Modélisation Conception de logiciel / Développement Agile

Chapitre 2-1 – Rappels UML
Diagramme de Classes et d'Objets

Véronique DESLANDRES
Licence DEVOPS, IUT de LYON

Plan de ce cours

- **Les diagrammes de classes (DCL)**
  - Mécanismes généraux
  - Les relations
  - Les propriétés
  - Un ex. d'implémentation (s36)
  - 2 Exercices (s44)
  - Fiche Je retiens… (s47)
- **Diagrammes d'objets --- 48**

Pré requis

- Notions de base d'UML acquises
  - Diagrammes de classes, des cas d'utilisation, de séquences
  - Ici ce ne sont que des rappels

- Si nécessaire : un cours complet avec l'exemple de l'aviation, par construction progressive (à partir de s55):
  https://fr.slideshare.net/MireilleBF/uml-classes-par-les-exemples

- Vidéos courtes :
  - **Rappels sur le DCL** : https://www.youtube.com/watch?v=8PmTJIrlS5w avec une exercice d'illustration (11')
  - **Un exercice commenté** qui rappelle les notions du DCL (jusqu'à 8'42)
    https://www.youtube.com/watch?v=YlcRl07eHXk

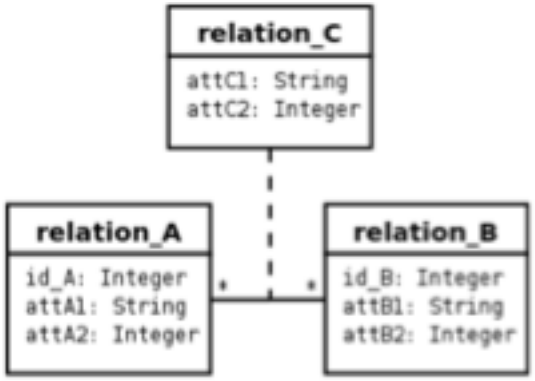
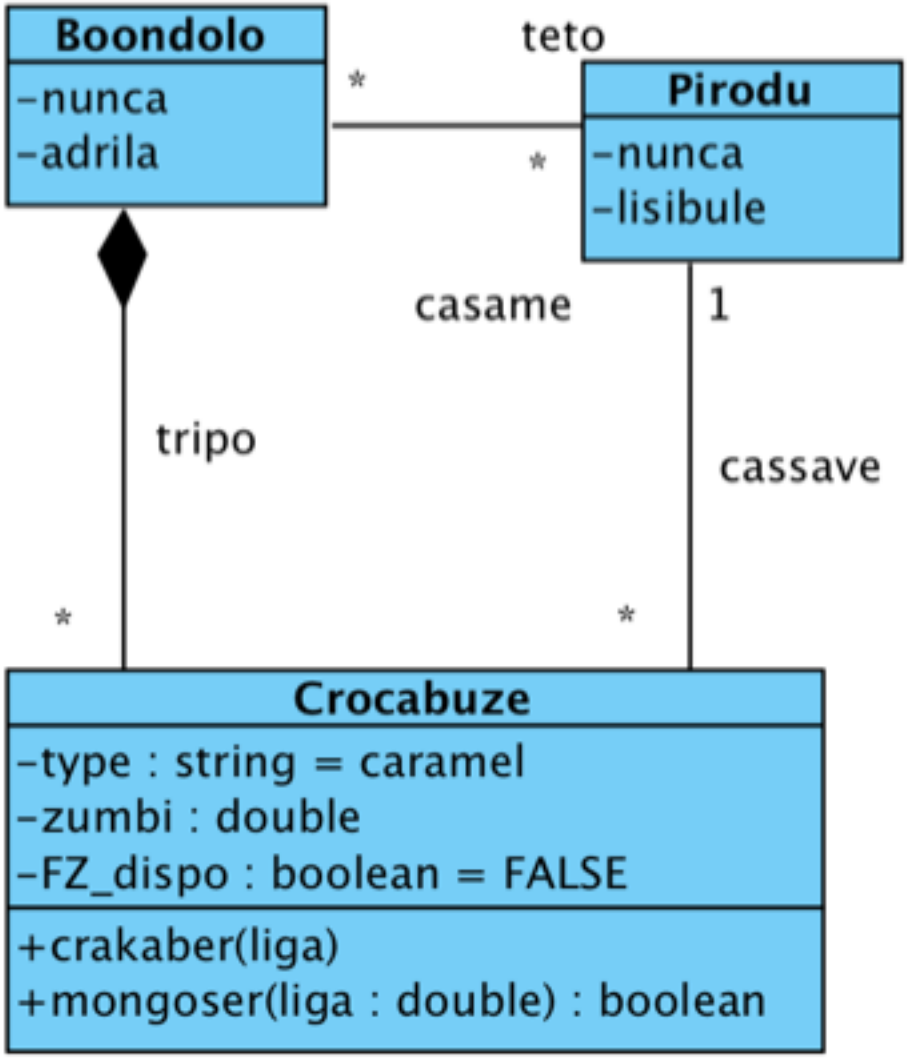
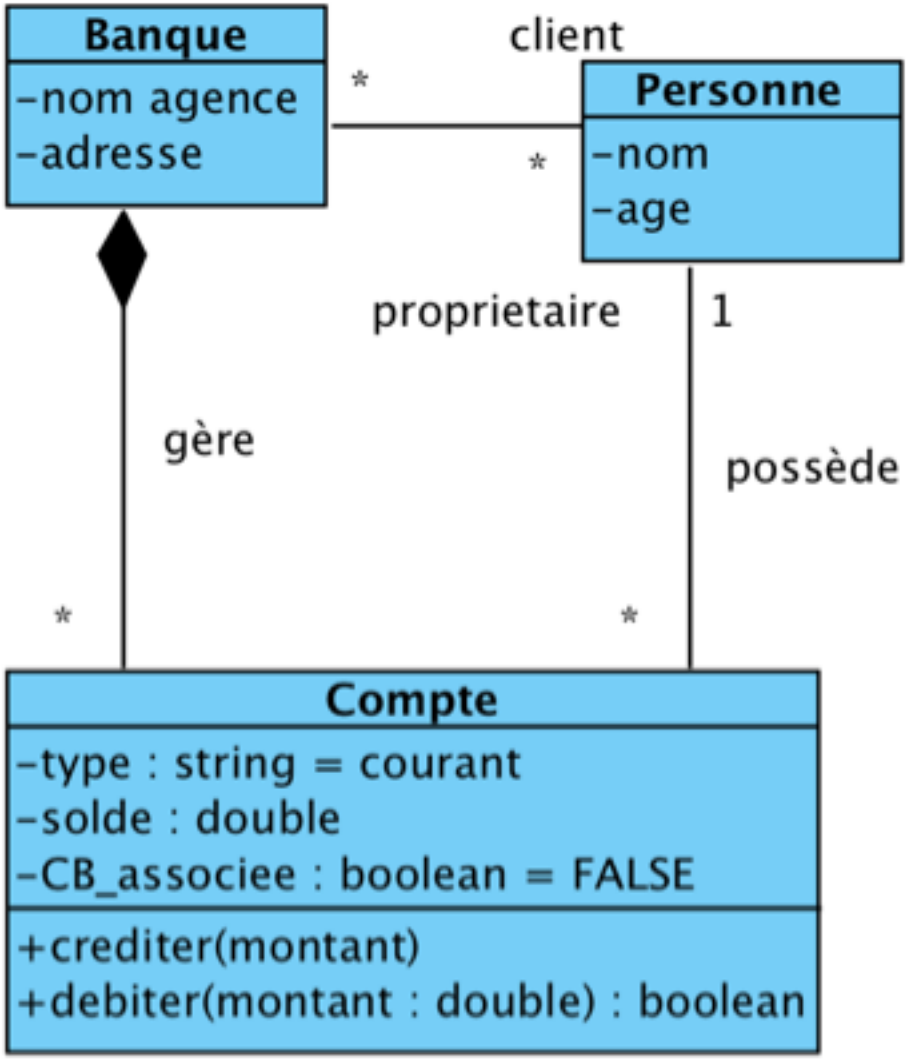


Diagramme de Classes

Que raconte ce diagramme ?



Ça va mieux ?


Banque
-nom agence
-adresse
client
*
*
Personne
-nom
-age
proprietaire
1
gère
possède
*
*
Compte
-type : string = courant
-solde : double
-CB_associee : boolean = FALSE
+crediter(montant)
+debiter(montant : double) : boolean

Un formalisme UML extensible

stéréotype
de classe

Étiquette
(tagged value)

«container»
FileAction
{Version = 3.2}

{fontionne en temps O(1)}

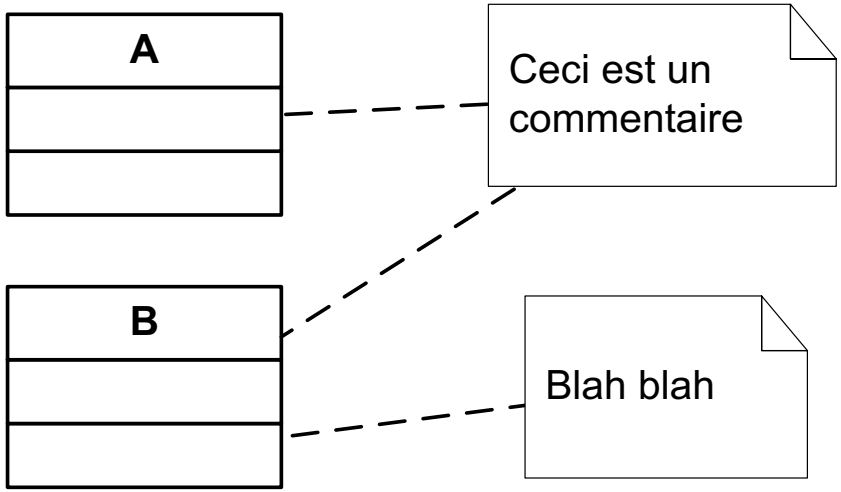
+ajouter(){sequential}
+enlever(){sequential}
«requête» +longueur(){sequential}
«organisation» +réorganiser(){sequential}

contrainte

stéréotype
d'opération

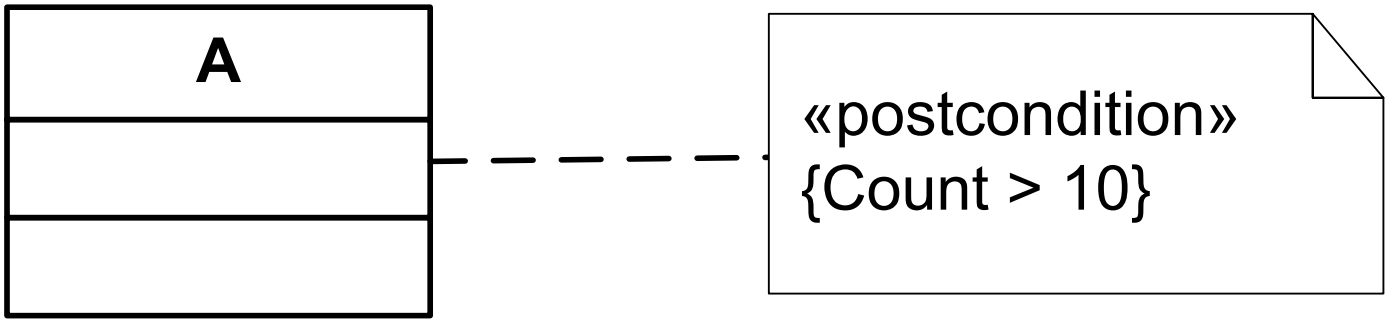
Les notes

- **Commentaire attaché à un ou plusieurs éléments de modélisation**
  - Appartient à la vue, pas au modèle
  - Peut être stéréotypée en contrainte : cf ci-après



Les contraintes

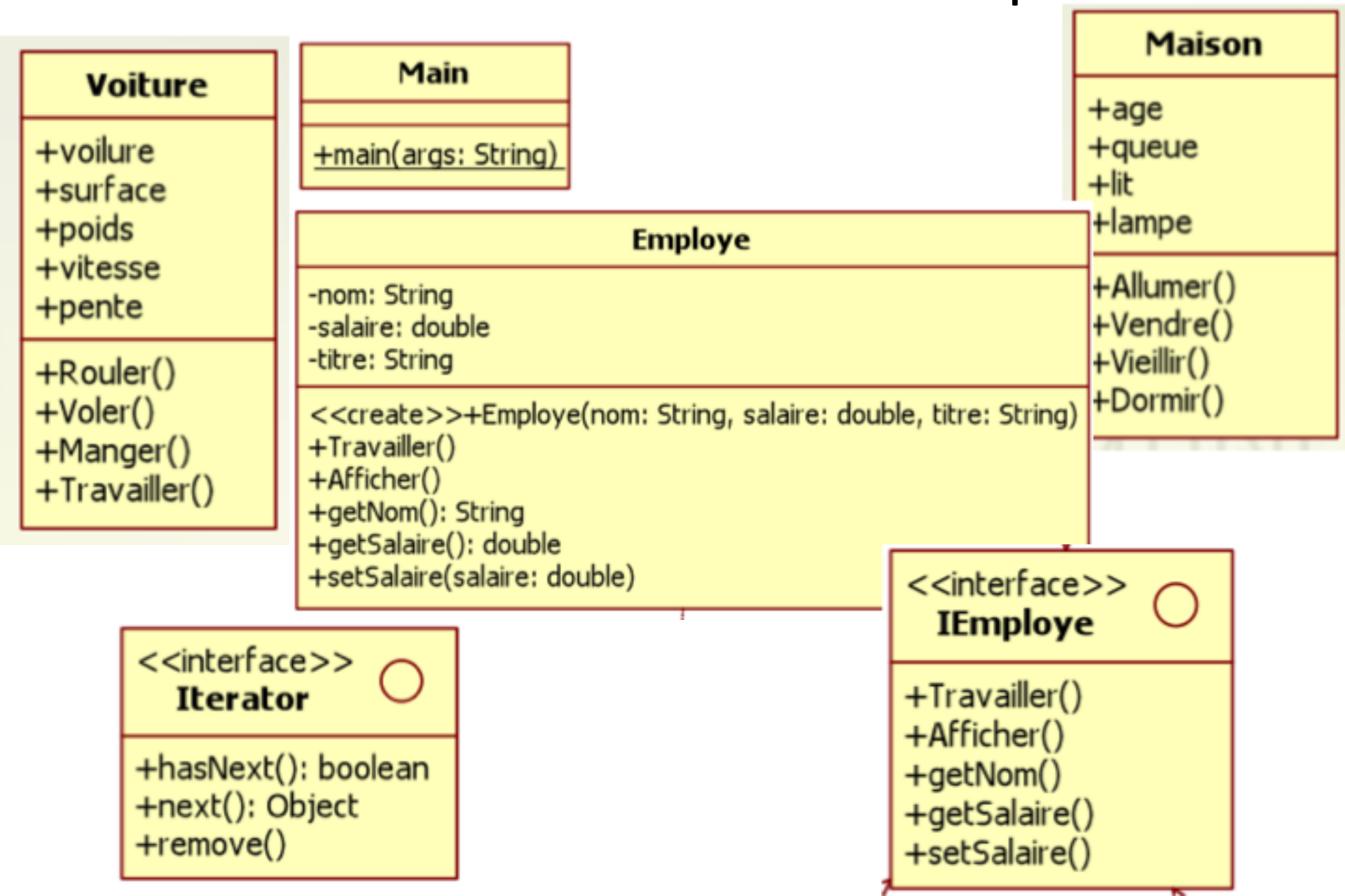
- Relation sémantique quelconque entre éléments de modélisation
- Exprimée en OCL (Object Constraint Language) ou en langage naturel
  - {contrainte}, inv, pre/post-condition
  - Utilisée dans certains AGL pour la génération de code



Utilisation DCL

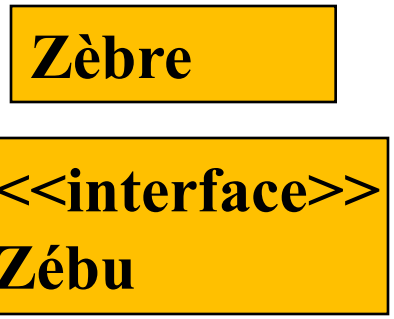
- Très nombreuses utilisations, à différents niveaux :
  - Pendant la capture des besoins :
    - **Modèle du domaine**
    - Classes correspondant à des objets du domaine
  - Pendant la conception / implémentation :
    - **Modèle de conception**
    - Classes correspondant à des objets logiciels

Classes du Domaine vs. Conception ?



Diagrammes de classes

- Les *diagrammes de classes* représentent un ensemble de classes, d'interfaces et de collaborations, ainsi que leurs relations.
- Ils présentent la vue de conception statique d'un système.



Représentation des relations structurelles qui existent entre les entités, indépendamment de leur cycle de vie

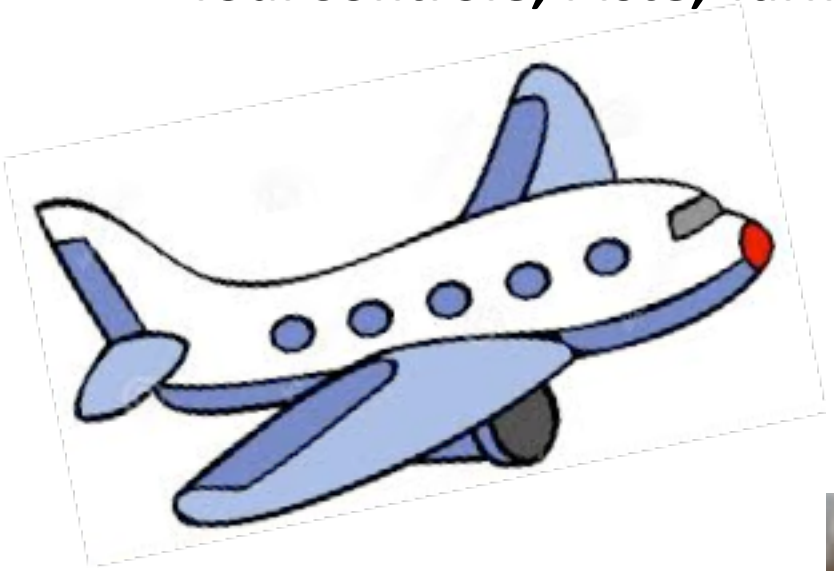
Relations statiques ?

- Ex.: un Contrat est associé à un Client
- Le fait qu'un Contrat soit signé et archivé relève de sa « dynamique ». On ne doit pas le modéliser dans le DCL
- SYN : relations structurelles
  - Validité dans la durée, le long terme

- Le fonctionnement **dynamique** d'une classe pourra être modélisé à travers des diagrammes d'état ou d'activité
  - Relation 'ponctuelle', le temps d'une action (message)

Relation « structurelle » ?

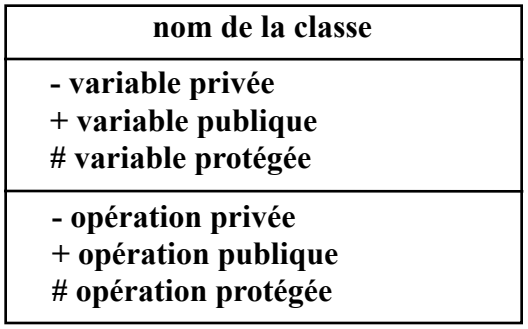
- Exemple avec un Système de Gestion Aéroport :
  - Si on envisage les classes **Avion, Compagnie** et **TourContrôle, Piste, Tarmac** … relations structurelles ?

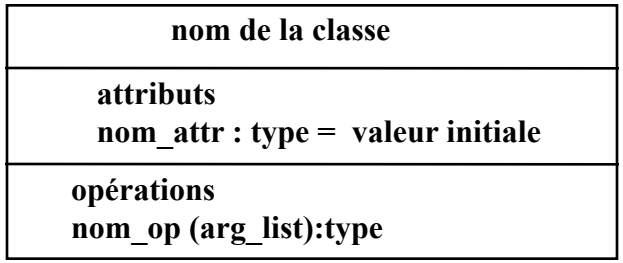


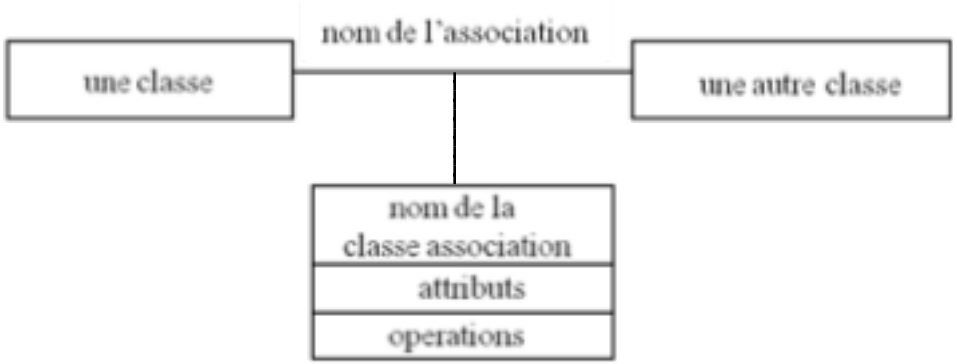
Relation « structurelle » ?

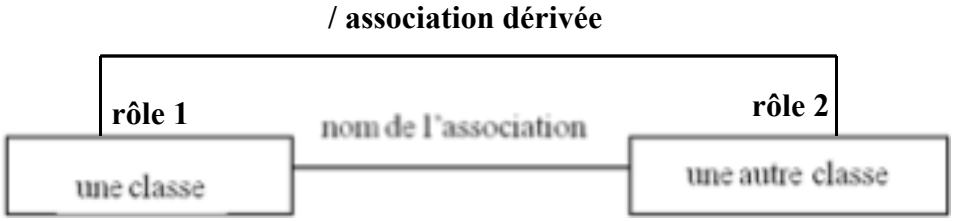
- Exemple avec un Système de Gestion Aéroport :
  - Relation *structurelle* entre **Avion** et **Compagnie,** entre **Aéroport** et **Piste** et **Tarmac**
  - La relation entre **Avion** et **TourContrôle** est de type dynamique, liée aux messages échangés (envoi de son identification, autorisation de décoller, etc.)

Diagrammes de classes





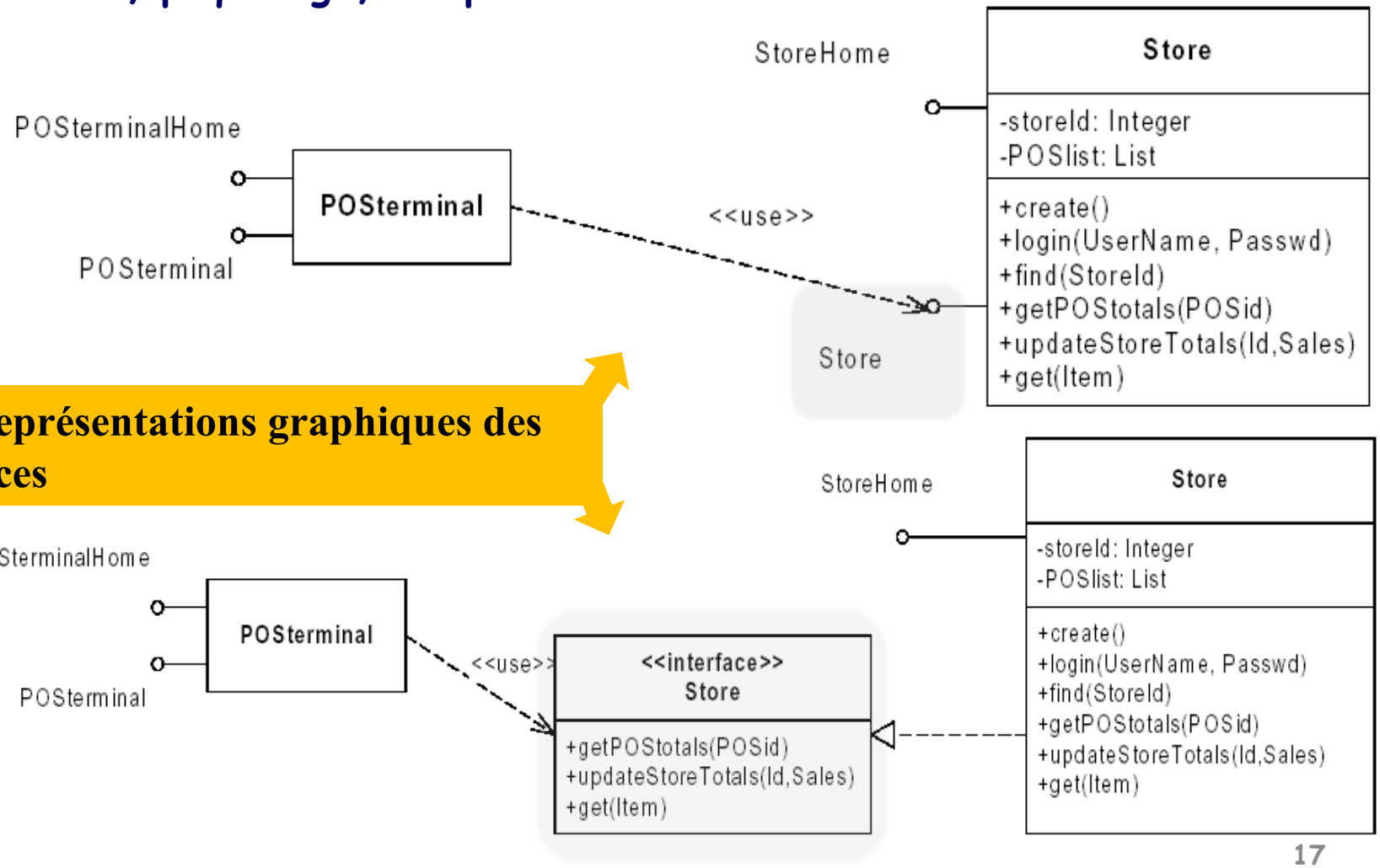




Variantes de classes

- **Interfaces : spécification des opérations visibles**
  - **Classes, paquetage, composants**

Implémentation de l'interface

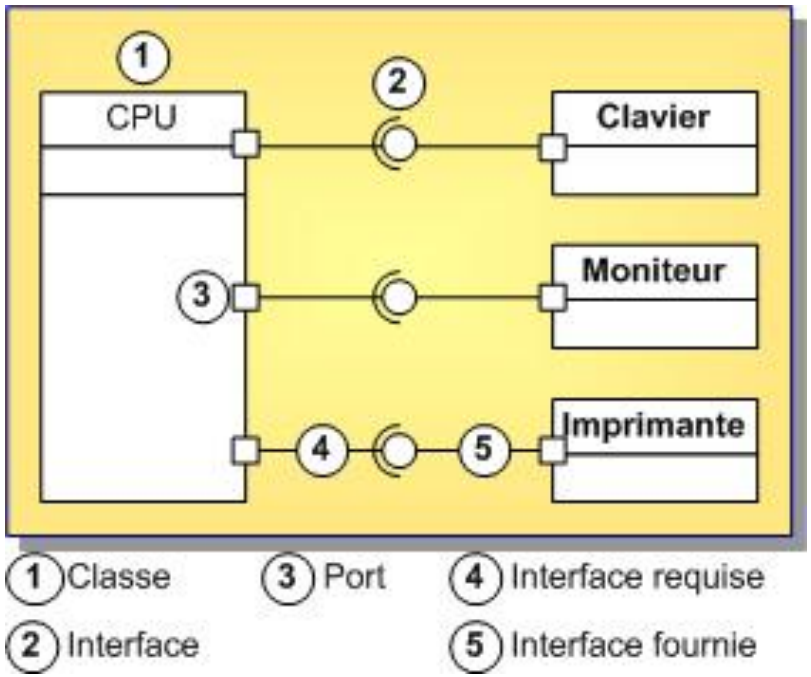


Interface / port (UML2)

- Une classe peut **offrir** un service :
  - (la notation se lit « lollipop »)

- Une classe peut **avoir besoin** d'un service :
  - (Notation « socket »)

- Un port spécifie un point d'interaction distinct entre ce classificateur et son environnement
  - ou entre le classificateur et ses parties internes

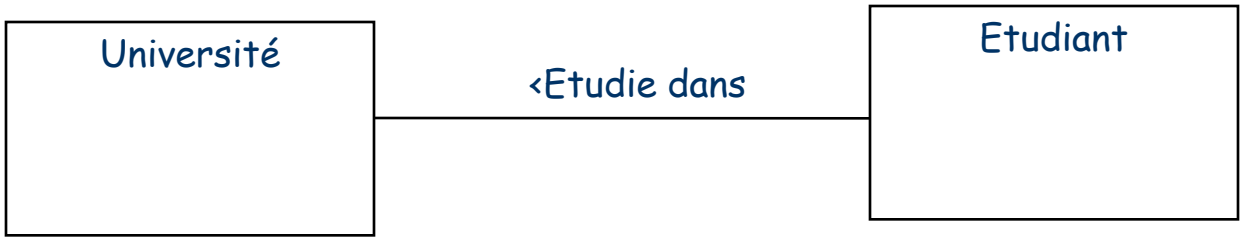


Les relations entre classes

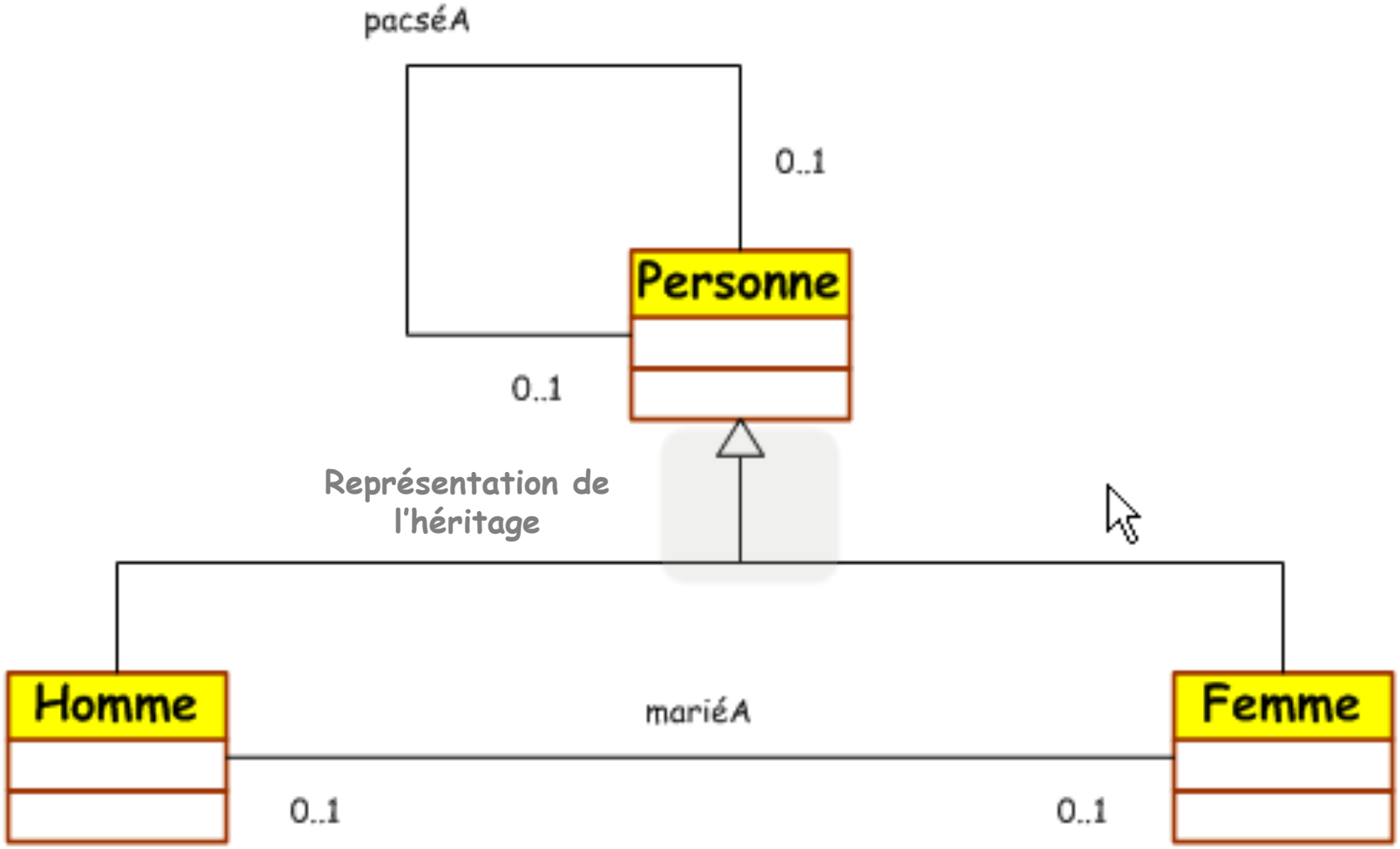
- L'association
- L'agrégation
- La composition
- L'héritage
- L'implémentation
- La dépendance

- L'association exprime une connexion sémantique *bidirectionnelle* entre classes
  - = abstraction des liens qui existent entre les instances

- Association nommée avec un sens :

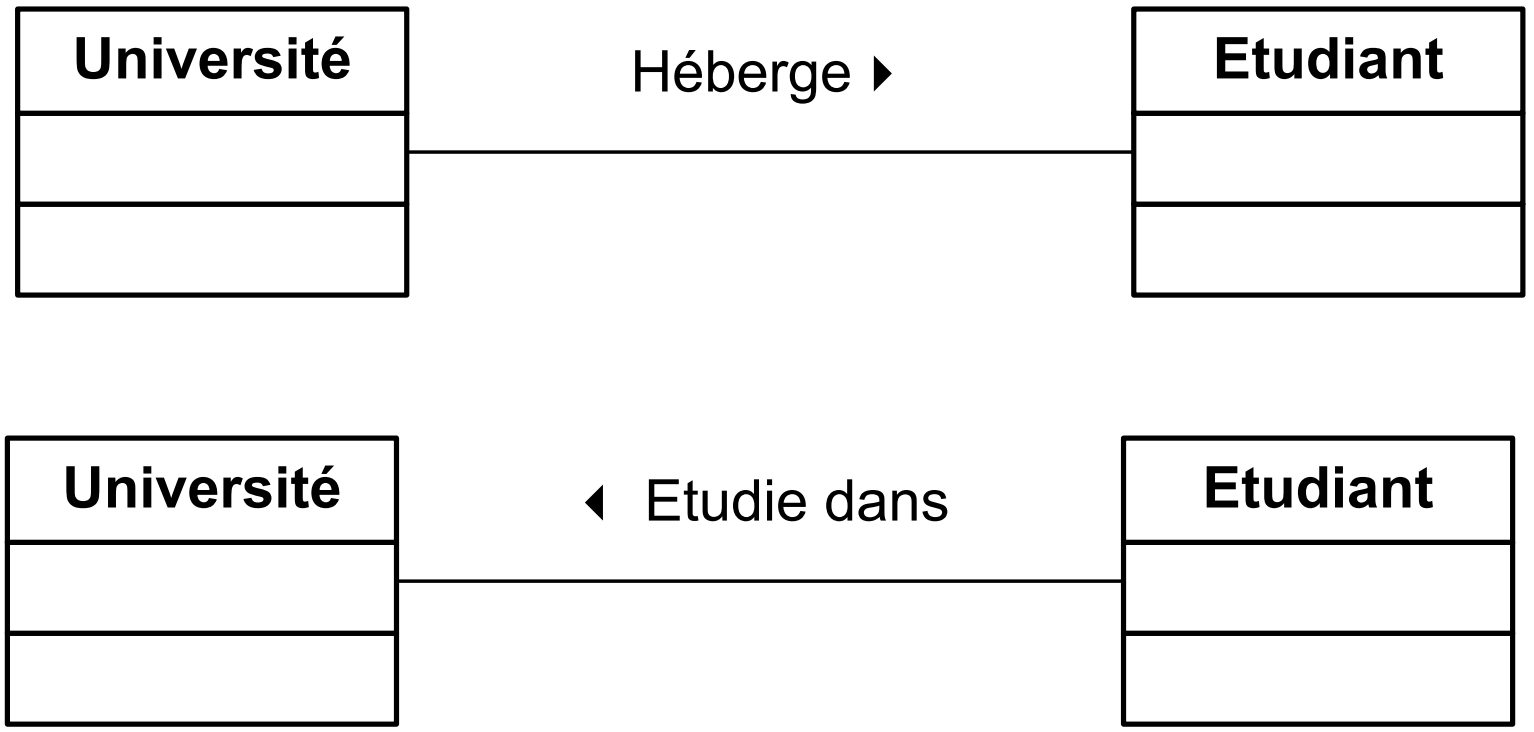


Exemple : associations et héritage



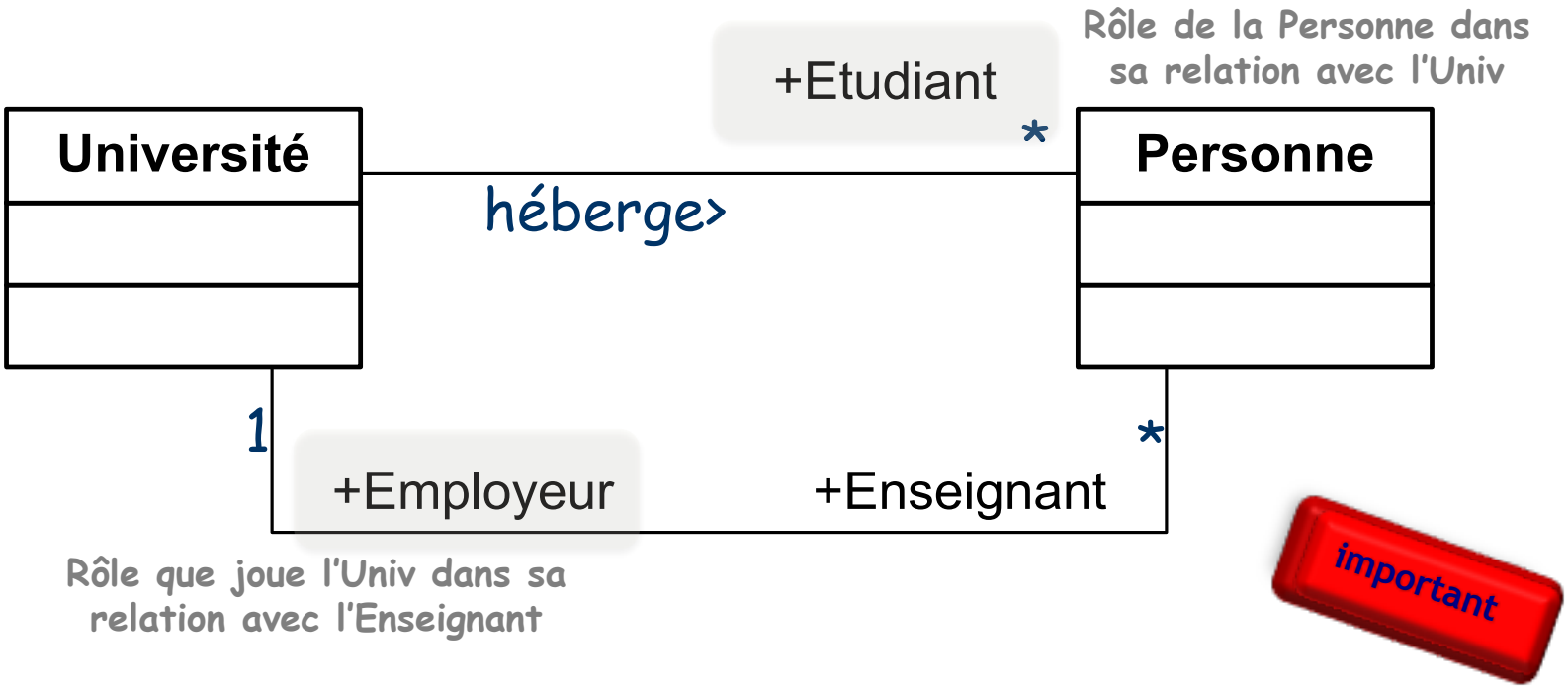
Soigner le nommage des associations

- **Indication du sens de lecture**



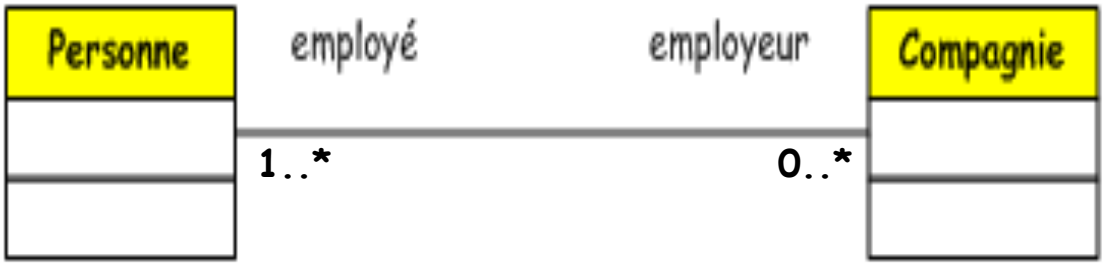
Nommage des rôles

- **Le rôle décrit une extrémité d'une association, c'est à dire le rôle que joue la classe dans la relation**



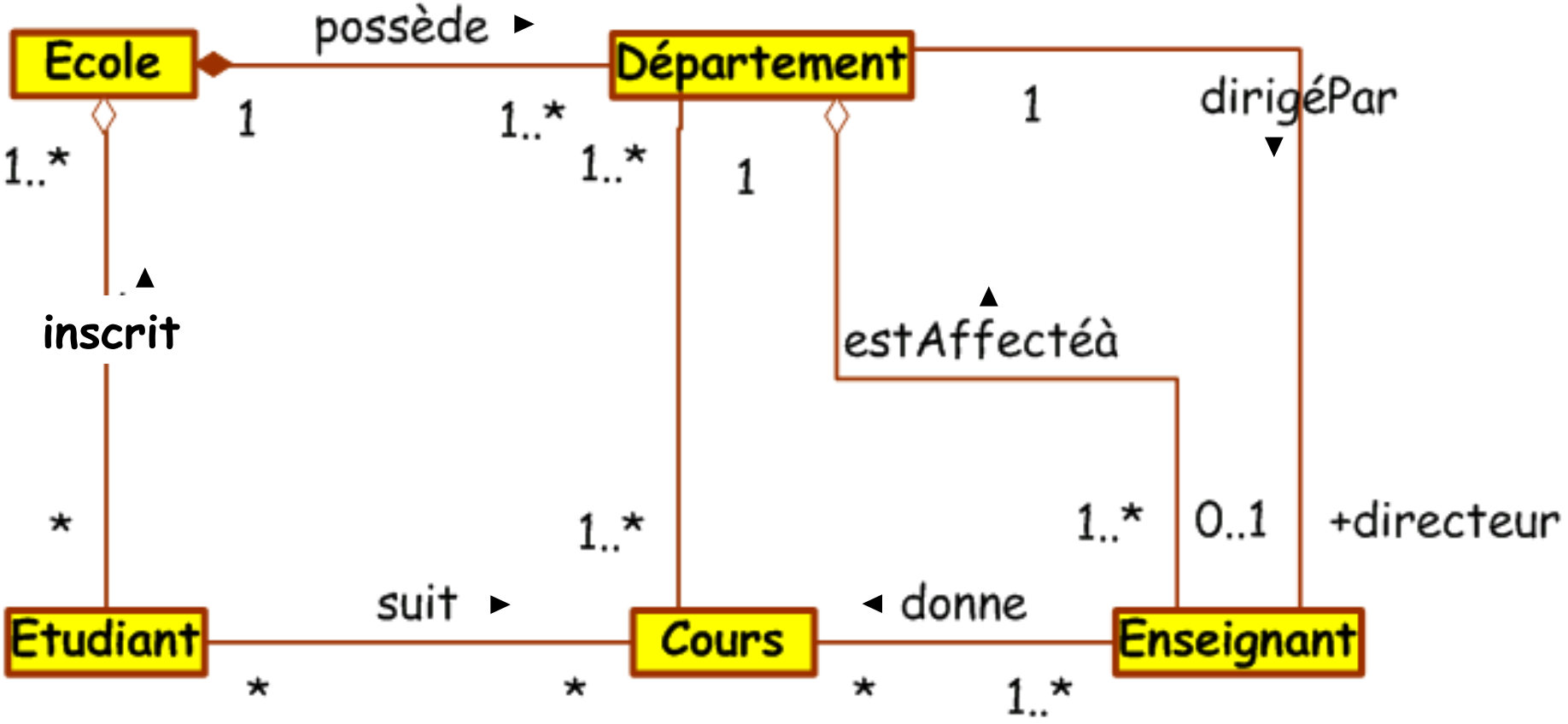
Il peut y avoir plus d'une association entre 2 classes

Multiplicité/Cardinalité des rôles

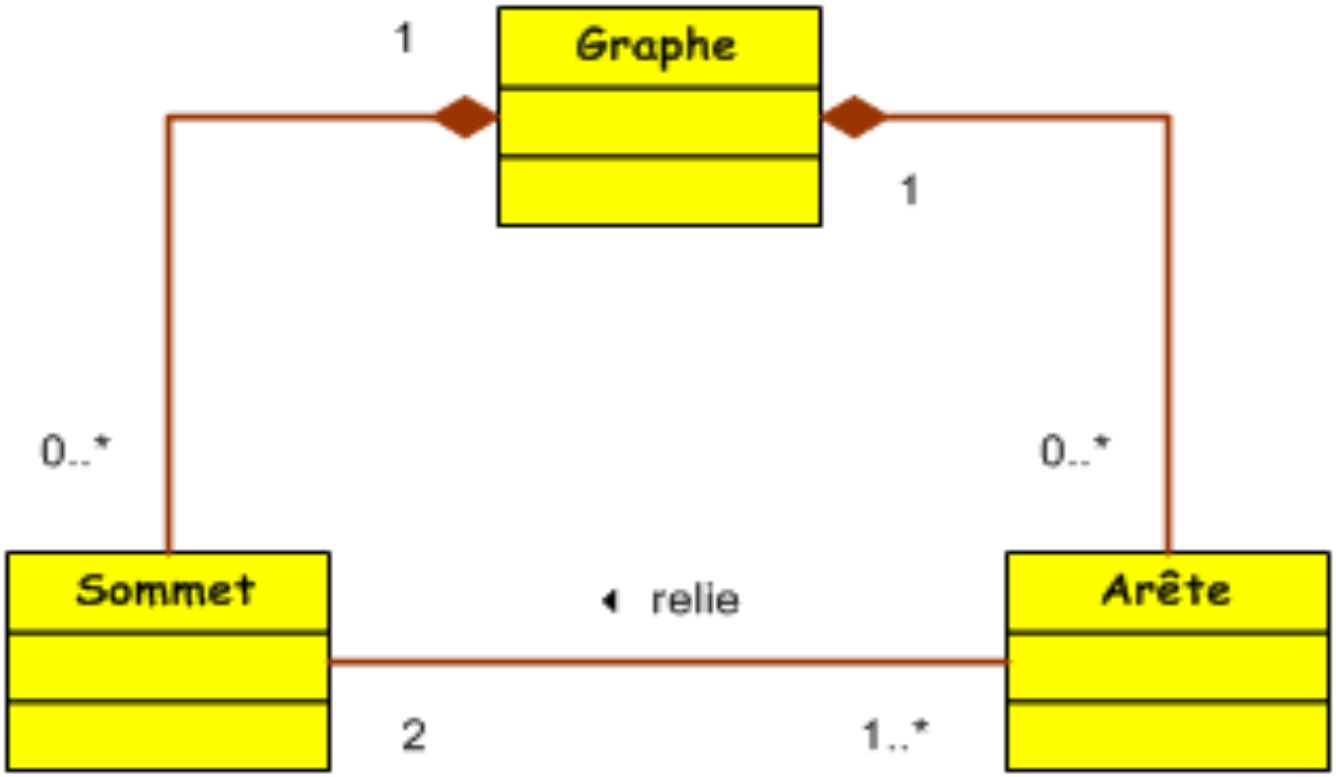


1	Un et un seul (cardinalité par défaut)
0..1	Zéro ou un
M .. N	de M à N (entiers naturels)
*	Plusieurs (entre 0 et N)
1 .. *	entre 1 et N
12	douze

Exemple : Ecole d'ingénieurs

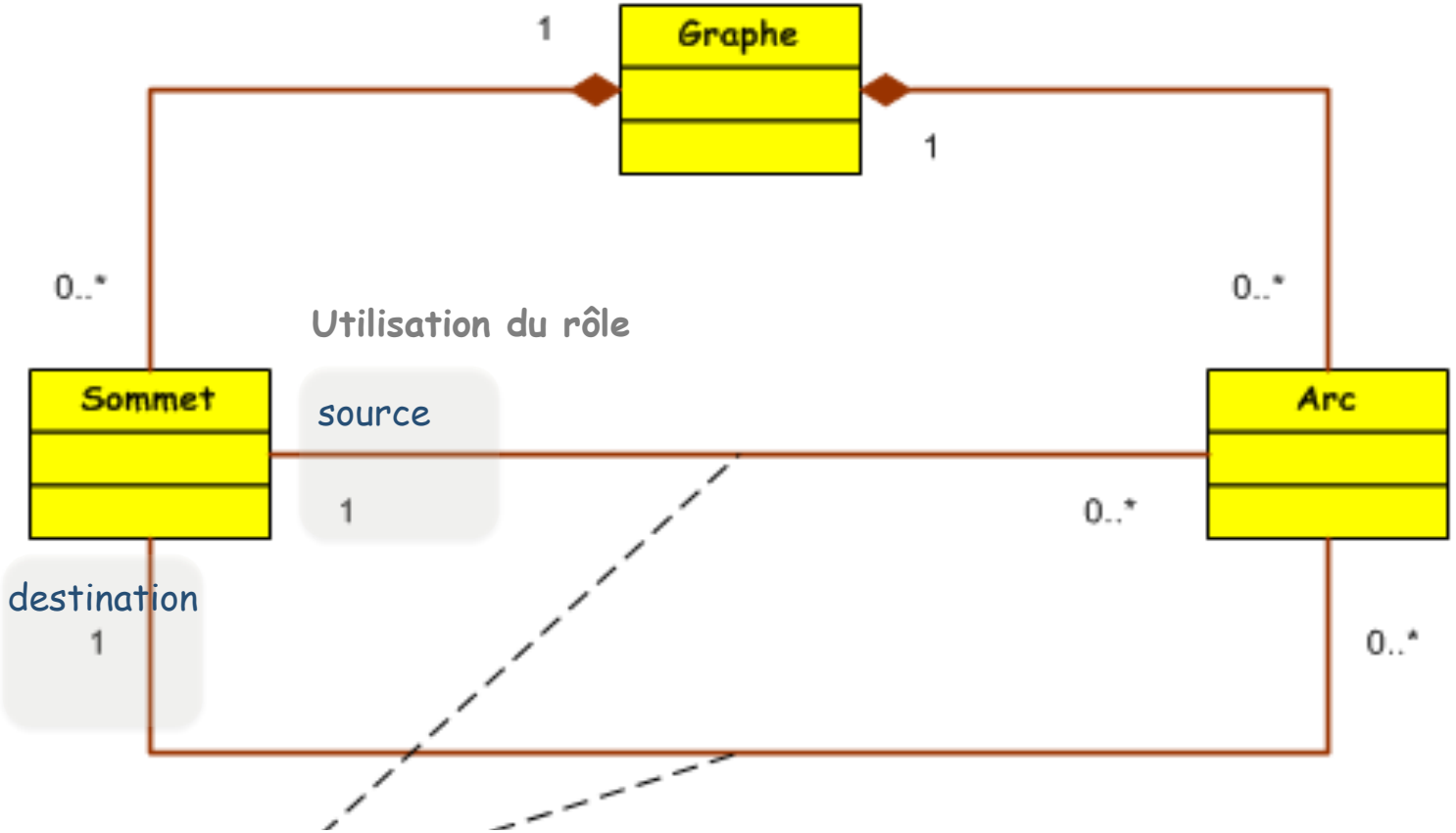


Exemple : graphe (1) non orienté



Un graphe possède des arêtes et des sommets, quand je crée le graphe je crée aussi ces éléments (idem quand je le supprime).
Une arête relie exactement 2 sommets.

Exemple : graphe (2) orienté



Ici avec les rôles, on a une information plus précise qui permet de distinguer les 2 sommets : la source et la destination, puisque l'arc est orienté

Navigabilité : traduit le sens de parcours d'une association

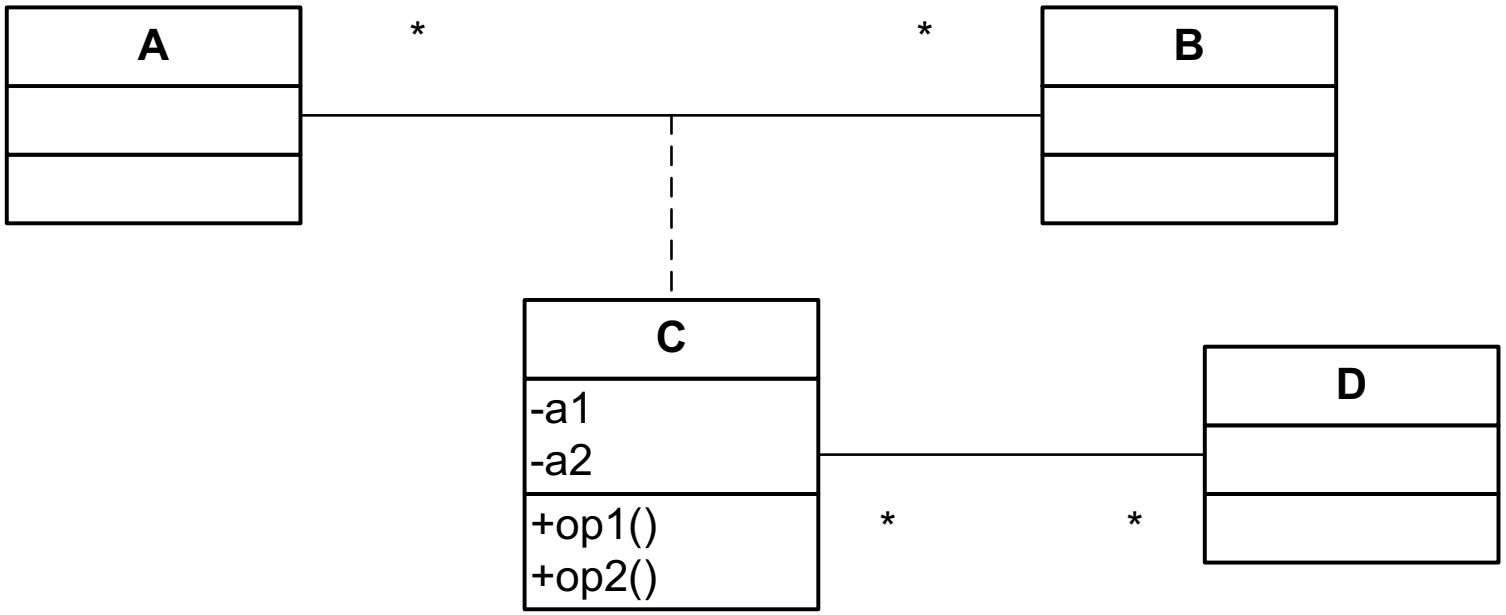
- **Par défaut une association est *navigable* dans les deux sens**
- **L'indication de navigabilité (flèche à une extrémité) suggère que seul un objet peut** directement **atteindre l'objet relié**

- **Etant donné un utilisateur, on désire pouvoir accéder à ses mots de passe**
- **Etant donné un mot de passe, on ne souhaite pas pouvoir accéder à l'utilisateur correspondant**

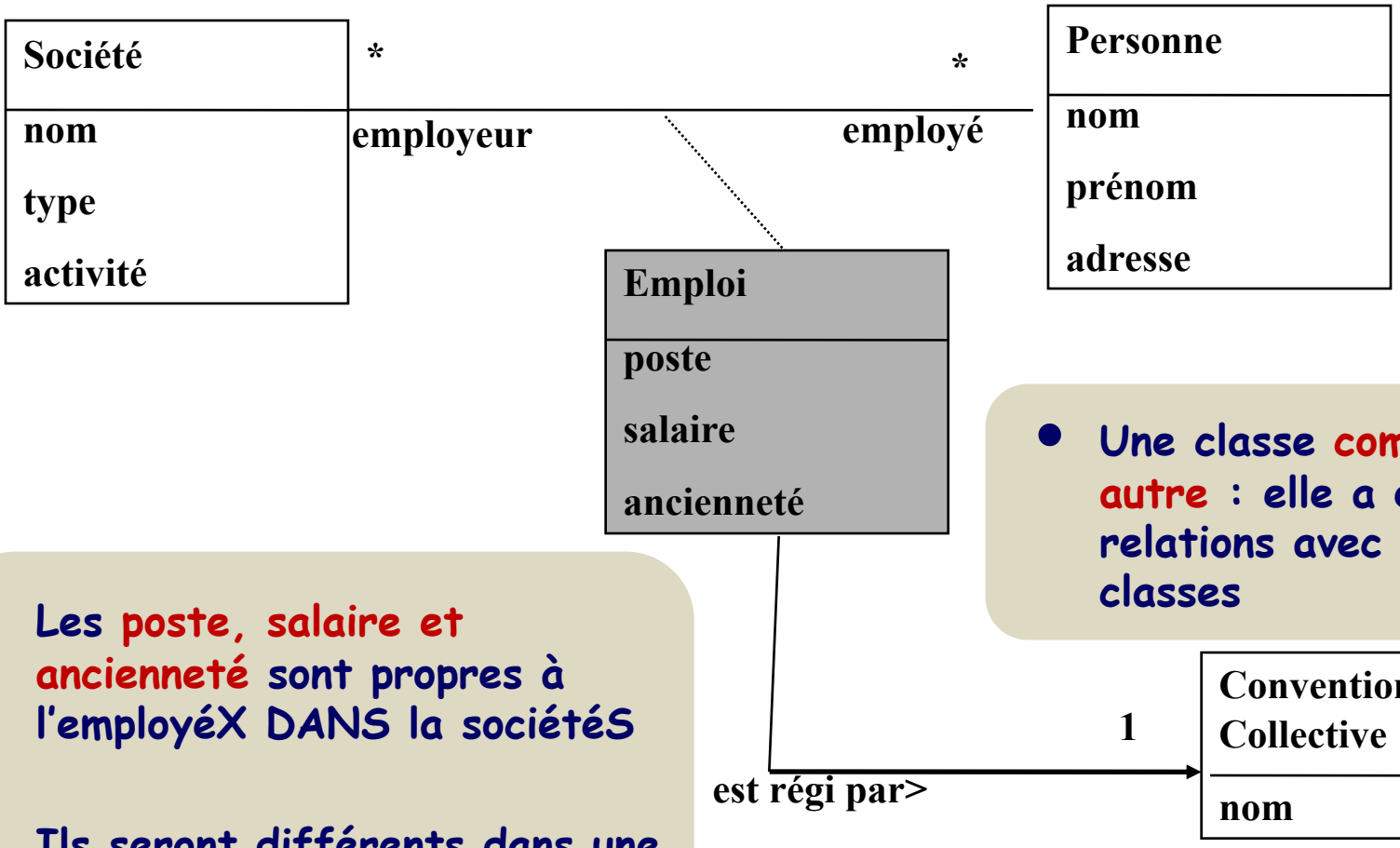
Les classes-associations

- Utilisée quand on a besoin d'en dire plus sur la relation *n,n* qui existe entre 2 classes :

  Ajout d'attributs ou d'opérations à la relation



Exemple de Classe d'association

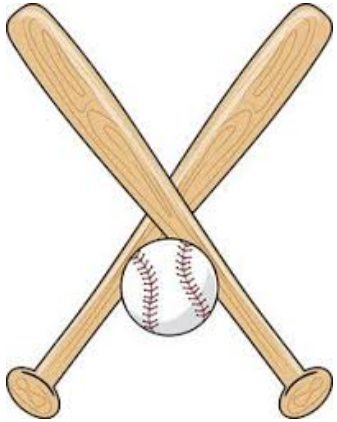


- Une classe comme une autre : elle a d'autres relations avec d'autres classes

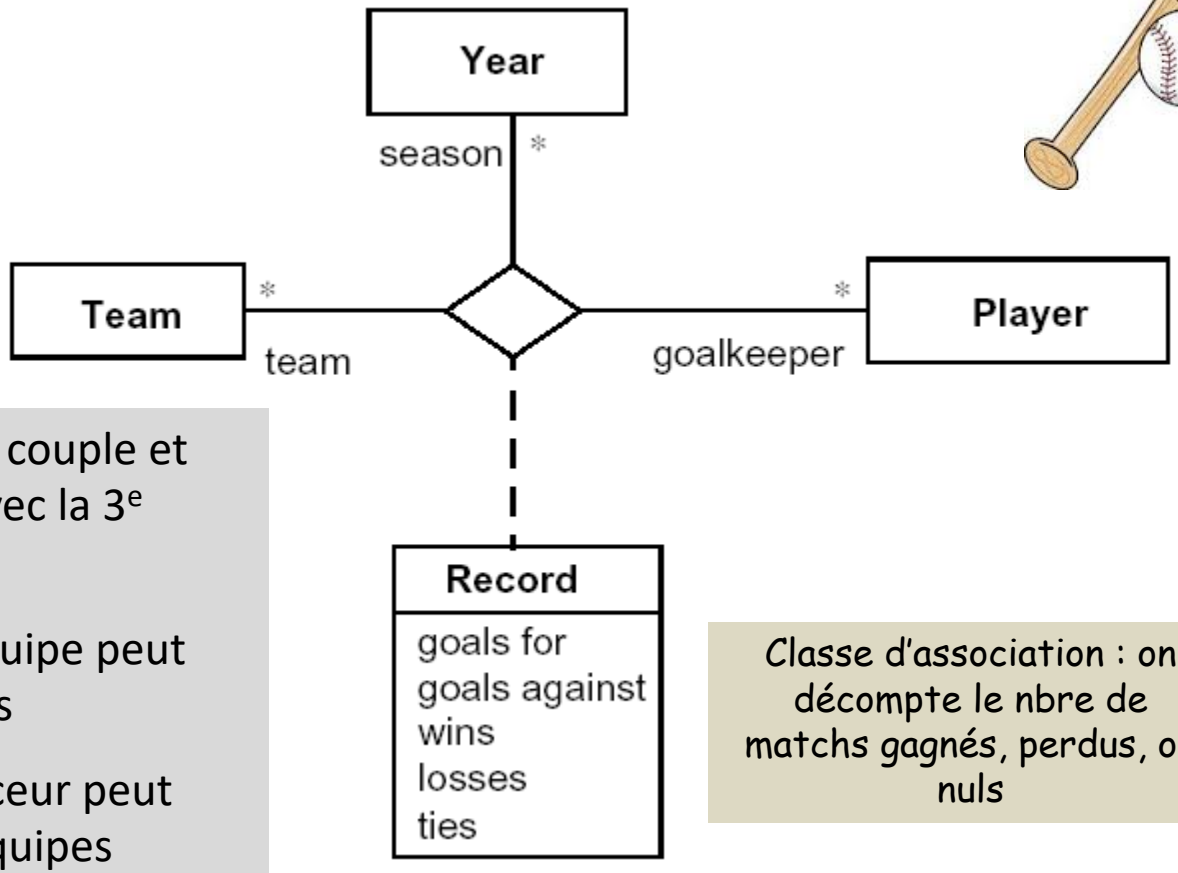
- Les poste, salaire et ancienneté sont propres à l'employéX DANS la sociétéS
- Ils seront différents dans une autre société

Associations ternaires (et plus)

- **Pas d'agrégation, pas de qualifieur**
- **Multiplicités plus difficiles à lire, à éviter**



BaseBall

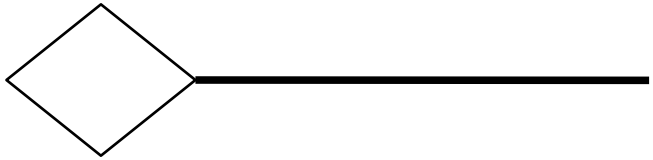


Multiplicité : on prend un couple et on définit la cardinalité avec la 3e classe. Ainsi ici :

- Sur une année, une équipe peut avoir *plusieurs* lanceurs
- Sur une année, un lanceur peut jouer dans *plusieurs* équipes
- Un lanceur peut jouer *plusieurs* années avec une même équipe

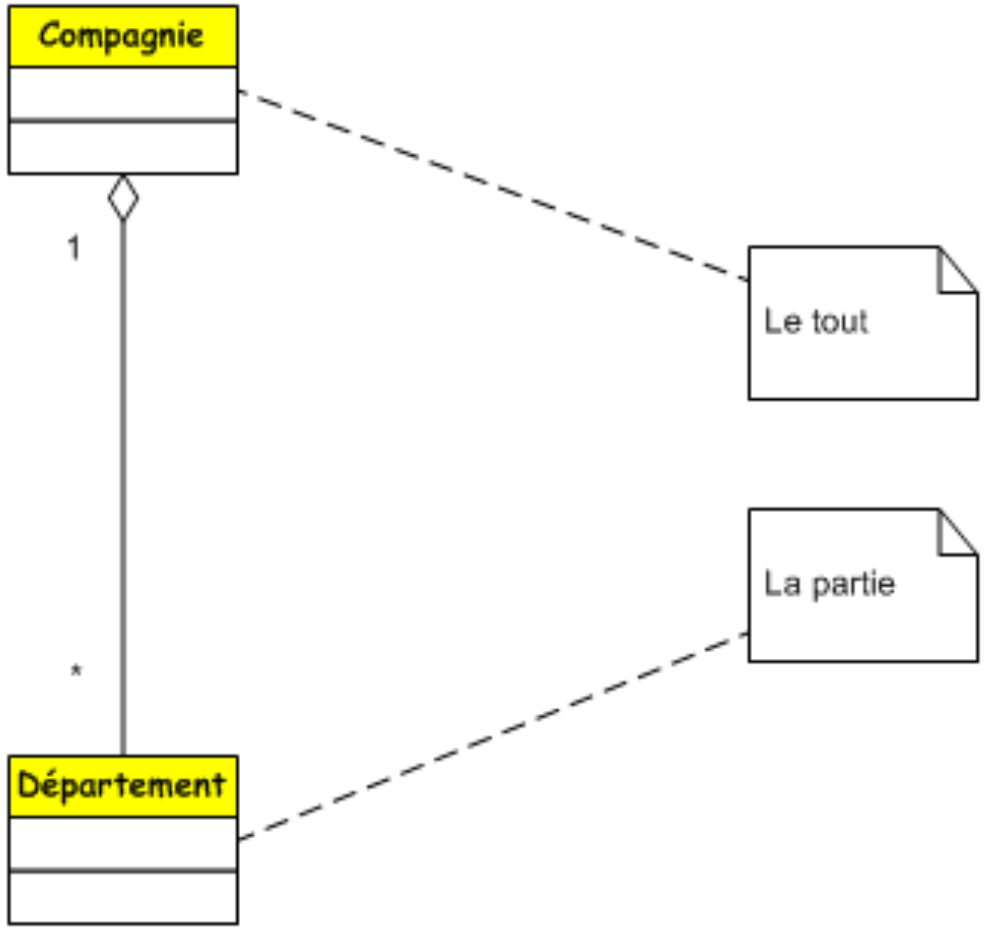
Classe d'association : on décompte le nbre de matchs gagnés, perdus, ou nuls

L'agrégation

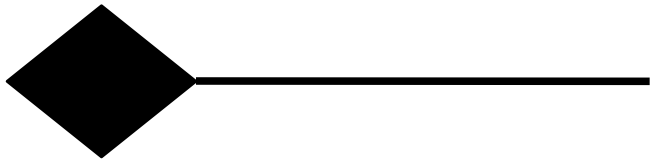


- Connexions bidirectionnelles antisymétriques
- Forme d'association qui exprime un couplage entre deux classes

- Représentation des relations traduisant :
  - maître et esclaves
  - tout et parties
  - composé et composant

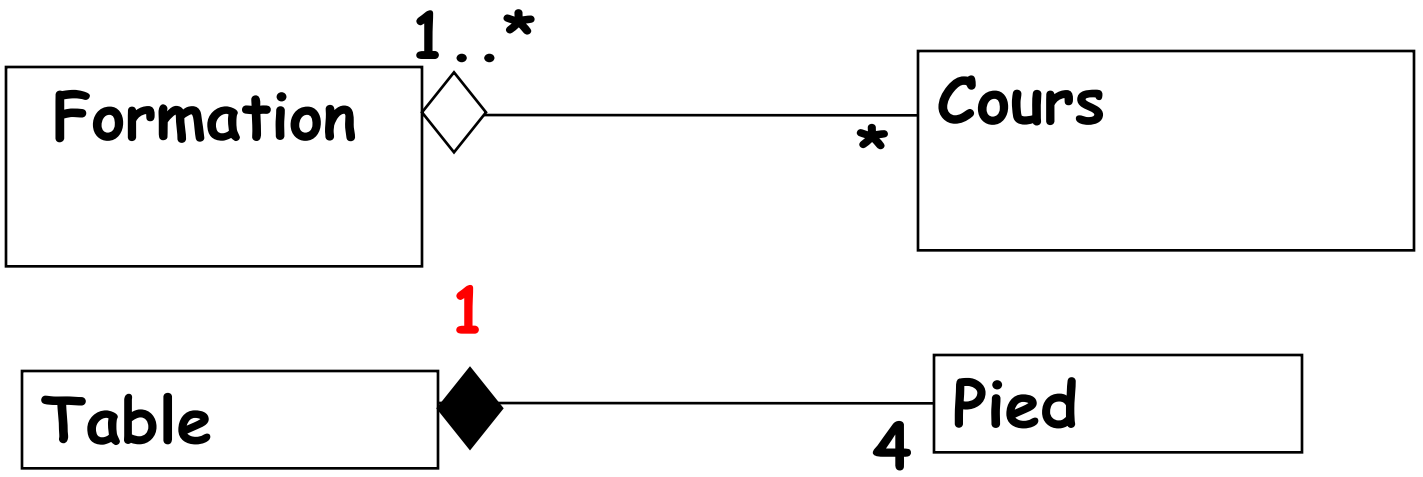


La composition



- = une agrégation forte
- Modélisation de la composition physique
  - Couplage très fort
  - Multiplicité au max de 1 du coté de l'agrégat
  - Propagation automatique de la destruction

Agrégation et composition



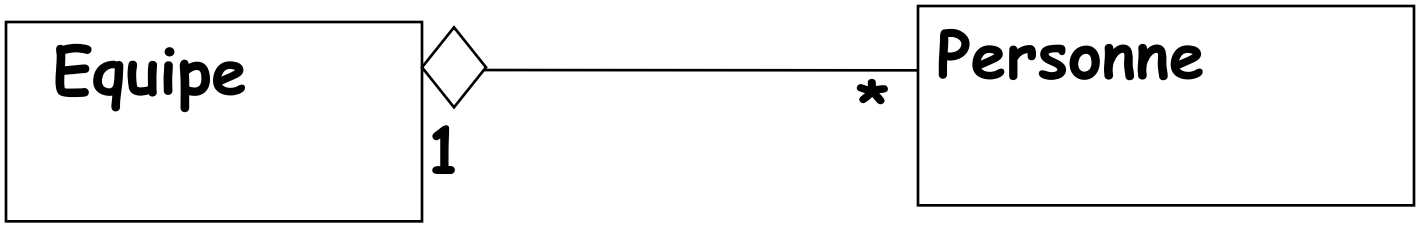
Relations non symétriques

Agrégation : la partie peut être partagée entre plusieurs entités, les cycles de vie des instances ne sont pas imbriqués

Composition : la partie est uniquement celle du composé, les cycles sont imbriqués (création et suppression)

Agrégation / Composition

- Une agrégation exprime qu'il existe une **contrainte d'intégrité** avec des classes dépendantes
  - C'est l'agrégat qui **est responsable** du respect de ces contraintes d'intégrité
  - Ex. une équipe comporte *n* personnes. Les personnes existent à part entière, de manière indépendante. L'équipe sait qu'elle est au complet par ex. quand toutes les personnes lui sont affectées.



Exercice : agrégation ou composition ?

- Une **voiture** possède 2 **essieux**, chaque **essieu** possède 2 **roues**
- Une **société** a des **salariés**
- Un **système de fichiers** possède des **répertoires;** un **répertoire** contient des **fichiers**
- Un **échiquier** est composé de 64 **cases**
- Toute **fenêtre** possède un bouton **Valider** et un bouton **Annuler**
- Une **page web** possède une **feuille de styles**

Implémentation d'une agrégation et composition

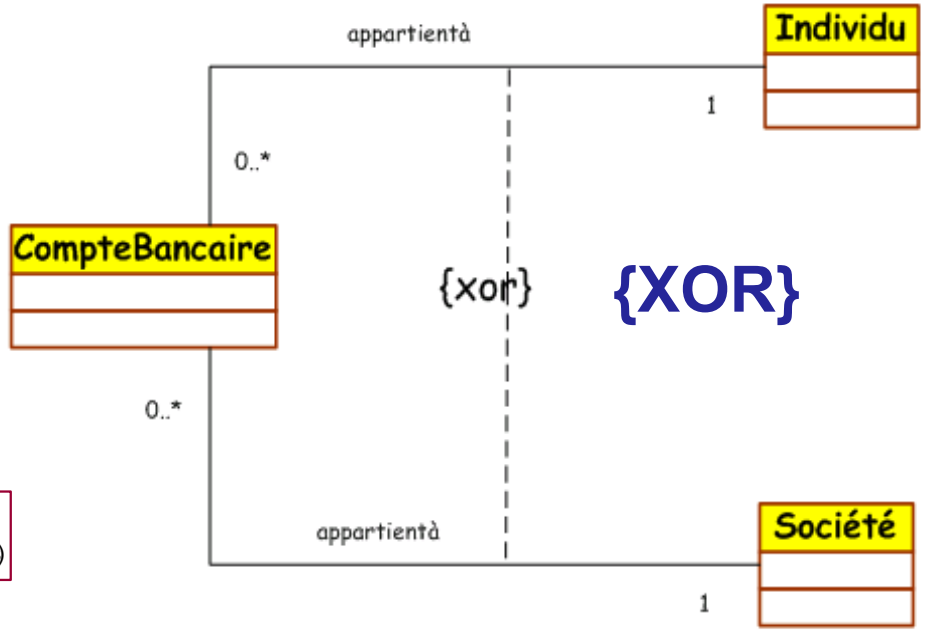
Exemple précédent de l'école

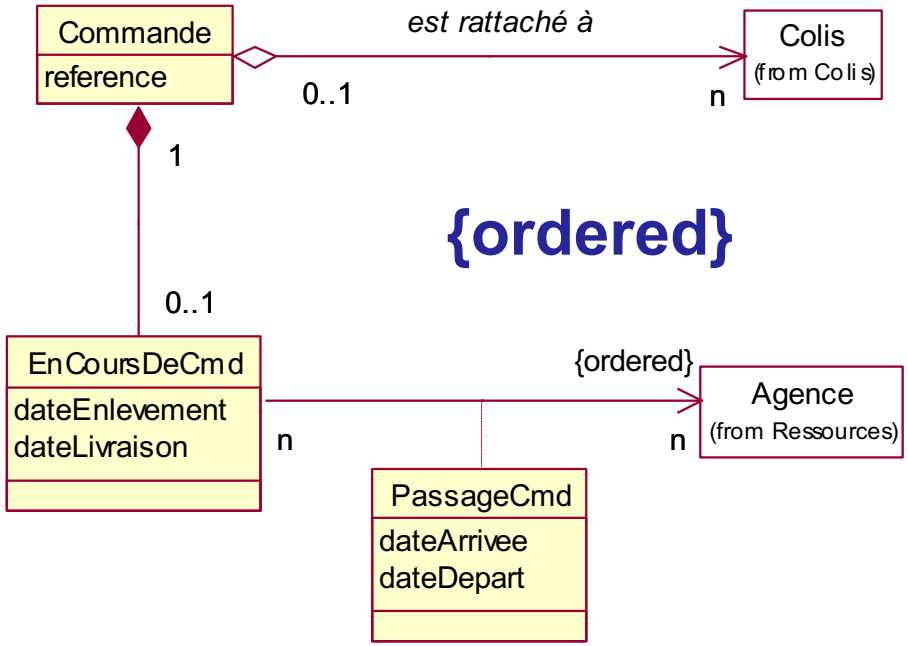
```
public class Ecole {
   private List<Departement> listDepartements;
   private List<Etudiant> listEtudiants;

   Ecole() {
   /* création et initialisation des départements
      dans le constructeur de l'école */
   /* + création du conteneur d'étudiants, vide */ ...
    }
    private setListEtudiants(List<Etudiant> uneListeEtu) {
       /* copie d'une liste dans une autre, les étudiants
          ont été créés par ailleurs */
    }
    private addEtudiant(Etudiant unEtu) {
       /* ajout de l'étudiant unEtu à la liste actuelle */
    }

}
```

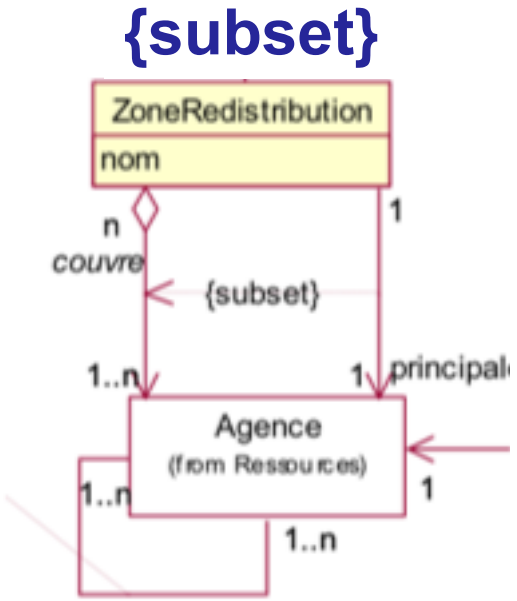
composition

agrégation

Contraintes sur les associations

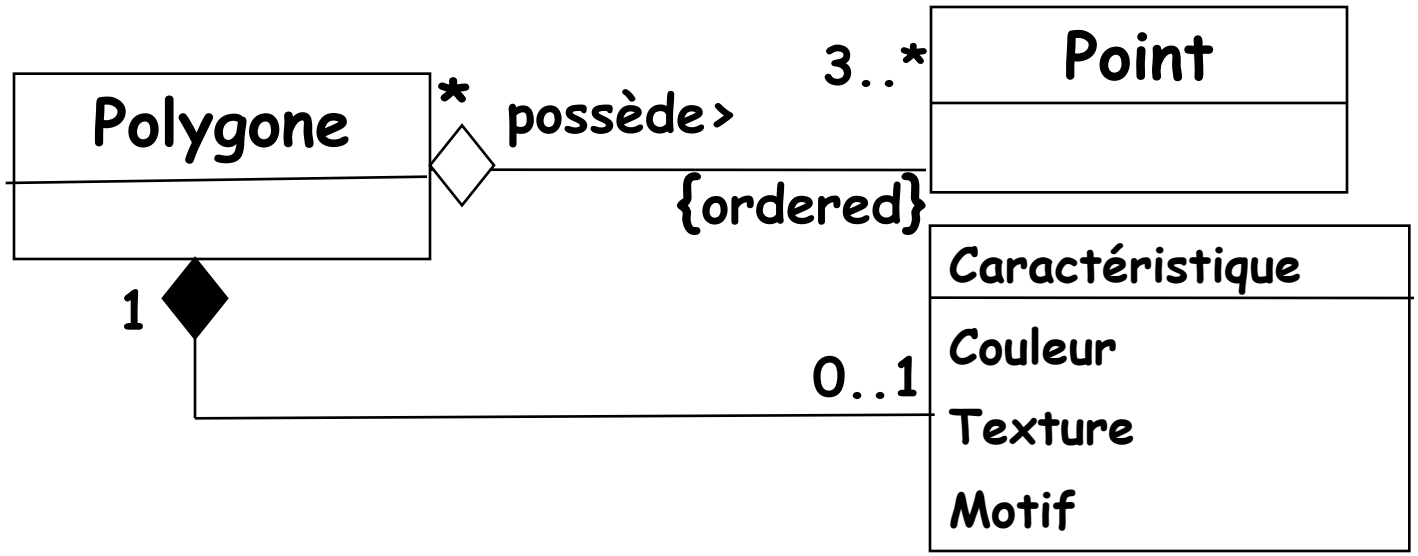






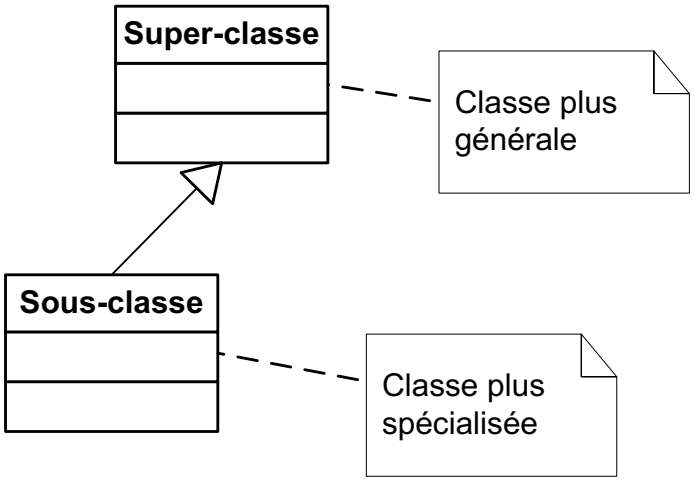
Exemple

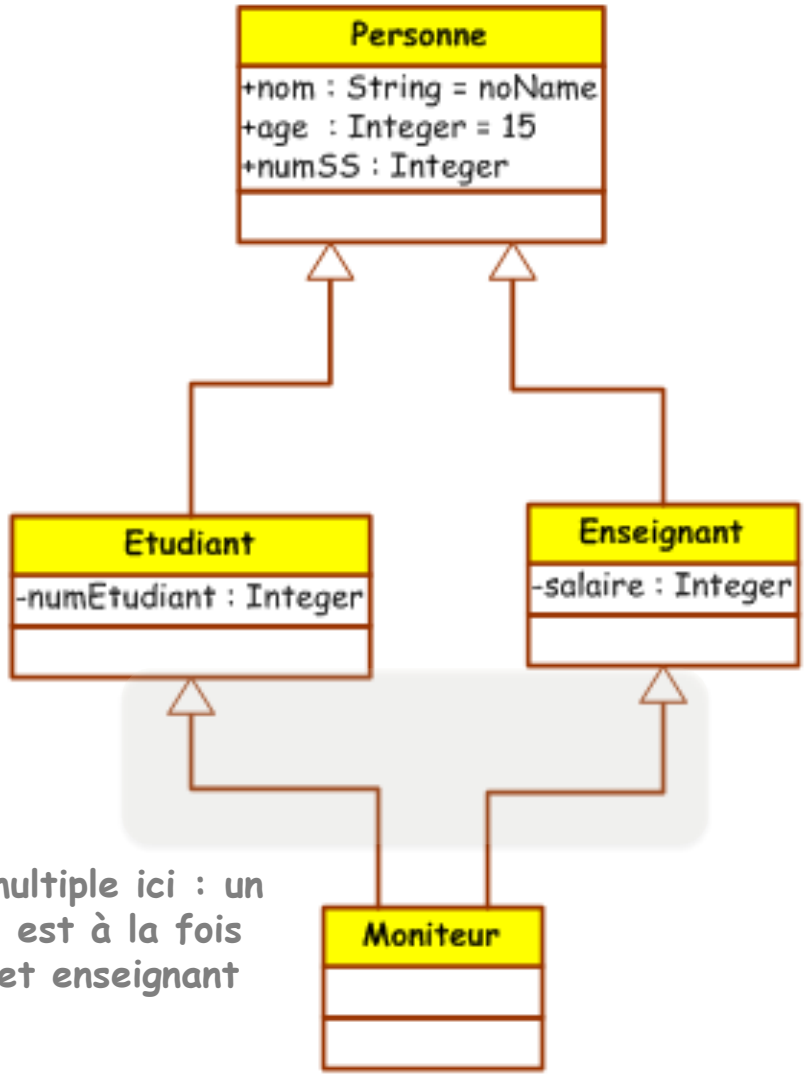
- Un polygone contient au moins 3 points
- C'est un agrégat de points ordonnés
- Il peut être composé d'une caractéristique



Relation d'héritage : hiérarchies de classes

- **Gérer la complexité**
  - **Arborescences de classes d'abstraction croissante**





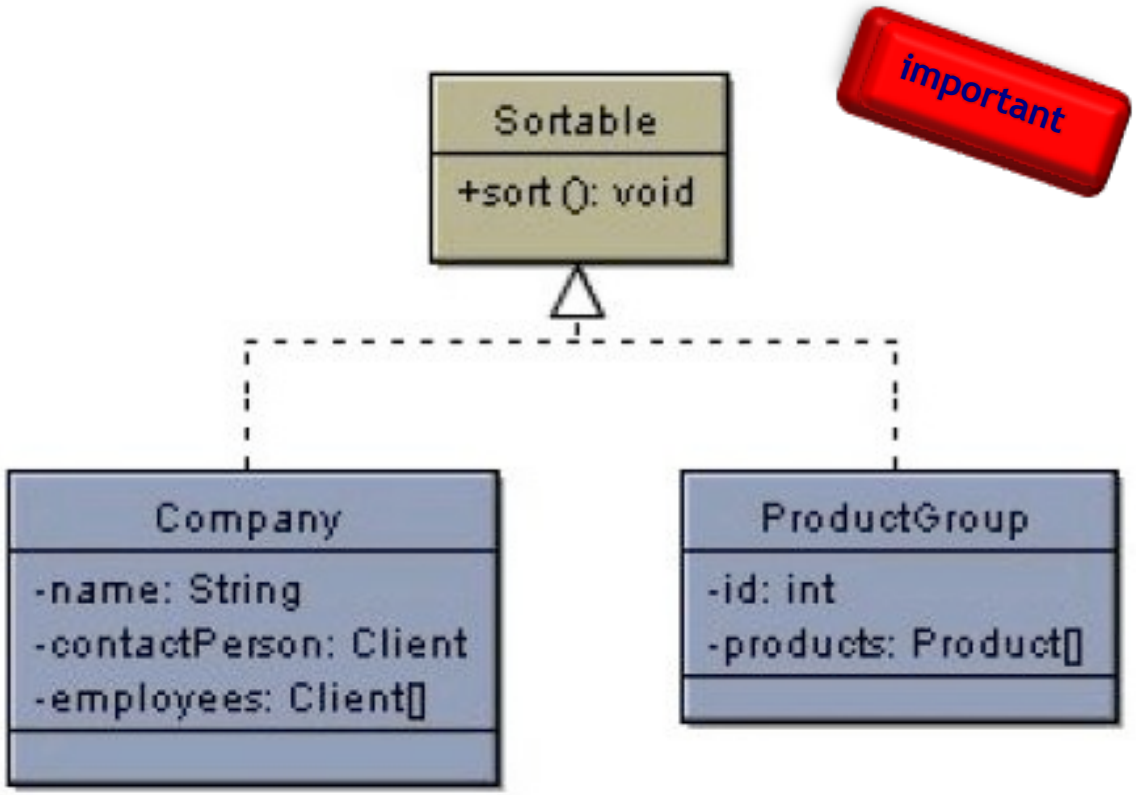
Héritage multiple ici : un 'moniteur' est à la fois étudiant et enseignant

Relation d'implémentation

Ici les classes Company et ProductGroup implémentent l'interface **Sortable**, ie définissent un code d'implémentation pour la méthode **sort**().

Trier des employés

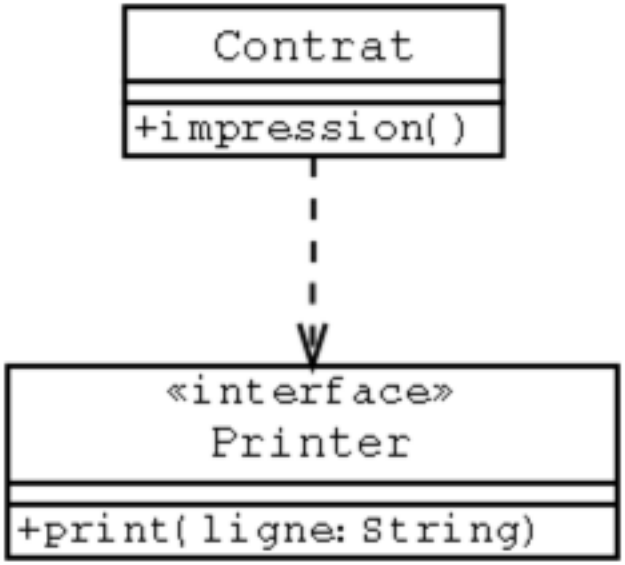
Trier des produits



Relation de dépendance

important

- **Relation entre deux éléments de modélisation**
  - Toute modification effectuée sur un élément de modélisation (*influent*) affecte l'autre élément (*dépendant*)
- **Dépendance : différent d'une relation *structurelle***
  - L'objet peut avoir besoin à un moment donné, des services d'un autre objet
  - Ex.: un contrat dispose d'un service d'impression (méthode `impression()`), qui utilise une méthode (`print()`), dont la spécification est déclarée par l'interface Printer.




```
class Contrat {

  ...
  public void impression() {
     Printer imprimante = PrinterFactory.getInstance();
     ...
     imprimante.print(client.getName());
    ...
  }
}
```

Relation de dépendance

Une classe A dépend d'une autre classe B quand celle-ci **utilise, à un moment dans son comportement, au moins un élément de B.**

Il sera donc nécessaire de **redéfinir une partie de A** si cette partie de **B est modifiée.**

- Relation la plus **générique** du diagramme de classe.
- Relation **non transitive** : si A dépend de B et B dépend de C, A ne dépend pas nécessairement de C

S'applique à:

- Diagramme de cas d'utilisation
- Digramme de classes
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement

On cherchera à limiter ces dépendances, pour éviter le plat de spaghettis

DCL : Règles de nommage

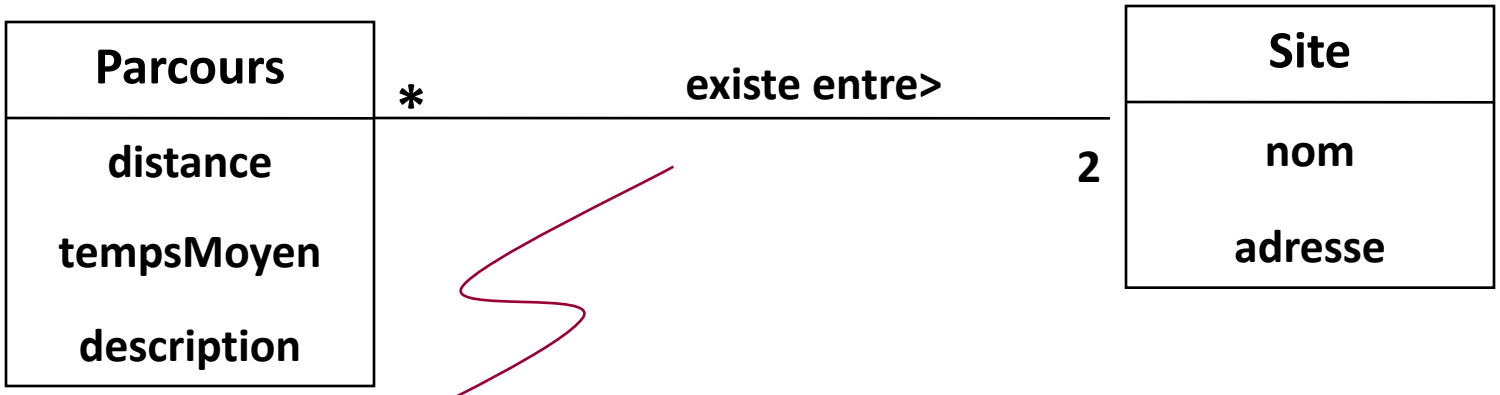
- Génération automatique de code → respecter au maximum les **chartes de nommage** ainsi que les bases de syntaxe des langages
  - Nom de classe commence par une **Maj** ; tout le reste en minuscule (sauf les constantes en maj.);
  - Noms de classe au **singulier** ;
  - Séparer les mots composés par des majuscules ;
  - Ecrire soit en FR soit en Anglais, pas de mélange

Exercice Diag. de classes

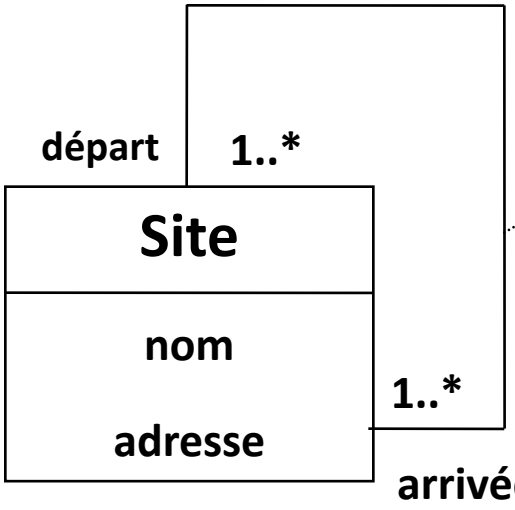
- Représenter de deux façons différentes le fait qu'un parcours entre 2 étapes part d'un lieu pour arriver à un autre :
  - Modèle avec une Relation
  - Modèle avec une Classe d'Association

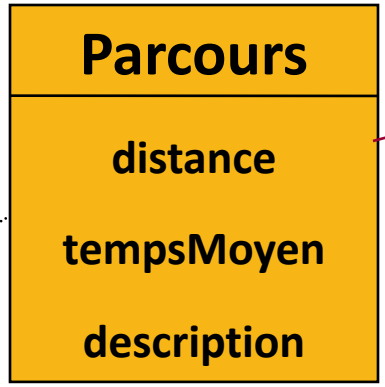
- A votre avis, quelle est la meilleure représentation ?

Solution exercice



- même niveau d'importance aux 2 classes
- *n* parcours possibles entre 2 sites



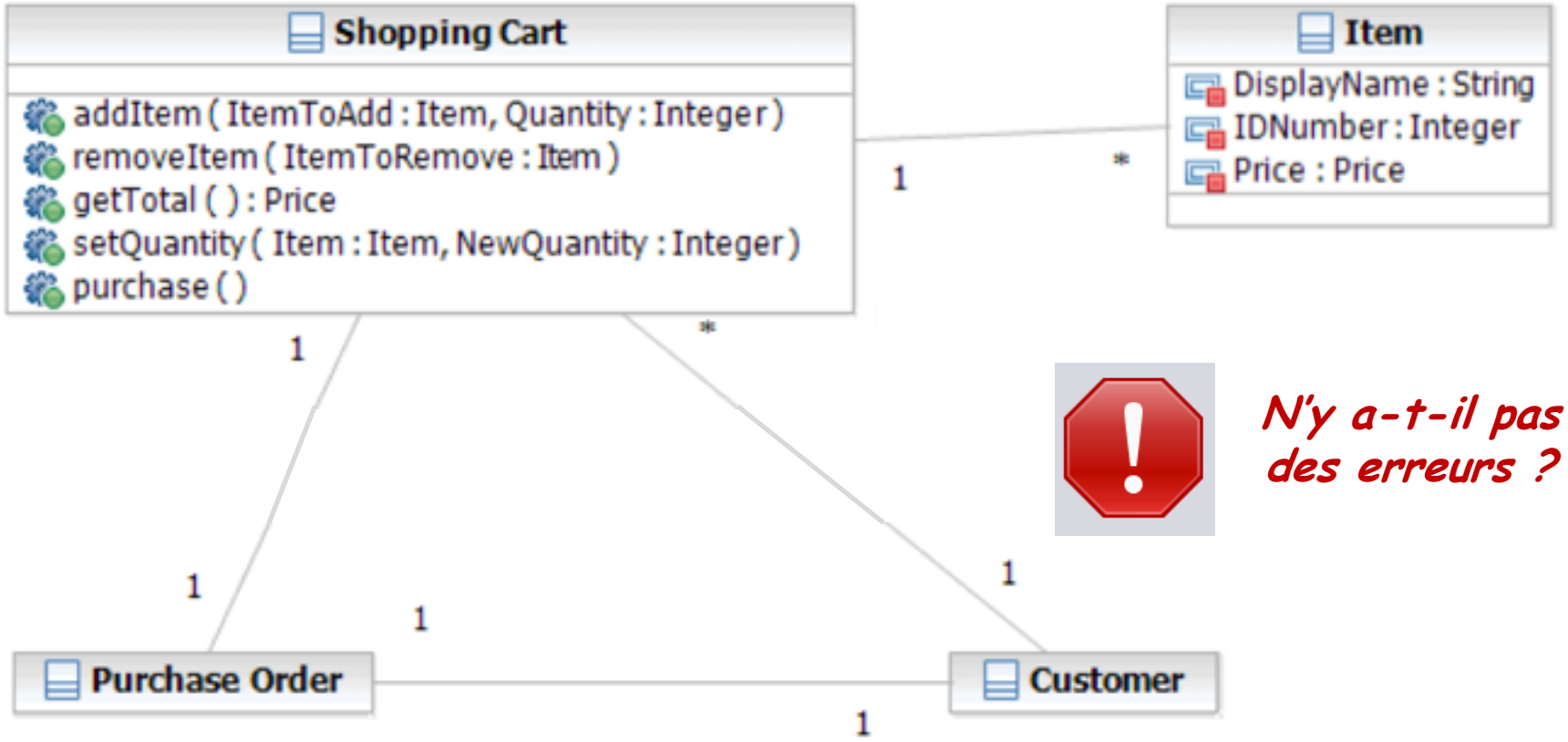


Classe d'association

Là un seul parcours possible entre 2 sites (subordonné aux sites)

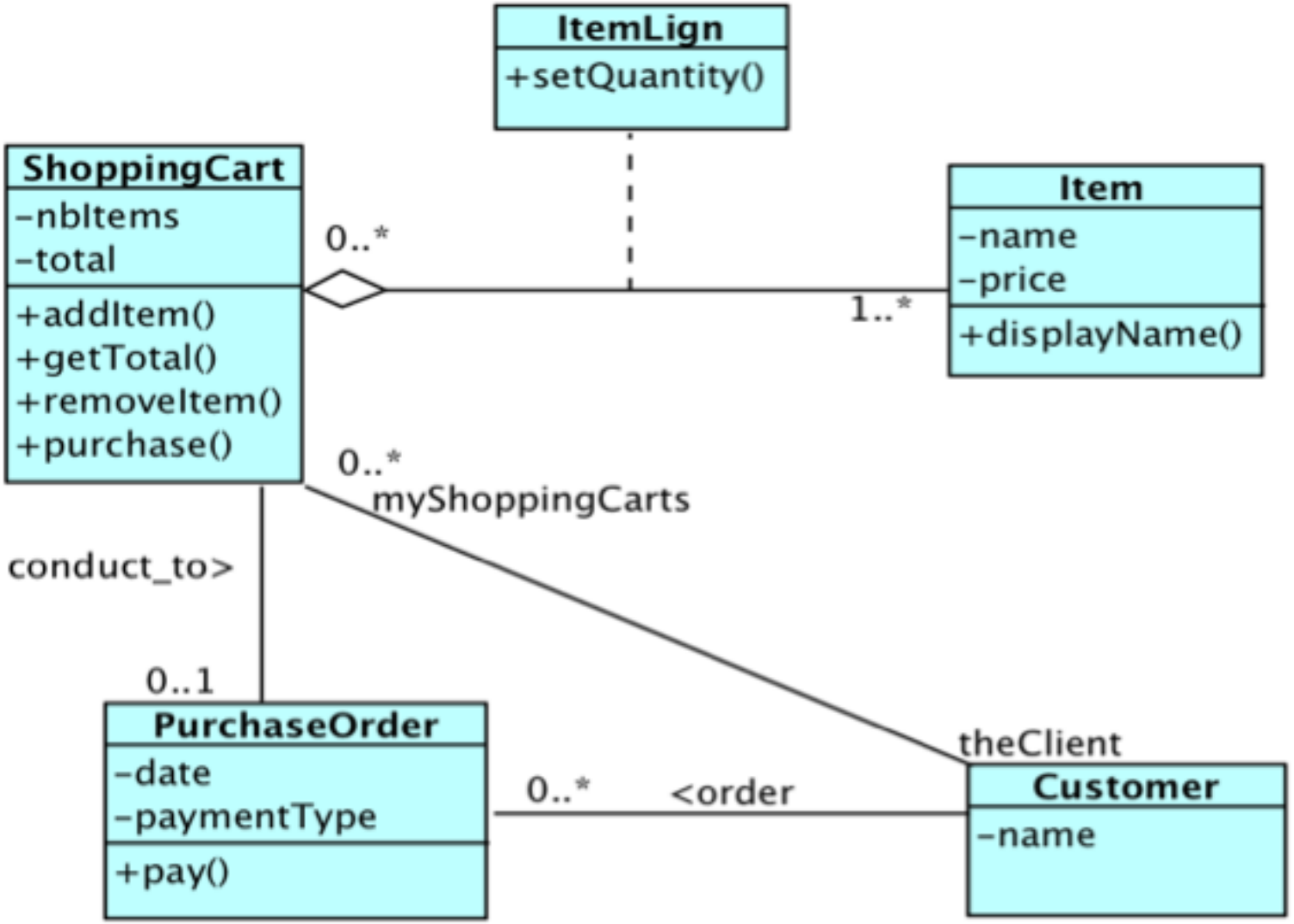
Que raconte ce diagramme

(trouvé sur le net) ?



http://www-01.ibm.com/support/knowledgecenter/SS8PJ7_9.1.1/com.ibm.xtools.modeler.doc/topics/cclassd.html?lang=fr

Version corrigée



DCL : conclusion

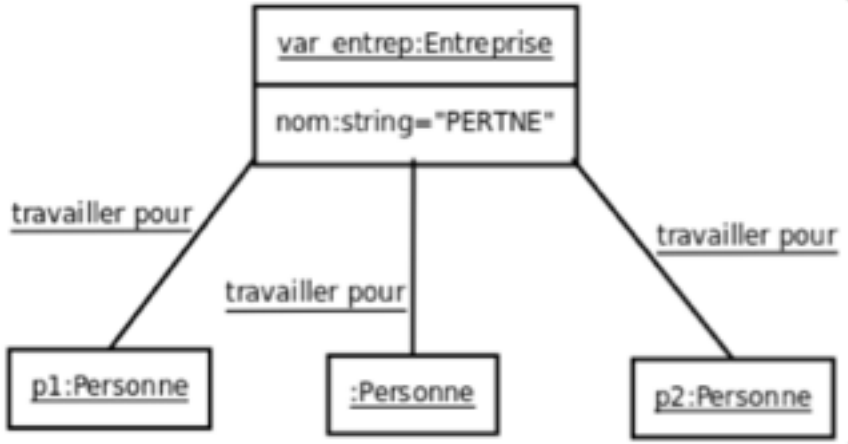
- Un DCL n'est pas forcément une représentation **exhaustive** d'un modèle
- Un diagramme **bien structuré** doit :
  - Être centré sur la communication d'une *vue* du système
  - Ne contenir que les éléments *essentiels à la compréhension* de cet aspect. Un diagramme peut cacher des parties du modèle si cela sert la communication

- *Les modèles ne sont pas **justes** ou **faux**; ils sont seulement plus ou moins utiles - Martin FOWLER*
- *« Un bon modèle n'est pas un modèle auquel on ne peut plus rien **ajouter**, mais un modèle auquel on ne peut plus rien **ENLEVER** » St Exupéry*

DCL : je retiens

- Les classes et les relations **structurelles**
- Les **différents types** de relations
- Importance des **cardinalités**, comment on les détermine
- La notion de **rôle d'une classe**
- Les **classes d'association**
- Les **propriétés** *ordered, XOR* et *subset* des associations

Diagrammes d'objets



Objets

Représentation graphique des objets

- **Le nom d'un objet est souligné**
  - **Nom : Classe**
  - **Nom**
  - **:Classe**

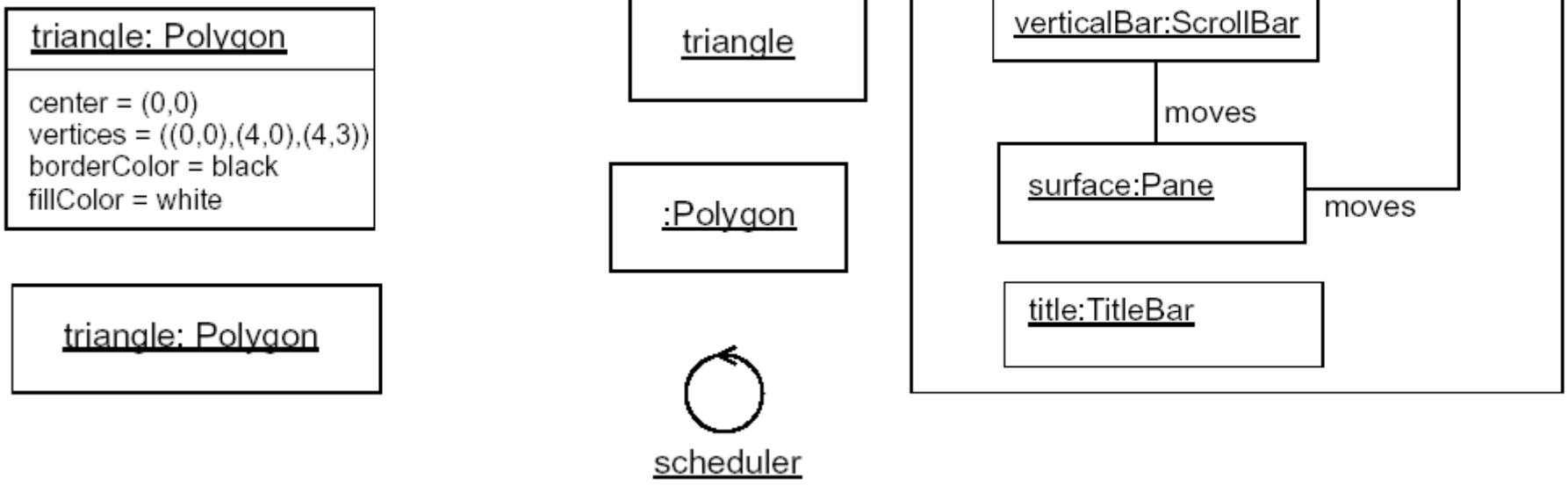
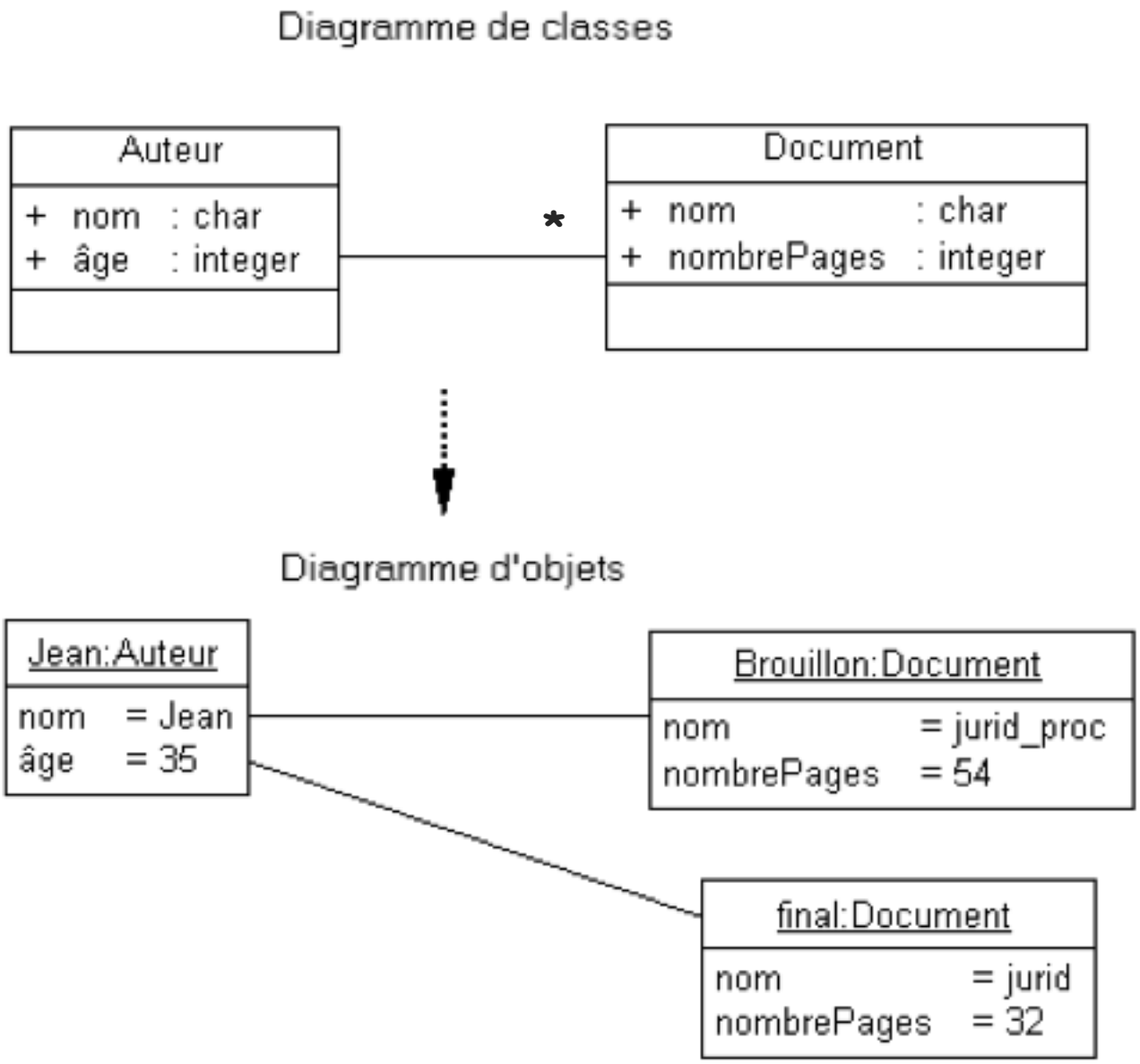


Diagramme d’objets

- Les *diagrammes d’objets* représentent un ensemble d’objets et leurs liens. Ce sont des vues **statiques** des instances des classes.
- Ils présentent la vue de conception d'un système, exactement comme les diagrammes de classes, mais à partir des données issues de **cas réels** ou de prototypes.

- Un diagramme d’objet est donc une **instance** d’une diagramme de classes



Source : documentation PowerAMC - Sybase

Utilisation des objets

- Le diagramme d'objets n'est quasiment pas utilisé
- Par contre on utilise les objets dans les diagrammes qui **modélisent le comportement** du système
  - Pour représenter la dynamique
  - Diagramme de séquences objets surtout
  - (Diagrammes d'activités)
- Liés à la notion de scénario (SCN)
  - Un SCN = instance d'un déroulement, avec des objets