Une introduction à la conception objet avec UML et Java

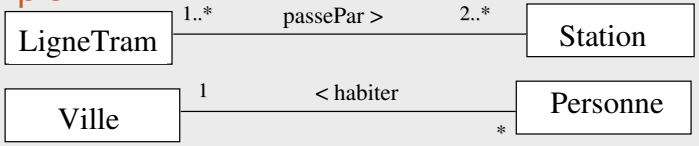
Chapitre 3 : Les relations

- Associations
- Orientation
- Agrégation et composition
- Dépendances

Multiplicités

- **Multiplicités** (ou cardinalités) :
  - Spécifie le nombre potentiel d'objets reliés à un objet donné (cf Merise)
  - Intervalle de la forme : i..j,k..l

    où i,j,k,l sont des entiers, * signifie « plusieurs »
    - ➔ ex 1..* signifie « un à plusieurs »
    - ➔ Raccourcis : 1 signifie un et un seulement ; * signifie 0 à plusieurs
- **Exemple :**



RQ 1 : Cardinalités en sens inverse de Merise !

RQ 2 : Le > est facultatif ; il indique le sens de lecture de la relation

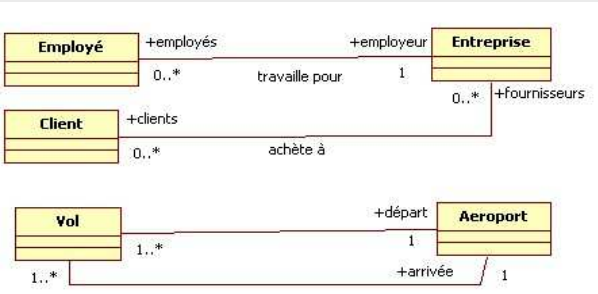
Les relations

- **Association ou relations**
  - Relation sémantique stable entre classes mettant en oeuvre des rôles.
  - Représente l'ensemble des liens entre les occurrences des classes participant à l'association
- **Degré d'une association**
  - Nombre de classes qui participent à l'association
  - Degré = (1..4)

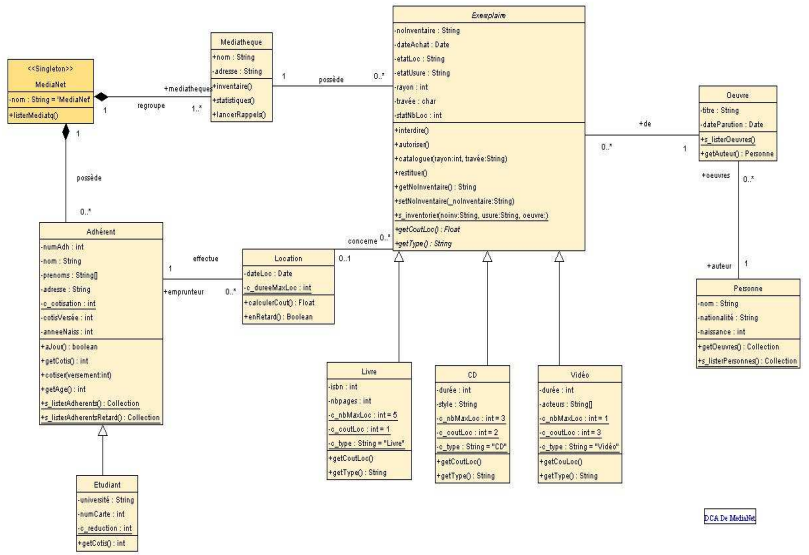
Comme pour le MCD, une association ne doit pas être modélisée par un attribut (--> implémentation)

Rôle

- Le "Rôle" est indiqué sur une extrémité de l 'association
- Il décrit la participation d'une classe dans l'association
  - ➔Facilite la lecture et la compréhension du modèle
  - ➔Important pour la sémantique du modèle et pour la génération de code POO
  - ➔Le nom du rôle est généralement différent du nom de l'association, il est implicitement identique au nom de la classe



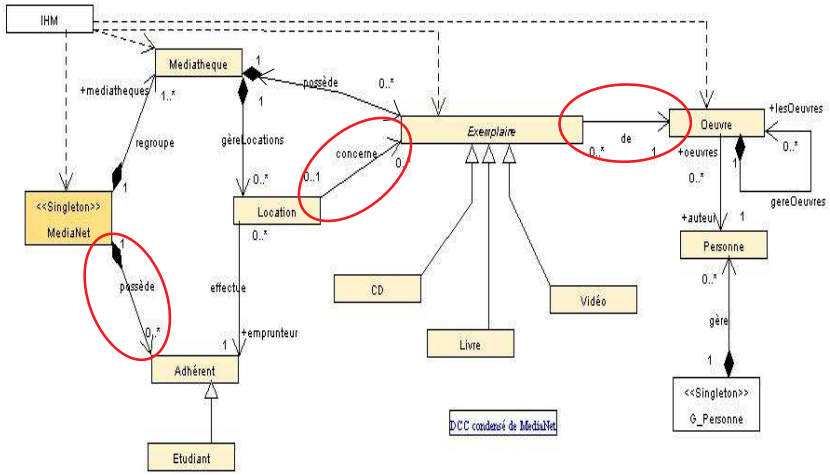
Exemple Medianet



Orientation

- Orientation des relations (démarche de conception)
  - Les relations peuvent etre orientées en fonction de leur dépendance mutuelle. Ex A---> B si :
    - ➔ Le concept de A s 'appuie sur celui de B
      Ex : location --> exemplaire
    - ➔ Le parcours du lien de A vers B est plus fréquent que l 'inverse
      Ex : Exemplaire --> Oeuvre
    - ➔ A est un agrégat (cf suite) composé d 'objets B
      Ex : MediaNet --> Mediathèque
  - NB : Le parcours inverse (B-->A) n 'est pas nécessairement impossible mais il est plus long
  - L'orientation introduit une dépendance : La classe A nécessite la connaissance de la classe B (importance pour le découpage en packages)

Orientation



Orientation

- La restriction du parcours peut amener le concepteur à modifier et à déplacer certaines méthodes
- Ex : get_oeuvres() (« lister les oeuvres écrites par une personne donnée ») était dans « Personne » avant l 'orientation ; après orientation, il doit migrer dans « Oeuvre » : get_oeuvres(p:Personne)

Orientation

- Si 2 concepts sont intimement liés, les 2 classes seront dans le même package et la relation peut être bi-orientée

  A <-->B (ou A --- B)

  NB 1 : Attention, trop de relation bi-orientées nuisent au découpage en packages et donc à la réutilisation

  NB 2 : UML représente les relations bi-orientées (ex : « possède »), sans flèches ; dans le cadre de ce cours on l'évitera :

  `Pas de flèche => Relation non encore orientée`

Génération du code

➢ Génération du code

- La modélisation UML, quand elle est faite avec un atelier de génie logiciel (AGL), permet de décliner rapidement le modèle de classe en un programme objet

  Selon les AGL, la quantité (et la qualité) de code généré est très variable

Génération du code

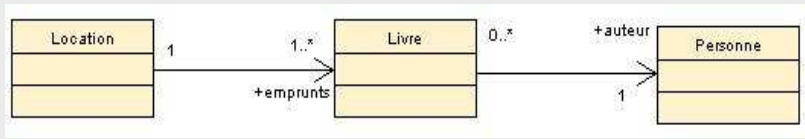
➢ L'orientation des relations influe sur le code généré :

- Une relation orientée de A vers B, se traduit par une référence (ou une liste de références) de la classe B dans la classe A

- Selon les AGL, plusieurs méthodes d'accès aux occurrences reliées sont ajoutées

Génération du code

➢ Exemple



- Dans «Livre » on trouvera comme attribut :

  ```
  public Personne auteur ;
  ```

- Dans «Location» on trouvera comme attribut :

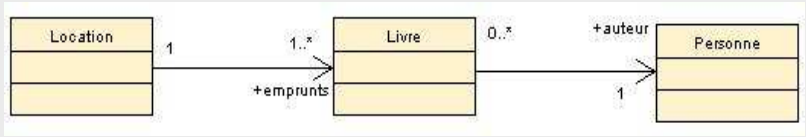
  ```
  public Vector<Livre> emprunts = new Vector<Livre>();
  ```

NB : Le nom de l'attribut est le nom du rôle de la classe cible dans l'association (ici « auteur »)

NB : Dans le cas d'une relation bi-orientée, il y a redondance puisqu'on crée des références des 2 côtés de la relation ; prendre garde, lors de chaque modification, à la mise à jour des références de chaque côté (AGL)

Génération du code

Exemple

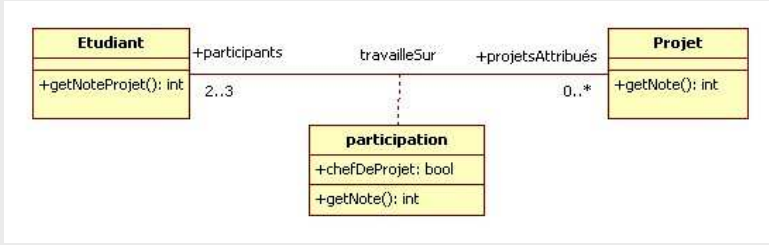


- *Dans l 'AGL, il faut choisir la structure de parcours :* `Vector, ArrayList, TreeSet...`
- *L 'AGL ajoute les méthodes d 'accès à la classe reliée*
  - `Personne getAuteur() ; Vector<Livre> getEmprunts();`
  - `ajouter, supprimer un emprunt, compter les emprunts...`
  - `Ajouter un élément à la position i`
  - `Supprimer le ième élément`
  - `...`

Autres associations

Classes /Associations

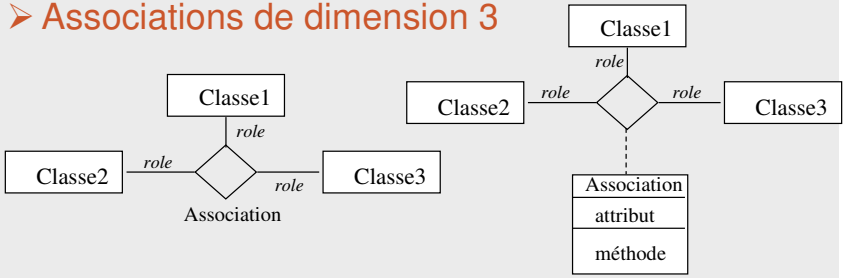
- Relations de plusieurs à plusieurs porteuses de données



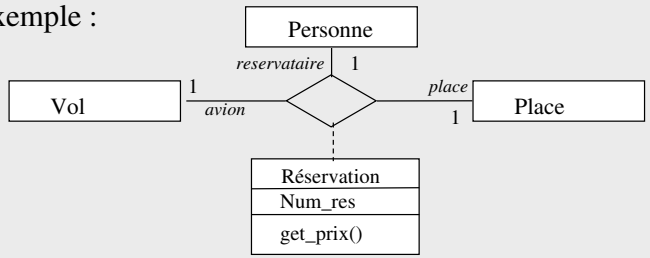
- Question : A quoi correspondent les 3 getNotes() ?

Autres associations

Associations de dimension 3

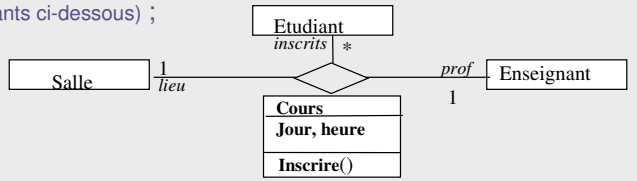


Exemple :

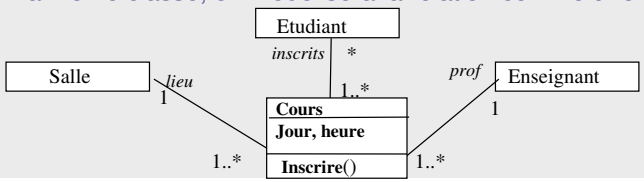


Autres associations

- RQ : On trouve quelque fois, dans certains modèles UML, des multiplicités sur les arcs pour signifier le nombre d 'objets impliqués dans une occurrence de l 'association (comme les étudiants ci-dessous) ;

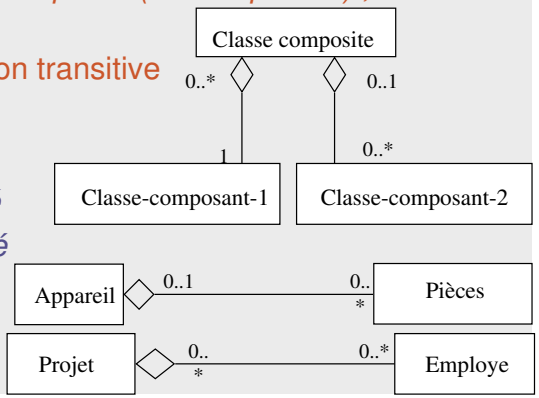


- Notez bien que, dans le cadre de cette formation, on limitera l 'utilisation des associations n-aires à des cardinalités 1 et si une occurrence de l 'association fait intervenir plusieurs objets de la même classe, on modélisera la relation comme une classe.



Agrégation/Composition

- L'agrégation est une association entre 2 classes, possédant une sémantique particulière : «*se compose de*» et «*fait partie de*»
- Une classe représente *un tout (l'agrégat, le composite)* ; l'autre représente *une partie (un composant) ; Ex : la nomenclature*
- C'est une association transitive
- Ex :
  - *Appareil/Pièces*
  - *Equipe/Employé*
  - *Service/Employé*
  - *Chambre/mur*



Agrégation/Composition

- Une agrégation peut vérifier ou non ces propriétés :
  - Partageable : Une partie peut être partagée simultanément par plusieurs « tout »
    - ➔*ex. : projet - employés* (cardinalité max = *)
      Non partageable => card max = 1
  - Isolable : Une partie peut exister seule sans être rattachée à un « tout »
    - ➔*ex. : Appareil - Pièces* (cardinalité min = 0)
      Non isolable => card min = 1
  - Amovible : Une partie peut changer de « tout » au cours de son cycle de vie
    - ➔*ex. : Service - Employés*

Agrégation/Composition

- Restriction de la notation UML :
  - Dans le cadre de ce cours on appliquera la restriction suivante (qui n'est pas imposée par UML) :
    - ➔Une association partageable sera représentée par une simple relation
    - ➔L 'agrégation et la composition (cf suite) seront réservées à des associations non-partageables
  - Dans les exemples précédents :
    - ➔ « appareil-pièces » reste une agrégation (isolable)
    - ➔ « projet-employé » devient une simple relation (plus de losange)
    - ➔ « employé-service » reste une agrégation (non isolable)

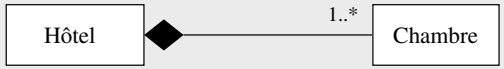
Agrégation/Composition

- Une **composition** est une agrégation non partageable, non isolable et non amovible
  - Multiplicité 1..1 et toujours la même classe composite reliée tout au long du cycle de vie (CIR)
  - Inclusion des cycles de vie : La partie de peut pas pré-exister ni survivre au « tout »

  - Suppression du tout => suppression des parties
  - La classe composite gère l'accès aux composants

Agrégation/Composition

- Exemples :
  - *Fenêtre/ascenseur (informatiques)*
  - *entreprise/services*
  - *immeuble/appartements ou hôtel/chambres*
- Notation :



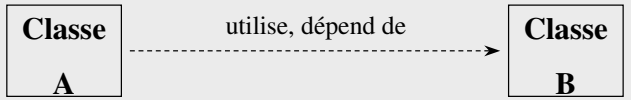
NB : La cardinalité coté agrégat n'est généralement pas représentée car elle est obligatoirement 1..1

« Un, un seul et toujours le même »

Liens de dépendance

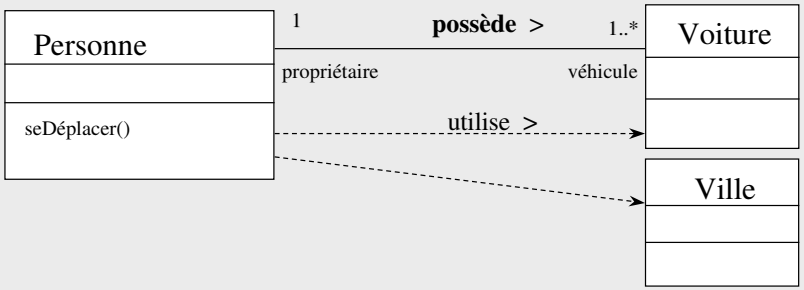
Les relations, les compositions, les héritages mettent en œuvre des liens forts, structurels entre les classes.

- Le lien de dépendance est un lien orienté plus fugace ou plus variable qui indique qu'une classe (A) en utilise une autre (B) et donc qu'un changement de spécification de B peut affecter A, l'inverse n'étant pas nécessairement vrai
  - Importance pour les packages et la réutilisation



Liens de dépendance

- Exemple



➔Le lien de dépendance manifeste l'utilisation de la classe cible dans les arguments ou le corps d'une méthode de la classe source

seDéplacer (V1 : Ville ,V2 : Ville , V3 : Voiture)