

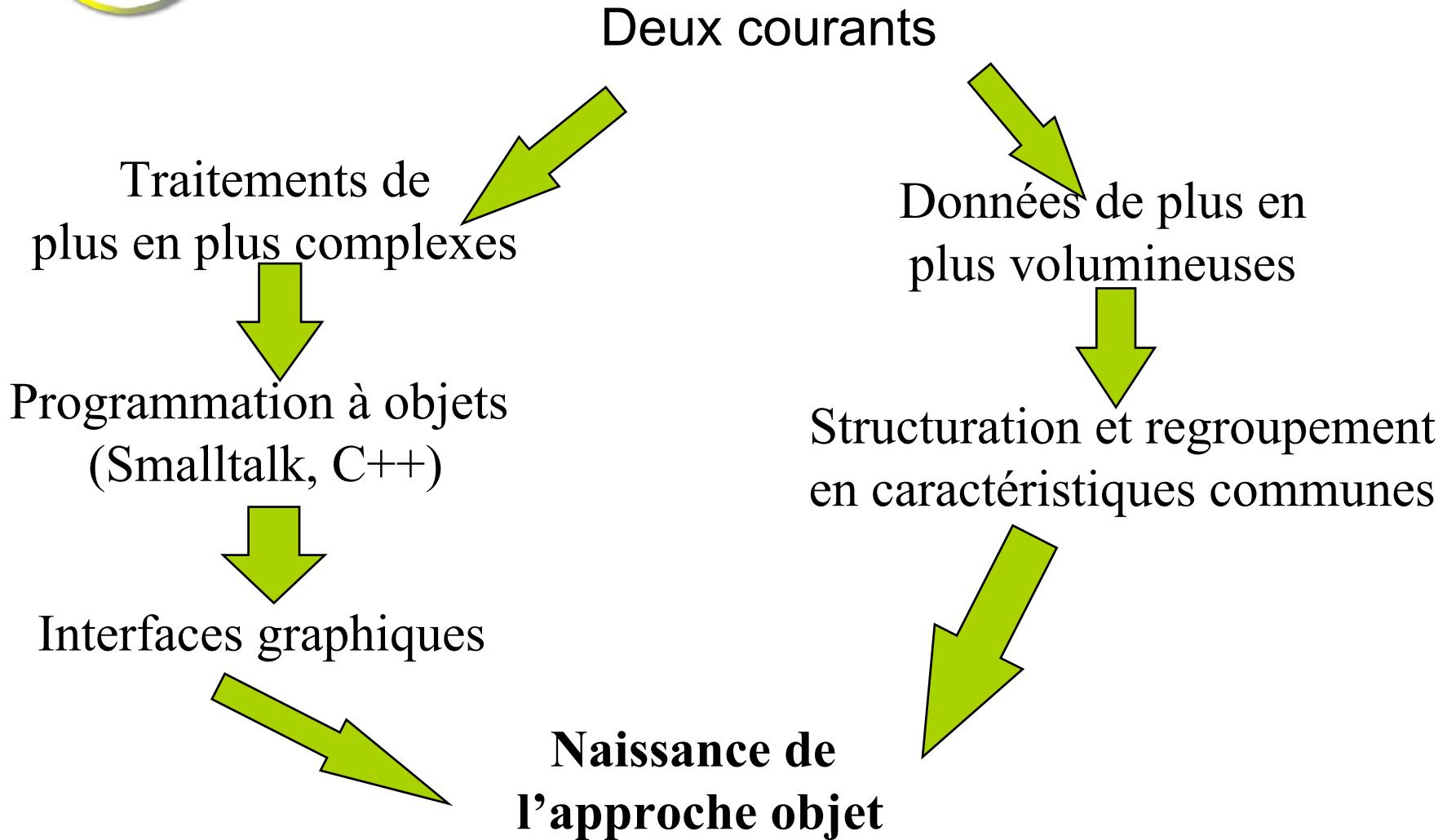


# Modélisation orientée objet

*Introduction aux concepts objet*  
*Diagramme de classe*



# Historique





## Historique

- ✿ 1966 : langage Simula (notion de classe)
- ✿ Début de l'utilisation : en Intelligence Artificielle puis dans le domaine des BD
- ✿ Apports de la modélisation objet :
  - interfaces conviviales (fenêtres, outils de pointage, pictogrammes...)
  - applications complexes (domaine médical...)
  - réutilisation



# Introduction

- ✿ Avantages de l'approche objet :

- Modélisation des objets de l'application
- Modularité des applications
- Réutilisabilité
- Extensibilité du code

- ✿ Principes :

- Abstraction
- Décomposition
- Connexion



# Notion d'objets

- ✿ Les entités conceptuelles du monde réel peuvent être représentées par des **objets**
  - Exemples d'objets :
    - Le nombre 12
    - L'étudiante Mathilde
    - Le dossier médical de Jeanne
  - Un objet est une abstraction d'une donnée du monde réel caractérisé par :
  - Objet = identité + état + comportement



# Objets

- ✿ Identité d'objet (object identifier ou OID) :
  - identifiant unique et invariant qui permet de référencer l'objet indépendamment des autres objets.
- ✿ Etat : une valeur simple ou complexe
- ✿ Comportement : ensemble d'opérations applicables à l'objet, permettant d'agir dessus.
- ✿ Exemple : Véhicule d'identité V1
  - V1 {
    - Numéro: 566GH38, Marque: Renault, Type: Clio;
    - créer(), démarrer(), rouler(), stopper(), détruire()



# Objets

- ✿ Identité et égalité d'objets sont deux notions différentes :
  - 2 objets o1 et o2 sont identiques si leurs OID sont égaux ( $o1==o2$ )
  - 2 objets sont égaux si leurs états sont égaux ( $o1=o2$ )
  - Si  $o1==o2 \Rightarrow o1=o2$



# Notion d'objets

- ✿ Ces objets servent à décrire le monde réel
- ✿ Les objets existent indépendamment les uns des autres
- ✿ Les objets possèdent des caractéristiques appelées **attributs**
  - Exemples : nom, prénom, âge, adresse sont des attributs de l'objet 'Etudiant Pierre'



# Représentation vectorielle

- ✿ Un objet est vu conceptuellement comme un triplet <OID, classe, état>
- ✿ La structure de chaque objet est un n-uplet (donné entre parenthèse)
- ✿ Exemple :
  - un employé a 2 attributs : son nom et la référence de son site
  - Un site a 3 attributs : son nom, la référence de l'employé chef, la liste [ ] des références des employés du site



# Représentation vectorielle

## Employés

- <o1, employé, (« Martin », o10)>
- <o2, employé, (« Dupont », o10)>
- <o3, employé, (« Martin », o11)>
- <o4, employé, (« Jules », o11)>
- <o5, employé, (« Jim », o11)>

## Sites

- <o10, site, (« Paris », o1, [o2,o3])>
- <o11, site, (« Grenoble », o5, [o3,o4])>



## Changements d'états d'objets

- ✿ L'identité d'objet est utile pour comprendre la sémantique des modifications d'objets comme des changements d'état
- ✿ Ex : délocalisation du site de Paris à Strasbourg
  - Avant : <o10, site, (« Paris », o1, [o2,o3])>
  - Après : <o10, site, (« Strasbourg », o1, [o2,o3])>



## Liens inter objets

- ✿ Grâce à l'OID, les liens entre objets sont représentés par des références.
- ✿ Lien monovalué : entre un objet source et un objet cible
- ✿ Lien multivalué : entre un objet source et plusieurs objets cibles

$<o10, \text{site}, (\text{« Paris »}, o1, [o2,o3])>$

Lien monovalué (entre employé et site) :  
le site o10 fait référence à l'employé chef o1

Lien multivalué :  
les employés o2 et o3 sont affectés au site o10



# Opérations et méthodes

- ✿ Une opération est une transformation qui est réalisée **par** ou **sur** un objet
- ✿ On appelle **méthode** l'écriture informatique de l'opération
- ✿ Exemple : L'attribut «âge » de l'objet « Patient Pierre» peut avoir la méthode :
  - ChangerAge



# Encapsulation

- ✿ La description d'un objet nécessite :
  - la description de sa structure (ses attributs)
  - la description de ses comportements associés à l'objet (ses méthodes)
- ✿ Méthodes et attributs ne sont pas visibles de l'extérieur de l'objet

On parle d'**encapsulation**



# Les classes d'objet

- ✿ Les objets qui possèdent les mêmes attributs et les mêmes méthodes sont regroupés dans des **classes**
- ✿ Les objets représentent des **occurrences** de la classe (on parle aussi d'instanciation)
- ✿ Dans les approches objet, tout objet appartient nécessairement à une classe



## Les classes d'objet

- ✿ Par exemple, tous les étudiants d'une université seront regroupés dans une classe ETUDIANT
- ✿ De la même manière, tous les enseignants de l'université seront regroupés au sein de la classe ENSEIGNANT



# Classe

- ✿ Elle fournit un mécanisme d'instanciation qui permet de créer un objet (opération new).
- ✿ Classe = type abstrait de données qui spécifie la structure et le comportement des objets
- ✿ Attributs = variables d'instances qui ont un nom et un type (type de base simple ou complexe ou classe).



# Classe

- ✿ Autre définition : type de donnée abstrait caractérisé par des propriétés (attributs et opérations) communes à ses objets et mécanisme permettant de créer des objets ayant ces propriétés.
- ✿ Deux niveaux de visibilité de l'objet : public et privé
  - Les opérations publiques constituent l'interface de la classe, elles sont accessibles à tout utilisateur de la classe.
  - Les opérations privées servent à supporter les opérations publiques et ne sont accessibles qu'à « l'implanteur » de la classe.



# Classe et encapsulation

- ✿ Le principe d'encapsulation impose que les attributs de la classe ne soient accessibles de l'extérieur que par appel de l'opération de lecture correspondante
- ✿ L'encapsulation permet d'isoler les programmes utilisateurs d'une classe des modifications de l'implémentation des attributs.



# Opérations

- ✿ On accède aux objets par un appel de procédures (opération).
- ✿ Ex : affectation d'un employé désigné par la variable emp au site Paris :
  - Send (emp, affecter, Paris)
- ✿ qui provoquera l'appel de l'opération :
  - Employé.affecter (oid-emp, Paris)
  - Oid-emp= référence de l'objet désigné par la variable emp



## Les liens entre les classes

- ✿ Les classes ne peuvent pas être isolées dans un modèle de données. Elles sont forcément liées entre elles.
- ✿ Si l'on veut par exemple représenter qu'un étudiant suit un cours donné par un enseignant, il est nécessaire de définir des liens entre ces différentes classes.



## Les liens entre les classes

- ✿ Il existe plusieurs natures de liens qui représentent une sémantique différente.
- ✿ En objet, on trouve les liens suivants :
  - Agrégation
  - Généralisation/spécialisation → héritage
  - Association



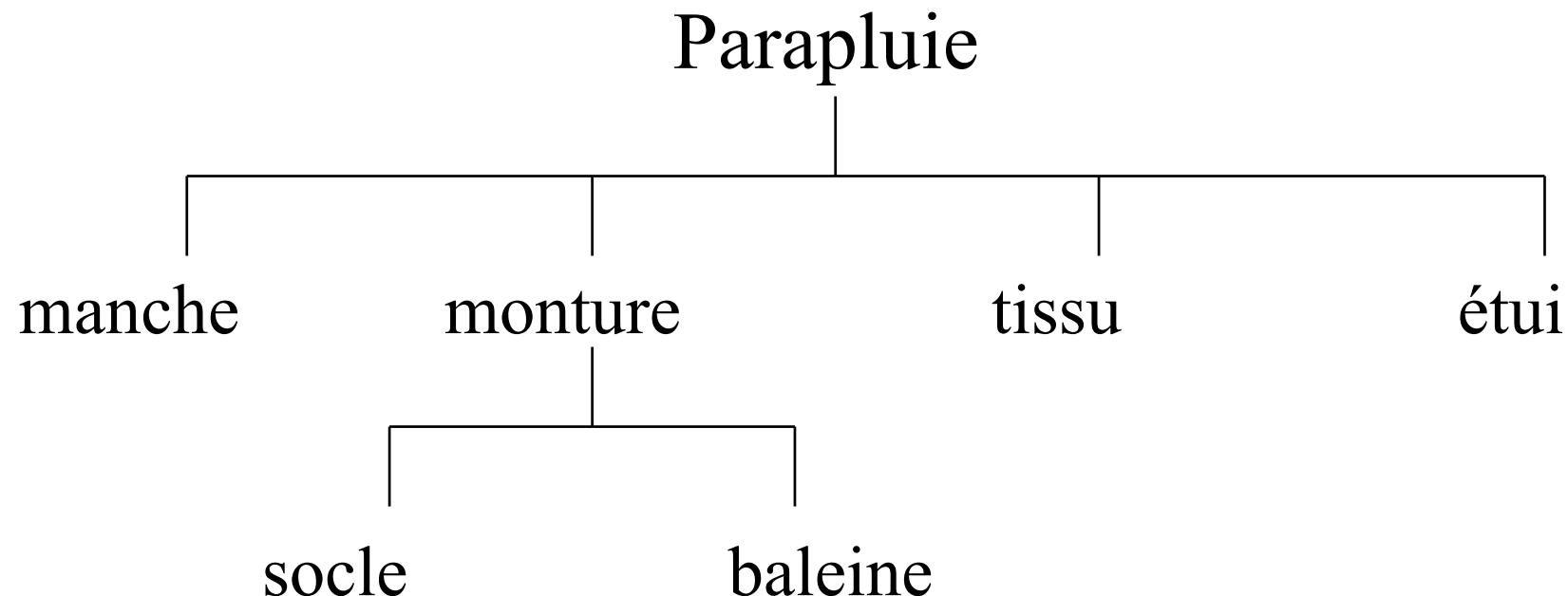
# L'agrégation

- ✿ L'agrégation permet de construire des objets complexes à partir d'autres objets appelés objets composants
- ✿ L'agrégation se matérialise par un graphe acyclique
- ✿ Le lien qui relie les objets composants signifie «composé» ou «est composé»



# L'agrégation

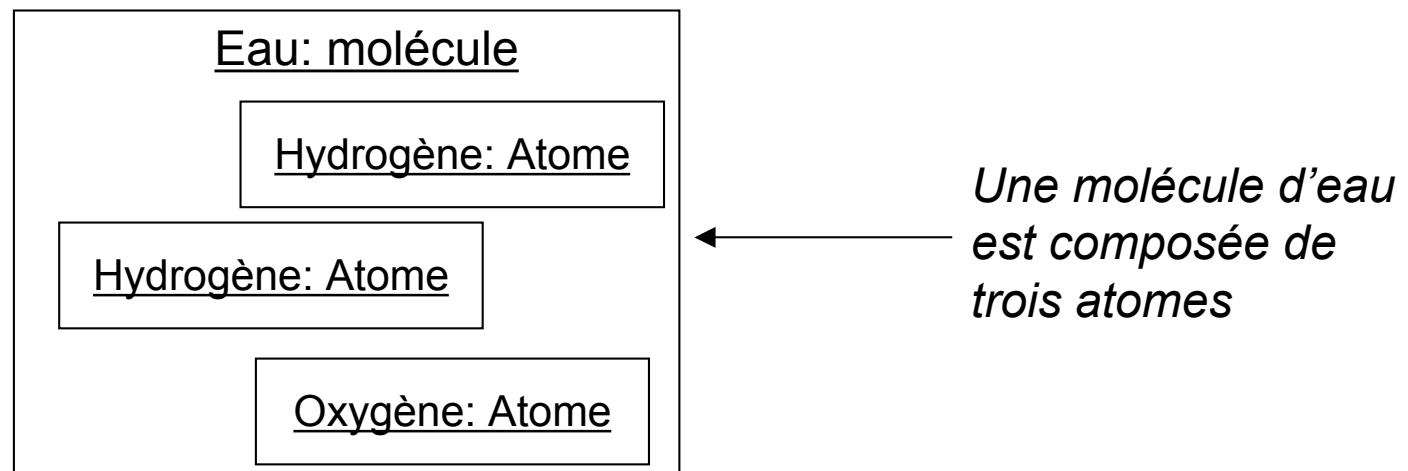
- ✿ Exemple d'agrégation : la nomenclature





# L'agrégation

- ✿ Les objets d'une classe sont les composants d'une autre classe
- ✿ Intérêt : partir d'objets de base pour arriver à des objets plus complexes



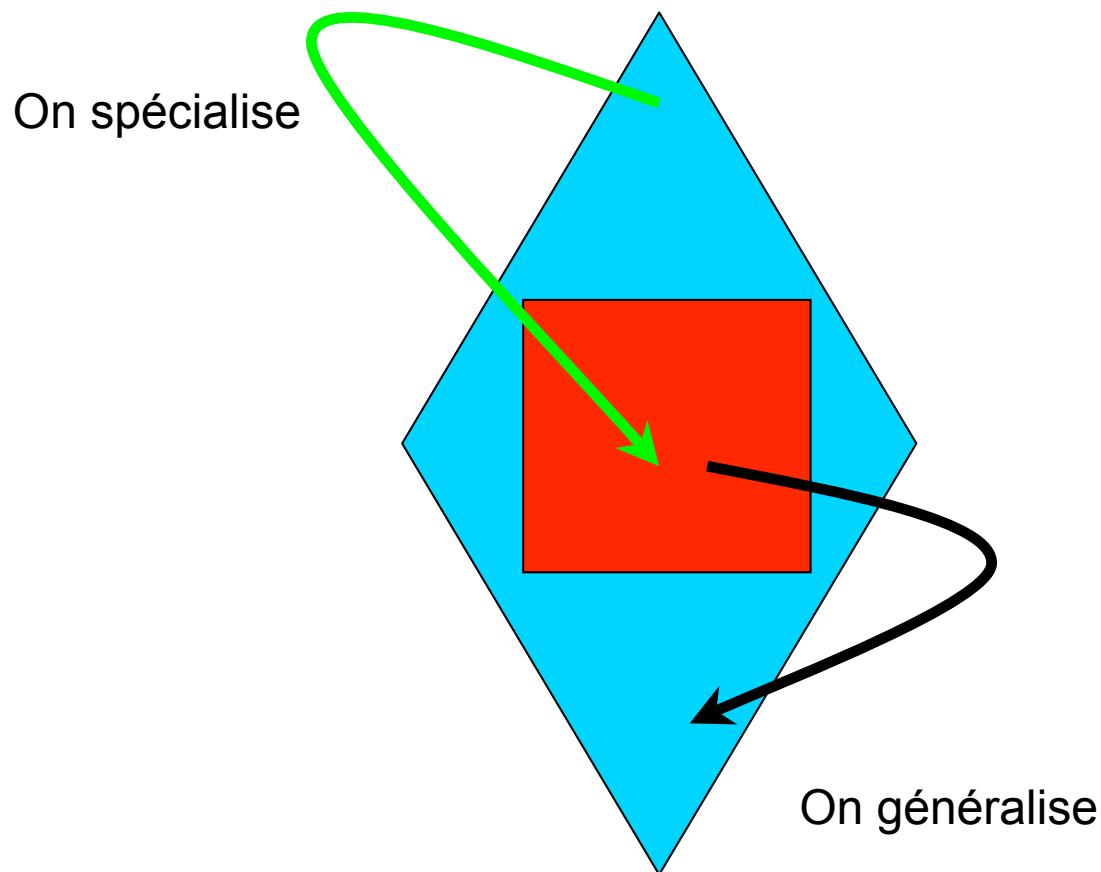


# Généralisation et spécialisation

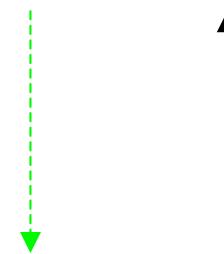
- ✿ Ces concepts permettent de modéliser l'emboîtement de classes les unes dans les autres
- ✿ Exemple :
  - Soient la classe C des carrés et la classe L des losanges
  - On dit que L généralise C et que C spécialise L
  - C est une sous-classe de L
  - L est une sur-classe de C



# Généralisation/spécialisation



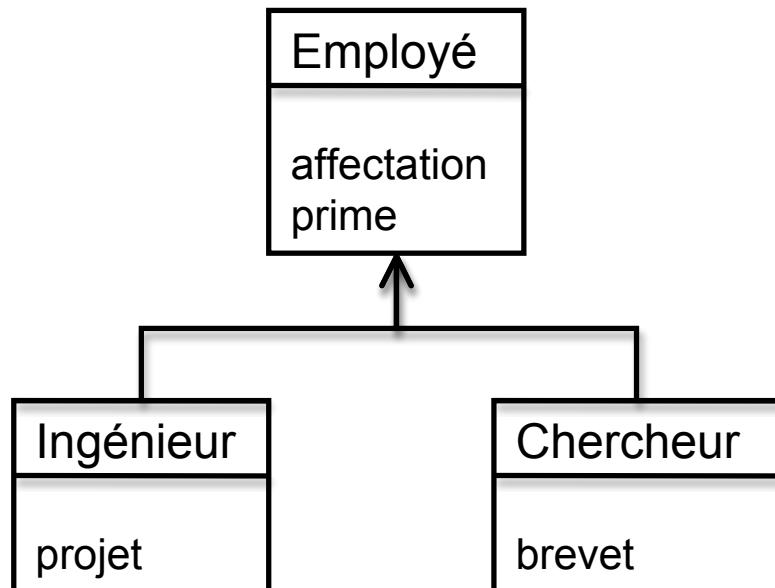
Losange = quadrilatère  
dont les 4 côtés sont  
de même longueur



Carré = losange qui a  
4 angles droits



# Héritage simple



L'héritage définit une relation d'ordre entre les classes :  
ingénieur ≤ employé  
chercheur ≤ employé

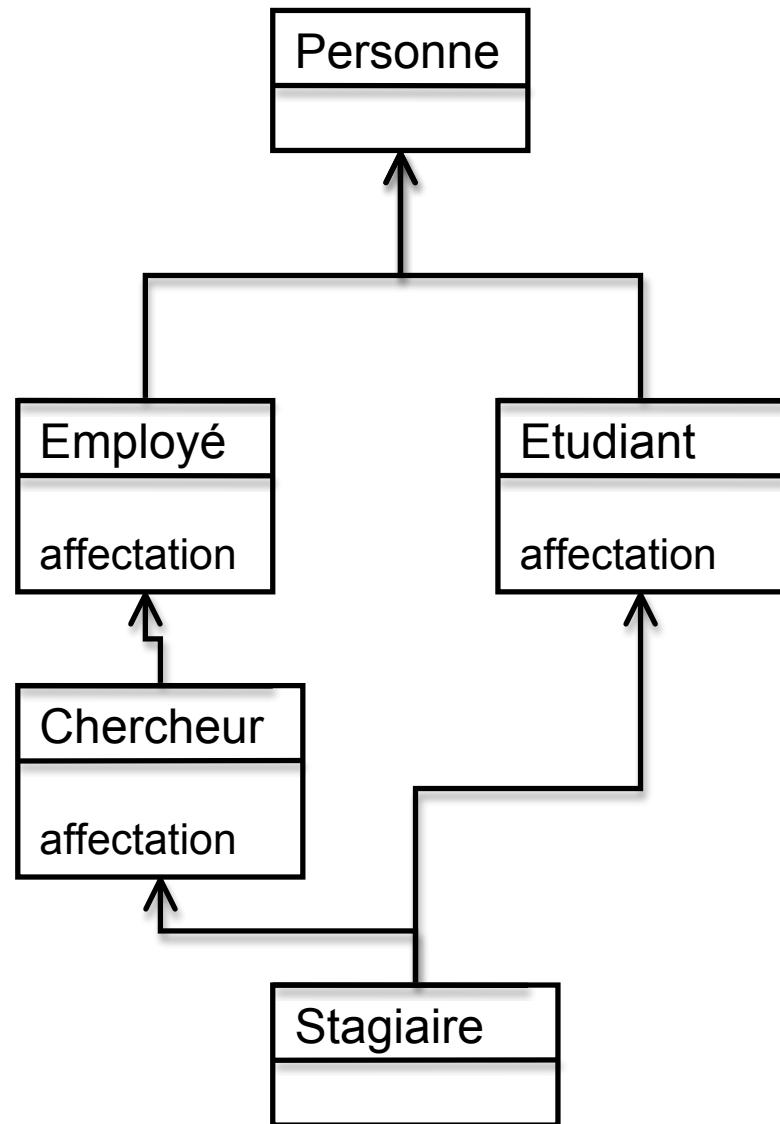
Exemple :

- <o1, employé, (« Martin », o10)>
- <o2, ingénieur, (« Dupont », o10, projetCAO)>
- <o3, chercheur, (« Simon », o10, [brevet1, brevet2])>

L'employé o1 est créé dans la classe Employé seulement pour être spécialisé ultérieurement



# Héritage multiple



L'héritage multiple permet à une classe de posséder plusieurs super-classes directes et d'hériter des propriétés de toutes ses super-classes.

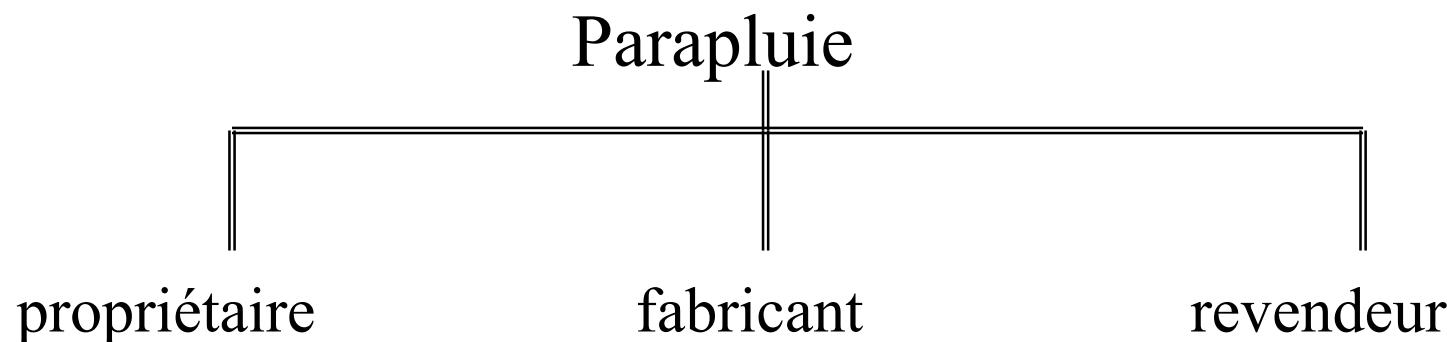
Il faut gérer les conflits :

- Interdiction des conflits de noms
- préfixer le nom de la propriété par la super-classe d'appartenance
- établir une priorité entre les super-classes pour n'obtenir qu'une seule propriété



## Les associations

- ✿ L'association est un mécanisme qui permet de regrouper des objets qui ne sont pas reliés par une relation d'agrégation ou par une relation de généralisation/spécialisation





# Le modèle UML

## (Unified Modeling Language)

- ✿ Normalisation des types de modélisation objet (OMT, Booch, OOSE)
- ✿ UML est un langage objet graphique
- ✿ UML n'est pas une méthode de conception
- ✿ UML est soumis à l'OMG (Object Management Group)



1997

Novembre : Acceptation  
Septembre : soumission finale  
Janvier : soumission OMG

Spécification  
sur internet

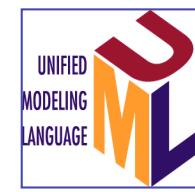
Autres  
Méthodes

OOAD  
Booch

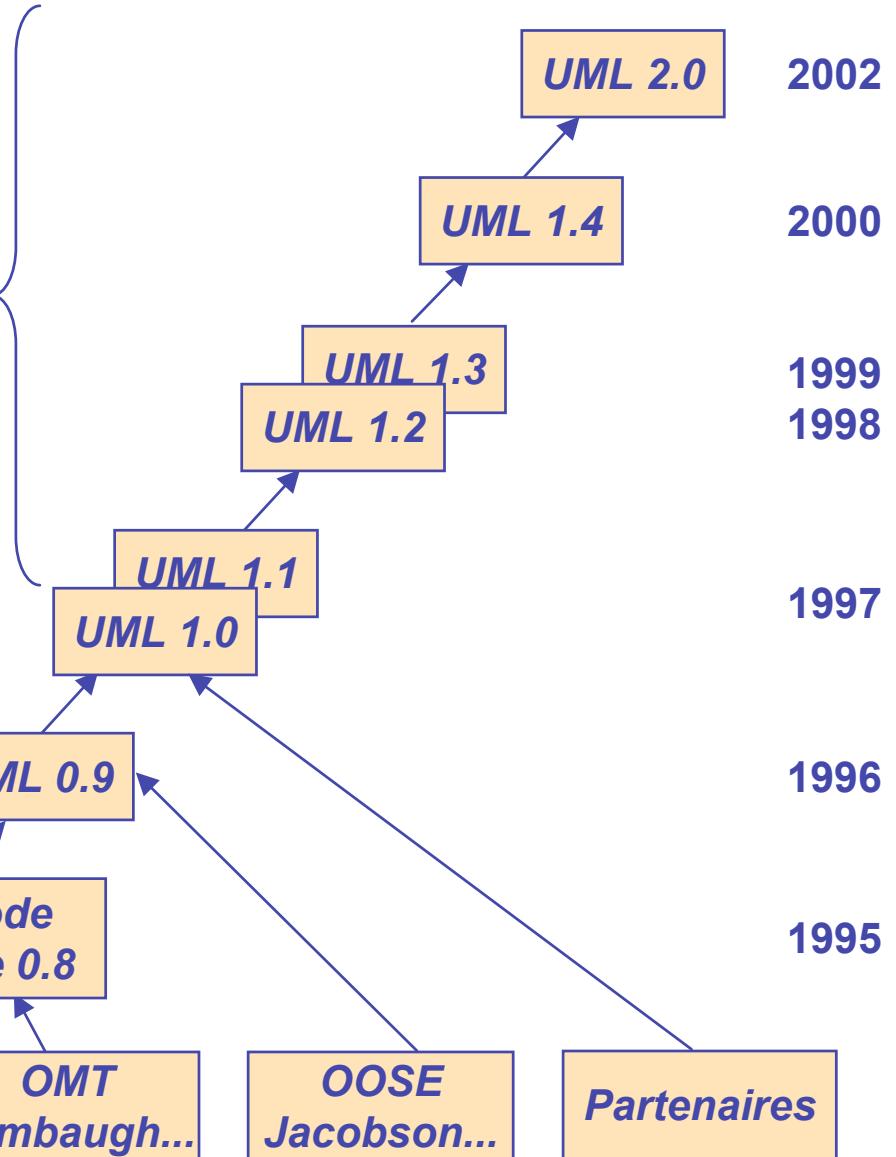
OMT  
Rumbaugh...

OOSE  
Jacobson...

Partenaires



# Genèse d'UML





# Un Modèle = Un point de vue sur le système

- ✿ Modèle de Classes qui capture la structure statique
- ✿ Modèle des Cas d'Utilisation – *Use Case, UC* – qui décrit les besoins, les fonctions
- ✿ Modèle d'Interaction qui représente les scénarios et les flots de messages
- ✿ Modèle des États qui exprime le comportement dynamique des objets, classes...
- ✿ Modèle de Réalisation qui montre des unités de travail
- ✿ Modèle de Déploiement qui précise la répartition des processus
- ✿ Descriptions abstraites pour capturer la sémantique d'un système



## Statique (ce que le système EST)

- diagramme de classes
- diagramme d'objets
- diagramme de composants
- diagramme de déploiement

## Dynamique

(comment le système  
EVOLUE)

- diagramme de séquence
- *diagramme de collaboration*
- diagramme d'états-transitions
- diagramme d'activités

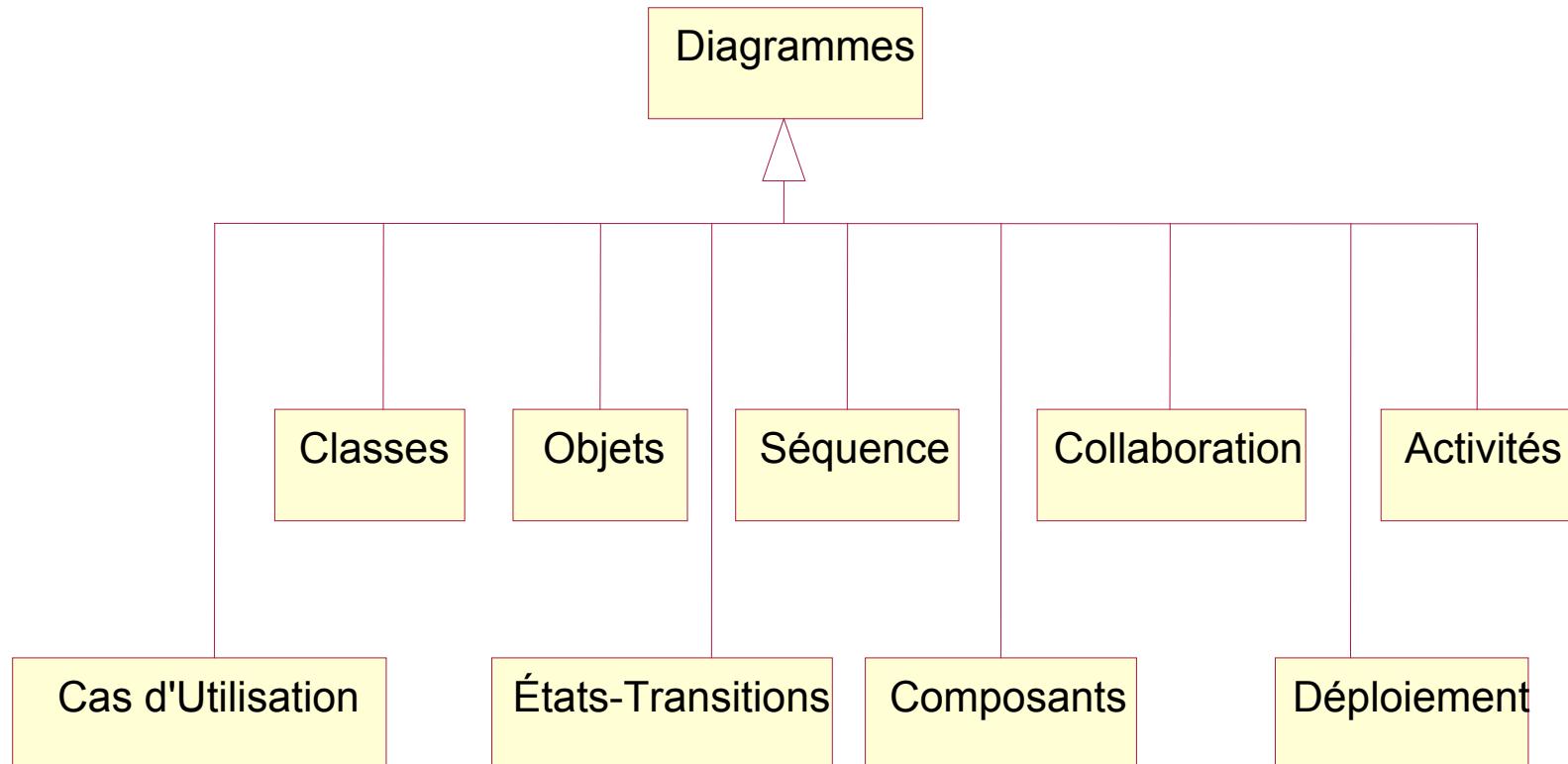
## Fonctionnel

(ce que le système FAIT)

- diagramme de cas d'utilisation
- *diagramme de collaboration*



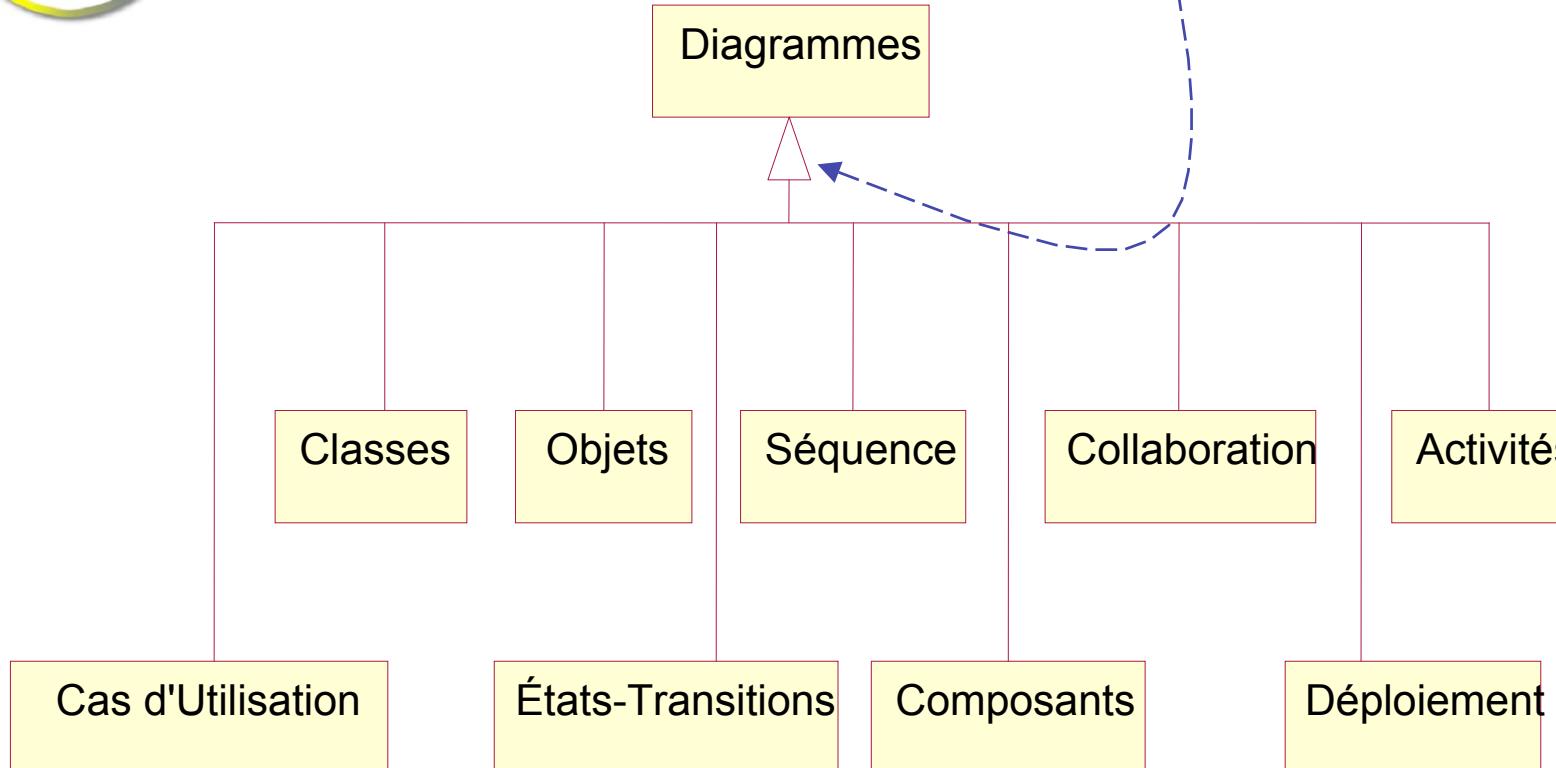
# Diagrammes



## Diagrammes

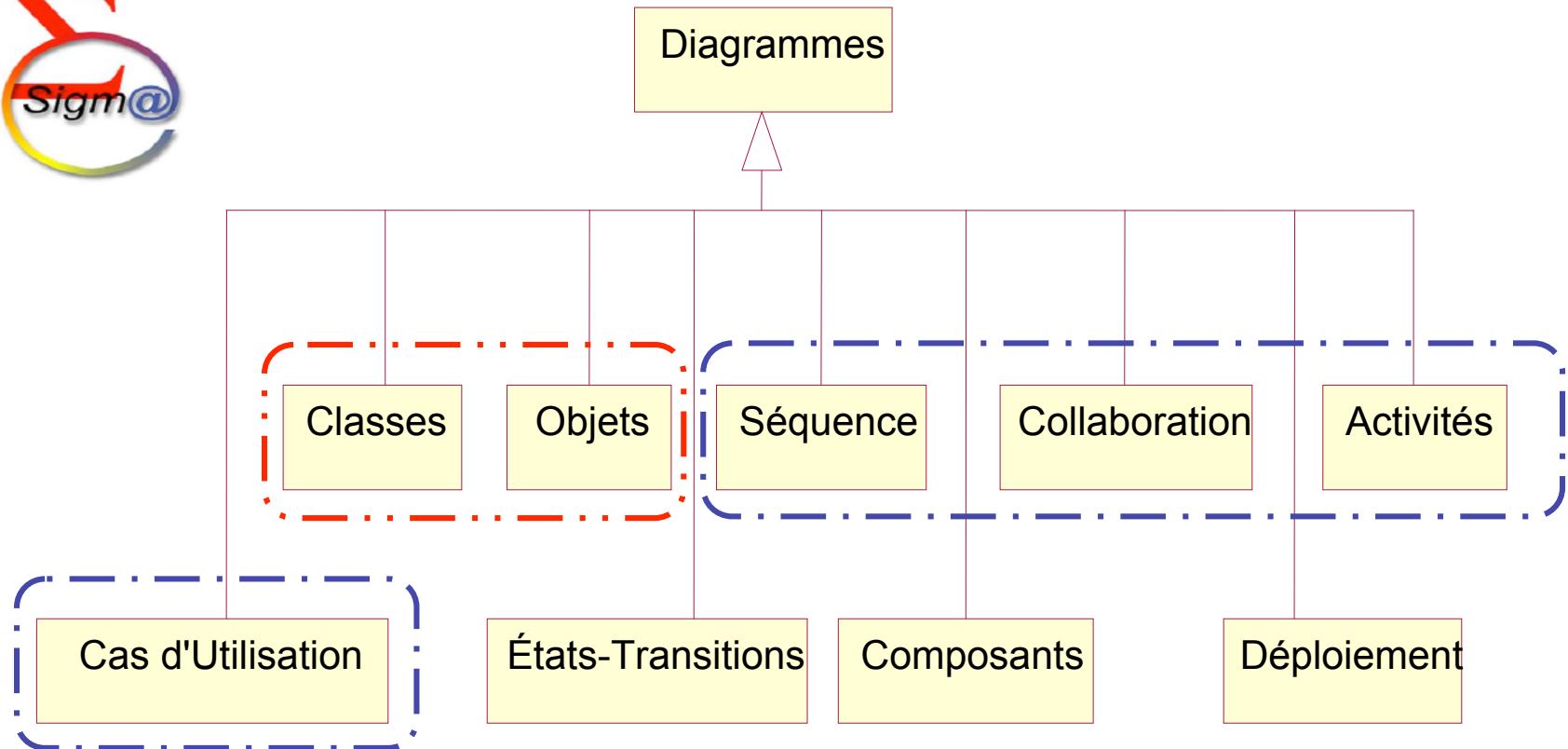


lien d'héritage : «est un diagramme»,  
chacun est une spécialisation de la classe diagramme



Un diagramme est un « langage graphique » de phrases traduisant entre les concepts « éléments du vocabulaire du diagramme », des relations matérialisées par des arcs du graphe, écrits avec une forme particulière.  
L'ensemble donne la syntaxe graphique du langage associé

## Diagrammes

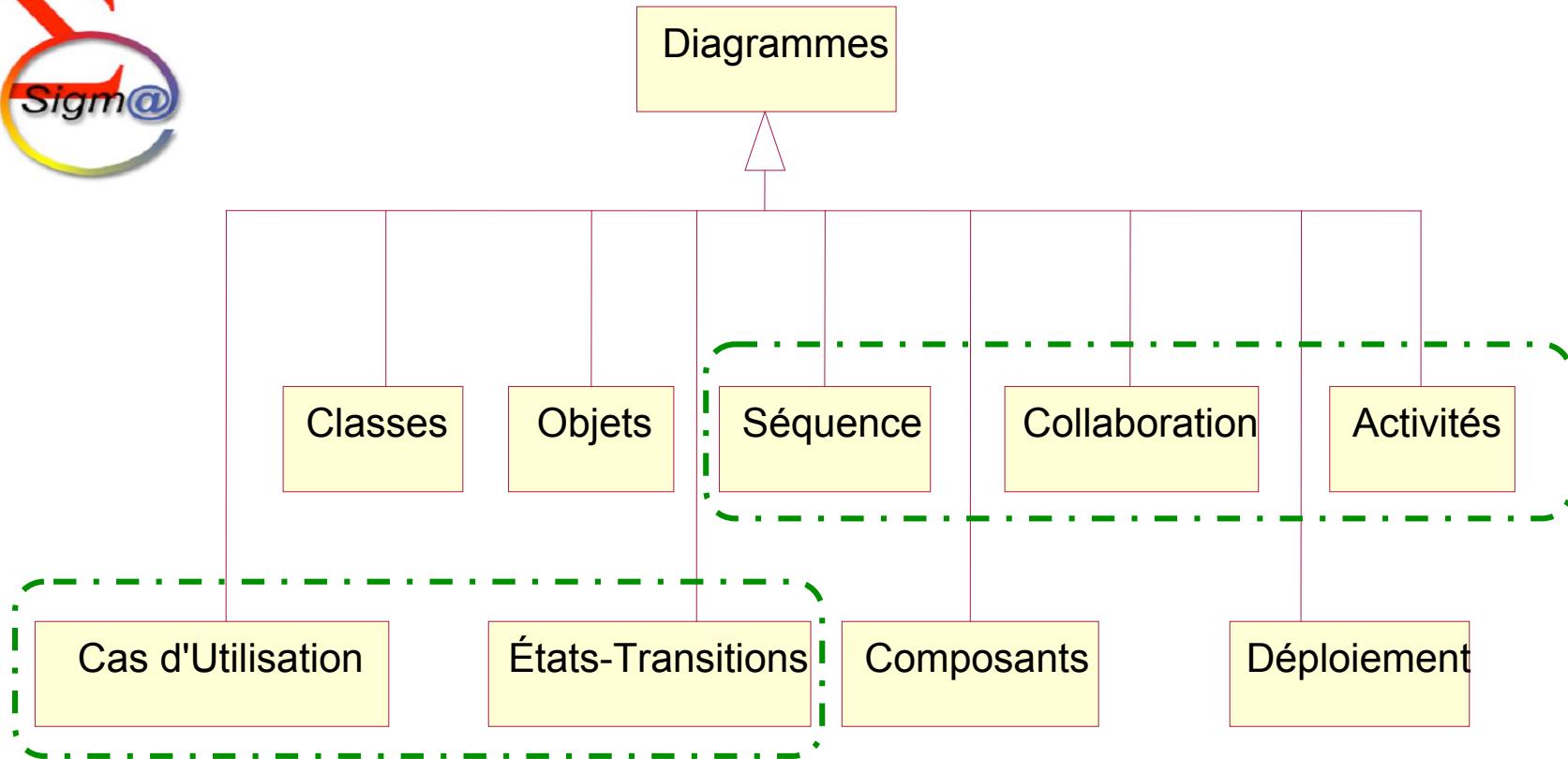


- ◆ Aspects **structurels** statiques —————
  - ◆ diagramme de **classes** : classes et relations statiques
  - ◆ diagramme des **objets** : objets et liens
- ◆ Aspects **fonctionnels et dynamiques** —————
  - ◆ diagramme de **cas d'utilisation** : acteurs et utilisation du système

Ce que le système  
**EST**

Ce que le système  
**FAIT**

## Diagrammes

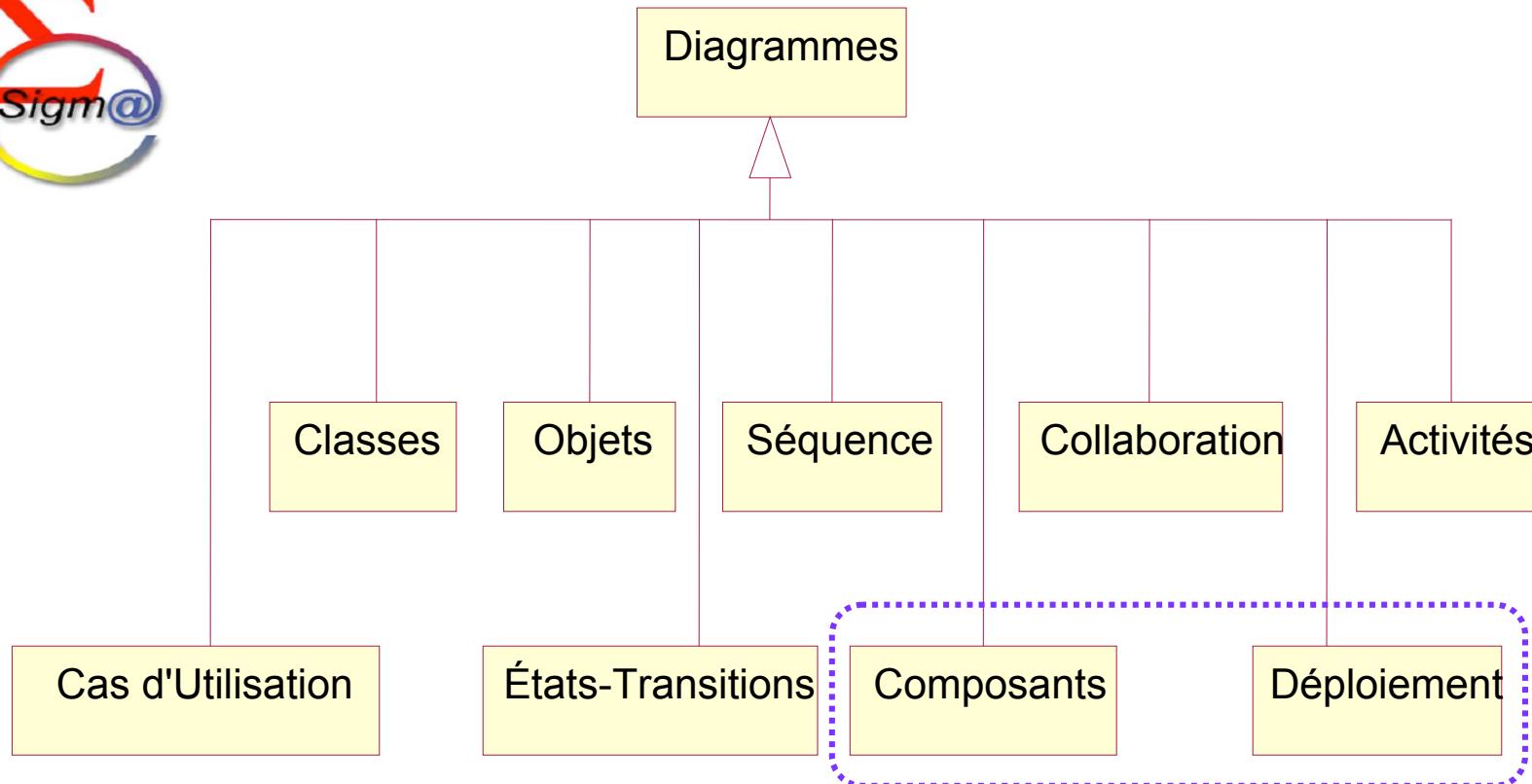


### ◆ Aspects dynamiques

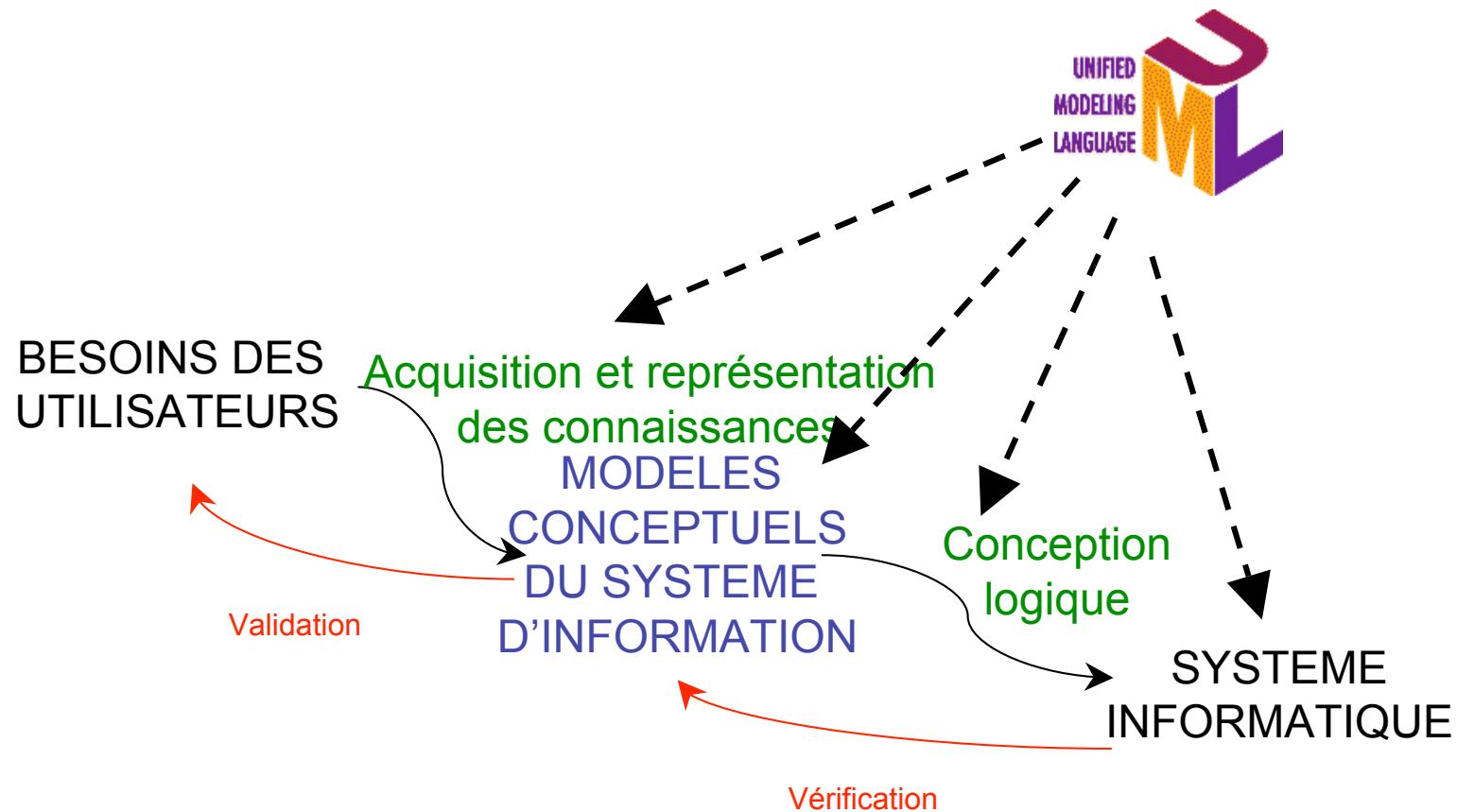
- ◆ diagramme de **séquence** : vision temporelle des interactions
- ◆ diagramme de **collaboration** : vision spatiale des interactions
- ◆ diagramme **d'états-transitions** : comportement des objets
- ◆ diagramme **d'activités** : flot de contrôle interne aux opérations

comment le système  
EVOLUE

## Diagrammes



- ◆ Aspects implantation
  - ◆ diagramme de composants : codage
  - ◆ diagramme de déploiement : implantation, distribution





Une classe est un ensemble d'objets ayant mêmes attributs, mêmes opérations, mêmes relations, et même sémantique.

### Classe avec ses attributs et ses opérations

Personne
- nom : Chaîne
- annéeNaiss : Entier
- téléphone [0..2] : Chaîne
- nbreEnfants : Entier
- marié : Booléen = vrai
- <u>moyenneNbreEnfants</u> : Réel
+ age?(annéeCour : Entier) : Entier
+ ajouterEnfant()
+ nomPers?() : Chaîne
+ <u>moyenneNbreEnfants?</u> () : Réel

Notation visibilité

+ public  
# protégé  
- privé

nom de la classe (*commence par une majuscule*)  
attribut

visibilité

nom (*commence par une minuscule*)

multiplicité (*1 par défaut*)

type de la valeur

valeur par défaut ou valeur initiale

attribut de classe (*souligné*)

opération

visibilité

nom (*commence par une minuscule*)

paramètres typés

type du résultat

opération de classe (*souligné*)

La valeur par défaut est affectée à l'attribut à la création des instances de la classe à moins qu'une autre valeur ne soit spécifiée.  
La multiplicité indique le nombre de valeurs possibles pour l'attribut



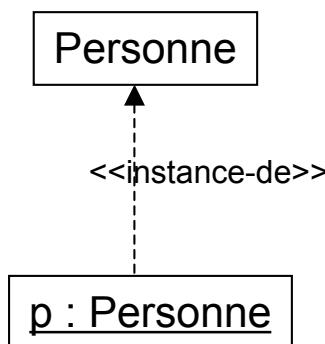
## CLASSES

Un objet est une instance (une occurrence) d'une classe.

Une classe est un modèle caractérisé par des propriétés(attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.

Un système informatique en exploitation est composé d'un ensemble d'objets de différentes classes qui collaborent.

instanciation



objet avec valeurs

<u>p</u> : Personne
nom = Durand
annéeNaiss = 1978
téléphone = (0476..., 0635....)
nbreEnfants = 1
marié = faux

nom de l'objet et  
nom de la classe (*soulignés*)

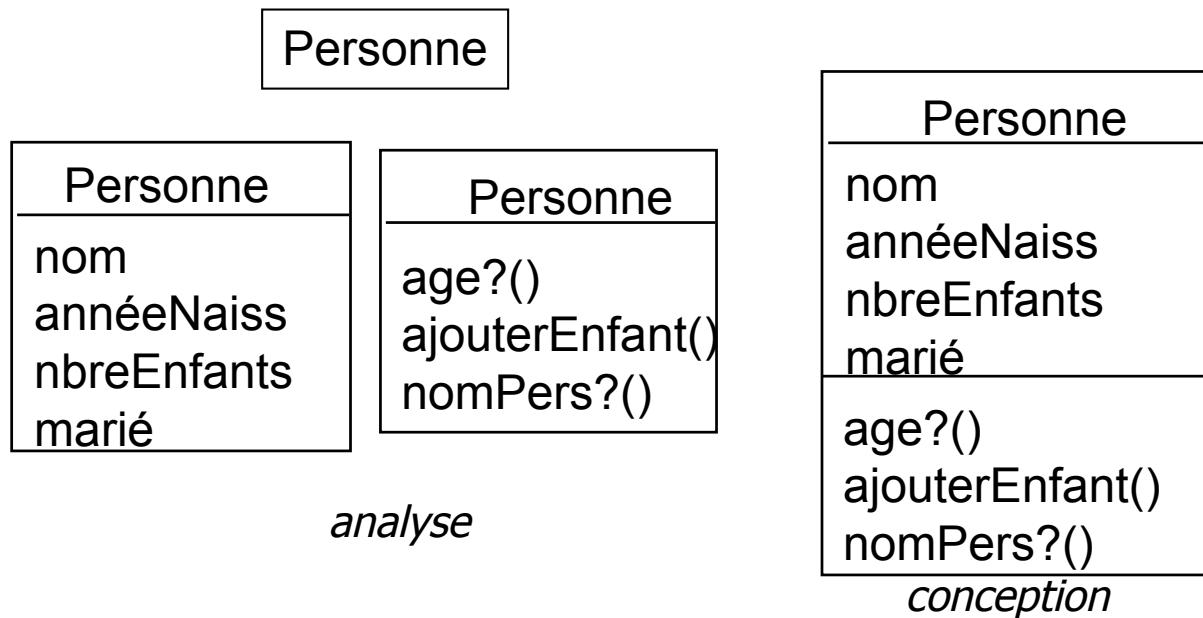
noms des attributs et valeurs

Note : *la valeur d'un attribut de classe est associée à la classe et non pas à chaque objet*

Le niveau de détail du diagramme est adapté au niveau d'abstraction correspondant à une phase du cycle de vie du logiciel et d'autre part au type de communication.

## Classe

Objet



p : Personne

: Personne

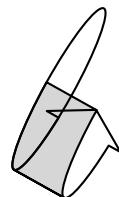
objet anonyme



## Attribut dérivé

Un attribut dérivé est un attribut dont la valeur est calculée à partir de celles d'autres attributs.

Personne
- nom : Chaîne
- annéeNaiss : Entier
- nbreEnfants : Entier
/ age : Entier

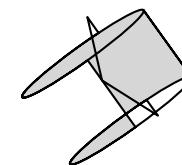


analyse

Expression du besoin d'accéder à l'âge à l'aide d'un attribut dérivé

Personne
- nom : Chaîne
- annéeNaiss : Entier
- nbreEnfants : Entier
+ age (annéeCour) : Entier

conception



réalisation

```
age?(annéeCours : Entier) : Entier
retourner (annéeCours-annéeNaiss );
```



Classe = attributs + opérations +

responsabilités

Vol
numvol hdep har
Définir-périodes()
<i>définit la desserte d'une ligne (ville de départ, ville d'arrivée) selon un horaire donné</i>

## CLASSES

Avion
no-immat : integer type : string rayon : integer hrvol : hrcumul = 000 ... ajouter-avion(no-immat, type, rayon) enreg-vol (durée : hrcumul) : hrcumul affecter-instance(numvol, jr, sem) mettre-révision : date
... <i>assure le transport du fret et des passagers selon les affectations correspondant à la politique commerciale</i>

Les **attributs** sont typés ou non, simples ou multiples, avec valeur initiale éventuelle.

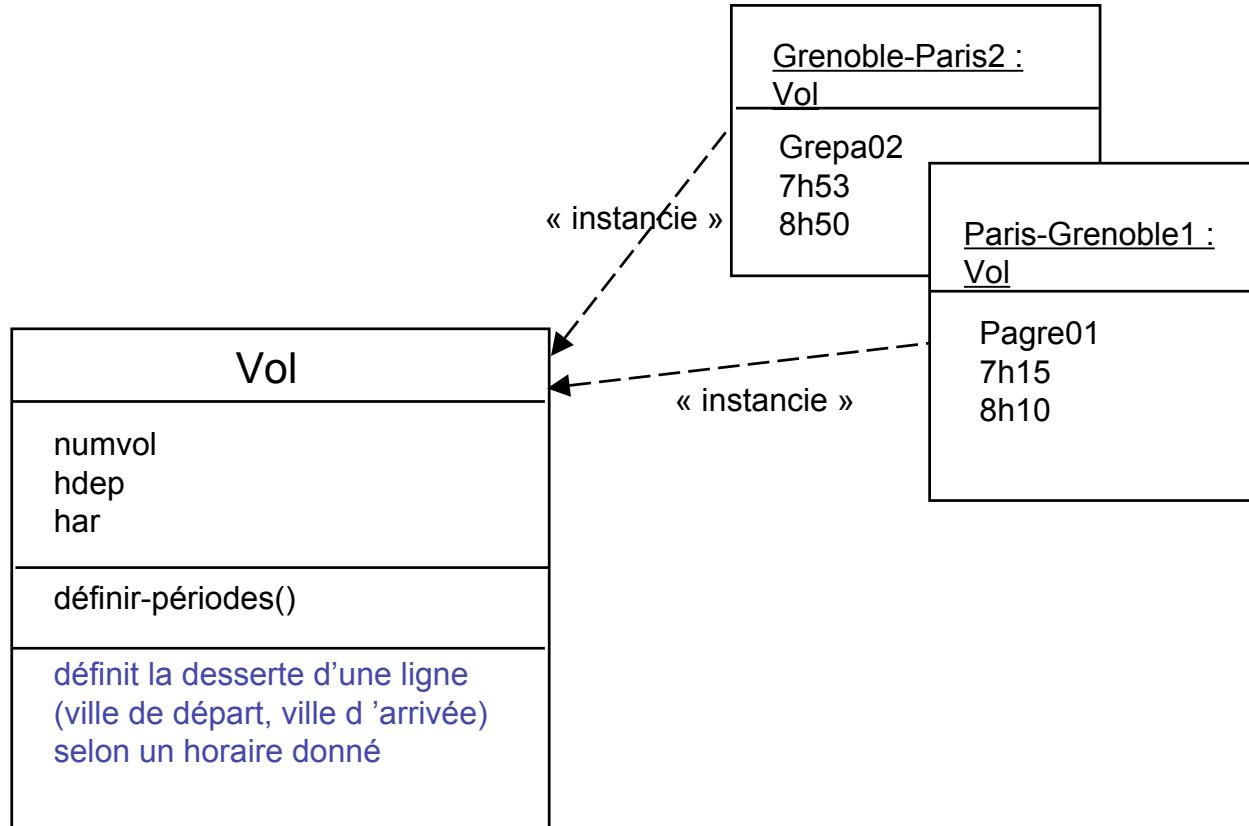
Les **opérations** sont avec ou sans leurs paramètres.

Les **responsabilités** sont très utiles en phase d'analyse de besoins pour décrire le rôle des objets de la classe par rapport à l'environnement, pour se concentrer sur le « pourquoi » et pas uniquement sur les structures (attributs) ou les comportements (opérations).

# Classe & objets

## CLASSES

Une classe correspond à un type (attributs, opérations, propriétés, ...) et c'est un conteneur d'objets conformes à ce type.





## Une syntaxe de base et des propriétés

[visibilité] [/] nom [multiplicité] [ : type ][= valeur initiale] [ {propriétés et contraintes} ]

4 valeurs de visibilité :

- + : public, visible de toute classe
- # : protégé, visible dans la classe et ses *sous-classes*
- : privé, visible dans la classe uniquement
- ~ : visible dans le paquetage uniquement

des propriétés :

- ordered, ordonné pour un attribut à valeurs multiples
- unique, unicité pour un attribut à valeurs multiples

...



...

-soldeCourant : float = 0

-/estADecouvert : boolean {estADecouvert == true si soldeCourant < 0 }

...

- largeur : int {largeur > 0}

-...

-villes\_Voyage [1..\*] : Ville

-villes\_Itinéraire [1..\*] : Ville {not unique, ordered}

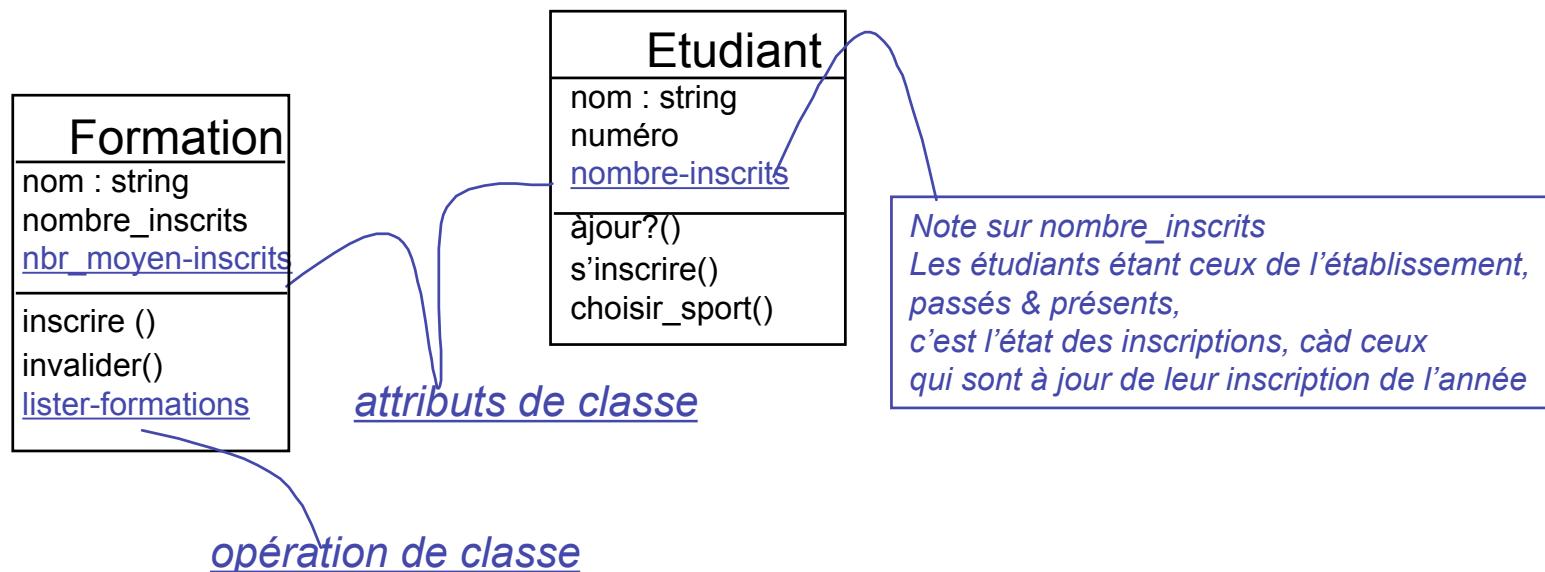
...



## Attributs et Opérations de classes

Les attributs et les opérations concernent tous les objets de la classe et s'appliquent sur chaque objet de la classe.

On peut définir des attributs et des opérations qui s'appliquent sur la classe elle-même, c'est-à-dire sur la collection des objets de la classe ou un sous-ensemble de ceux-ci.

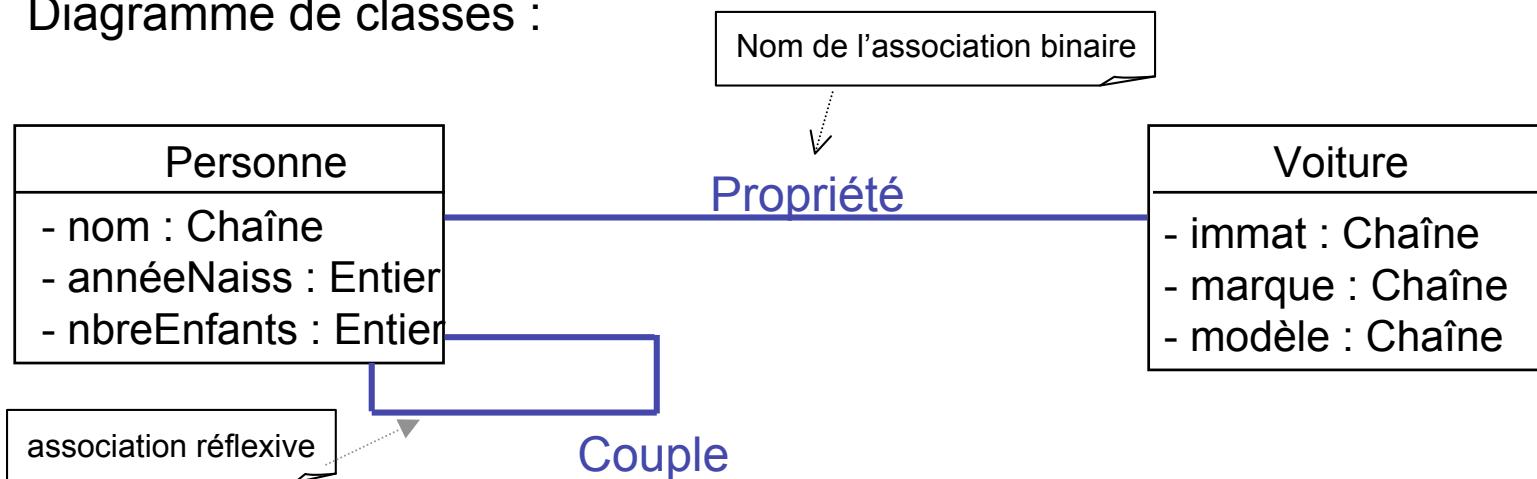


**Le concept d'association ne fait pas partie des concepts élémentaires du paradigme objet.**

**Une association entre deux classes est une relation ou ensemble de liens entre des objets de ces classes.**

**Une association binaire peut être vue comme un sous-ensemble d'un produit cartésien entre les ensembles des objets des classes associées.**

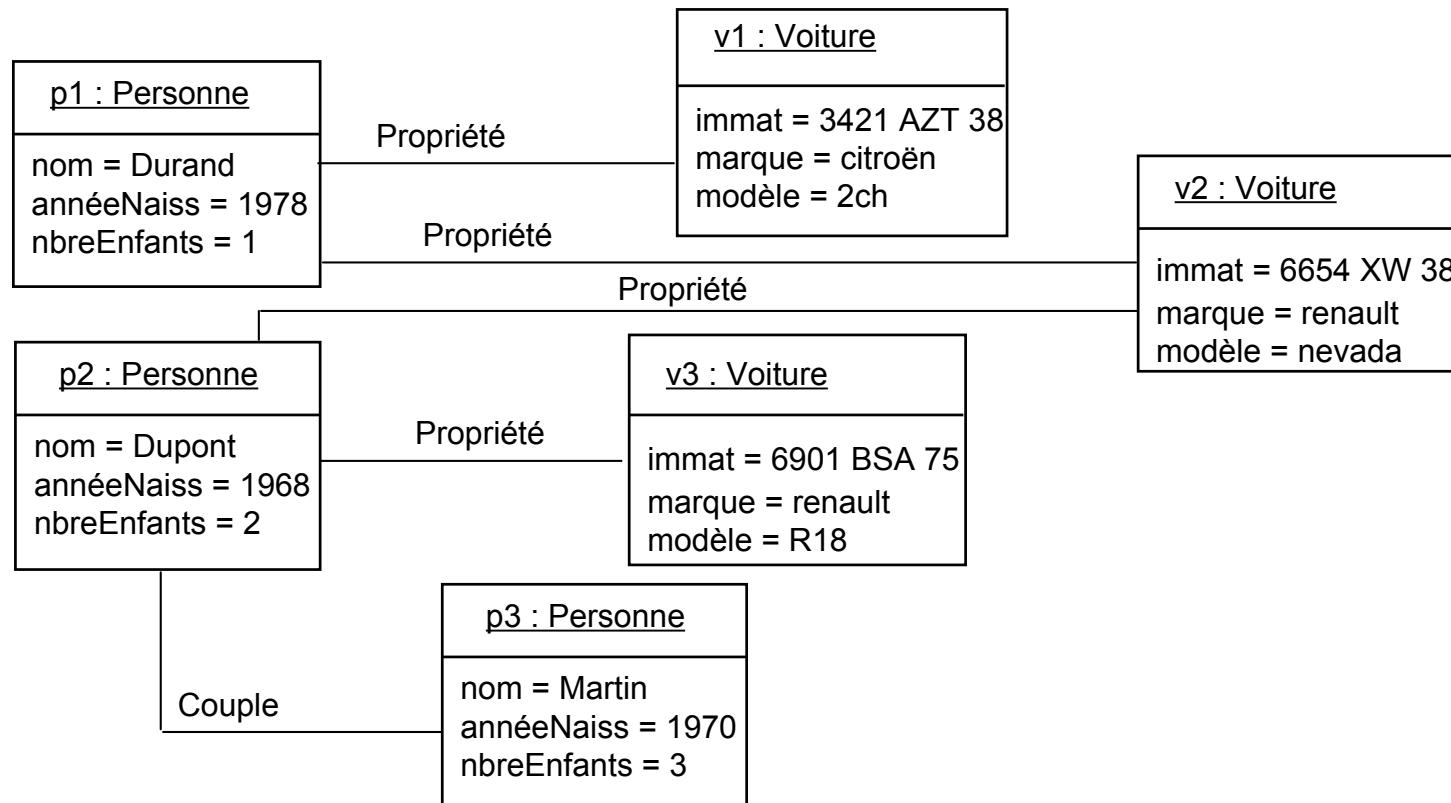
Diagramme de classes :



**Un lien indique une connexion entre des objets.**

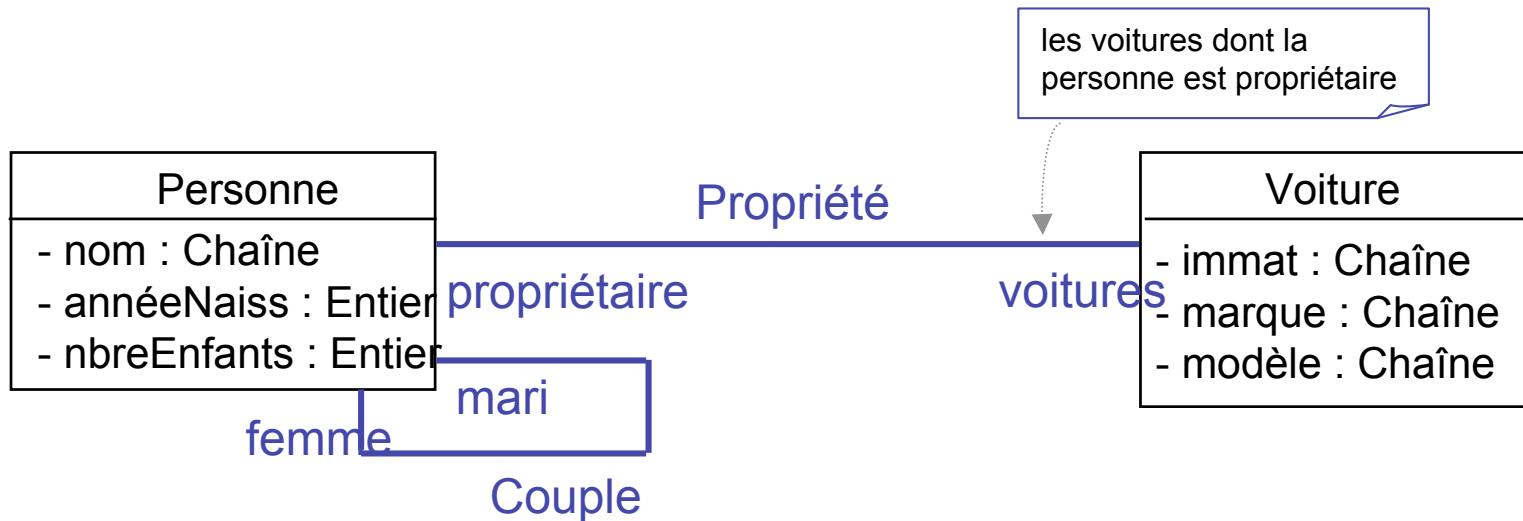
**Un lien est une instance d'association.**

Diagramme d'objets :



## Rôles d'une association

- Une association possède 2 rôles inverses l'un de l'autre.
- Un rôle indique comment une classe source voit la classe destination.
- Le nom du rôle est écrit du côté de la classe qui joue ce rôle.

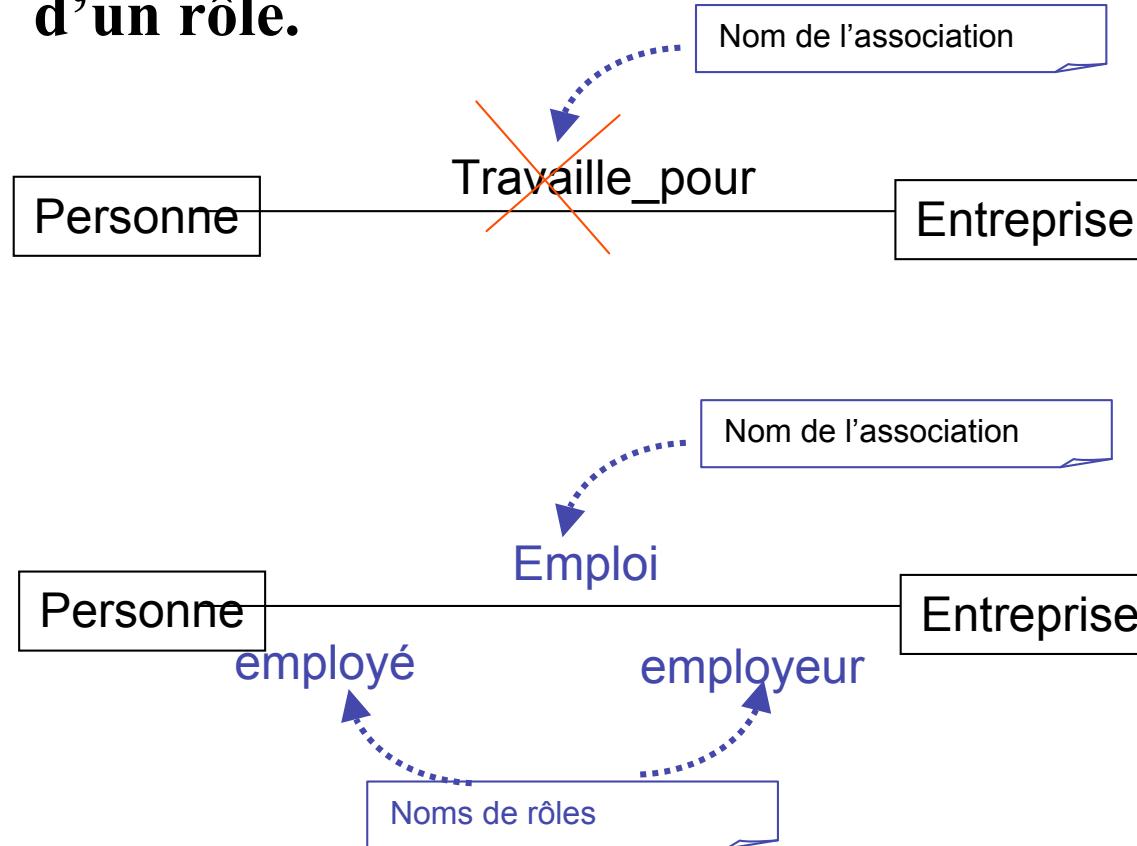


Le nom de l'association ne doit pas être confondu avec un nom de rôle.

Il est souvent plus facile de nommer les rôles d'une association plutôt que l'association elle-même. Dans ce cas, le nom de l'association peut être omis.



**Il ne faut pas confondre le nom de l'association avec le nom d'un rôle.**



⚠ Il est recommandé de donner des noms d'associations qui ne nécessitent pas d'indiquer de sens de lecture, c'est-à-dire des noms d'associations qui sont des noms des « couples » liés, des noms de **produits cartésiens**.

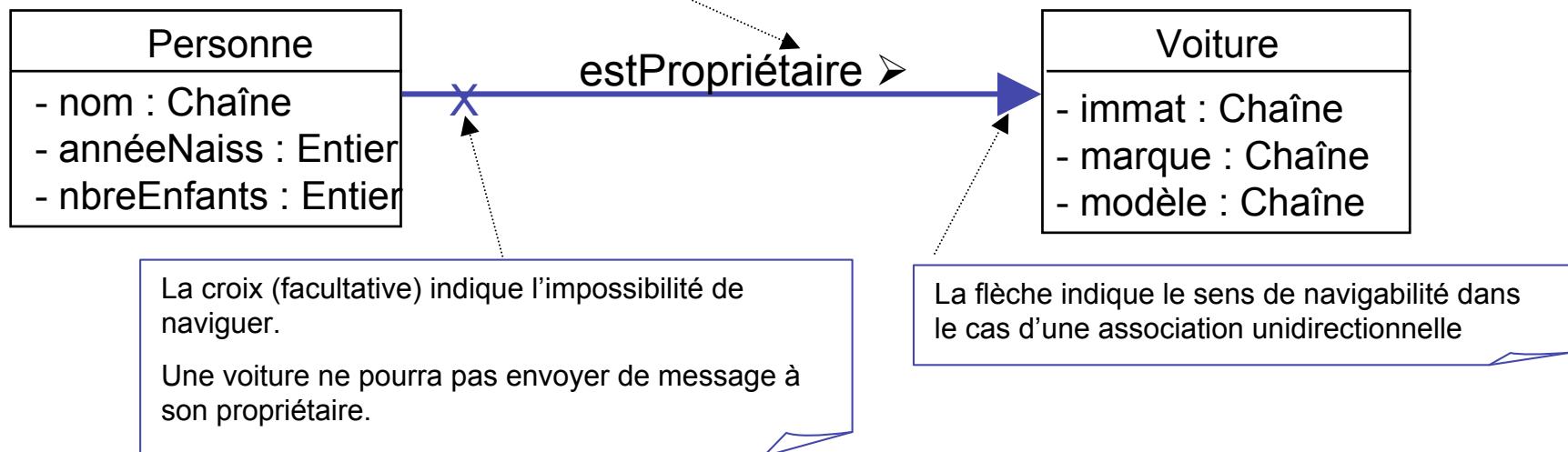
## Sens de lecture et navigation

Le **sens de lecture** est une « décoration » qui indique comment interpréter le nom de l'association.

L'indication de **navigabilité** d'un rôle exprime l'**obligation** pour un objet source d'identifier le ou les objets cibles.

Une association est **bidirectionnelle** quand il y a **navigation** dans les 2 sens (par défaut).

Une association **unidirectionnelle** n'est **navigable** que dans un seul sens.

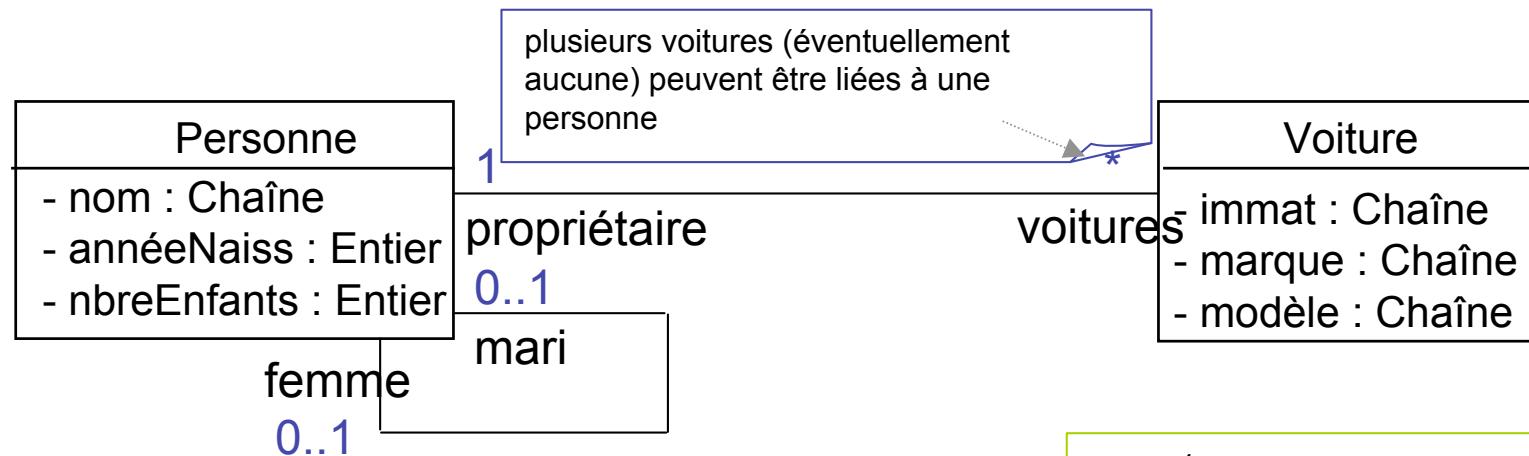


Il est recommandé de « différer » l'introduction de sens de lecture et de navigabilités à la conception.

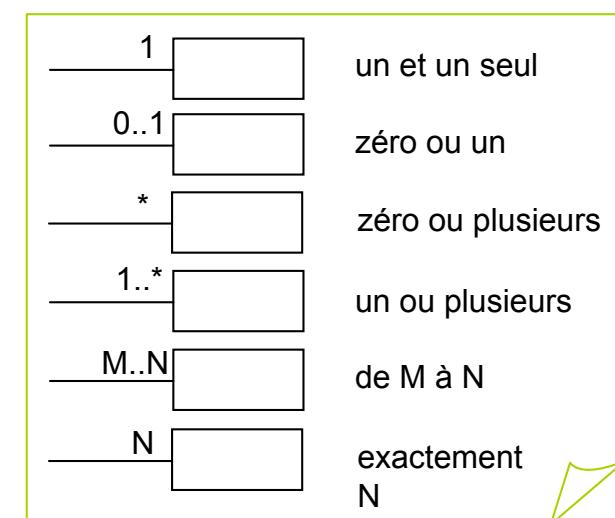
## Multiplicité d'un rôle

La multiplicité est portée par le rôle. Elle précise combien d'objets de la classe destination peuvent être liés à un objet de la classe source. On parle aussi de **cardinalité** du rôle.

La multiplicité est notée du côté de la classe destination.



### Notations de la multiplicité



## Traduction des rôles

**Sous**

de la **réalisation**, un rôle navigable est traduit par un attribut.



Classe Personne  
Attributs  
nom : Chaine;  
...  
voitures : Ensemble [Voiture];  
Méthodes  
ajouterVoiture (v:Voiture);  
voitures.inserer(v);  
v.affecterProprio (self);  
fin ajouterVoiture;  
fin Personne;

Classe Voiture  
Attributs  
marque : Chaine;  
propriétaire : Personne;  
Méthodes  
affecterProprio (p:Personne);  
Propriétaire := p;  
fin affecterProprio;  
fin Voiture;



Classe Personne  
Attributs  
nom : Chaine;  
...  
voitures : Ensemble [Voiture];  
Méthodes  
ajouterVoiture (v:Voiture);  
voitures.inserer(v);  
fin ajouterVoiture;  
fin Personne;

Classe Voiture  
Attributs  
marque : Chaine;  
...  
Méthodes  
Le rôle inverse n'est pas traduit.  
fin Voiture;

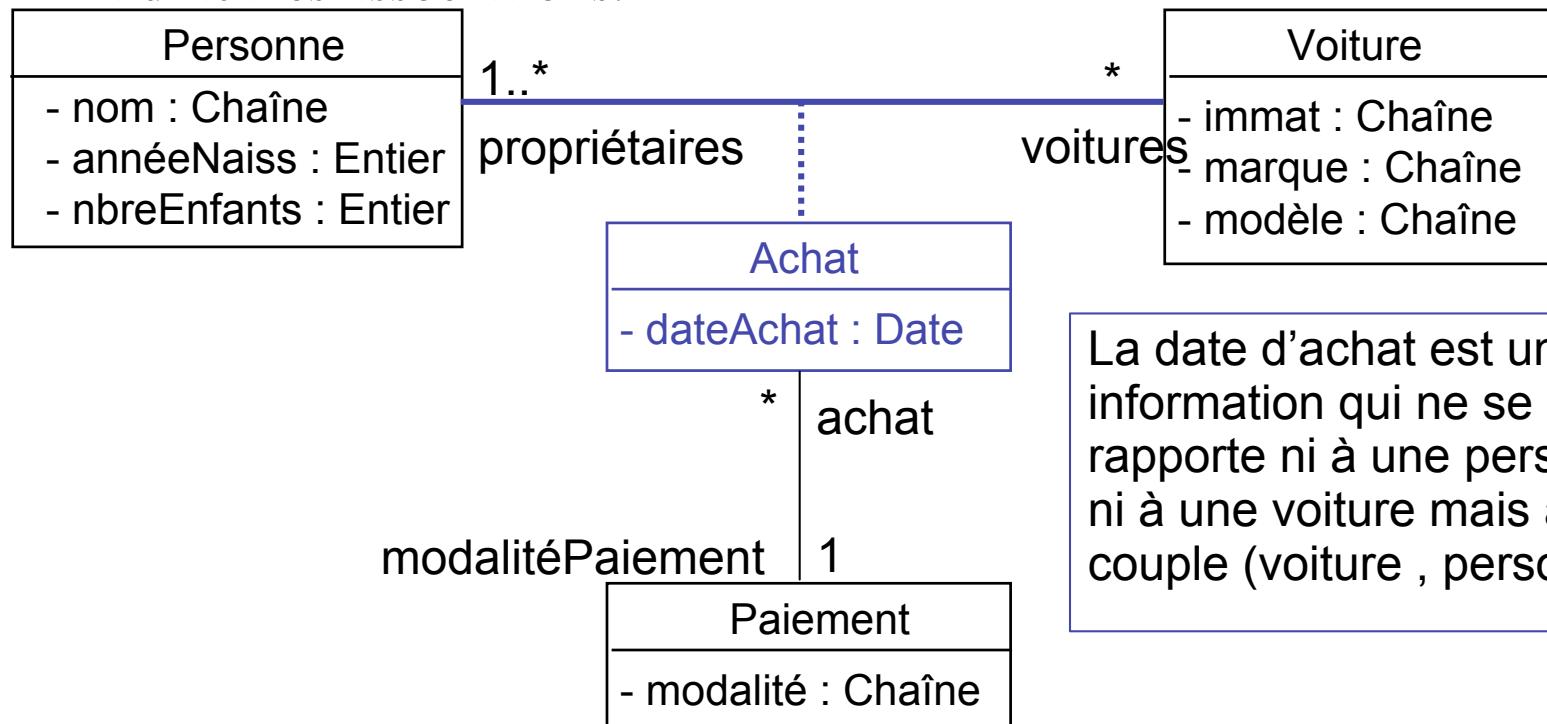


## Classe-association ou classe associative

**Une association peut posséder des attributs et des opérations.**

**Elle est alors représentée à l'aide d'une classe-association.**

**Une classe-association est une classe à part entière et peut donc participer à d'autres associations.**

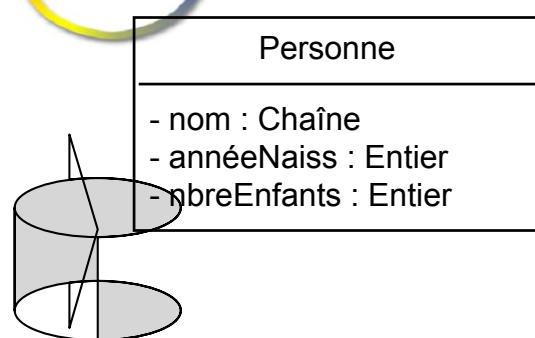


La date d'achat est une information qui ne se rapporte ni à une personne, ni à une voiture mais à un couple (voiture , personne)



## Transformation d'une classe-association

CLASSES

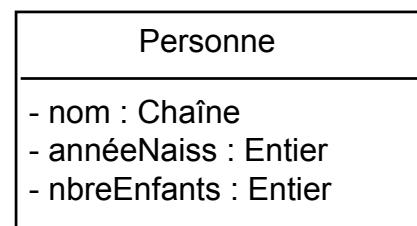
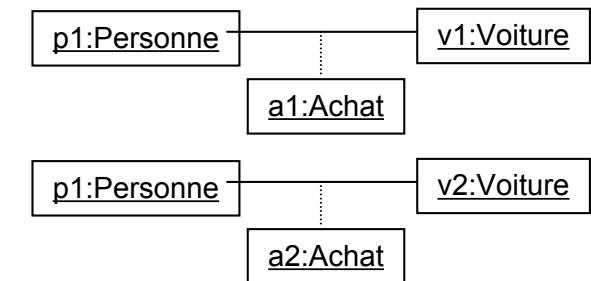
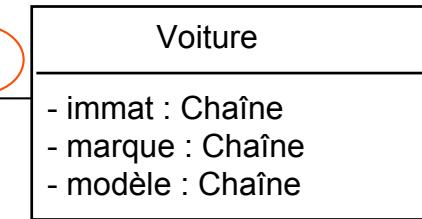


Une personne peut effectuer plusieurs achats, mais pas de

la même voiture.

propriétaires

voitures



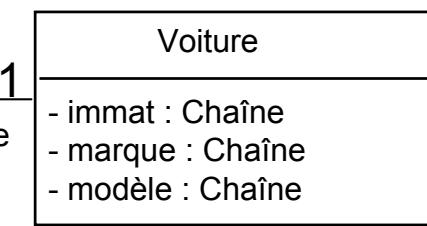
1

\*

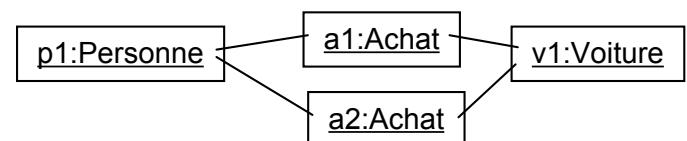
propriétaire

1..\*

voiture



Une personne peut effectuer plusieurs achats de la même voiture.

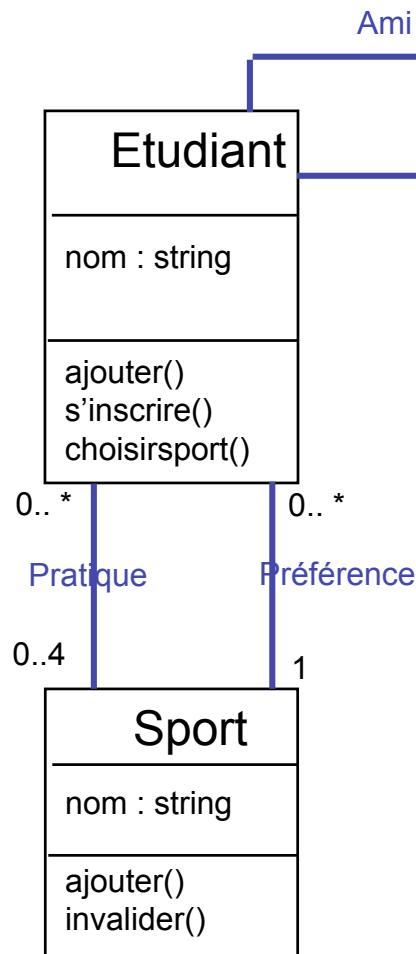




## Associations multiples

Plusieurs associations peuvent être définies entre deux classes.

Une association peut être définie sur la même classe.



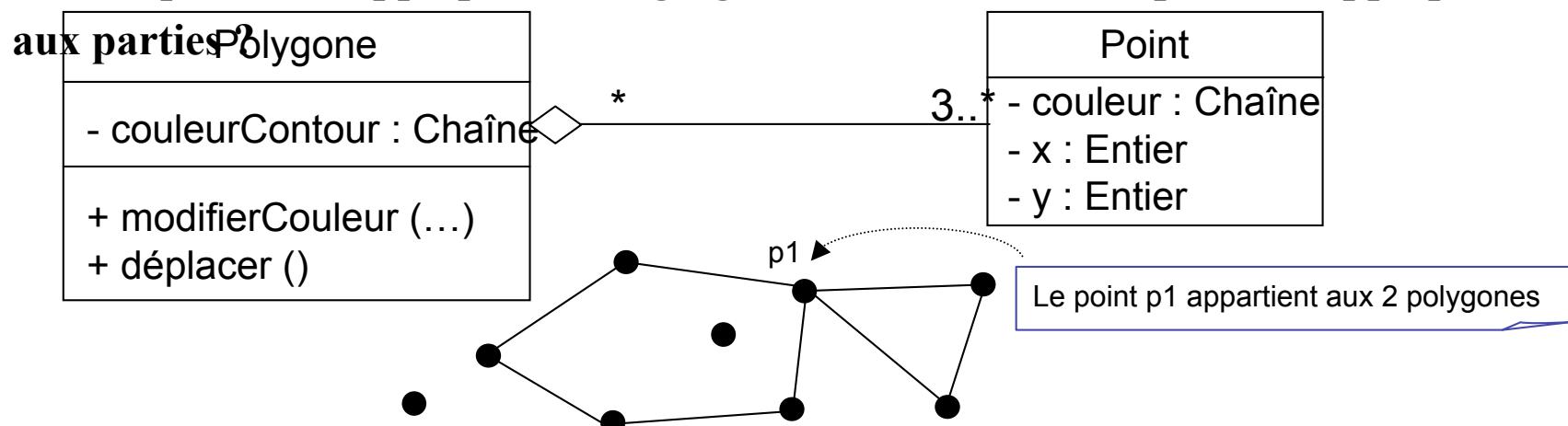
**Forme particulière d'association non symétrique dans laquelle l'une des extrémités joue un rôle prédominant par rapport à l'autre.**

**Les 2 objets liés par une agrégation sont distingués : l'un fait "partie de" l'autre, considéré comme l'objet global (l'agrégat).**

**Un objet "partie" peut être partagé (pas d'exclusivité).**

**Reconnaître une agrégation :**

- Des valeurs d'attributs sont-elles propagées de l'agrégat vers les parties ?
- Des opérations appliquées à l'agrégat sont-elles automatiquement appliquées aux parties



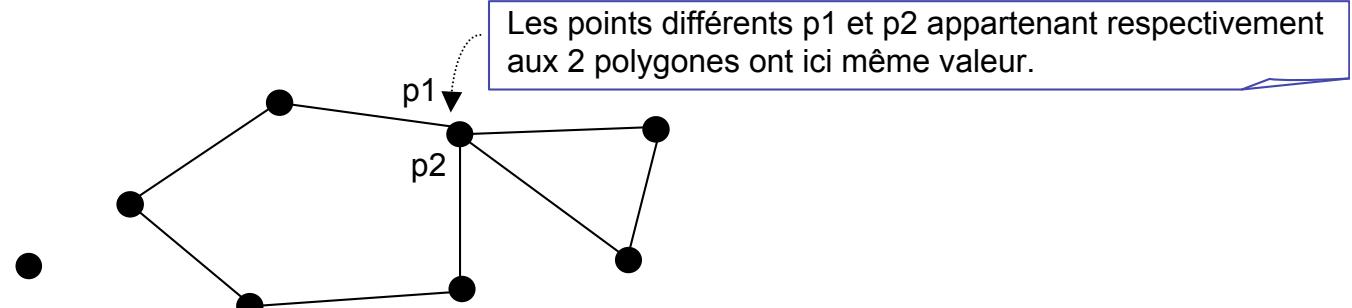
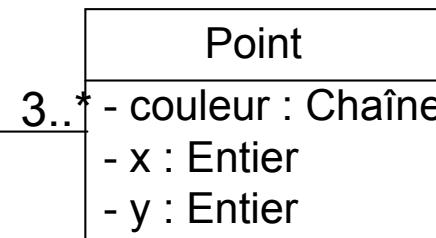
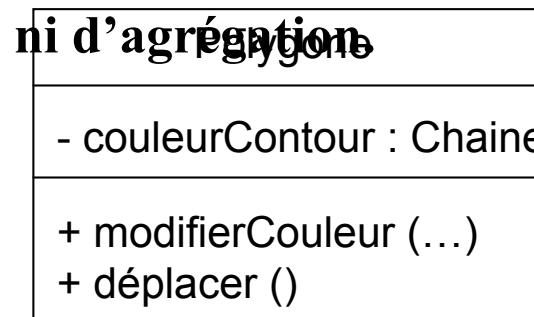
**Forme particulière d'agrégation avec une dépendance forte entre composé et composant.**

**Un objet composant n'appartient qu'à un seul composé (pas de partage).**

**Le composant est détruit lors de la destruction du composé.**

**Le composant peut éventuellement être créé hors de son composé.**

**Le composant n'intervient pas dans d'autres associations de composition ni d'agrégation**

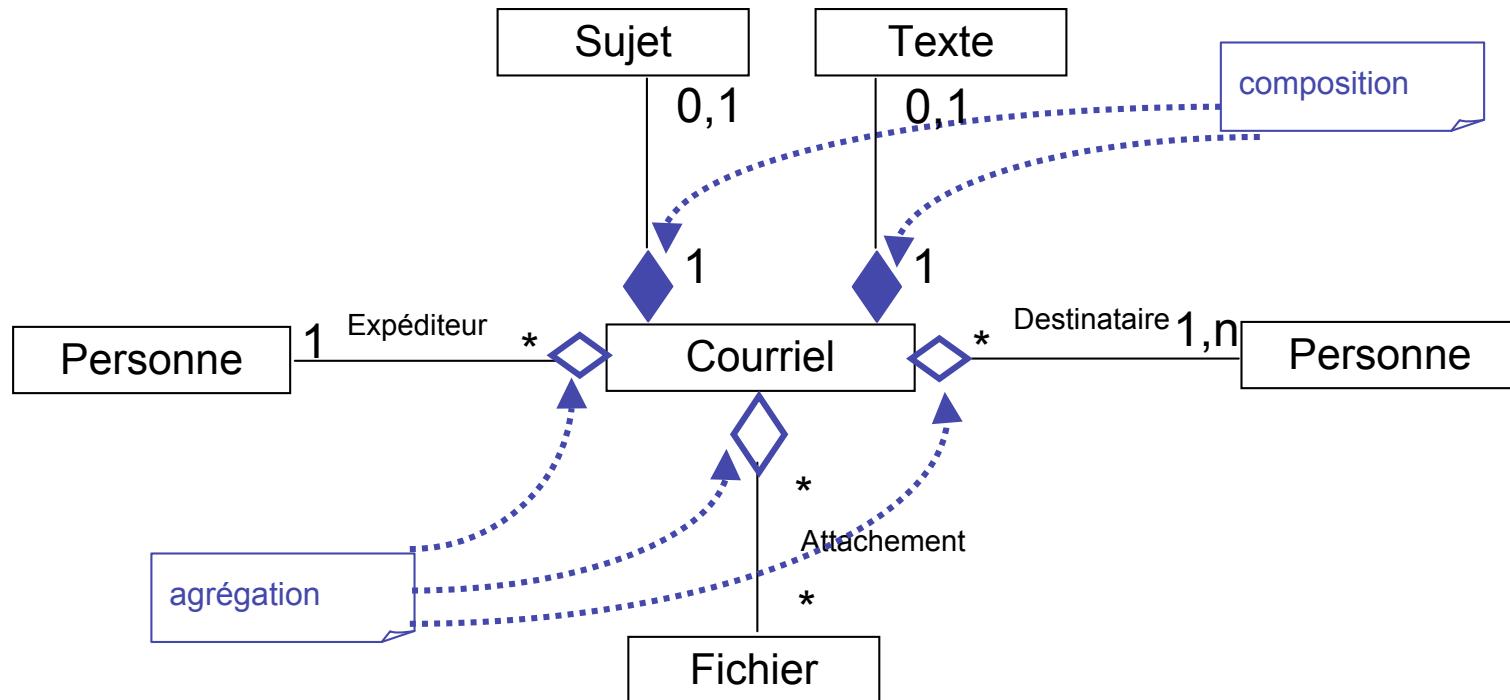




## Agrégation & Composition

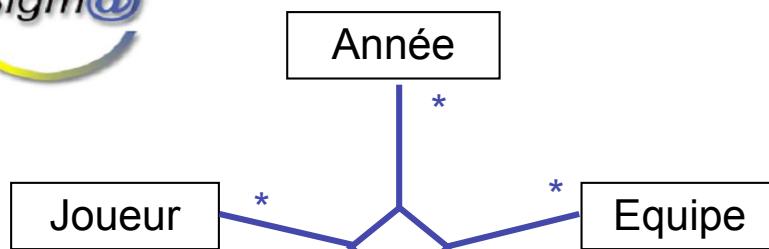
CLASSES

2 associations « tout-partie » à sémantique différente.





## Association ternaire



JOUEUR	EQUIPE	ANNEE	nbreVictoires	nbreDéfaites
dupont	louvetaux	96	6	2
durand	louvetaux	96	3	4
dupont	totoche	96	4	5
dupont	totoché	97	5	3

### Multiplicités :

Un joueur, une année donnée, peut jouer dans plusieurs équipes

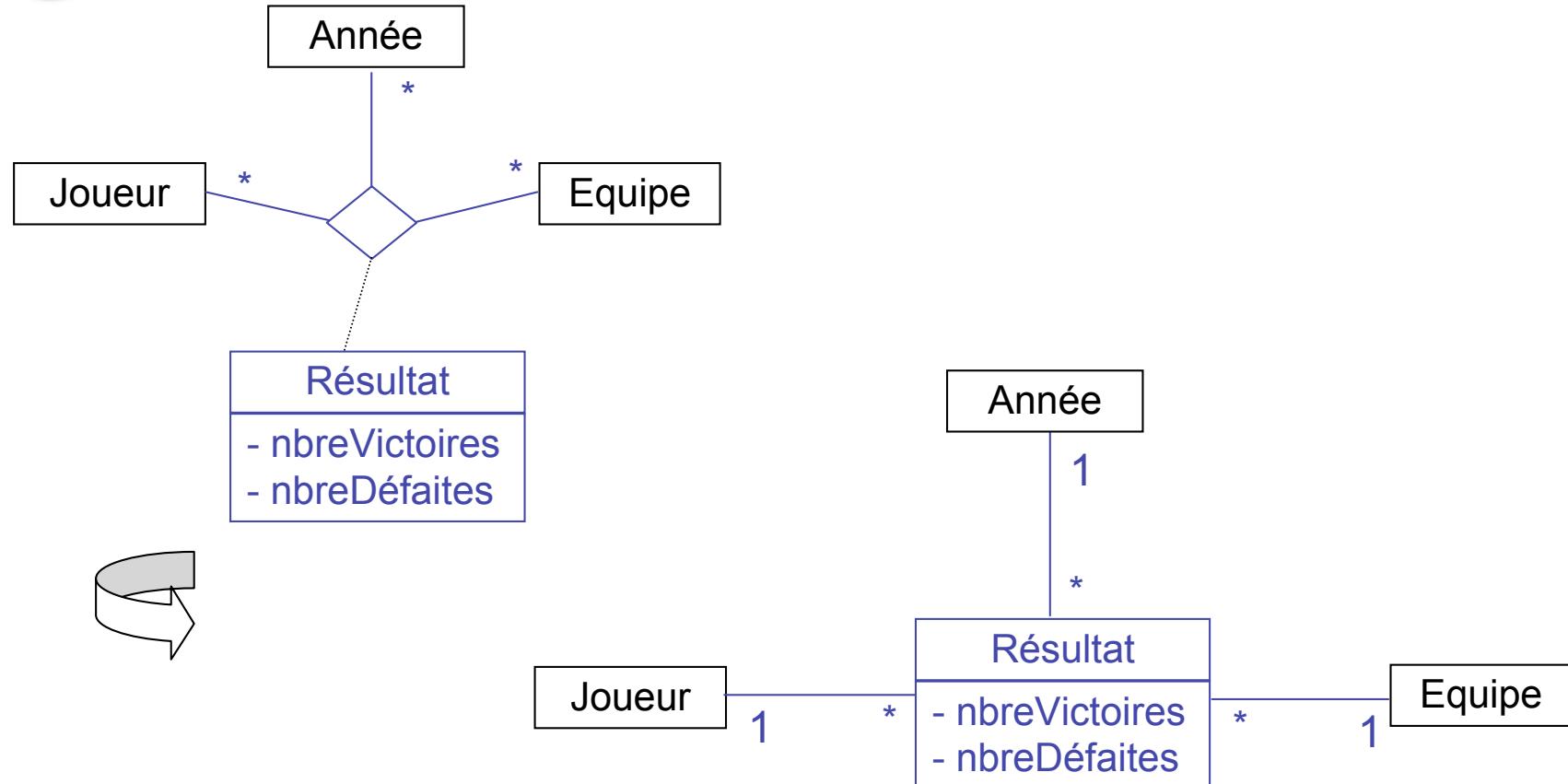
Un joueur peut jouer plusieurs années dans une même équipe

Une équipe, une année donnée, est composée de plusieurs joueurs



## Transformation d'une association ternaire

CLASSES

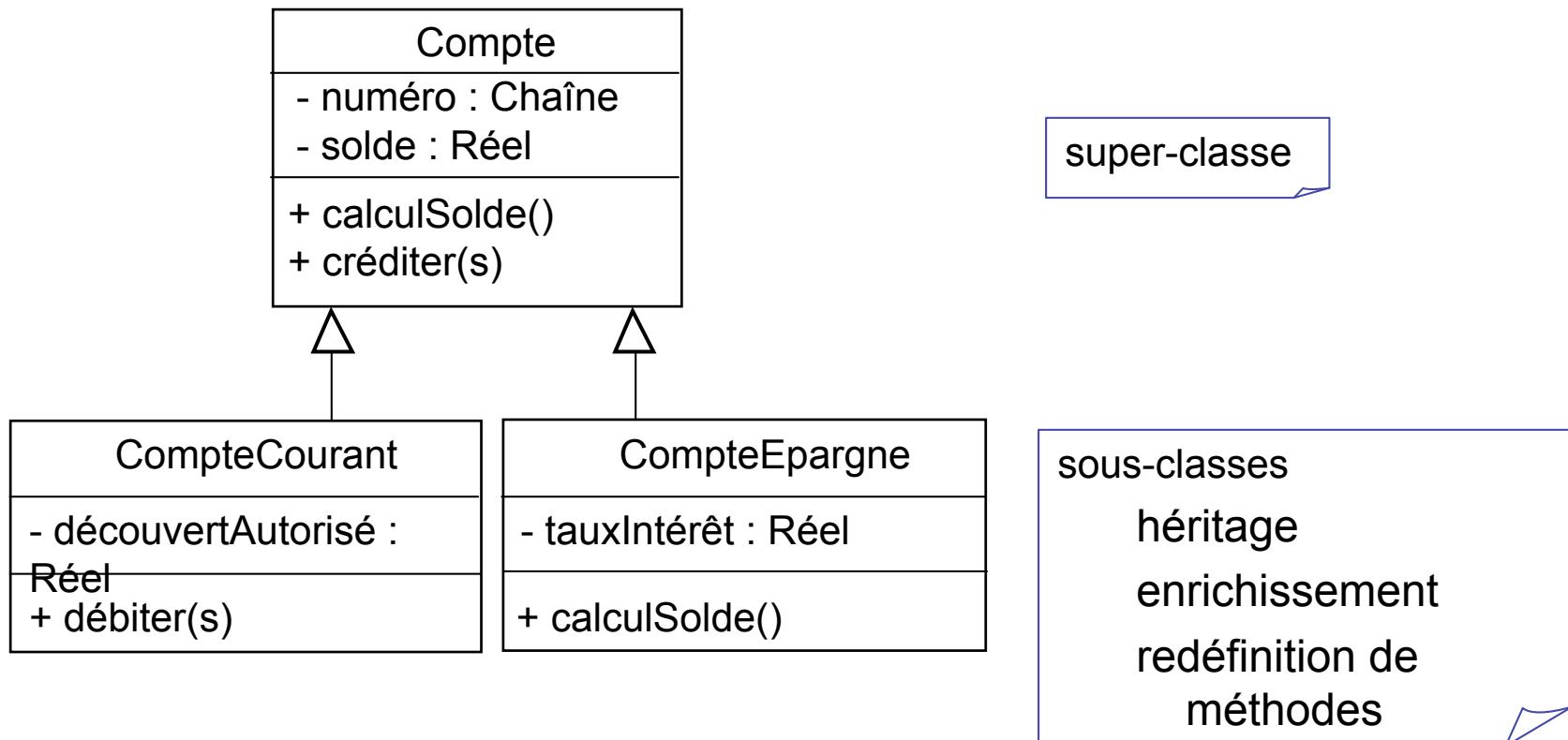


Mais dans le diagramme ci-dessus, un joueur peut avoir plusieurs résultats la même année avec la même équipe.



## Généralisation-spécialisation

- Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.
- Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.



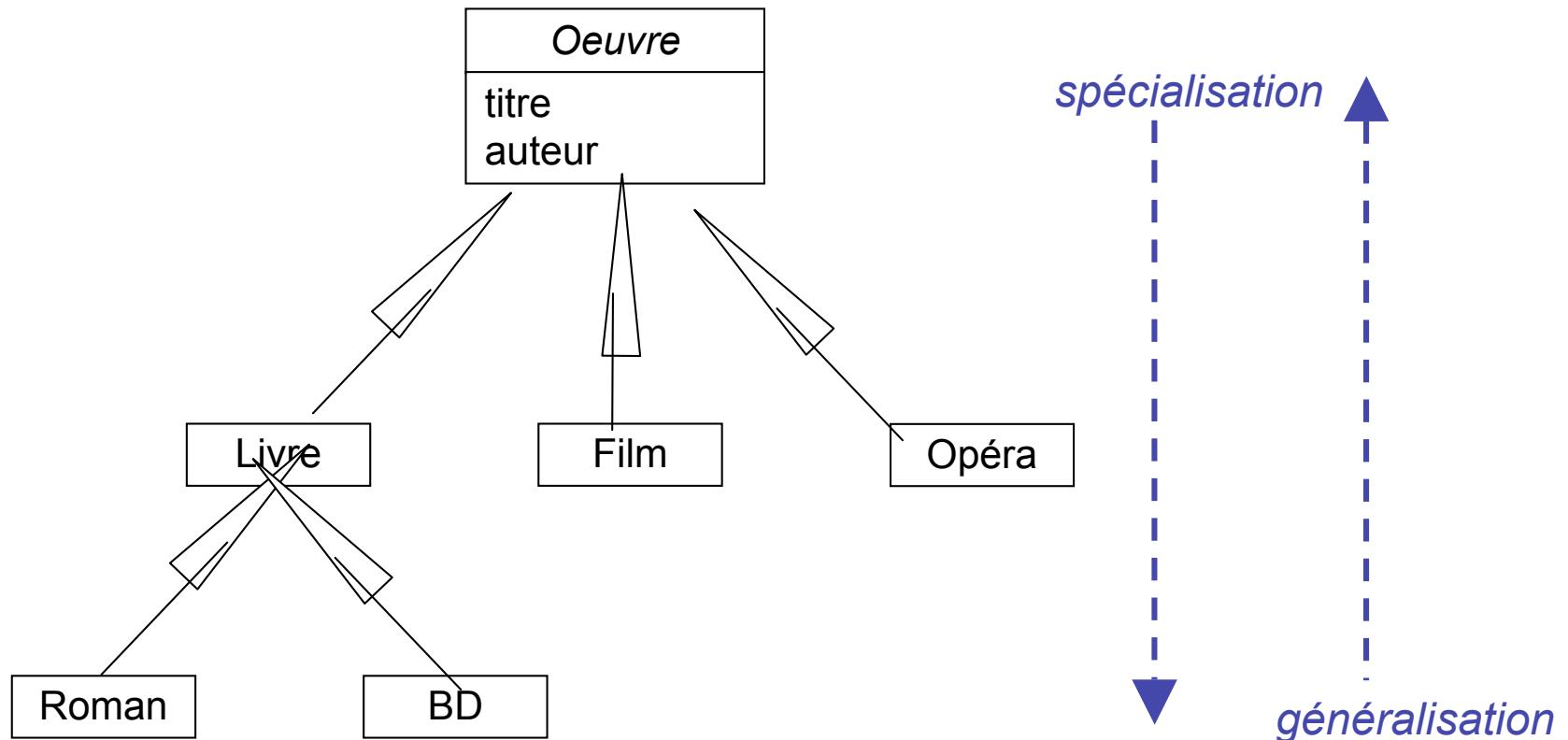
## CLASSES



### Généralisation-spécialisation

La spécialisation et la généralisation permettent de construire des hiérarchies de classes.

- La généralisation et la spécialisation évitent la duplication et encouragent la réutilisation.



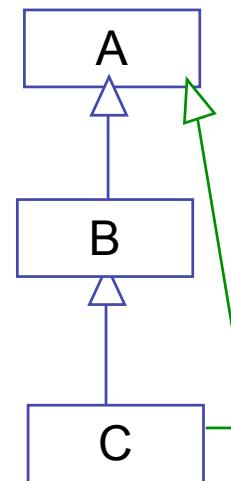
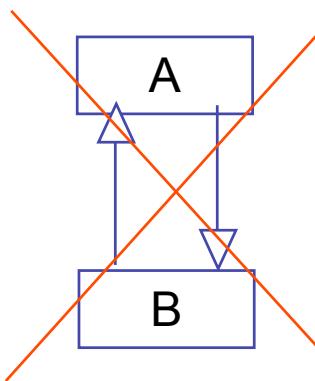
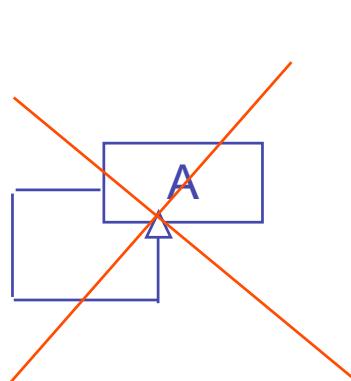


## Généralisation-spécialisation

Sens : «est un» , «est une sorte de» , «est de la famille des»

Propriétés :

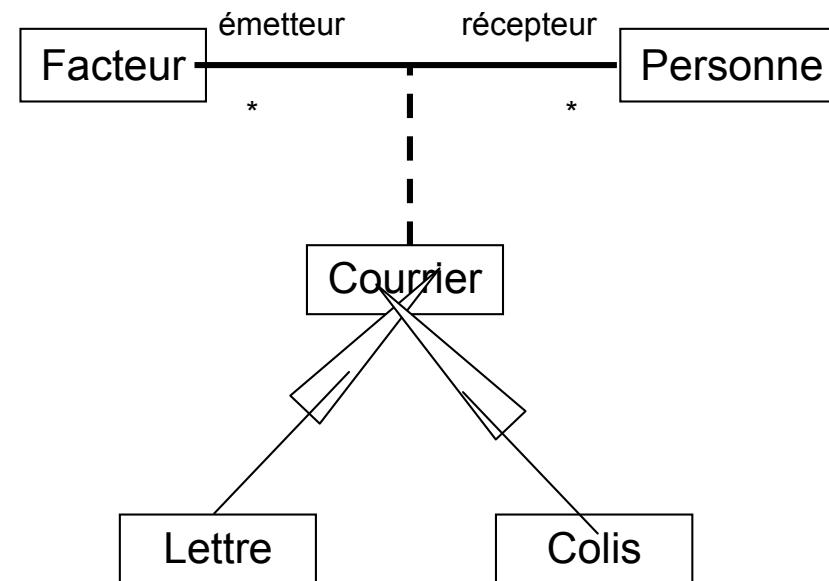
- non réflexive : A n'hérite pas de lui-même ; il EST lui-même
- non-symétrique : B sous-classe de A, interdit A sous-classe de B (pas de cycle)
- transitive : C sous classe de B, B sous-classe de A ==> C sous-classe de A





## Généralisation-spécialisation

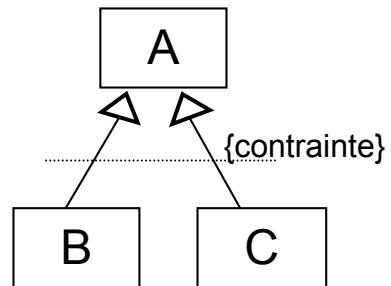
Une classe d'association peut être spécialisée



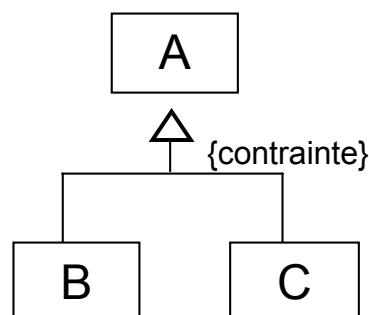


## Contraintes sur la généralisation-spécialisation

- B spécialise A :
- \* les objets de B sont implicitement des objets de A
  - \* les propriétés de A s'appliquent aux objets de B



*Autre notation (en « ratelier »)*



{disjoint} ou {exclusif} :

un objet de A est instance d'au plus une sous-classe

conséquence : B et C n'ont aucune sous-classe commune

{chevauchement} ou {inclusif} :

un objet de A peut être instance de plusieurs sous-classes

conséquence : B et C peuvent avoir une sous-classe commune

{complète} :

un objet de A est instance d'au moins une sous-classe

conséquence : la classification de A est terminée

{incomplète} :

un objet de A peut n'être instance d'aucune sous-classe

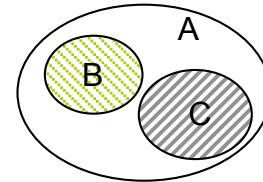
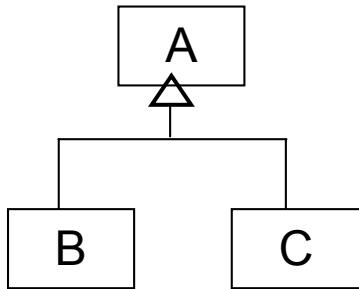
conséquence : la classification de A est extensible



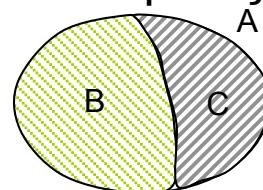
## Contraintes sur la généralisation-spécialisation

CLASSES

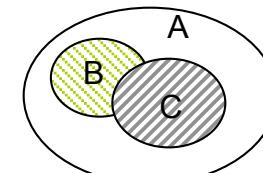
### Cas possibles



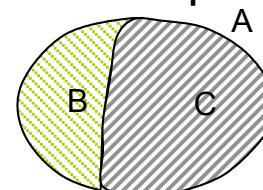
{disjoint,  
incomplète}



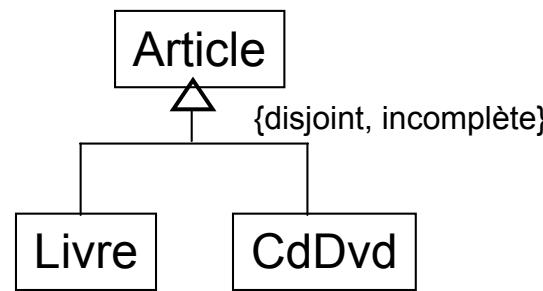
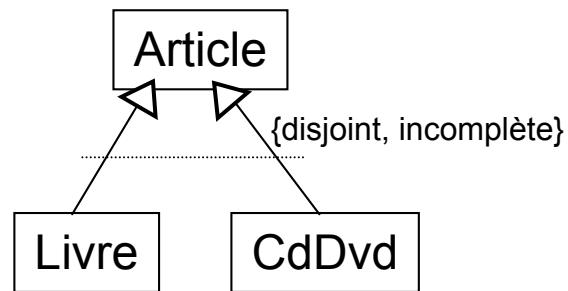
{disjoint,  
complète}



{chevauchement,  
incomplète}

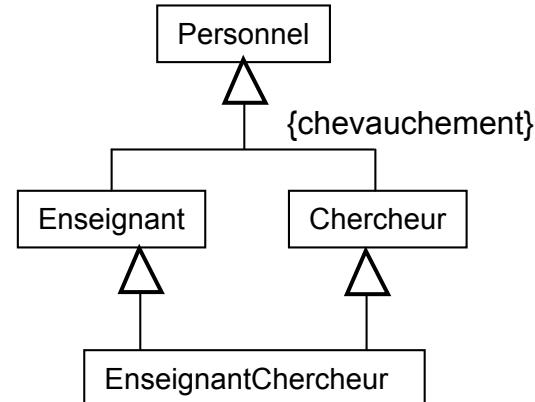


{chevauchement,  
complète}

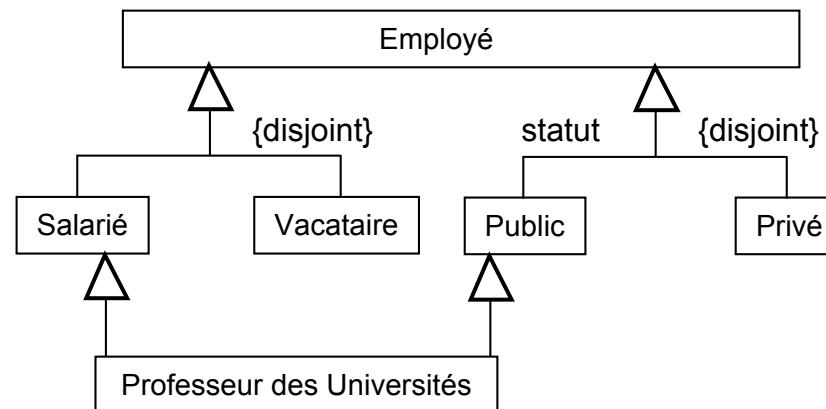




### ➤ Dans une relation de spécialisation avec chevauchement



### ➤ Dans plusieurs branches de spécialisation selon des critères différents





➤ **Contrainte sur une classe**

{abstraite} : classe ne pouvant pas être instanciée (*concrète par défaut*)

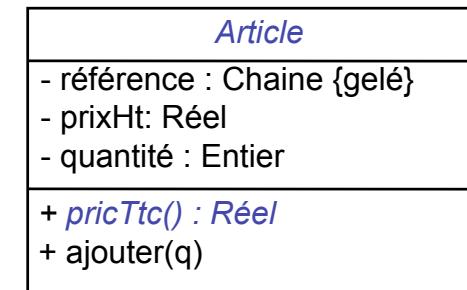
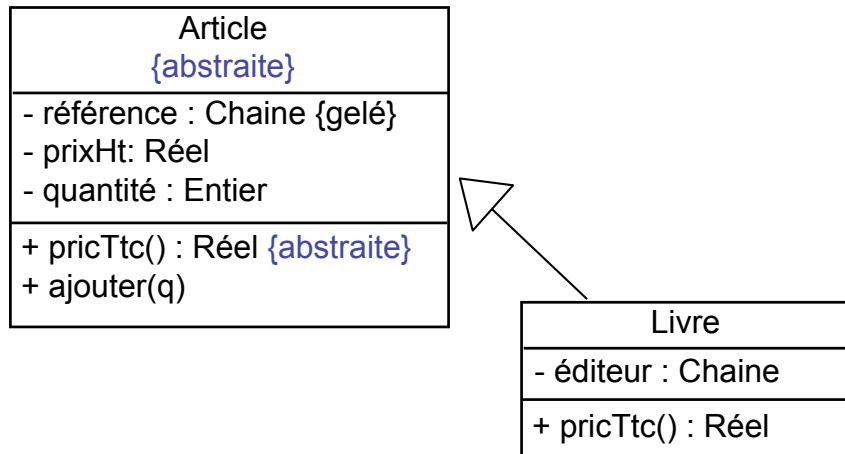
➤ **Contraintes sur un attribut**

{gelé} : mise à jour interdite

➤ **Contraintes sur une opération**

{abstraite} : opération non implémentée. L'implémentation devra être réalisée par une sous-classe concrète.

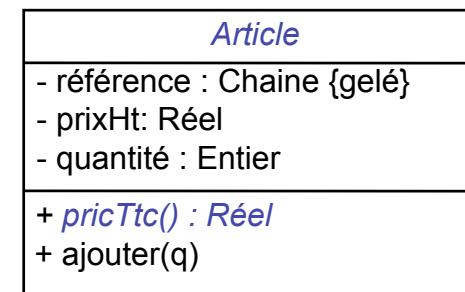
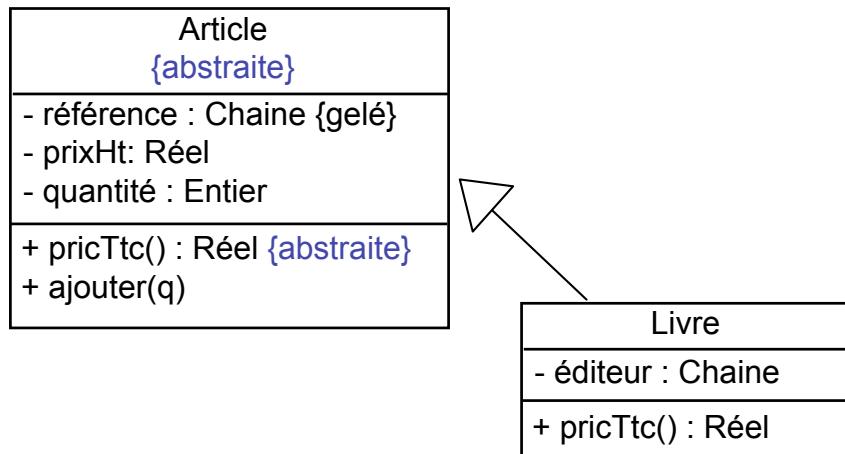
{est-feuille} : redéfinition interdite



autre notation pour classe et opération abstraites  
(en italique)



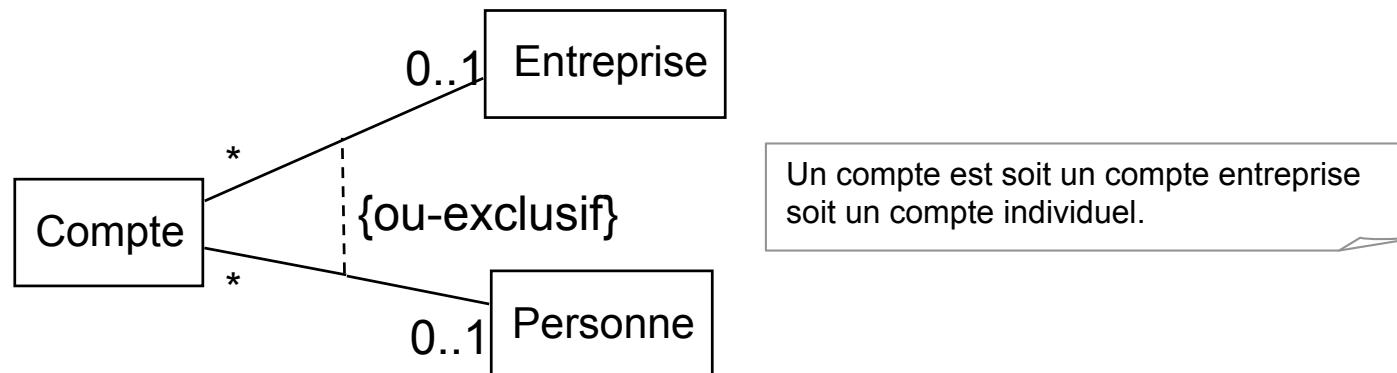
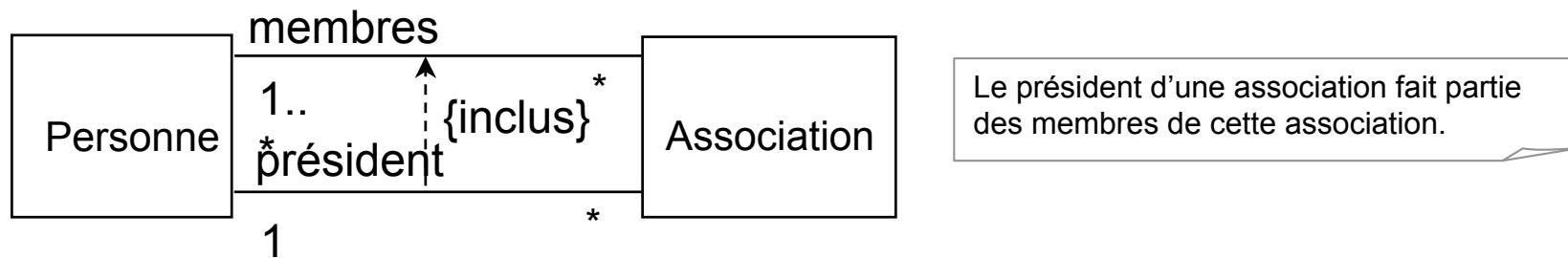
- **Contrainte sur une classe**  
**{abstraite}** : classe ne pouvant pas être instanciée (*concrète par défaut*)
- **Contraintes sur un attribut**  
**{gelé}** : mise à jour interdite
- **Contraintes sur une opération**  
**{abstraite}** : opération non implémentée. L'implémentation devra être réalisée par une sous-classe concrète.



autre notation pour classe et opération abstraites  
(en italique)

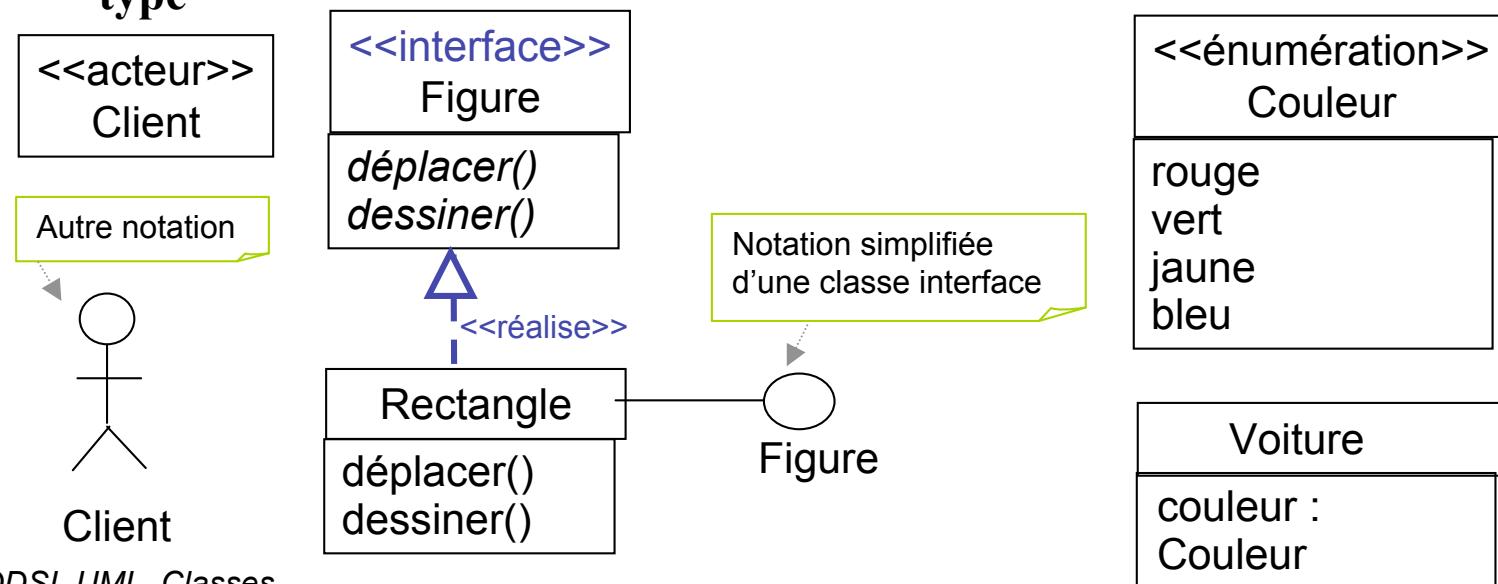


## ➤ Contraintes inter-liens : inclusion, exclusion, totalité





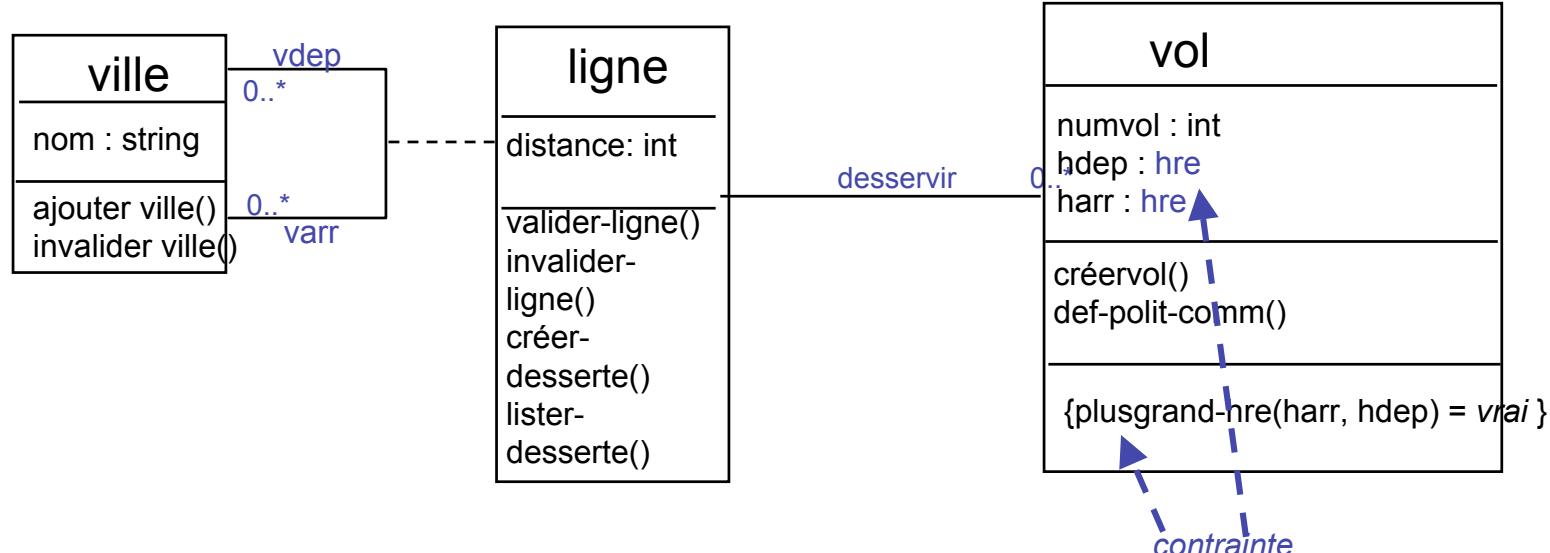
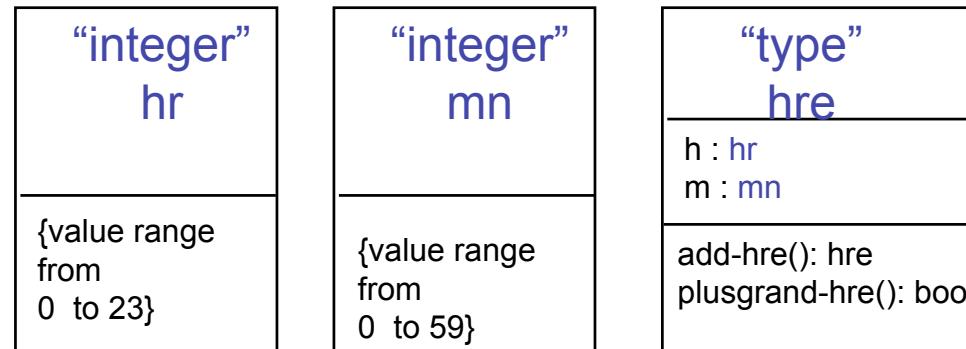
- Un stéréotype permet d'apposer une sémantique particulière aux éléments de modélisation
- Quelques stéréotypes de classe prédéfinis :
  - <<acteur>>** : classe modélisant un ensemble de rôles joués par un acteur
  - <<interface>>** : classe contenant uniquement les opérations publiques abstraites qu'une classe concrète doit implémenter si elle veut disposer de cette interface
  - <<énumération>>** : classe définissant en extension l'ensemble des valeurs d'un type





## Stéréotypes et contraintes

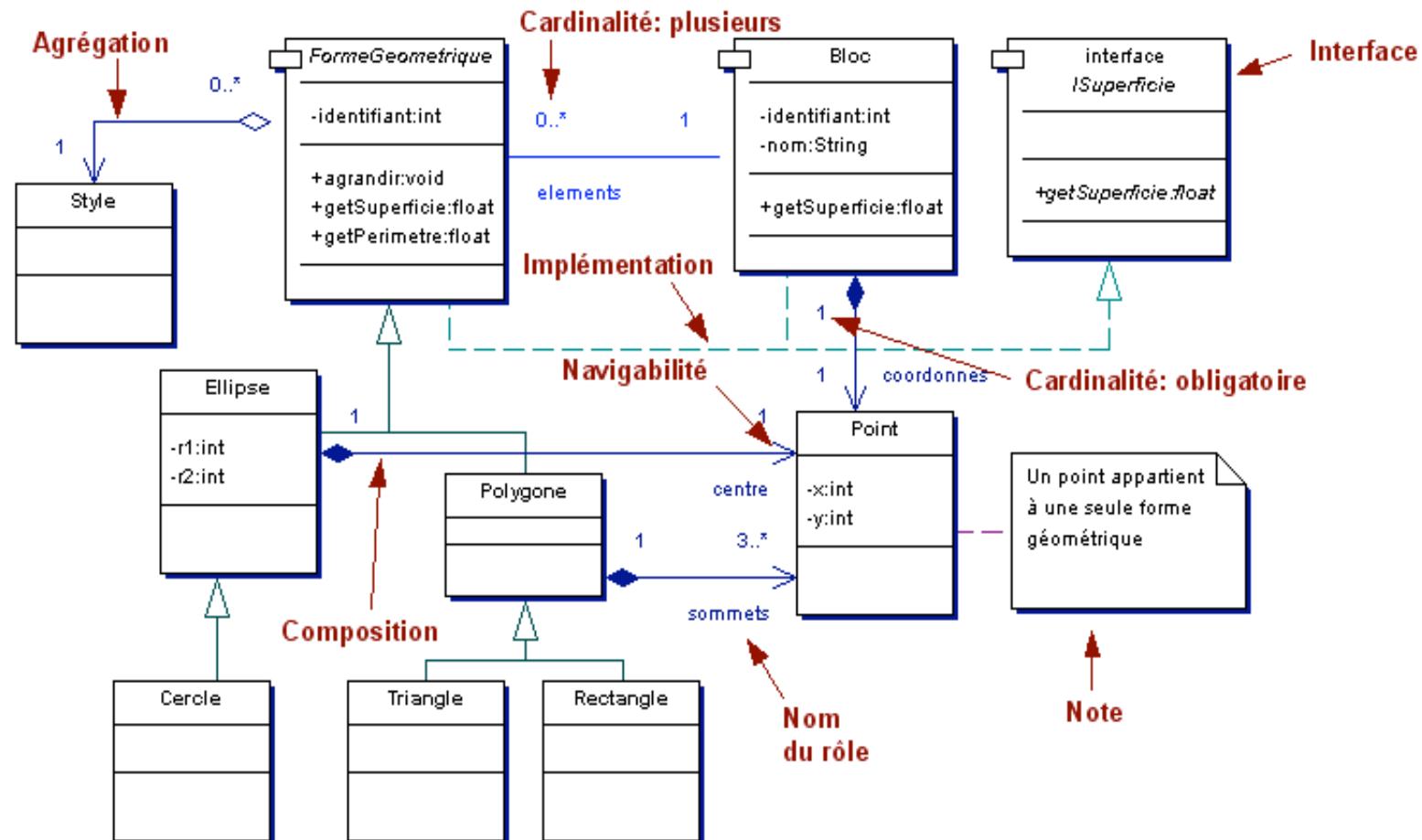
CLASSES





## Diagrammes de classes

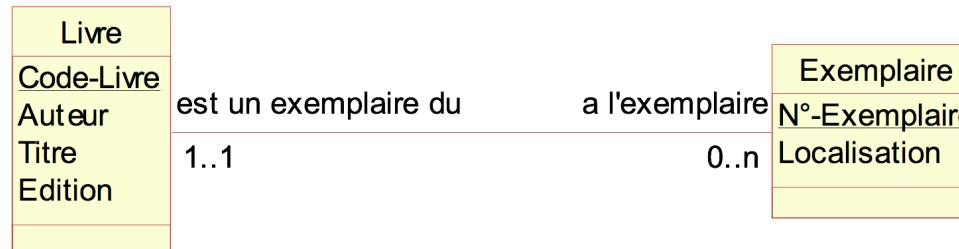
Classe, objet, association, lien, rôle, multiplicité, navigabilité, visibilité, agrégation, composition, spécialisation, généralisation, contrainte, stéréotype





# **Transformation du diagramme de classe en modèle relationnel**

# 1. Règle1: présence de la cardinalité (?..1) d'un côté de l'association



- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champs de table
- L'identifiant de la classe qui est associée à la cardinalité (?..1) (ex: Livre) devient le clé étrangère de l'autre classe (ex: Exemplaire)

Livre (code-livre, auteur, titre, edition)  
Exemplaire (N°-Exemplaire, localisation, code-livre)



**Contrainte d'intégrité référentielle:**

CléEtrangère  $\subseteq$  CléPrimaire

Ex: Exemplaire.Code-Livre  $\subseteq$  Livre.Code-Livre



# Règle 1 - Exemple

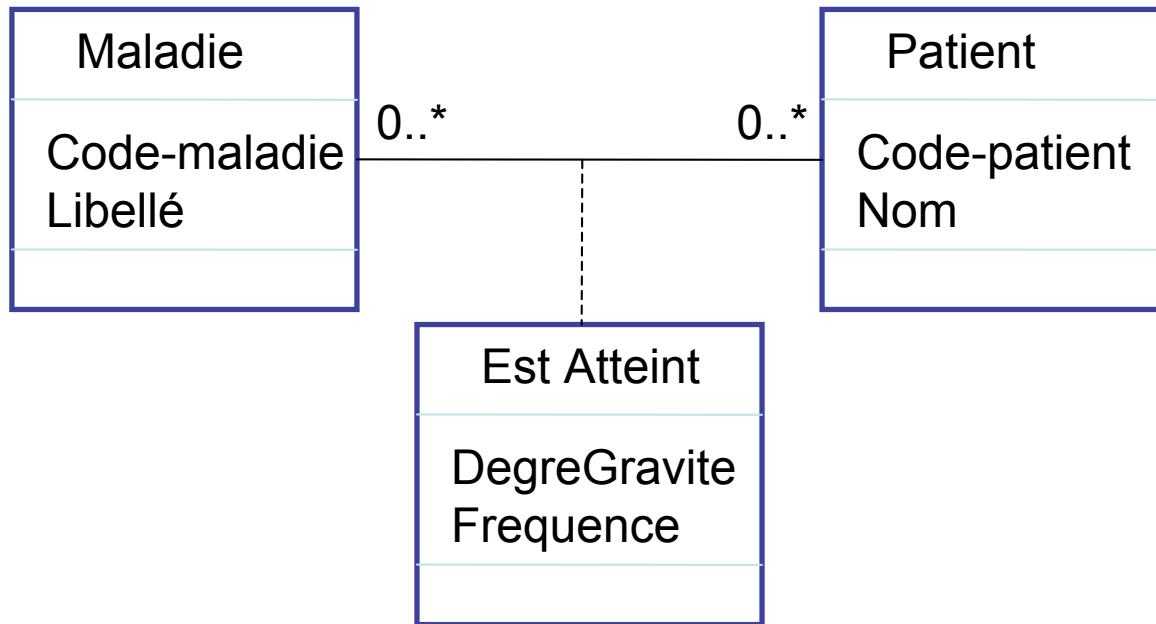
Livre			
<b>Code-Livre</b>	<b>Auteur</b>	<b>Titre</b>	<b>Edition</b>
1	Kundera	Titre1	Edition 1
2	Kundera	Titre2	Edition 2
3	Maalouf	Titre3	Edition 3
4	Hugo	Titre4	Edition 4

Exemplaire		
<b>N°-Exemplaire</b>	<b>Localisation</b>	<b>Code-Livre</b>
10	Localisation 1	1
20	Localisation 2	1
30	Localisation 1	2
40	Localisation 3	3
50	Localisation 4	5 !!!

Impossible !

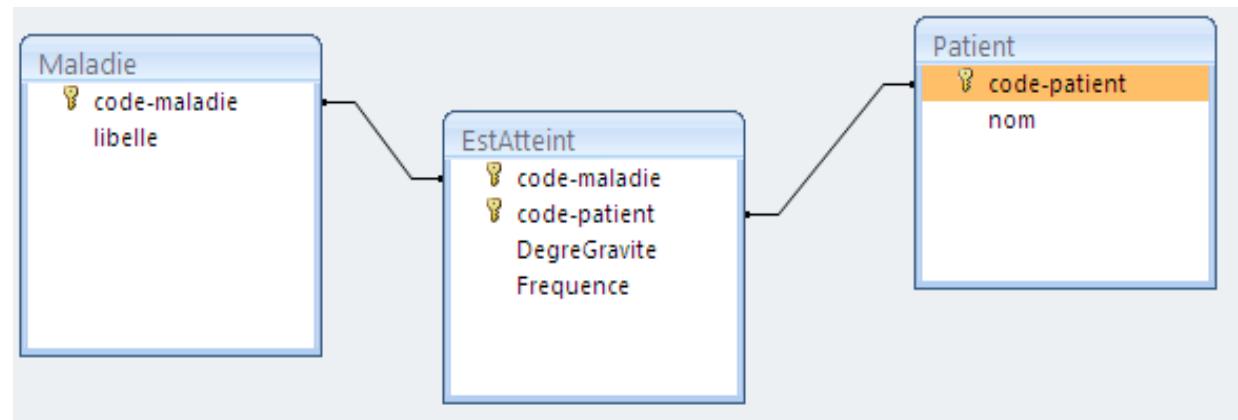
Contrainte d'intégrité référentielle :  
Exemplaire.Code-Livre  $\subseteq$  Livre.Code-Livre

## 2. Règle2: présence de (?..N) des deux côtés de l'association



- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champs de table
- L'association se transforme en une table. Cette table a comme champs l'identifiant de chacune des deux classes, plus d'éventuels autres attributs.  
 $\text{EstAtteint.CodeMaladie} \subseteq \text{Maladie.CodeMaladie}$   
 $\text{EstAtteint.NumPatient} \subseteq \text{Patient.NumPatient}$

- ✿ Schéma relationnel :
- Maladie (code-maladie, libelle)
- Patient (code-patient, nom)
- Est Atteint (code-maladie, code-patient,  
degreGravite, frequence)





# Règle 2 -Exemple

**MALADIE**

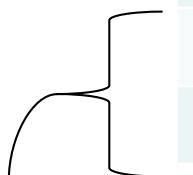
Code-maladie	libelle
M1	Grippe
M2	Angine
M3	Diabète

**PATIENT**

Code-patient	nom
P1	Loulou
P2	Riri
P3	Fifi

**ESTATTEINT**

Code-maladie	Code-patient	DegreGravite	Frequence
M1	P2	5	2fois/an
M1	P3	1	1fois/2ans
M2	P1	1	
M4	P1		
M2	P4		

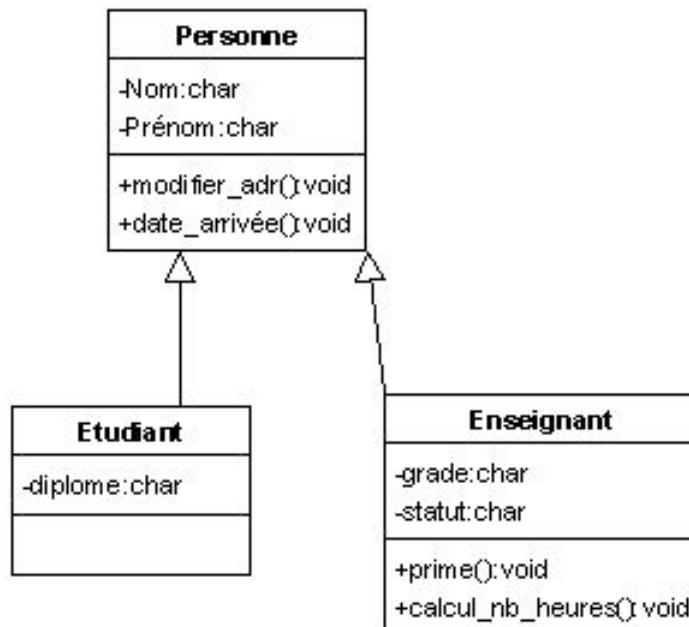


→ **IMPOSSIBLE !**  $\text{EstAtteint.Code-maladie} \subseteq \text{Maladie.Code-maladie}$   
 $\text{EstAtteint.Code-patient} \subseteq \text{Patient.Code-patient}$



## METHODE 1

# Règle 3 : généralisation/spécialisation



Created with Poseidon for UML Community Edition. Not for Commercial Use.

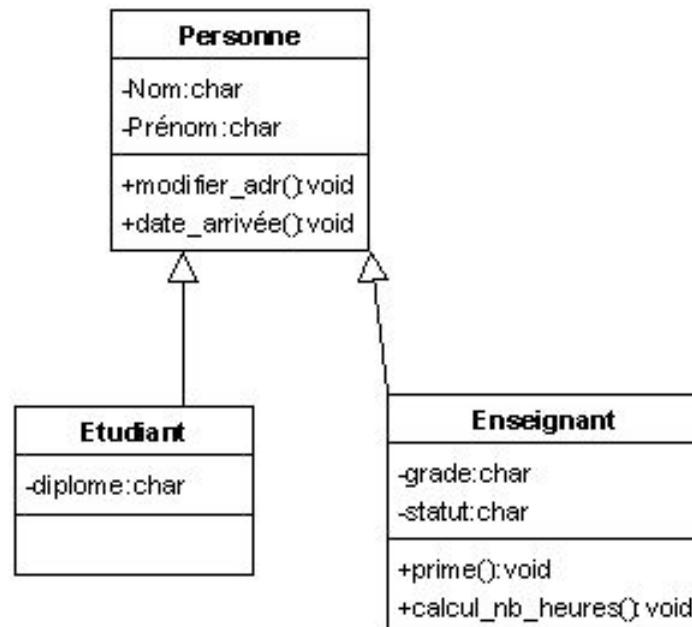
1. Créer une table avec tous les attributs des classes
2. Ajouter un attribut pour distinguer le type des objets

Personne (numer, typeper, nom, prenom, diplôme, grade, statut)

{etudiant, enseignant}

84

# Règle 3 : généralisation/spécialisation



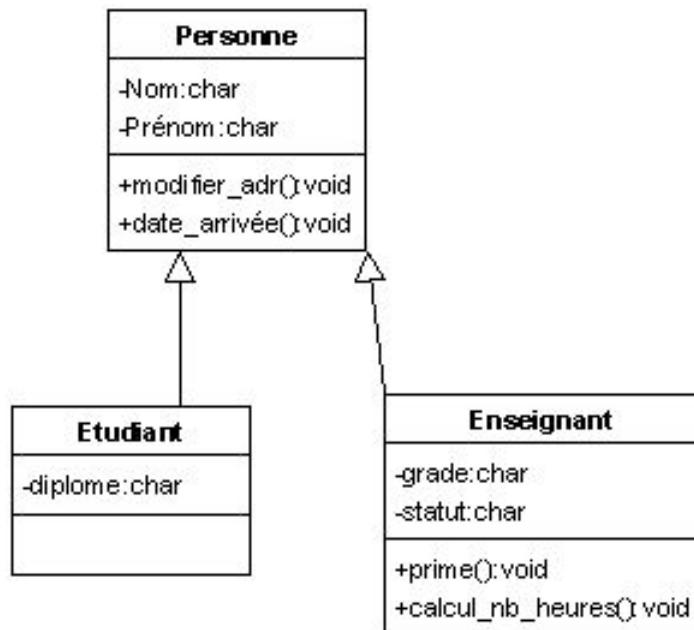
Created with Poseidon for UML Community Edition. Not for Commercial Use.

1. Créer une table pour chaque sous-type
2. Chaque table contient les attributs génériques + ses attributs spécifiques

Etudiant (numEtu, nom, prenom, diplôme)

Enseignant (numEns, nom, prenom, grade, statut)

## Règle 3 : généralisation/spécialisation



Created with Poseidon for UML Community Edition. Not for Commercial Use.

1. Créer une table par classe et des associations
2. Ajouter l'identifiant de la sur-classe dans les sous-classes

Personne (NumPer, nom, prenom)

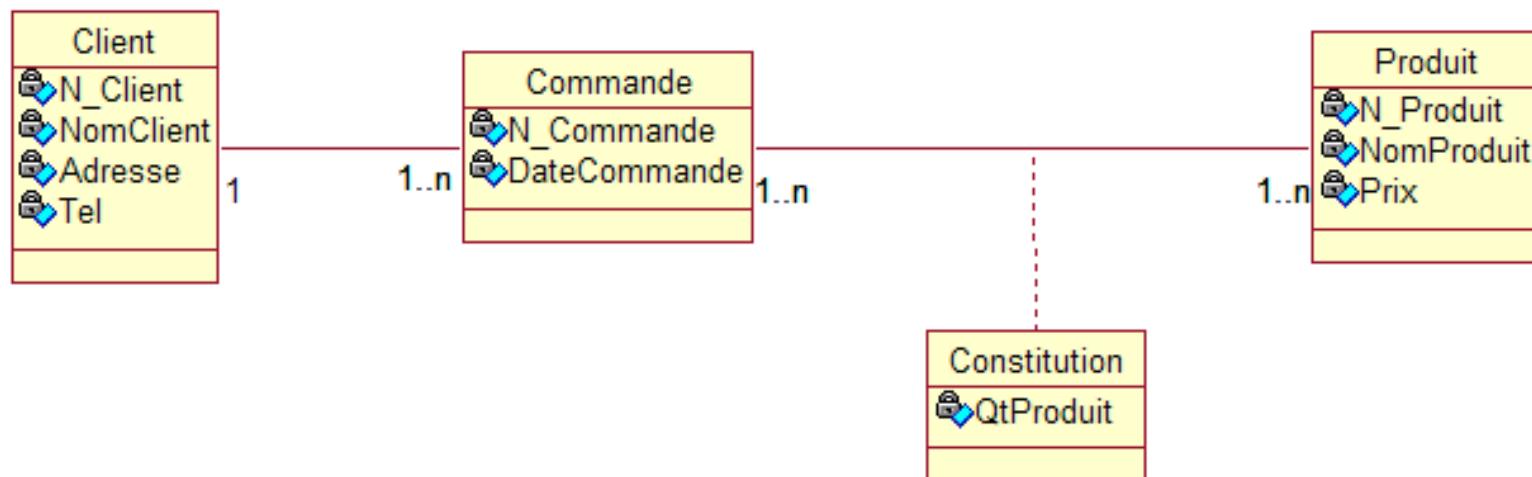
Etudiant (numEtu, diplôme, NumPer)

Enseignant (numEns, grade, statut, NumPer)



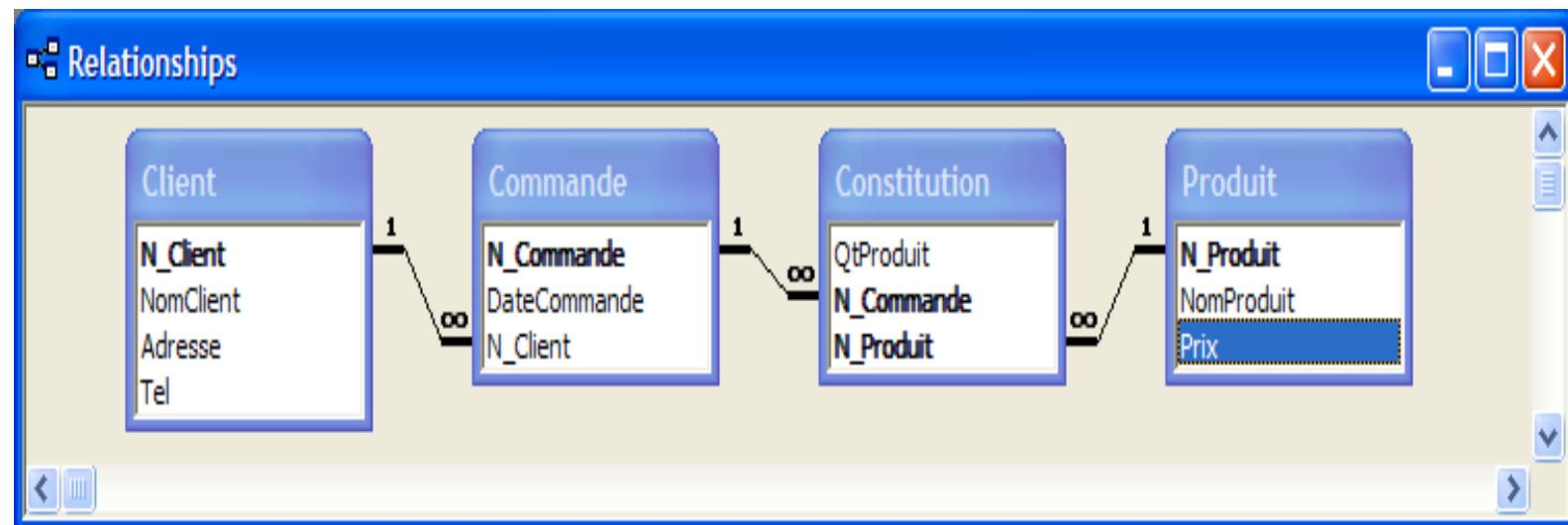
# Exercice 1

- ✿ Construire le modèle relationnel correspondant au diagramme de classe suivant:





# Exercice 1: Solution





Client

<u>N_Client</u>	<u>NomClient</u>	<u>Adresse</u>	<u>Tel</u>
1	Michel	Paris	123456
2	David	Bruxelles	456298
3	Manuel	Chicago	876230
4	Lucas	Miami	937402
5	Tintin	Bruxelles	384043

Commande

<u>N_Commande</u>	<u>DateCommande</u>	<u>N_Client</u>
1	12/09/ 00	1
2	15/03/ 01	1
3	12/09/ 01	3
4	10/01/ 06	3
5	20/10/0 7	4
6	15/02/0 7	5

Constitution

<u>QtProduit</u>	<u>N_Commande</u>	<u>N_Produit</u>
1	5	200
1	5	400
2	3	500
3	6	500
2	5	600
2	4	600

Produit

<u>N_Produit</u>	<u>NomProduit</u>	<u>Prix</u>
100	Produit1	5.000,00
200	Produit 2	20.000,00
300	Produit 3	10.000,00
400	Produit 4	200.000,00
500	Produit 5	500,00
600	Produit 6	1.200,00

Contraintes d'intégrité référentielle :

$\text{Commande.N_Client} \subseteq \text{Client.N_Client}$

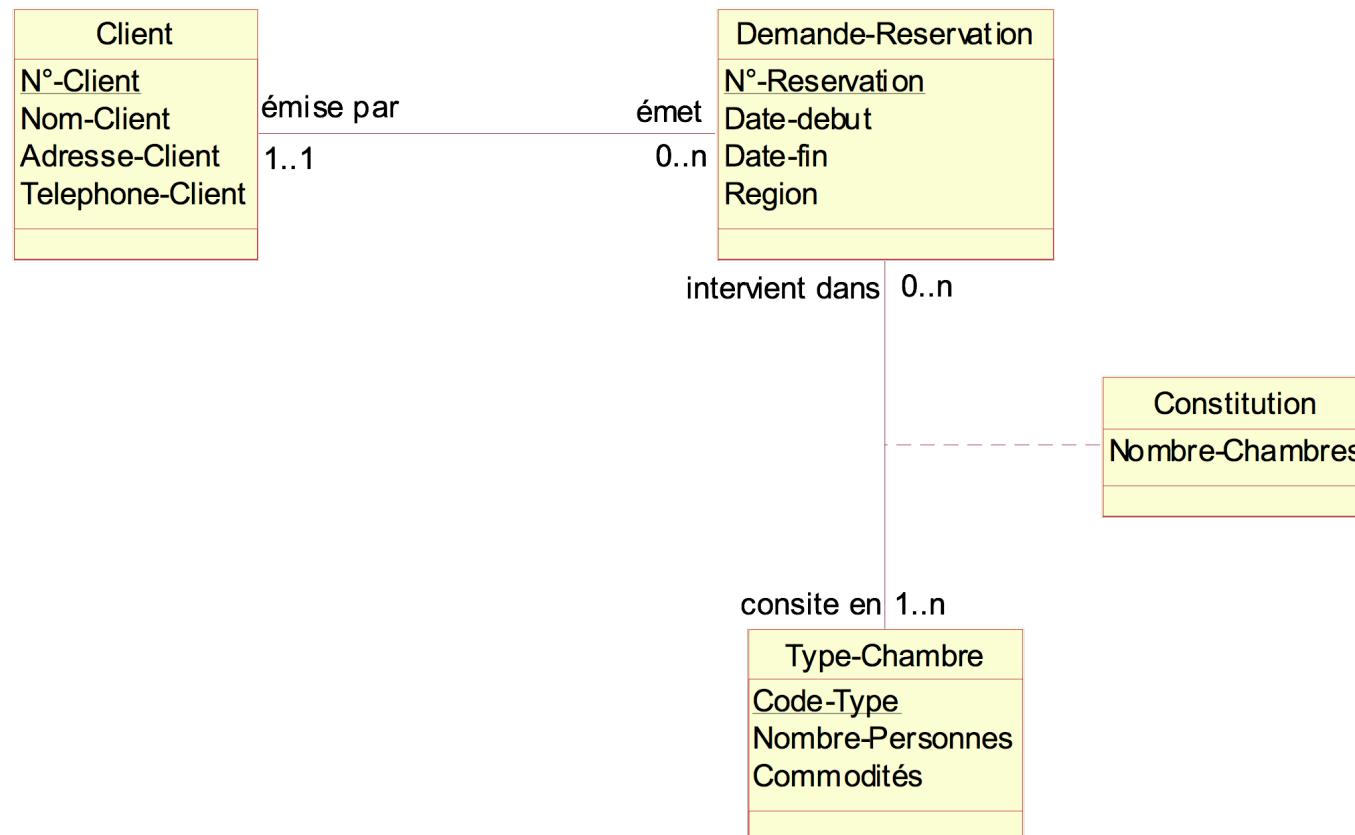
$\text{Constitution.N_Commande} \subseteq \text{Commande.N_Commande}$

$\text{Constitution.N_Produit} \subseteq \text{Produit.N_Produit}$



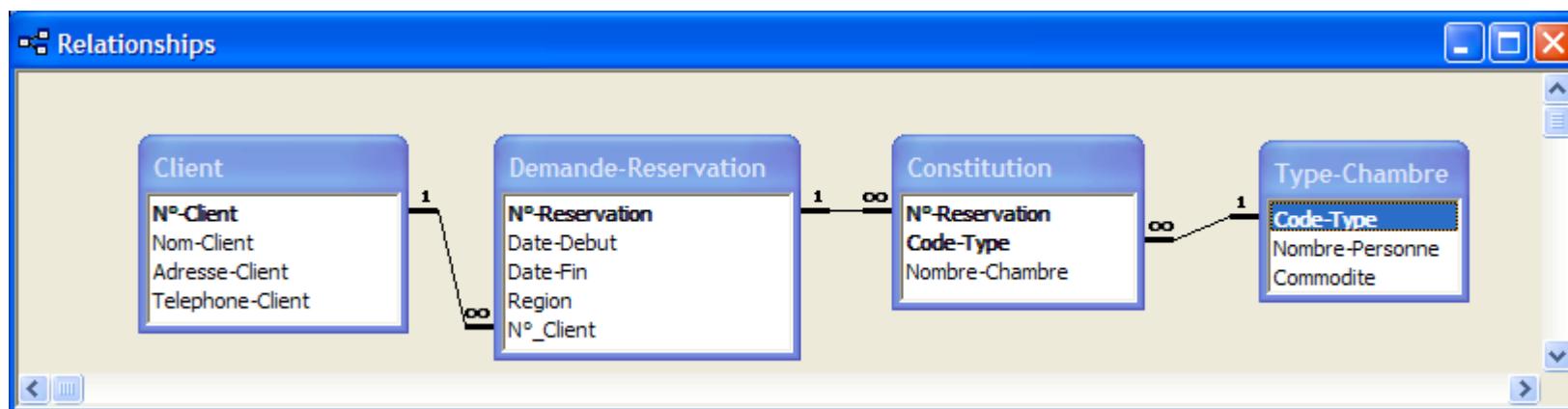
# Exercice 2

Construire le modèle relationnel correspondant au diagramme de classe suivant:





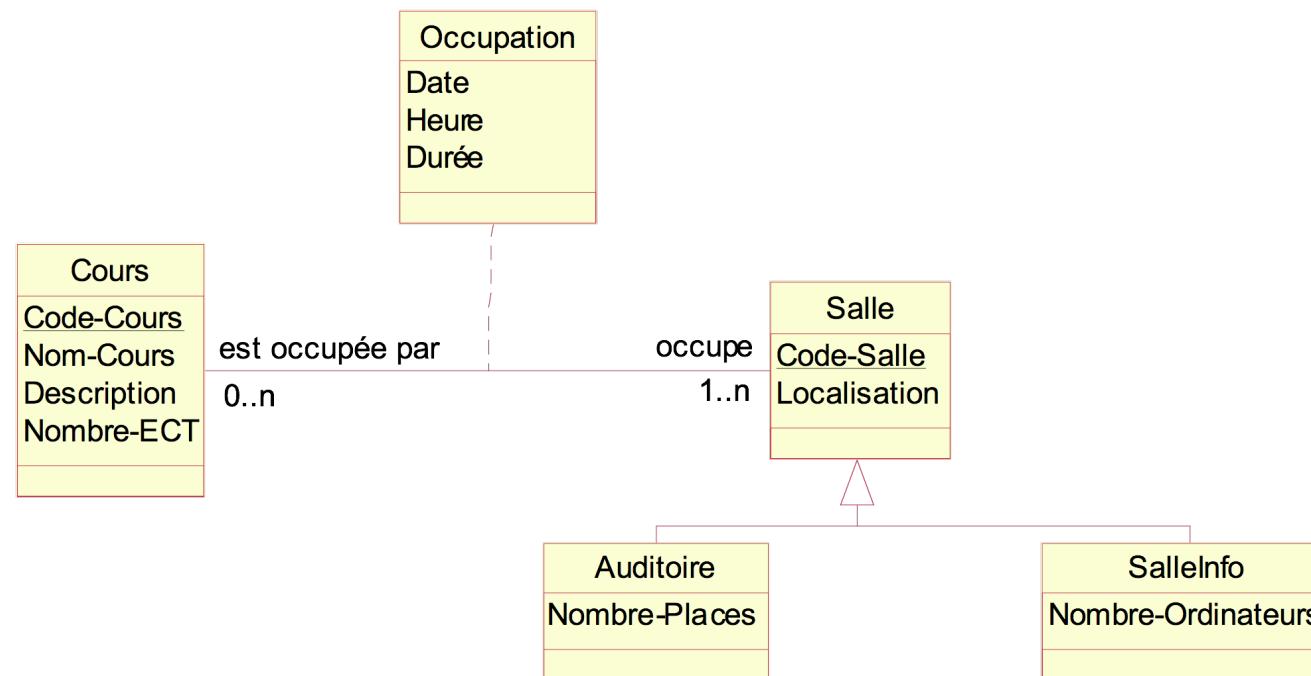
## Exercice 2: Solution





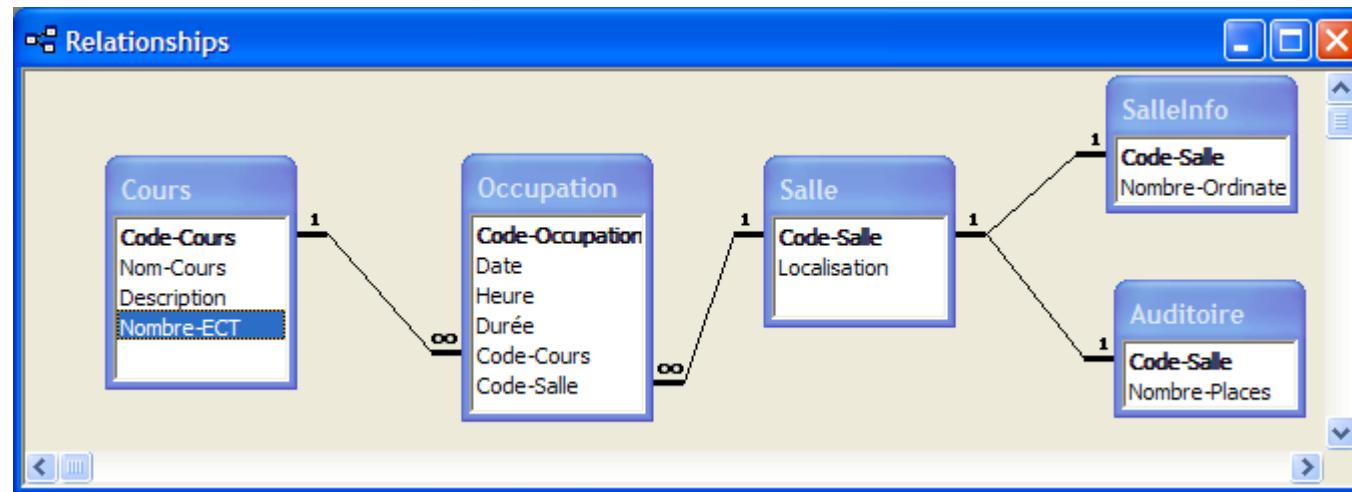
# Exercice 3

Construire le modèle relationnel correspondant au diagramme de classe suivant:





# Exercice 3: Solution





# Conclusion

- ✿ Si les identifiants n'existent pas, il est nécessaire de les créer dans le modèle relationnel.
- ✿ Il est possible d'être amené à ajouter des attributs n'existant pas dans le diagramme de classe.
- ✿ Dans tous les cas, il faudra faire attention à la compatibilité entre les 2 modèles une fois la transformation réalisée.



# Actions de conception de SI

## Méthode européenne (MERISE)

Modèle entité-association =  
(Modèle Conceptuel de Données  
ou MCD) + modèles dynamiques



Modèle relationnel (théorie  
de la normalisation)



Implantation dans un SGBD  
relationnel : Oracle, DB2,  
Access, etc.

## Méthode anglosaxonne (OMT, Booch, etc.)

Diagramme de classe UML + modèles  
dynamiques



Implémentation direct dans un langage  
orienté objet (Java, C++)



Implantation dans un SGBD objet : Poët  
OU  
Implantation dans un SGBD-R qui intègre une  
couche objet : Oracle par ex



# Actions de conception de SI

