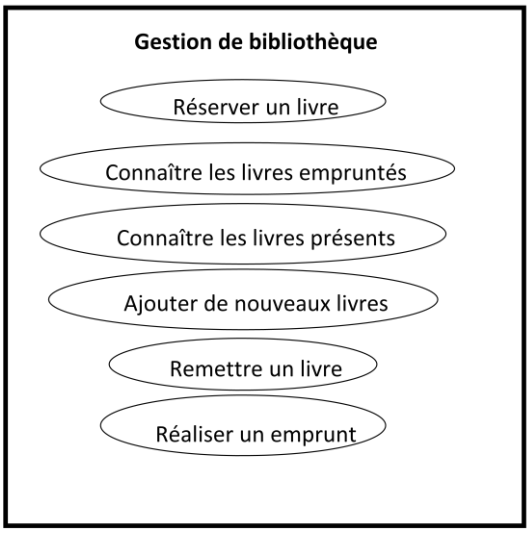
II.4.2.4 Modélisation d'un Système : Obtenir les Cas d'Utilisation

- Identifier les acteurs qui utilisent, qui gèrent, qui exécutent des fonctions spécifiques.
- Organiser les acteurs par relation d'héritage.
- Pour chaque acteur, rechercher les cas d'utilisation avec le système. En particulier, ceux qui modifient l'état du système ou qui attendent une réponse du système.
- Ne pas oublier les variantes d'interactions (cas d'erreur, cas interdits).

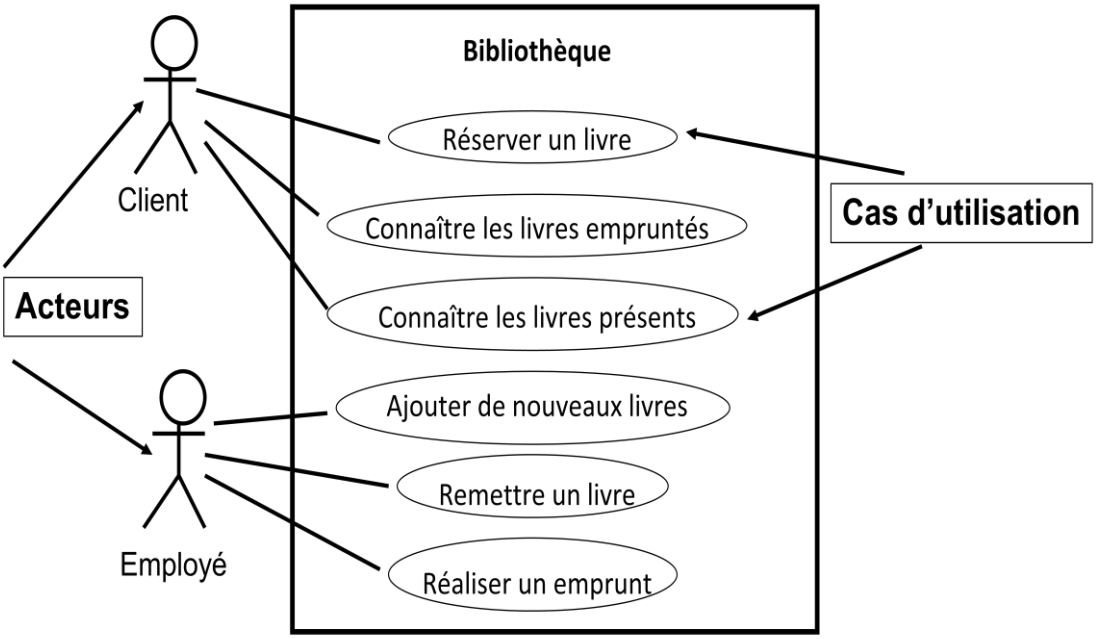
- Organiser ces interactions par héritage, par utilisation et par extension.

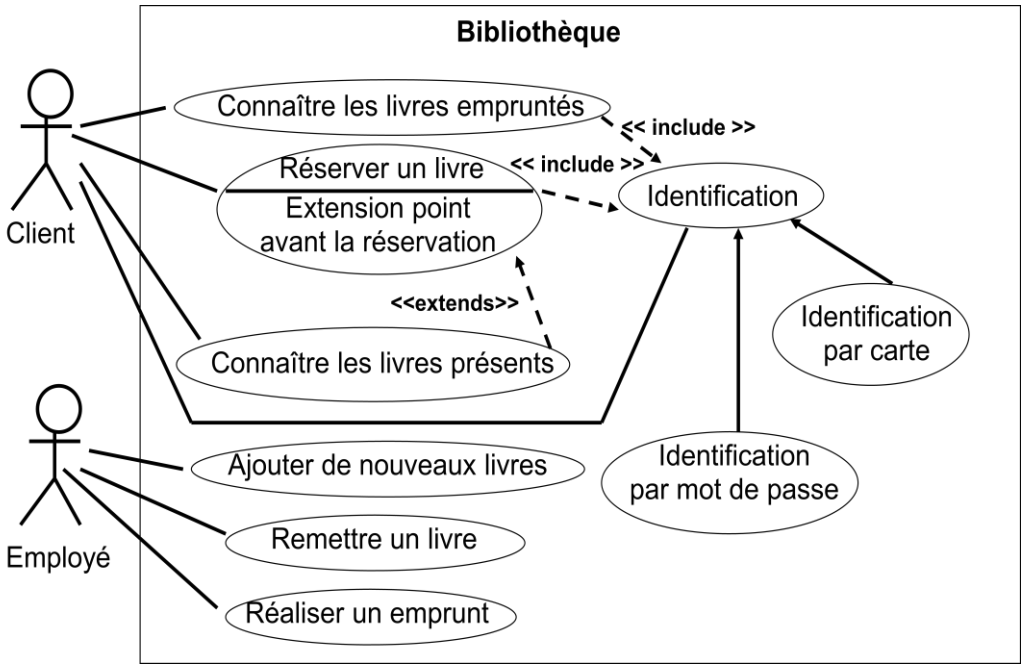
II.4.3 Système

- Le système définit l'application informatique, il ne contient donc pas les acteurs, mais les cas d'utilisation et leurs associations.



II.5 Elaboration du Diagramme Des UC de la Gestion de Bibliothèque





II.6 Elaboration du Modèle des Cas d'Utilisation de la Gestion de Bibliothèque

II.6.1 Scénarios d'un cas d'utilisation : Définition

- La description d'un cas d'utilisation se fait par des scénarios qui définissent la suite logique des interactions qui constituent ce cas.
- On peut définir des scénarios simples ou des scénarios plus détaillés faisant intervenir les variantes, les cas d'erreurs etc.
- Cette description se fait de manière simple, par un texte compréhensible par les personnes du domaine de l'application.
- Elle précise ce que fait l'acteur et ce que fait le système
- La description détaillée pourra préciser les contraintes de l'acteur et celles du système.

II.6.2 Scénarios du UC : Réservation d'un livre

Réservation d'un livre

description simplifiée

Le client se présente devant un terminal:

• (1) Le système affiche un message d'accueil.

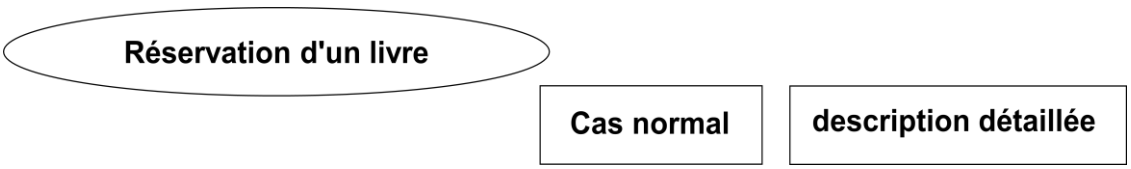
• (2) Le client choisit l'opération réservation parmi les différentes opérations

proposées.

- (3) Le système lui demande de s'authentifier.
- (4) Le client donne son identification (nom, mot de passe).
- (5) Le système lui demande de choisir un livre.
- (6) Le client précise le livre qu'il désire.
- (7) Le système lui précise si un exemplaire du livre lui est réservé.

- Pré-conditions:
  - ✓ Le client doit être inscrit à la bibliothèque ;
  - ✓ Le client ne doit pas avoir atteint le nombre maximum de réservation ;
  - ✓ Un exemplaire du livre doit être enregistré ;
- Post-conditions: (Si l'opération s'est bien déroulée) ;
  - ✓ Le client a une réservation supplémentaire ;
  - ✓ Le nombre d'exemplaires disponibles du livre est décrémenté de 1.

Scénarios du cas d'utilisation Réservation d'un livre



1. (1) Le système affiche un message d'accueil sur le terminal avec un choix d'opérations ;
2. (2) Le client choisit l'opération réservation parmi les différentes opérations proposées ;
3. (3) Le système lui demande de s'authentifier ;
4. (4) Le client donne son identification (nom, mot de passe) ;
5. (5) Le système demande le titre du livre en donnant la possibilité de choisir dans une liste ;
6. (6) Le client précise le livre qu'il désire ;

7. (7) Le système lui précise qu'un exemplaire du livre lui est réservé.

Scénarios du cas d'utilisation réservation d'un livre (Variantes)

- ✓ Variante 1 : en (6) le client demande à connaître les livres présents ;
- ✓ Variante 2 : en (4) le client n'est pas reconnu, dans ce cas, tant qu'il n'est pas reconnu, on lui redemande de s'authentifier ;
- ✓ Variante 3 : en (4) le client est reconnu, mais le mot de passe est incorrect, il peut avoir 5 essais, par la suite le client sera interdit pendant 1 heure ;
- ✓ Variante 4 : en (4) le client n'a plus le droit de réserver ;
- ✓ Variante 5 en (7) le livre n'est pas libre.

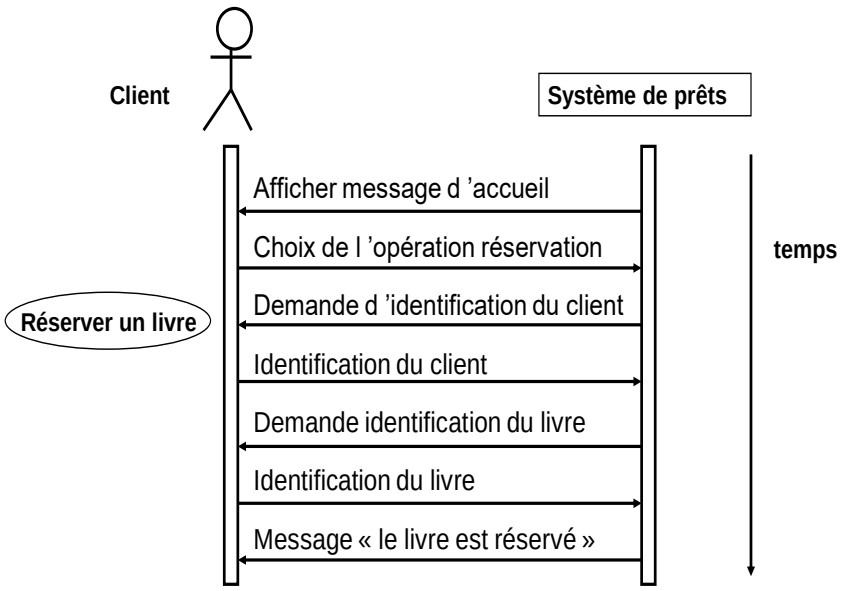
II.6.3 Représentation du Scénario par diagramme de séquences

➢ Suite aux descriptions textuelles, le scénario peut être représenté en utilisant un diagramme de séquences.

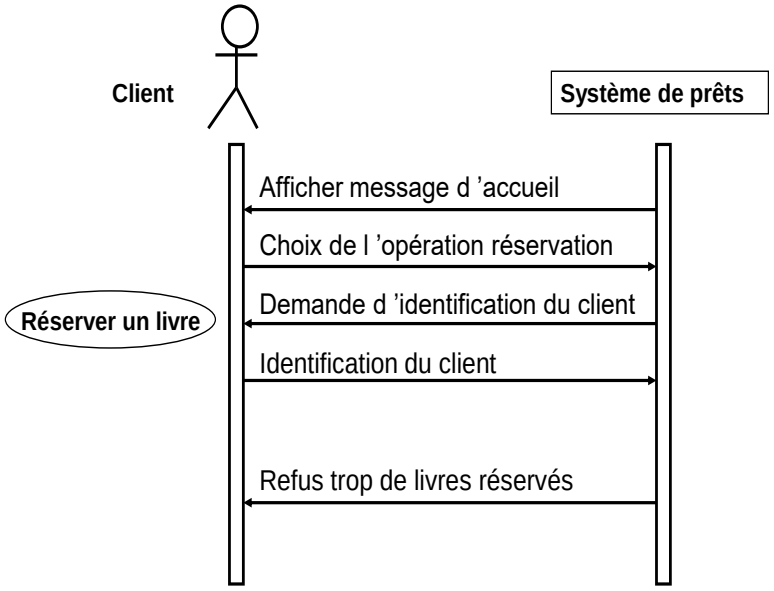
Le diagramme de séquences permet :

- De visualiser l'aspect temporel des interactions
- De connaître le sens des interactions (acteur vers système ou contraire)

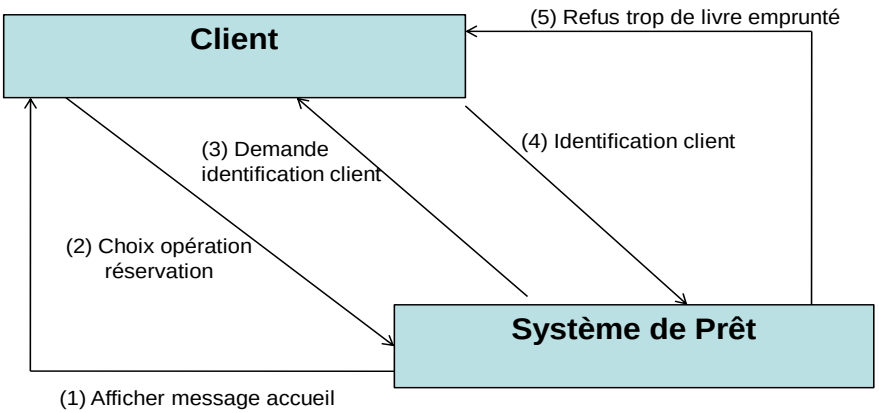
Diagramme de Séquence



Variation du scénario



Déscription à l'aide de diagramme de communication



II.7 Cas d ’utilisation: Distributeur de billets

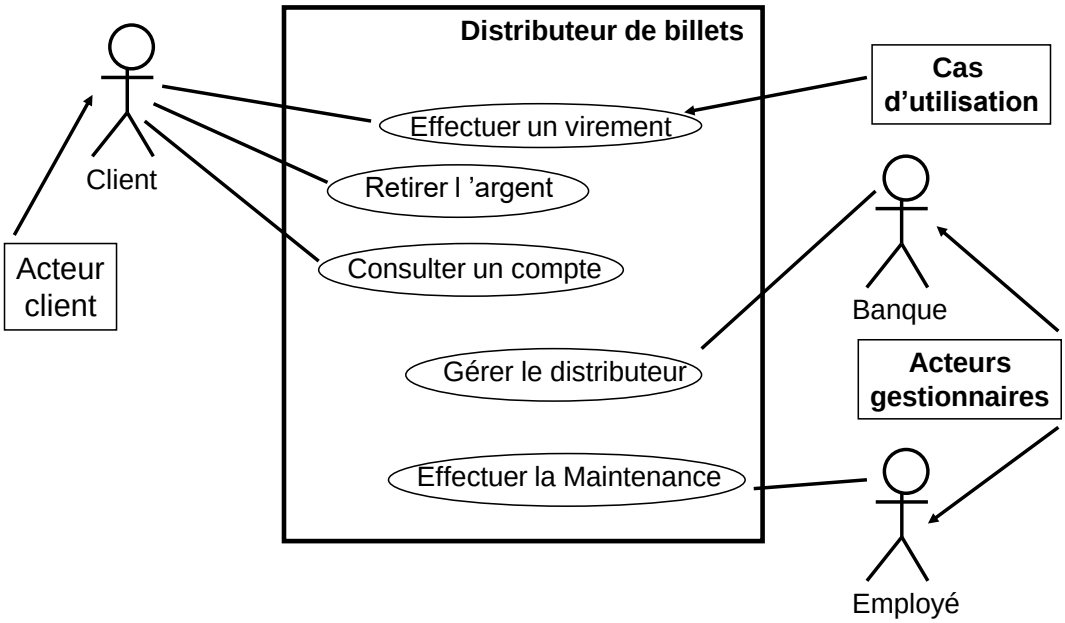
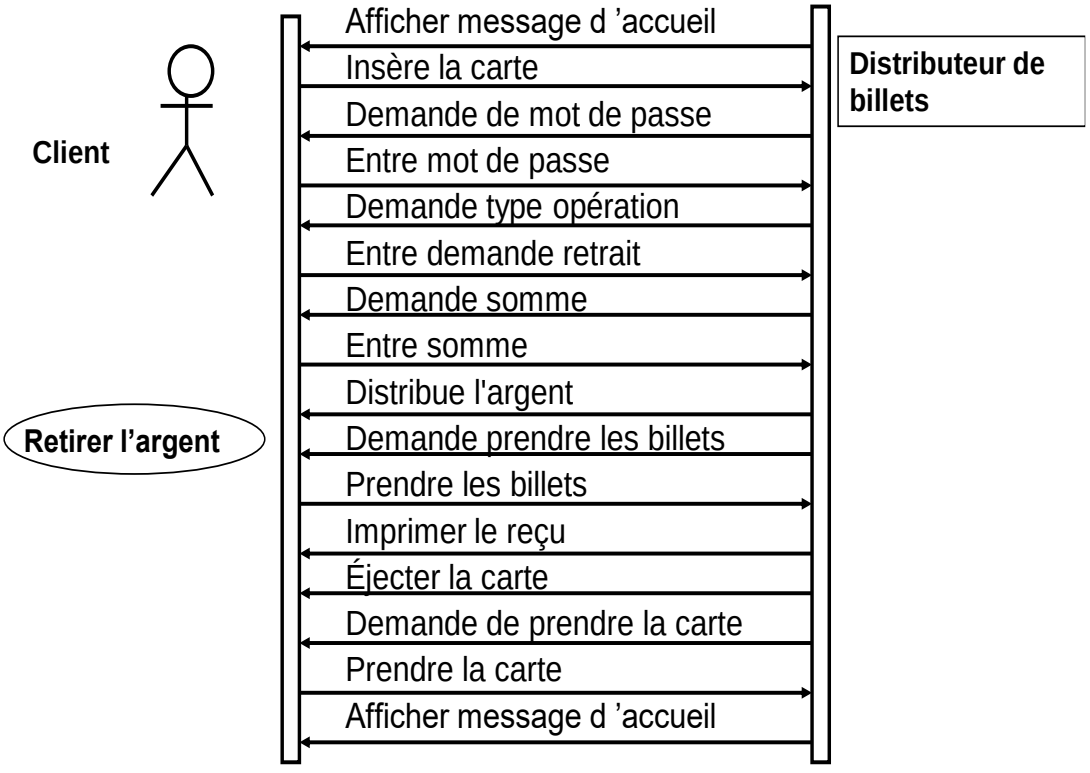


Diagramme de séquences : Use Case Retirer de l'argent



CHAPITRE III

CONCEPTS OBJETS
&
DIAGRAMME DE CLASSES
-
CLASS DIAGRAM

"Le Ce n'est pas que je suis si intelligent, c'est que je reste plus longtemps avec les problèmes"

Albert Einstein

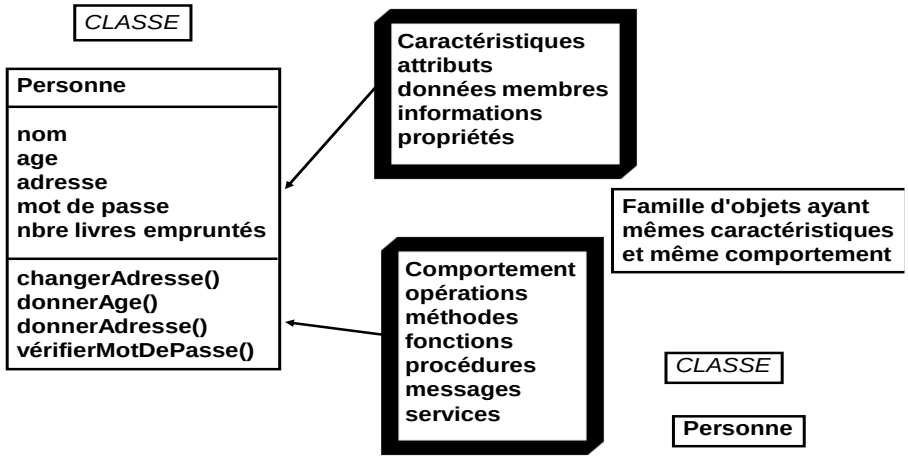
SECTION A

CONCEPTS OBJETS

"Le plus important pour un scientifique n'est pas ses diplômes, ni le nombre de ses années d'étude, ni même son expérience, mais tout simplement son intuition"

Albert Einstein

III.A.1 CLASSE : DEFINITION



LA CLASSE

Une **classe** est une description d'un ensemble d'objets ayant une sémantique, des attributs des méthodes et des relations en commun.

La modélisation objet consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (arbre, personne, téléphone, etc.) ou bien virtuelle (sécurité sociale, compte bancaire, etc.). Un objet est caractérisé par plusieurs notions:

Attributs / Méthodes / Identité :

- *Les attributs* (on parle parfois de propriétés): Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet
- *Les méthodes* (appelées parfois fonctions membres): Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier.
- *L'identité*: L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro d'immatriculation, etc.).

Attributs / Operations: Syntaxe

- Attributs :

  nomAttribut : type = valeur initiale

  Exemples :

  prenom : String

  size : int = 100

- Operations :

  nomOpération (param1 : type1, param2 : type2…) : typeResultat

  Exemples :

  getName() : String

  setName(newName : String)

Operations (Méthodes)

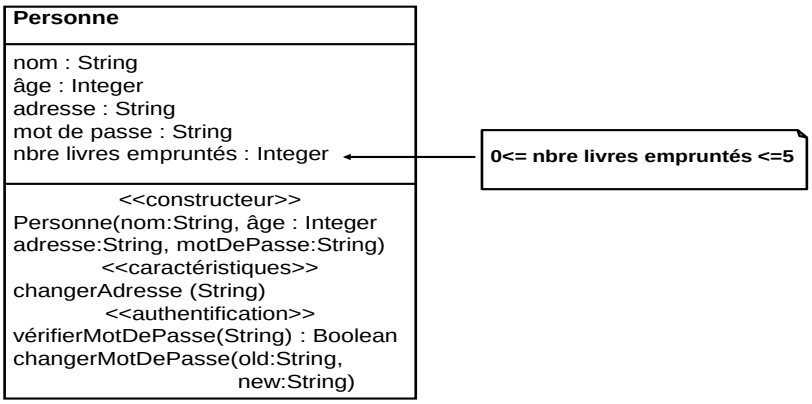
Implémentation d'un service offert par l'objet, correspondant à une partie de ses responsabilités

- Accesseurs : une opération qui renvoie une information sur l'état de l'objet (fonction)
- Modificateur : une opération qui modifie l'état de l'objet (procédure)
- Constructeur : une opération de la classe qui permet d'initialiser une nouvelle instance

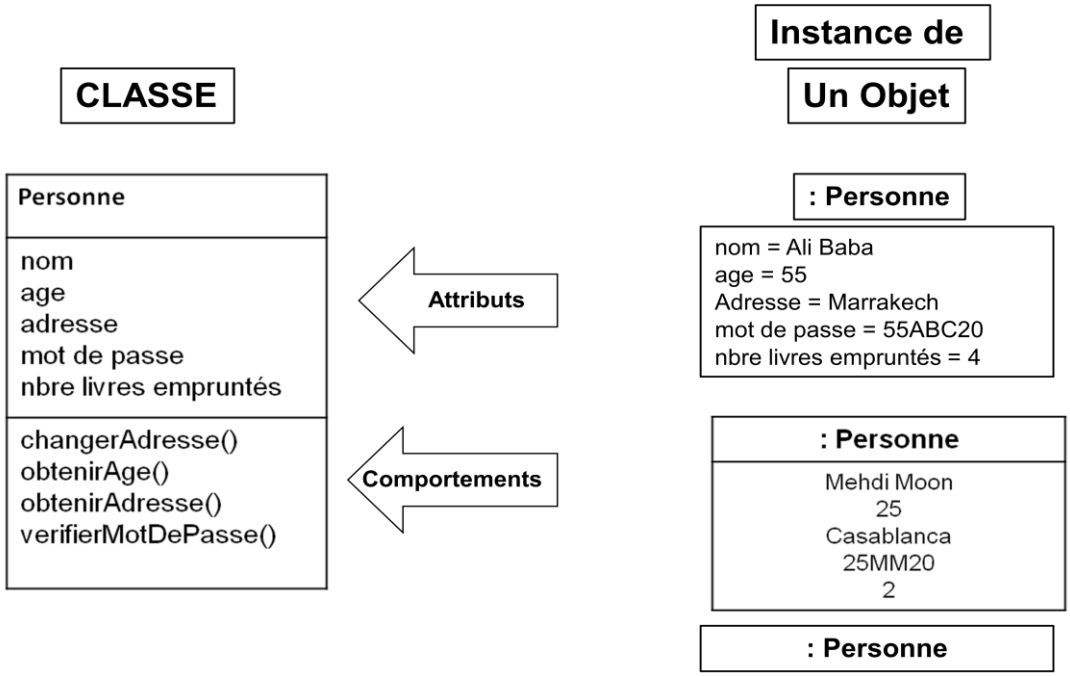
Les propriétés de classe sont soulignées

- Mots-clé 'static' en java

La Classe et ses membres



III.A.2 Classe & Objet



III.A.3 Protection des Attributs et des Opérations

III.A.3.1 Principe de l'encapsulation

- Peut-on accéder à tous les attributs ou à toutes les méthodes d'un objet ? NON
- La classe définit ce qui est accessible.
- C'est le principe de l'encapsulation.
- Un objet complexe ne peut être utilisé qu'au travers de ce qui est accessible.
- En effet, la programmation orientée objet permet de cacher l'implémentation d'un objet en ne lui permettant d'accéder aux attributs que, uniquement par l'intermédiaire de méthodes crées à cet effet (on parle d'interface).

Exemple :

- On ne peut utiliser une *voiture* qu'à travers son *volant*, son *frein*, son *accélérateur*, etc.
- L'accès au *carburateur* est impossible sauf par les méthodes qui le font de manière cohérente (méthode *accélérer* de l'*accélérateur*).

III.A.3.2 Usage et Notation

- Les **attributs** sont en général inaccessibles (secret). Ils sont alors qualifiés de :
  - ✓ "***protected***" (notation UML : **#)**
  - ✓ ou "***prIV.Ate***" (notation UML: **-)**

  Leur lecture ou modification n'est possible qu'au travers de certaines opérations (accesseurs : *changerAdresse()*, *obtenirAge()*, etc.)
- Les **opérations** sont en général accessibles (publiques) : Elles sont alors qualifiées de :
  - ✓ "***public***" (notation UML: **+**)

  Certaines opérations peuvent cependant être privées et certains attributs peuvent être publics (non souhaitable / principe d'encapsulation).

Visibilité des Attributs / Opérations :

- Public (+)

Visible et utilisable par toute autre classe (utilisation très limitée pour les attributs)

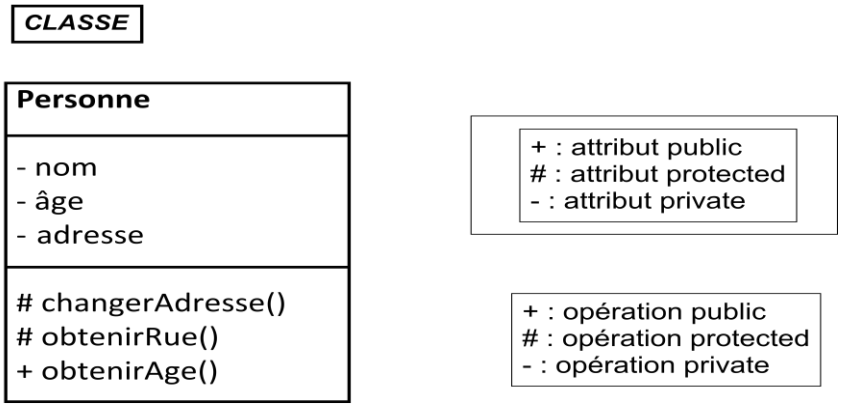
- Protected (#)

Visible et utilisable par toute les spécialisation de la classe

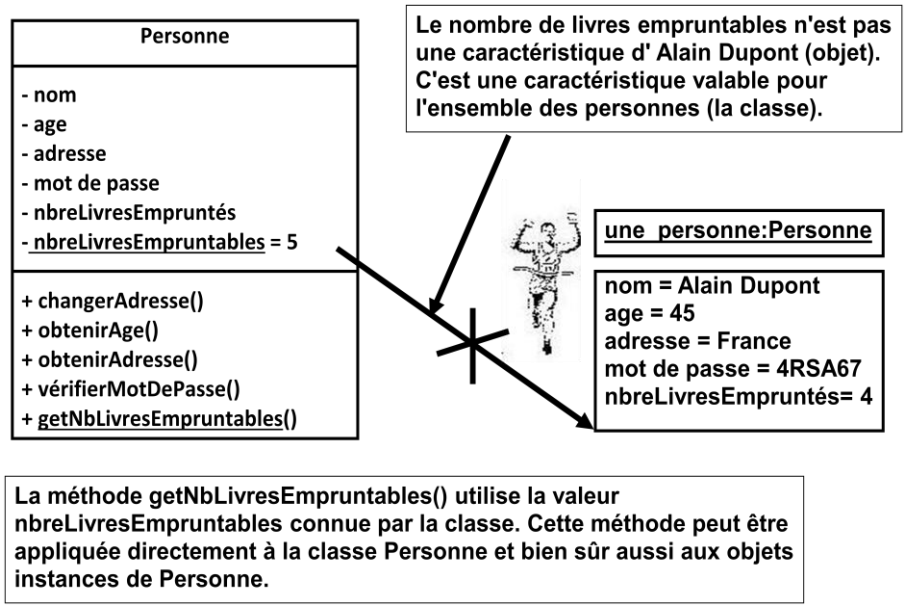
- PrIV.Ate (-)

Visible uniquement par la classe elle-même.

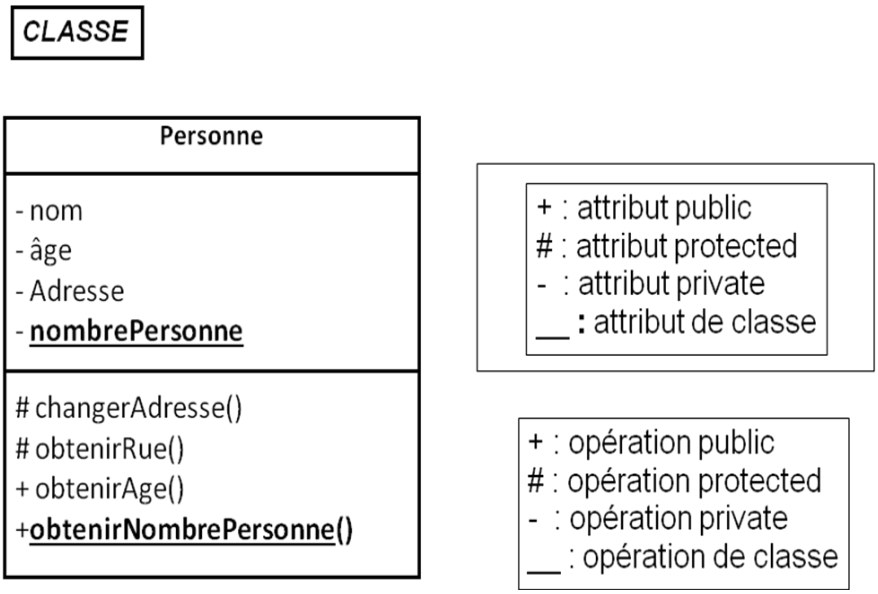
Notation UML



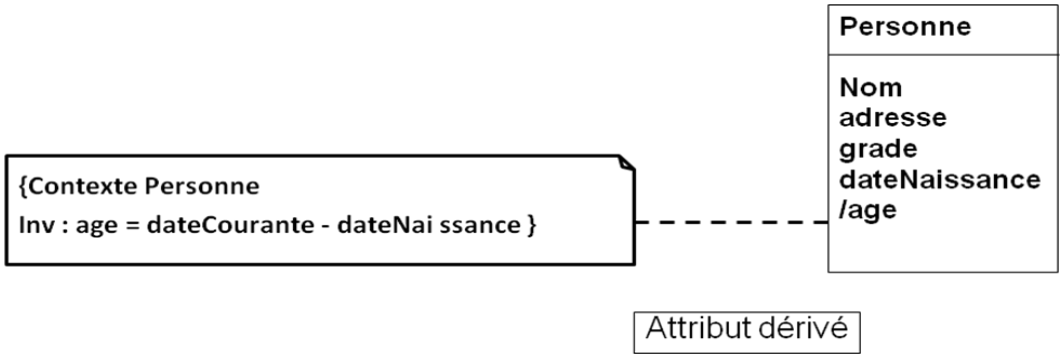
III.A.3.3 Attributs et Opérations de Classe



Attributs et Opérations de Classe : Notation UML



III.A.3.4 Attribut dérivé

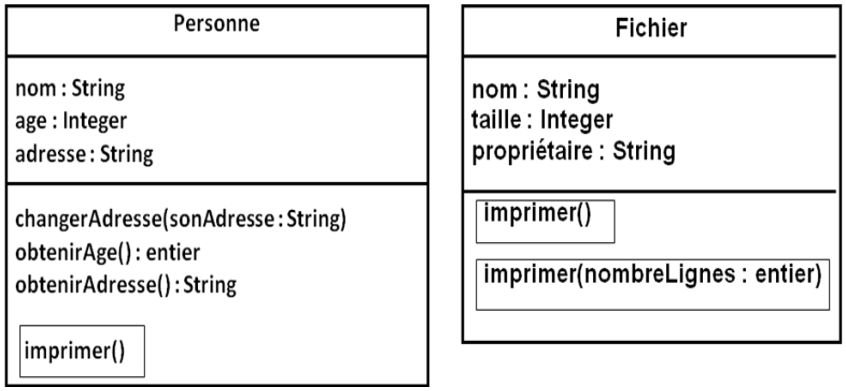


Syntaxe OCL : Object Constraint Langage

{Contexte Personne

Inv : age = dateCourante - dateNai ssance }

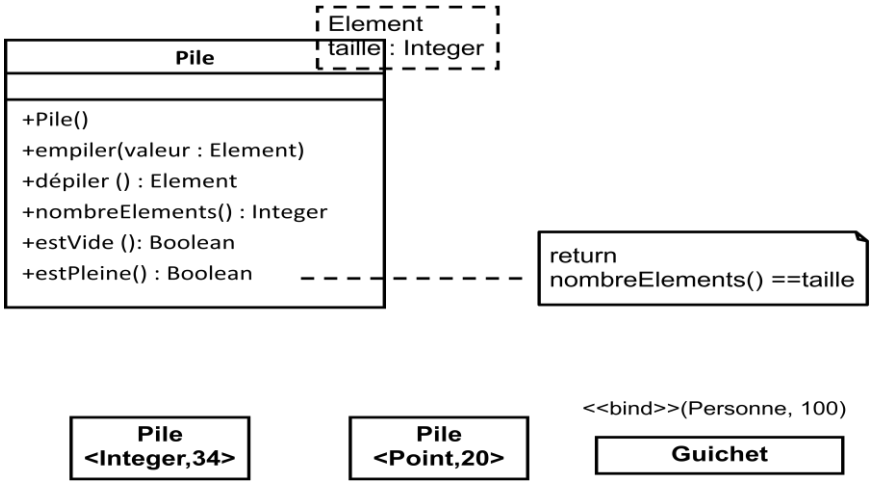
III.A.4 Surcharge & Polymorphisme



Définition :

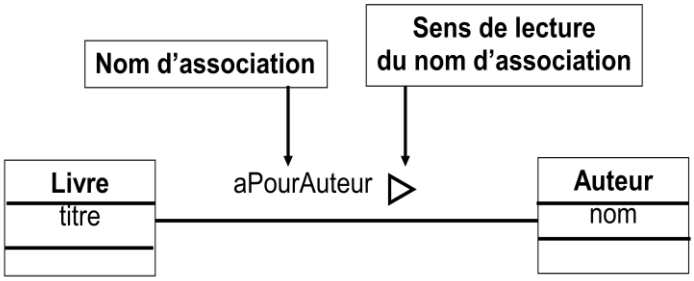
- Le **polymorphisme** représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Ce qui signifie qu'une même opération peut se traduire différemment selon l'objet sur laquelle elle s'applique.
- La **surcharge** (prototype de fonctions : plusieurs méthodes portant le même nom à l'intérieur d'une classe). La surcharge est la possibilité de donner le même nom à des méthodes ayant des signatures différentes.
- Une méthode est identifiée par sa **signature** (Nom de la méthode; Nombre et/ou type de chaque paramètre).

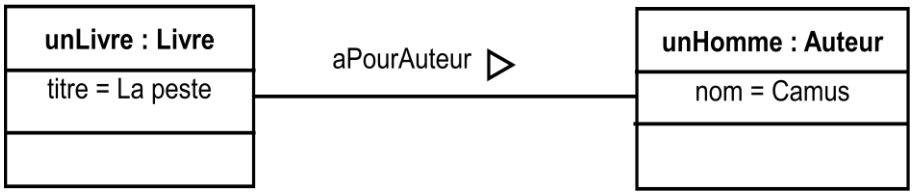
Classes Paramétrables :



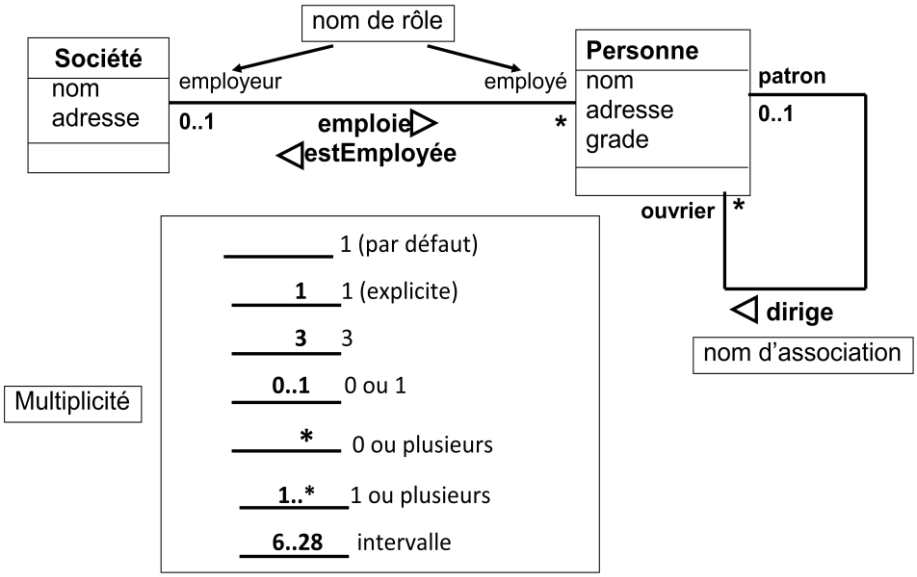
III.A.5 Principaux Concepts du Diagramme de Classe

III.A.5.1 Associations entre Classes



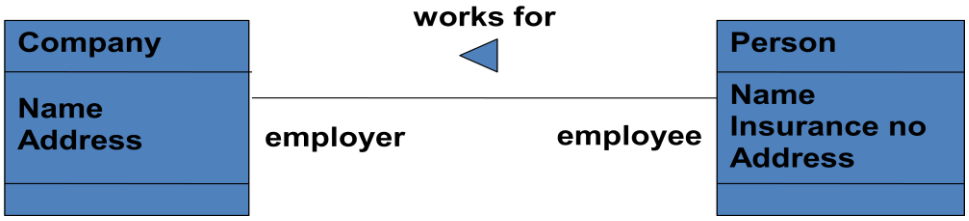


III.A.5.2 Multiplicité, Nom d'Association, Nom de Rôle



III.A.5.3 Nommage des Rôles

- Le rôle décrit une extrémité d'une association ;
- Les noms des rôles sont obligatoires pour les associations réflexives ;



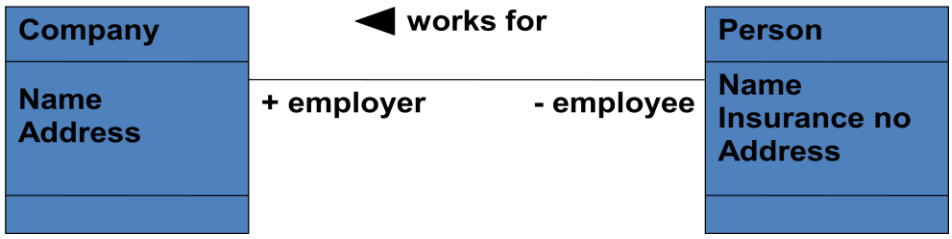
Visibilité des Rôles :

- Les rôles définissent des responsabilités
- Connaissant une personne on doit pouvoir retrouver ses employeurs

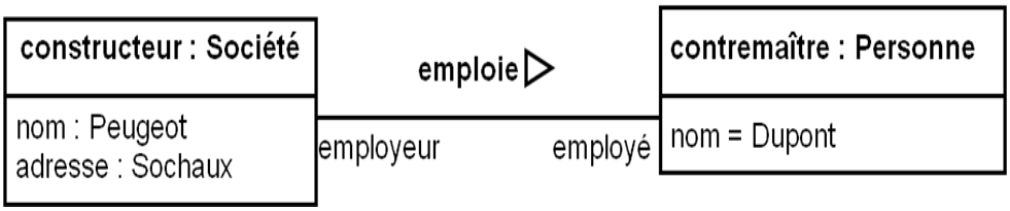
  il doit y avoir une méthode publique dans « person » qui renvoie une collection d'instances de « company »

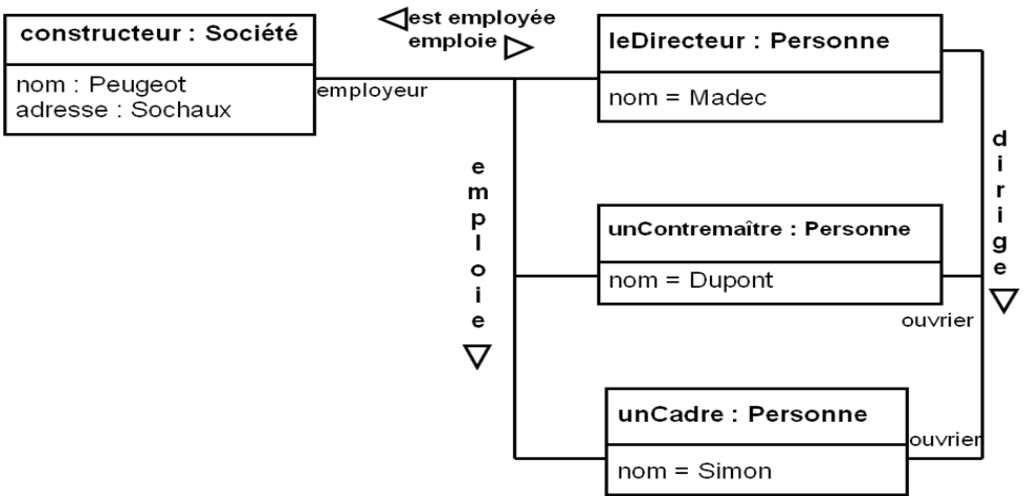
Exemple java :

Public Collection<Company> getEmployers()

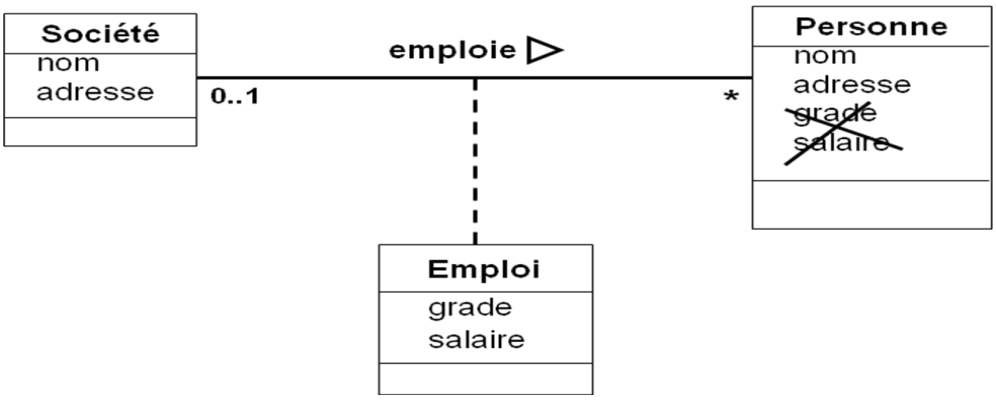


III.A.5.4 Lien entre Objets

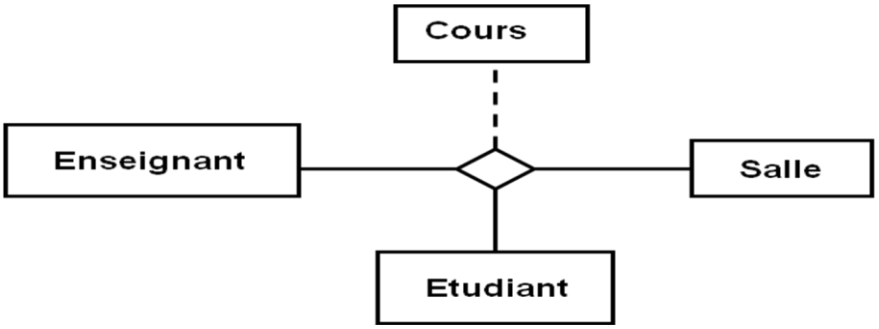




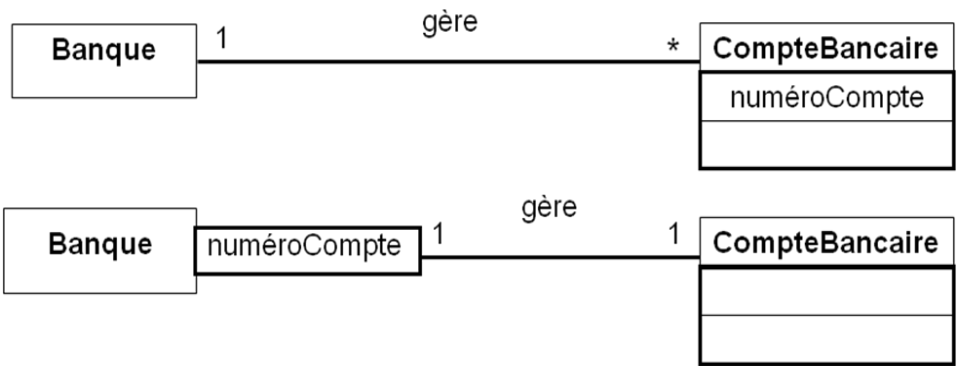
III.A.5.5 Classe d'Association



Associations ternaires :



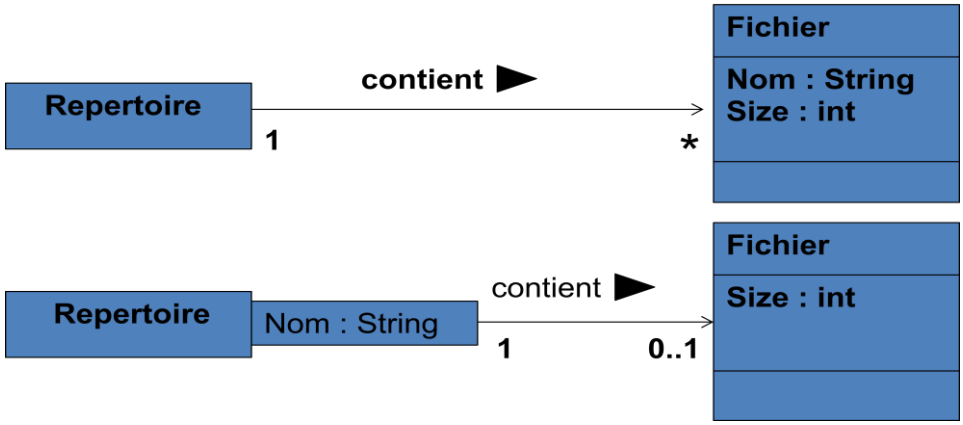
III.A.5.6 Qualificateurs : restriction de la cardinalité



La **restriction** (dite **qualification** en UML) d'une association consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association. Elle est réalisée au moyen d'une clé. La clé appartient à l'association et non aux classes associées.

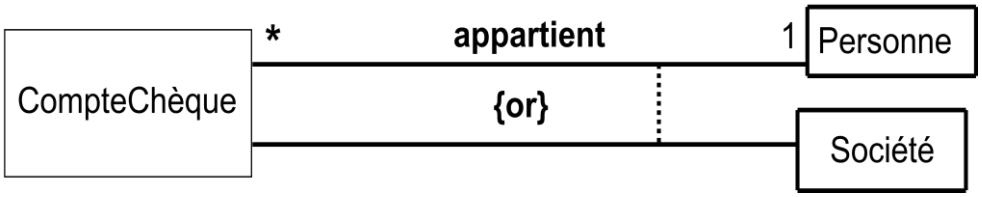
Association Qualifiée :

Une association qualifiée met en relation deux classes sur la base d'une clef.

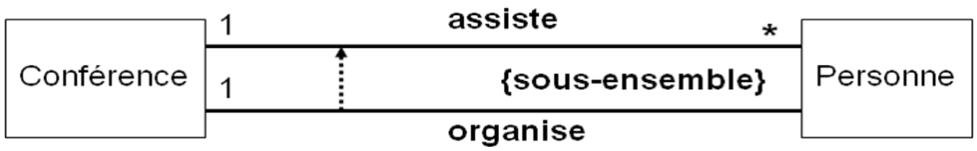


III.A.5.7 Contraintes sur les Associations

Contrainte d'association : porte sur une relation ou sur un groupe de relations (notée {contrainte}).



La contrainte **{Ou-exclusif}** précise que, pour un objet donné, une seule association parmi un groupe d'associations est valide.



La contrainte **{sous-ensemble}** indique qu'une collection est incluse dans une autre collection.

III.A.5.7.1 Object Constraint Language OCL

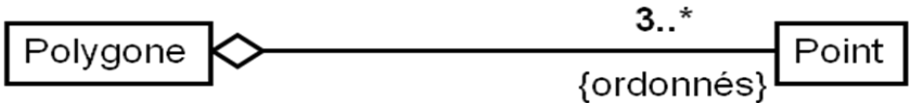
- OCL (Object Constraint Language) est un langage informatique d'expression des contraintes utilisé par UML. C'est une contribution d'IBM à UML 1.1.
- Ce langage formel est volontairement simple d'accès et représente un juste milieu entre langage naturel et langage mathématique. Il permet ainsi de limiter les ambigüités dans la spécification des contraintes logicielles. Sa grammaire simple lui permet d'être interprété par des outils logiciels pour faire de la programmation par contrat et vérifier qu'un logiciel répond à ses spécifications techniques.
- Pour spécifier complètement une application, les diagrammes UML seuls sont généralement insuffisants. Il y'a toujours une nécessité de rajouter des contraints.
- Comment exprimer ces contraintes ?
  - Langue naturelle mais avec manque de précision, compréhension pouvant être ambiguë ;
  - Langage formel avec sémantique précise : par exemple OCL .
- OCL : Object Constraint Language
  - Langage de contraintes orienté-objet ;
  - Langage formel (mais « simple » à utiliser) avec une syntaxe, une grammaire, une sémantique (manipulable par un outil) ;
  - S'applique entre autres sur les diagrammes UML.
- OCL peut s'appliquer sur la plupart des diagrammes UML
- Il sert, entre autres, à spécifier des :
  - Invariants sur des classes ;

- Pré et post conditions sur des opérations ;
- Gardes sur transitions de diagrammes d'états ou de messages de diagrammes de séquence/collaboration ;
- Des ensembles d'objets destinataires pour un envoi de message ;
- Des attributs dérivés ;
- Des stéréotypes.

III.A.6 Types d'Associations

III.A.6.1 Agrégation

C'est une association qui exprime un **couplage fort** lié à une **relation de subordination**, elle est **asymétrique** du type **ensemble/élément**.



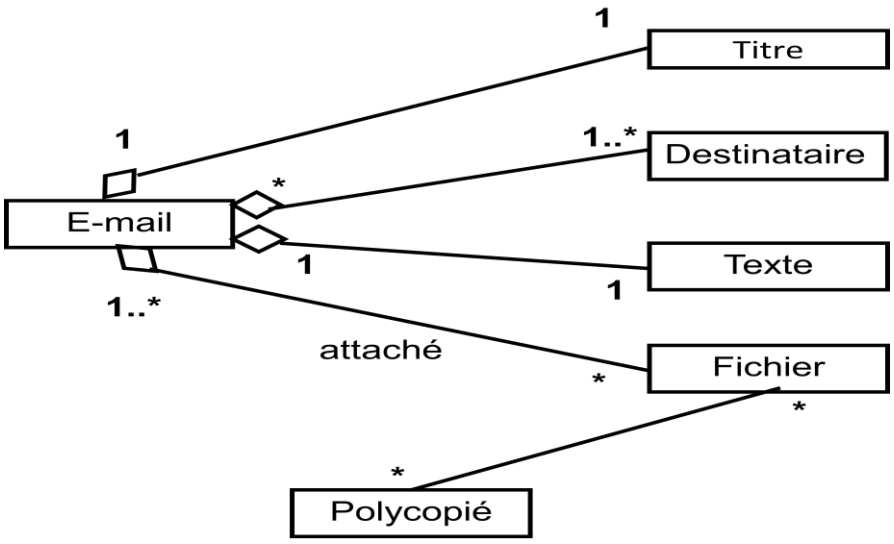
Règles permettant de choisir une agrégation :

- ✓ Est ce que c'est une partie de?
- ✓ Les opérations appliquées à l'ensemble sont elles appliquées à l'élément?
- ✓ Les changements d'états sont-ils liés ?

Mais

- ✓ Un élément agrégé peut être lié à d'autres classes ;
- ✓ La suppression de l'ensemble n'entraîne pas celle de l'élément.

Agrégation : Exemple



III.A.6.2 Composition : Agrégation forte

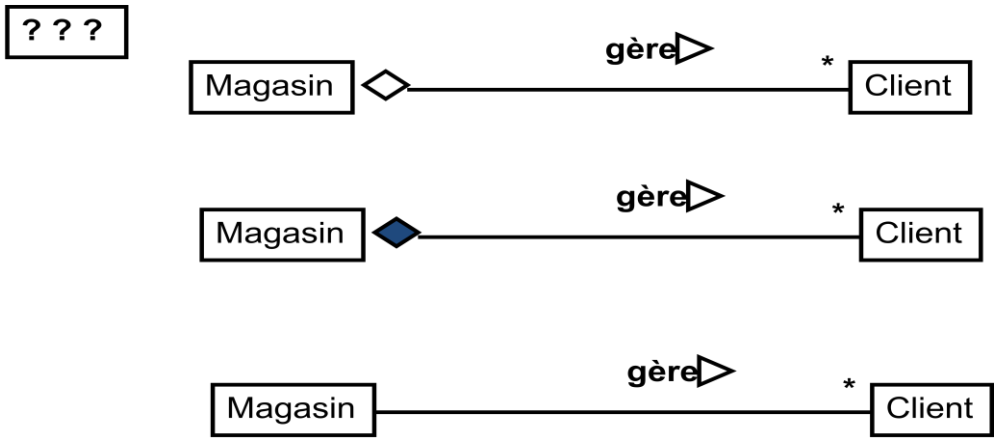
- La composition est une **agrégation forte** qui **lie les cycles de vie** entre le composé (ensemble) et les composants (éléments).
- Le choix entre composition et agrégation peut être laissé à la phase de conception.

Association, Agrégation ou Composition ?

Règles obligatoires pour la composition :

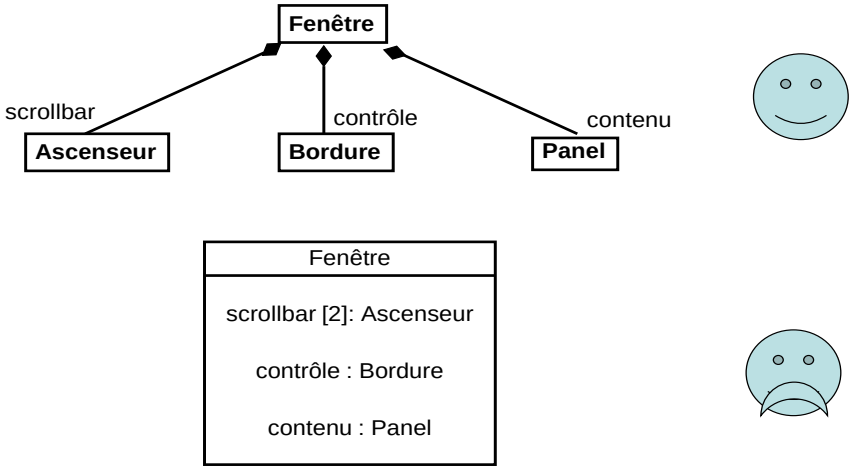
- La suppression du composé entraîne elle la suppression des composants ?
- Les attributs du composé sont-ils utilisés dans les composants ?
- Les composants sont des instances du composé ?
- Un composant ne peut pas être en relation avec d'autres classes externes au composé.

III.A.6.3 Exemples



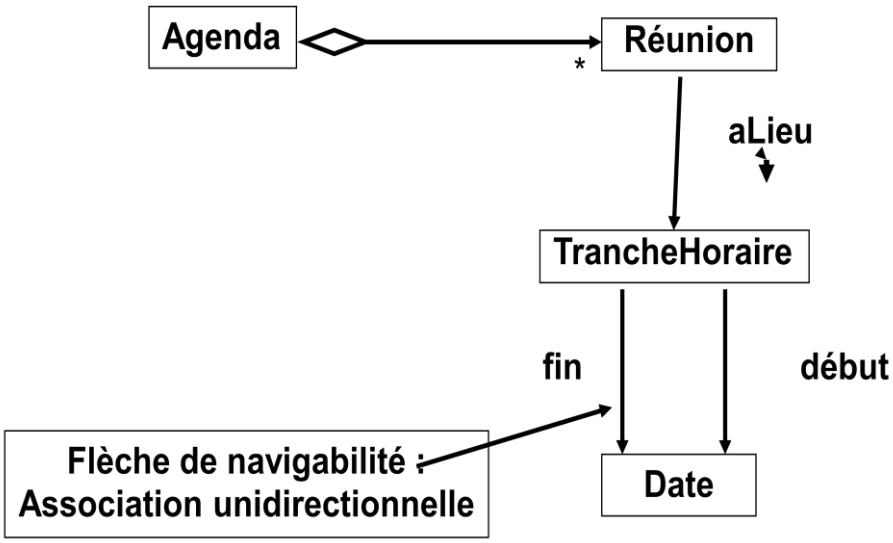
La relation « gère » est-ce que c'est une Association simple ; une Agrégation ou bien une composition ?

Exemple de Composition :



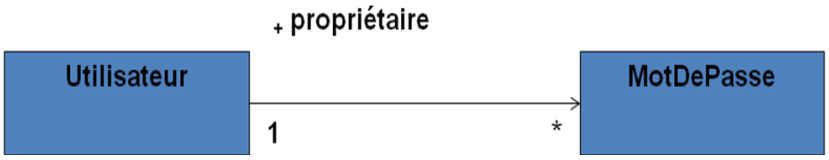
III.A.7 Navigabilité

- ✓ Par défaut une association est bidirectionnelle: à partir d'un objet d'une des 2 classes, on peut atteindre les objets de l'autre classe.
- ✓ Il est possible de réduire la portée en la rendant unidirectionnelle.
- ✓ En général, ce choix se fait dans la phase de conception.

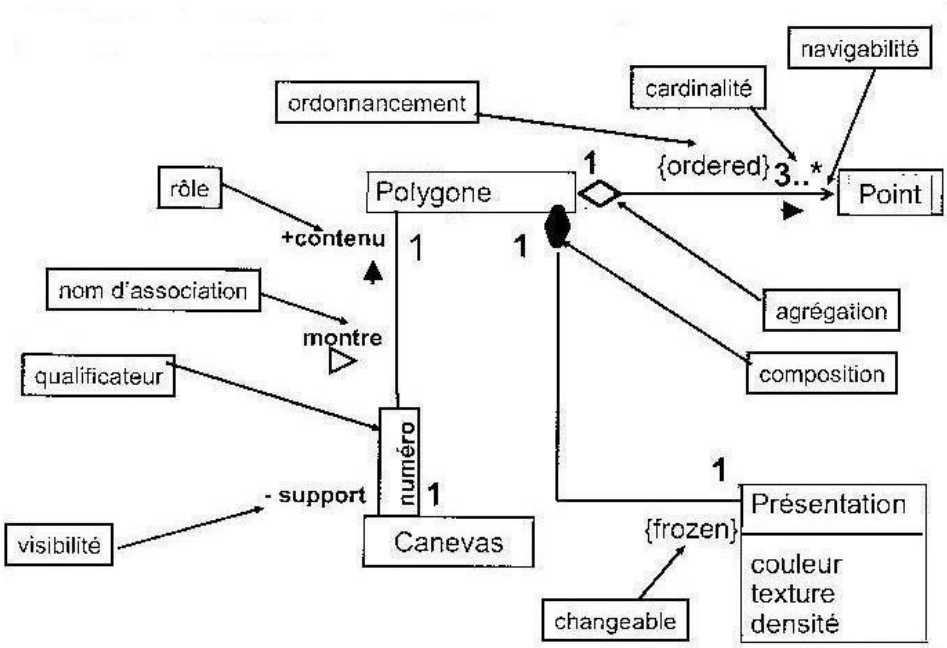


Navigabilité : Exemple

- ✓ Etant donné un utilisateur; on désire pouvoir accéder a ces mots de passe
- ✓ Etant donné un mot de passe on ne souhaite pas accéder à l'utilisateur correspondant



III.A.8 Éléments sur une association : Résumé



Propriétés de mise à jour de liens *:*

- ✓ La contrainte prédéfinie {frozen} interdit l'ajout, la suppression ou la mise à jour des liens d'un objet vers les objets de la classe associée, après l'initialisation du premier.
- ✓ La contrainte prédéfinie {addOnly} autorise l'ajout de nouveaux liens, mais pas leur suppression ni leur mise à jour. A défaut, tout est autorisé.

Ordre sur les liens *:*

Dans le cas d'une multiplicité >1 les liens peuvent être :

- ✓ unordered (c'est la valeur par défaut)
- ✓ ordered
- ✓ sorted : niveau implémentation (ordonnés sur une valeur interne)

III.A.9 Héritage

III.A.9.1 Héritage : Nouvelles classes dérivées de classes existantes

BUT

- ✓ Permettre une réutilisation optimale des classes déjà écrites, utilisées et validées.
- ✓ Réutilisation de la structure des données héritées.
- ✓ Réutilisation du code des services hérités.

PRINCIPE

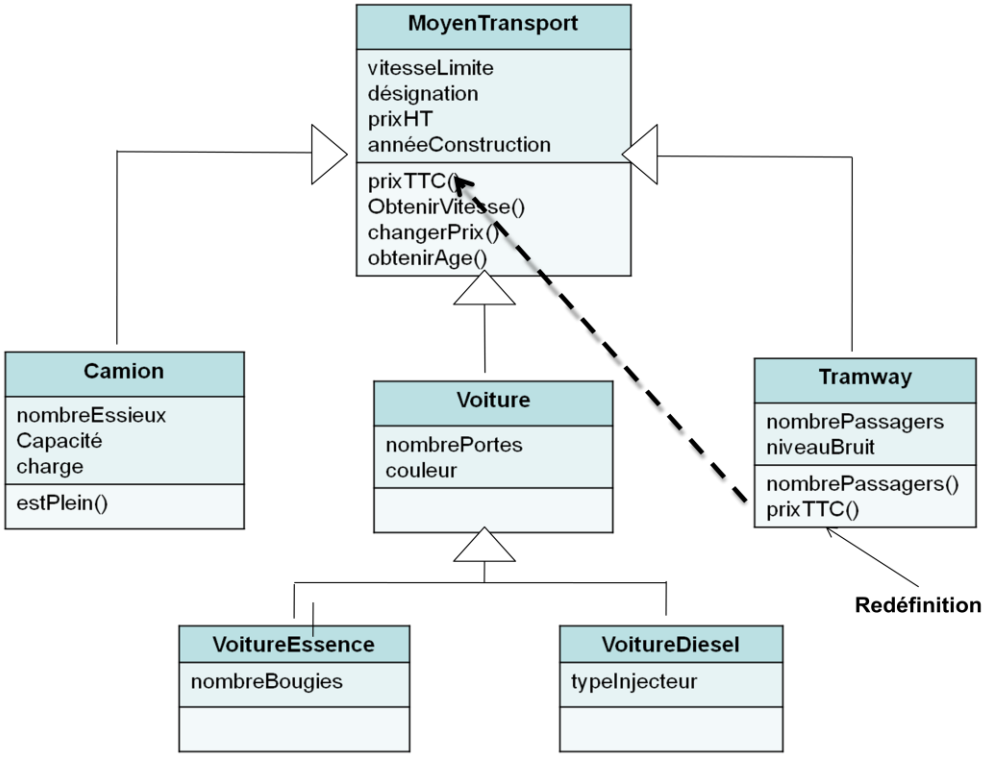
- ✓ Ne pas modifier les classes déjà écrites cela modifierait l'utilisation qui en est faite.
- ✓ Ne pas hésiter à créer des classes, extensions d'autres déjà validées.

III.A.9.1.1 Généralisation / Spécialisation

Relation de spécialisation « **est-un »**, « **is-a-kind-of »** entre classes :

- ✓ La classe spécialisée (sous-classe) hérite des méthodes des attributs et des relations de la classe générale (super-classe).
- ✓ Elle peut ajouter des attributs / méthodes.
- ✓ Elle peut redéfinir le comportement des méthodes (mais pas les attributs).

III.A.9.1.2 Héritage : Adaptation



III.A.9.2 Héritage : Ajout d'une classe de base (analyse)

BUT 2

Permettre une factorisation des caractéristiques et des comportements communs à plusieurs classes.

- ✓ Mise en commun des structures des données.
- ✓ Mise en commun du code des services.

PRINCIPE

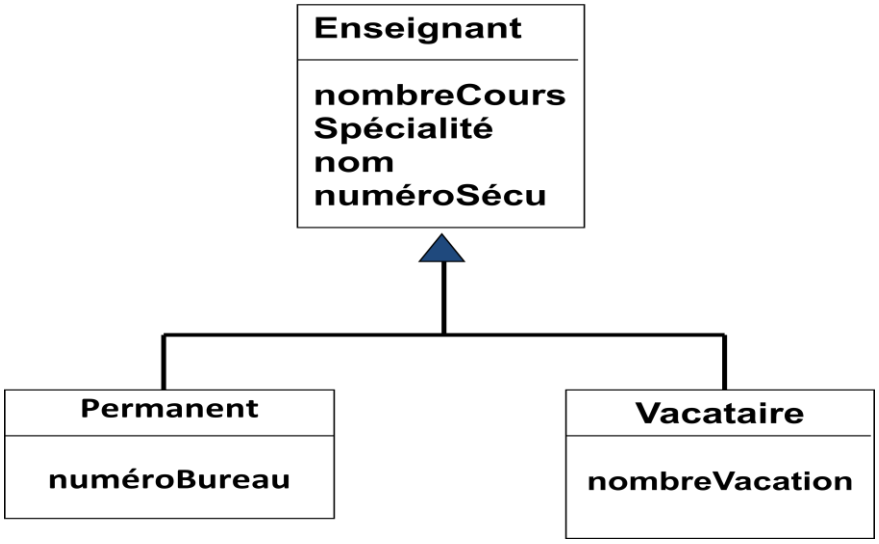
Lorsque plusieurs classes ont des caractéristiques et des comportements communs la création d'une classe ancêtre permet de regrouper ce qui est commun.

Cette classe ancêtre peut correspondre à une classe concrète ou à une classe abstraite.

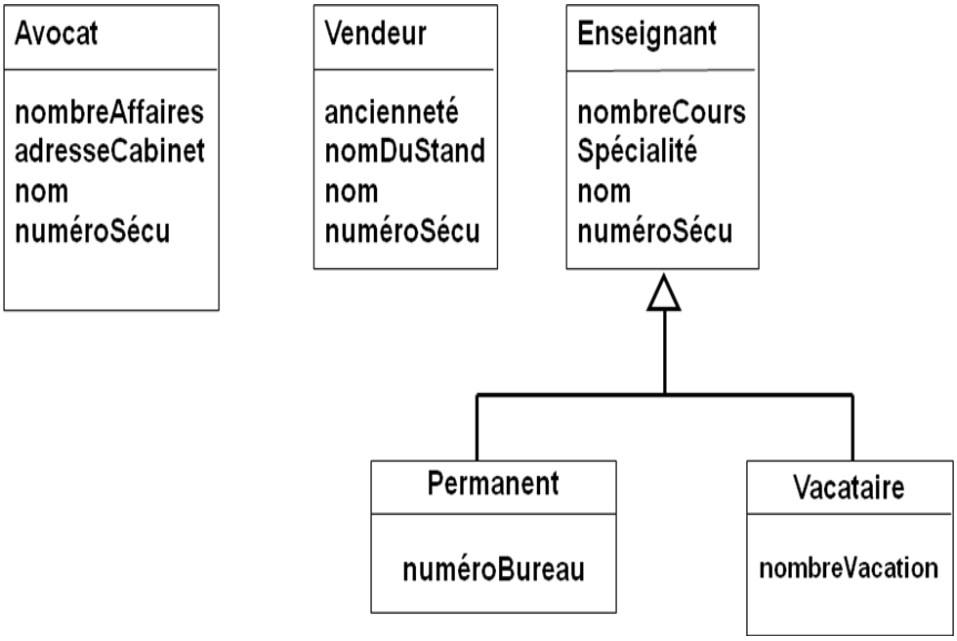
III.A.9.3 Héritage : Généralisation, Factorisation des propriétés

Soit les deux classes suivantes : Permanent et Vacataire

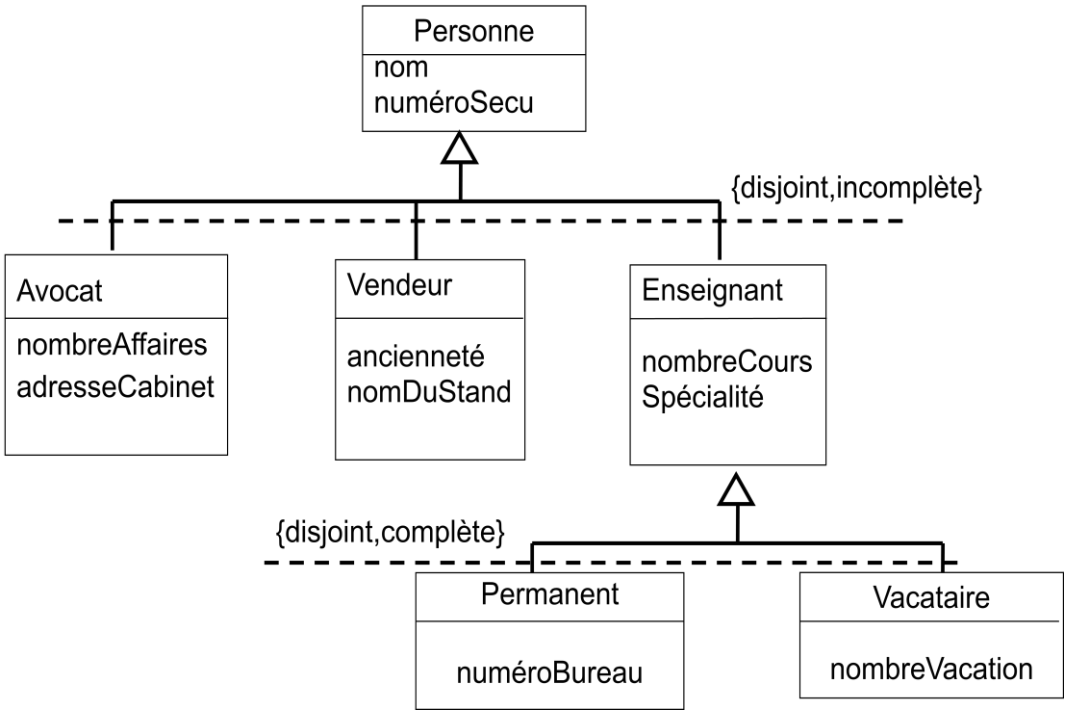
On va factoriser les propriétés de ces deux classes en créant une super classe **Enseignant.**



On suppose qu'il y a d'autre corps métier ; Ex : classes **Avocat, Vendeur** etc.



On va encore factoriser les propriétés communes aux trois classes **Avocat**, **Vendeur** et **Enseignant** en créant une super classe **Personne**.



- La contrainte OCL ***{Complète}*** indique la généralisation est terminée : tout ajout de sous-classe est alors impossible.
- A l'inverse, la contrainte ***{Incomplète}*** indique une généralisation extensible.
- La contrainte ***{Disjoint}*** (ou ***{exclusif}***) indique la participation exclusive d'un objet à l'une des collections spécialisées.

III.A.9.4 Héritage : Polymorphisme

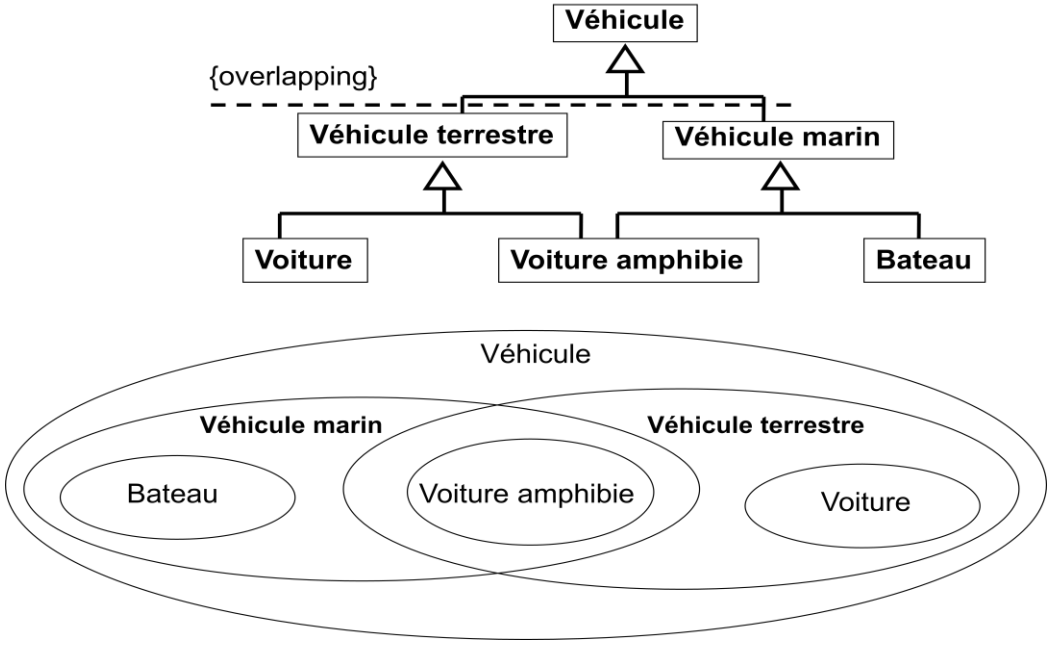
BUT 3

- Créer des sous-types (sous-classes). Une sous-classe sera du même type que la classe dont elle hérite (super-classe).
- Ceci permet de mettre en œuvre le polymorphisme et la liaison dynamique

PRINCIPE

- Un objet d'une classe donnée pourra toujours faire référence à des objets de ses sous classes (polymorphisme).
- Une opération exécutée par un objet sera celle que connaît l'objet dont il fait référence (liaison dynamique).

III.A.9.5 Héritage multiple et répété



- La contrainte OCL ***{overlapping}*** *:* indique que la super-classe possède des instances qui peuvent se recouvrir lors de la réunion de celles de ses sous-classes.
- Une instance (indirecte) de véhicule peut être une instance de véhicule terrestre ou une instance de véhicule aquatique ou encore les deux (soit une instance de véhicule amphibie). Le contraire, lorsqu'il y a une partition exacte des instances par les

sous-classes, se note : ***{disjoint}*.** C'est la contrainte par défaut et qui correspond à un graphe en forme d'arbre.

- La réunion des instances de la classe Voiture avec celles de véhicule amphibie donne les instances de véhicule terrestre.

III.A.10 Classe Abstraite

- C'est une classe **non instanciable** définissant au moins un mécanisme général instanciable par des classes filles.
- Elle contient des opérations non définies :

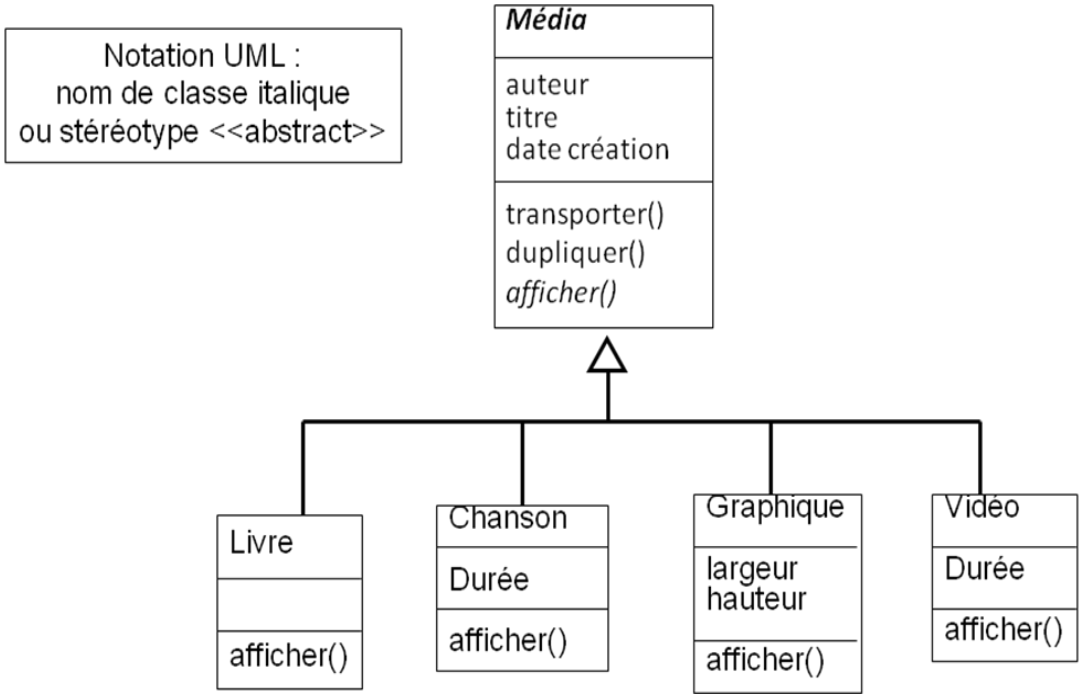
Notation : abstract (java) ; virtual (C++)

- Doit être spécialisée en une ou plusieurs classes non abstraites.

Notation : **stéréotype «abstract»,** ou le nom de la classe est **noté en italique.**

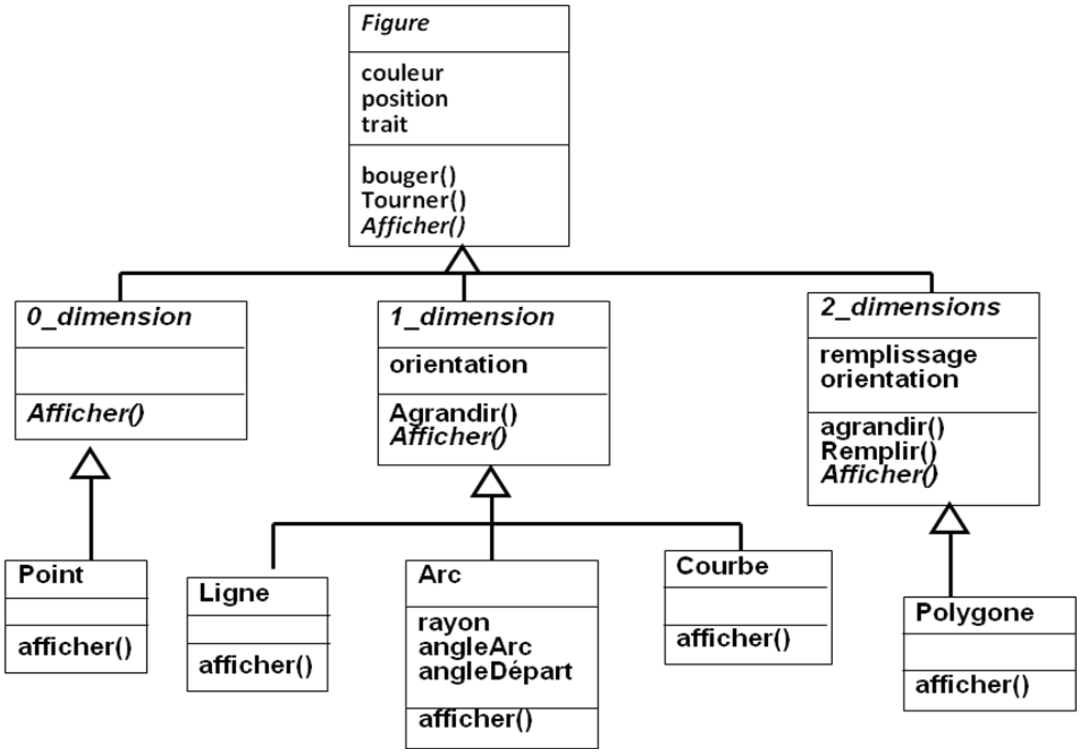
III.A.10.1 Exemple 1 : Classe abstraite Media

- Un média peut être transporté, dupliqué, affiché.
- Le transport et la duplication sont indépendants du type du média (copie de fichiers).
- Par contre, tout média peut être affiché et ce n'est pas la même chose pour l'audio, la vidéo, le graphisme, le texte.
- Un média ne peut pas définir comment s'afficher tant qu'il ne sait pas ce qu'il est.
- Il n'y a pas d'instance de la classe média.
- Un média n'existe qu'en tant que livre, chanson, graphique ou vidéo.



La classe Media est une classe abstraite parce qu'elle définie une méthode abstraite qui est la méthode ***afficher().***

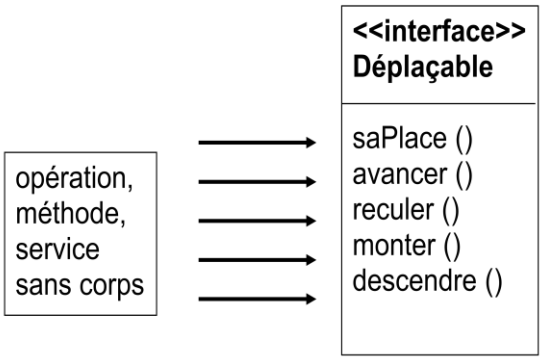
III.A.10.2 Exemple 2 : Classe abstraite Figure



- La classe **Figure** est une classe abstraite parce qu'elle définie une méthode abstraite qui est la méthode ***afficher().***
- Les sous classes **0_dimension, 1_dimension** et **2_dimension** sont aussi des classes abstraites parce que la méthode ***afficher()*** n'est toujours pas définie à ce niveau ; il faut connaitre le type de figure pour pouvoir le faire.

III.A.11 Interfaces

- Une interface est une **classe sans attribut dont toutes les opérations sont abstraites** / virtuelles (non définie).
- Elle ne peut pas être instanciée.
- Elle doit être réalisée (implémentée) par des classes non abstraites.
- Elle peut hériter d'une autre interface.
- Une **interface** permet de décrire le **comportement** d'une entité (classe, paquetage ou composant), c'est à dire un savoir faire sous la forme d'une **liste d'opérations**.
- Une interface ne peut donner lieu à **aucune implémentation**.
- **Une classe** peut déclarer qu'elle **implémente une interface**. Elle doit alors implémenter toutes les opérations de cette interface. Elle peut ensuite être utilisée partout où ce comportement est exigé.



Une interface n'est PAS une classe
C'est une liste de services

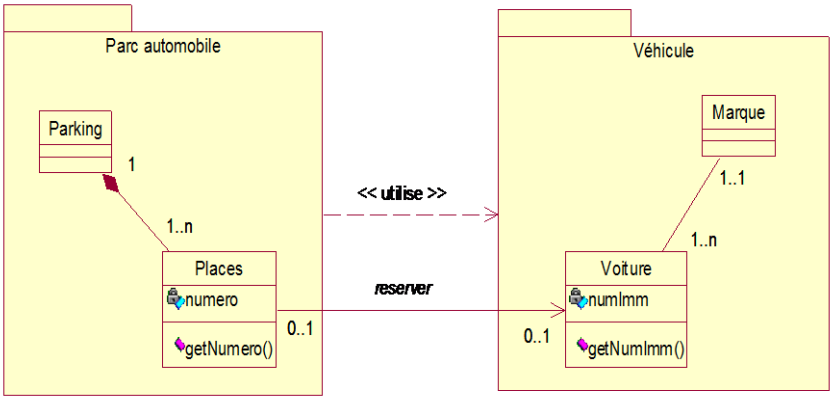
Elle ne peut pas servir à créer un objet

Une interface exprime un savoir faire

TYPE = CLASSE + INTERFACE

III.A.12 Paquetages

- Une application est constituée de plusieurs lasses, des dizaines ou des centaines. Il est important de les organiser en groupes (en fonction de certains critères surtout logiques).
- C'est le paquetage (package) qui permet ce regroupement.
- Un paquetage regroupe des classes, des interfaces, des paquetages.
- Il permet d'encapsuler certains éléments de la modélisation. Un élément du paquetage peut être inaccessible de l'extérieur du paquetage, il n'est alors connu que par les éléments du même paquetage.
- Il met en œuvre un espace de nommage.



Véhicule : : Voiture signifie que la classe Voiture appartient au paquetage Véhicule.

.

SECTION B

DIAGRAMME DE CLASSES DE LA GESTION DE BIBLIOTHEQUE

-

Recherche à partir du Cahier des Charges Recherche par responsabilité

"Au cœur de toute difficulté se cache une possibilité "

Albert Einstein

III.B.1 Phases de la modélisation Objet

- Identifier les classes candidates.
- Préparer le dictionnaire de données : classes retenues.
- Identifier les associations entre classes (en incluant les agrégations).
- Identifier les attributs.
- Organiser et simplifier les classes en utilisant l'héritage.
- Supprimer les associations inutiles
- Vérifier que le diagramme inclut toutes les demandes du cahier des charges.
- Itérer et affiner le modèle.
- Grouper les classes en modules (paquetages).

III.B.1.1 Identifier les Classes : Classes Candidates

- Un gérant de bibliothèque désire automatiser la gestion des prêts.
- Il commande un logiciel permettant aux utilisateurs de connaître les livres présents, d'en réserver jusqu'à 2. L'adhérent peut connaître la liste des livres qu'il a empruntés ou réservés.
- L'adhérent possède un mot de passe qui lui est donné à son inscription.
- L'emprunt est toujours réalisé par les employés qui travaillent à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre).
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts réalisés dans la bibliothèque

Les classes candidates sont :

gérant	bibliothèque	gestion
prêts	logiciel	utilisateur
livres	adhérent	liste
mot de passe	inscription	emprunt
employés	emprunteur	ensemble

III.B.1.2 Classes retenues

- gérant non pertinente, n'intervient pas.
- *bibliothèque oui* responsabilité : gérer les livres, adhérents, prêts.
- gestion non vague.
- *prêts oui* responsabilité : contenir les infos et actions sur les prêts.
- logiciel non vague.
- utilisateurs choix entre utilisateur, adhérent, emprunteur.
- *livres oui* responsabilité : permettre de connaître son état.
- *adhérent oui* responsabilité : permettre à la personne d'être identifiée .
- liste non implémentation ou conception.
- mot de passe non attribut.
- Inscription non action.
- emprunt non action.
- *employés oui* responsabilité : reconnaître qui a fait un prêt, etc.
- emprunteur choix entre utilisateur, adhérent, emprunteur.
- ensemble non implémentation ou conception.

III.B.1.3 Dictionnaire des Données

- bibliothèque : organisme gérant une collection de livres qui peuvent être empruntés par ses adhérents. Une bibliothèque est gérée par ses employés.
- prêt : un prêt est caractérisé par le numéro du livre, la date, la durée. Il ne peut être fait que par un adhérent.
- livre : ouvrage pouvant être emprunté.
- adhérent : personne inscrite à la bibliothèque.
- employé : personne travaillant à la bibliothèque.

III.B.1.4 Chercher les Associations

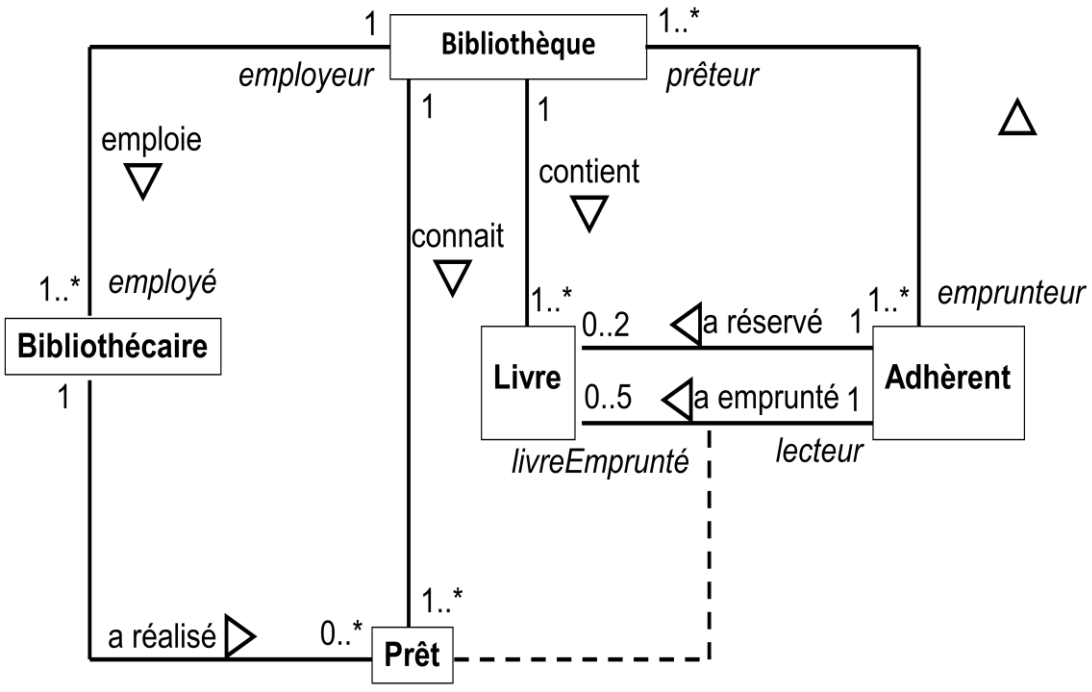
- Un gérant de bibliothèque désire automatiser la gestion des prêts.
- Il commande un logiciel permettant aux utilisateurs de *connaître* les livres présents, d'en *réserver* jusqu'à 2. L'adhérent peut connaître la liste des livres qu'il *a* *empruntés* ou *réservés*.

- L'adhérent possède un mot de passe qui lui est donné à son inscription.
- L'emprunt *est* toujours *réalisé* par les employés qui *travaillent* à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre).
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts *réalisés* dans la bibliothèque

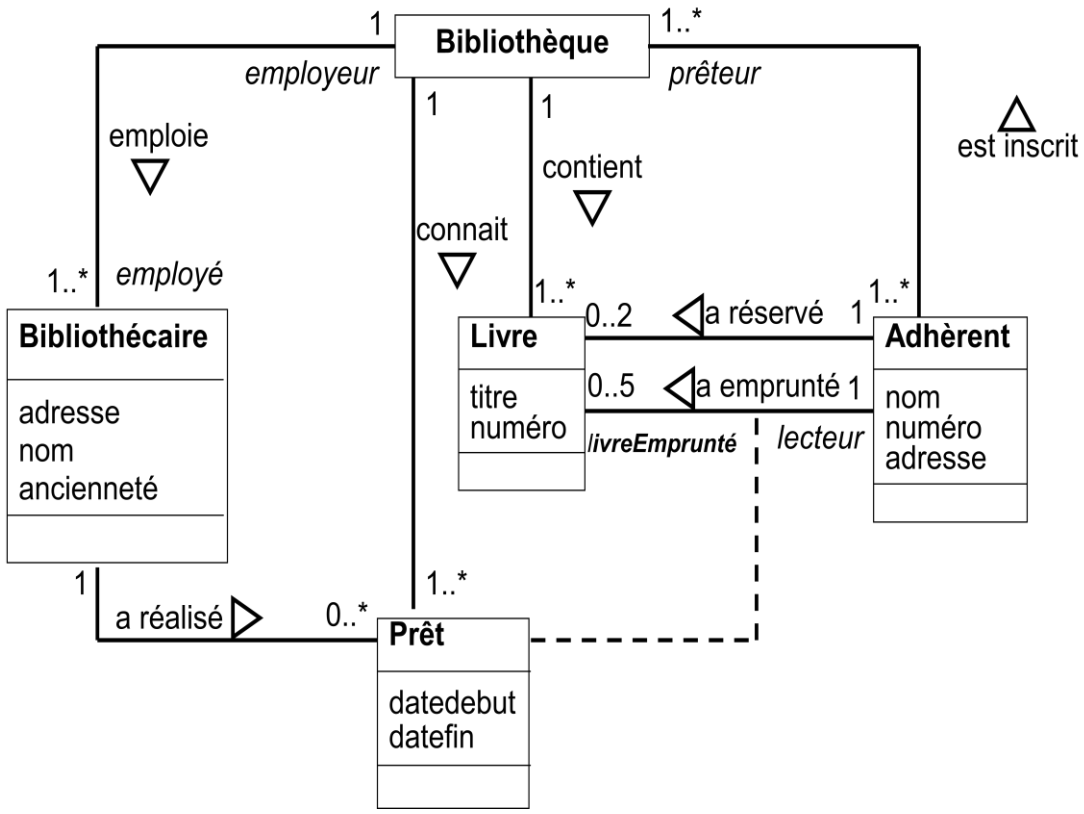
Associations sous entendues :

- Un adhérent *est inscrit* à la bibliothèque.
- La bibliothèque *contient* des livres.

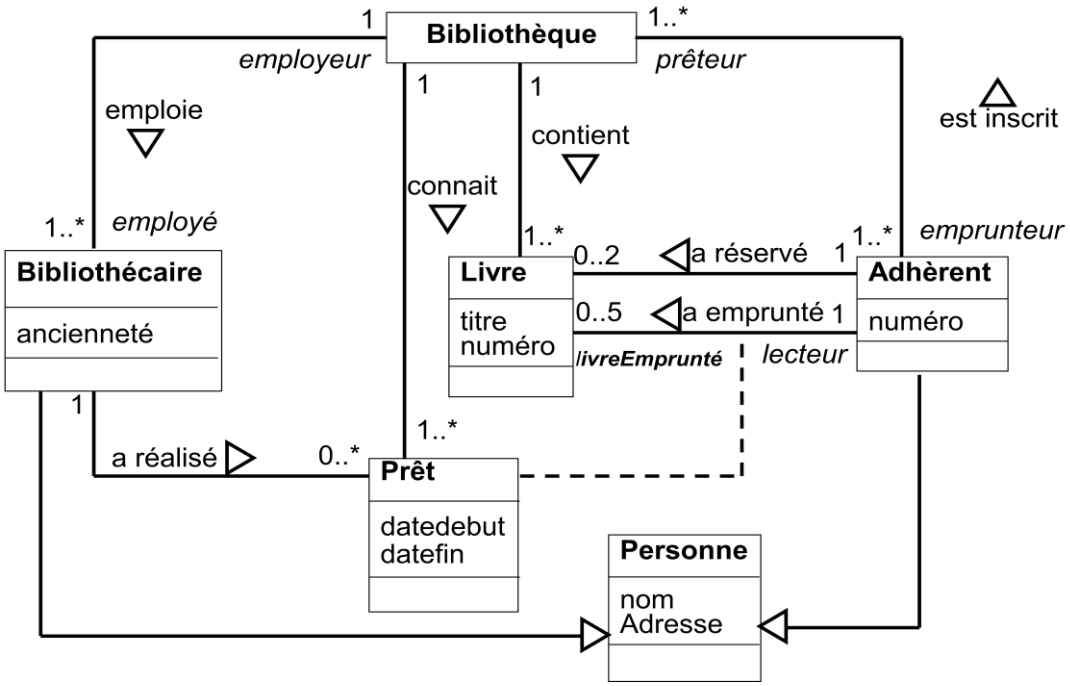
Associations :



III.B.1.5 Chercher les Attributs



III.B.1.6 Généraliser par Héritage



SECTION C

DIAGRAMME D'OBJETS

-

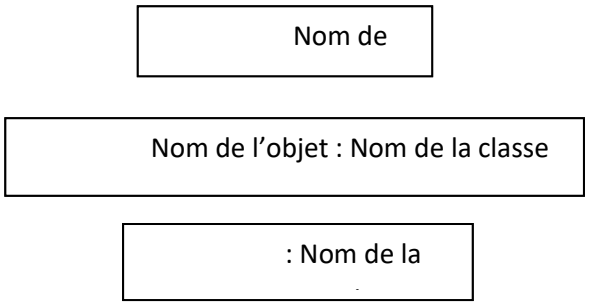
OBJECT DIAGRAM

" Un problème ne peut être résolu en réfléchissant de la même manière qu'il a été créé "

Albert Einstein

III.C.1 Diagramme d'Objets : Définition

- Un diagramme d'objets représente les liens structurels entre instances des classes.
- Il facilite la compréhension de structures complexes.
- Trois représentations possibles des instances.



- Les **valeurs des attributs** sont optionnelles ainsi que les liens entre objets.

Diagramme de Classe

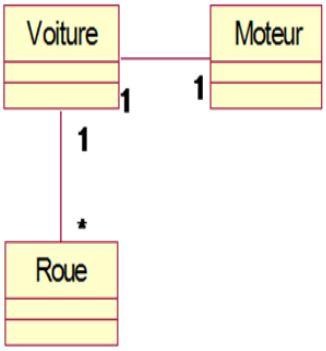
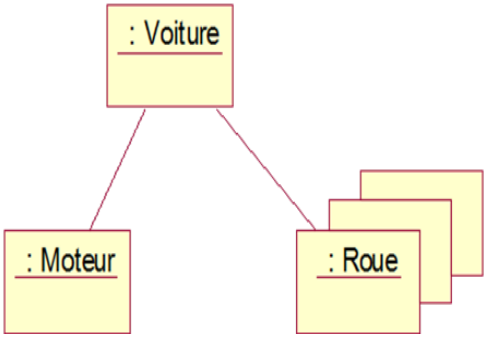
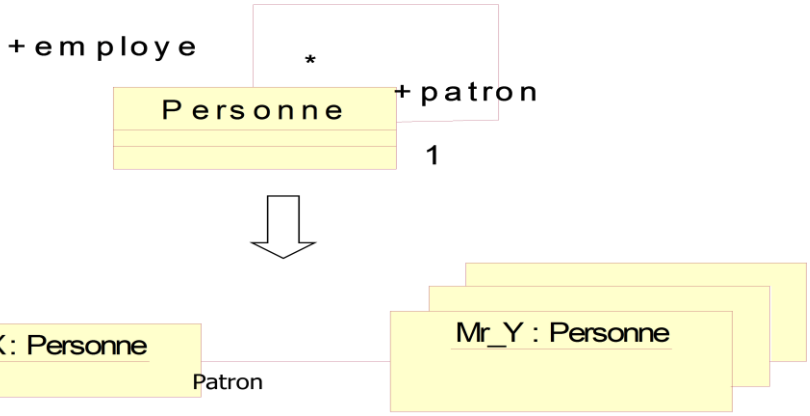


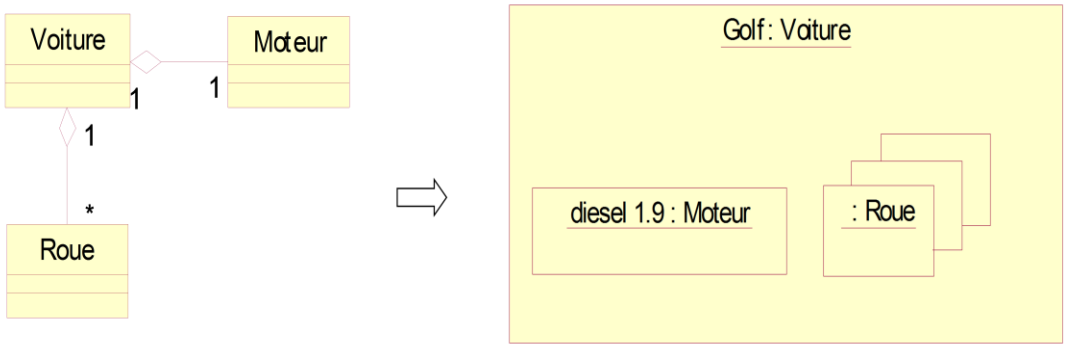
Diagramme d'Objet



- Les **liens**, instances des associations **réflexives** peuvent relier un objet à lui-même.



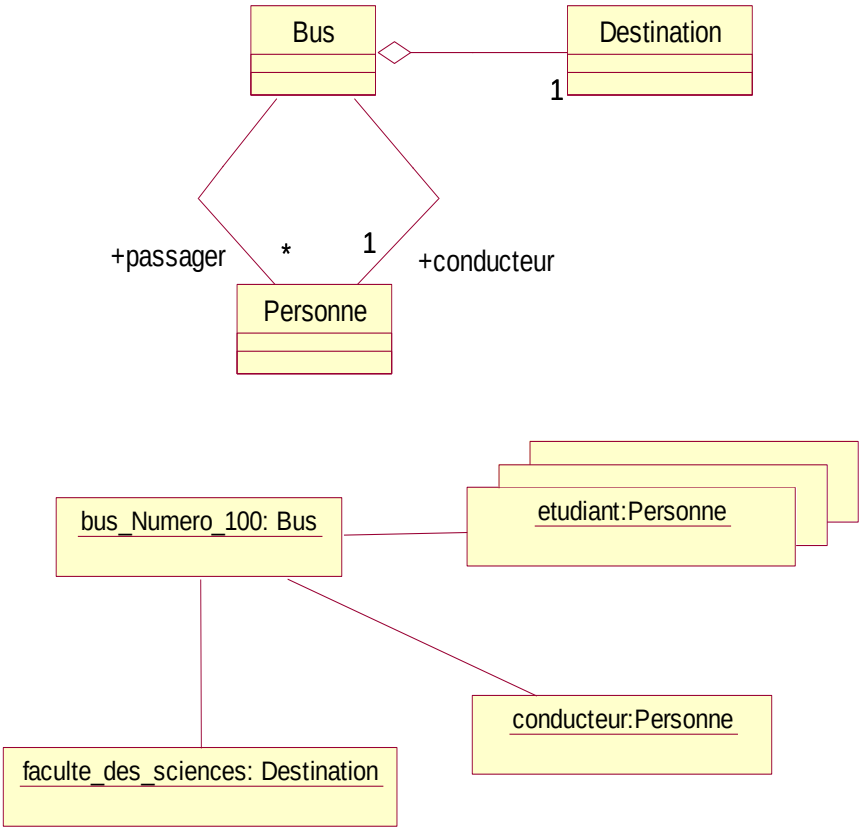
- Les objets composés de sous-objets peuvent être visualisés.



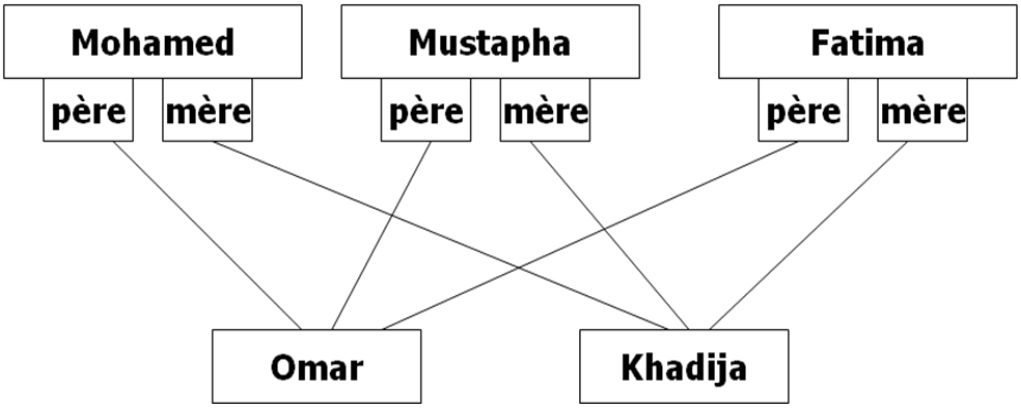
III.C.2 Diagramme d'Objets: Structures complexes

- Les diagrammes d'objets facilitent la compréhension et l'élaboration d'un diagramme de classes.

Exemple1 :



Exemple 2 :



III.C. 3 Résumé