


# Qu'est ce que le génie logiciel ?

(source : wikipédia)



l'ensemble des activités de **conception**, de mise en œuvre des produits et des procédures tendant à rationaliser la **production** du logiciel et son **suivi** *de façon plus pratique :*

procédures, méthodes, langages, ateliers, imposés ou préconisés par des **normes** adaptées à l'environnement d'utilisation, afin de favoriser la **production et la maintenance** de composants logiciels de qualité



# **INTRODUCTION**

# 1. Pourquoi la modélisation du domaine?



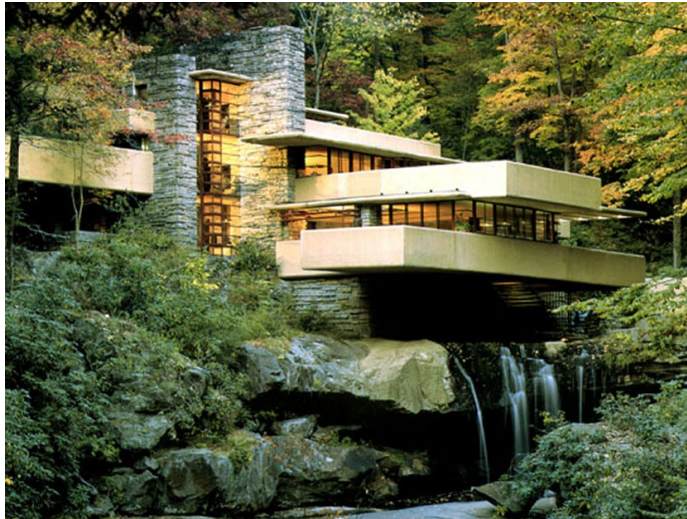
Dans une approche de développement tout objet, un logiciel va être décrit (Analyse) puis construit (Conception) sous la forme d'une collection d'objets collaborant.

La philosophie est qu'un programme est structuré dans un style dans lequel :

chaque cas d'utilisation est résolu par un groupe d'objets  
chaque objet prend en charge une partie cohérente des traitements se retrouvant dans un ou plusieurs cas.

Les objets utilisés lors du développement pour décrire l'application sont une abstraction des « objets » du monde réel concernés par la future application.

Objets du monde réel



Objets informatiques

UneMaison

UneCascade

UnEtage

Abstraction



Une *abstraction* est un résumé, un condensé.

C'est un des moyens utilisés par l'homme pour gérer la complexité du monde qui l'entoure.

Elle vise à *dégager les seules caractéristiques essentielles au problème posé* en ignorant les détails sans intérêt.

Une abstraction se définit par rapport à un point de vue.

<u>: unObjet</u>
prénom = « Falbala » date_naissance = 29/01/1993 adresse = 54 rue Pennec tel = 02-97-50-50-23
getTel() ..

Point de vue  
« Camarade »

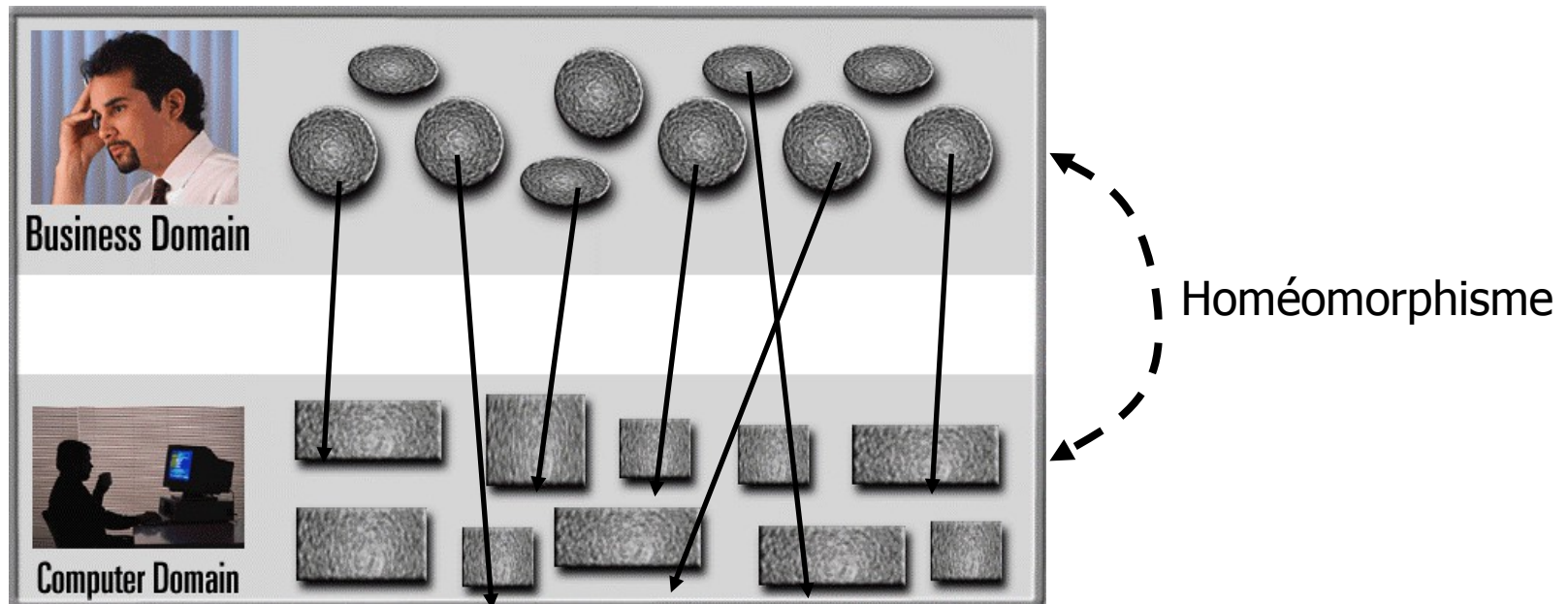
<u>: unObjet</u>
nom = « Balbala » origine = « STI » Redoublant = FALSE noteOMGL1.1 = 18
noter(note : int) ...

Point de vue « Professeur »



Pourquoi chercher à « mapper » la structure du logiciel sur la structure du domaine métier?

On promeut ainsi une sorte d'homéomorphisme (application bijective) entre la structure conceptuelle du monde réel concerné par l'applcatif et la structure interne du logiciel.





Cette notion d'homéomorphisme est cependant partielle...

Mais elle trouve sa quintessence dans le développement de la partie persistante d'une application (**Base De Données**).

Les classes (POO) deviennent des schémas relationnels (BDD)

Les associations (POO) deviennent des clés étrangères où des tables association (BDD)

Cf. la deuxième partie de ce module à partir de novembre





Cet homéomorphisme facilite :

La recherche d'une solution

La compréhension de la structure d'une application

La localisation des changements


L'évaluation des coûts d'une modification

## 2. L'analyse du domaine dans le cycle de vie




**La première étape** d'un développement consiste à établir un *modèle du domaine* pour identifier les objets du monde réel susceptibles d'être utilisés lors du développement et de la conception de la BDD

un bon développeur doit savoir construire un tel modèle (cette première partie de module !)



**La deuxième étape** consiste à établir les exigences fonctionnelles (cas d 'utilisation) et non fonctionnelles (Interfaces Homme/Machine, Attributs de qualité, etc.)  
cf Module de GL en semestre 2.



**La troisième étape** (Analyse) consiste à réécrire les scénarii des cas en assignant à certains objets du modèle de domaine des parties de comportement (des responsabilités).

Les objets se voient affectés des compétences dont ils sont les « fournisseurs ». Ils utilisent éventuellement pour les réaliser les compétences d'autres objets dont ils sont les « clients ».

La collaboration des objets ainsi définie doit reproduire les scénarii.

(module GL en semestre 3)



**A ce stade les objets sont encore conceptuels, de purs objets UML.**

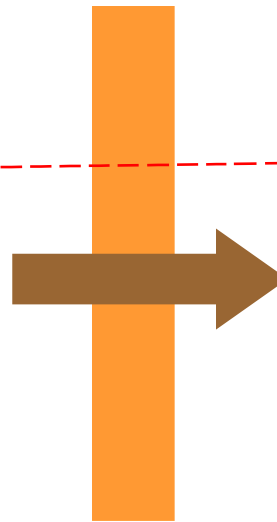
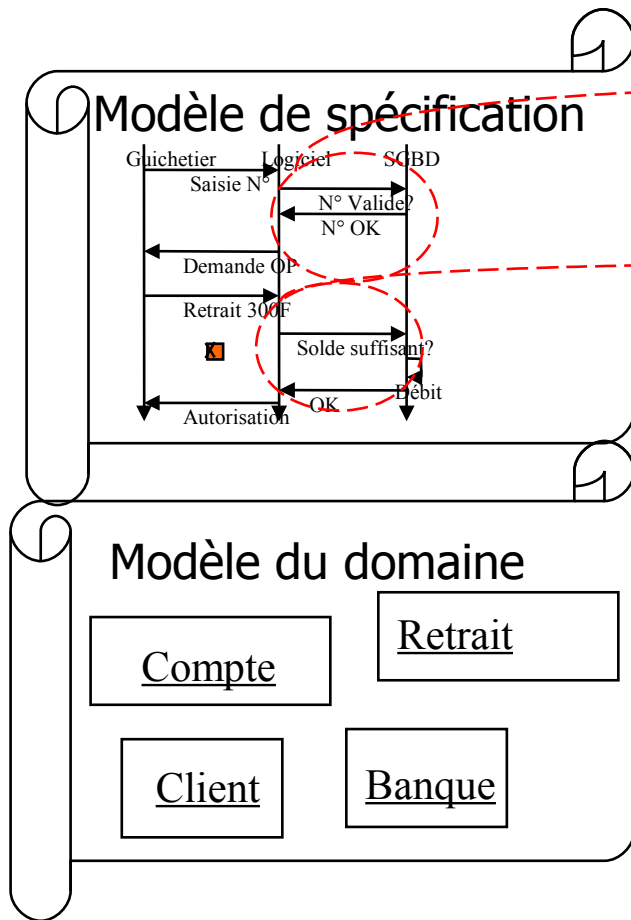
**Ils ne font en particulier l'hypothèse d'aucune technologie de programmation, d'interfaçage homme-machine, de systèmes supports.**



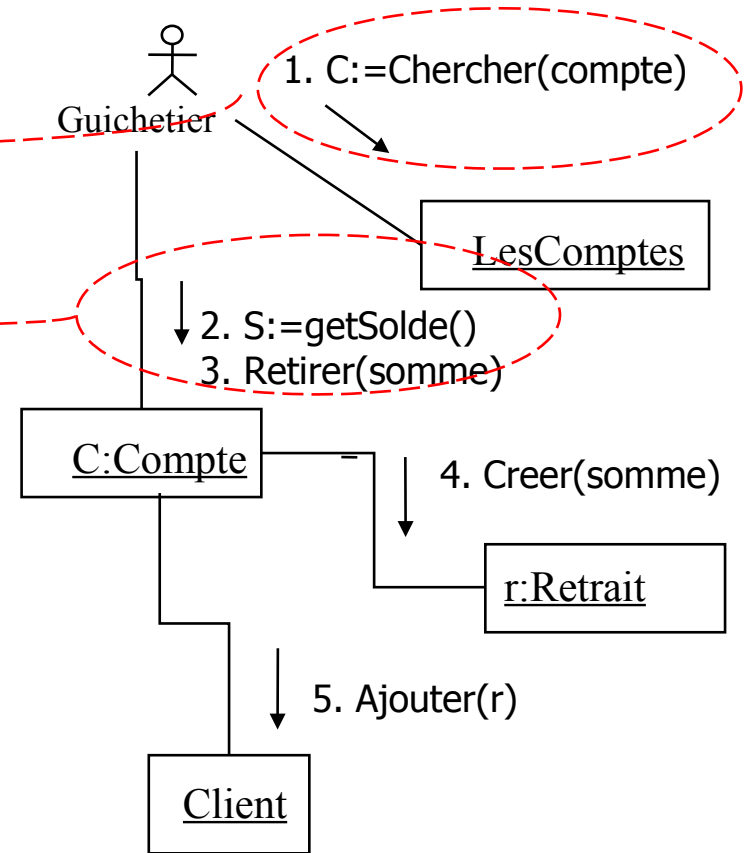
Cette étape difficile produit le *modèle d'analyse*.


Le cœur de cette étape est l'assignation de responsabilité. Cette activité est d'une importance capitale car elle fixe nombre des propriétés non fonctionnelles de la future application.

# La troisième étape : (OMGL4)



Analyse





Dans une quatrième étape, Les objets  
(conceptuels) identifiés en analyse vont  
devoir être implantés dans des « entités »  
informatiques exécutables :

des objets au sens de la POO

des tuples au sens des SGBDR

des fonctions, procédures, variables au sens de la PI  
( Programmation Impérative )





L'architecture d 'analyse s 'enrichit et se transforme :

des objets purement informatique apparaissent : piles, listes, tableaux, des objets de bibliothèques (IHM, pilotes, etc.).

des types sont utilisés ou construits

des liens entre objets sont créés ou défaits

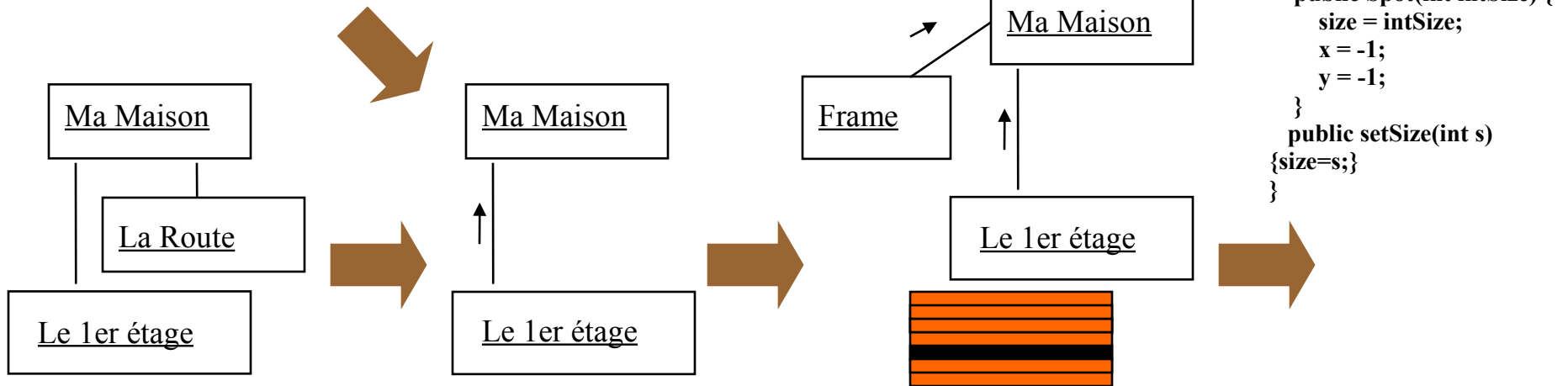
etc...

On obtient alors un modèle tenant lieu de *carte* pour le code à écrire : le **modèle de conception**

Certaines compétences de cartographie de codes seront introduites en semestre 2.

# Au final, le cycle de vie (très simplifié) est :

Modèle de spécification OMGL2.1 semestre 2



Modèle du domaine  
(Objets UML  
conceptuels orientés  
domaine)

Modèle de Conception  
(Objets UML conformant  
au sens de la POO)


Modèle d'Analyse  
(Objets UML  
conceptuels orientés  
application avec des  
responsabilités)

Code  
(objets JAVA,  
C++, C#)

Semestre 1

Semestre 3

Semestre 2



Encore une fois, il est important de noter que le même paradigme et la même notation sont utilisés à toutes les étapes du cycle de vie.

Pourtant sémantiquement un objet UML d'un modèle de domaine s'interprète différemment d'un objet UML d'un modèle d'analyse, lui même différent d'un objet UML d'un modèle de conception.

# **Les modules de GL de 1ère année**

## **(PPN 2013)**



M 1104 : introduction aux Bases de Données

M 2104 : Bases de la conception orientée objet

M 2106 : Programmation et administration des bases de données



# M 1104 : Introduction aux BDD

## Le langage diagramme de classes pour la modélisation de domaine



*François Pouit, Matthieu LE LAIN, Anthony RIDARD*

2016



# Objectifs du Module



Connaître les notations UML *diagramme de classes* et *diagramme d'objets* utiles à la modélisation de domaine

association binaire

classe association

généralisation

associations n-aires

qualification

contraintes prédéfinies et utilisateurs

(Classe +attribut)



# **Chapitre 1**

## **Aperçu de la modélisation de Domaine**

# 1. diagrammes de classes et d'objets



## Diagramme de classes pour la MD

Pour décrire la structure statique d'un domaine en termes de classes et de relations entre ces classes.

Il n'est donc pas question ici de représenter la dynamique du domaine

## Diagramme d'objets pour la MD

Tout diagramme d'objets est l'**instance** d'un diagramme de classes. C'est une photographie à un instant  $t$  d'un système.

Pour expliquer par l'exemple un domaine complexe décrit dans un diagramme de classes (approche descendante, fréquente)

Pour identifier les classes et les relations (approche ascendante, rare)






Diagramme de classes		Diagramme d'objets	
Classe Attribut	Concept Propriété	Objet Attribut valué	entité Caractéristique
Association binaire n-aire Agrégation Composition	Relation	Lien binaire n-aire d'agrégation de composition	Connexion
Généralisation	Classification	Mise à plat des propriétés	
Contraintes		Restriction des structures possibles	

# Représentation d'une Classe

Etudiant
age : int adresse : String fc : boolean = false;
Etudiant(a:int,s:string) setAge(a:int) : void getAge() : int setFC(f:boolean):void

Etudiant

Etudiant
#Etudiant(a:int,s:string) -setAge(a:int) : void +getAge() : int ~setFC(f:boolean):void

Etudiant
+getAge() : int ...

Etudiant
-age : int +adresse : String ~fc : boolean = false;

La classe représente une abstraction d'un concept du domaine : voiture, trajectoire, idée, ...

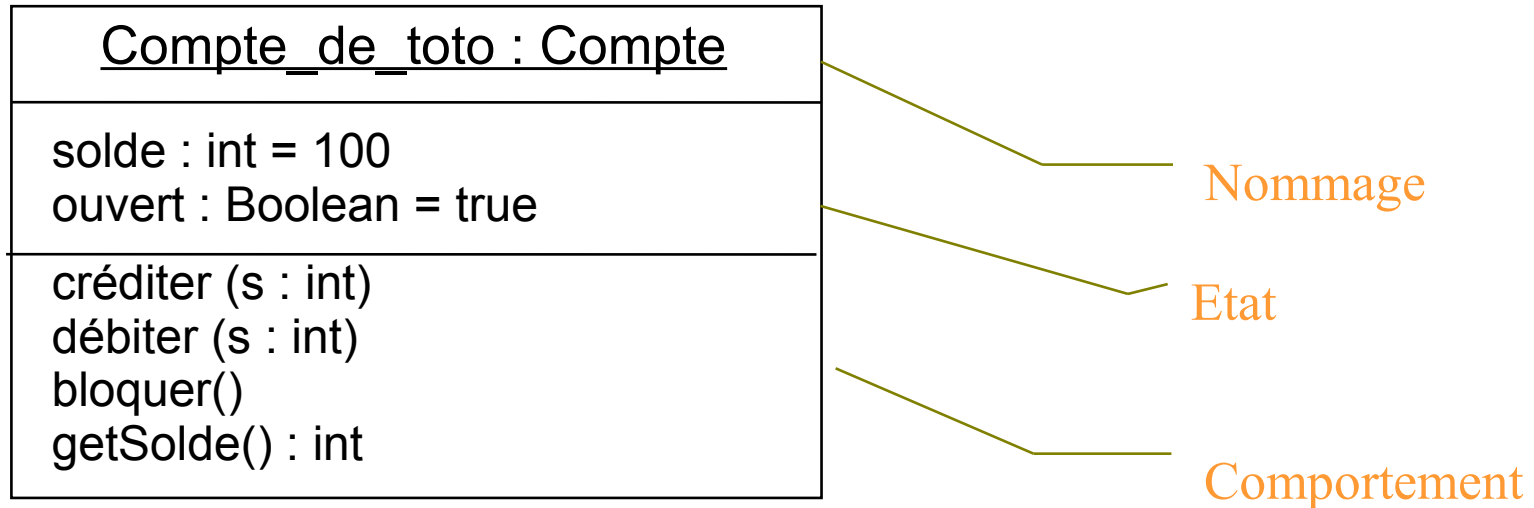
Une classe est un « moule » à partir duquel il est possible de créer des objets qui présenteront :

- les mêmes attributs (et valeurs par défaut) et
- les mêmes méthodes

La classe est donc un OUTIL facilitant la construction d'un nombre quelconque d'objets se ressemblant.

Le processus permettant d'obtenir un objet depuis une classe se nomme **instanciation**

# Représentation d'un Objet




Compte\_de\_toto

<u>Compte_de_toto</u>
solde = 100 état = TRUE

<u>Compte_de_toto</u>
créditer () ...

<u>Compte_de_toto</u>
créditer () débiter () bloquer()




Toutes les instances d'une même classe partagent des propriétés communes.

Elles ont les mêmes attributs et les mêmes méthodes

p1:Point
-x =100 -y = 50
+setX(a:int) +getX(): int +setY(a:int) +getY(): int

p2:Point
-x =10 -y = 10
+setX(a:int) +getX(): int +setY(a:int) +getY(): int

Point
-x : int -y : int
Point(a:int, b: int) +setX(a:int) +getX(): int +setY(a:int) +getY(): int



Compte
solde : int
créditer (s : Somme) débiter (s : Somme) Compte(s:int)

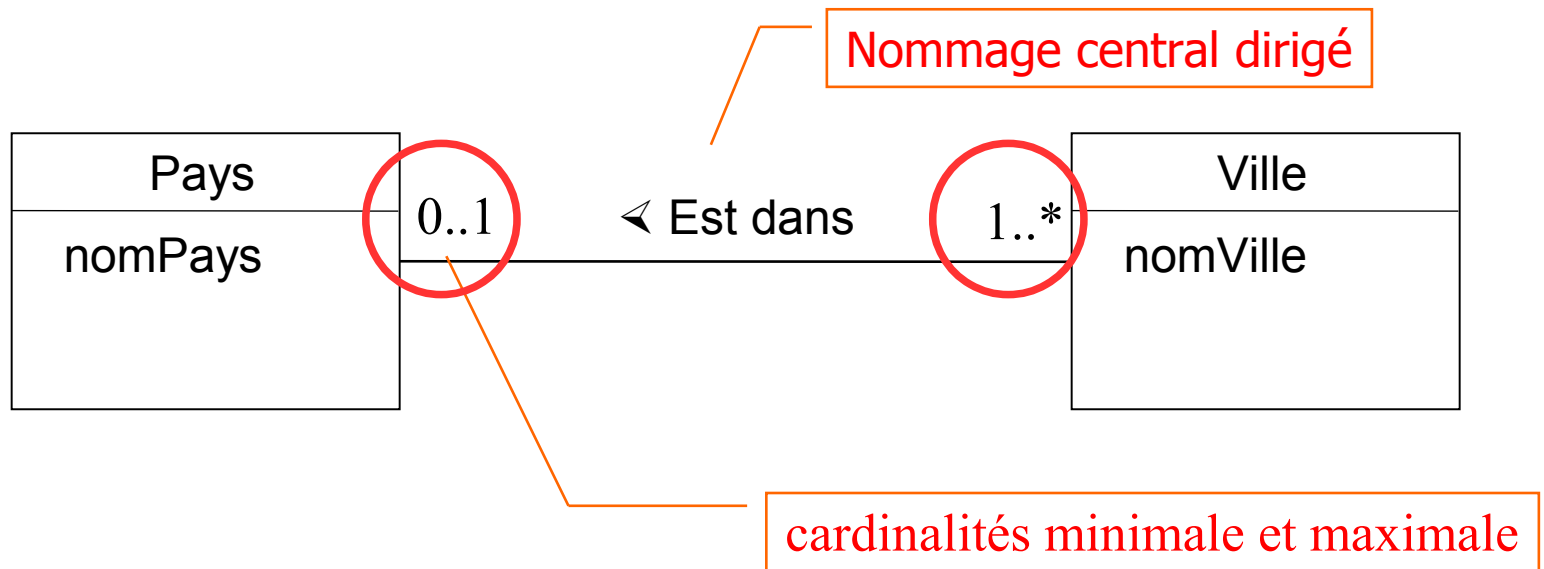
<u>: Compte</u>
solde =0
créditer () débiter ()

Anonyme

<u>c : Compte</u>
solde =10
créditer () débiter ()

Nommé

## 2. Les associations



Interprétation :




## Diagrammes d'objets compatibles





Diagrammes d'objets non compatibles



L'association exprime, en analyse,  
l'existence d'une connexion sémantique  
entre des concepts du domaine

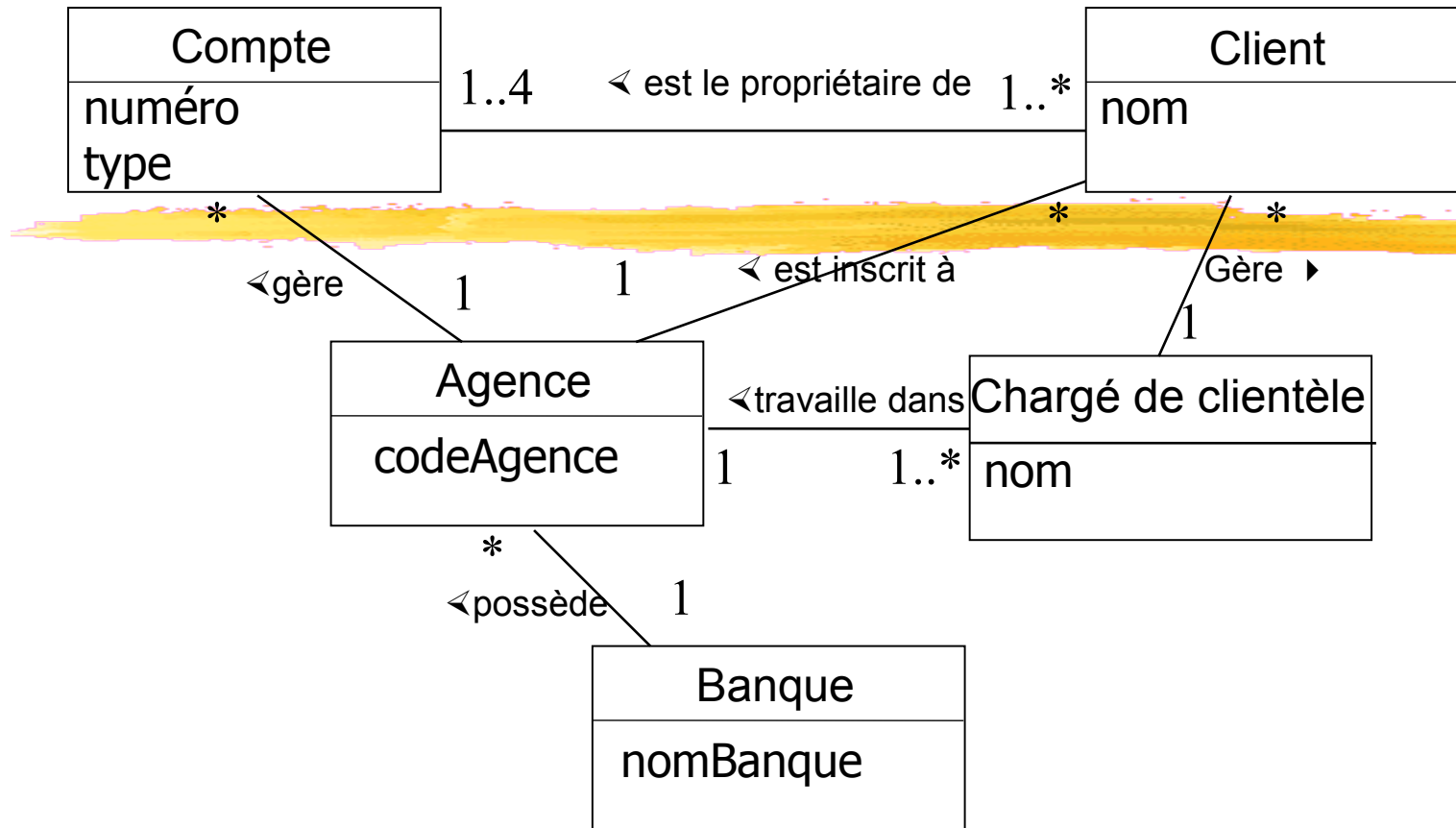
Une association est une abstraction des  
liens qui existent entre les entités du  
domaine modélisé.

Exemple : chaque compte bancaire est la  
propriété d'au moins 1 client



## Expression des cardinalités

1	$\{1\}$ ( ou 1..1)
0..1	$\{0, 1\}$
m .. n	$\{m, m+1, \dots, n-1, n\}$
*	IN (équivalente à 0..*)



Une banque peut ne gérer aucun compte?

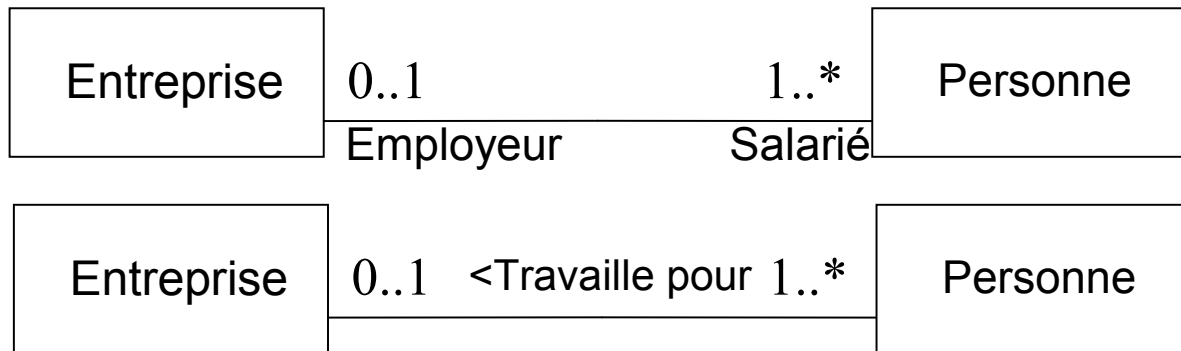
Un chargé de clientèle peut travailler dans plusieurs banques?

Un client a au moins 2 comptes?

Une banque peut n'avoir aucun chargé de clientèle?

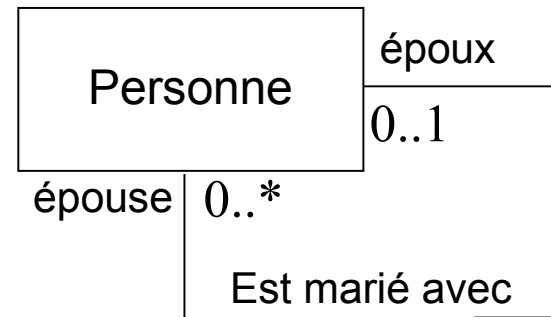
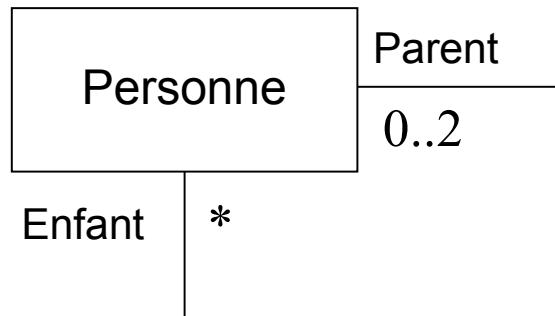
Un client ne peut pas avoir des comptes dans plusieurs agences?

## Rôles ou nommage?



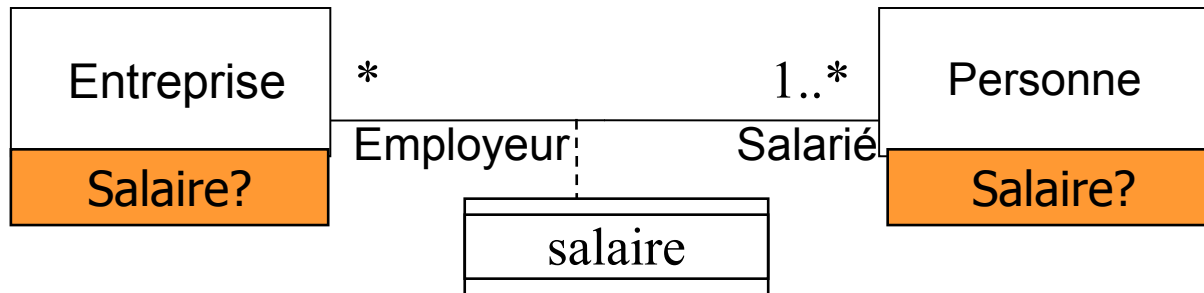
On utilisera la version apportant la plus grande clarté sémantique et lisibilité

Les rôles sont souvent indispensables  
dans le cas des associations internes

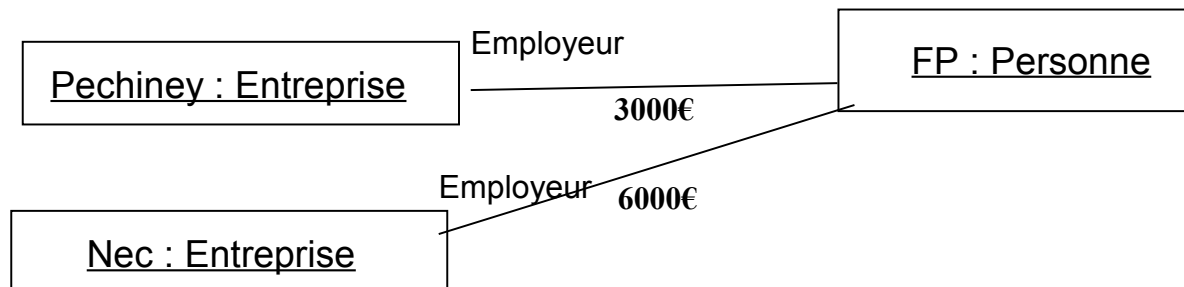


# 3. Attribut porté et classe association

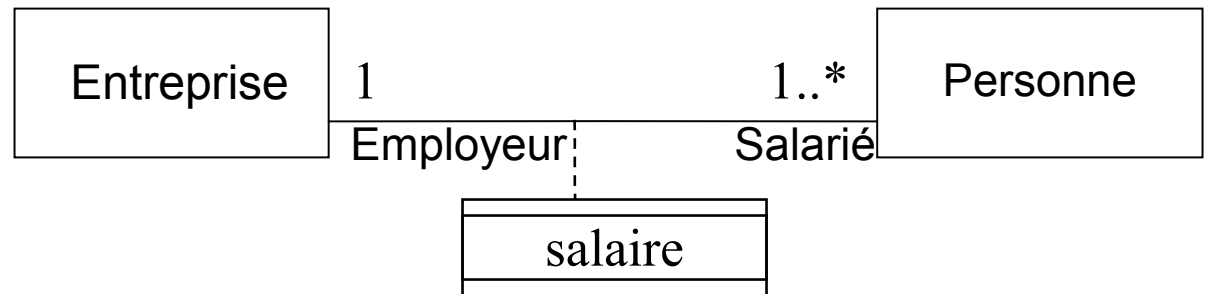
Domaine : Une personne travaille dans un nombre quelconque d'entreprises et une entreprise emploie au moins une personne. Dans chaque entreprise dans laquelle elle travaille une personne perçoit un salaire.



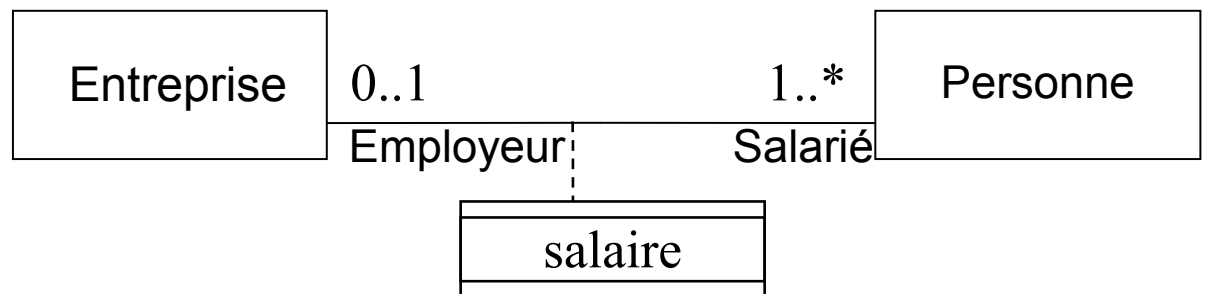
Le « salaire » n'a de sens que sur un lien car étant dépendante des deux instances liées et non d'une seule




Dans la quasi totalité des cas, la présence d'attributs portés sur une association 1-N est une erreur de modélisation (salaire à mettre côté attribut dans personne)



Une exception:  
les informations à existence conditionnée

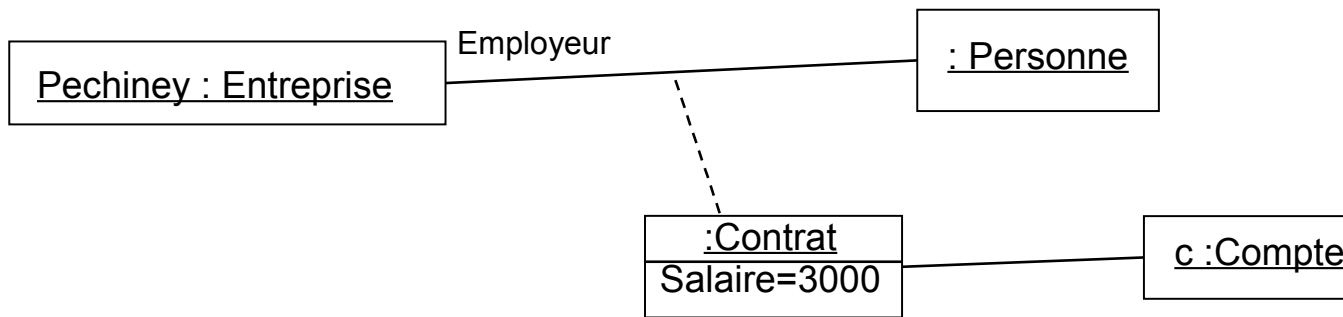
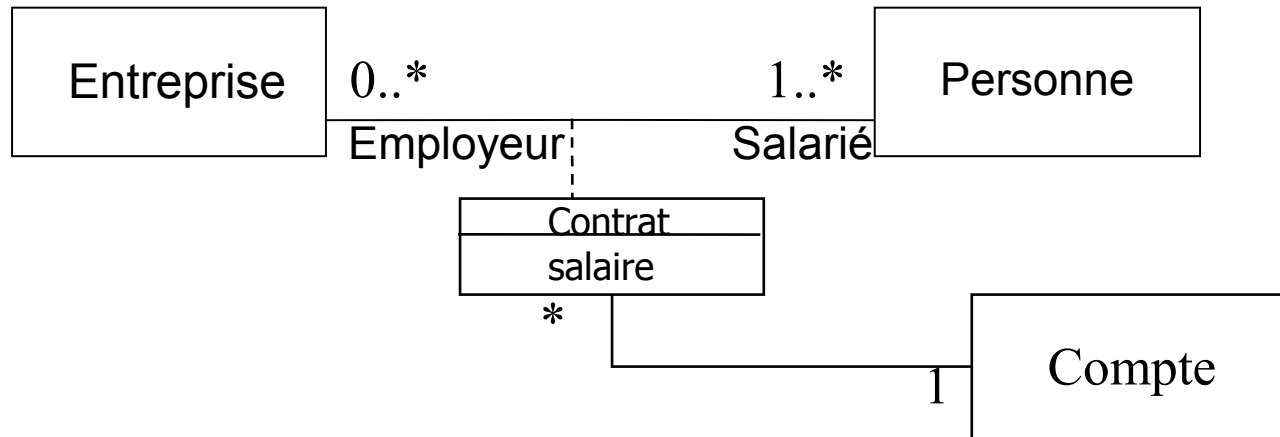






En fait la notion d'attribut porté présente dans d'autres notations n'existe pas en UML. On la simule en usant le concept bien plus puissant de *classe association*.

La classe association offre à une association le statut de classe avec tous les outils afférents : attributs et, ..., méthodes, associations, généralisations, etc.

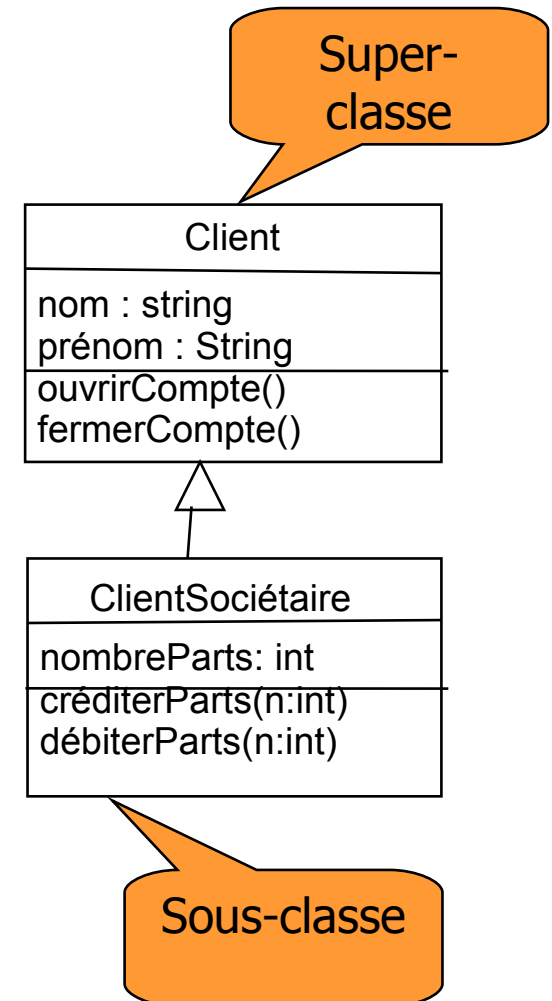


# 4. Généralisation et association

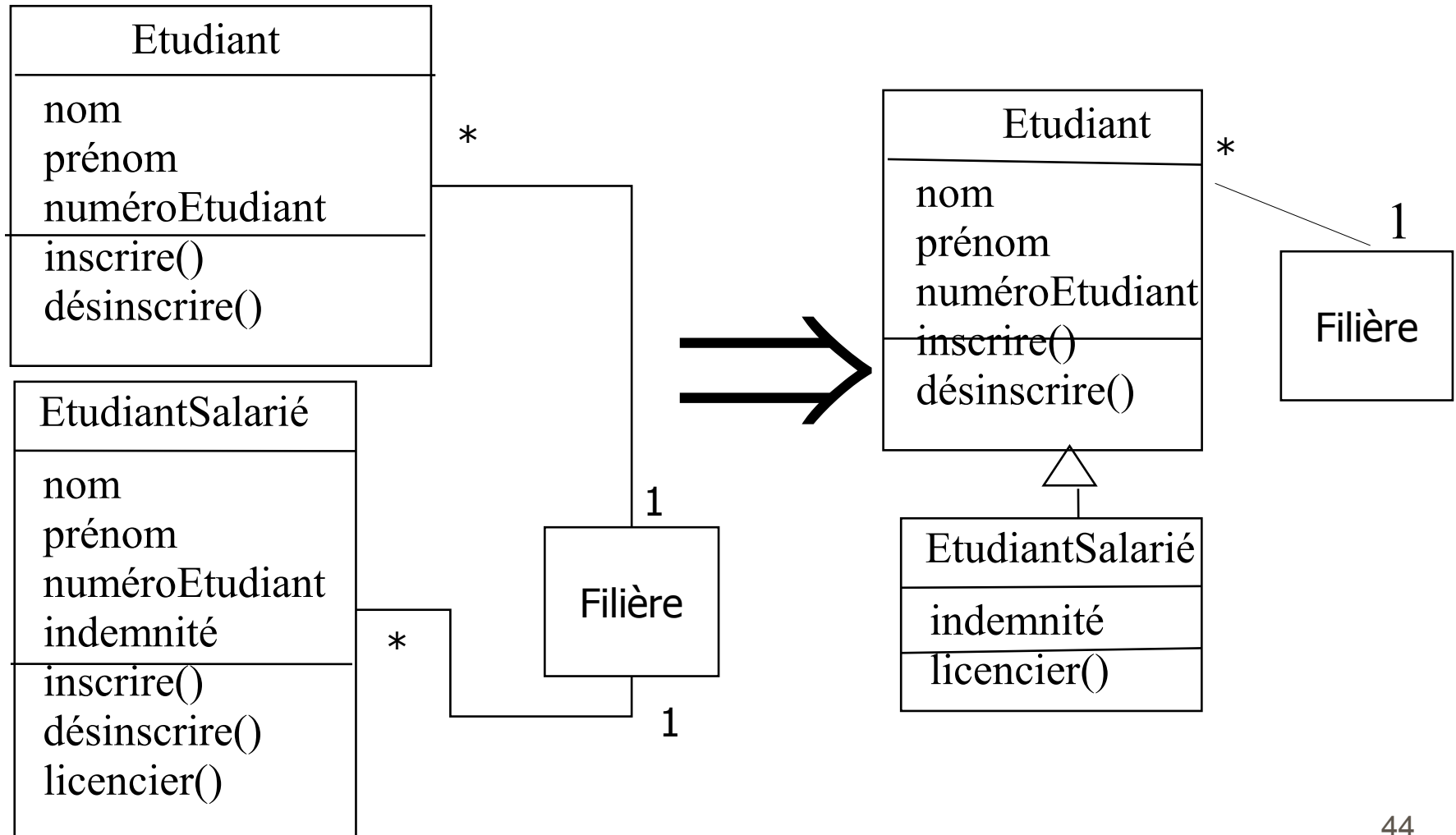
La généralisation manifeste la ressemblance de deux concepts, dont l'un peut être vu comme un cas particulier de l'autre.

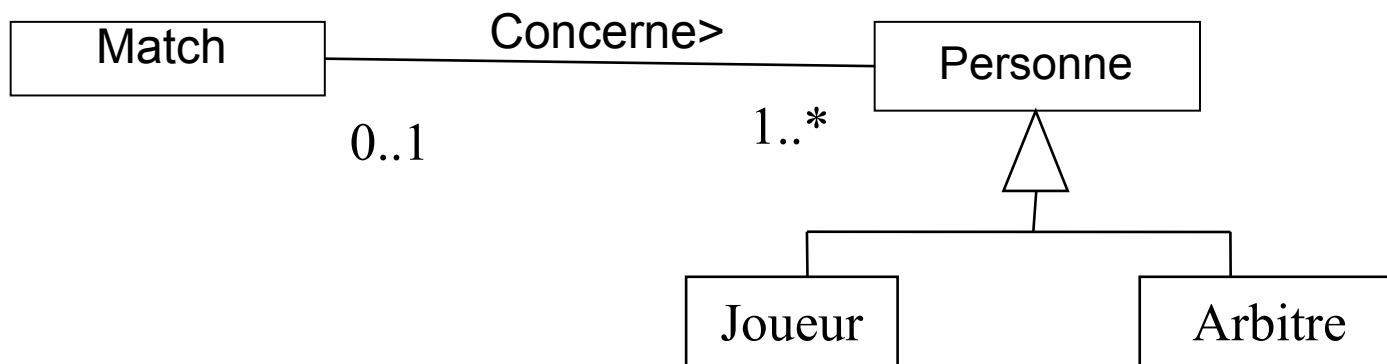
Un client sociétaire est un client qui possède des parts dans la banque.

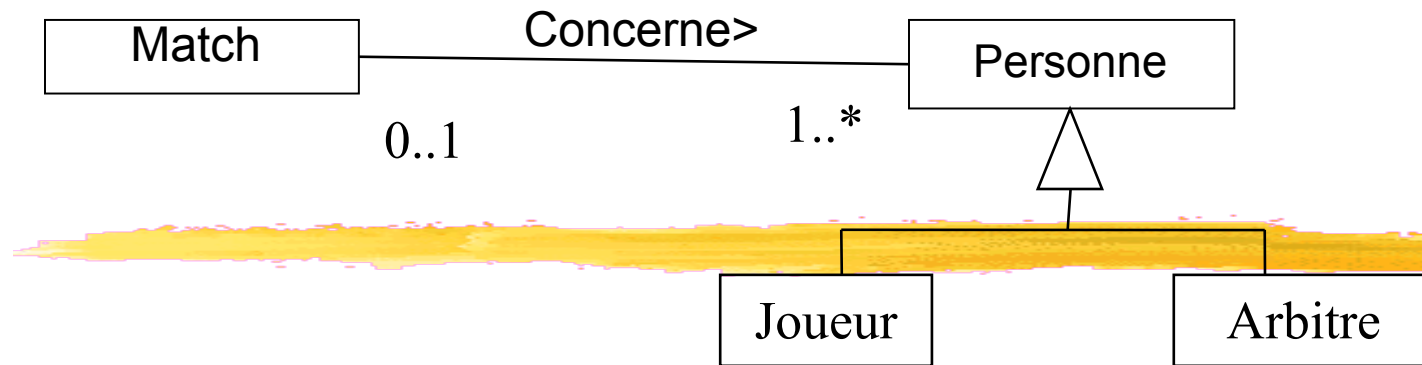
La sous-classe présente (hérite) toutes les propriétés (attributs, méthodes, associations, contraintes) de sa super-classe sans qu'il soit besoin de les rappeler.



La généralisation est en modélisation un outil de gestion de la complexité par la factorisation qu'elle rend possible



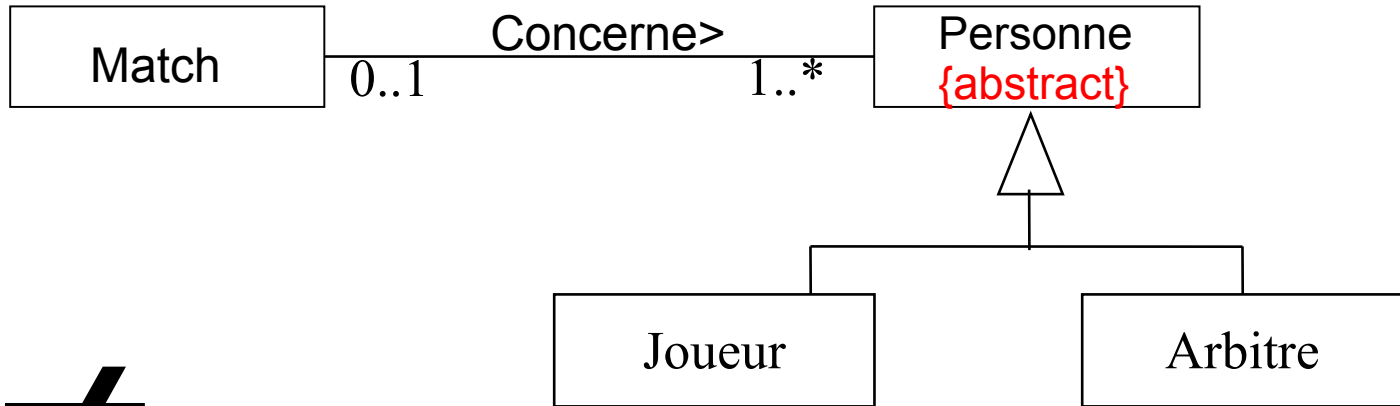




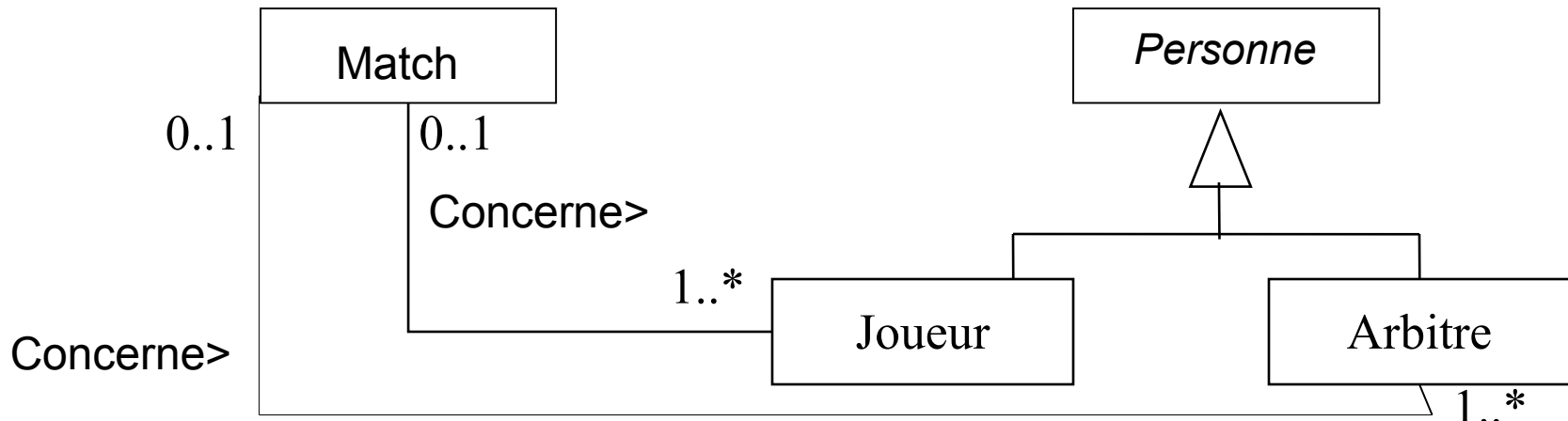
Les associations sont héritées lors d'une généralisation UML

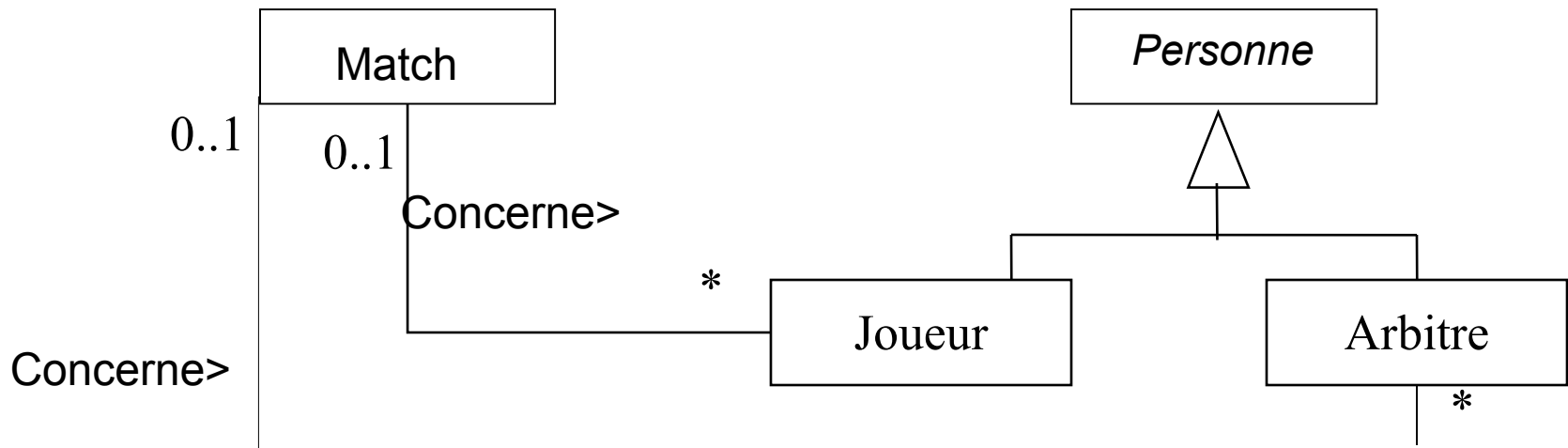
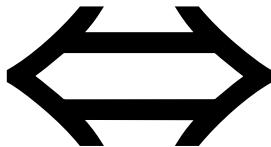
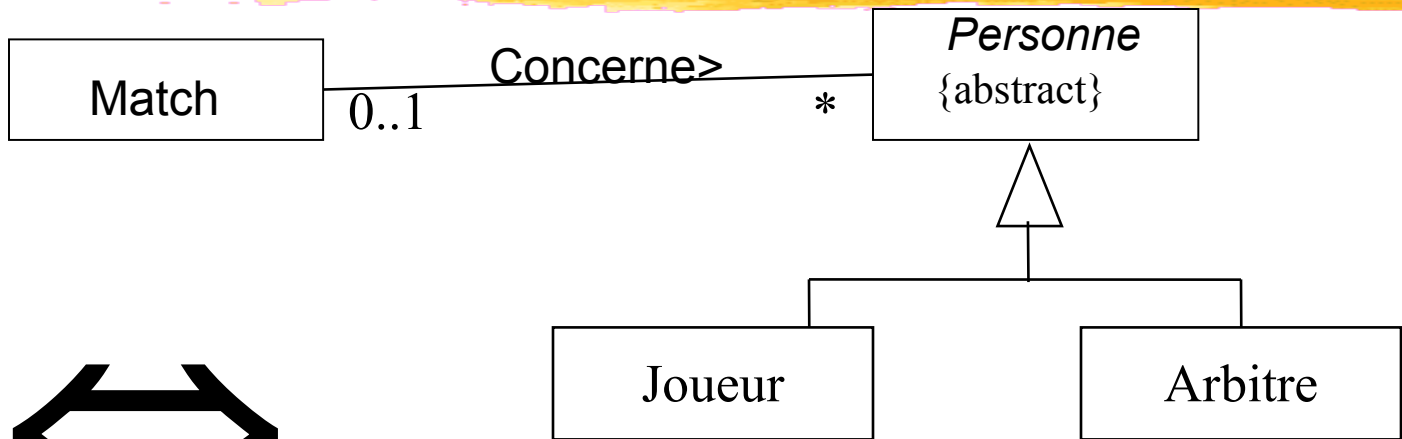
D.O. compatibles

# Classe abstraite

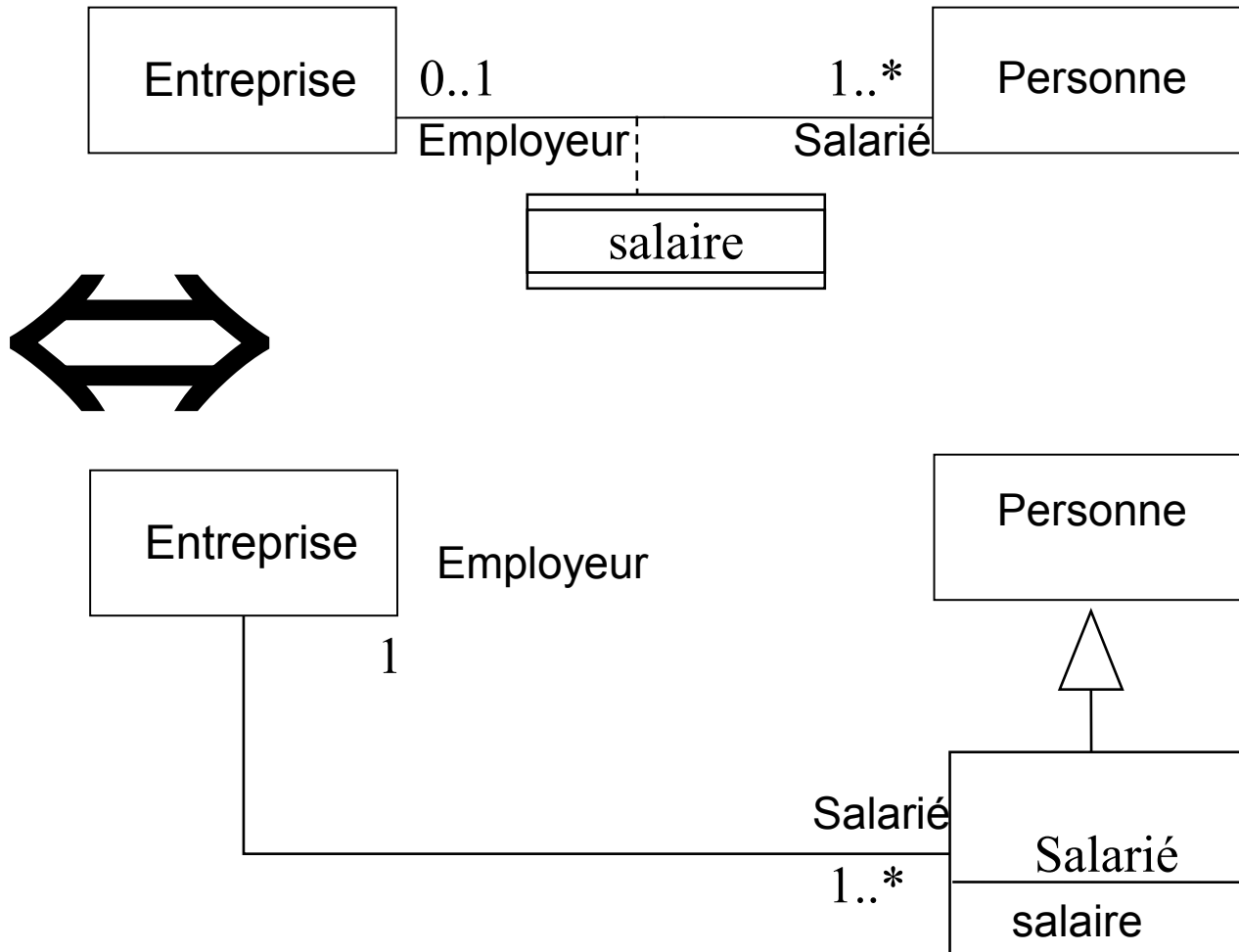


≠










# 5. Exemple de Modélisation de domaine




Avec seulement les concepts de « classe » et « d'association » on dispose déjà d'une grande puissance de modélisation

Une banque possède plusieurs agences réparties en France.  
Chaque agence a son nom, et son adresse.

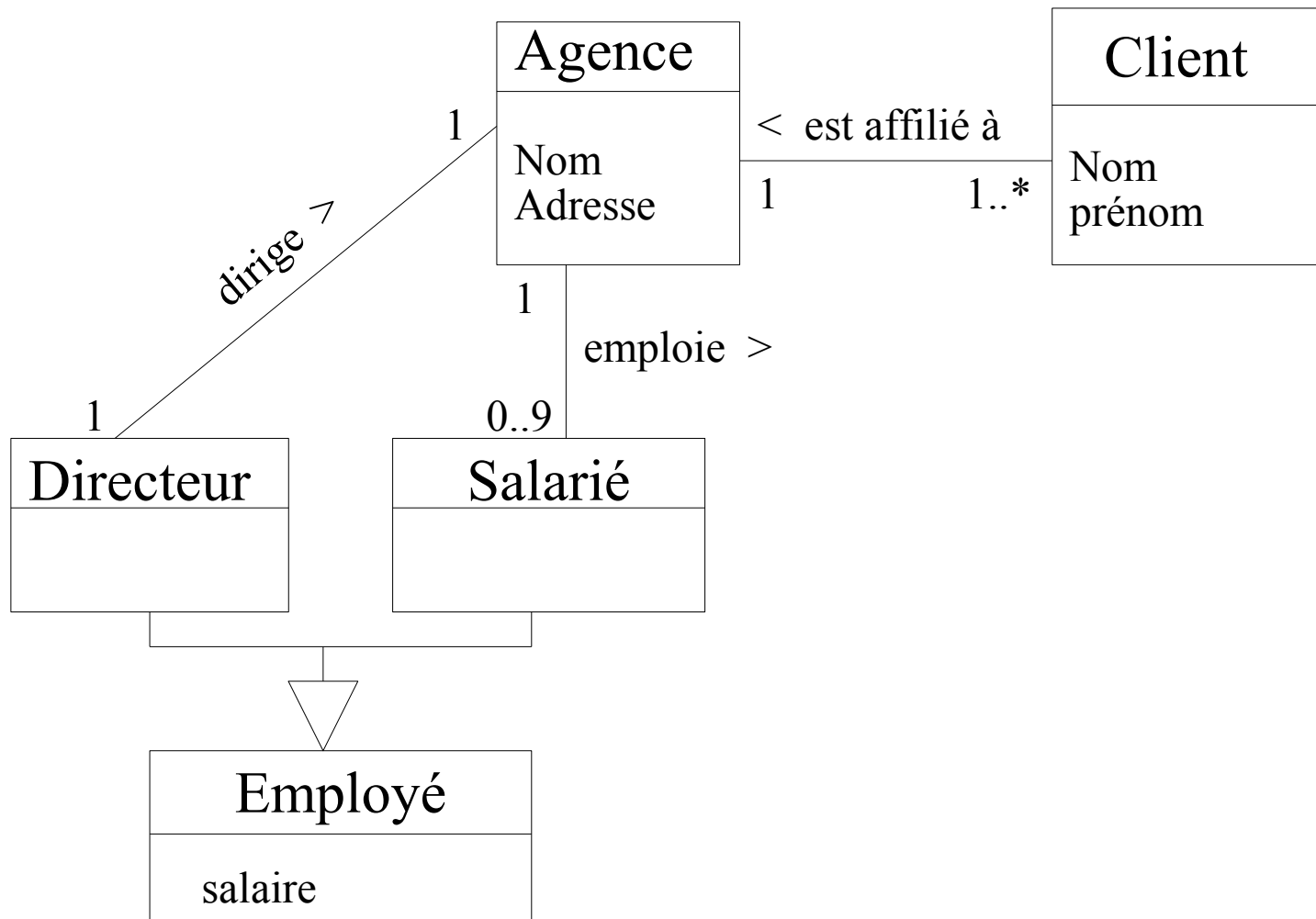


Chaque agence emploie au maximum 10 employés dont un seul est le directeur.

Chaque employé a un nom et un salaire.

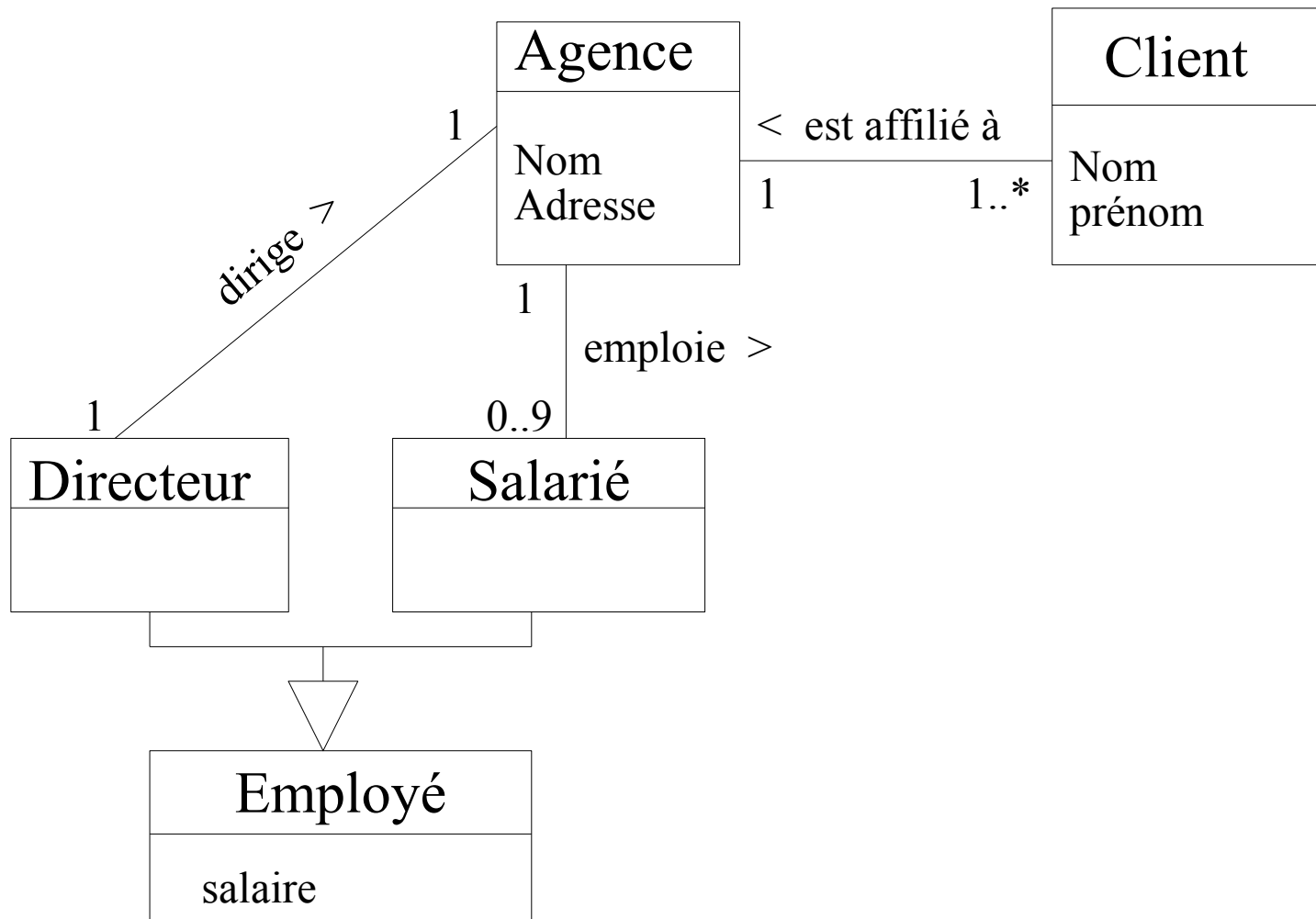


A chaque agence sont affiliés de nombreux clients ( dont on garde le nom et le prénom ).



qui peuvent chacun posséder plusieurs comptes. Il existe trois types de comptes : le compte « jeune », le compte « adulte » et le compte d'épargne. Bien sûr chaque type de compte a son propre taux d'intérêt.

Chaque compte est repéré par son numéro de compte.





## **Chapitre 2**

# **Associations Binaires**

# 1. Approche formelle

## Ensemble

collection d'entités toutes distinctes

en extension :

en compréhension :

## Ensemble des parties

Ensemble de tous les sous ensembles

$$P(A) = \{\{1\}, \{2\}, \{7\}, \{1,2\}, \{1,7\}, \{2,7\}, \{1,2,7\}, \emptyset\}$$


## Cardinal d'un ensemble

Le nombre de ses éléments

$$\text{card}(\mathcal{A}) = 3$$

$$\text{card}(\mathcal{P}(\mathcal{A})) = 2^{\text{card}(\mathcal{A})} = 2^3 = 8$$





On appelle couple de deux éléments  $x$  et  $y$  noté  $(x, y)$  l'ensemble  $\{\{x\}, \{x, y\}\}$ .

Cette définition met en valeur le caractère ordonné d'un couple :  $(a, b) \neq (b, a)$

En effet, cet ensemble vérifie la propriété d'ordre suivante :

$$(a, b) = (a', b') \Leftrightarrow a = a' \text{ et } b = b'$$



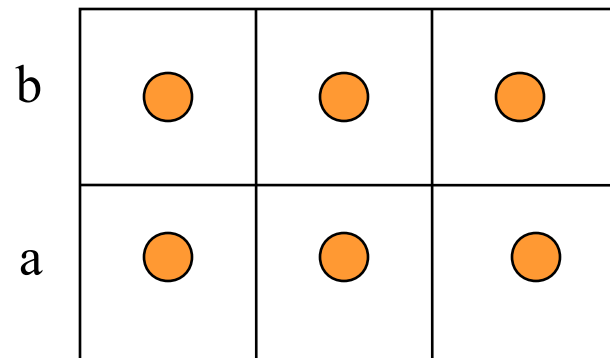
Produit cartésien de deux ensembles

On appelle produit cartésien de A et B noté  $\mathcal{A} \times \mathcal{B}$ ,  
l'**ensemble** des couples  $(x, y)$  tels que  $x \in \mathcal{A}$  et  
 $y \in \mathcal{B}$ .

Comme c 'est un ensemble il ne peut y avoir deux fois le  
même couple!

# Diagramme cartésien binaire

Seconde composante



1

2

7





Première composante

# Relation binaire

Une partie du produit cartésien de 2 ensembles

$\mathcal{R} \in \mathcal{P}(\mathcal{A} \times \mathcal{B})$  ou de façon équivalente  $\mathcal{R} \subset (\mathcal{A} \times \mathcal{B})$

$$\subset \{1,2,7\} \times \{a,b\}$$

b			
a			
	1	2	7

Personne




Voiture




1


0..2

NON

OUI

V2			
V1			
	P1	P2	P3

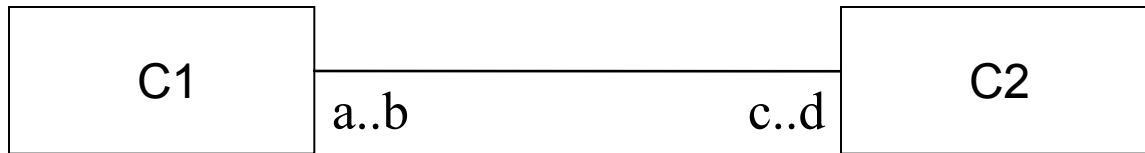
V3			
V2			
V1			
	P1	P2	P3



Une association  $\mathcal{A}$  définit donc une famille de relation  $\mathcal{R}_{\mathcal{A}}(t)$  indexée par le temps.

Toutes ces relations  $\mathcal{R}_{\mathcal{A}}(t_o)$ ,  $\mathcal{R}_{\mathcal{A}}(t_1)$ ,  $\mathcal{R}_{\mathcal{A}}(t_2)$ , *etc.* doivent vérifier les contraintes de cardinalités

Les cardinalités expriment des **invariants** temporels (une propriété qui perdure).

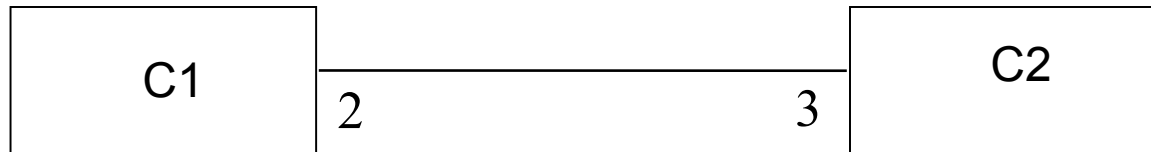


Si on dispose de N instances de C1 :  
le plus souvent toutes les valeurs de N ne  
sont pas permises. Si  $N > 0$  et  $c > 0$  :

$\text{domaine}(N) \subset \{i \in \mathbb{N} / i \geq a\}$

Les valeurs de N permises sont celles qui,  
respectueuses des cardinalités,  
conduisent à un nombre entier de C2.

## Exemple :



Il n'est pas possible d'avoir  $N=1$  instance de C1 car il faut au moins 3 instances de C2 mais alors au moins 2 instances de C1.

$N = 2$  OUI

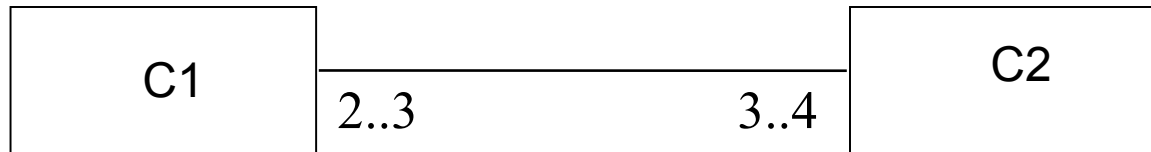
$N = 3$  NON

$N = 4$  OUI

$N = 5$  NON ...

$Dom(\mathcal{N}) = \{2n \mid n \geq 1\}$





Il n'est pas possible d'avoir  $N=1$  instance de C1 car il faut au moins 3 instances de C2 mais alors au moins 2 instances de C1. Donc  $N=1$  n'est pas une valeur permise!

Il est possible d'avoir  $N=2$  instances de C1 et il faut au minimum  $\text{Max}((2*3)/3, 3)=3$  instances de C2.

Il est possible d'avoir  $N=3$  instances de C1, il faut alors au minimum  $\text{Max}((3*3)/3, 3)=3$  instances de C2

Il est possible d'avoir  $N=4$  instances de C1, il faut alors au minimum 4 instances de C2.

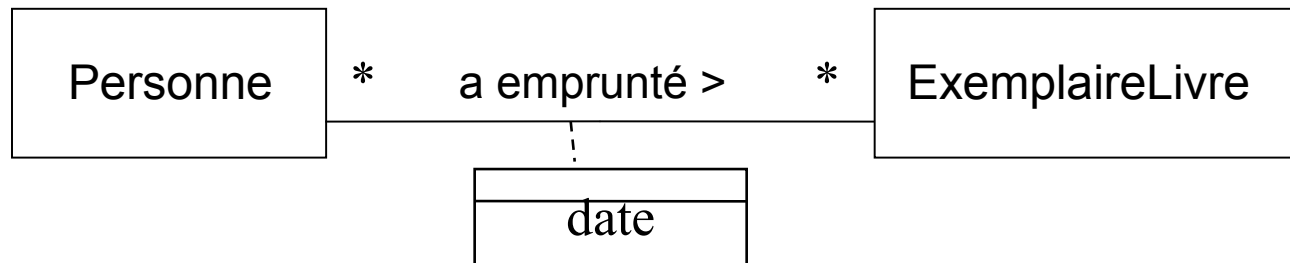
$\text{Dom}(N)=\{n > 1\}$

## **2. Intérêt de cette formalisation**

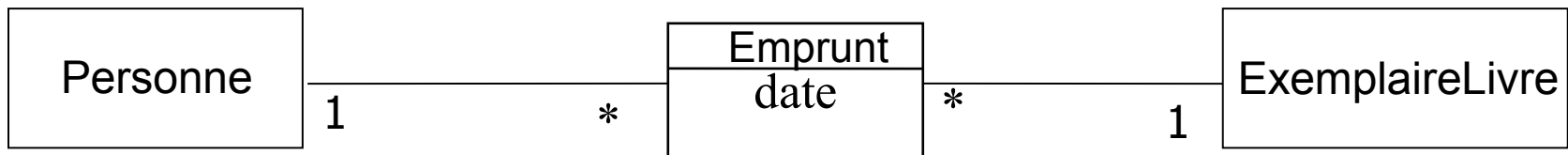


Vous devez toujours prendre garde à ne pas confondre les sémantiques que vous affichez au travers des nommages et rôles et les contraintes effectivement (mathématiquement) assurées par la notation UML

Exemple : le modèle d'une bibliothèque dans laquelle, on veut conserver les emprunts des ouvrages

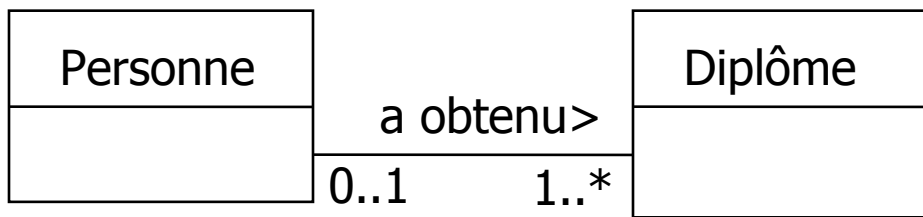


Une personne ne peut avoir emprunté qu'une seule fois un ouvrage car un couple  $(a, b)$  ne peut apparaître qu'une seule fois dans  $\mathcal{A} \times \mathcal{B}$ .  
Le modèle proposé est donc faux.



Le modèle est presque correct!

Mais il n'empêche pas qu'un même exemplaire puisse être emprunté à la même date :  
par deux personnes différentes (ubiquité!)  
ou par la même personne (absurdité!)



## Attention à ce que dit :

A chaque instant une personne a au moins un diplôme

A chaque instant un diplôme n'est possédé que par au plus une personne

## et ne dit pas un diagramme de classes :

Une personne conserve (à vie) les diplômes obtenus (pas de suppression)

Un diplôme est nominatif (pas de réaffectation)

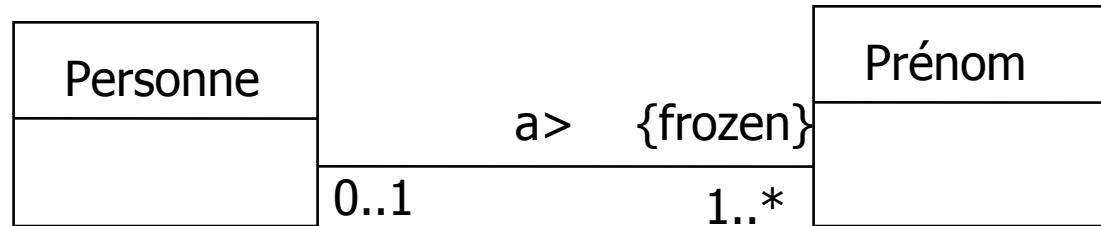
Le nombre des diplômes est figé (pas d'ajout)

frozen : on ne peut ni supprimer ni rajouter

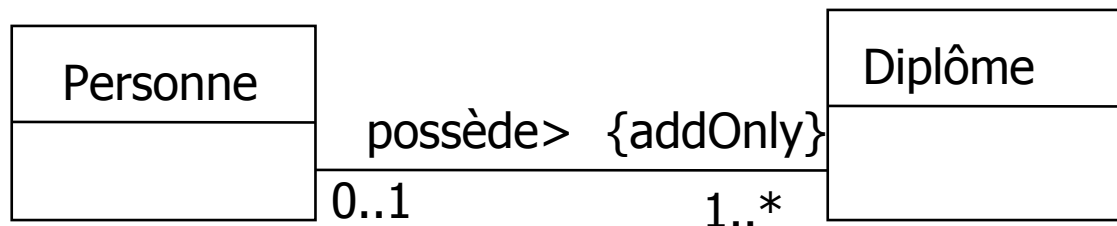
addOnly : on ne peut pas supprimer

## Ainsi selon le domaine :

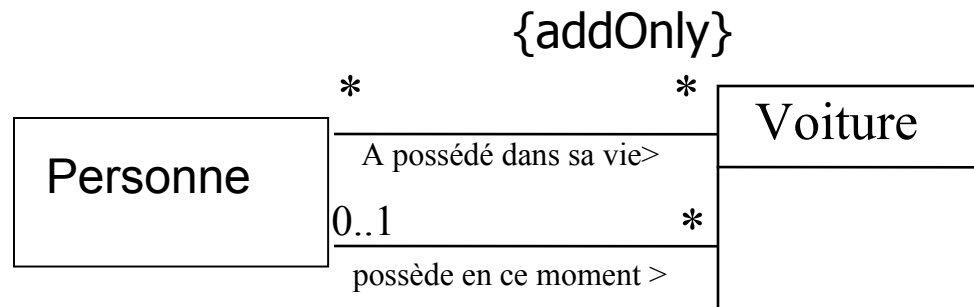
Une personne à des prénoms définis une fois pour toute à sa naissance



Un diplôme un fois acquis, l'est pour la vie OUF!




### 3. Le problème de l'archivage



Ce modèle empêche qu'une voiture qui a été possédée à  $t_n$  par une personne, ne le soit plus à  $t_m \neq t_n$ .

Cependant, il n'impose pas qu'une voiture possédée puis cédée soit archivée!

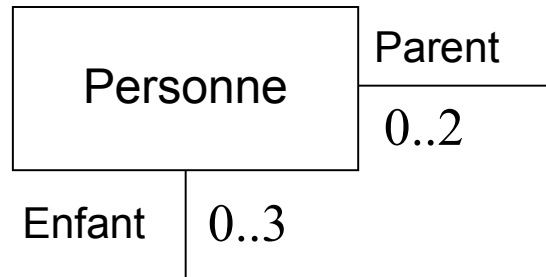
En fait ici, on suppose que le modèle objet à l'instant  $t$  est la vérité ultime et qu'il n'est pas modifié dans les traitements dans un sens contraire à la sémantique qu'il cherche à exprimer (ici en n'archivant pas une possession qui se termine).



Dans les faits, on ne marque pas ces contraintes implicites à la sémantique du nommage pour ne pas alourdir les diagrammes. Il faudra cependant veiller au respect de ces contraintes lorsque l'on écrira la partie traitement

Ne décorez donc pas les associations de contraintes *addOnly* ou *frozen*

# Le cas des associations internes



Enfant

P3				
P2				
P1	●			
	P1	P2	P3	Parent

Enfant

P3				
P2	●			
P1		●		
	P1	P2	P3	Parent

Enfant

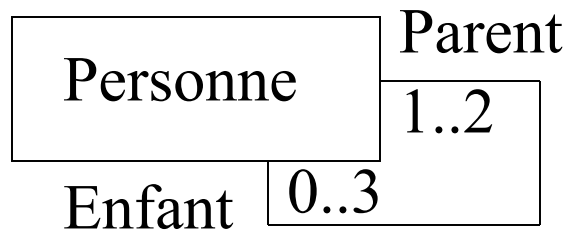
P3				
P2	●	●		
P1	●	●		
	P1	P2	P3	Parent

Une personne peut être son propre enfant et parent. Mais une seule fois!  
 $(a,a) \in \mathcal{A} \times \mathcal{A}$

Une personne peut avoir un enfant qui est son parent  
 $(a,b) \neq (b,a) \in \mathcal{A} \times \mathcal{A}$

Avec deux Personnes on peut avoir 4 liens!  
 $(a,b) \neq (b,a) \neq (a,a) \neq (b,b)$   
 $\in \mathcal{A} \times \mathcal{A}$





Une personne a au moins un Parent, qui a à son tour au moins un parent etc...

Les Cardinalités doivent toujours contenir le 0.



# **Chapitre 3**

## **Les Contraintes**

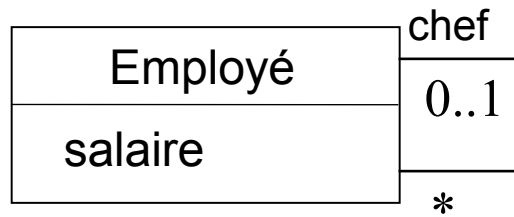
# 1. Les Contraintes



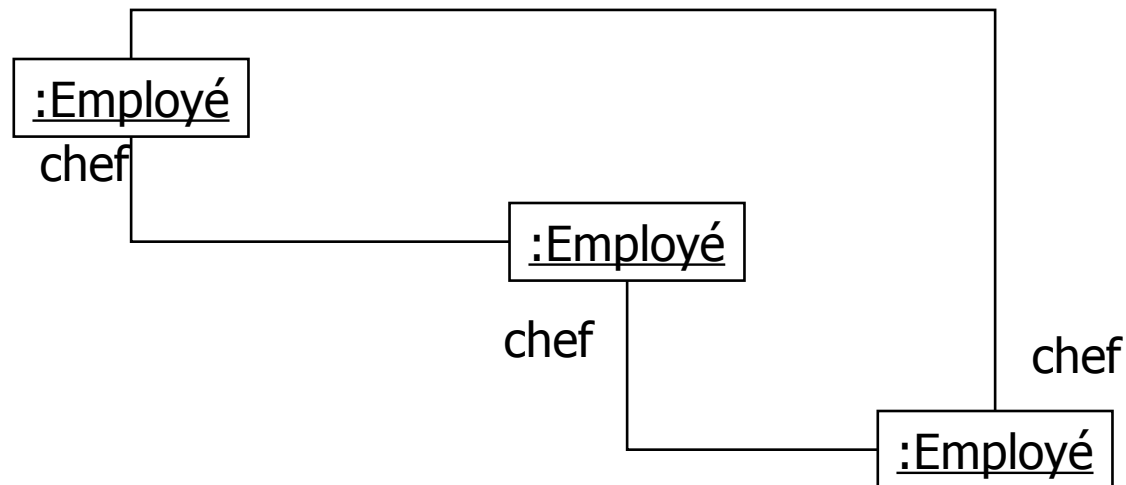
Il est fréquent de ne pouvoir exprimer toutes les informations d'un domaine à l'aide des seuls outils graphiques UML.


Soit car le modèle UML est trop permissif

Soit car le modèle UML n'est pas assez expressif



Un employé peut avoir comme supérieur l'un de ses subalternes!

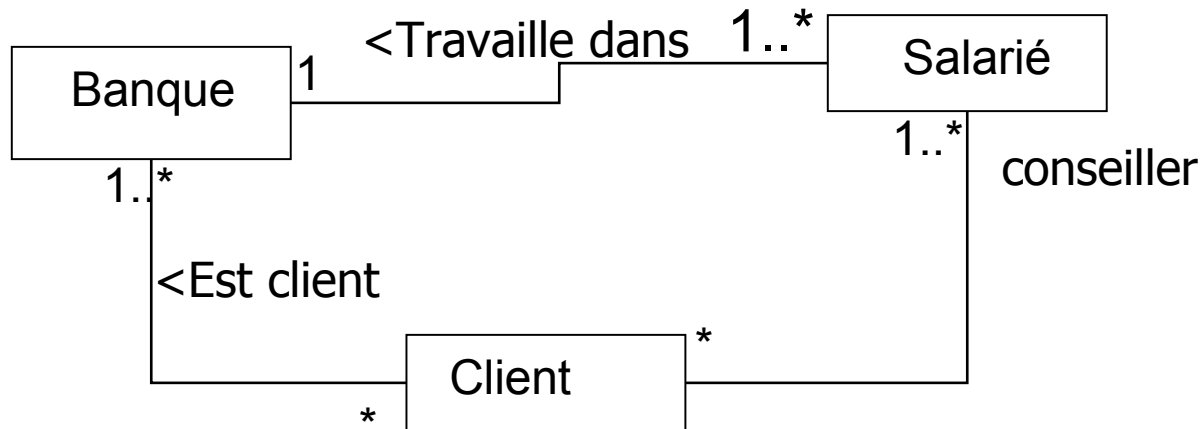




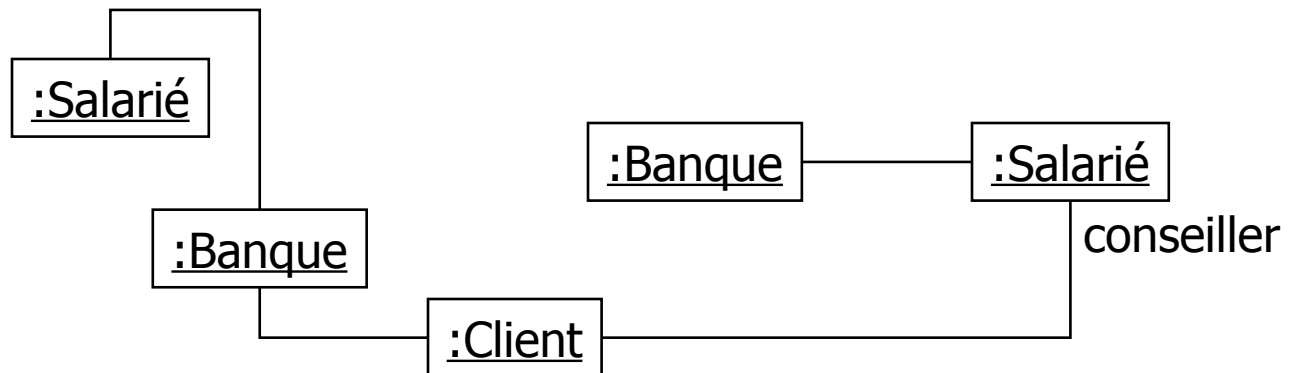
Personne
dateNaissance dateMariage


Une personne peut s'être mariée avant d'être née!

<u>:Personne</u>
dateNaissance=02/11/68 dateMariage= 01/10/67



Un client peut être conseillé par un salarié d'une banque dans laquelle il n'est pas client!



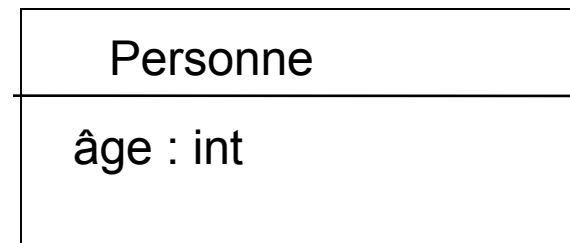
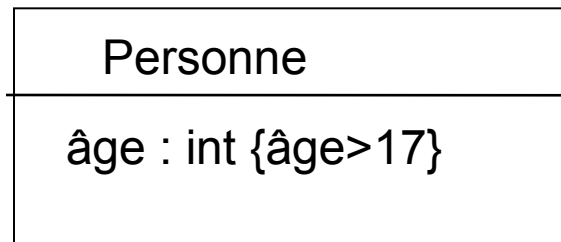


Il est donc souvent nécessaire pour modéliser fidèlement un domaine d'ajouter aux diagrammes de classes des **contraintes** qui vont restreindre les formes instanciables. Les contraintes comptent autant en partie que le diagramme de classes.

# Les contraintes peuvent se représenter :

près des éléments concernés sur le diagramme  
entre {...}

en annexe du diagramme en précisant leur contexte  
(préférable : c'est ce que l'on fera)



Personne  
âge>17





Il y a deux grands types de contraintes :

Les contraintes statiques

Les contraintes dynamiques

Ces contraintes peuvent être :

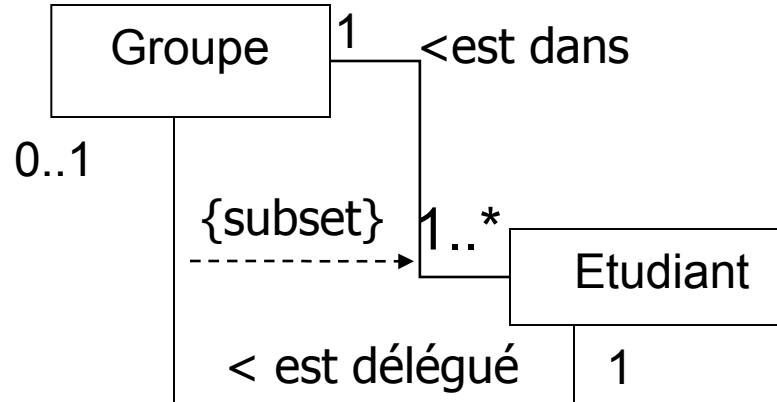
intra-classe

inter-classes

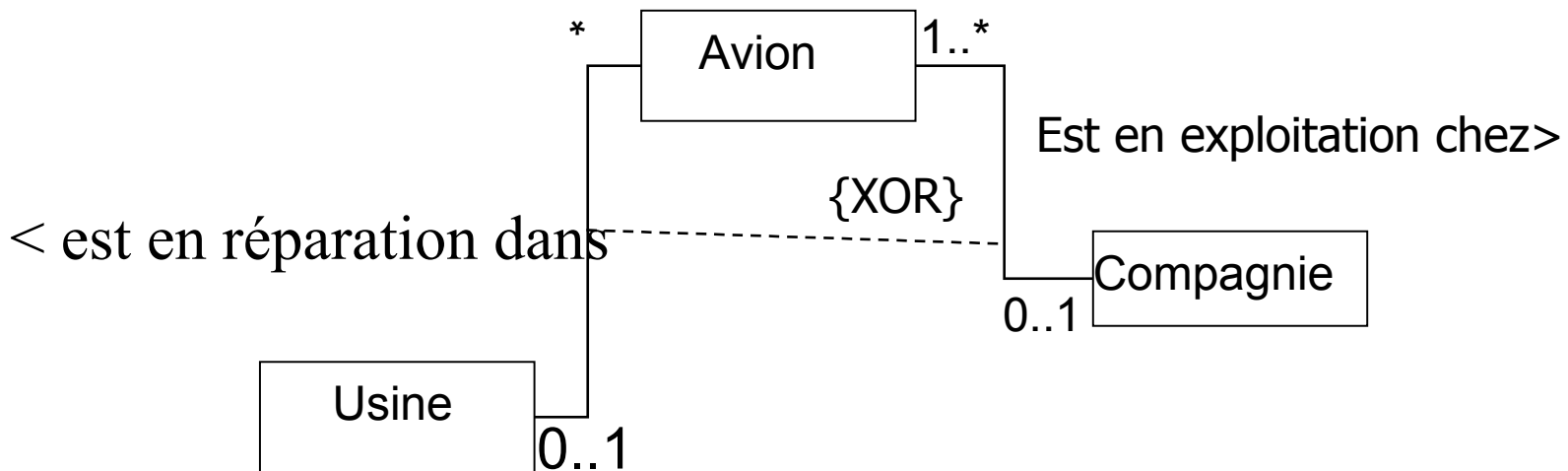


En UML certaines contraintes statiques  
très fréquentes sont prédéfinies :

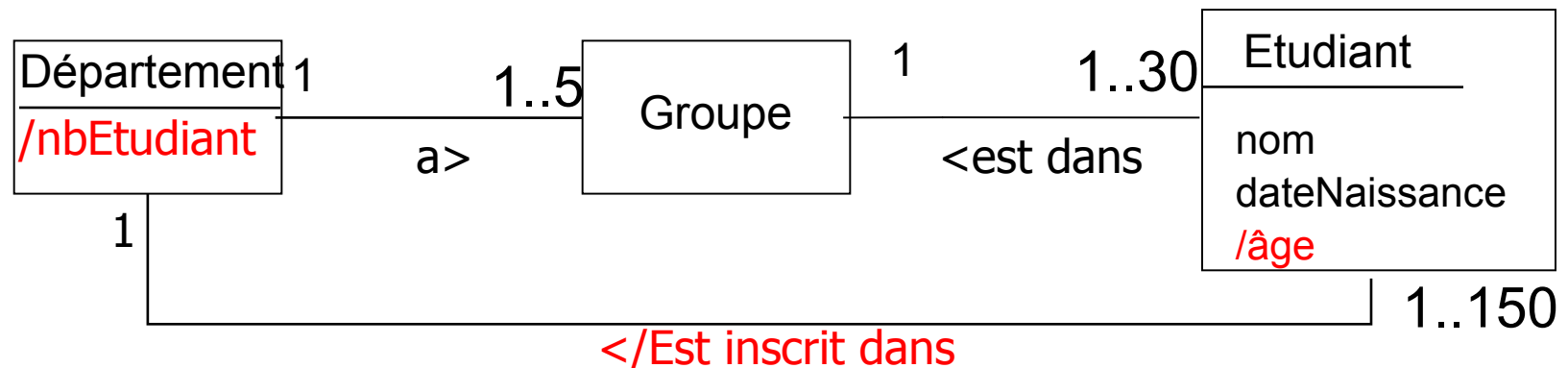
XOR, SUBSET, COMPLETE, DISJOINT,  
Dérivabilité, etc.




Délégué  $\subseteq$  Est\_dans



Un attribut ou une association est dit dérivable s'il peut s'obtenir (être calculé) à l'aide d'autres informations présentes dans le diagramme. Il se note précédé d'un « / »

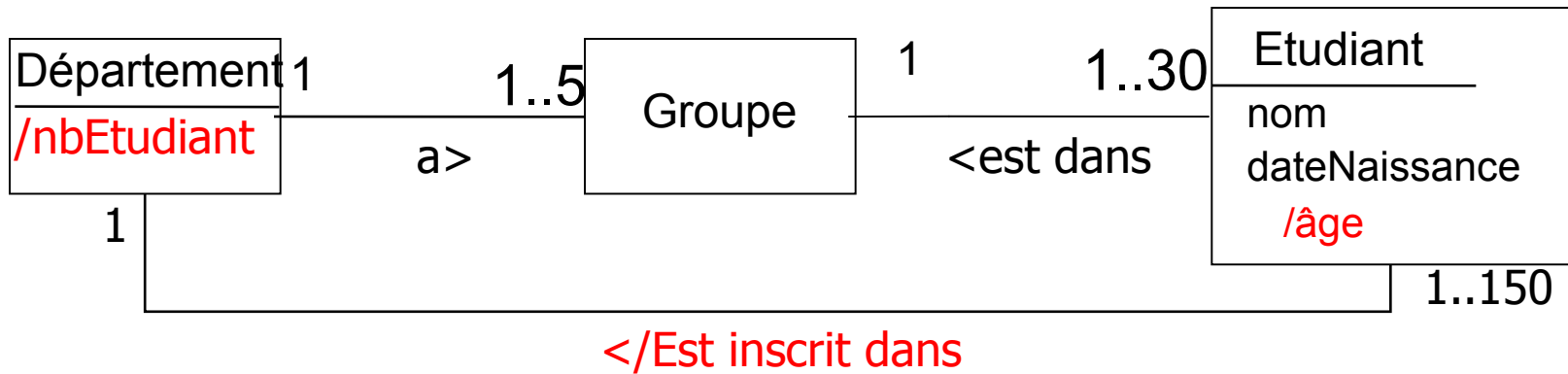




Les attributs et associations dérivés doivent être laissés sur le diagramme car ils manifestent une information que l'on souhaite gérer.

Par contre, il convient :

- de décorer du « / » les attributs et associations concernés
- d'indiquer précisément de quelle façon sont calculés ces éléments



Département  
- nbEtudiant =

Etudiant  
- âge =

/Est inscrit dans




Pour les autres contraintes statiques UML propose le langage textuel OCL.

Ce langage n'est pas fait pour la description de contraintes dynamiques. L'écriture de certaines d'entre-elles est cependant possible dans le cadre de pré/post-condition :

```
setSalaire(sal); {pré : sal > salaire}
```





A l' IUT on utilisera de contraintes écrites  
textuellement en (bon) français, placées  
en annexe du diagramme

La syntaxe d'une contrainte :

Nom de la classe concernée

Contrainte A  
Contrainte B  
etc...



Pour le domaine fini (convention IUT)

$\text{dom}(\text{couleur}) = \{R, V, B\}$

Pour le caractère identifiant :

attribut (1) ou (x)

le x pour numéroté les identifiants potentiels multiples

Etudiant
Nom (1) Prénom (1) âge

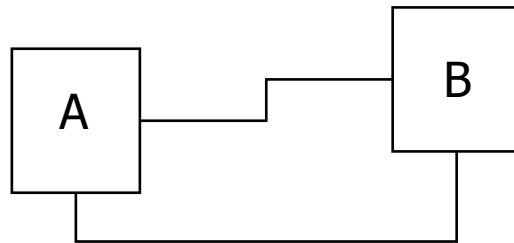
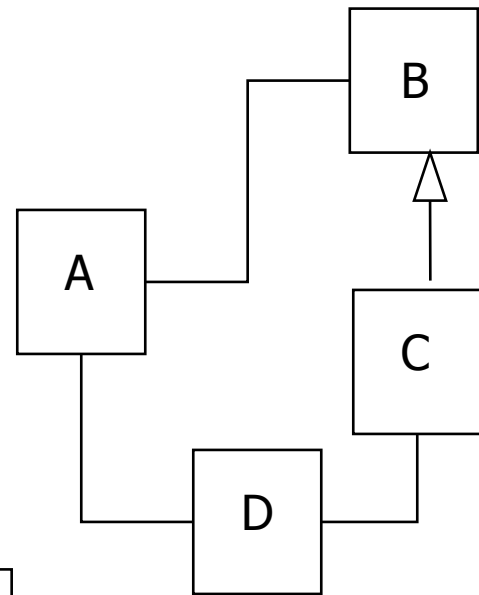
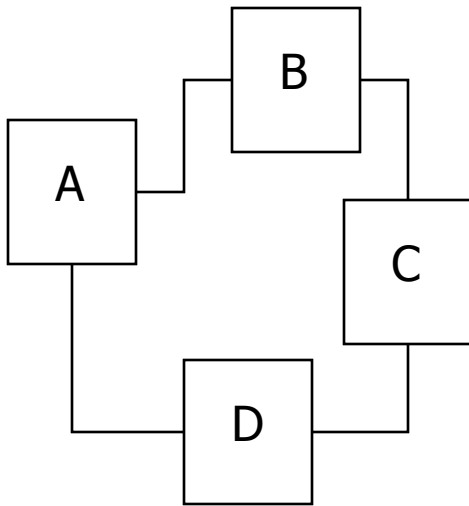
Etudiant
Nom (1) Prénom (1) numEtud (2)

## 2. Les contraintes de cycle



Des contraintes très fréquentes en modélisation de domaine sont celles des **vrais cycles non simplifiables**.

Un cycle correspond à la présence, dans le graphe des classes, d'un parcours traversant des associations et des généralisations menant d'une classe vers elle même





Il y a trois sortes de cycles :

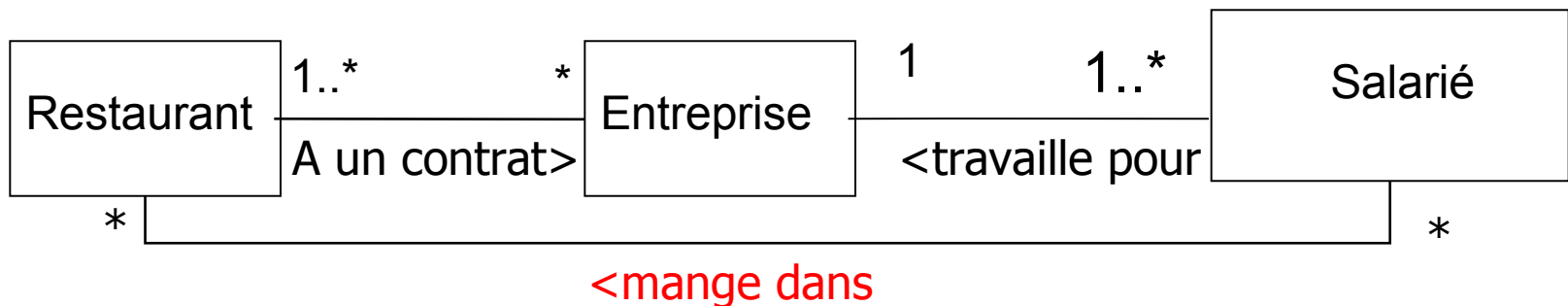
VRAI CYCLE

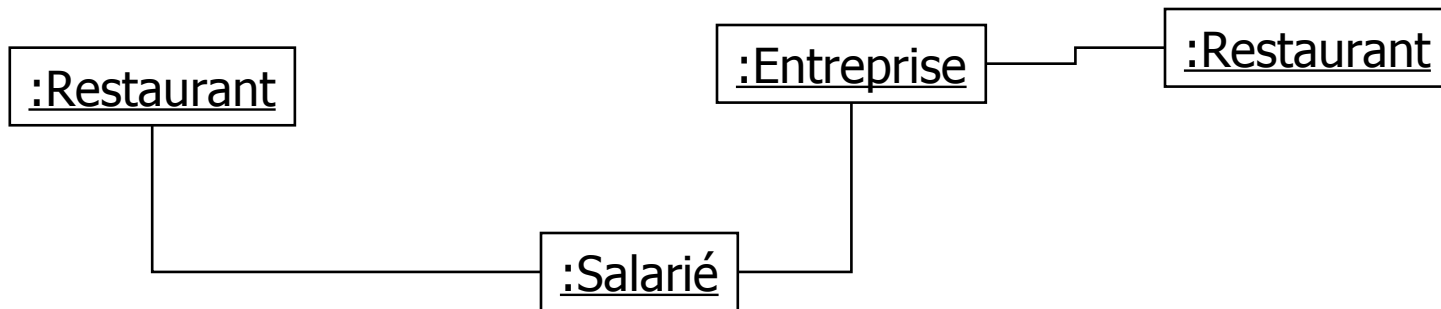
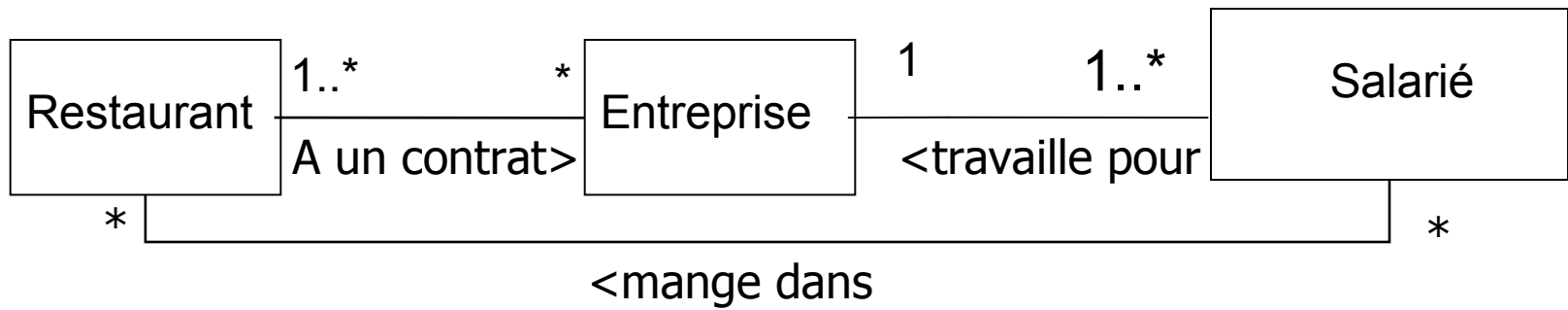
Simplifiable

Non Simplifiable

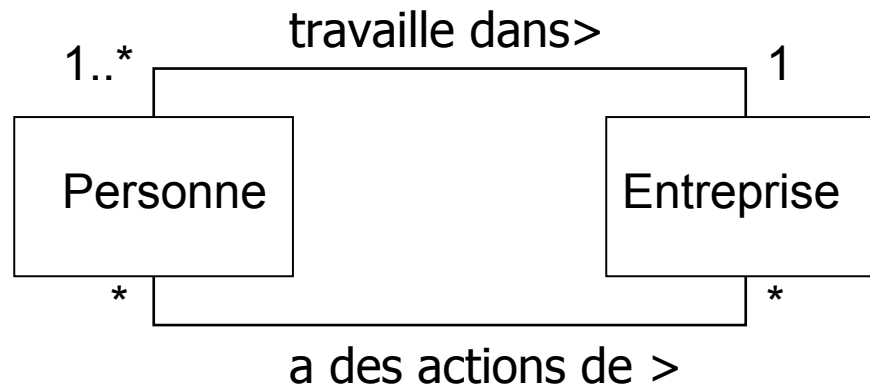
FAUX CYCLE

Un **vrai cycle** est un cycle dans lequel toutes les associations sont sémantiquement liées dans le domaine.



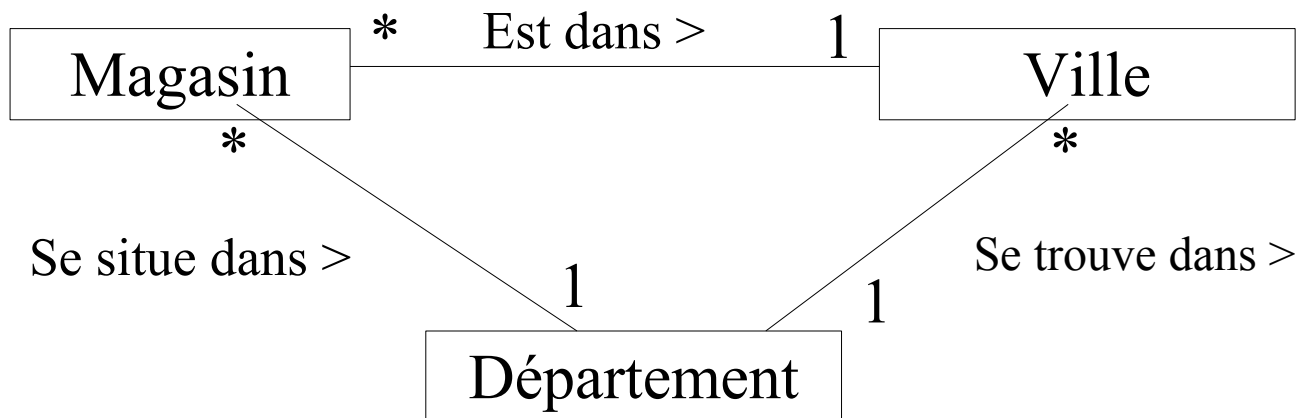


FAUX cycle : les associations n'ont aucun rapport entre elles





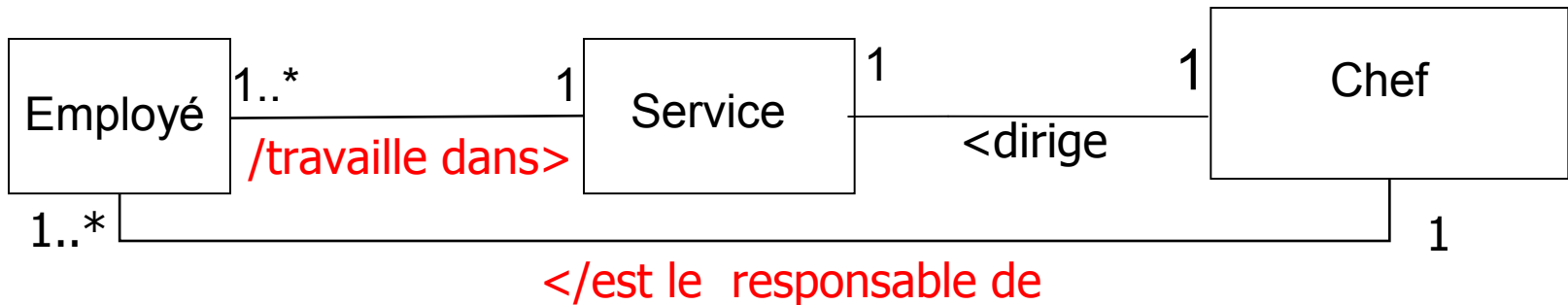
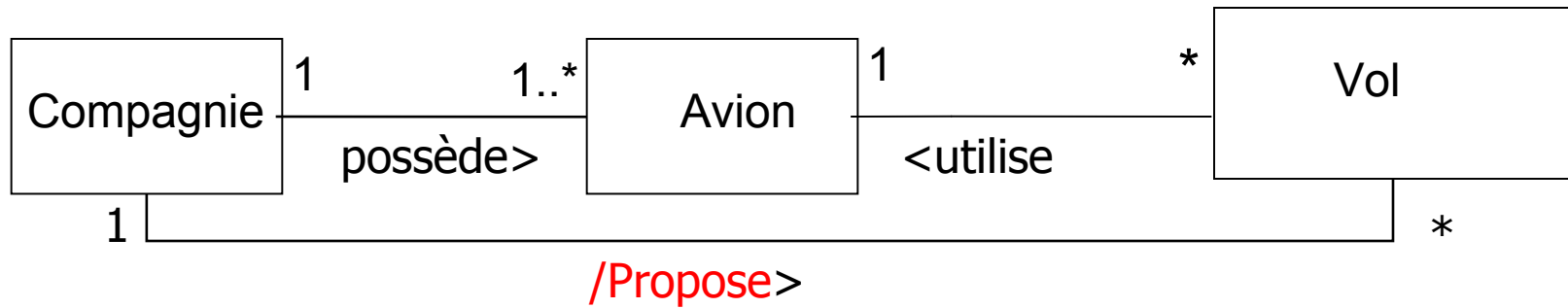
Un vrai cycle permet souvent la génération d'instances non conformes au domaine





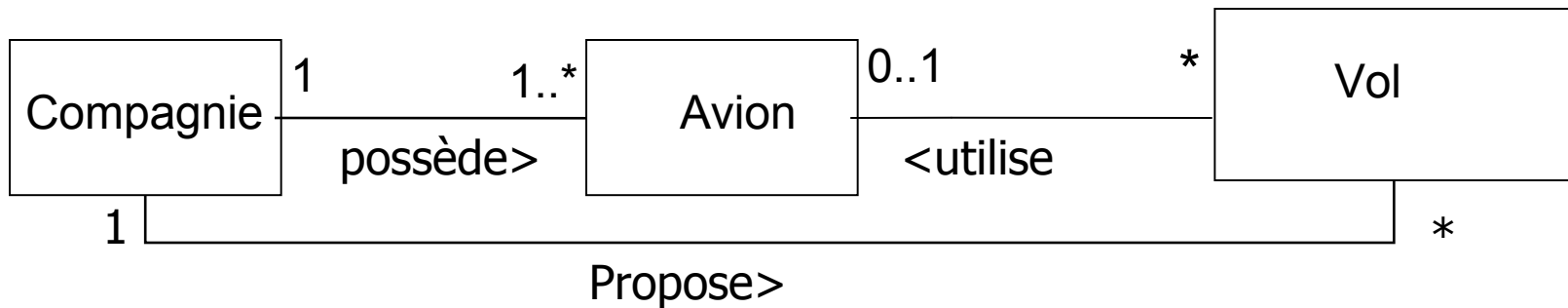
Un **vrai cycle** est simplifiable si l'une de des associations participantes est dérivable.

Note : il se peut même que plusieurs d'entre elles le soient.



2 Vrais cycle simplifiables

## Vrai cycle non simplifiable




Contrainte :

Compagnie, Avion, Vol

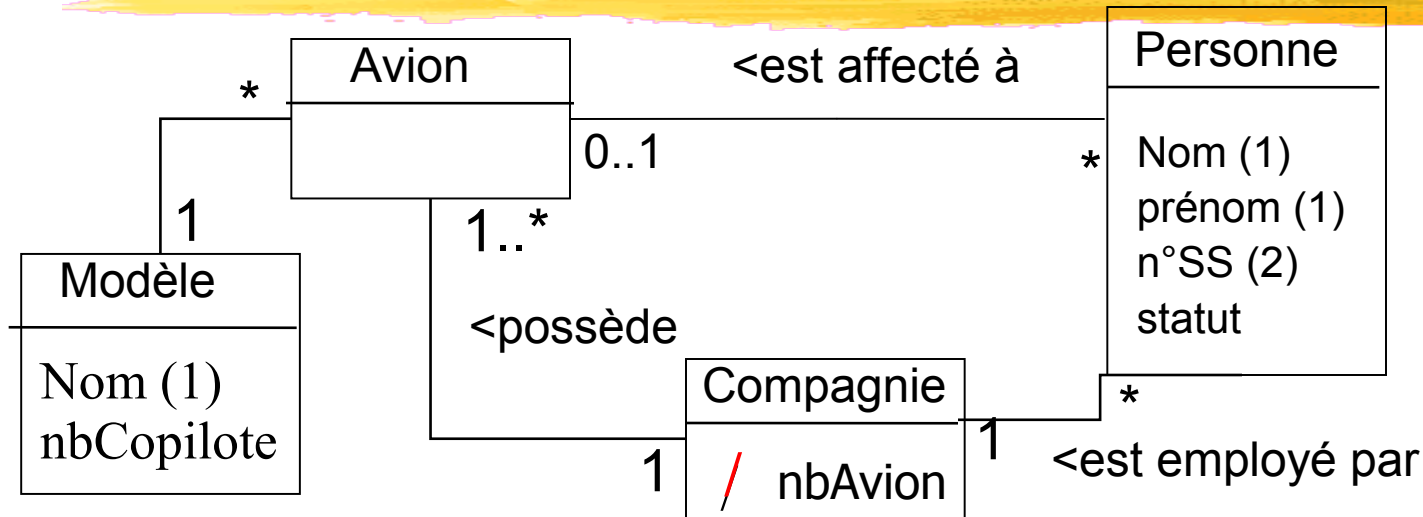
Une compagnie ne peut proposer un vol que si elle possède un avion utilisé par ce vol.

*ici la cardinalité 0 peut faire en sorte qu'un vol n'utilise pas d'avion : le lien Propose n'est pas calculable*



On résoud le problème des vrais cycles en :  
notant dérivable l'une des associations dérivables  
d'un cycle simplifiable  
rajoutant une contrainte à un vrai cycle non  
simplifiable

# 3. Exemple



## Personne

- dom(statut)={pilote, copilote, Stewart, hôtesse}

## Avion

- un avion a 1 pilote et autant de copilote que le nécessite son modèle

## Compagnie

- nbAvion =

## Avion-Personne-Compagnie

-



# **Chapitre 4**

## **Les Associations n-aires**

### **La qualification**

# 1. Approche formelle

## N-uplet

On appelle n-uplet de n éléments  $(x_i)_{i=1..n}$

noté  $(x_1, x_2, \dots, x_n)$  l'ensemble  $((\dots((x_1, x_2), x_3), \dots), x_n)$ .

Il vérifie :

$$(x_1, x_2, \dots, x_n) = (x'_1, x'_2, \dots, x'_n) \Leftrightarrow x_1 = x'_1 \dots \text{et } x_n = x'_n$$





## Produit cartésien généralisé

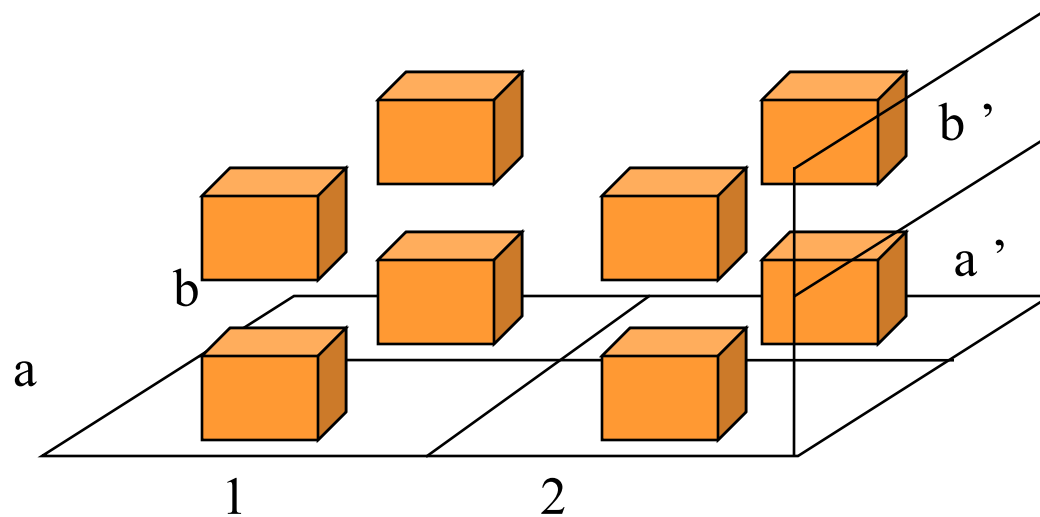
On appelle produit cartésien des  $(\mathcal{A}_i)_{i=1..n}$   
noté  $\Pi \mathcal{A}_i$ , l'ensemble des n-uplets  $(x_1, x_2, \dots, x_n)$

tels que

$$x_1 \in \mathcal{A}_1, \dots, x_n \in \mathcal{A}_n.$$

**Remarque :** *comme c'est un ensemble il ne peut y avoir deux fois le même n-uplet!*

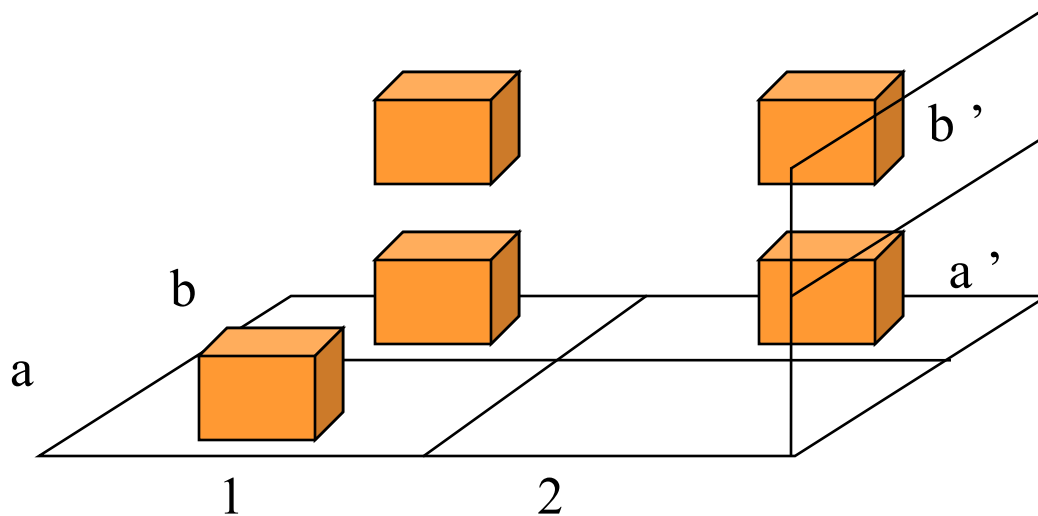
# Diagramme cartésien ternaire



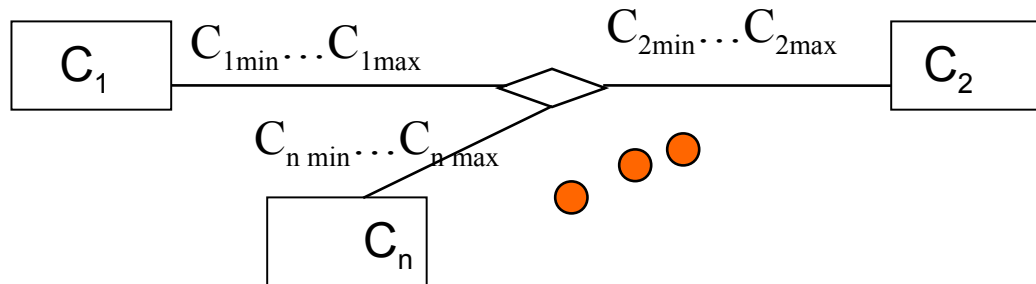
# Relation n-aire

Une partie du produit cartésien de  $n$  ensembles

$\mathcal{R} \in \mathcal{P}(\mathcal{A} \times \mathcal{B} \times \mathcal{C} \dots)$  ou de façon équivalente  $\mathcal{R} \subset (\mathcal{A} \times \mathcal{B} \times \mathcal{C} \dots)$



# Définition formelle



Soit  $I(C_1, t)$ ,  $I(C_2, t)$ , ...,  $I(C_n, t)$  respectivement l'ensemble

des instances des classes  $C_1, C_2, \dots, C_n$  à l'instant  $t$ .

Soit  $R(t)$  la relation n-aire issue des liens

instances de l'association entre  $C_1, C_2, \dots, C_n$  à l'instant  $t$



$\forall t,$

$\forall i \in [1, n],$

$\forall$  le  $n-1$  uplet sans  $i$ -ème composante

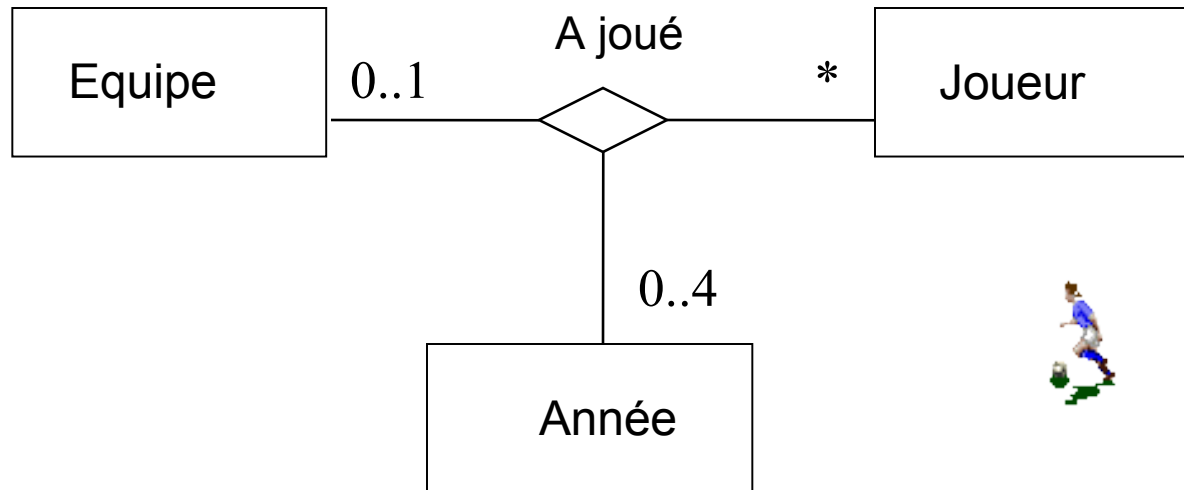
$(c_1, c_2, c_3, \dots, c_{i-1}, c_{i+1}, \dots, c_n) \in I(C_1, t) \times \dots \times I(C_{i-1}, t) \times I(C_{i+1}, t) \times \dots \times I(C_n, t)$

$C_i \min \leq \text{card}(\{c_i \in I(C_i, t) \mid (c_1, c_2, \dots, c_{i-1}, \dots, c_n) \in \mathcal{R}(t)\}) \leq C_i \max$

*card* : à tout instant  $t$ , le nombre d'instances de  $C_i$  en relation avec les autres

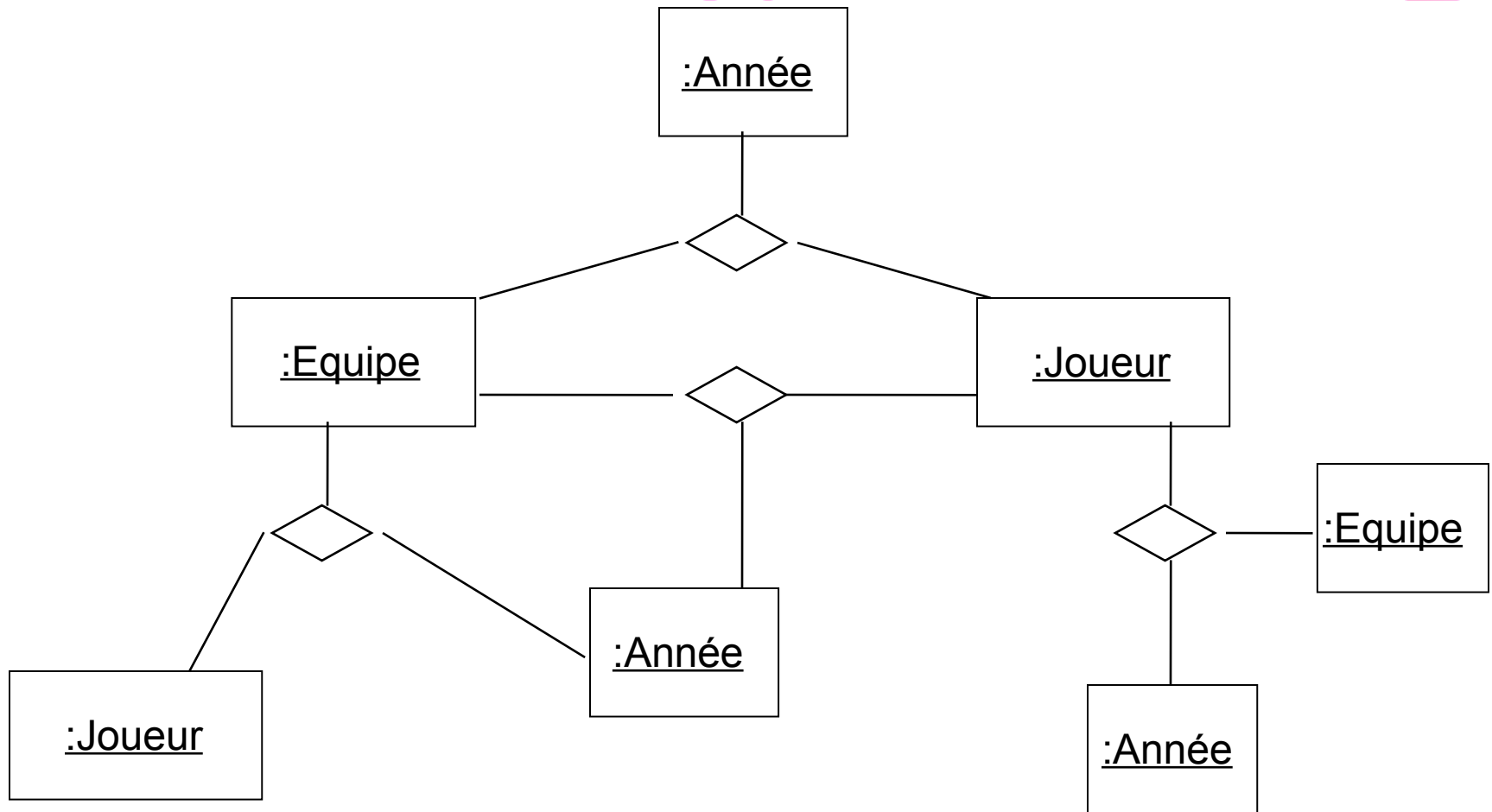
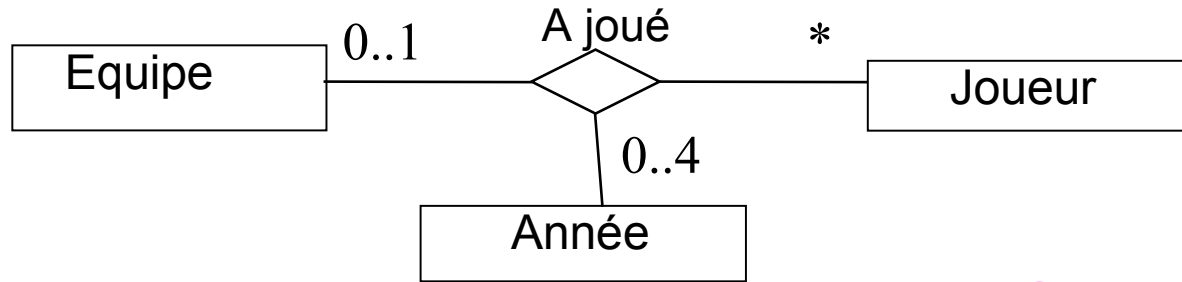
objets est compris entre  $C_i \min$  et  $C_i \max$ .

## 2. Les associations ternaires

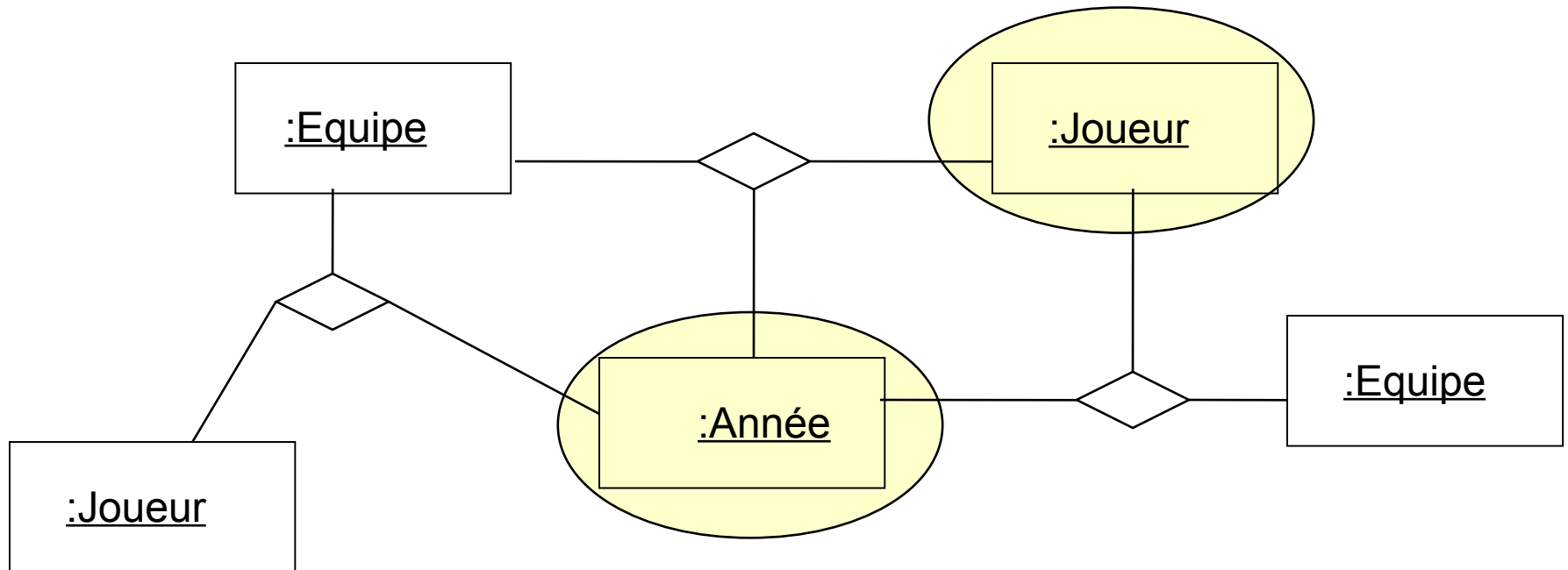
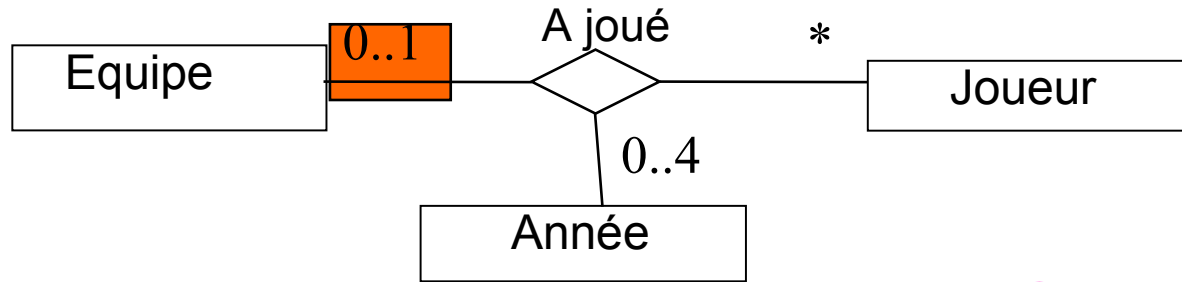


**Sémantique :**

- 
- 
-



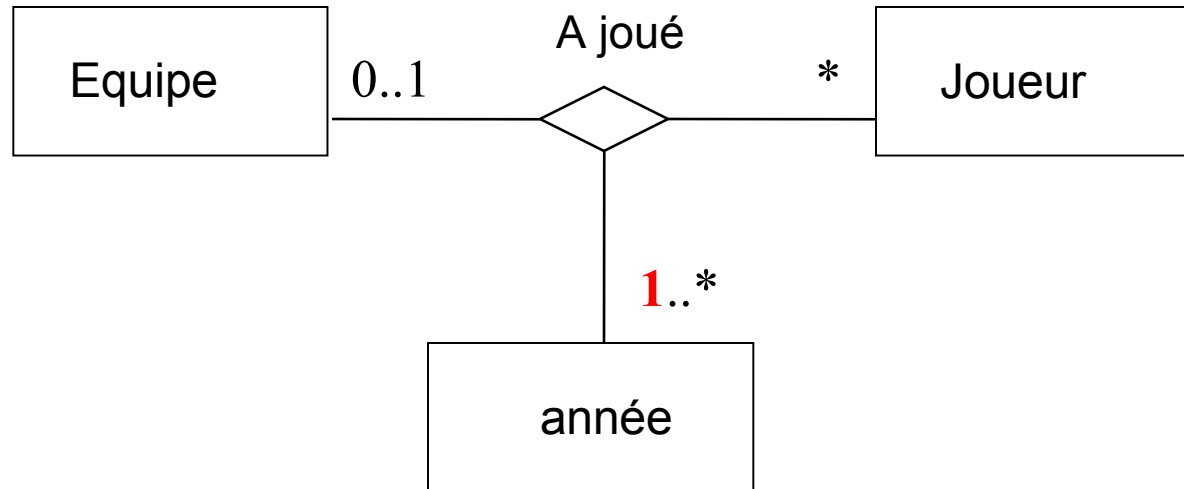
Une instance compatible



Une instance non compatible



## Attention aux cardinalités :



Un joueur doit avoir joué dans toutes les équipes!

Car à tout couple (*équipe, joueur*) doit correspondre au moins une année

**Les cardinalités min sont à mettre à 0 dans 99.99% des cas!**



Les N-aires sont un mal peu fréquent mais nécessaire.

Essayons de modéliser avec les outils connus le domaine suivant.

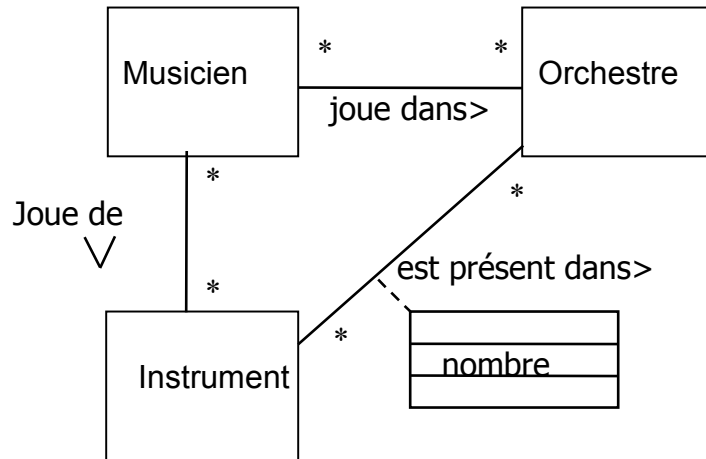
*(1) Un musicien joue de plusieurs instruments dans des orchestres.*

*(2) Il peut jouer dans plusieurs orchestres.*

*(3) Dans un orchestre, il ne peut jouer que d'un seul instrument.*

*(4) Dans un orchestre on joue de tous les instruments*

*(5) Dans un orchestre, un même instrument peut être joué par plusieurs personnes*



## 1ère approche : binaires

(1) :

(2) :

(3) :

(4) :

(5) :

*(1) Un musicien joue de plusieurs instruments dans des orchestres.*

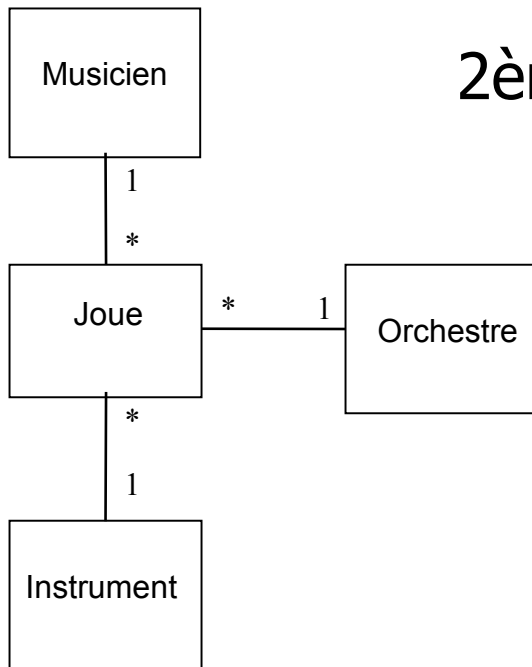
*(2) Il peut jouer dans plusieurs orchestres.*

*(3) Dans un orchestre, il ne peut jouer que d'un seul instrument.*

*(4) Dans un orchestre on joue de tous les instruments*

*(5) Dans un orchestre, un même instrument peut être joué par plusieurs personnes*

## 2ème approche : Réification



(1) :

(2) :

(3) :

(4) :

(5) :

*(1) Un musicien joue de plusieurs instruments dans des orchestres.*

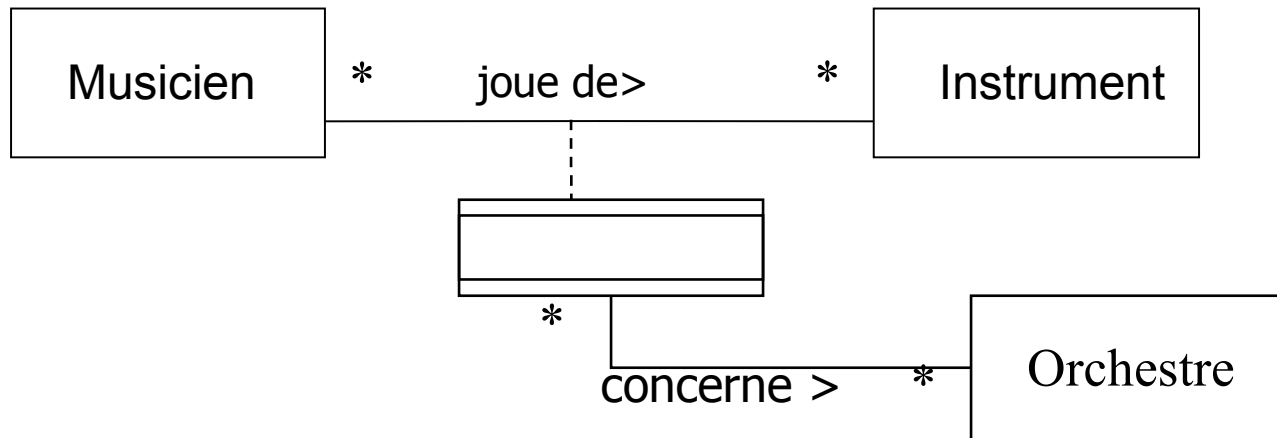
*(2) Il peut jouer dans plusieurs orchestres.*

*(3) Dans un orchestre, il ne peut jouer que d'un seul instrument.*

*(4) Dans un orchestre on joue de tous les instruments*

*(5) Dans un orchestre, un même instrument peut être joué par plusieurs personnes*

### 3ème approche : classe association



(1) :  
(2) :  
(3) :  
(4) :  
(5) :

*(1) Un musicien joue de plusieurs instruments dans des orchestres.*

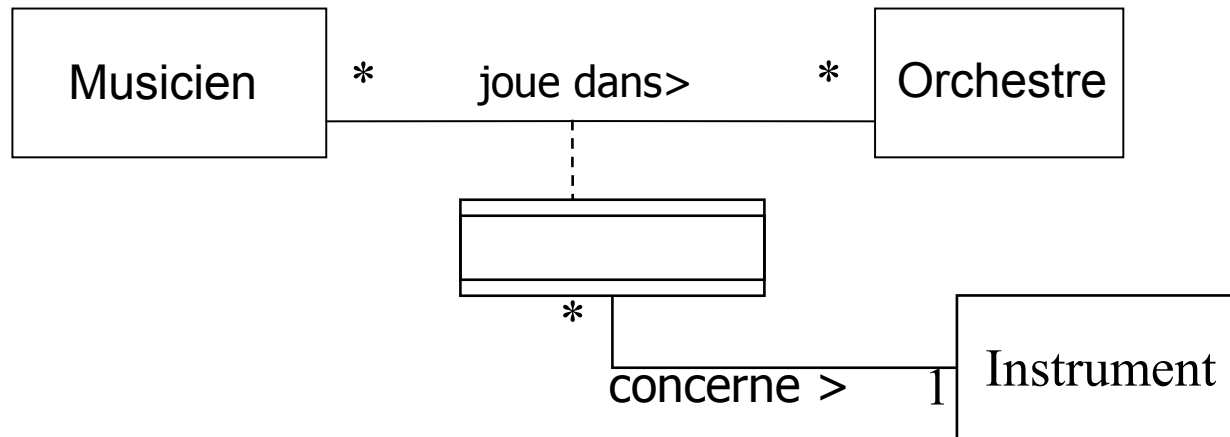
*(2) Il peut jouer dans plusieurs orchestres.*

*(3) Dans un orchestre, il ne peut jouer que d'un seul instrument.*

*(4) Dans un orchestre on joue de tous les instruments*

*(5) Dans un orchestre, un même instrument peut être joué par plusieurs personnes*

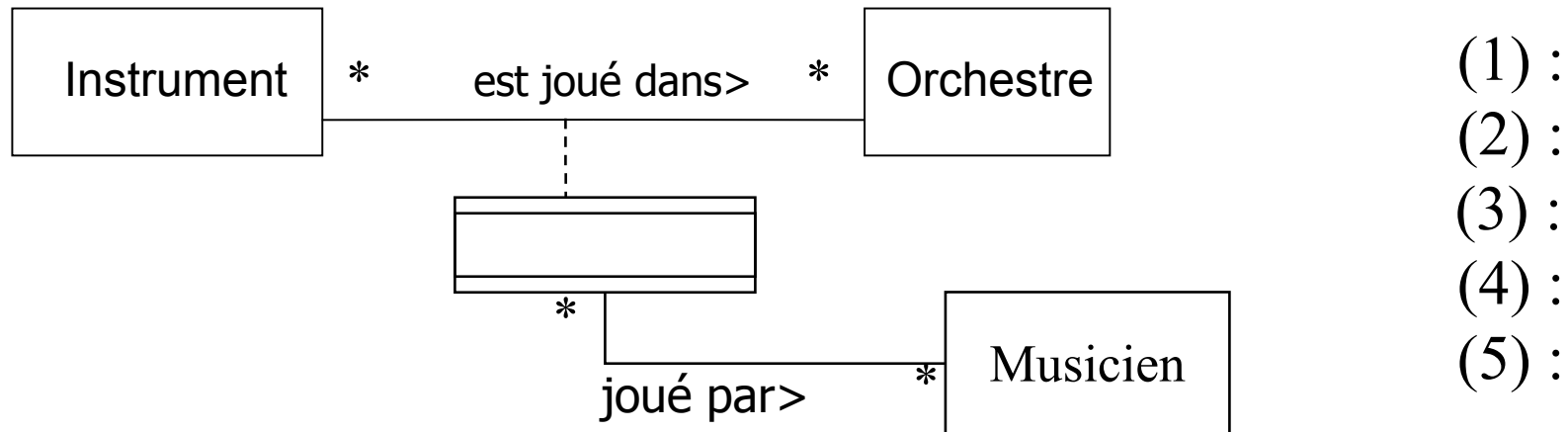
## 4ème approche : classe association bis



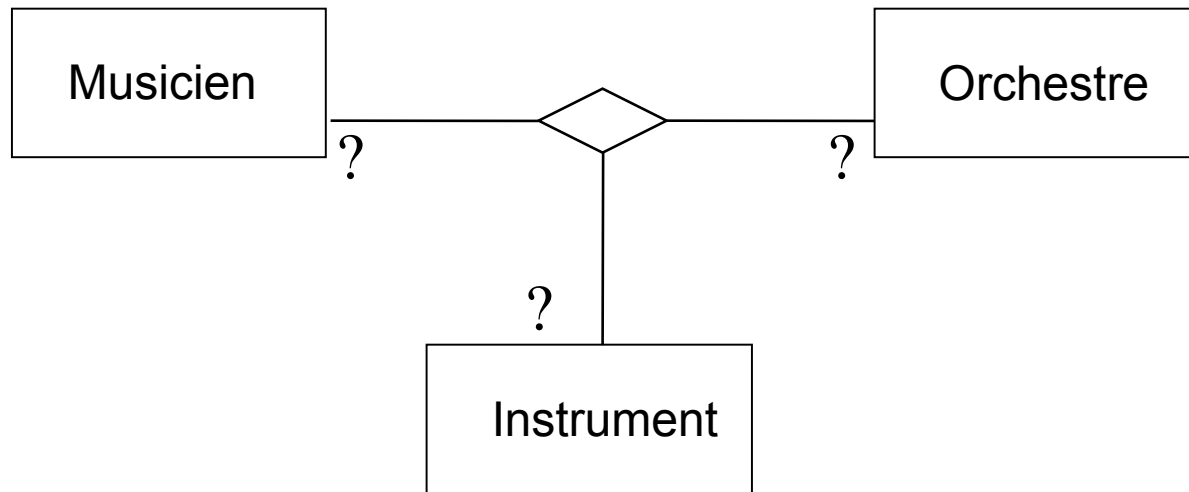
(1) :  
(2) :  
(3) :  
(4) :  
(5) :

- (1) *Un musicien joue de plusieurs instruments dans des orchestres.*
- (2) *Il peut jouer dans plusieurs orchestres.*
- (3) *Dans un orchestre, il ne peut jouer que d'un seul instrument.*
- (4) *Dans un orchestre on joue de tous les instruments*
- (5) *Dans un orchestre, un même instrument peut être joué par plusieurs personnes*

## 5ème approche : classe association Tris



- (1) Un musicien joue de plusieurs instruments dans des orchestres.
- (2) Il peut jouer dans plusieurs orchestres.
- (3) Dans un orchestre, il ne peut jouer que d'un seul instrument.
- (4) Dans un orchestre on joue de tous les instruments
- (5) Dans un orchestre, un même instrument peut être joué par plusieurs personnes



- (1) *Un musicien joue de plusieurs instruments dans des orchestres.*
- (2) *Il peut jouer dans plusieurs orchestres.*
- (3) *Dans un orchestre, il ne peut jouer que d'un seul instrument.*
- (4) *Dans un orchestre on joue de tous les instruments*
- (5) *Dans un orchestre, un même instrument peut être joué par plusieurs personnes*



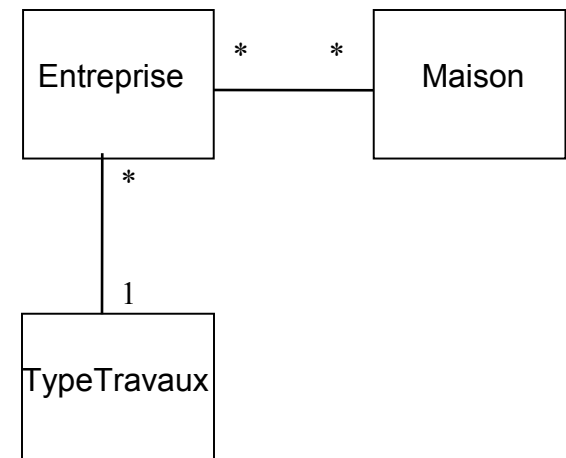
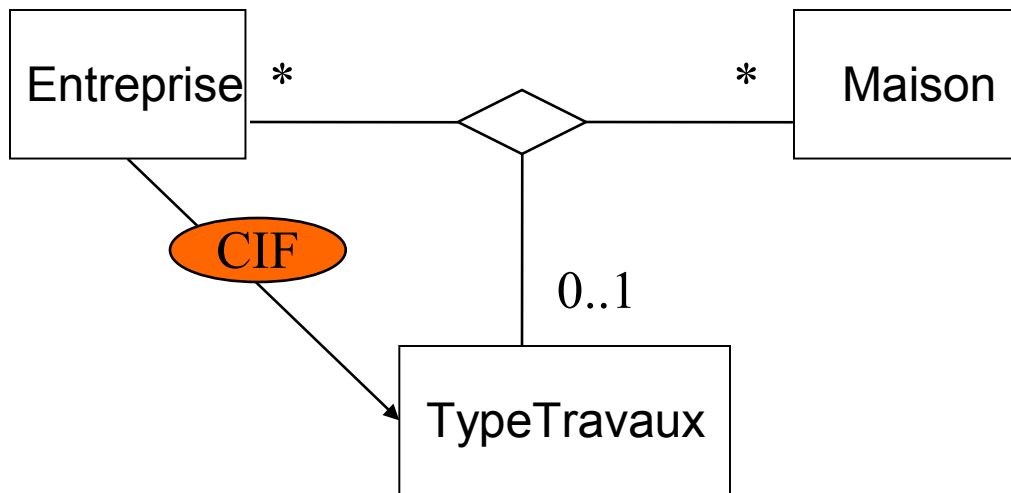
# Mais attention...

Les n-aires sont parfois simplifiables

*Une maison a connu différents types de travaux.*


*Un type de travaux a été réalisé par plusieurs entreprises.*

*Une entreprise est spécialisée dans un et un seul type de travaux.*



**Il y a une Contrainte d'Intégrité Fonctionnelle**

**(Dépendance fonctionnelle entre Entreprise et travaux)**



La simplification est toujours possible si l'arité de l'unique CIF est inférieure strictement à celle de l'association.

(cad que le nb de classe concernant la CIF est strictement inférieur au nb de classe de la ternaire.)

Cependant plusieurs dépendances fonctionnelles peuvent coexister au sein d'une N-aire et celles d'arité N empêche alors toute décomposition

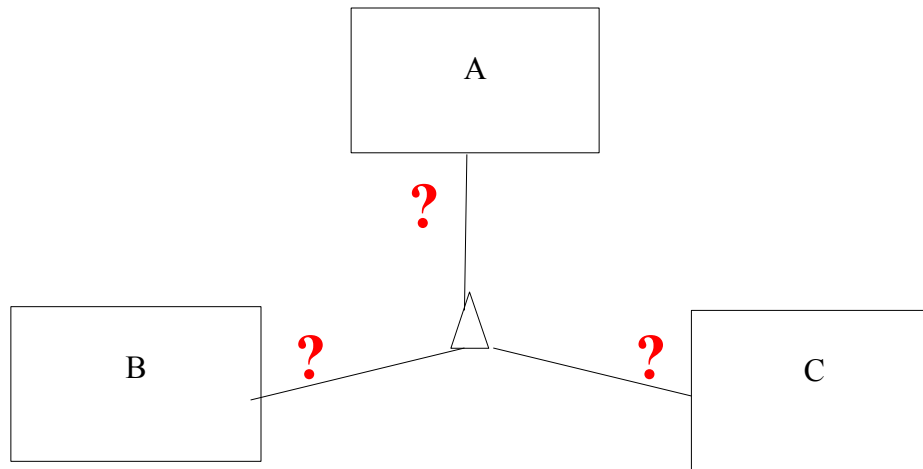
Soient les ensembles :

$A = \{x, y, z\}$ ,  $B = \{*, /\}$ ,  $C = \{4, 5\}$ .

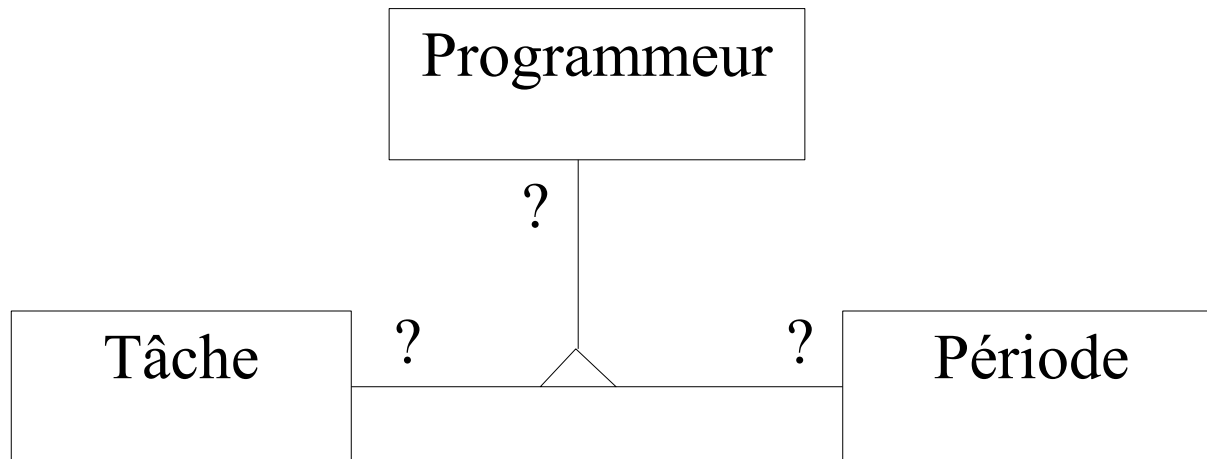
Soit  $R$  la relation construite sur  $A$ ,  $B$  et  $C$  :

$R = \{(x, *, 4), (x, *, 5), (x, /, 4), (y, *, 4), (y, *, 5), (y, /, 5), (z, *, 5)\}$

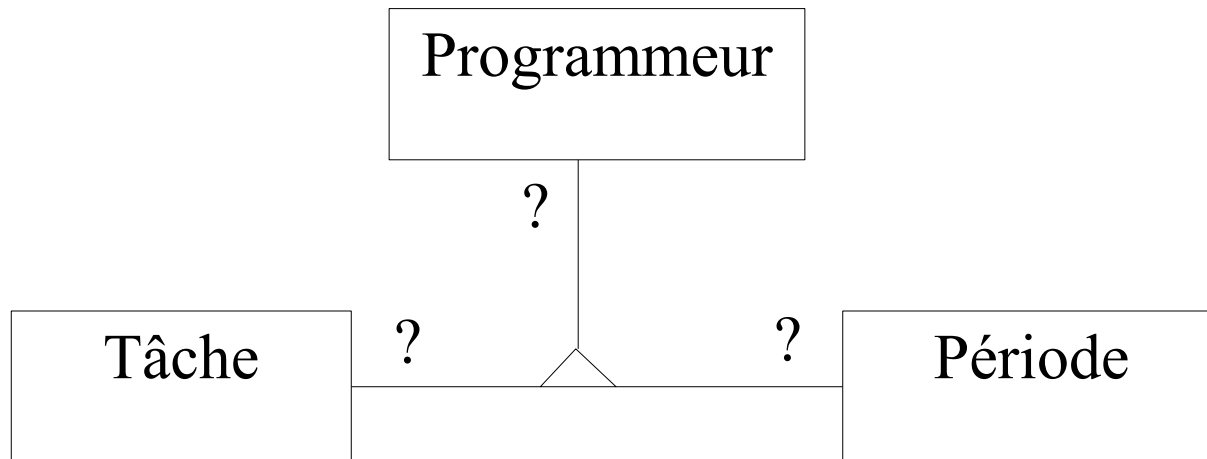
Donnez une représentation UML capable d'engendrer cette relation avec une **association ternaire** UML ayant des domaines de cardinalité **minimaux**.



***Une société de service souhaite conserver l'ensemble des tâches effectuées par ses Analystes Programmeurs. Elle conserve les périodes pendant lesquelles une tâche doit être effectuée. Un programmeur ne peut pas travailler sur plusieurs tâches à la fois et il n'utilise qu'une période donnée pour travailler sur une tâche. Une tâche peut être effectuée par plusieurs programmeurs pendant la même période.***

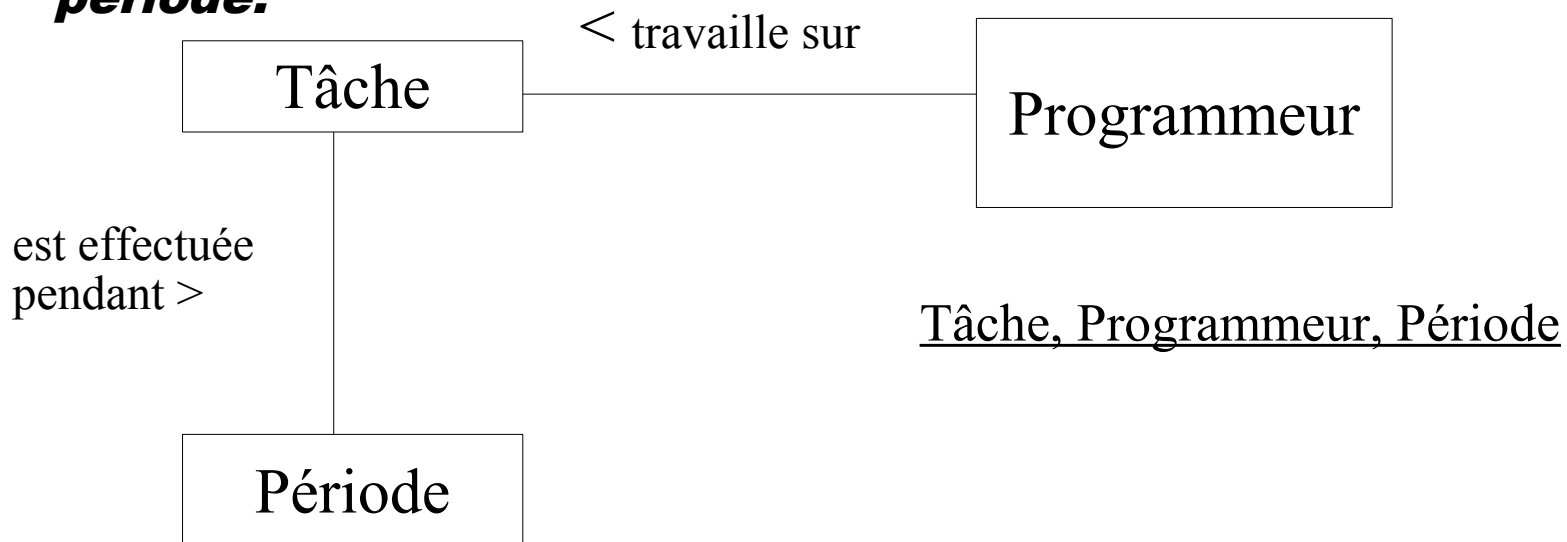


**Une société de service souhaite conserver l'ensemble des tâches effectuées par ses Analystes Programmeurs. Elle conserve les périodes pendant lesquelles une tâche doit être effectuée. Un programmeur ne peut travailler pendant une période donnée que sur au plus une tâche, et une tâche peut être effectuée par plusieurs programmeurs pendant la même période.**



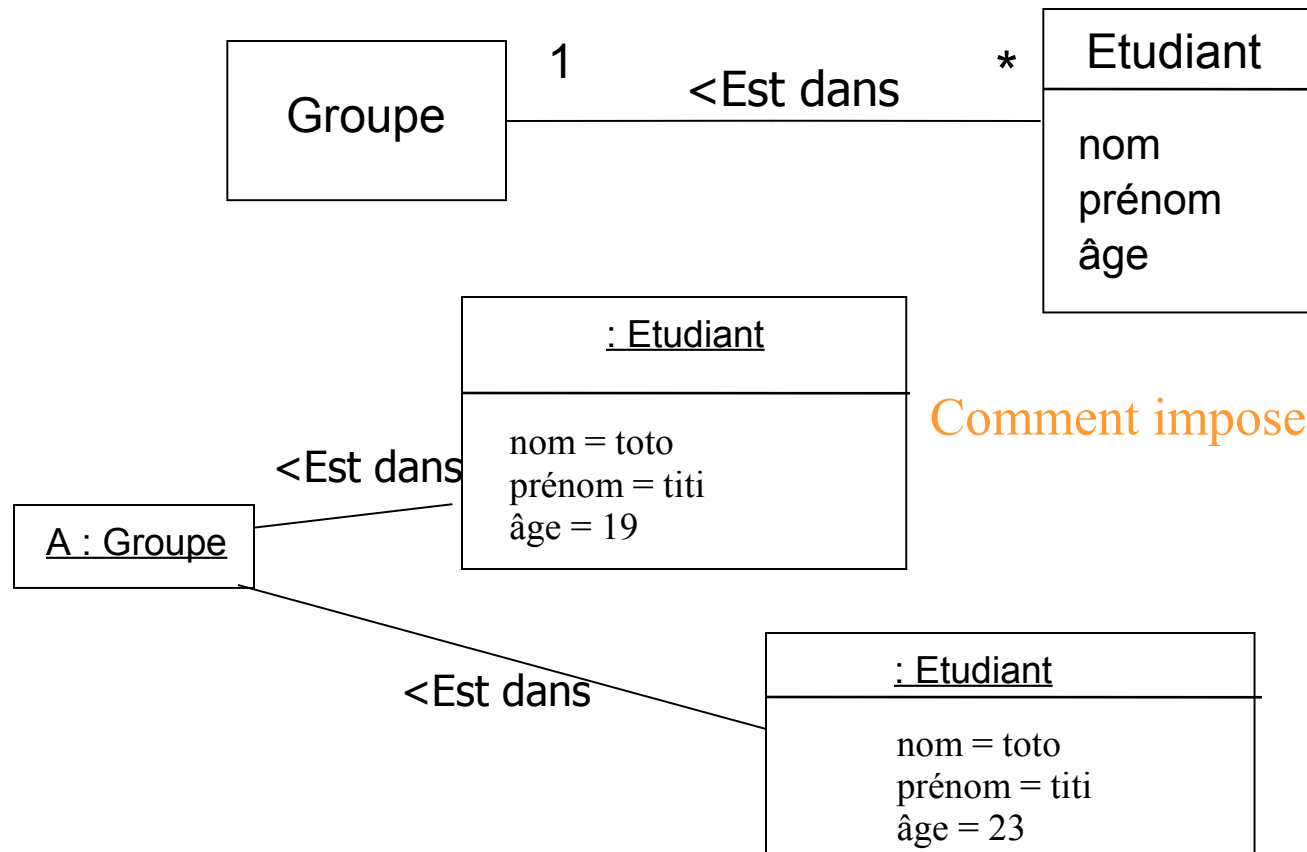
***Une société de service souhaite conserver l'ensemble des tâches effectuées par ses Analystes Programmeurs. Elle conserve la période pendant laquelle une tâche doit être effectuée. Un programmeur peut travailler pendant une période donnée sur plusieurs tâches, et une tâche peut être effectuée par plusieurs programmeurs pendant la même période.***

**Une société de service souhaite conserver l'ensemble des tâches effectuées par ses Analystes Programmeurs. Elle conserve la période pendant laquelle une tâche doit être effectuée. Un programmeur ne peut travailler pendant une période donnée que sur au plus une tâche, et une tâche peut être effectuée par plusieurs programmeurs pendant la même période.**



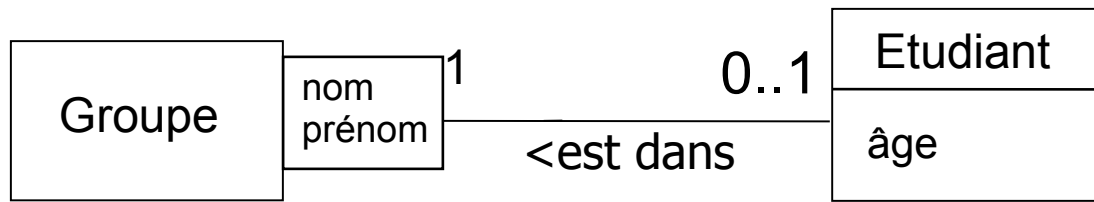
# 3. Les qualificateurs

*Dans un groupe, deux étudiants ne peuvent pas avoir le même nom/prénom*



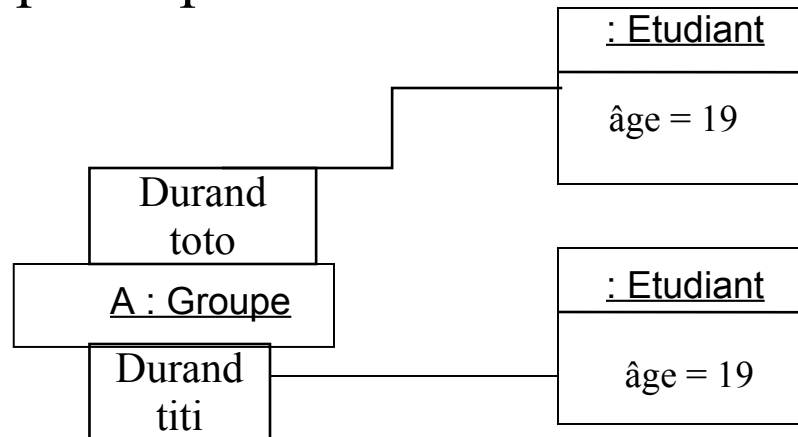
Comment imposer cette contrainte?





Se lit : Dans 1 Groupe le couple *(nom, prénom)* identifie au plus 1 étudiant

parmi ceux qui participent à l'association « *est dans* ».

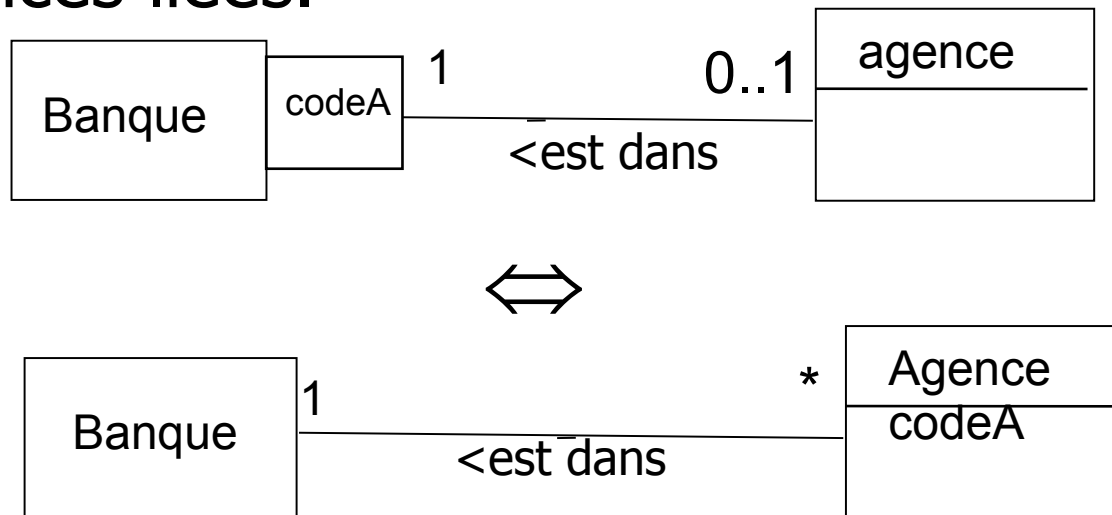


à un triplet (Groupe, Prénom, Nom) est associé au plus un étudiant.

À un étudiants est associé 1 triplet (Groupe, Prénom, Nom) .

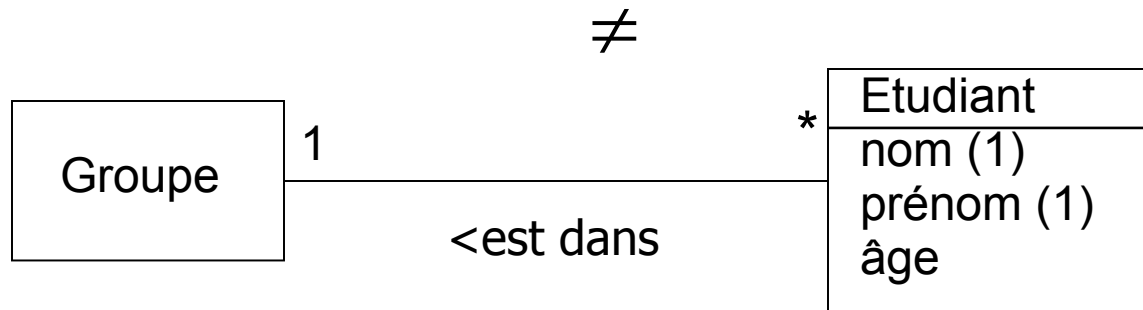
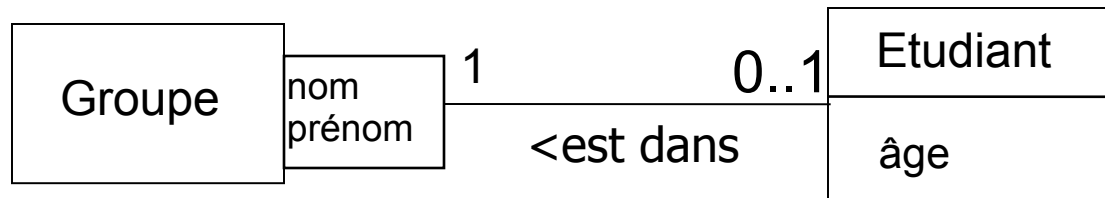
La qualification ne concerne en UML que les associations binaires

Elle ne réduit pas les liens d'une association mais elle vient poser une contrainte sur la valeur des attributs des instances liées.

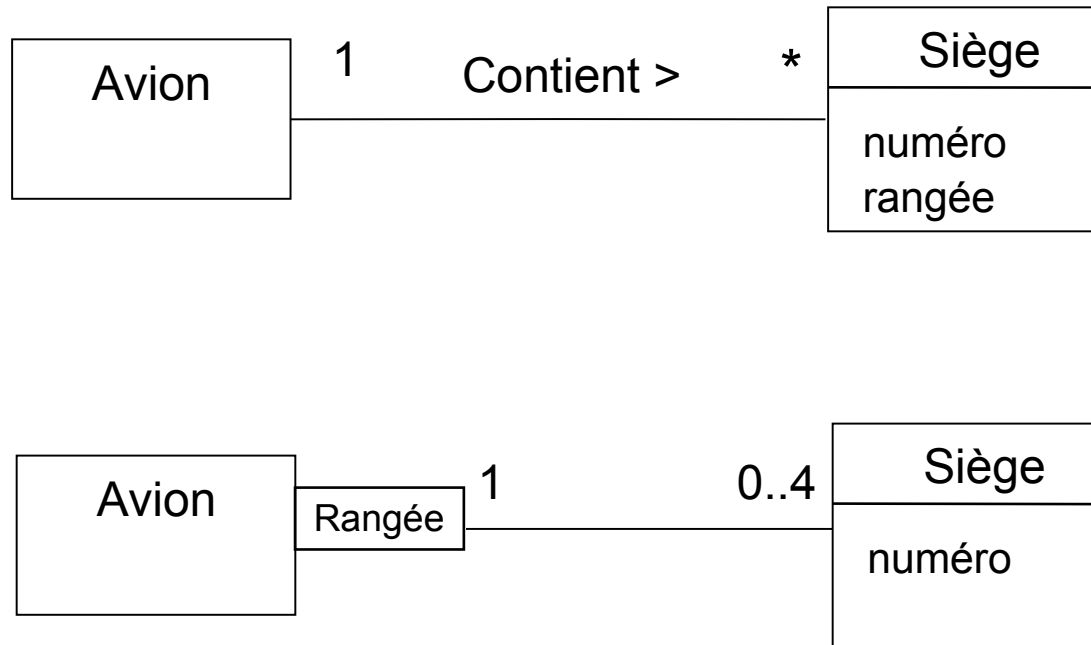


Dans une banque, deux agences ont des `codeA`  $\neq$

La qualification est essentiellement utilisée pour marquer graphiquement et explicitement les **identifiants faibles**



La restriction « virtuelle » est très majoritairement de plusieurs (\*) vers 1 mais pas seulement





Un qualificateur peut être :

des attributs de la classe

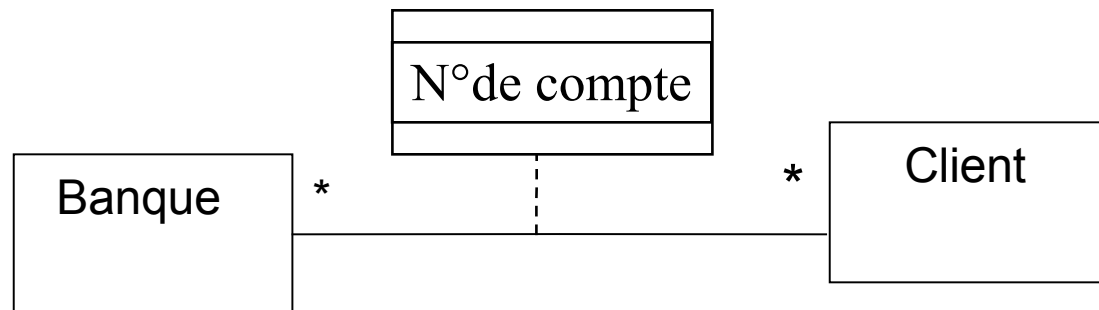
mais aussi

et/ou des attributs portés par l'association

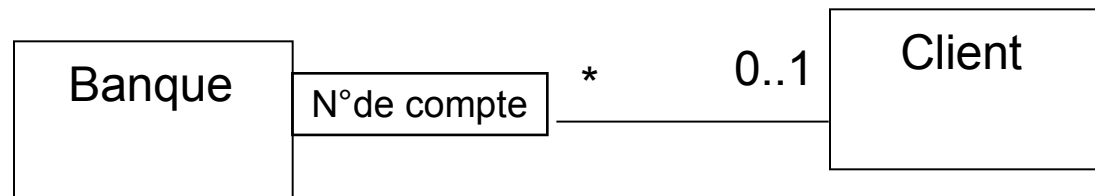
Cas attribut porté à l'origine :

*Dans une banque donnée un numéro de compte n'est attribué qu'à au plus un client.*

D1



D2



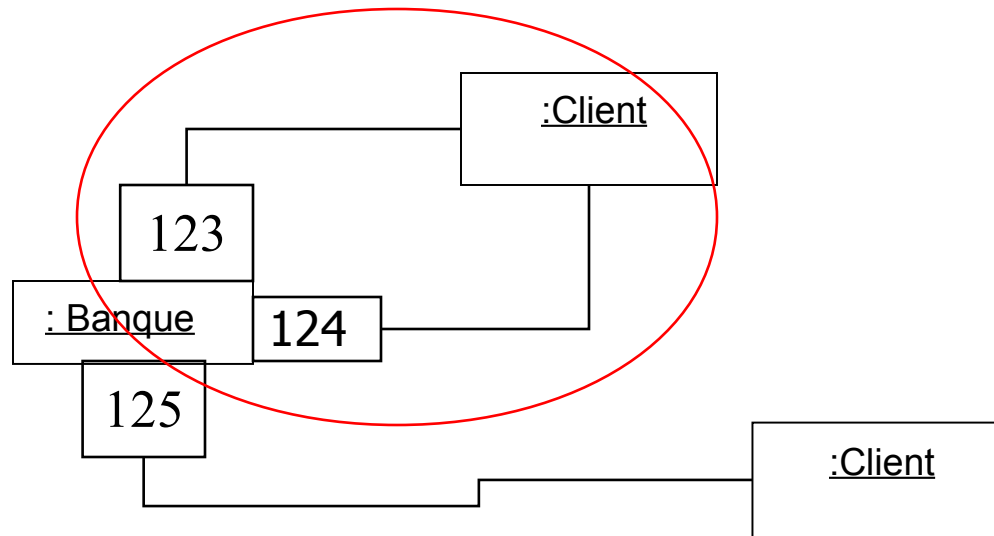


En fait les deux diagrammes précédents ne sont pas équivalents.

Car \* manifeste le nombre des couples (*banque*, *numéro*) auxquels peuvent être associés un client et non le nombre de banque uniquement!

**Conséquence** : avec D2 un Client peut avoir plusieurs comptes dans la même banque via des numéros de compte différents, ce que ne permet pas D1.

Compatible avec D2, pas avec D1 !!!

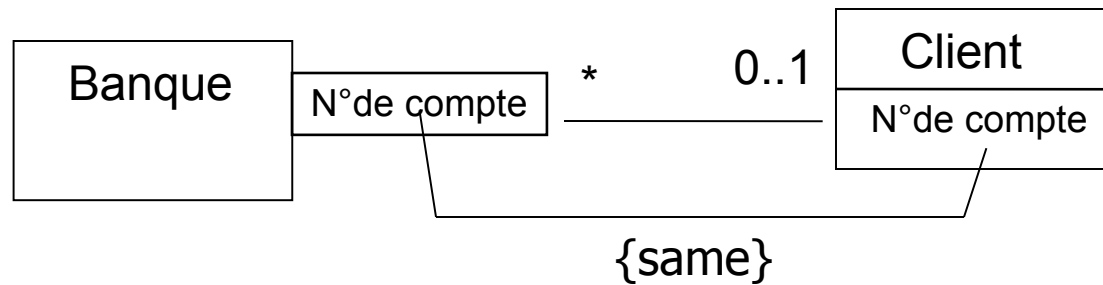




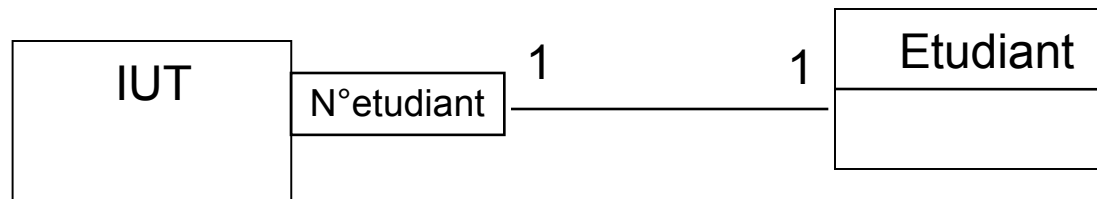


La qualification avec un \* coté qualificateur  
introduit donc la possibilité d'avoir plusieurs  
liens entre deux instances!

# Voici le diagramme qualifié équivalent au diagramme non qualifié



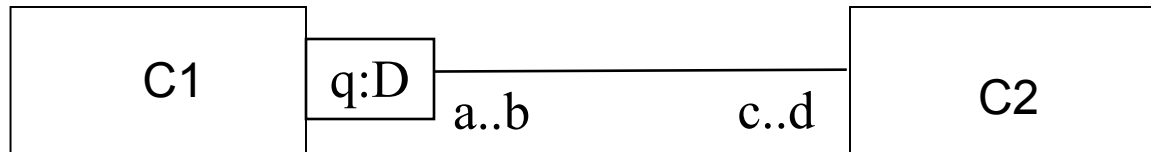
Ici, le client n'ayant qu'un seul n° de compte, il ne peut pas avoir plusieurs comptes dans une même banque.



Si le qualificateur a un domaine très grand (float, double), il est peu probable que l'on puisse mettre à 1 la cardinalité coté non qualifié.

Car sinon ici, cela signifie que chaque *IUT* a autant d'*étudiants* qu'il y a de valeurs possibles de *n°étudiant*

# Comparaison association binaire qualifiée et association ternaire



Soit  $I(C_1, t)$ ,  $I(C_2, t)$  respectivement l'ensemble des instances des classes  $C_1$  et  $C_2$  à l'instant  $t$  et  $\mathcal{D}$  l'ensemble des valeurs possibles pour le qualificateur  $q$

Soit  $\mathcal{R}(t)$  la relation ternaire issue des liens instances de l'association qualifiée entre  $C_1$ ,  $C_2$  ET le domaine  $\mathcal{D}$  à l'instant  $t$

$\forall t,$

$\forall (c_1, \nu) \in I(C_1, t) \times \mathcal{D},$

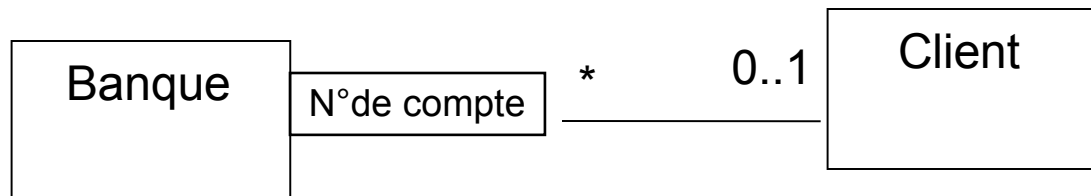
$c \leq \text{card}(\{c_2 \in I(C_2, t) \mid (c_1, c_2, \nu) \in \mathcal{R}(t)\}) \leq d$

$\mathcal{ET}$

$\forall c_2 \in I(C_2, t),$

$a \leq \text{card}(\{(c_1, \nu) \in I(C_1, t) \times \mathcal{D} \mid (c_1, c_2, \nu) \in \mathcal{R}(t)\})$   
 $\leq b$

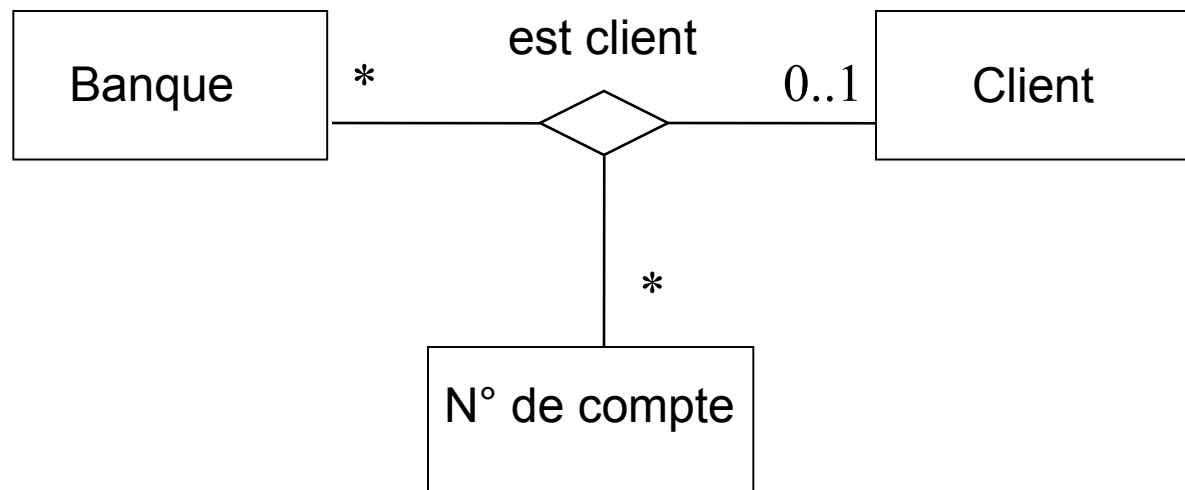
Il est important de noter le caractère dissymétrique de la lecture des multiplicités dans le sens direct qualifié vers cible (un couple vers un élément) et dans le sens inverse cible vers qualifié (un élément vers un couple).



1 couple (banque, numéro) → au plus 1 client  
1 client → \* couples (banque,numéro)

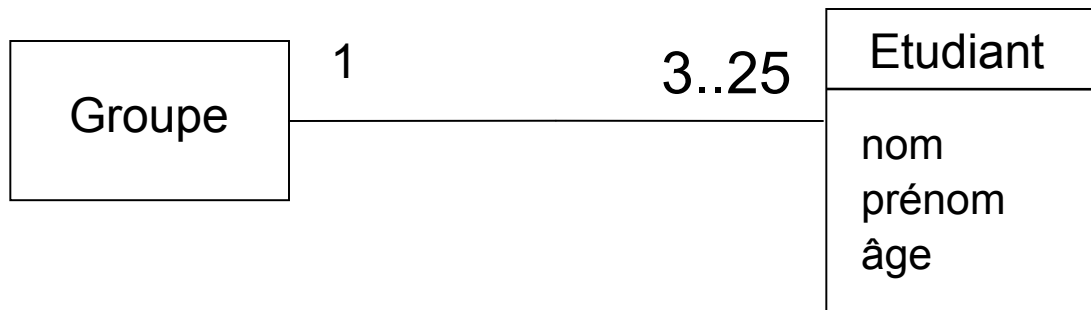
La qualification UML donne la possibilité à deux instances d'être liées par plusieurs liens instances de l'association binaire qualifiée.

Cette propriété place la qualification entre l'association binaire et l'association ternaire

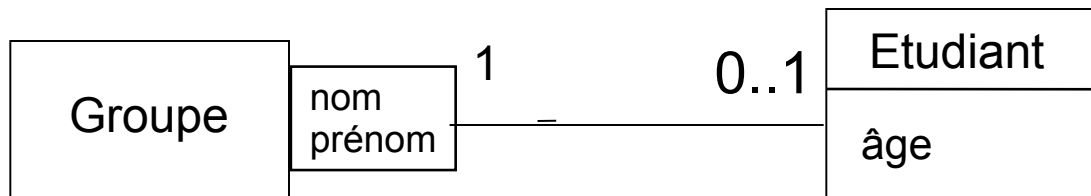


# Attention :

## La qualification peut nuire

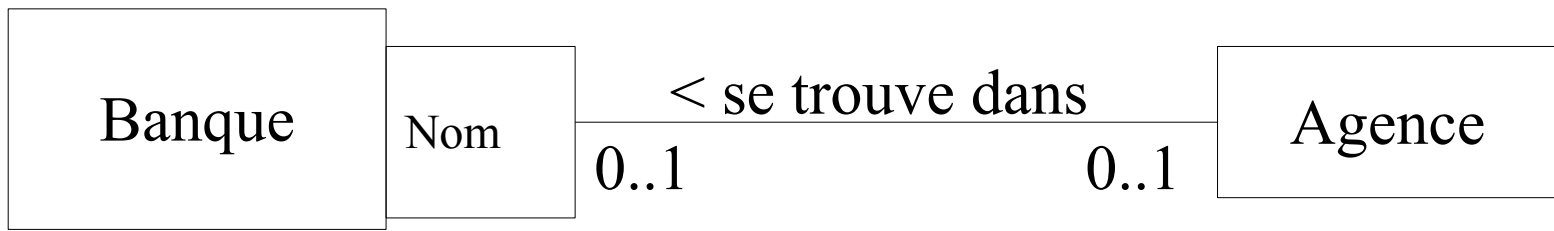


*Dans un groupe, deux étudiants ne peuvent pas avoir le même nom et prénom*

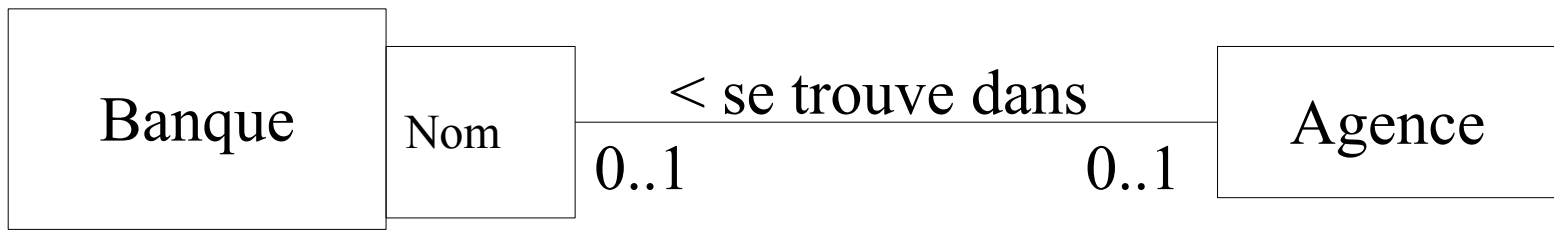


*On perd ici l'information que dans un groupe il y a entre 3 et 25 étudiants*

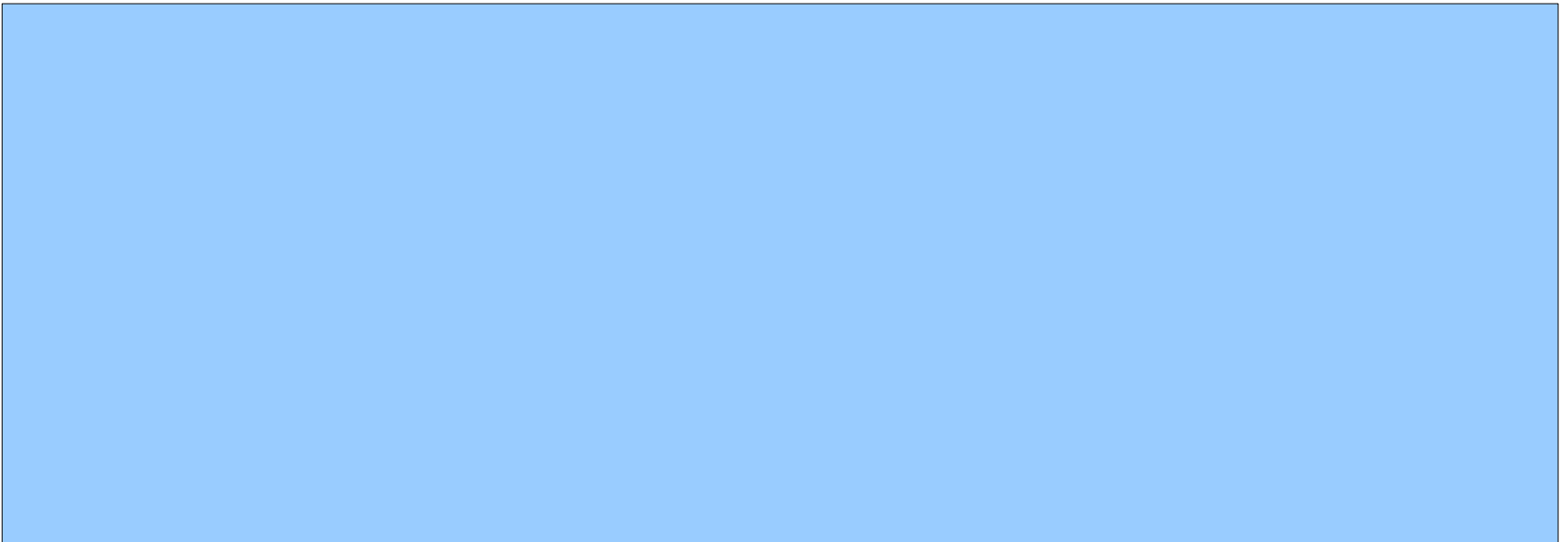


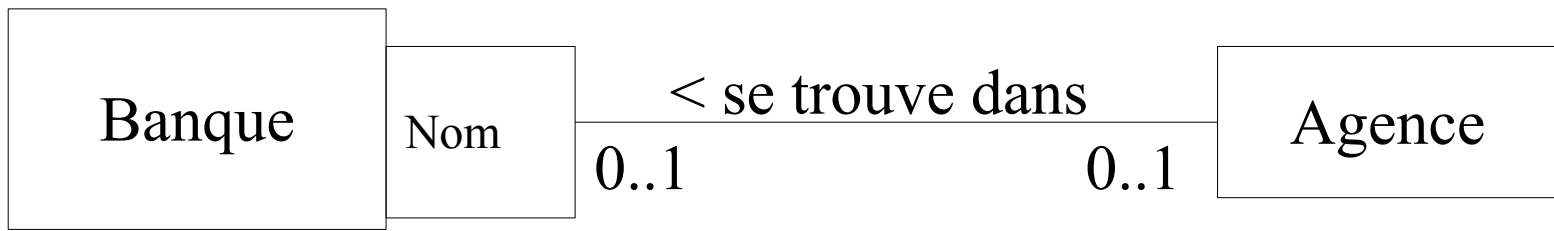


Donnez la signification textuelle de ce diagramme

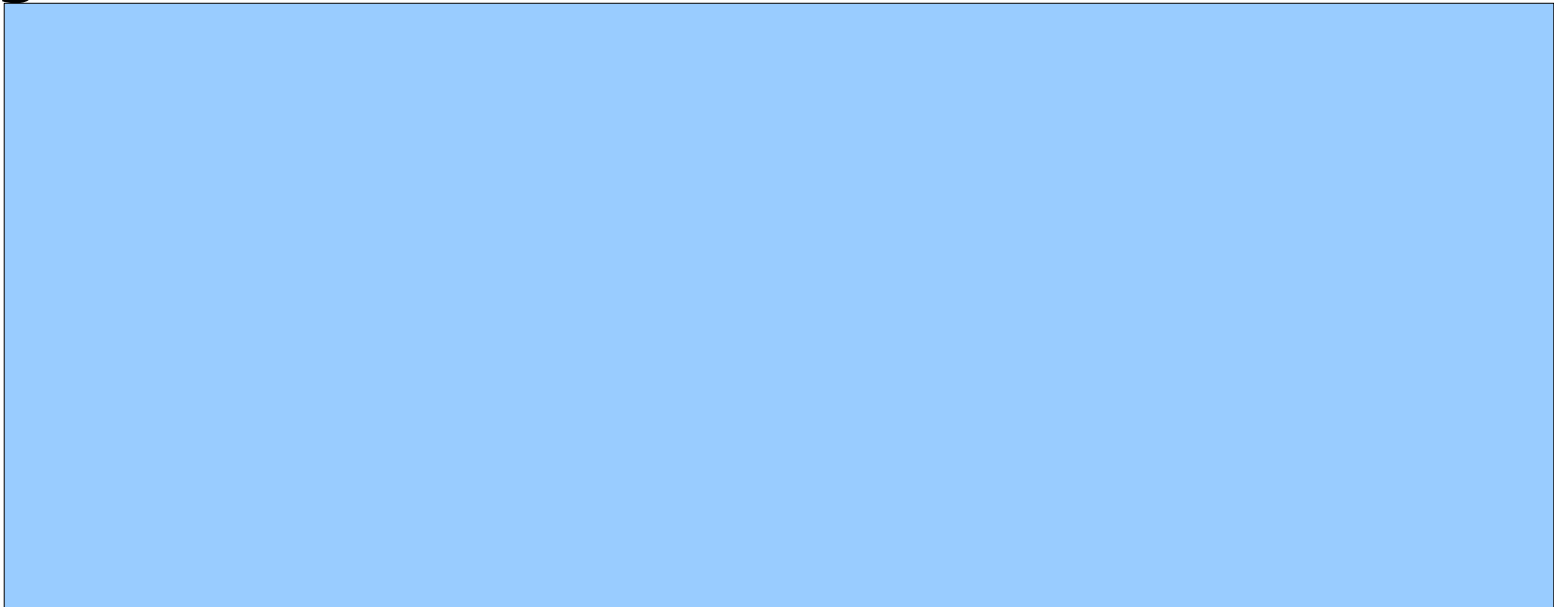


Une agence peut-elle appartenir à plusieurs Banques ? Justifiez par un diagramme d'objet.

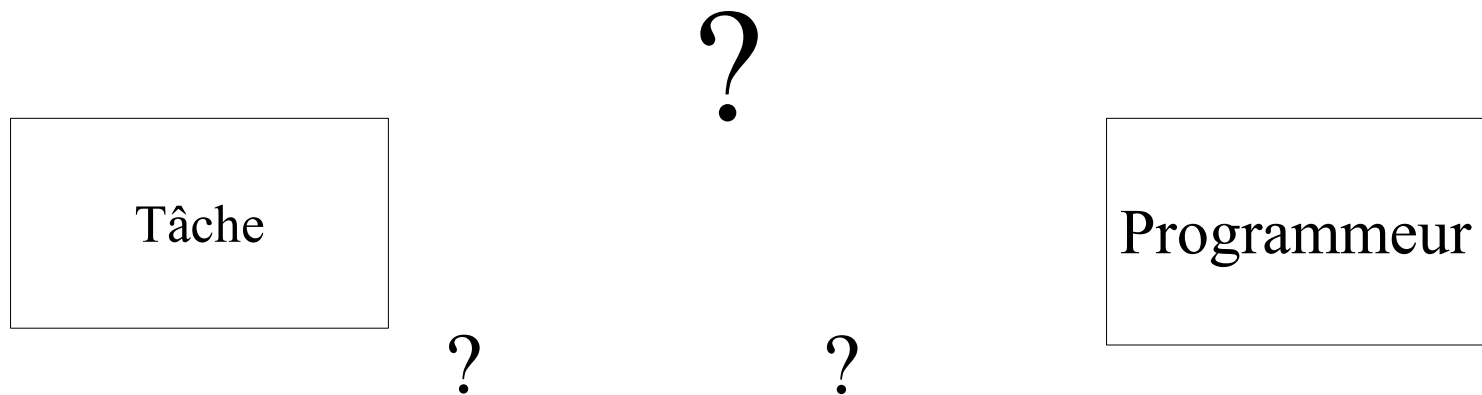




Donnez un diagramme d'objet compatible, contenant une banque, deux noms et trois agences



*Une société de service souhaite conserver l'ensemble des tâches effectuées par ses Analystes Programmeurs. Elle conserve la période pendant laquelle une tâche doit être effectuée. Un programmeur ne peut travailler pendant une période donnée que sur au plus une tâche, et une tâche peut être effectuée par plusieurs programmeurs pendant la même période.*





# **Chapitre 5**

## **Développement de la partie persistante d'une application**

# 1. Processus du Passage du diagramme de classe au schéma relationnel



On applique des « règles de traduction »  
Diagramme de classe/Schéma relationnel.

Le schéma obtenu doit préciser :

Le domaine des attributs

Les clés primaires et secondaires

Les clés étrangères


Les contraintes d'unicité et de présence obligatoire

Les contraintes statiques et les autres.



On ne s'intéresse dans les diapos suivantes qu'aux nouvelles contraintes liées aux cardinalités.

Bien sûr, toutes les contraintes issues de l'étape de modélisation devront ensuite être rajoutées.



On adopte dans ce cours, un point de vue cohérent qui vous permet de pouvoir opérer ce passage *diagramme de classes – schéma relationnel*.

Il faut savoir qu'il y a d'autres implémentations possibles avec d'excellentes raisons. Nous en donnerons un exemple en remarque.



# Règle 1



Chaque classe est traduite par une relation.  
Les attributs de la classe deviennent des attributs de la relation, et les clés de la classe deviennent des clés de la relation.



A
a1(1) a2(2) a3

$A(a1(1), a2(2), a3)$

## Règle 2 : dépendances fonctionnelles fortes




Si une association est fonctionnelle et implique une dépendance fonctionnelle forte (multiplicité 1), alors elle se traduit par une clé étrangère à présence obligatoire dans la relation source.

A		B
a1(1)	1	b1(1)
a2	*	b2

Implantation :

$A(a1(1), a2)$

$B(b1(1), b2, b3 = @A[a1] \textbf{NN})$



Si l'association est doublement fonctionnelle,  
( multiplicité 1- 0..1) il faut rajouter une  
contrainte d'unicité sur la clé étrangère : la  
clé étrangère devient une clé secondaire.

A		B
a1(1)	1	b1(1)
a2	0..1	b2

Implantation :


A(a1(1), a2)

B(b1(1), b2, b3 = @A[a1] **NN UQ**)

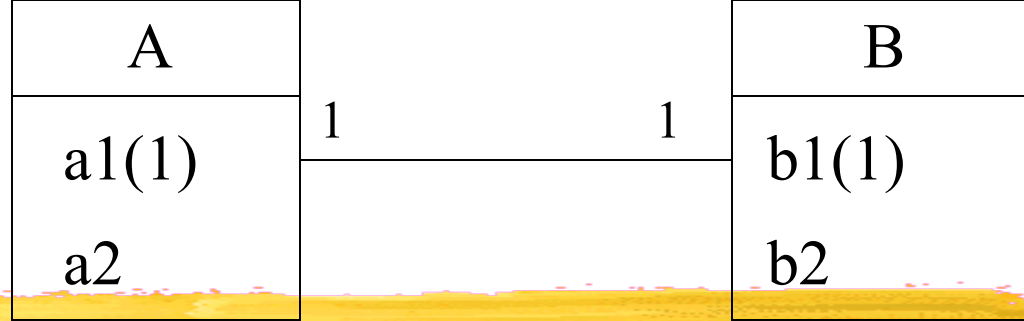
ou de façon équivalente :

A(a1(1), a2)

B(b1(1), b2, b3 = @A[a1] (2))



Si l'association est doublement fortement fonctionnelle (multiplicité 1-1), on rajoute la contrainte d'unicité (la clé étrangère devient une clé secondaire) et on rajoute une contrainte d'inclusion :



Implantation :

$A(a1(1), a2)$

$B(b1(1), b2, b3 = @A[a1](2))$

avec la contrainte  $A[a1] \subset B[b3]$

Le déport de la clé étrangère est possible dans A.



## Règle 3 : dépendances fonctionnelles faibles




Si une association est fonctionnelle mais n'implique aucune dépendance forte (multiplicité 0..1), elle se traduit dans le schéma relationnel par une clé étrangère à présence facultative (pas de contrainte) dans la relation source de la dépendance.

A		B
a1(1)	0..1 *	b1(1)
a2		b2

Implantation :

$A(a1(1), a2)$

$B(b1(1), b2, b3 = @A[a1])$



Si la dépendance fonctionnelle est double  
(multiplicité 0..1 - 0..1) on rajoute une  
contrainte d'unicité sur la clé étrangère.

A		B
a1(1)	0..1	b1(1)
a2	0..1	b2

Implantation :

A(a1(1), a2, @b1 **UQ**)

B(b1(1), b2)

Le déport de clé se fait dans la table ou il y aura le moins de t-uples (ici A)

## Règle 4 : Pas de dépendance fonctionnelle



Si une association n'est pas fonctionnelle, alors elle se traduit dans le schéma relationnel par une nouvelle relation (appelée relation-association). Cette relation a pour attribut les deux clés étrangères constituées par les identifiants des classes de l'association. Ces deux clés forment la clé primaire de cette nouvelle relation.

A		B
a1(1)	* < c *	b1(1)
a2		b2

Implantation :

A(a1(1), a2)

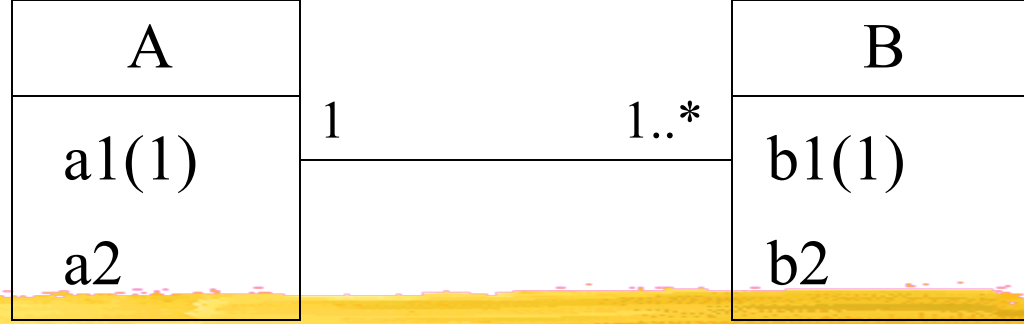
B(b1(1), b2 )

C(@a1(1), @b1(1))

## Règle 5 : dépendance fonctionnelle totale



Une Multiplicite 1..\* implique dans le schéma relationnel une dépendance fonctionnelle totale, et se traduit par une contrainte de totalité.



Implantation :

$A(a1(1), a2)$

$B(b1(1), b2, b3 = @A[a1] \text{ **NN}**)$

et  $A[a1] \subset B[b3]$



A		B
a1(1)	0..1	b1(1)
a2	1..*	b2

Implantation :

$A(a1(1), a2)$

$B(b1(1), b2, b3 = @A[a1] )$

et  $A[a1] \subset B[b3]$

A		B
a1(1)	1..* < c *	b1(1)
a2		b2

Implantation :

$A(a1(1), a2)$

$B(b1(1), b2)$

$C(c1 = @A[a1](1), c2 = @B[b1](1))$

avec  $B[b1] \subset C[c2]$



Attention aux cardinalités minimales  
différentes de 0 et 1

A		B
a1(1)	1..* < c 3..*	b1(1)
a2		b2

Implantation :

$A(a1(1), a2)$

$B(b1(1), b2)$

$C(@a1(1), @b1(1), c1)$

avec  $B[b1] \subset C[c1]$

et  $\forall x \in A[a1], \text{card}(c\{a1 = x\}) \geq 3$ .

## Règle 6 : les attribut portés



Dans une association non fonctionnelle les attributs portés sont à transmettre en attribut de la relation-association.


(les attributs portés dans une association fonctionnelle sont des abérations : ils doivent se retrouver dans l'une des deux associations)

## Règle 7 : vérification



En procédant de cette manière on peut vérifier au final, que le nombre d'association dans le diagramme de classes est égal à la somme du nombre de clés étrangères dans le schéma relationnel et du nombre de relations-associations.

## 2. Les associations internes



A	0..1
a1(1)	rôle1
a2	0..1
	rôle2

Implantation :

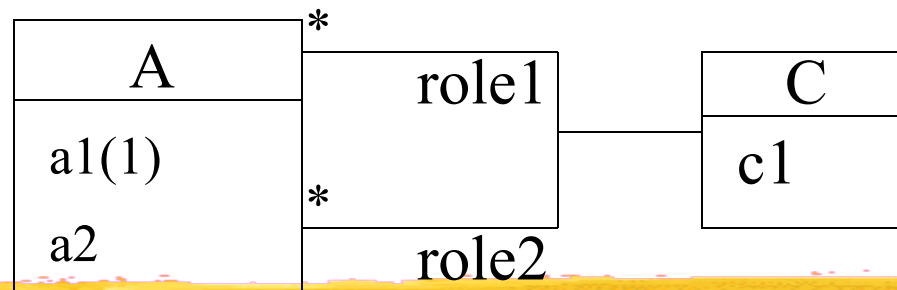
$A(\underline{a1}, a2, \text{rôle1} = @A[a1] \textbf{UQ})$

A	0..1
a1(1)	role1
a2	* role2

Implantation :

$A(a1(1), a2, \text{rôle1} = @A[a1])$



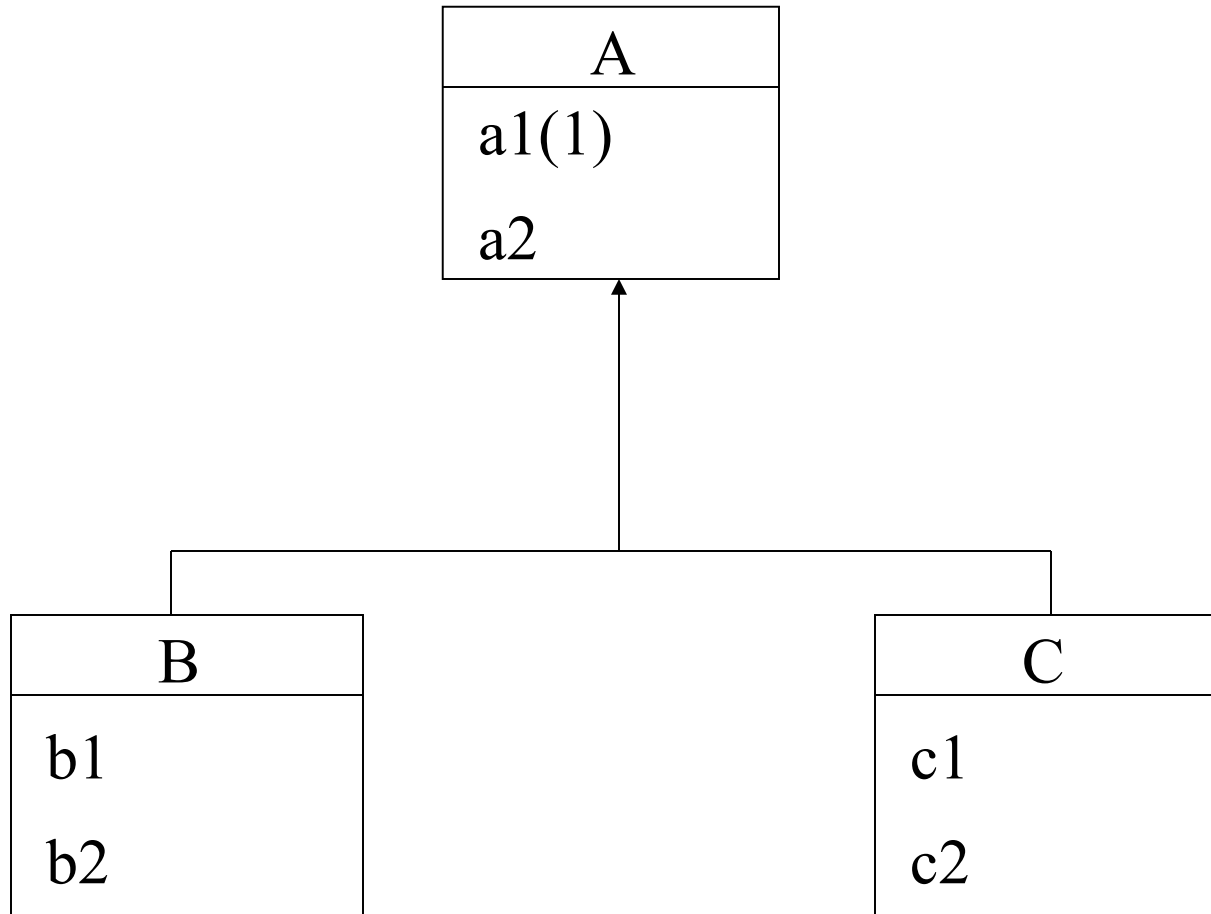


Implantation :

$A(a1(1), a2)$

$C(\text{rôle1} = @A[a1] (1), \text{rôle2} = @A[a2] (1), c1)$

### 3. La généralisation





## Implantation :

$A(\underline{a1}, a2, \text{type NN})$

$B(\underline{@a1}, b1, b2)$

$C(\underline{@a1}, c1, c2)$

## 4. Remarque



comme il a été dit précédemment, il y a d'autres possibilités suivant les cas.

Regardez le cas suivant et supposez, que la table A possède beaucoup de t-uples, et que l'association n'est réalisée que dans peu de cas. l'implémentation proposée utilisera moins de place mémoire, que pour celle proposée plus haut :

A		B
a1(1)	0..1	b1(1)
a2		b2

Implantation :

A(a1(1), a2)

B(b1(1), b2)

C(c1 = @A[a1] (1) UQ, c2 = @B[B1] (1) UQ)

au lieu de

A(a1(1), a2, @b1 **UQ**)

B(b1(1), b2)

## 5. Exemple

