Module *S1-10 – Qualité de Développement – ASPE*

Chapitre 1 – Rappels UML

Diagramme de Classes

Véronique DESLANDRES
IUT de LYON

Plan de ce cours

- Les diagrammes de classes (DCL)

- Classes du domaine vs de Conception (7)
- Les relations (14)
- Rôle des classes (16)
- Classe d'association (20)
- Agrégation / composition (24)
- Fiche Je retiens… (38)

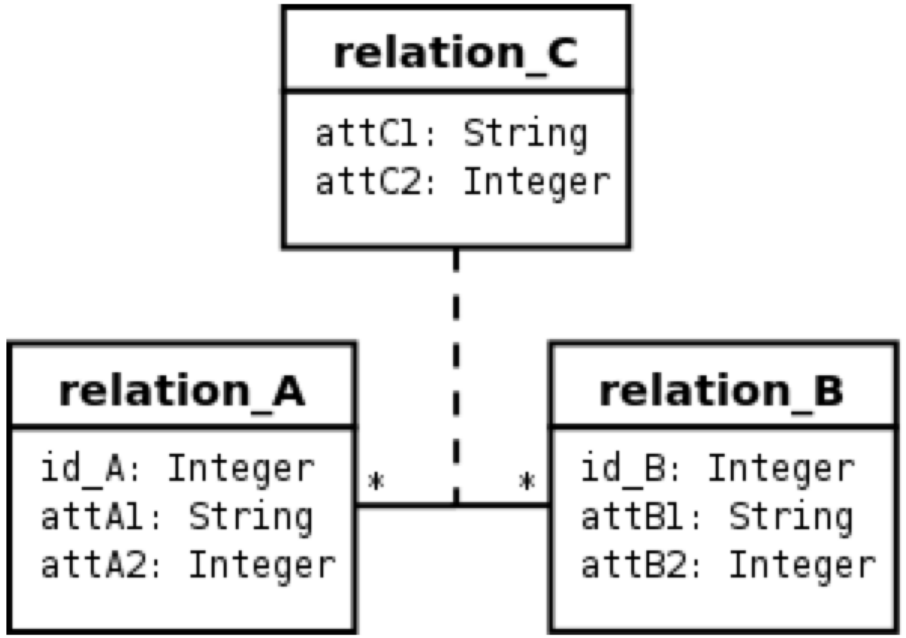
Pré requis

- Notions de base d'UML acquises
  - Diagrammes de classes et d'objets
  - Ici uniquement 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

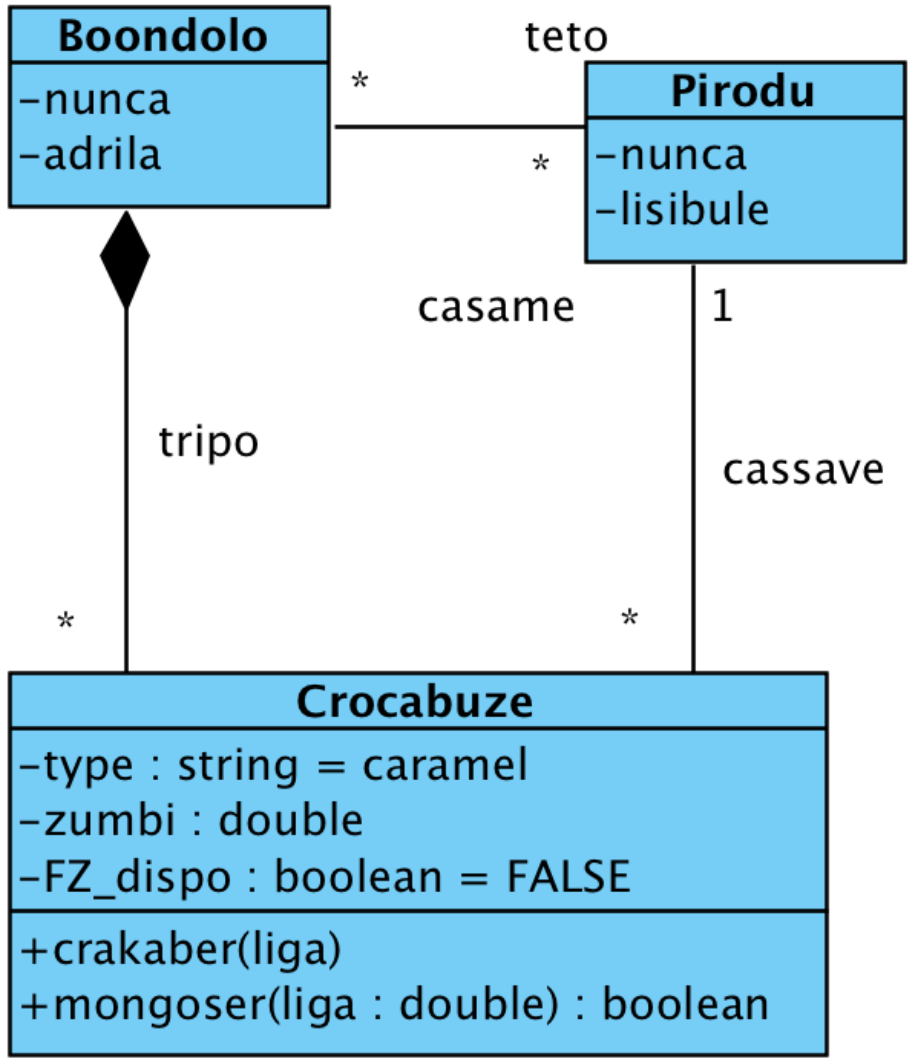
- Cours complet (doc 12 pages) : http://remy-manu.no-ip.biz/UML/Cours/coursUML3.pdf



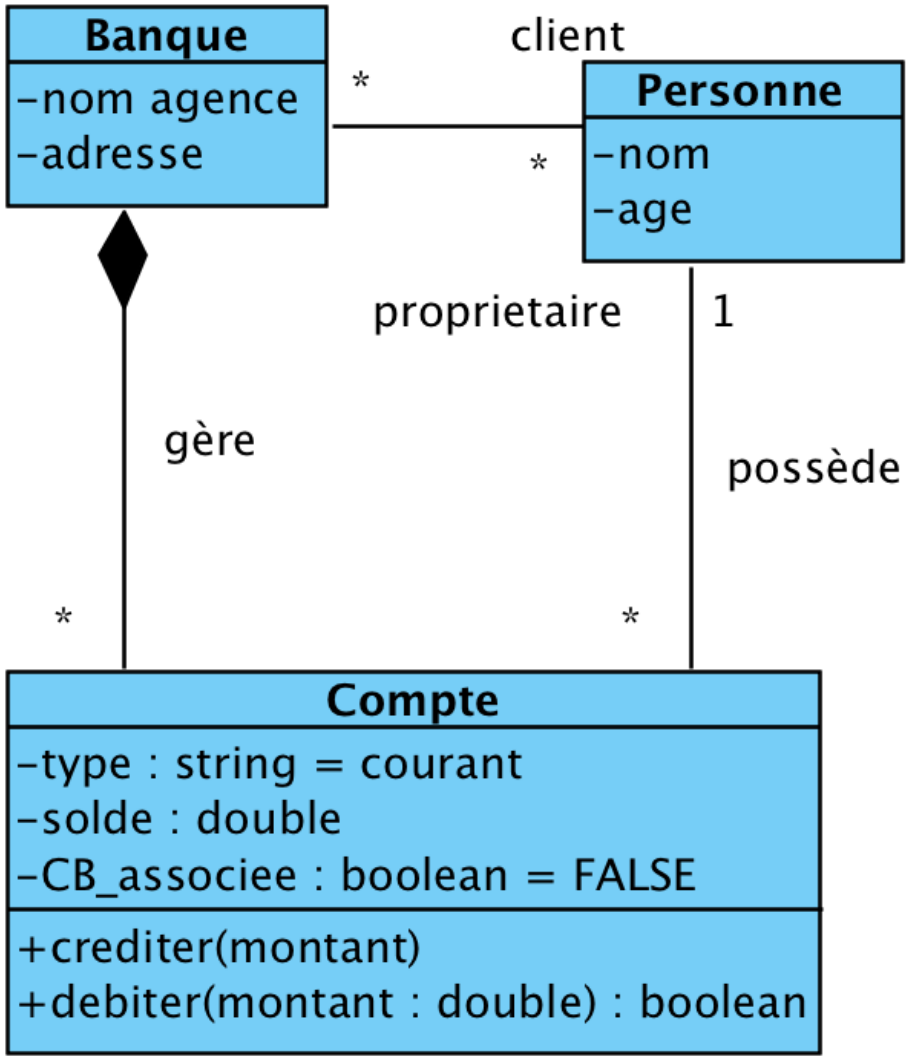
Modélisation des entités et de leur structure

DIAGRAMME DE CLASSES

Que raconte ce diagramme ?



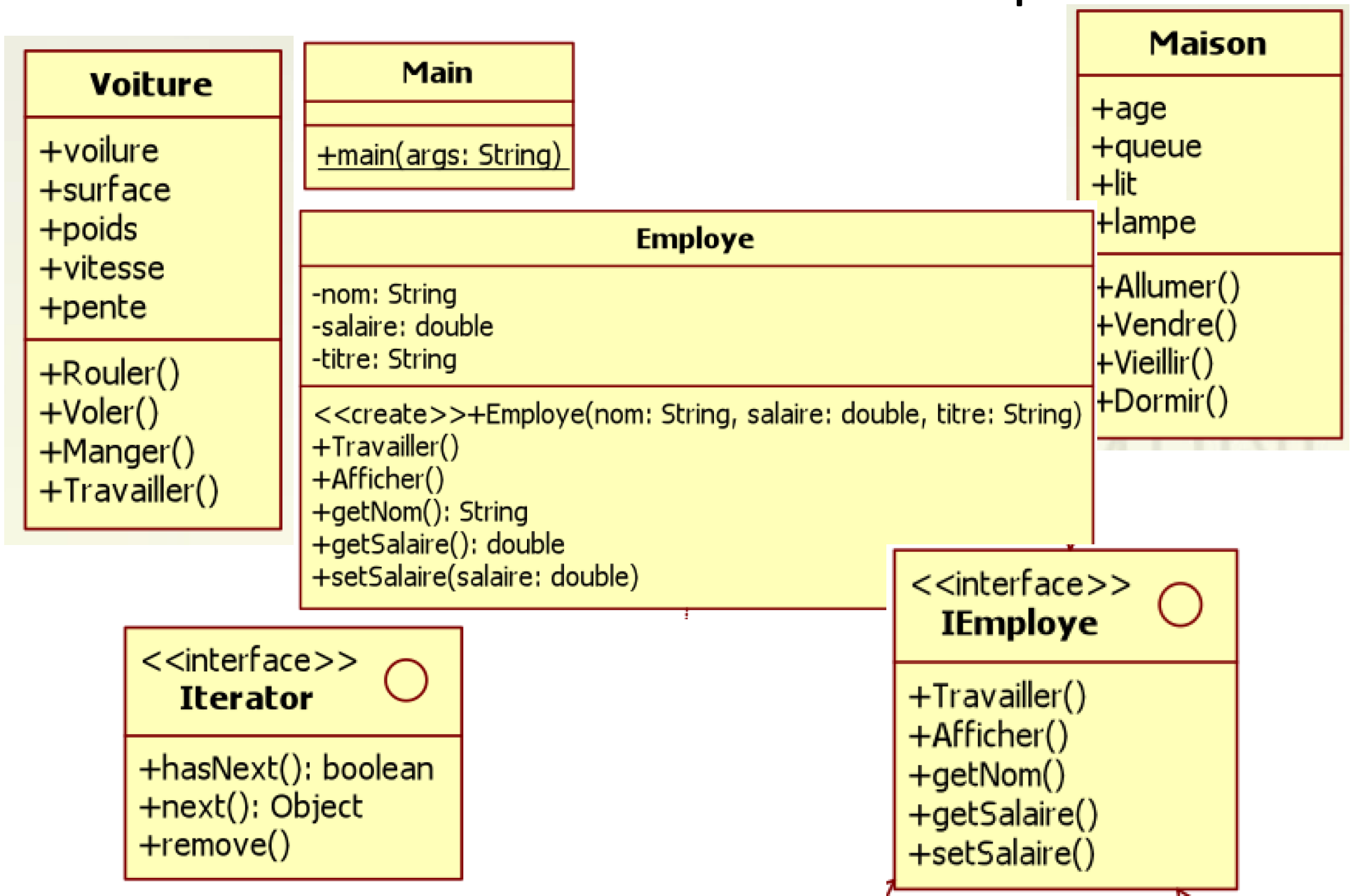
Ça va mieux !


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

Utilisation DCL

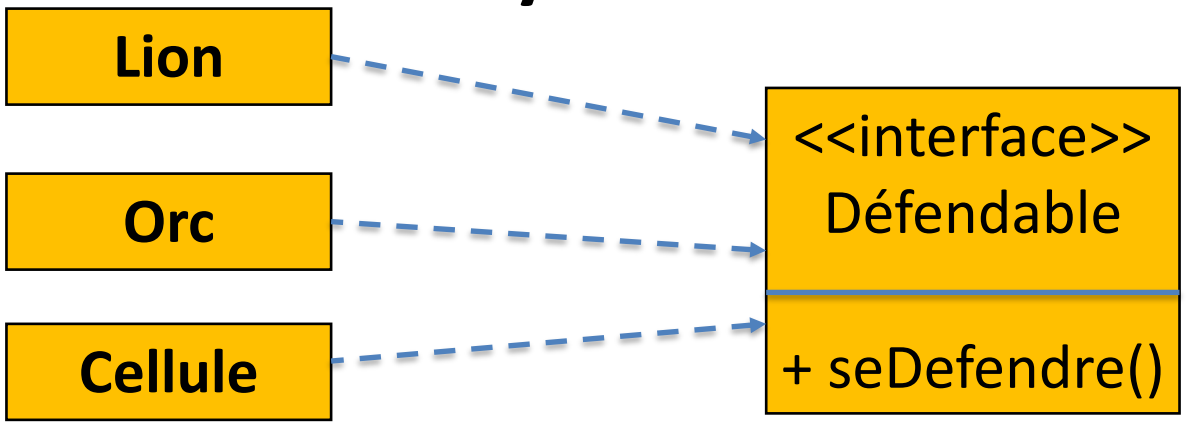
- Très nombreuses utilisations, à différents niveaux :
  - Pendant la capture des besoins :
    - **Modèle du domaine**
    - Classes correspondant à des objets du domaine
  - Pendant la conception / implémentation :
    - **Modèle de conception**
    - Classes correspondant à des objets logiciels

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 statique** d'un système
- Représentation des **relations** qui existent entre les entités, **indépendamment de leur cycle de vie**



Kesaco, relations statiques ?

- Ex.: **un Contrat est associé à un Client**
- Synonyme : relations **structurelles**
  - Validité dans la durée, le long terme

- Le fait qu'un Contrat soit signé avec un Client et archivé par le Documentaliste relève de sa « dynamique ». On ne doit pas le modéliser comme une relation dans le DCL (mais par les opérations de la classe Contrat)
  - On ne décrit jamais QUI fait QUOI dans un DCL

- 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 » ?

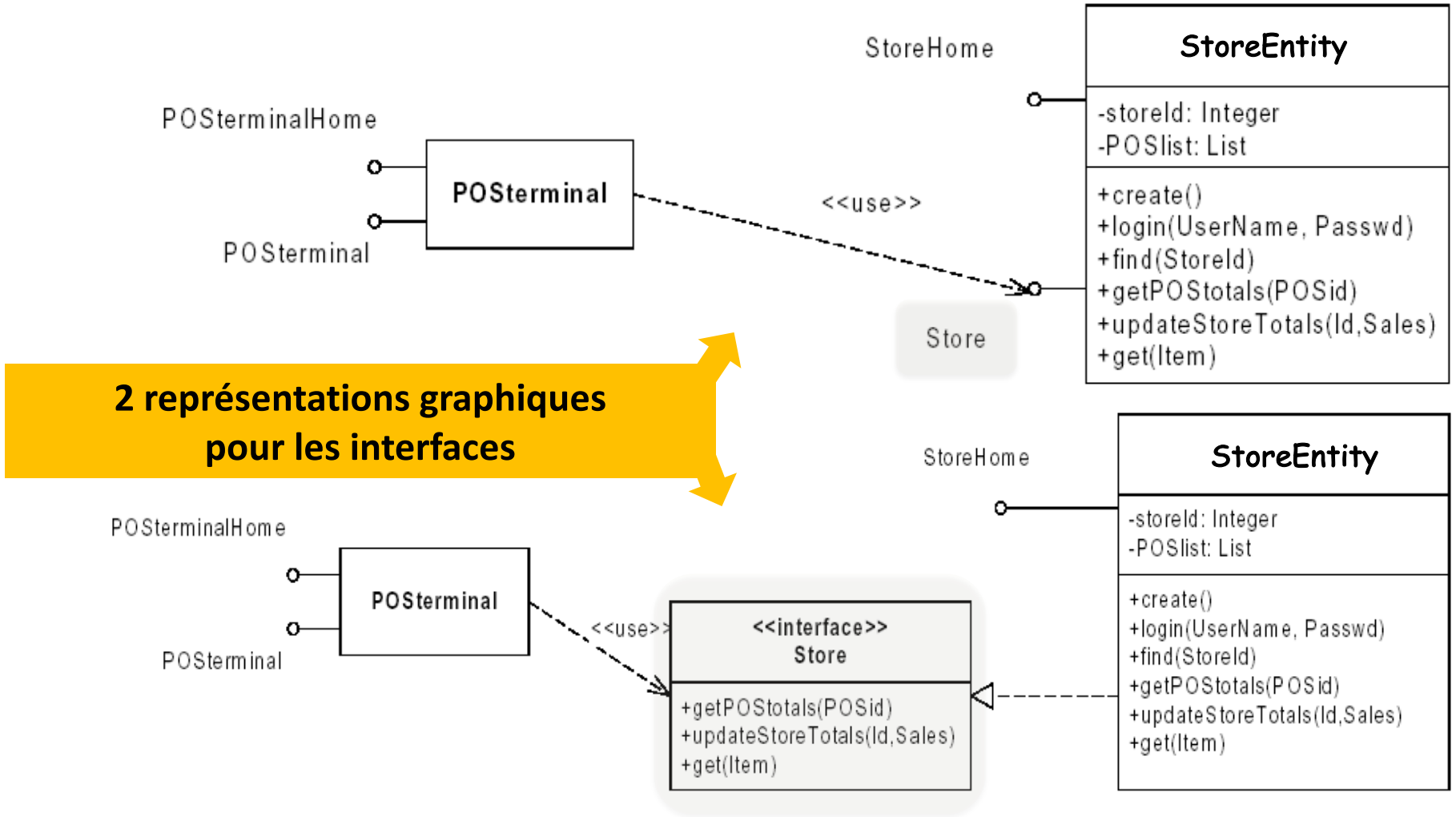
- Exemple avec un Système de Gestion Aéroport :
  - Si on envisage les classes **Avion, Compagnie** et **TourContrôle, Piste, Tarmac** … relations structurelles ?

Relation « structurelle » ?

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

- Interface : spécification des méthodes, sans indiquer comment les réaliser
- Exprime un comportement : Clonable

Implémentation de 2 interfaces



Les relations entre classes

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

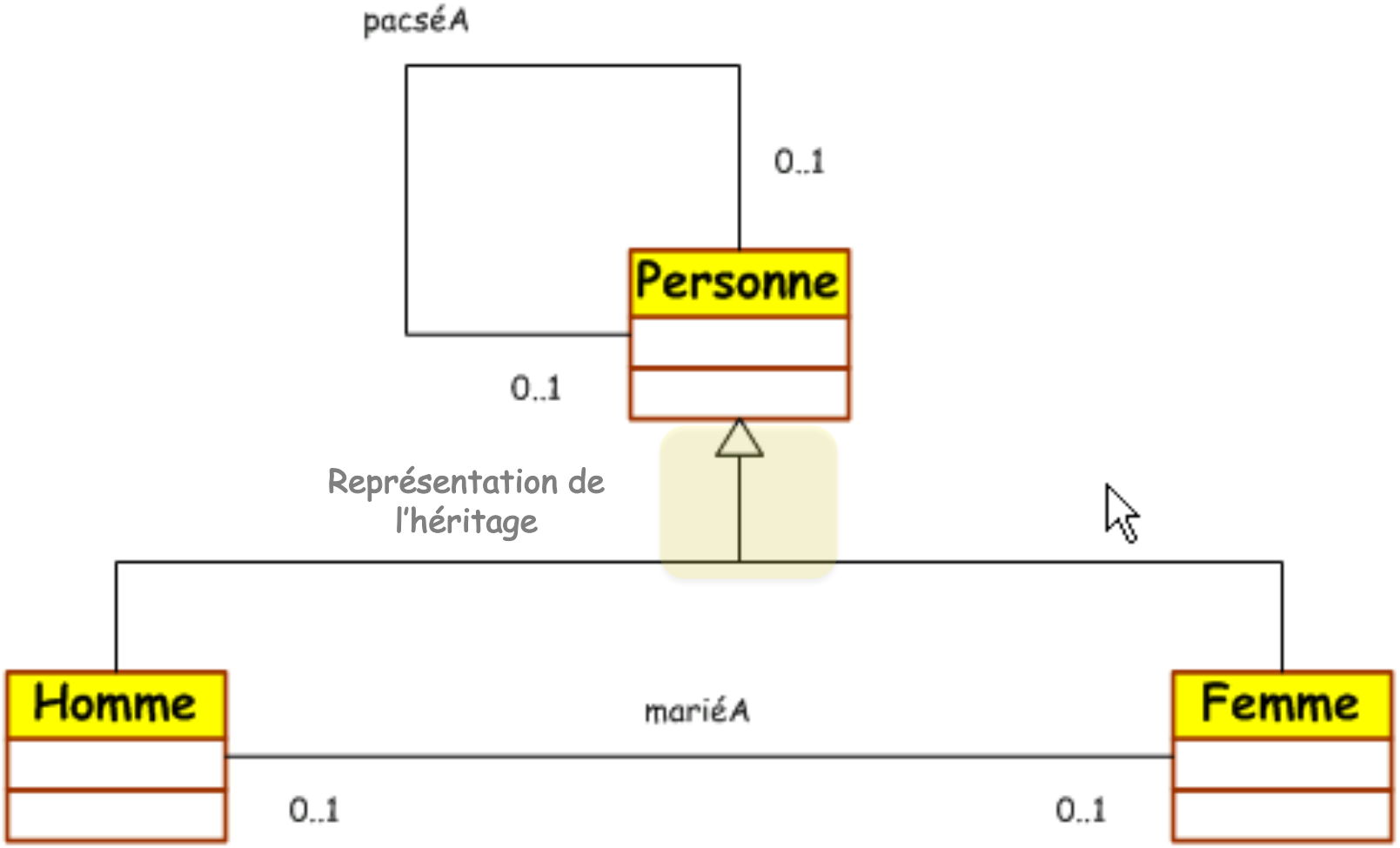
L'association **exprime une connexion sémantique *bidirectionnelle* entre classes**

= abstraction des liens qui existent entre les instances

Association nommée avec un sens :

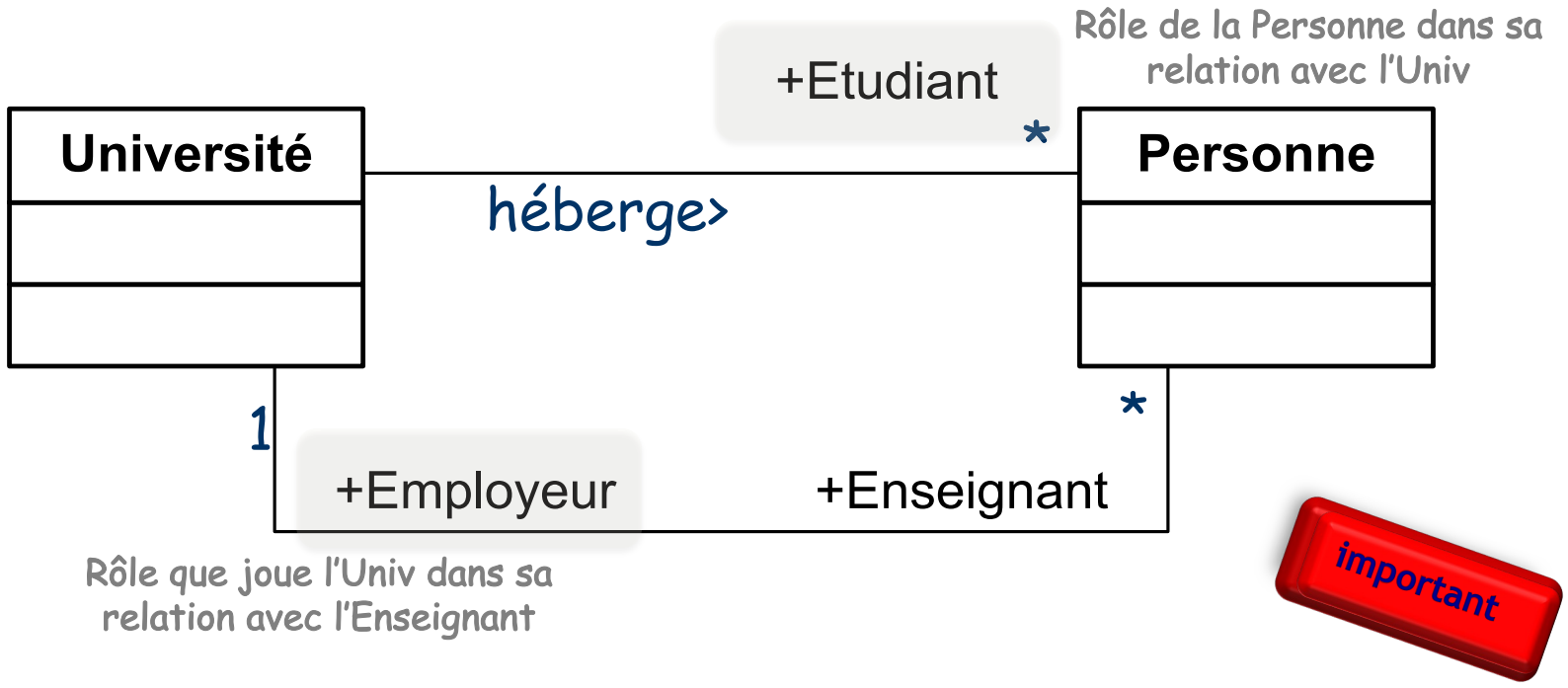
Exemple : associations et héritage

(avant le mariage pour tous)



Rôle des classes

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

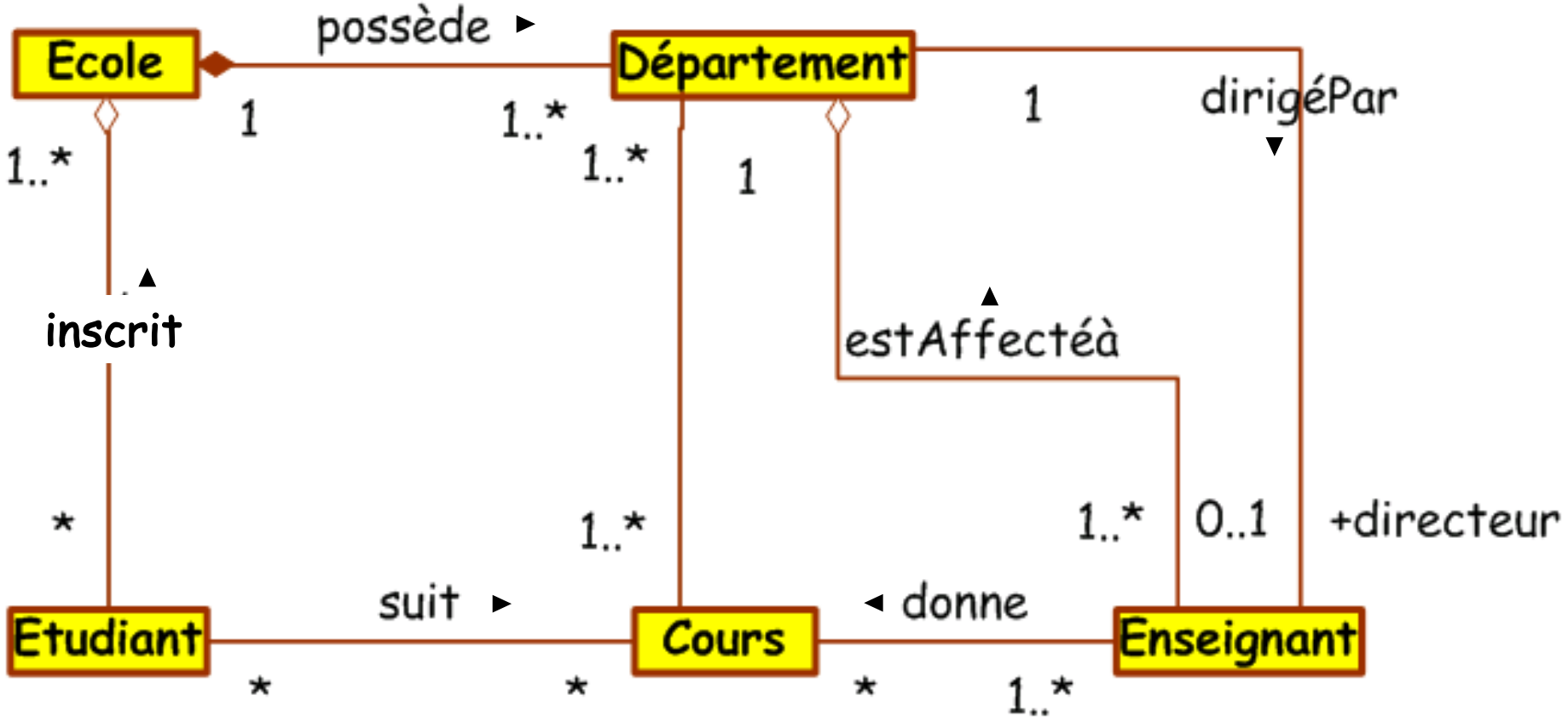


Il peut y avoir plus d'une association entre 2 classes

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 : école d'ingénieurs



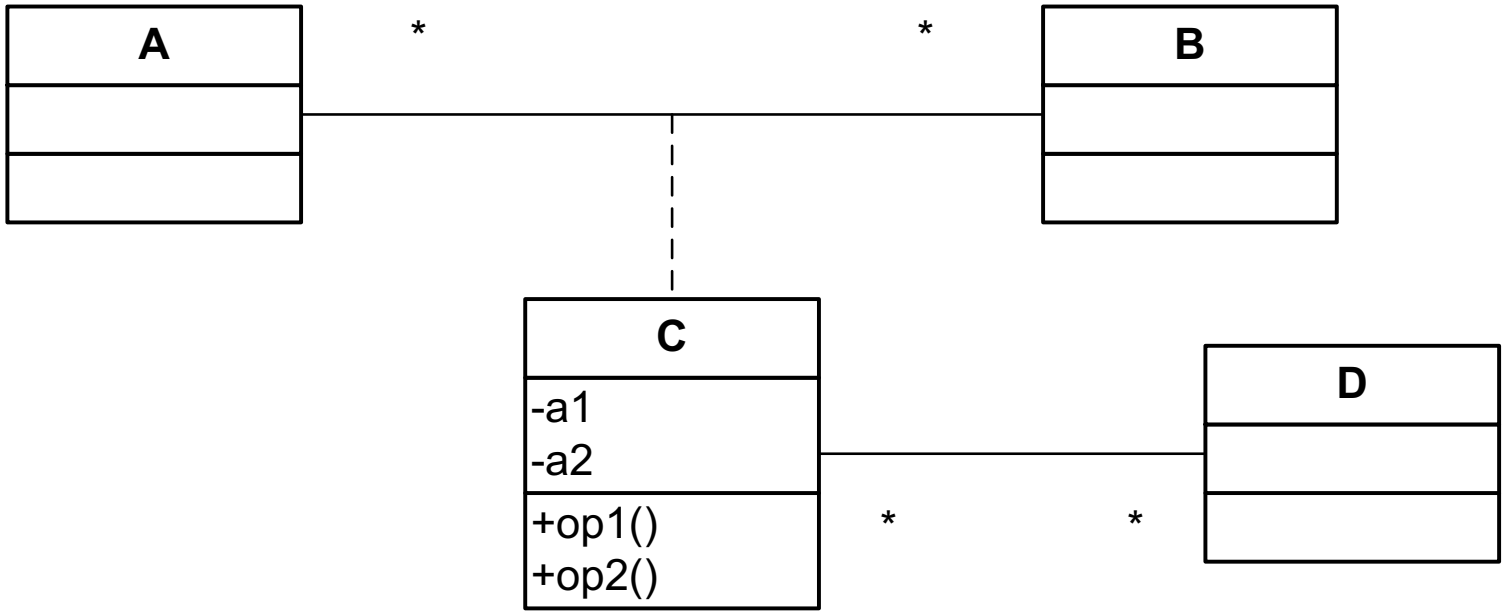
Navigabilité : traduit le sens de parcours d'une association

- 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

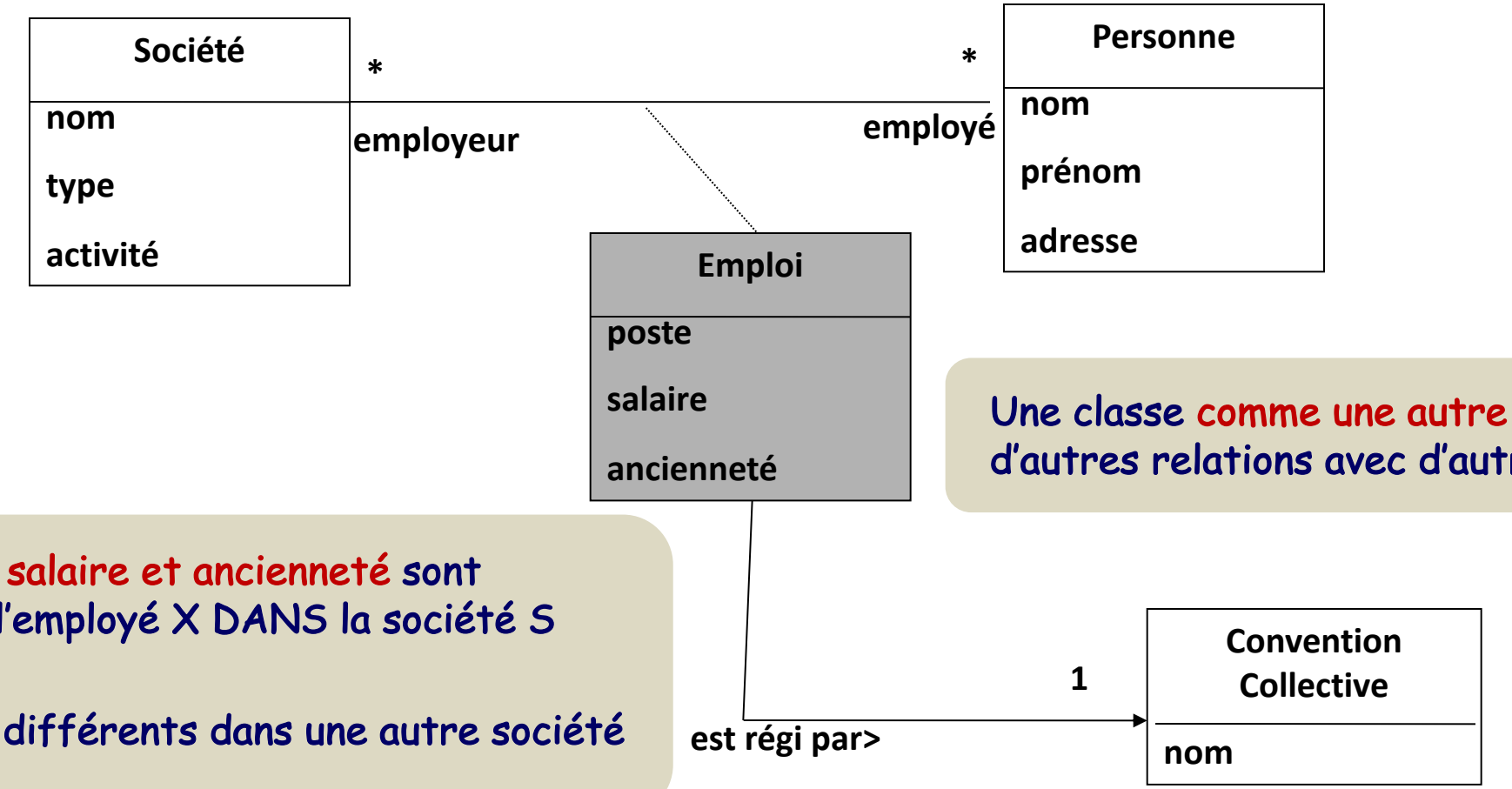
Les classes-associations

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



Ajout d'attributs ou d'opérations à la relation

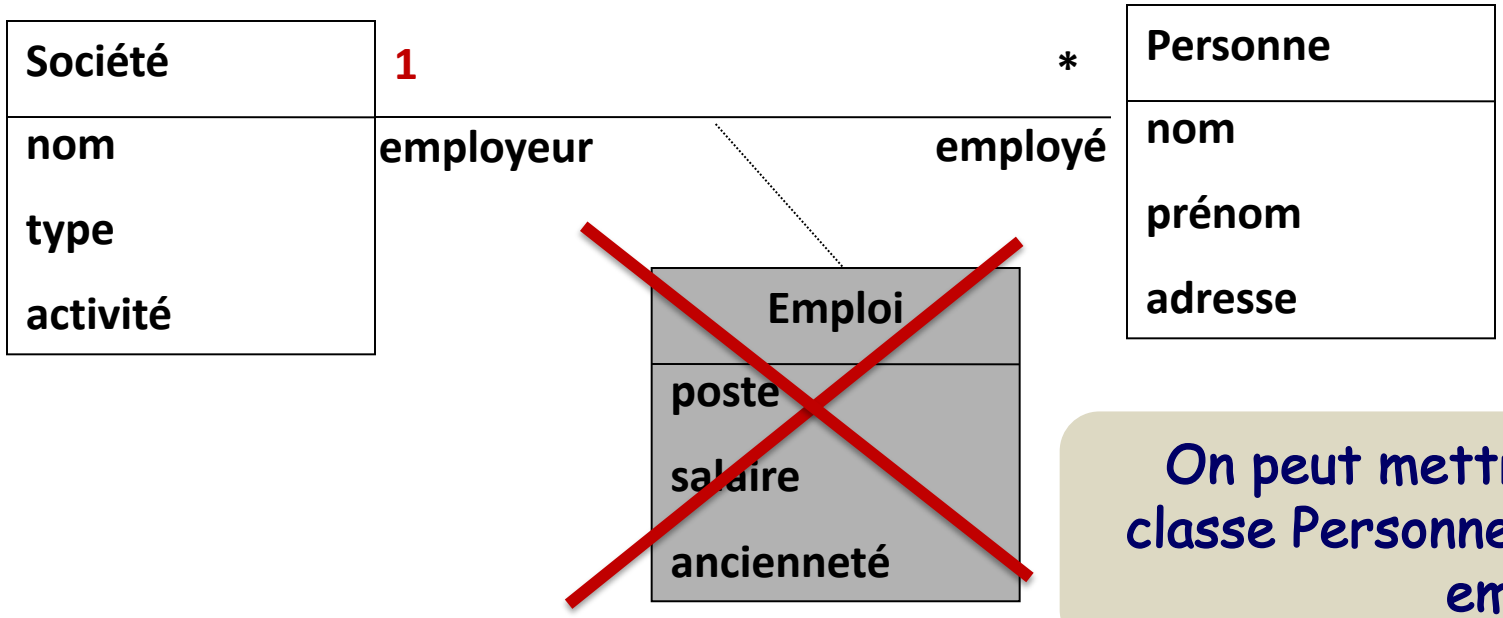
Exemple de Classe d'association



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

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

ERREUR de Classe d'association

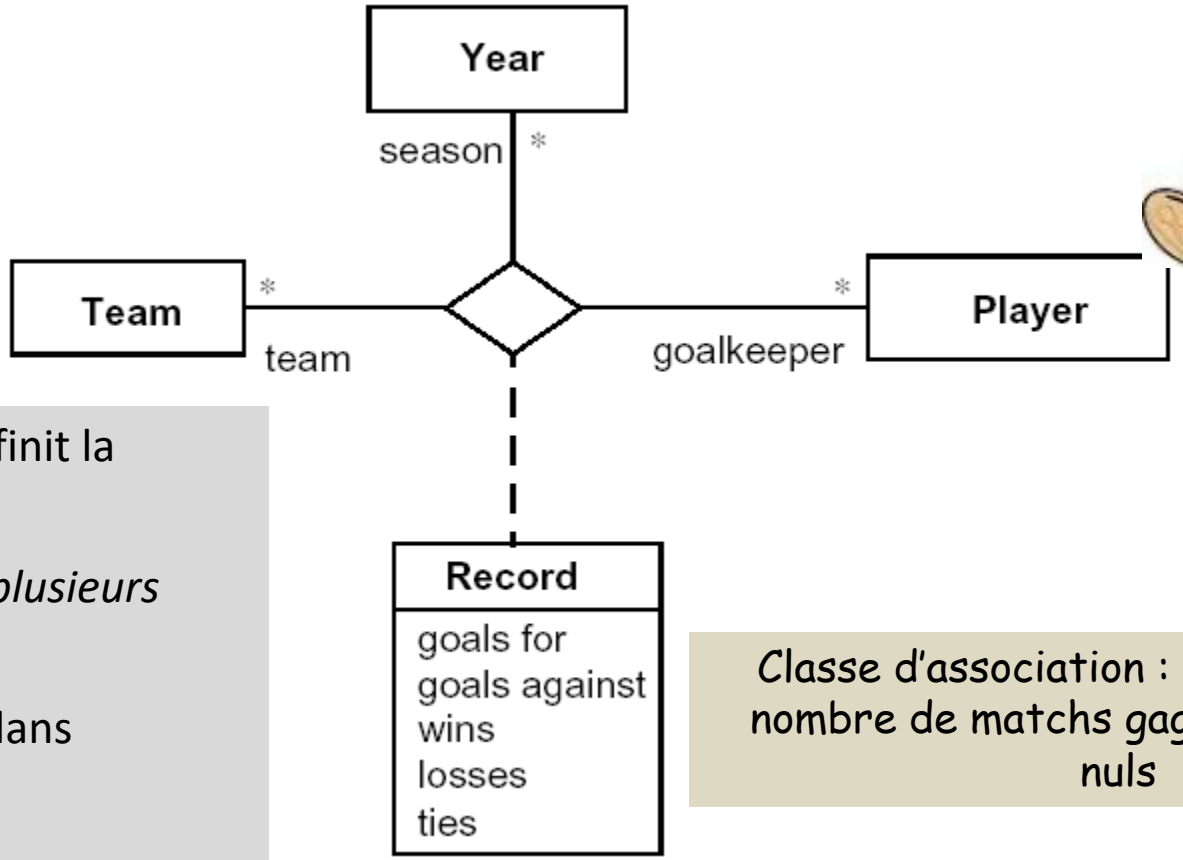


On peut mettre ces infos dans la classe Personne puisqu'en lien avec 1 employeur

Classe d'association : uniquement sur des relations *n,n*

Associations ternaires (et plus)

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

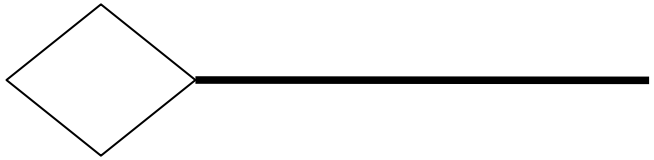


Multiplicité : on prend un couple et on définit la cardinalité avec la 3e classe. Ainsi ici :

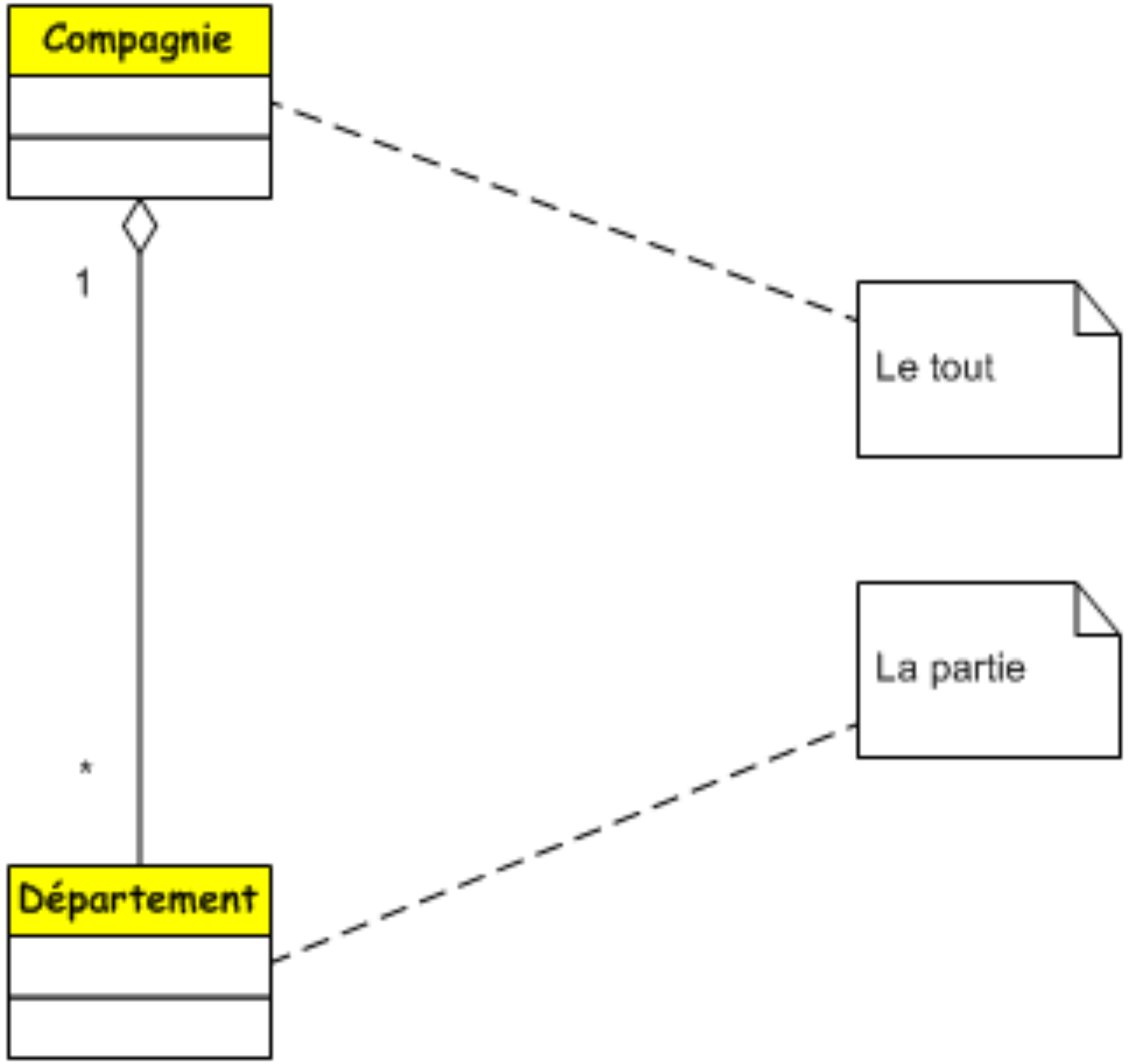
- Sur une année, une équipe peut avoir *plusieurs* lanceurs
- Sur une année, un lanceur peut jouer dans *plusieurs* équipes
- Un lanceur peut jouer *plusieurs* années avec une même équipe

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

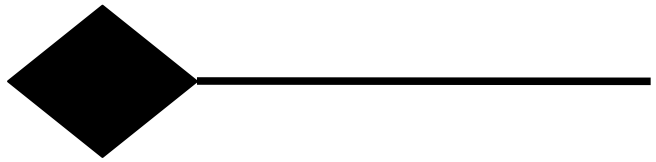
L'agrégation



- Connexions bidirectionnelles antisymétriques
- Représentation des relations traduisant :
  - maître et esclaves
  - tout et parties
  - composé et composant
- Exprime un couplage entre deux classes

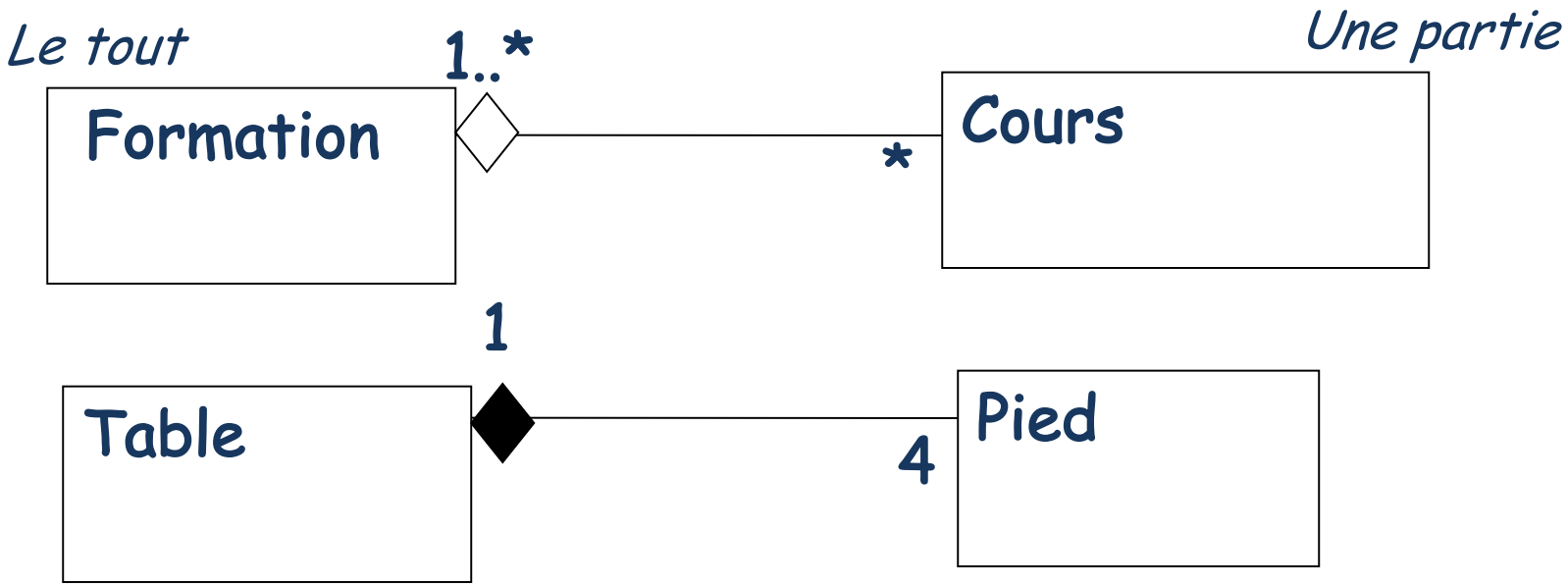


La composition



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

Agrégation et composition : 2 relations dissymétriques



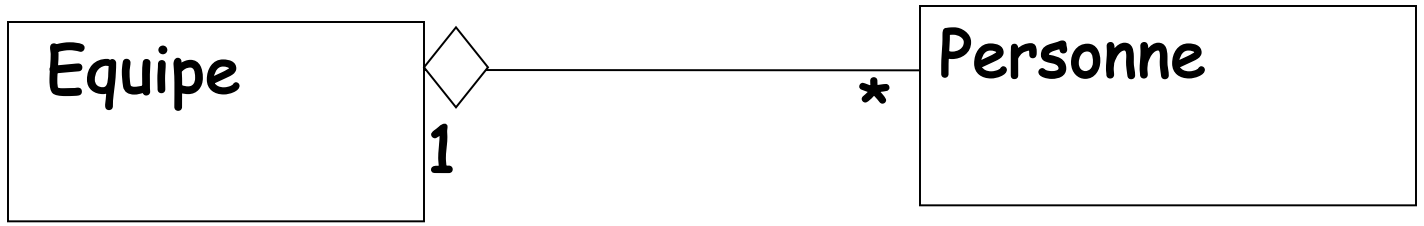
Agrégation : la partie peut être partagée entre plusieurs entités, les cycles de vie des instances ne sont pas imbriqués

Quand on supprime la formation, le cours peut encore exister.

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 **échiquier** est composé de 64 **cases**
- La **fenêtre** possède un bouton **Valider** et un bouton **Annuler**
- Une **page web** possède une **feuille de styles**

Implémentation d'une agrégation et composition

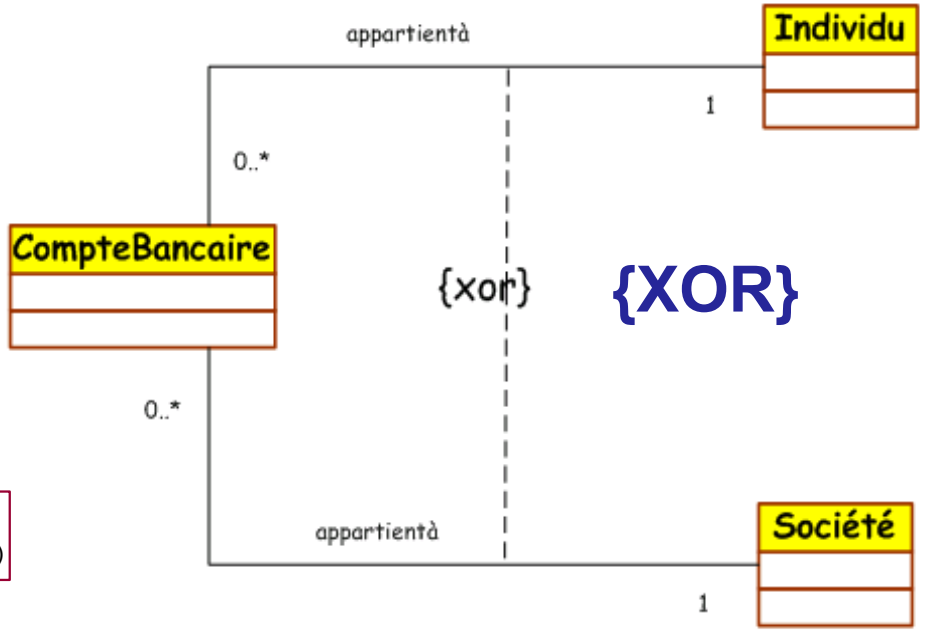
Exemple précédent de l'école

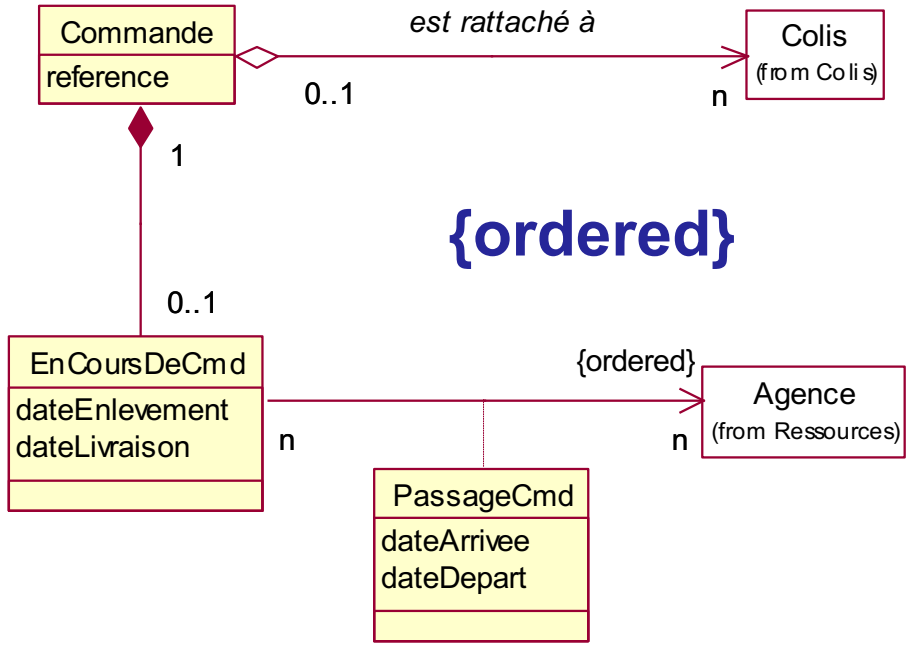
```
public class Ecole {
   private List<Departement> listDepartements;
   private List<Etudiant> listEtudiants;

   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 */
    }

}
```

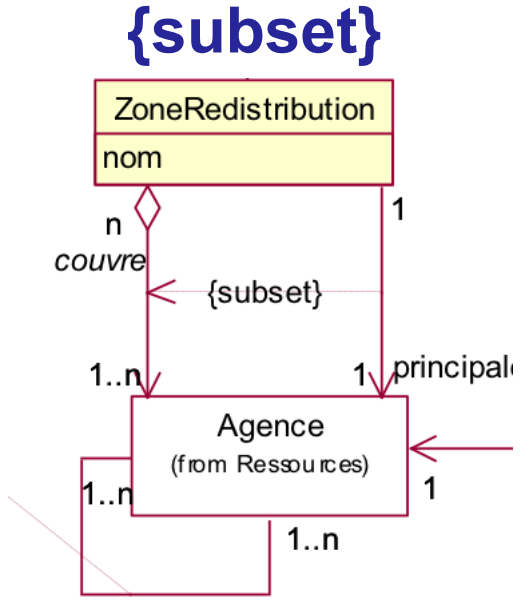
composition

agrégation

Contraintes sur les associations

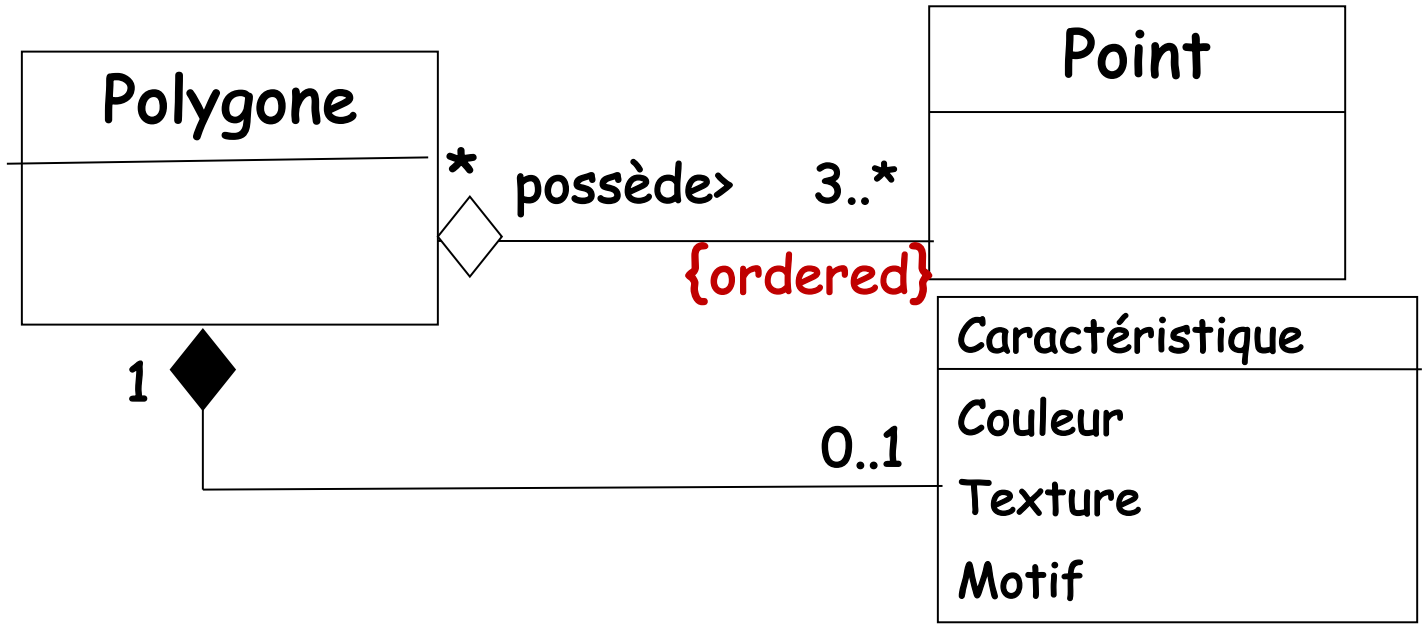






Exemple de contrainte d'association

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

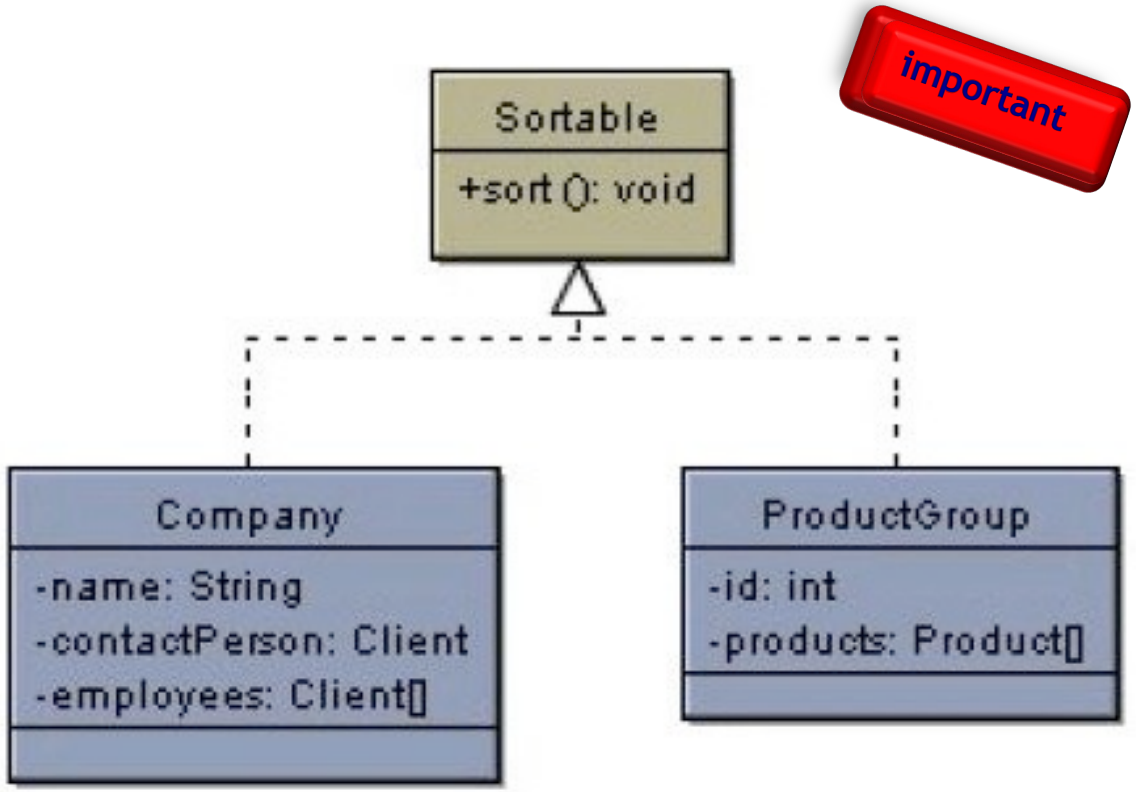


Relation d'implémentation

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

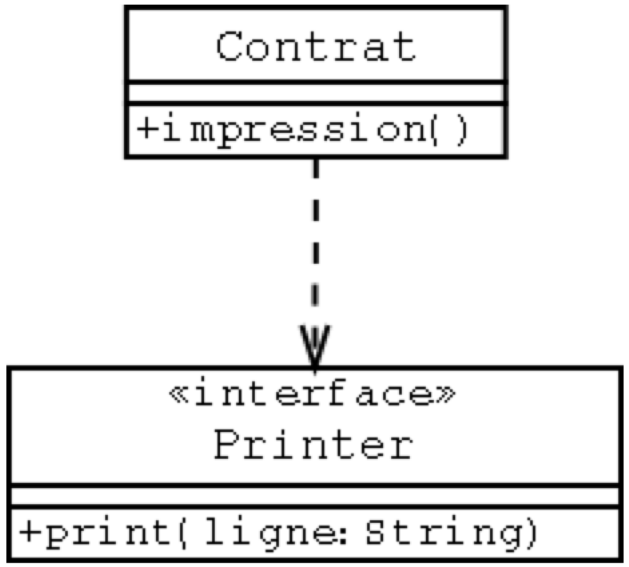
Trier des employés

Trier des produits



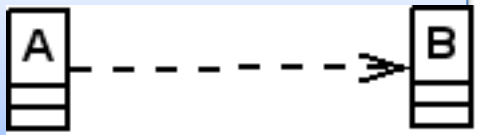
Relation de dépendance

- **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

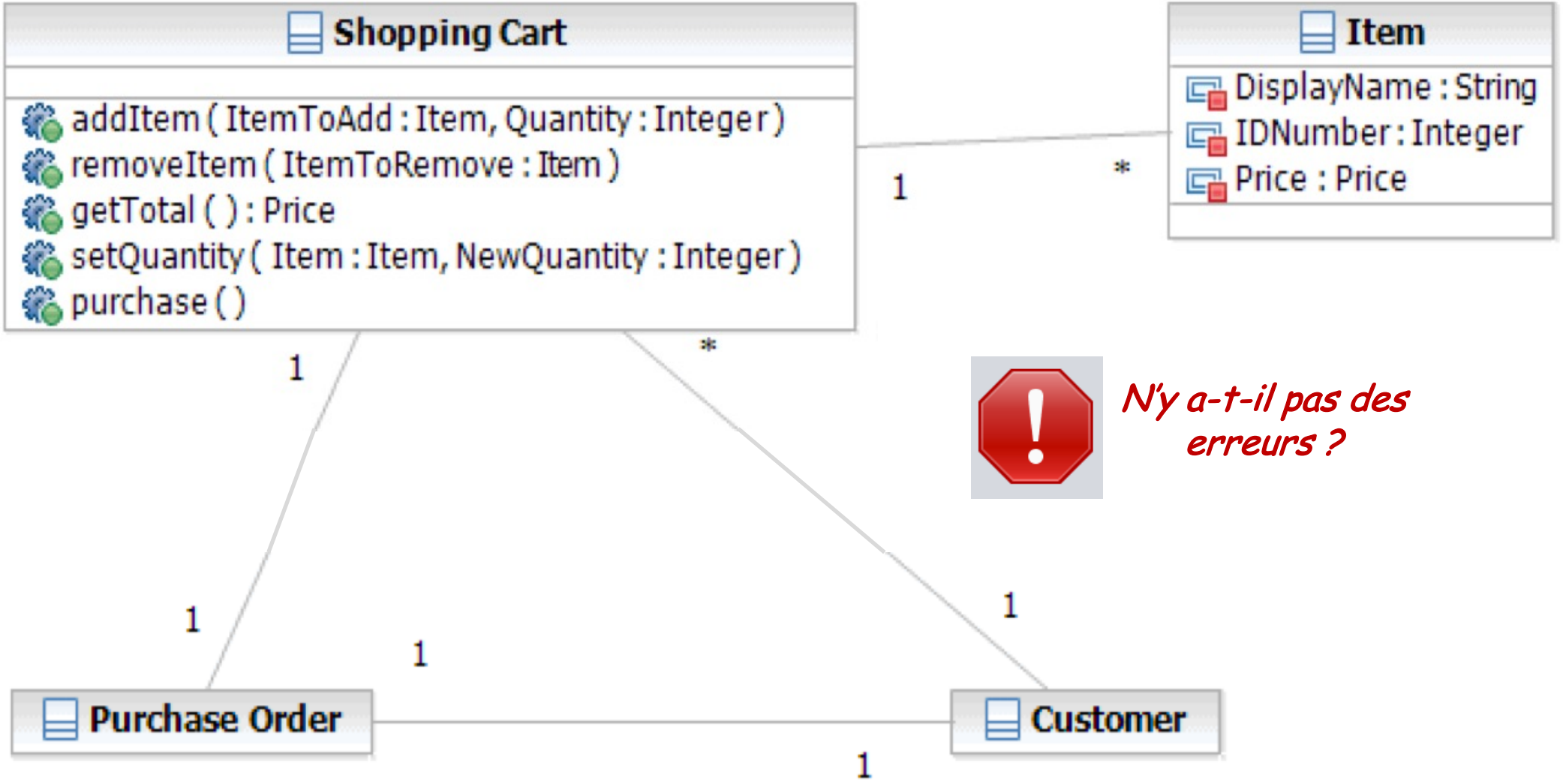
S'applique à:

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

'n cherchera à limiter ces épendances, pour éviter le plat de spaghettis

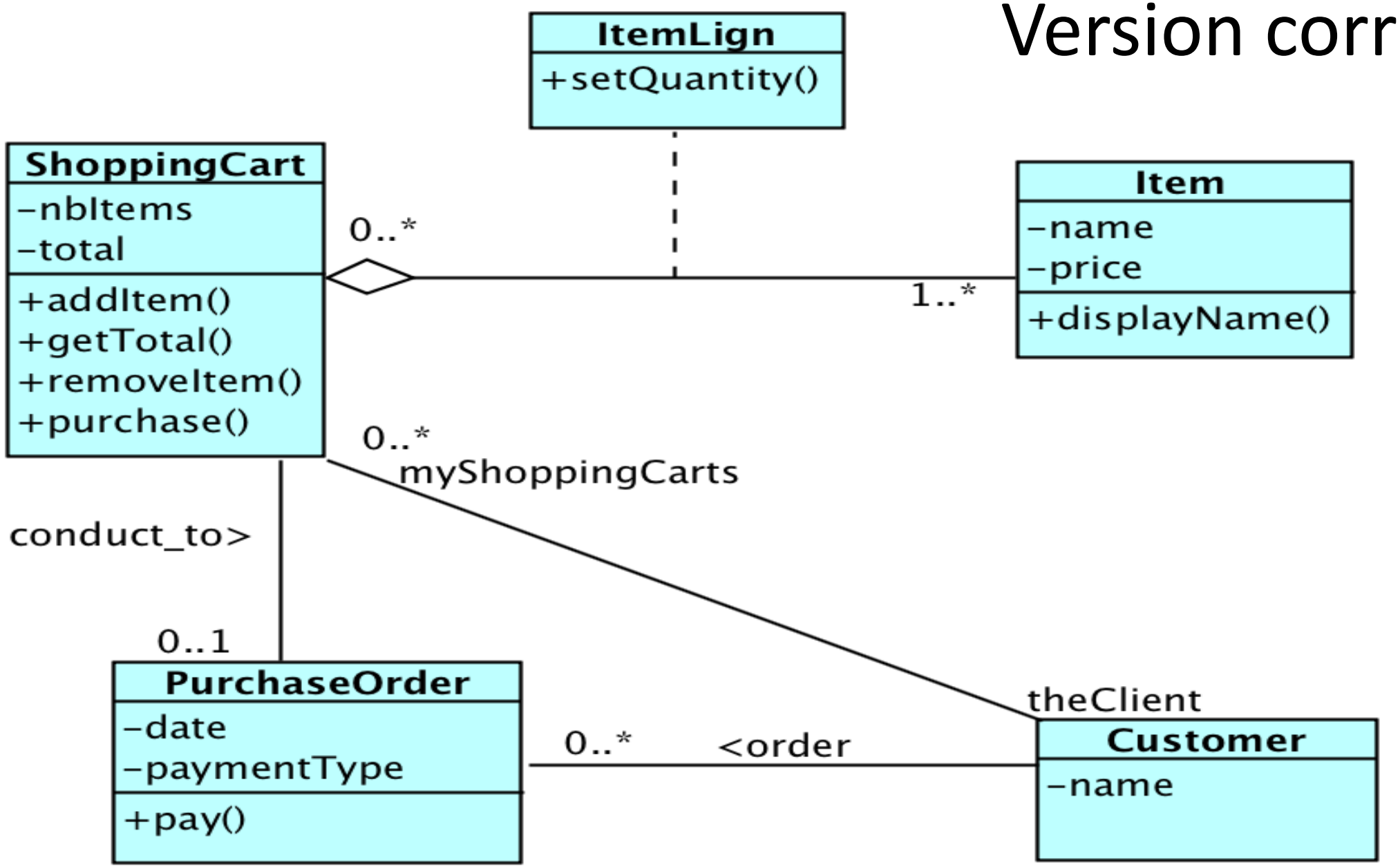
Que raconte ce diagramme

(trouvé sur le cloud) ?



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 sur les relations n,n**
- Les **contraintes** d'associations : *ordered, subset, XOR*