

# **Diagramme des Classes et des objets**

Youcef Hammal

Cours de Génie Logiciel

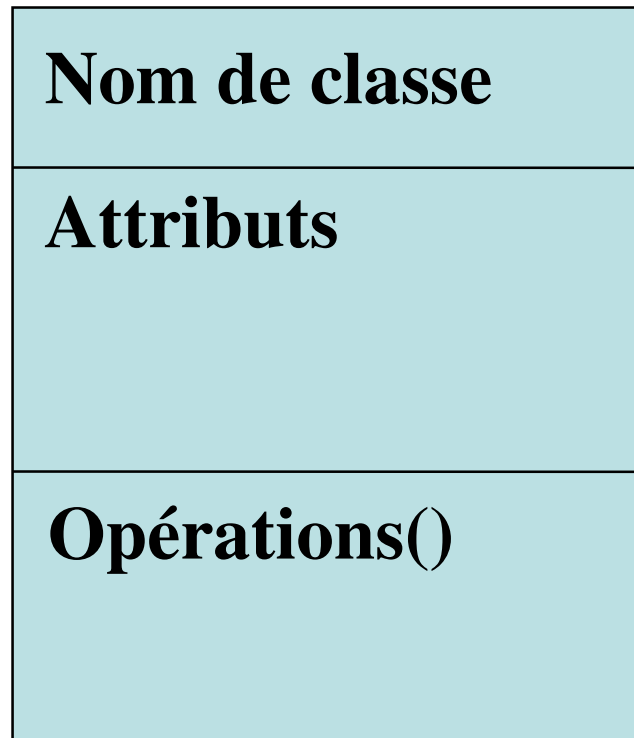
# Les classes

- Le monde réel est constitué de très nombreux objets en interaction.
- Ces objets constituent des amalgames souvent trop complexes pour être compris dans leur intégralité du premier coup.
- Pour réduire cette complexité et comprendre ainsi le monde qui l'entoure, l'être humain a appris à regrouper les éléments qui se ressemblent et à distinguer des structures de plus haut niveau, débarrassées de détails inutiles.

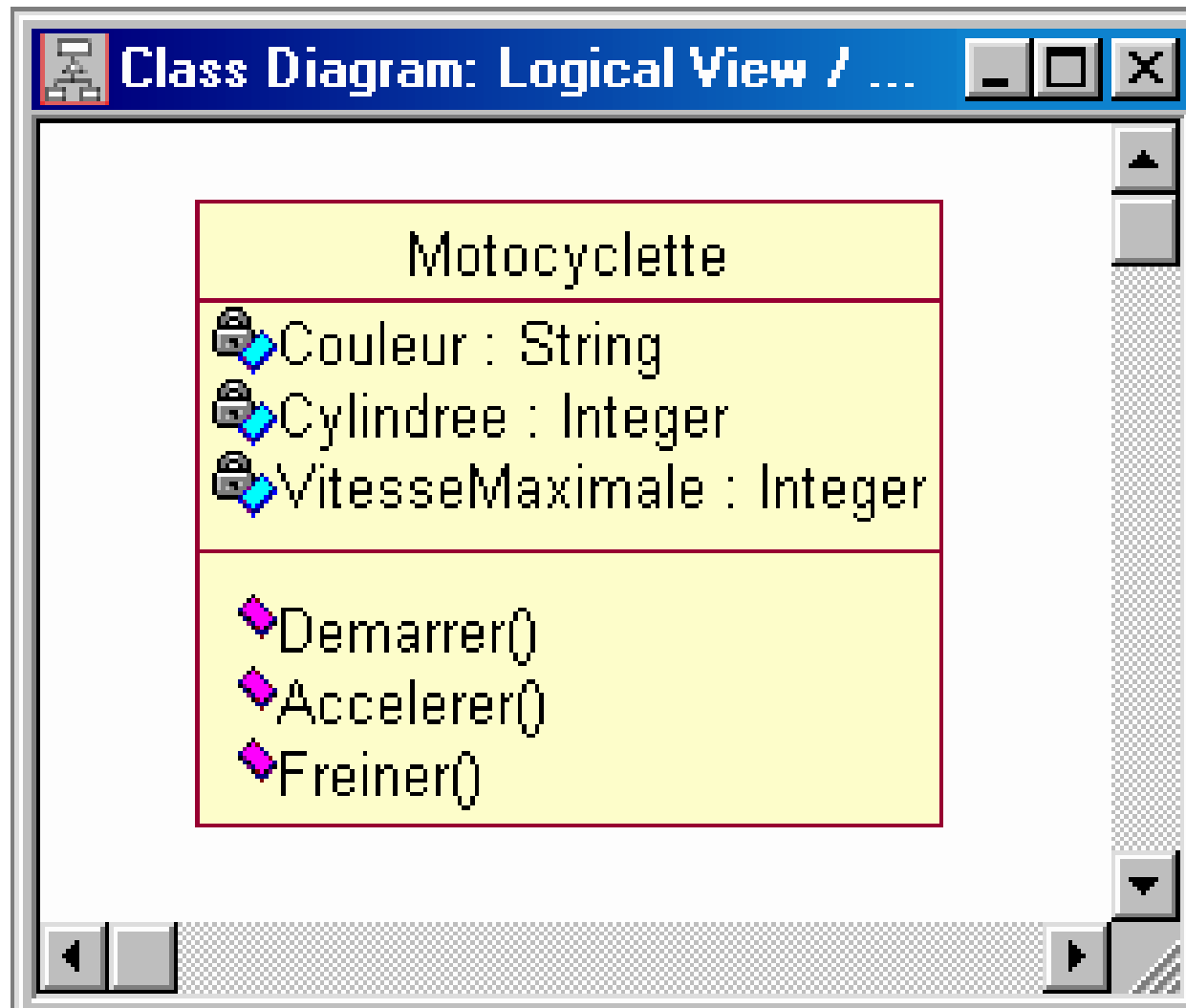
# Classe & objets

- La classe décrit le domaine de définition d'un ensemble d'objets.
- Chaque objet appartient à une classe.
- Les généralités sont contenues dans la classe et les particularités sont contenues dans les objets.
- Les objets informatiques sont construits à partir de la classe, par un processus appelé instantiation. De ce fait, tout objet est une instance de classe.

# Représentation graphique des classes



# Exemple



# Contrat

- Une classe passe un **contrat** avec d'autres classes :
  - elle **s'engage à fournir les services publiés** dans sa spécification
  - et les autres classes **s'engagent à ne pas faire usage de connaissances** autres que celles connues dans cette spécification.
- La séparation entre la spécification et la réalisation des classes participe à l'élévation du niveau d'abstraction.
- Les traits remarquables sont décrits dans les spécifications alors que les détails sont confinés dans les réalisations.
- **L'occultation des détails de réalisation** est appelée **encapsulation**.

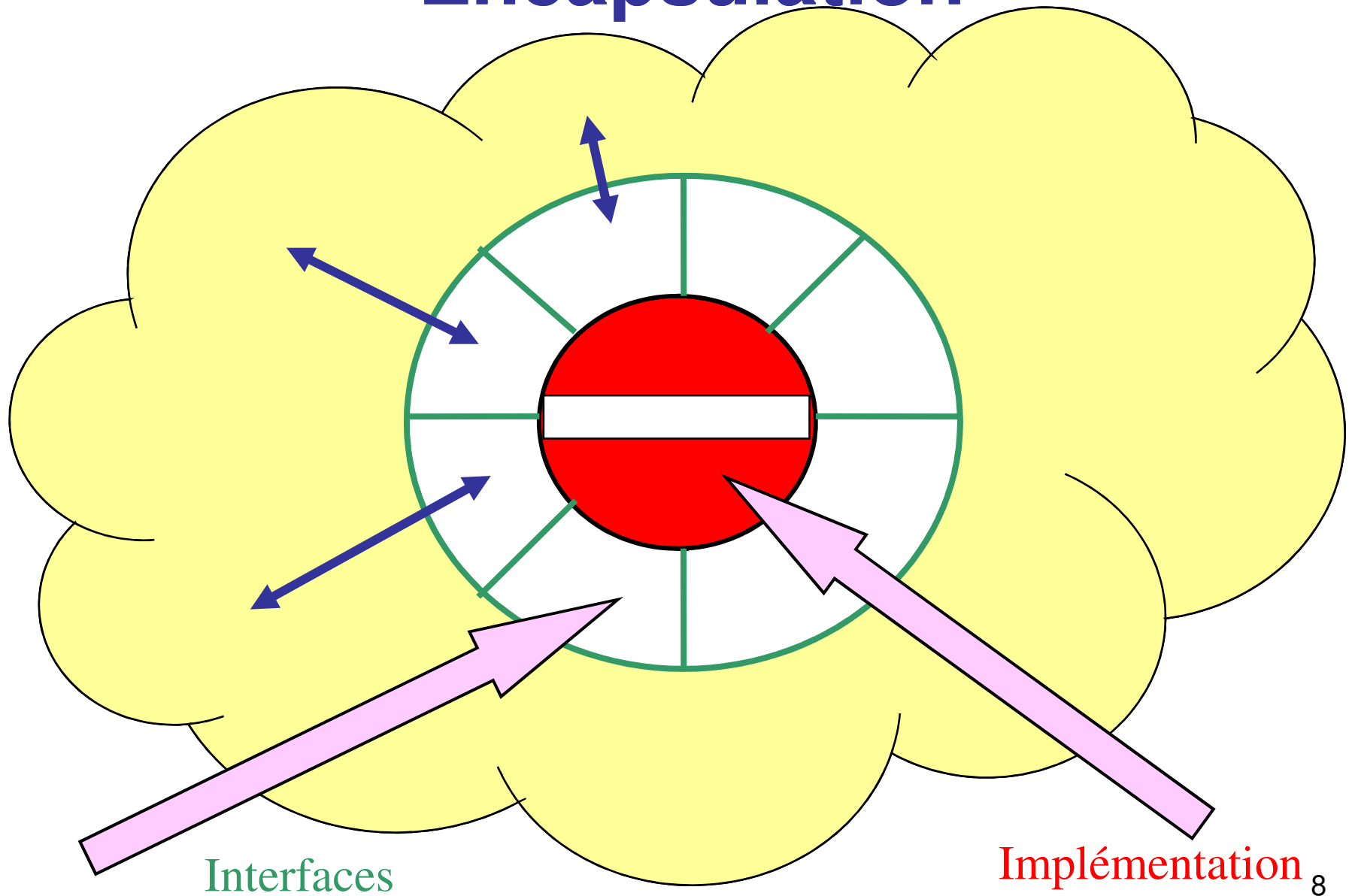
# Compartiments supplémentaires

Nom de classe
Attributs
Opérations
Responsabilités
Exceptions

→ Messages

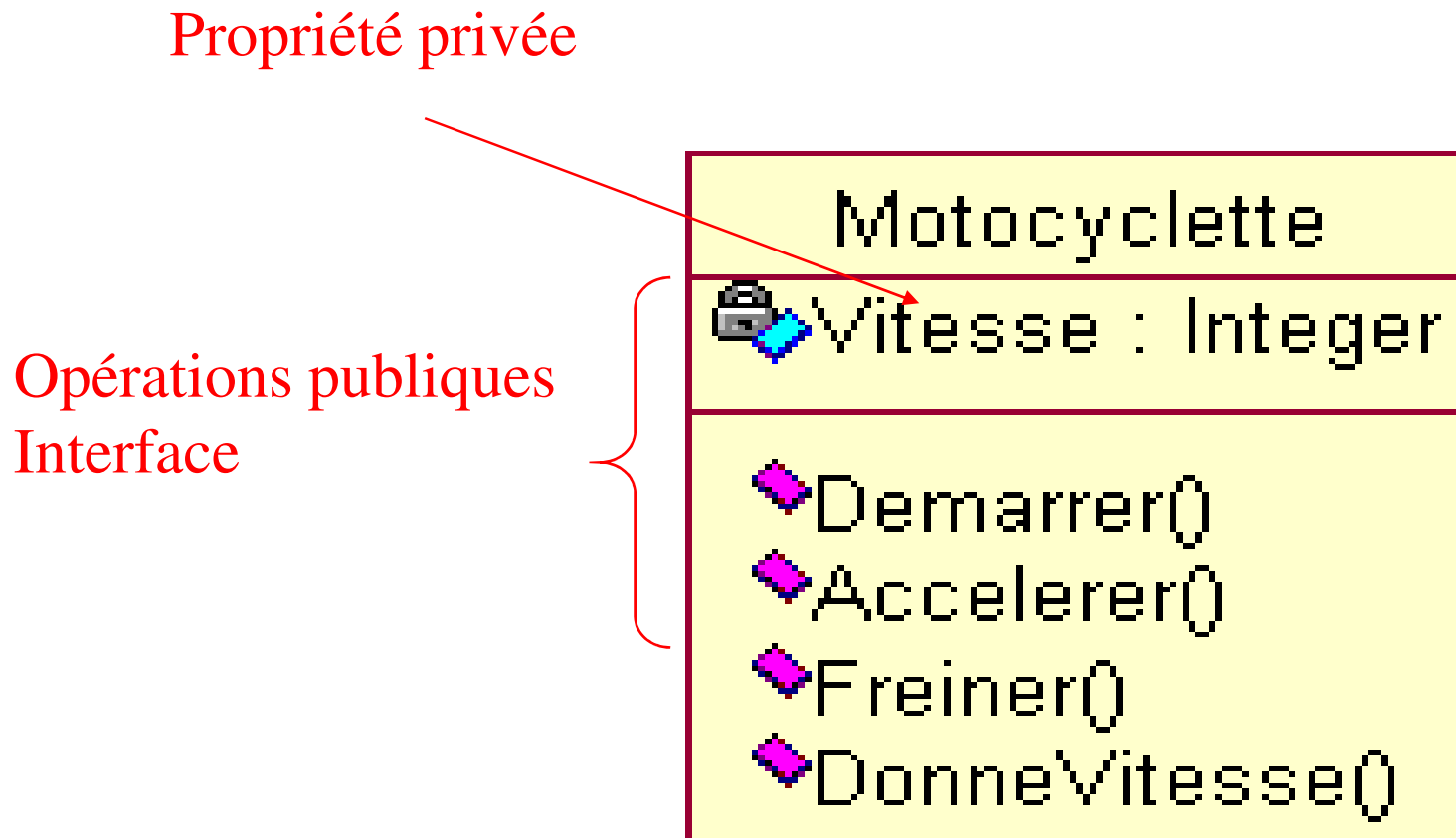
→ Contrat

# Encapsulation





# Exemple d'encapsulation



# Règles de visibilité

- Par défaut, les valeurs d'attributs d'un objet sont **encapsulés** dans l'objet et ne peuvent pas être manipulés directement par les autres objets.
- Toutes les interactions entre les objets sont effectués en déclenchant les diverses opérations déclarées dans la spécification de la classe et accessibles depuis les autres objets.
- ~~• L'intérêt de briser l'encapsulation est par exemple de réduire le temps d'accès aux attributs...~~

# Visibilité

- La visibilité est l'un des détails les plus importants que l'on puisse spécifier pour les attributs et les opérations d'une classe.
- La visibilité d'une caractéristique détermine si d'autres classes peuvent l'utiliser.
- En UML, on utilise 4 niveaux de visibilité.
  1. **Public** ou +
  2. **Protected** ou #
  3. **Private** ou –
  4. **Package** ou ~


# Règles de visibilité - niveaux


## Règles de visibilité


+ Attribut public  
# Attribut protégé  
- Attribut privé


+ Opération publique()  
# Opération protégée  
- Opération privée


## Vehicule

 Vitesse : Integer

 Demarrer()

 Accelerer()

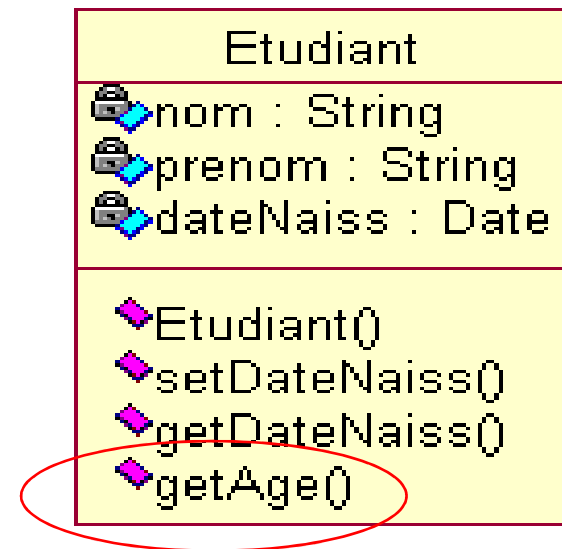
 Freiner()

 DebrancherABS()

 DonneVitesse()

# Critères d'encapsulation

- Les critères d'encapsulation reposent sur la **forte cohérence interne** à l'intérieur d'une classe et sur le **faible couplage** entre les classes.
- Il ne suffit pas pour obtenir une bonne abstraction, de rassembler des données et de fournir des opérations pour la lecture et l'écriture de ces données.
- *Une classe doit offrir une valeur ajoutée par rapport à la simple juxtaposition d'informations.*



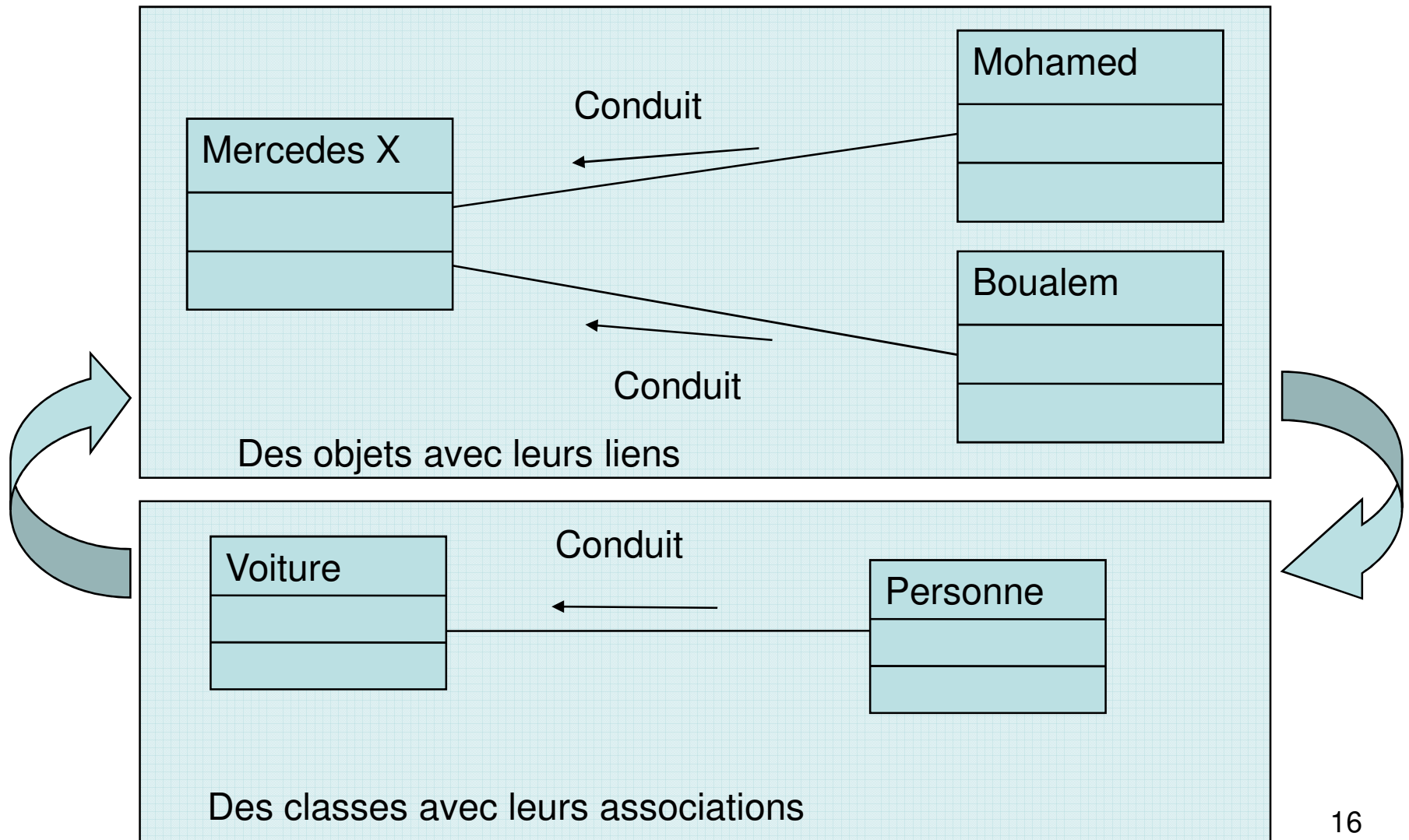
# Les relations entre les classes

- *A chaque famille de **liens entre objets** correspond une **relation entre les classes** de ces mêmes objets.*
- *2 sortes de relations entre les classes*
  - *Les associations*
  - *Les agrégations*

# L'association

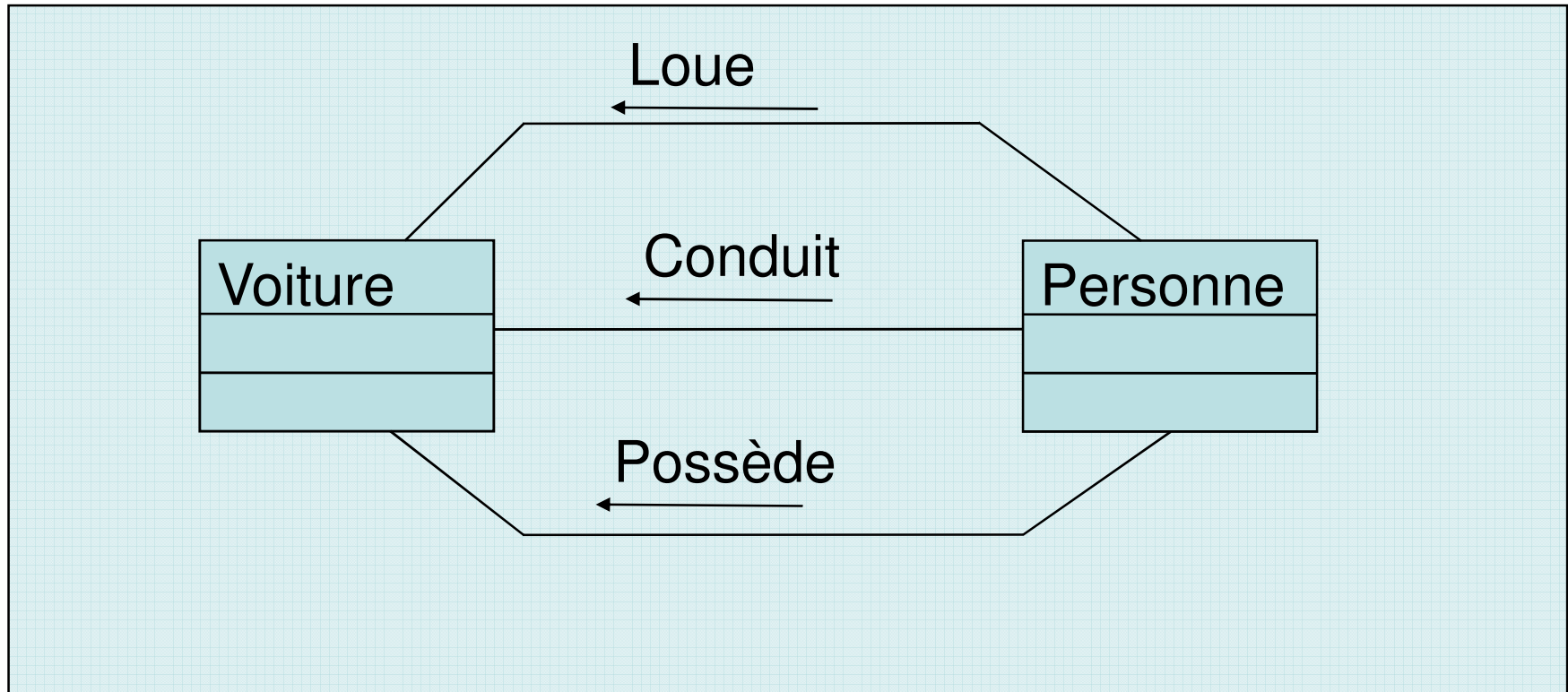
- L'association exprime une **connexion sémantique bidirectionnelle** entre classes.
- Une association est une abstraction des liens qui existent entre les objets instances des classes associées.
- Les associations se représentent de la même manière que les liens.
- Certaines limites aux interactions sont imposées:
  - Limiter le nombre de participants (efficacité maximale).
  - Vérifier la qualification des participants.
  - Définir le rôle de chaque participant.

# Liens entre objets et association entre classes



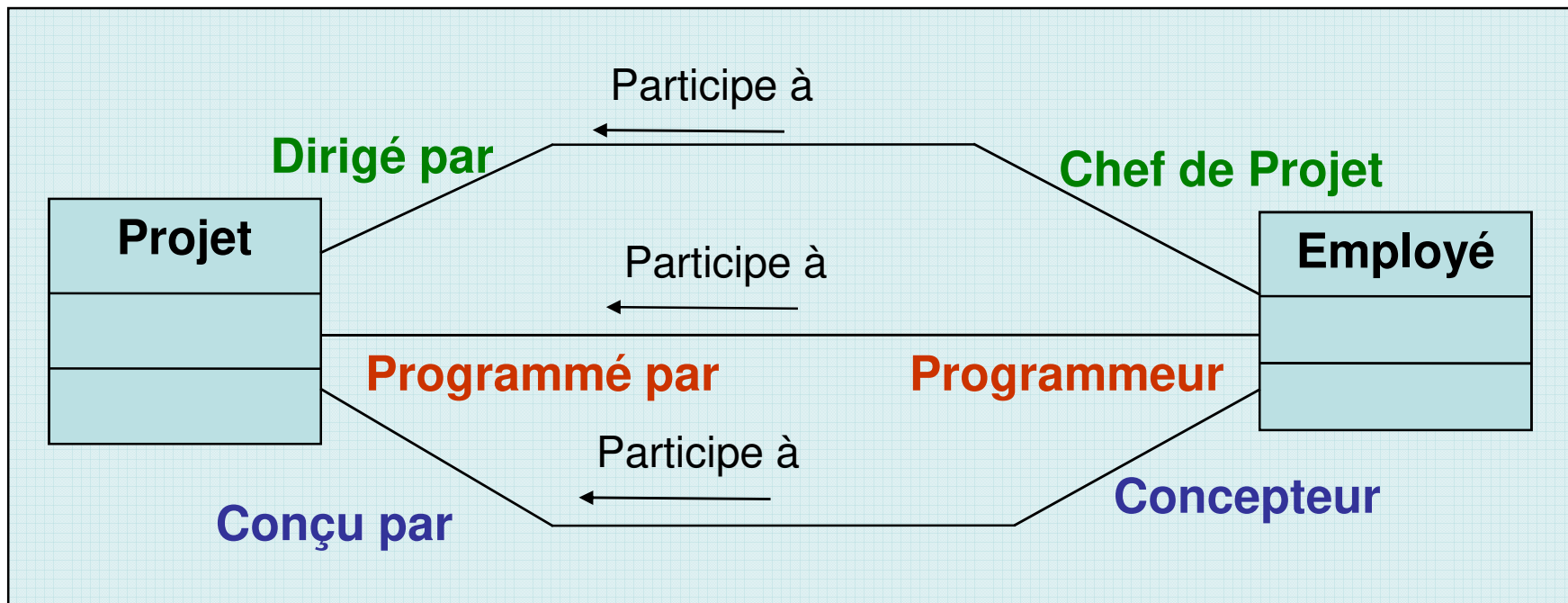


## Exemple :



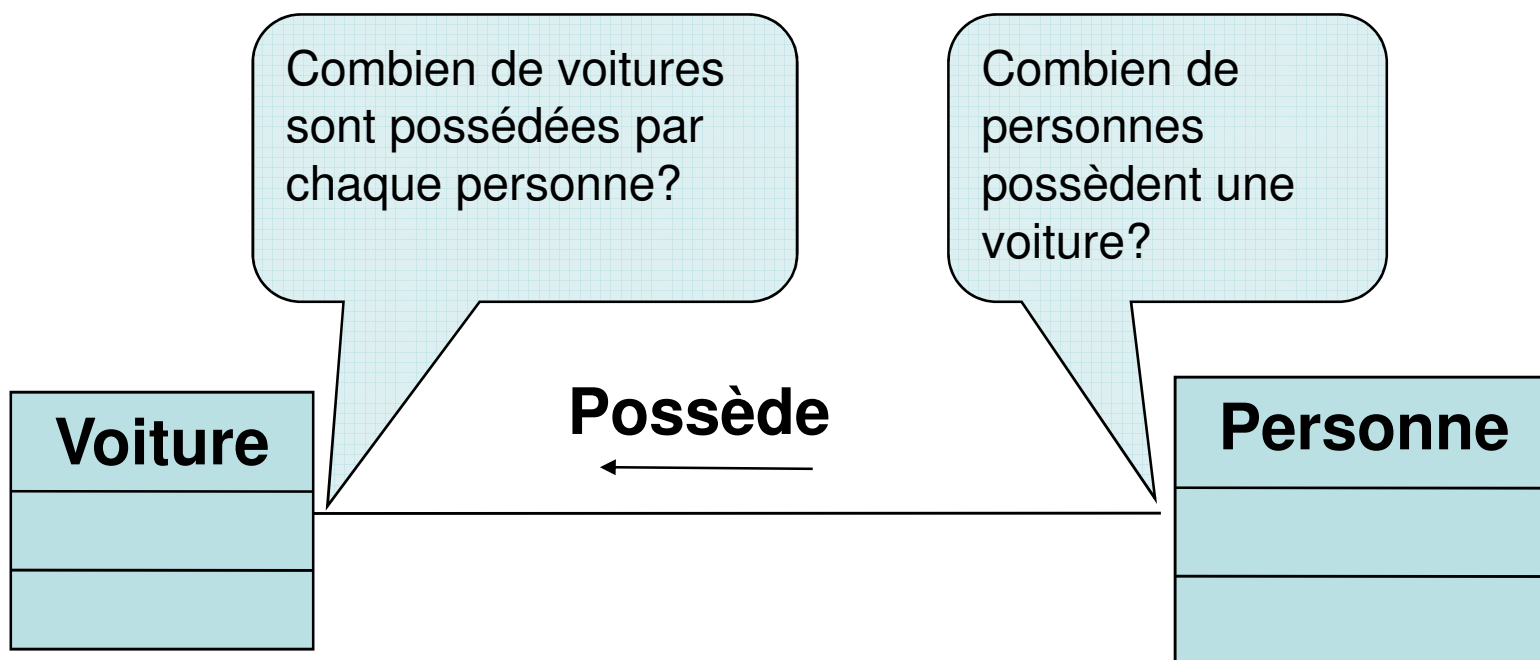
# Rôles

- Il est possible de préciser **le rôle d'une classe au sein d'une association**: un nom de rôle peut être spécifié de part et d'autre de l'association.



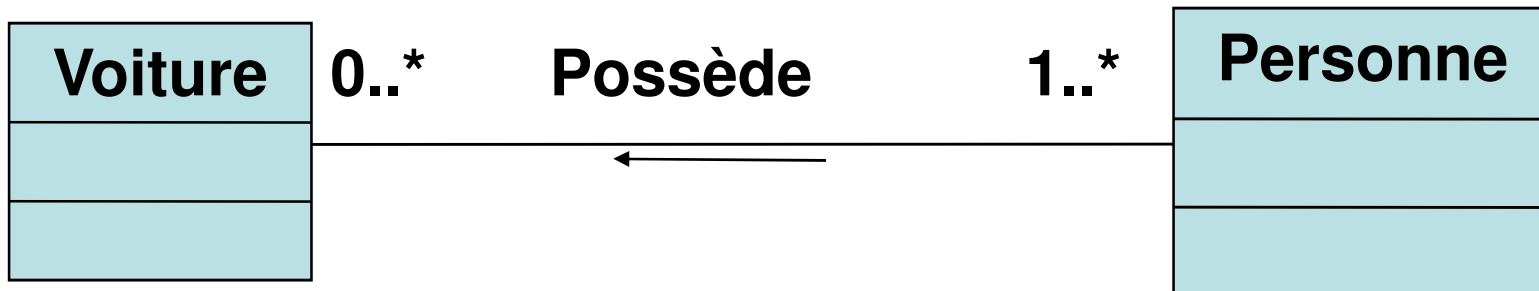
# Multiplicité (1)

- Les rôles portent également une information de multiplicité qui précise le nombre d'instances qui participent à la relation



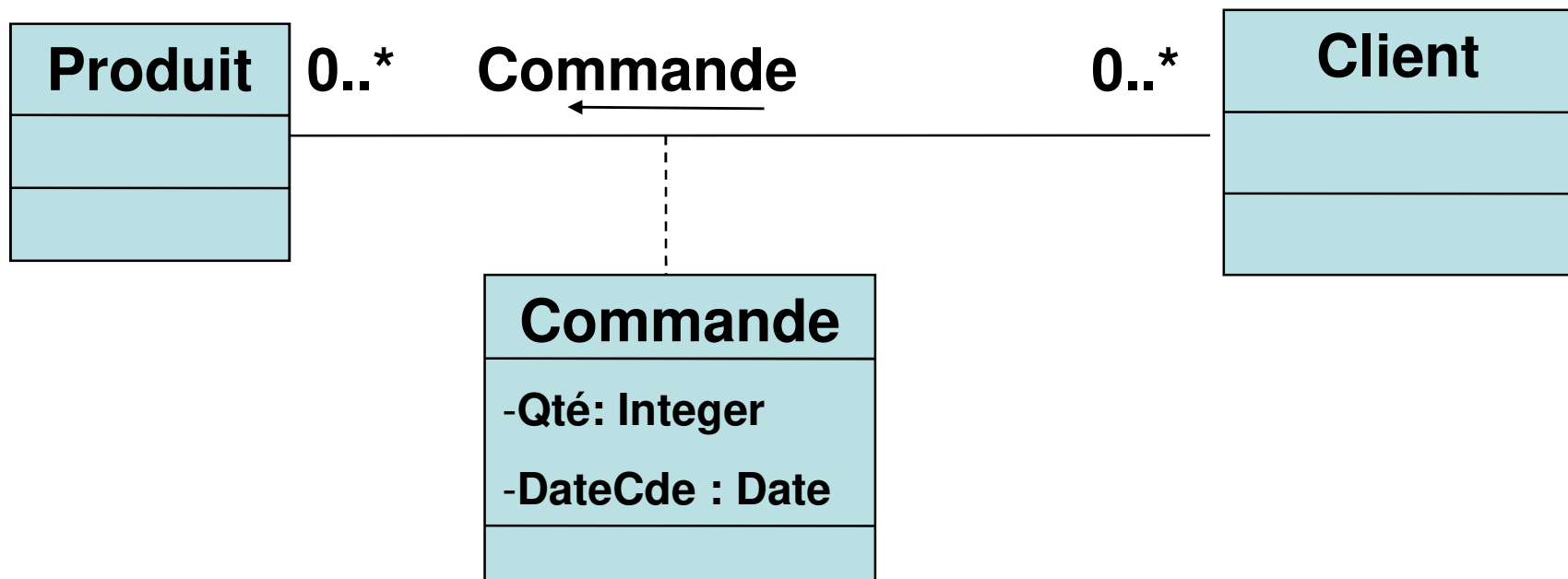
## Multiplicité (2)

1 <b>ou</b> 1..1	Un et un seul
0..1	Zéro ou un
M..N	De M à N (entiers naturels)
* <b>ou</b> 0..*	De zéro à plusieurs
1..*	De un à plusieurs



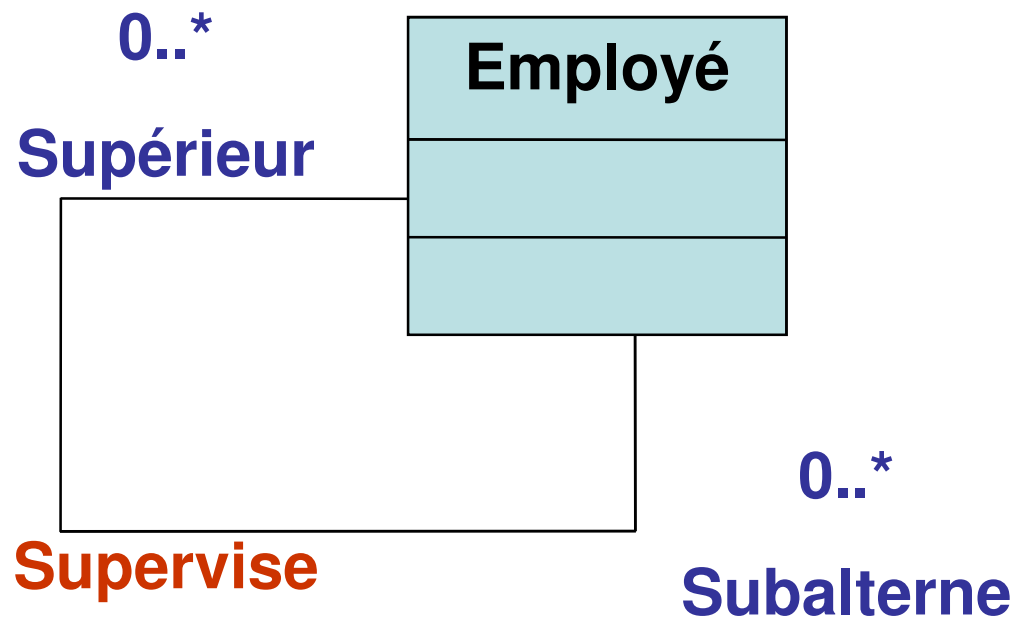
# Les classes d'associations

Une classe d'association encapsule les informations concernant une association.



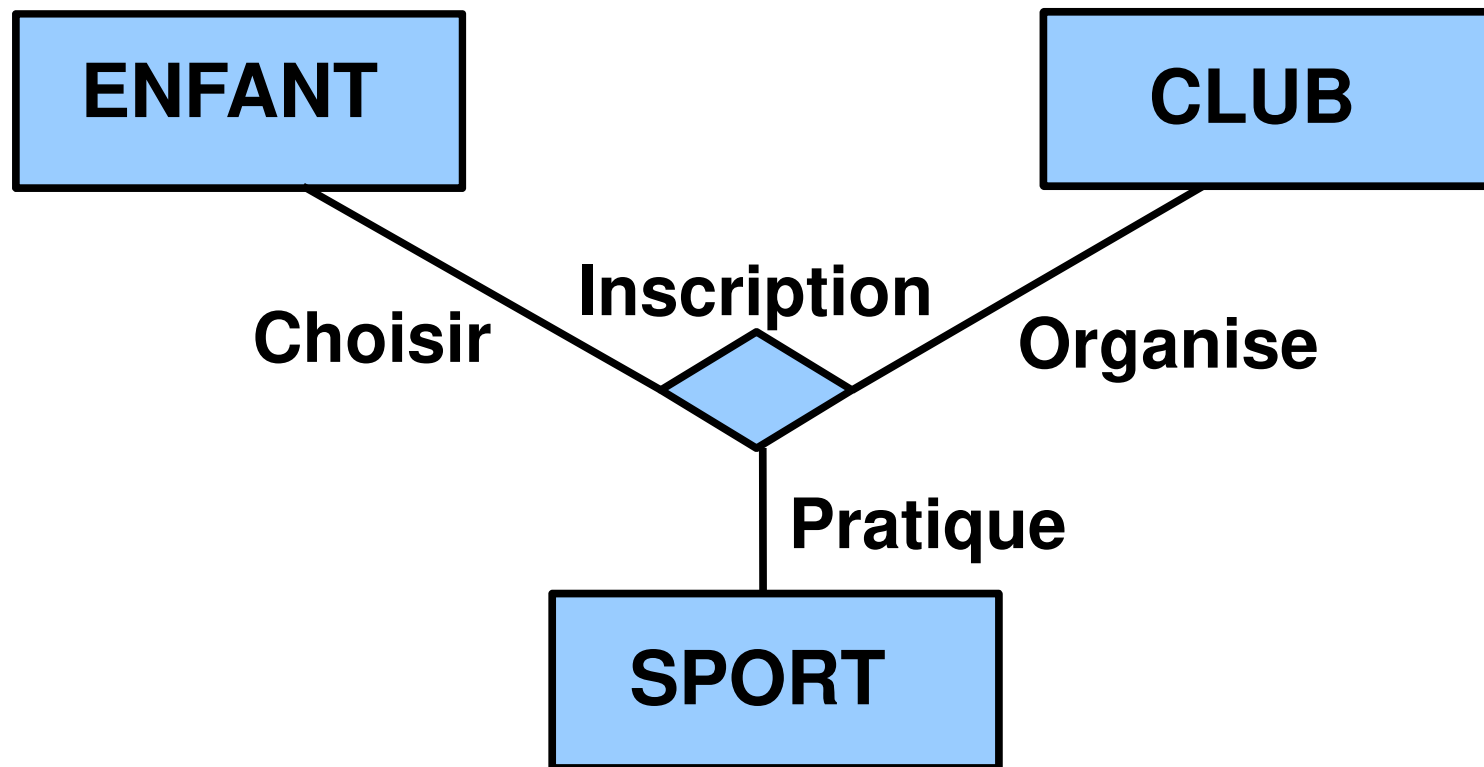
# Les associations réflexives

Une association est réflexive si elle s'applique à des objets d'une même classe.



# Les relations n-aires

- **Les relations n-aires** sont modélisables, On les détecte lorsqu'il est nécessaire d'avoir un identifiant multiple.

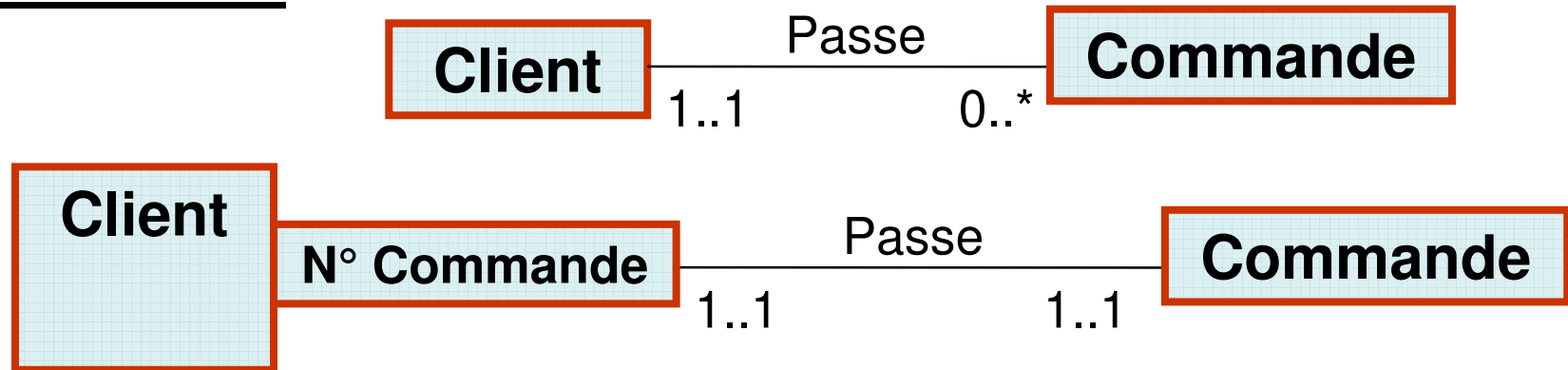


# Qualification d'un lien

- L'objectif est de prévoir un partitionnement d'ensemble qui peut favoriser les performances.
- Il s'agit d'un concept technique.
- Sémantique (Qualificateur):
- C'est un attribut ou un ensemble d'attributs de l'association dont les valeurs partitionnent l'ensemble des objets associés avec un objet à travers une association.
- Le couple (Objet, valeur unique de l'attribut) sélectionne une partition dans l'ensemble des objets de la classe cible associée.
- Les associations qualifiées ont à peu près la même fonction que les index avec une notation différente.



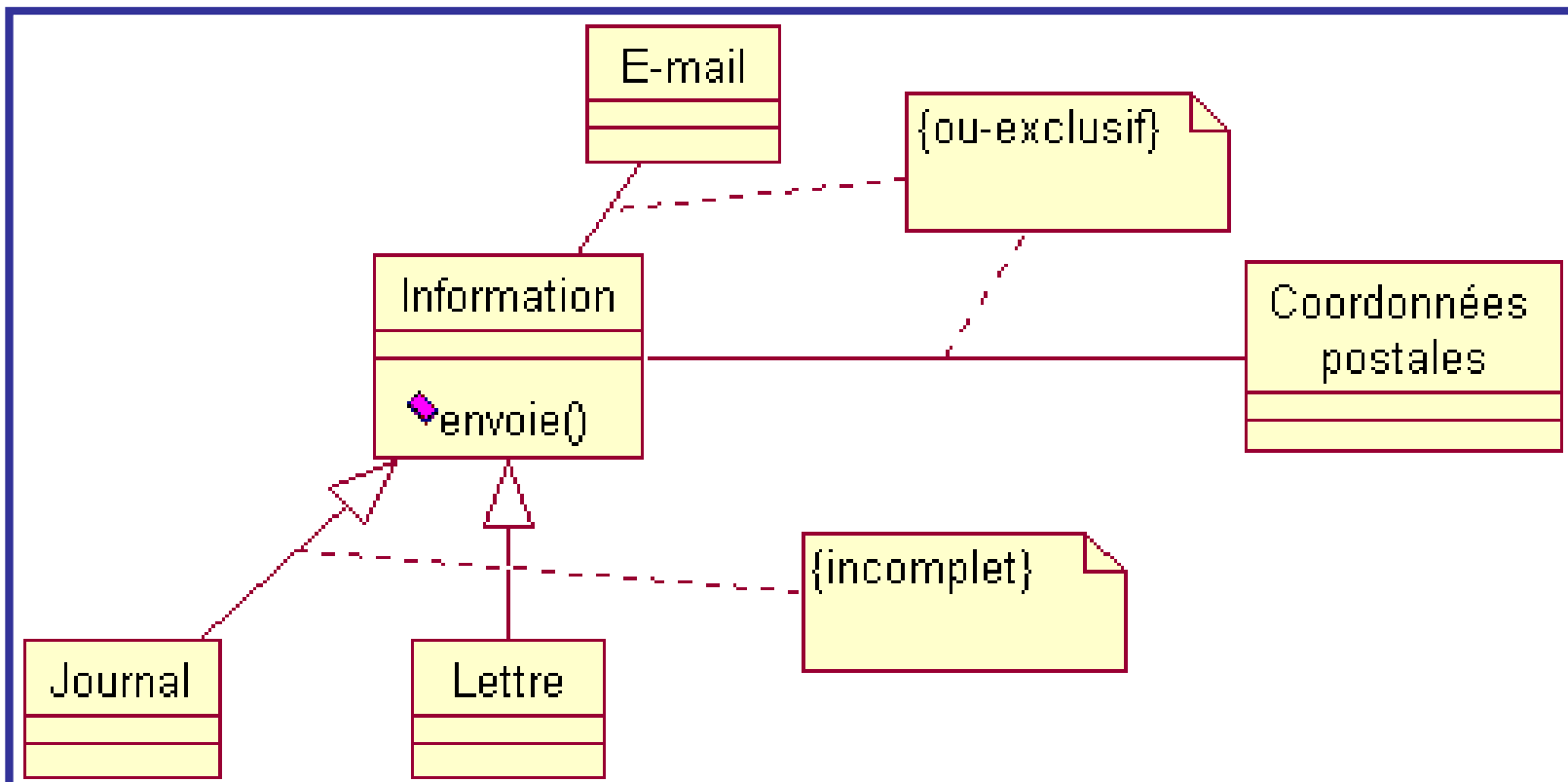
# Formalisme:



- L'Intérêt: Faire apparaître un attribut déterminant non identifiant, qui réduit la multiplicité de l'ensemble d'objets recherchés.
- La multiplicité associée peut-être:
  - 0..1: Une unique valeur est sélectionnée, mais chaque valeur possible du qualificateur ne donne pas nécessairement un objet sélectionné.
  - 1: Chaque valeur possible du qualificateur sélectionne un unique objet cible.
  - \*: Le qualifieur est un index qui partitionne les objets cibles en sous-ensembles.
- Ainsi, dans l'exemple, la multiplicité du côté commande est passée de 0..\* à 1..1 grâce au fait que le qualificateur fournit une clé unique pour les objets Commande.

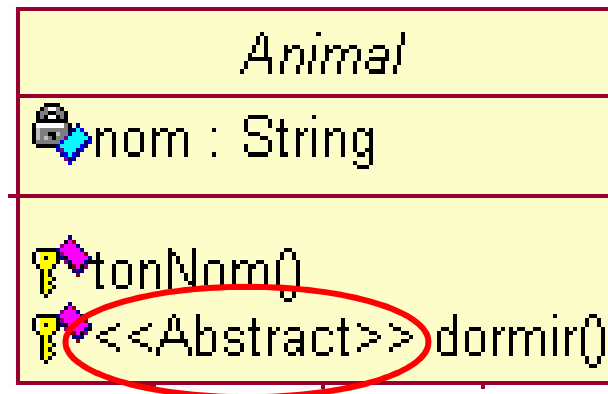
# Les contraintes

- Une contrainte est une relation sémantique quelconque entre éléments de modélisation qui définit **des propositions devant être maintenues à vraies** pour garantir la validité du système modélisé.

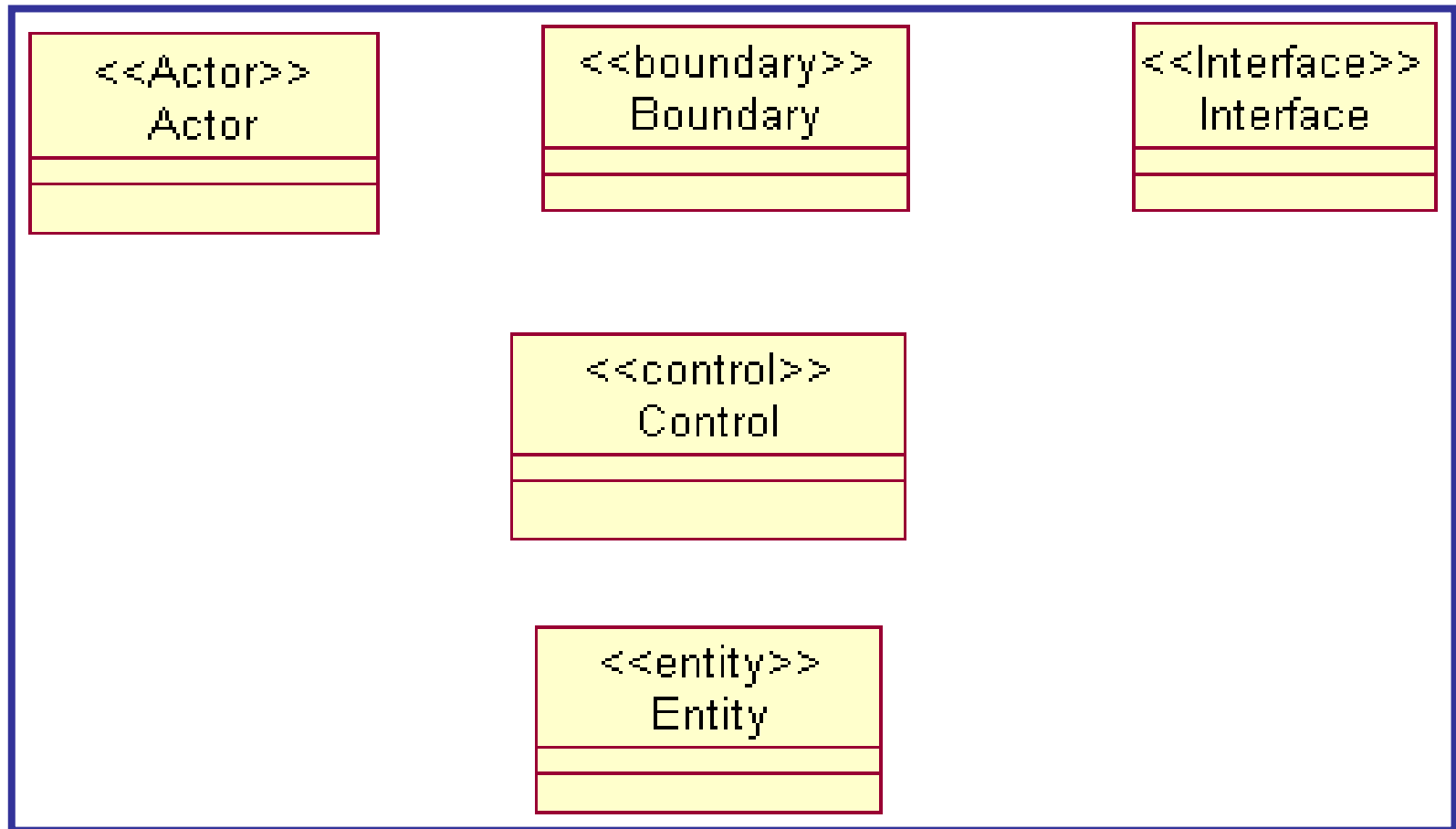


# Les stéréotypes

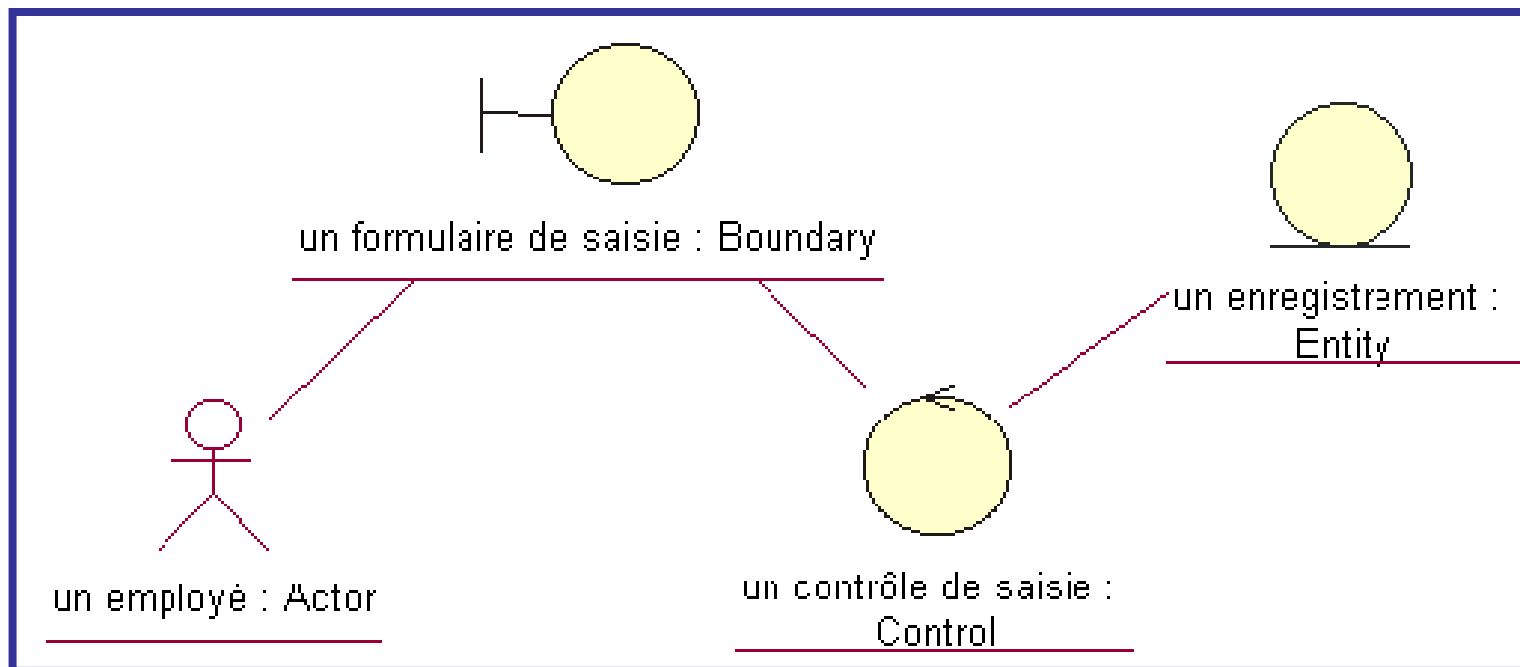
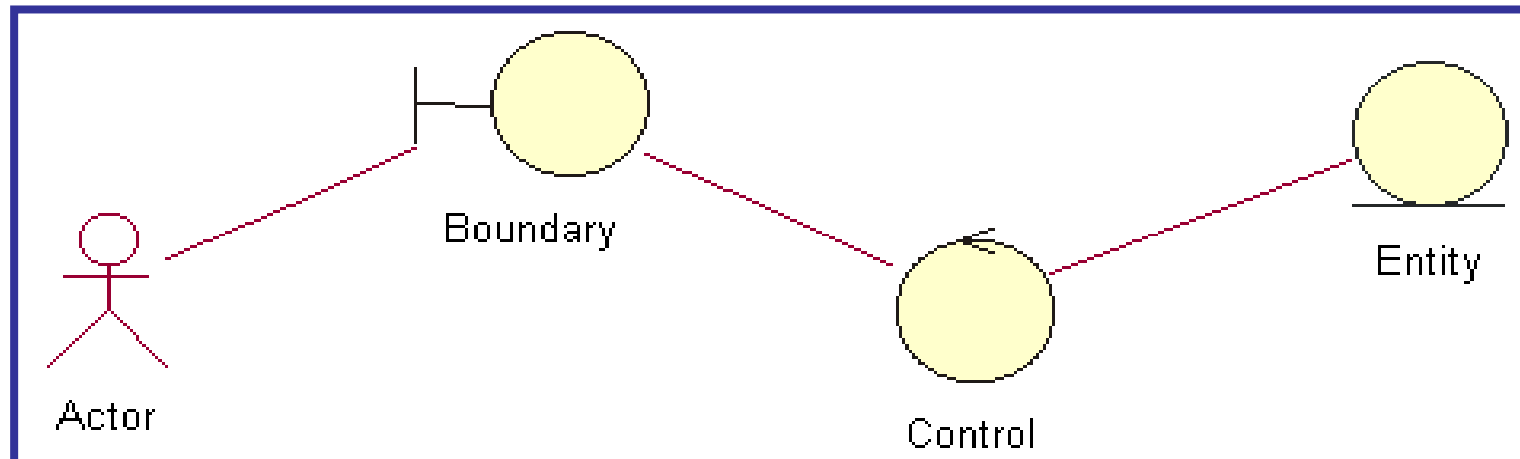
- Les stéréotypes font partie des mécanismes d'extensibilités, prévus par UML.
- Un stéréotype permet la **métaclassification** d'un élément d'UML. Il permet aux utilisateurs (méthodologistes, constructeurs d'outils, analystes et concepteurs) d'ajouter de nouvelles classes d'éléments de modélisation, en plus du noyau prédéfini par UML.



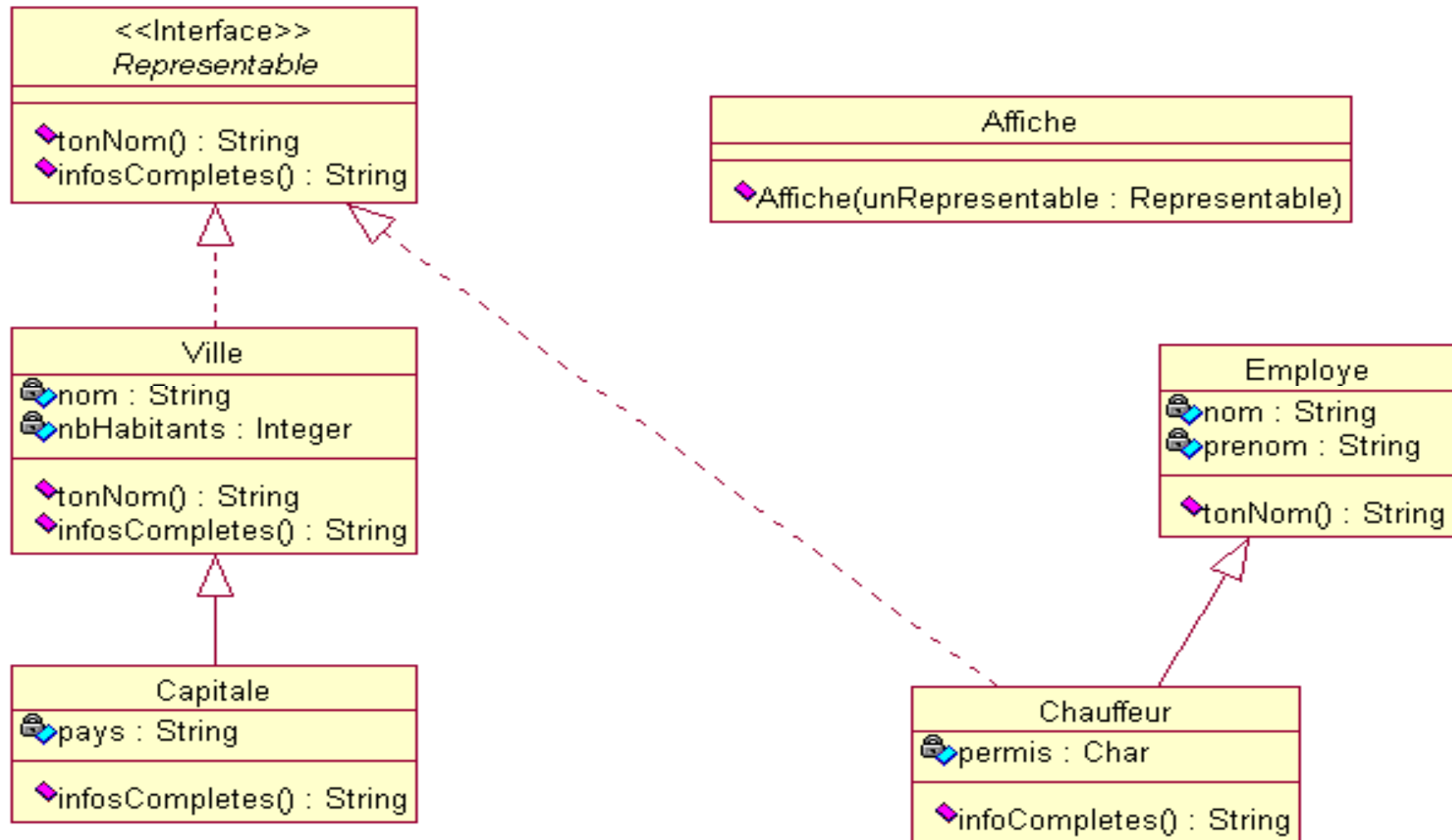
# La classification des classes (1)



# La classification des classes (2)



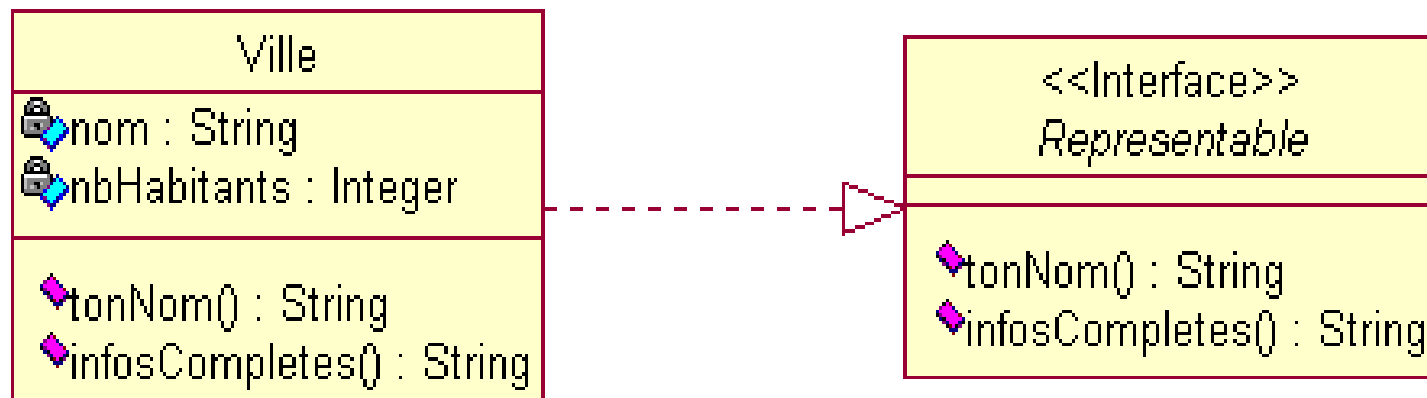
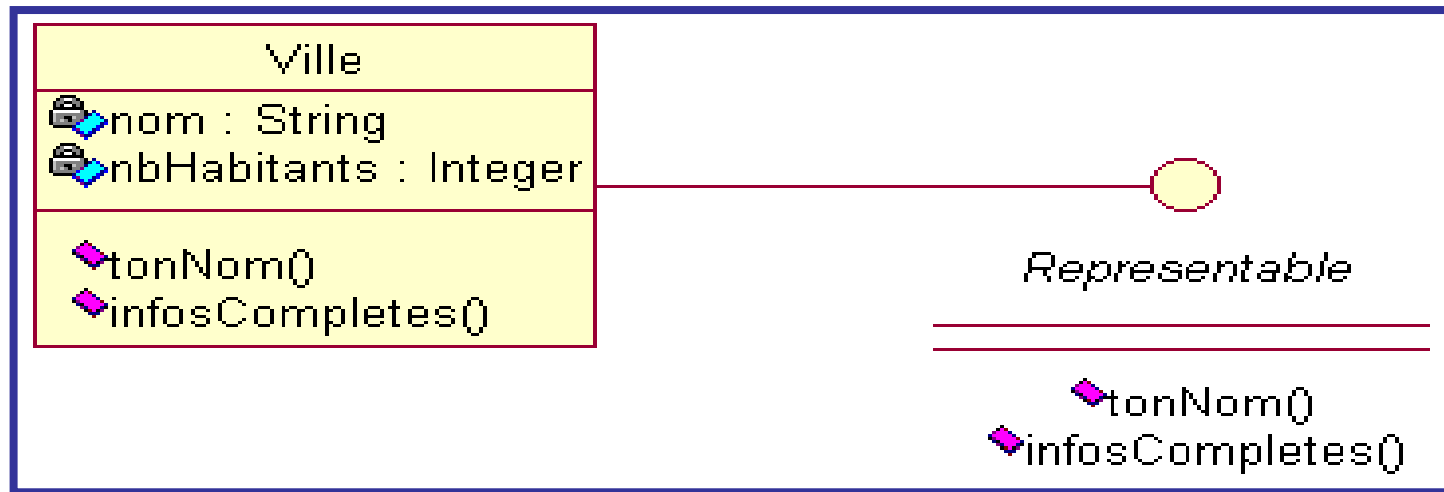
# Les interfaces



# Finalité des interfaces

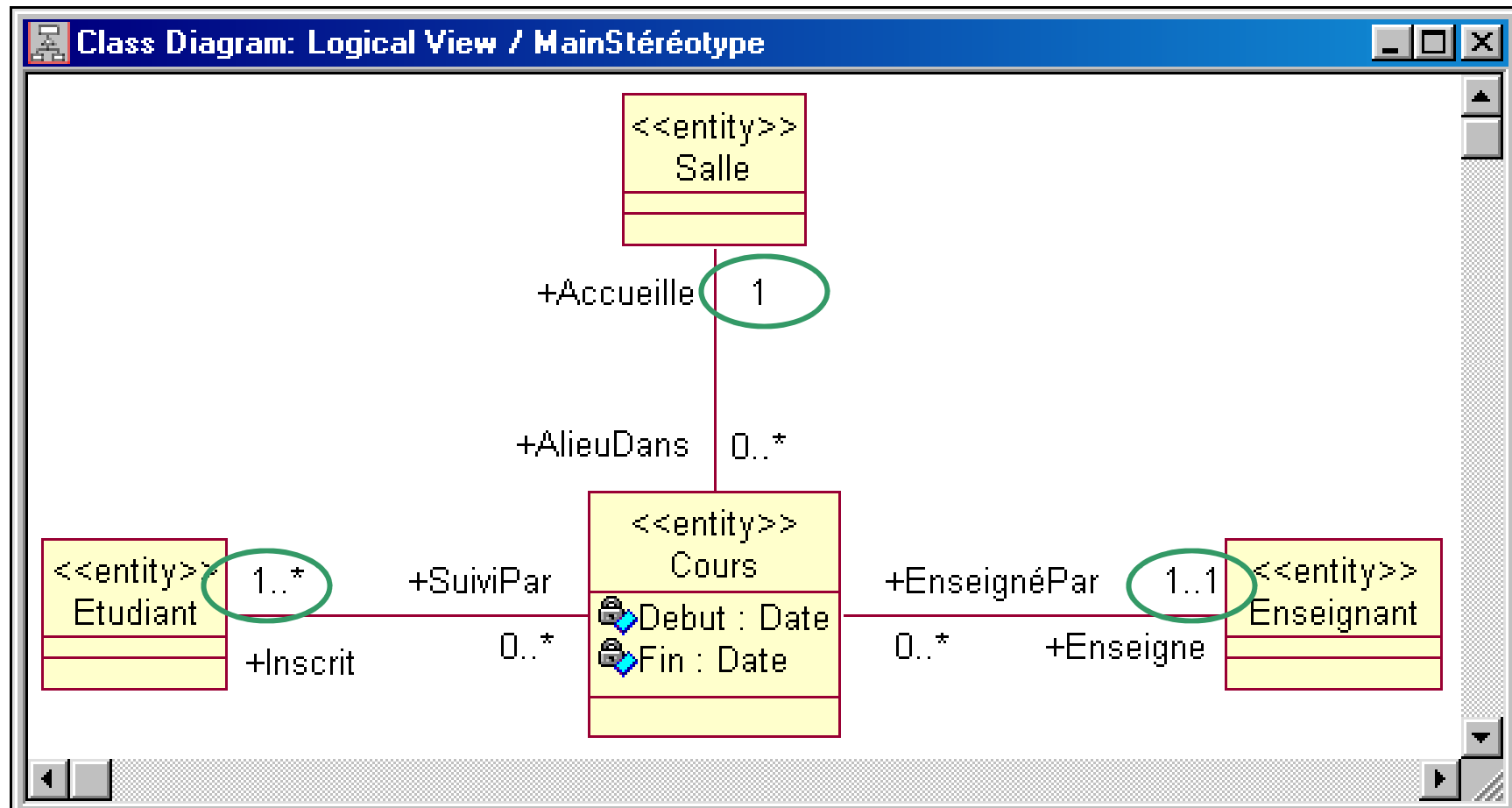
- Une interface décrit le comportement d'une classe, d'un composant, d'un sous-système, d'un paquetage ou de tout autre classificateur
- Une interface possède **uniquement** la spécification d'un comportement visible, sous forme **d'un ensemble d'opérations** (pas d'attributs et d'associations), et ne fournit aucune implémentation de ses services.

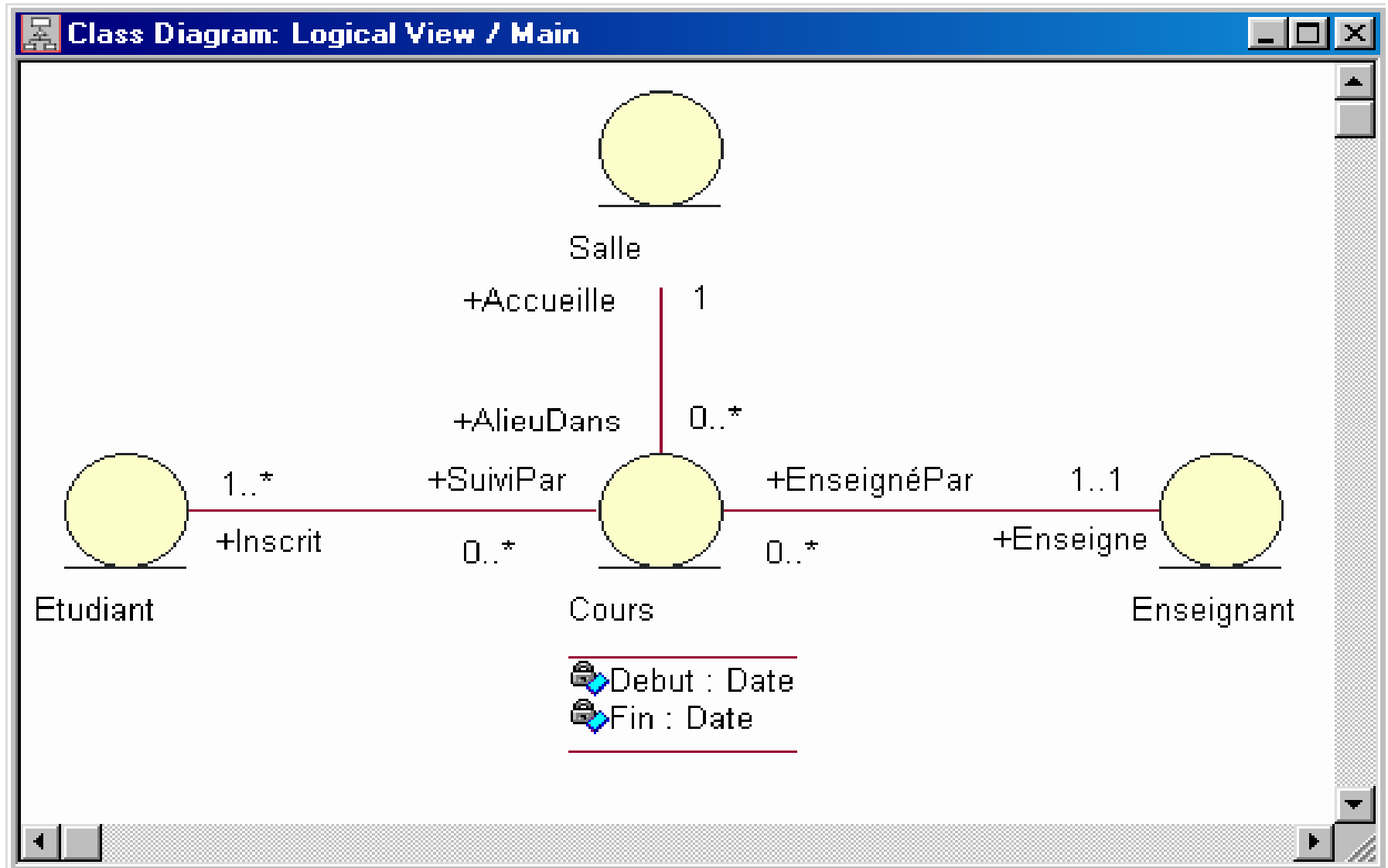
# Finalité et réalisation des interfaces



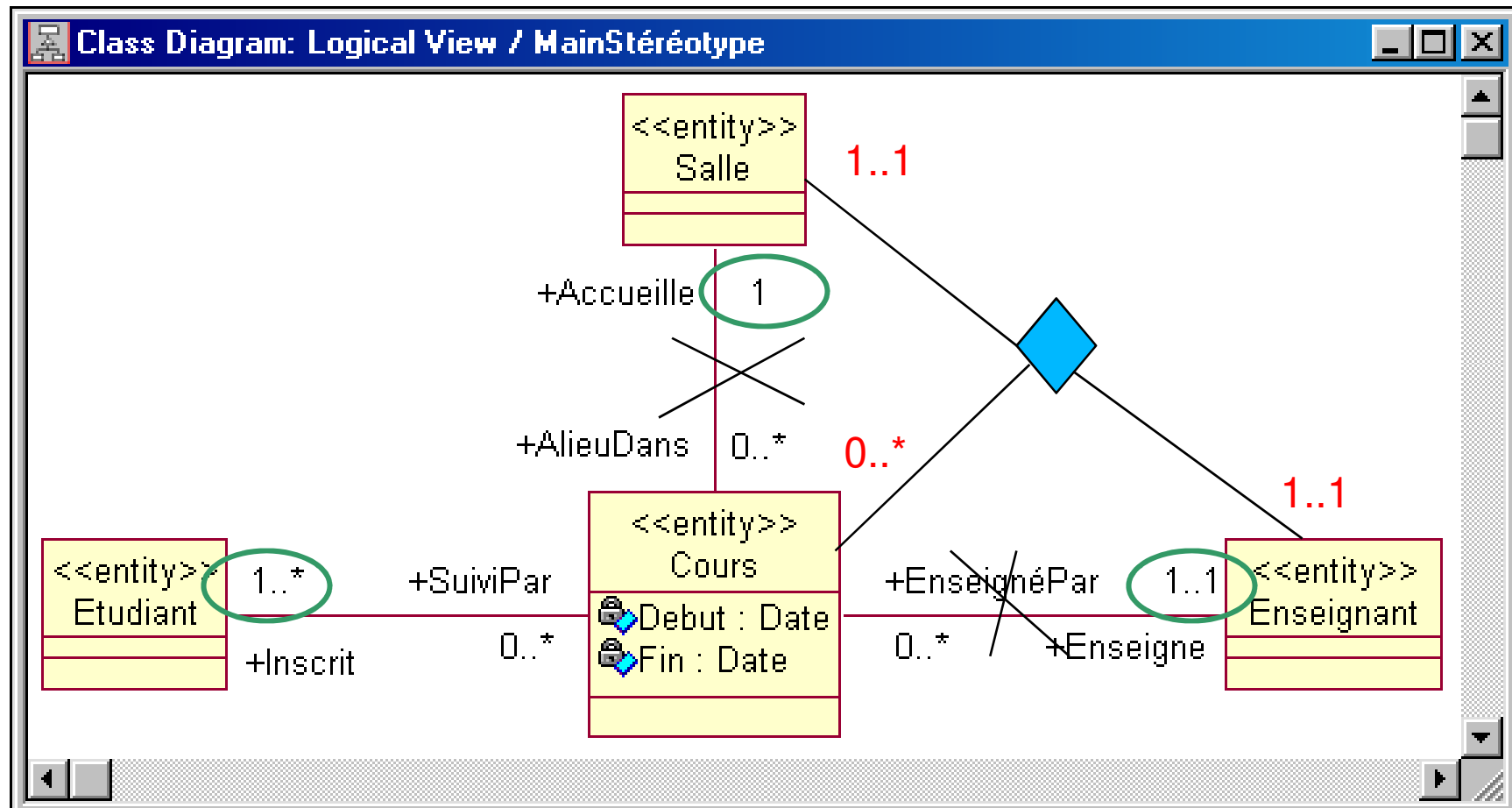


# Exemple :





# Exemple : relation ternaire



# L'agrégation

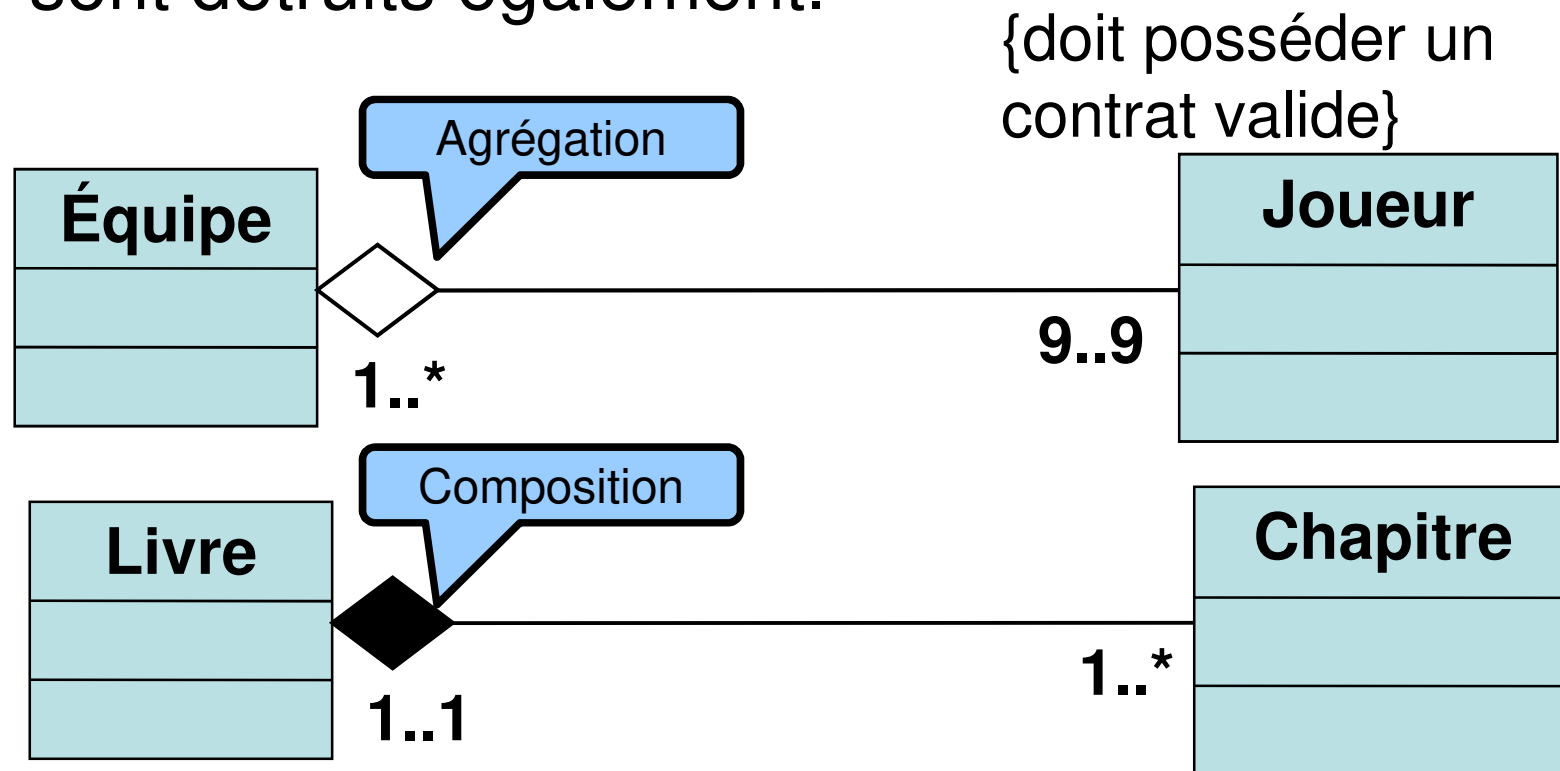
- Une relation exprime une forme de couplage entre abstractions. La force de ce couplage dépend de la nature de la relation dans le domaine du problème.
- Par défaut, l'association représente un couplage faible, les classes associées restant relativement indépendantes l'une de l'autre.
- L'agrégation est une type particulier d'association qui exprime un couplage plus fort entre classes. Une des classes joue un rôle plus important que l'autre dans la relation.
- L'agrégation permet de représenter des relations de type maître et esclaves, tout et parties ou composés et composants.

# La composition

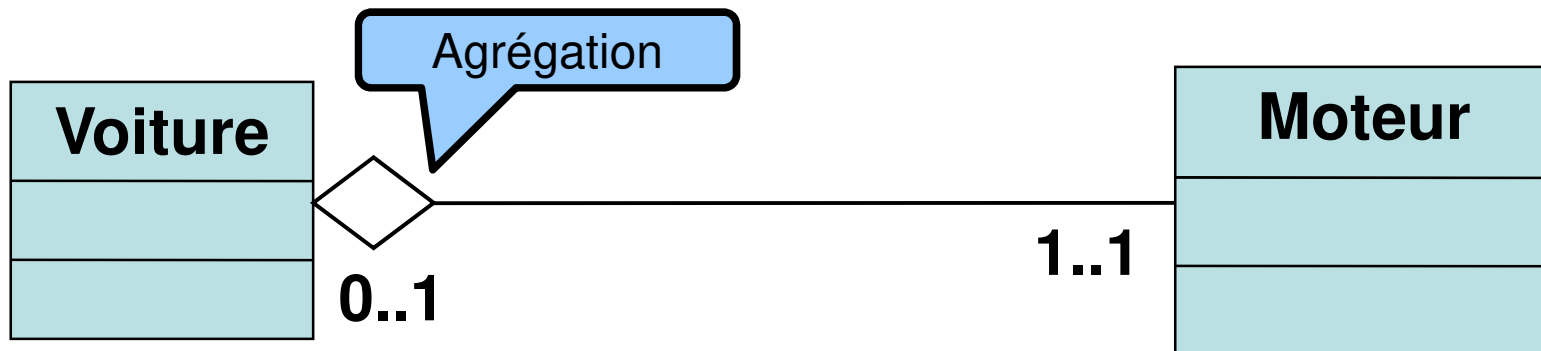
- La composition est une forme d'agrégation avec un couplage plus important.
- Ce couplage de composition indique que **les composants ne sont pas partageables** et que **la destruction de l'agrégat entraîne la destruction des composants** agrégés.
- La valeur maximale de multiplicité du côté du conteneur ne doit pas excéder 1 puisque les objets, instances de la classe des composants, doivent tous appartenir au même objet conteneur.

## Exemples :

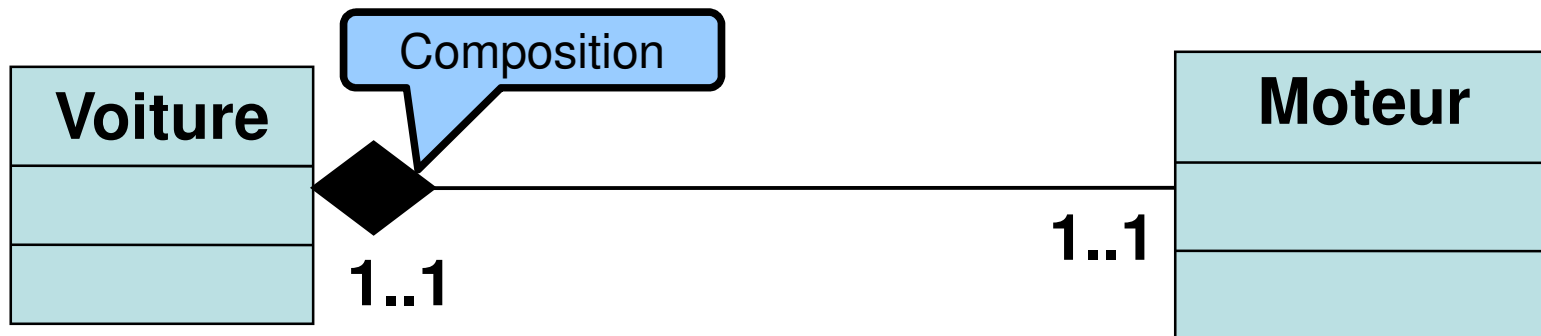
- Si une équipe est dissoute, les membres continuent d'avoir une existence propre (Agrégation).
- En revanche, si un livre est détruit, ses chapitres sont détruits également.



**Atelier de mécanique (désassemblage) → agrégation**



**Agence de vente de voiture → Composition**



# Agrégation et composition

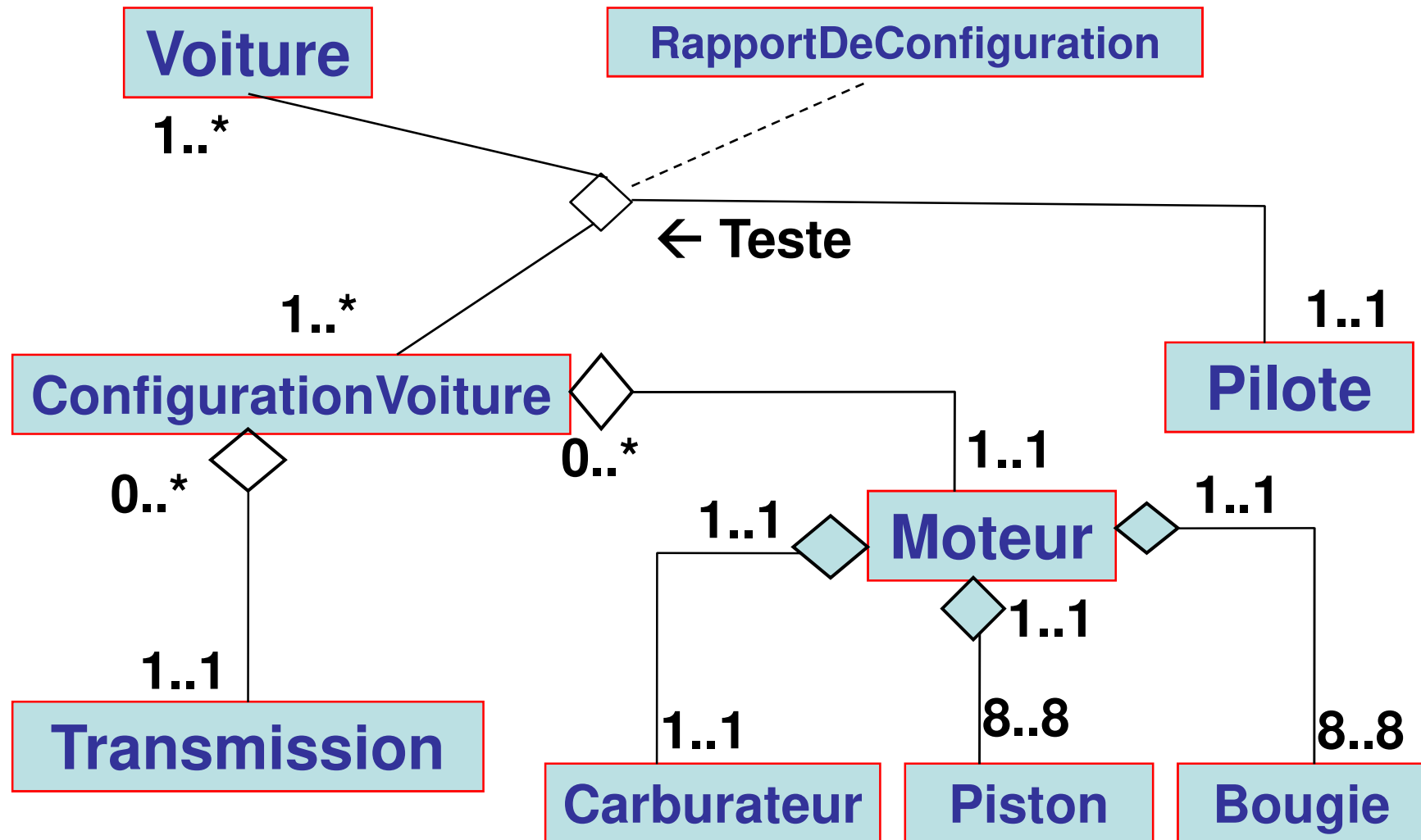
- Une **agrégation se fait par référence** du ou des « enfant(s) » dans le contexte du « parent »
  - By reference
- Une **composition se fait par valeur**, c'est-à-dire copie du ou des « enfant(s) » dans le contexte du « parent »
  - By value



## Exercice: Voiture de course

- Une entreprise teste plusieurs Voitures de courses utilisant un nouveau moteur à 8 cylindres et une nouvelle transmission.
- Une fois les moteurs assemblés, les pistons, les carburateurs et les bougies ne peuvent plus être échangés entre les moteurs en raison des modifications causées par la haute température.
- Nous voulons enregistrer les performances effectuées par chaque voiture, ainsi que les combinaisons de moteurs et de transmissions utilisées dans chaque voiture.
- Les pilotes essaient chaque voiture et font un compte rendu de leurs performances.
- Le système doit enregistrer les configurations et les comptes rendus.

# Exemple: Voiture de course



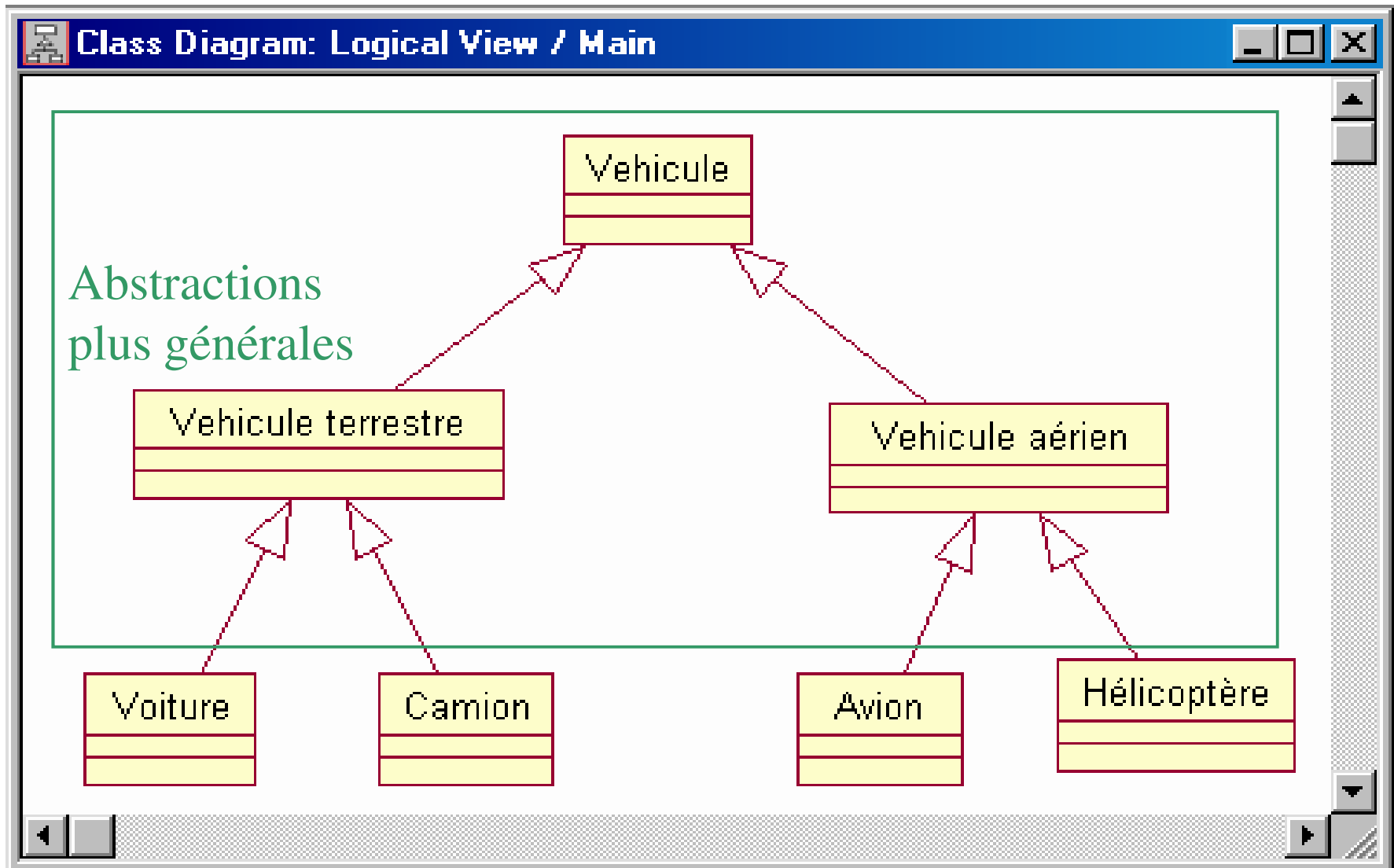
# Les hiérarchies de classes

- Les hiérarchies de classes ou classifications permettent de gérer la complexité en ordonnant les objets au sein **d'arborescences de classes d'abstraction croissante**.
- La généralisation: il s'agit de prendre des classes existantes (déjà mises en évidence) et de créer de nouvelles classes qui regroupent leurs parties communes; **il faut aller du plus spécifique vers le plus général**.
- La spécialisation: il s'agit de sélectionner des classes existantes (déjà identifiées) et d'en dériver de nouvelles classes plus spécialisées, en **spécifiant simplement les différences**.

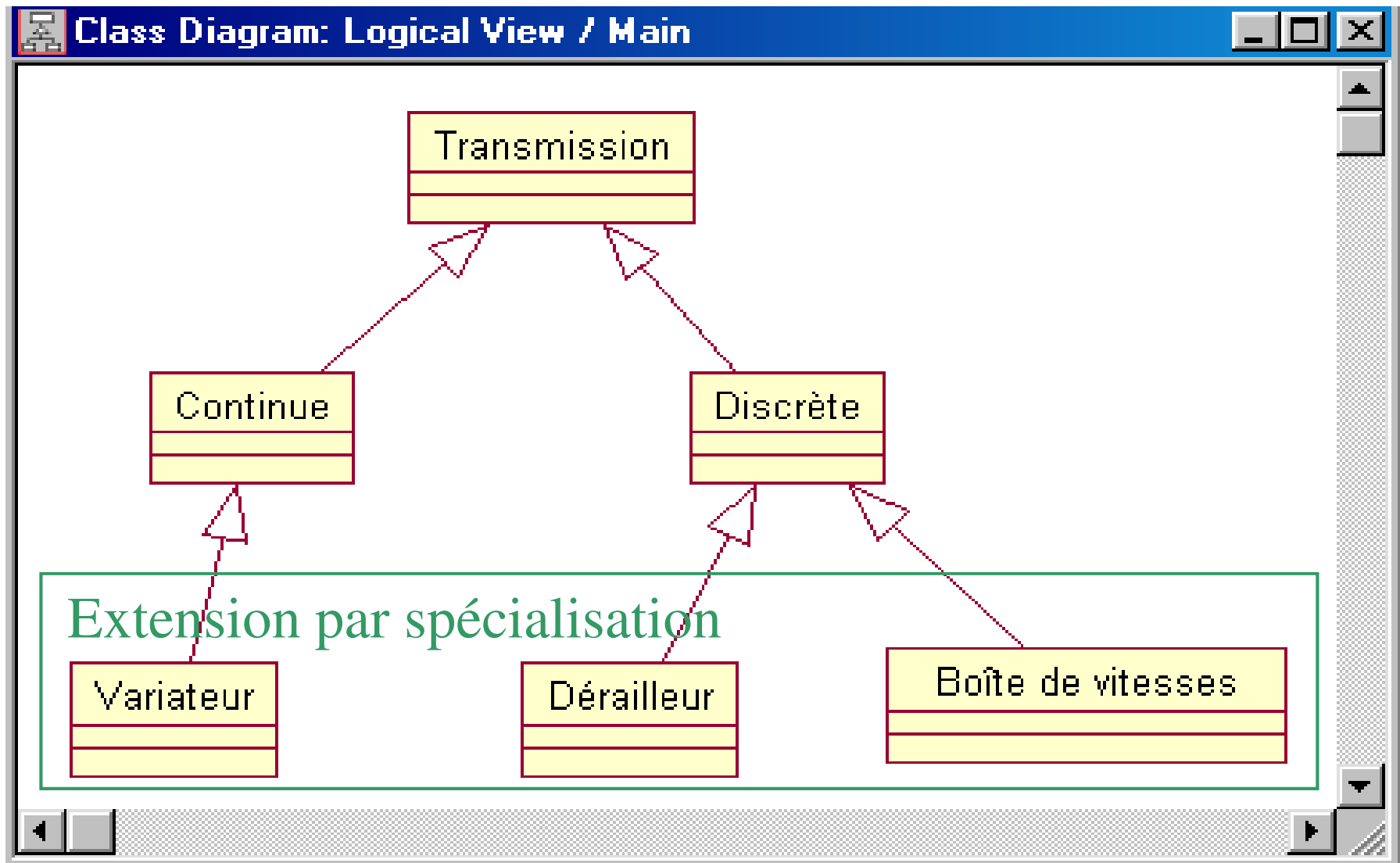
# Généralisation

- La généralisation consiste à factoriser les éléments communs d'un ensemble de classes dans une classe plus générale appelée super-classe. Les classes sont ordonnées selon une hiérarchie; **une super-classe est une abstraction de ses sous-classes.**
- La généralisation est une démarche assez difficile car elle demande une bonne capacité d'abstraction. La mise au point d'une hiérarchie est délicate et **itérative.**
- Les arbres de classes ne poussent pas à partir de leur racine. Au contraire, ils se déterminent en partant des feuilles car les feuilles appartiennent au monde réel alors que les niveaux supérieurs sont des **abstractions construites pour ordonner et comprendre.**

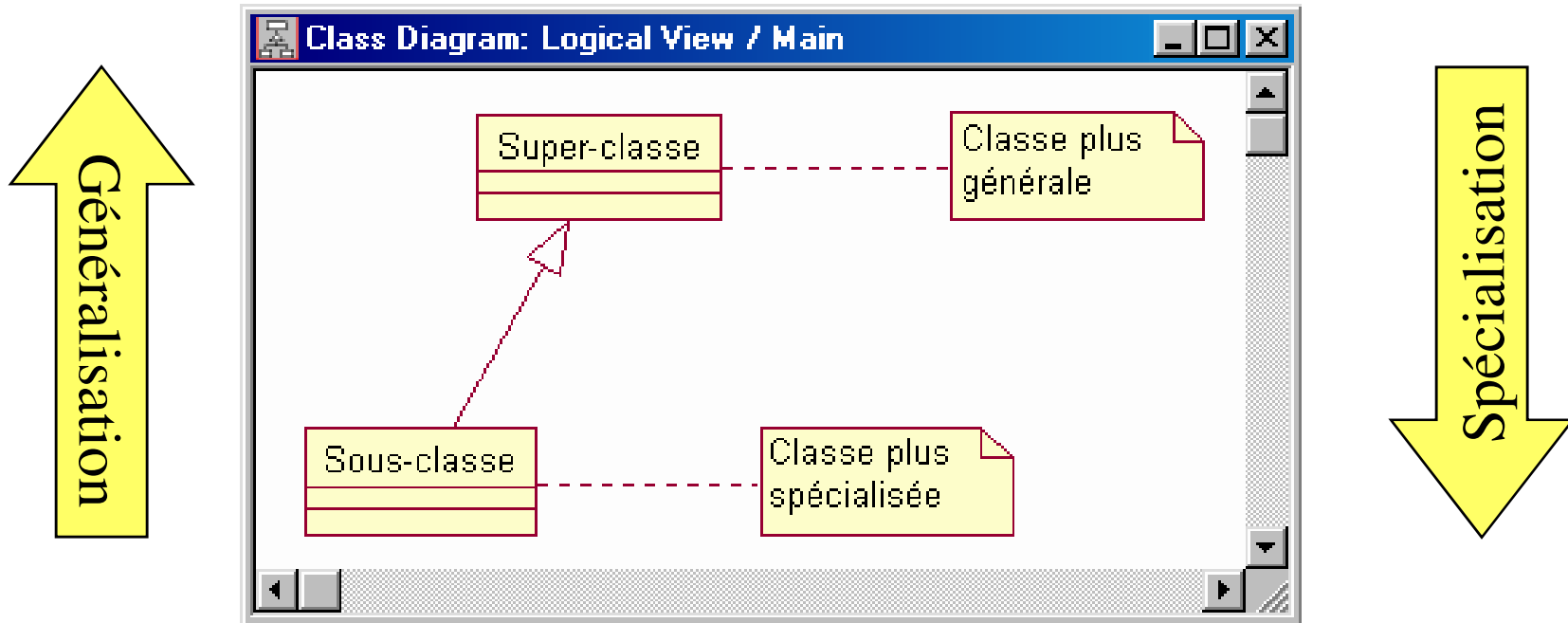
# Exemple de hiérarchie de classes



# Spécialisation



# Règles de généralisation (1)



- La généralisation ne porte aucun nom particulier; elle signifie toujours: **est un** ou **est une sorte de**.
- La généralisation ne concerne que les classes, elle n'est pas instantiable en liens et, de fait, ne porte aucune indication de multiplicité.

# Règles de généralisation (2)

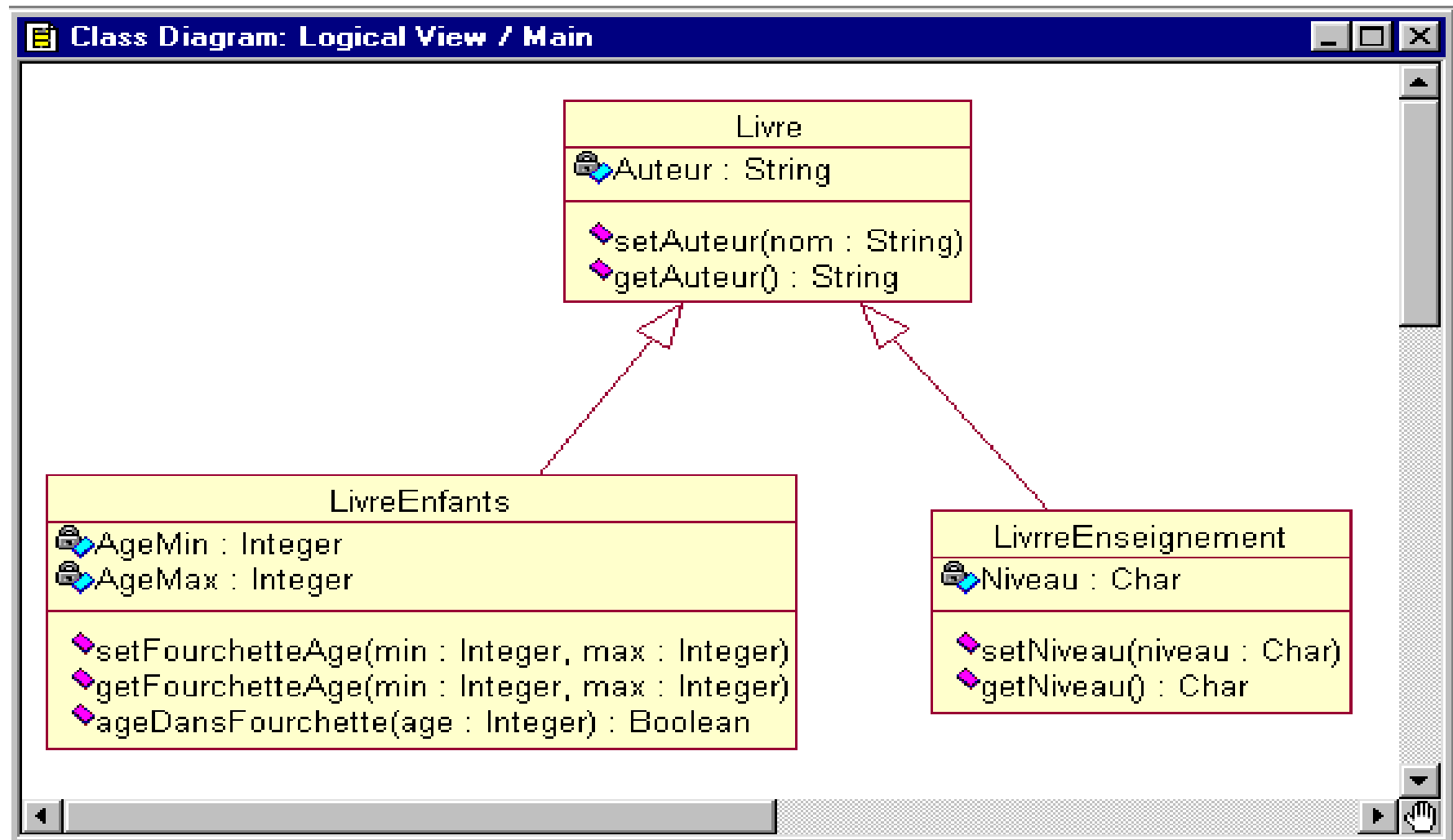
- La généralisation ne porte **ni nom particulier ni valeur de multiplicité**.
- La généralisation est une **relation non réflexive**: une classe ne peut pas dériver d'elle-même.
- La généralisation est une **relation non symétrique**: si une classe B dérive d'une classe A, alors la classe A ne peut pas dériver de la classe B.
- La généralisation est par contre une **relation transitive**: si C dérive d'une classe B qui dérive elle-même d'une classe A, alors C dérive également de A.



# Des ensembles aux classes

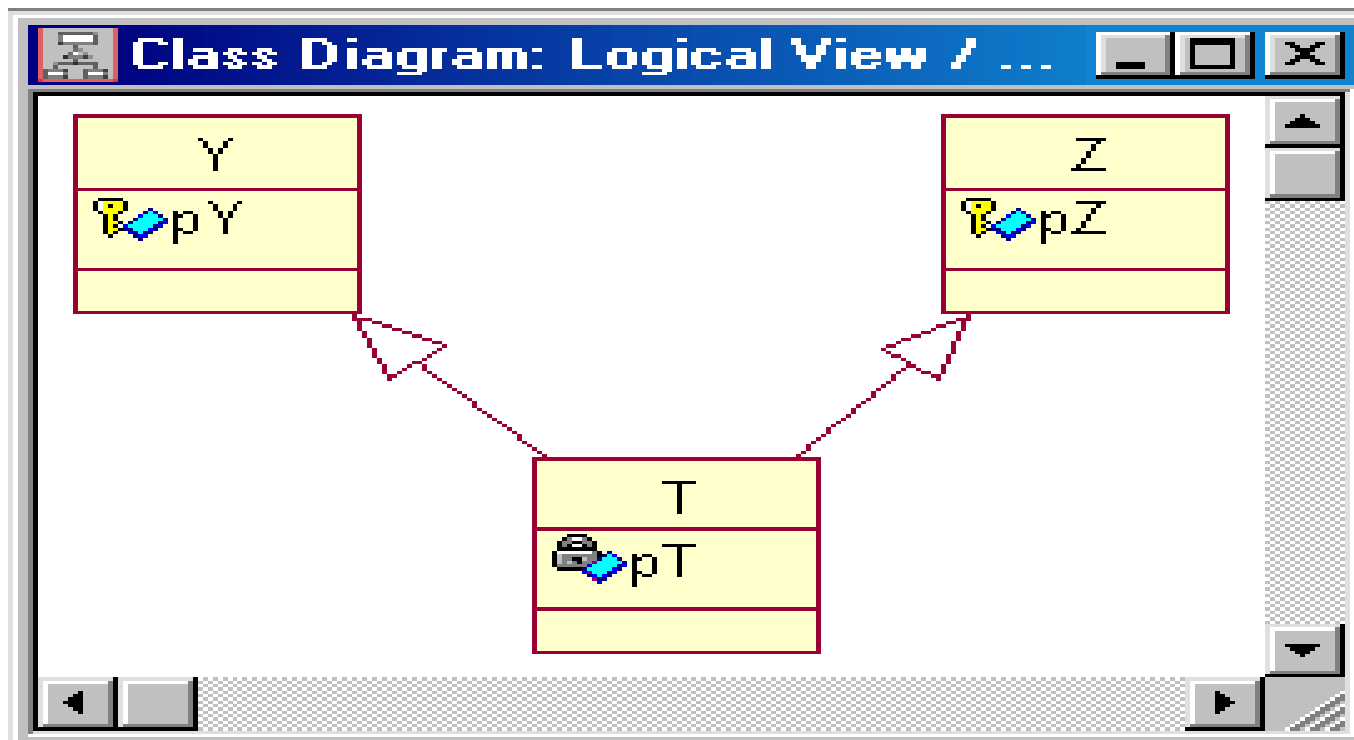
- Les classes et les sous-classes sont l'équivalent des ensembles et des sous-ensembles.
- La notion de **généralisation correspond à la relation d'inclusion des ensembles**.
- De ce fait, les objets instances d'une classe donnée sont décrits par la propriété caractéristique de leur classe, mais également par les propriétés caractéristiques de toutes les classes parents de leur classe.
- Les sous-classes ne peuvent pas nier les propriétés caractéristiques de leurs classes parentes.

## Exemple : attributs et méthodes hérités.



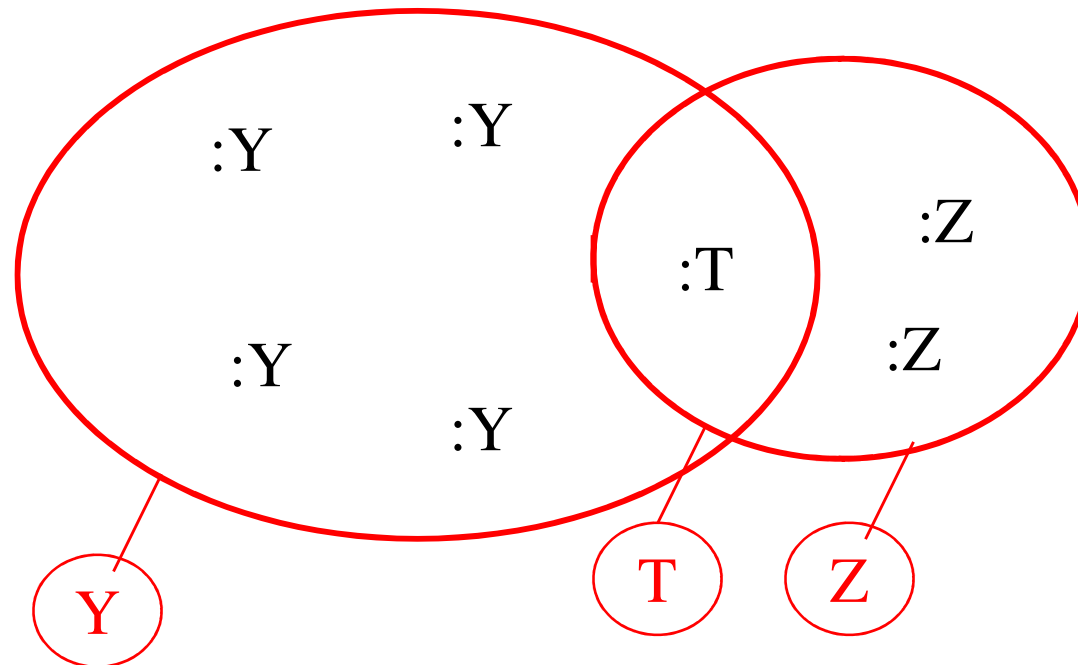
# Généralisation multiple (1)

- *La généralisation - sous sa forme dite multiple – existe également entre arbres de classes disjoints.*



# Généralisation multiple (2)

Dans le monde des ensembles la généralisation multiple correspond à l'intersection de deux ensembles qui ne sont pas sous-ensembles du même sur-ensemble

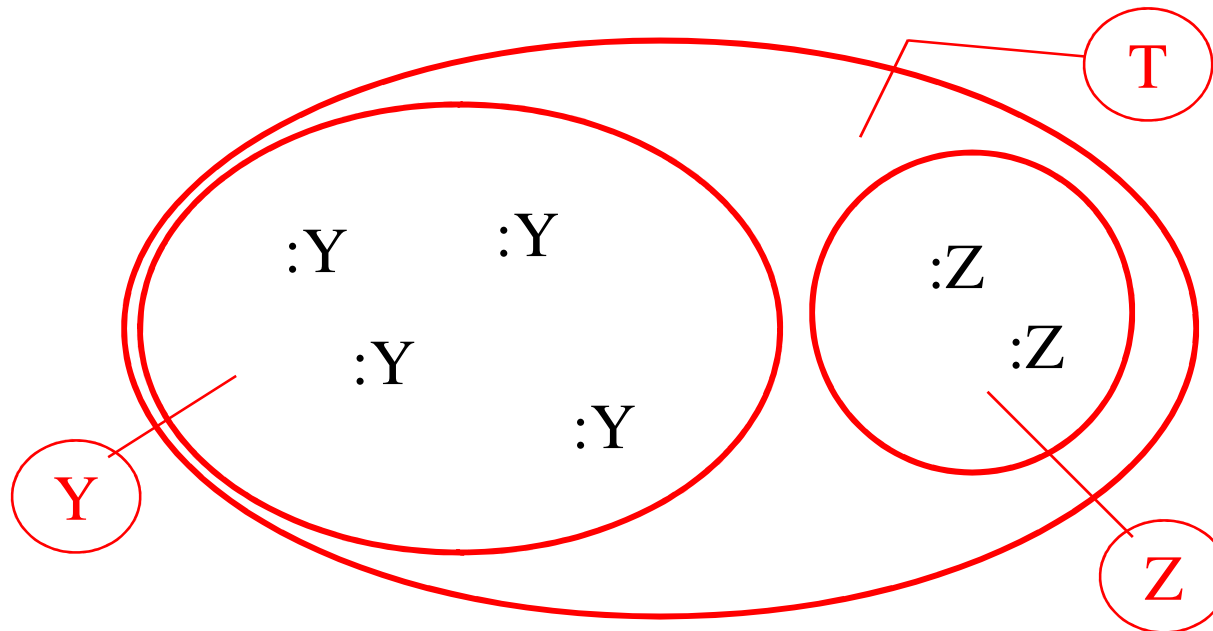


## Généralisation multiple (3)

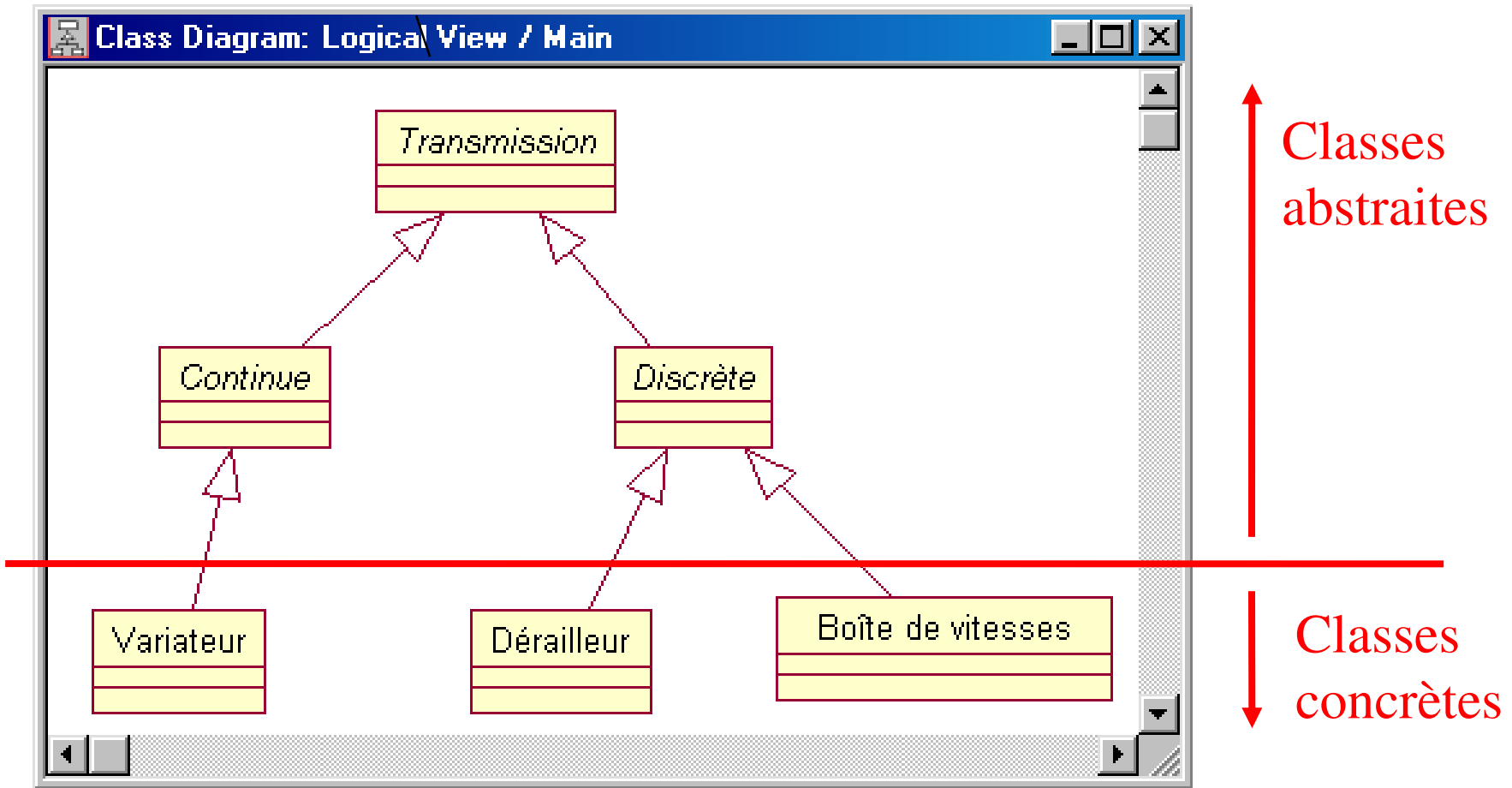
- Pour que la généralisation multiple puisse être mise en œuvre, il faut que les langages de programmation « objets » supportent l'héritage multiple.
- Dans notre exemple, comment le compilateur peut-il garantir, lors de l'implémentation de la classe T, qu'il n'y ait pas d'effet de bord ou de conflit entre les propriétés pZ héritée de la classe Z et pY héritée de la classe Y?
- Par exemple, JAVA ne supporte pas l'héritage multiple.

# Classe abstraite

- Une classe abstraite ne donne pas directement des objets. Elle sert en fait de spécification plus abstraite pour des objets instances de ses sous-classes.
- Le principal intérêt de cette démarche est de réduire le niveau de détails dans les descriptions des sous-classes.
- Le nom d'une classe abstraite est en italique dans les diagrammes de classes.



# Exemple de classes abstraites



# De la difficulté de classer

- *Les classifications doivent avant tout être **stables et extensibles**.*
- *Il n'y a pas une classification, mais des classifications, chacune adaptée à un **usage donné**.*
- *Les classifications doivent comporter des **niveaux d'abstraction équilibrés**.*
- *Le critère de génération d'une classe doit être **statique**.*



# L'héritage

- L'héritage est une technique offerte par les langages de programmation pour **construire une classe à partir d'une ou de plusieurs autres classes**, en partageant des attributs, des opérations et parfois des contraintes au sein d'une hiérarchie de classes.
- Les classes enfants héritent des caractéristiques de leurs classes parents; les attributs et les opérations déclarés dans la classe parent, sont accessibles dans la classe enfant, comme s'ils avaient été déclarés localement.

# L'héritage multiple

- L'héritage n'effectue pas une union des propriétés caractéristiques des classes, mais plutôt une somme de ces propriétés.
- Ainsi, certaines caractéristiques peuvent être indûment dupliquées dans les sous-classes.
- L'héritage multiple doit donc être manié avec beaucoup de précautions car les techniques de réalisation de l'héritage peuvent induire des problèmes de collisions de noms lors de la propagation des attributs et des opérations des classes parents vers les sous-classes.

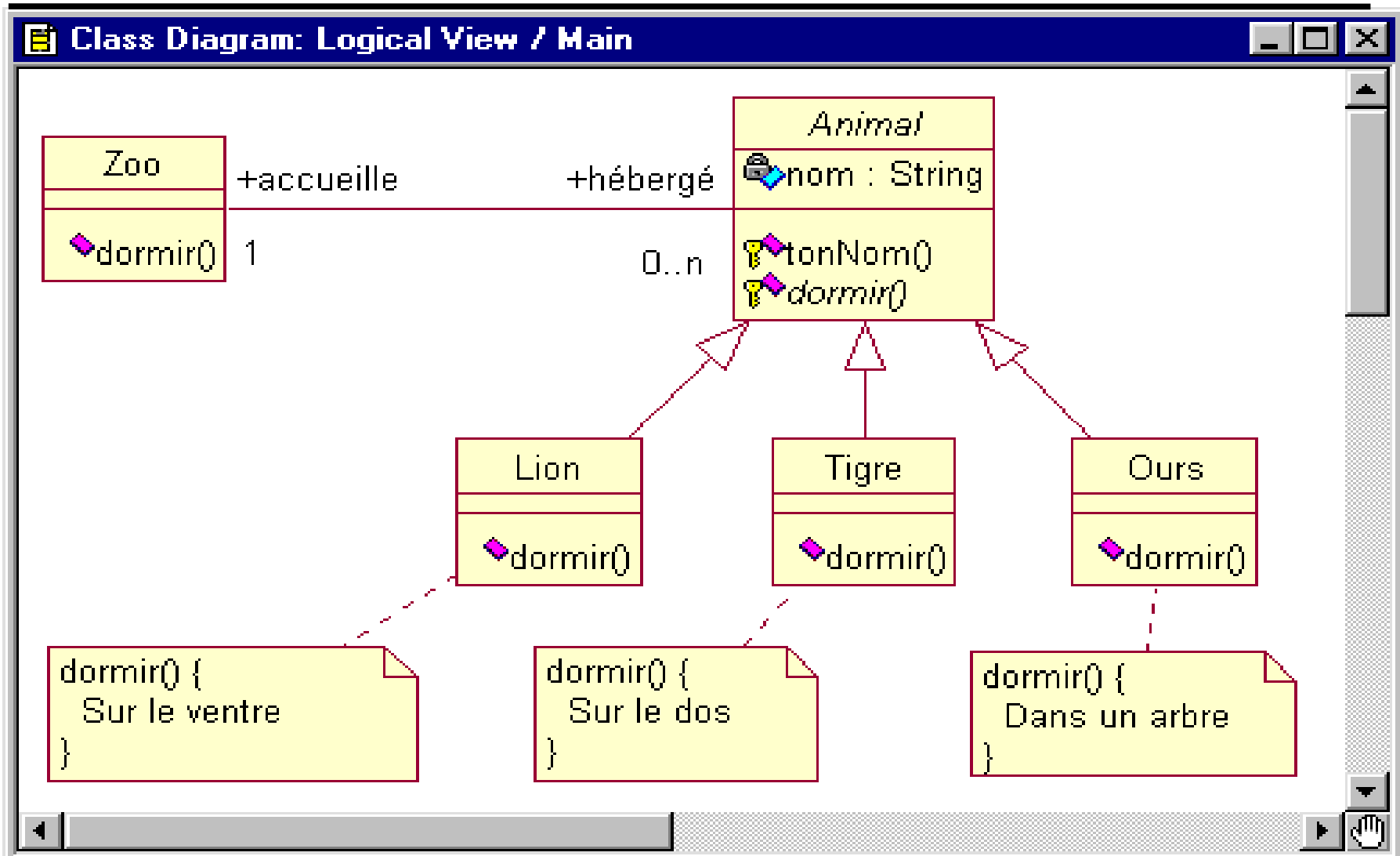
# Le principe de substitution

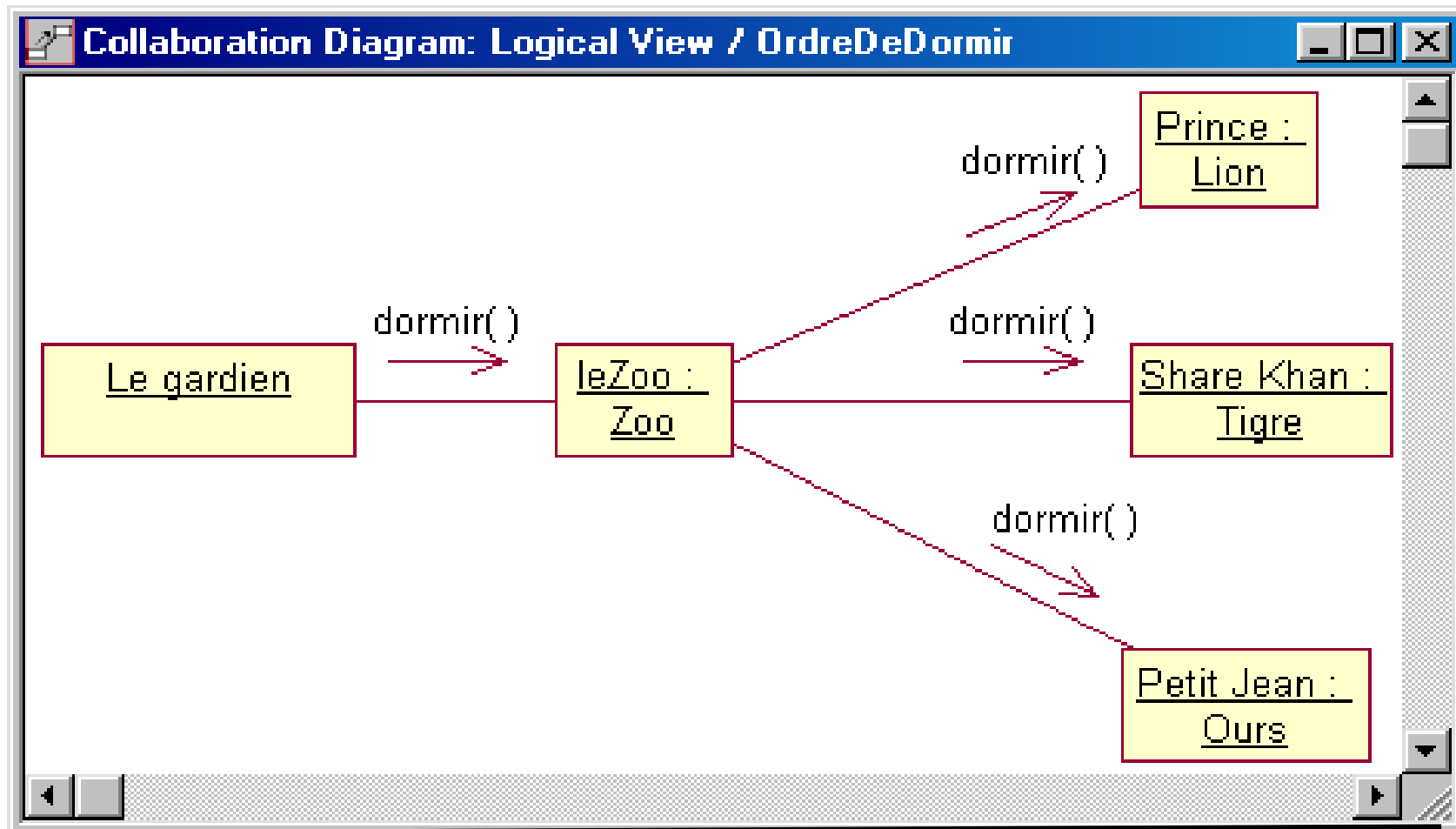
- Le principe de substitution affirme qu'il doit être possible de **substituer** n'importe quel objet **instance d'une sous-classe** à n'importe quel objet **instance d'une superclasse** sans que la sémantique du programme écrit dans les termes de la superclasse ne soit affectée.
- Comme les langages objet permettent la redéfinition des opérations dans les sous-classes, les programmeurs peuvent involontairement introduire des incohérences entre la spécification d'une superclasse et la réalisation dans une de ses sous-classes. Ces incohérences concernent les **contraintes exprimées de manière programmée et non déclarative**.

# Le polymorphisme

- Le terme polymorphisme décrit la caractéristique d'un élément qui peut prendre plusieurs formes, comme l'eau qui se trouve à l'état solide, liquide ou gazeux.
- En informatique, le polymorphisme désigne un concept de la théorie des types, selon lequel **un nom d'objet peut désigner des instances de classes différentes** issues d'une même arborescence.

# Exemple





```
public class Gardien {
public static void main (String arg[]){
    // Crée la liste des animaux du zoo
    Zoo leZoo = new Zoo();
    // Ordre de dormir à tous les animaux
    leZoo.dormir();
}
}
```

```
public class Zoo {  
    // Liste des animaux du zoo  
    private static Vector animaux = new Vector();  
    // Constructeur de la classe Zoo  
    public Zoo () {  
        // Instantiation des animaux  
        Lion unLion = new Lion("Prince");  
        Tigre unTigre = new Tigre("Share Khan");  
        Ours unOurs = new Ours("Petit Jean");  
        // ajout de chaque animal dans la liste  
        animaux.addElement(unLion);  
        animaux.addElement(unTigre);  
        animaux.addElement(unOurs);  
    }  
}
```

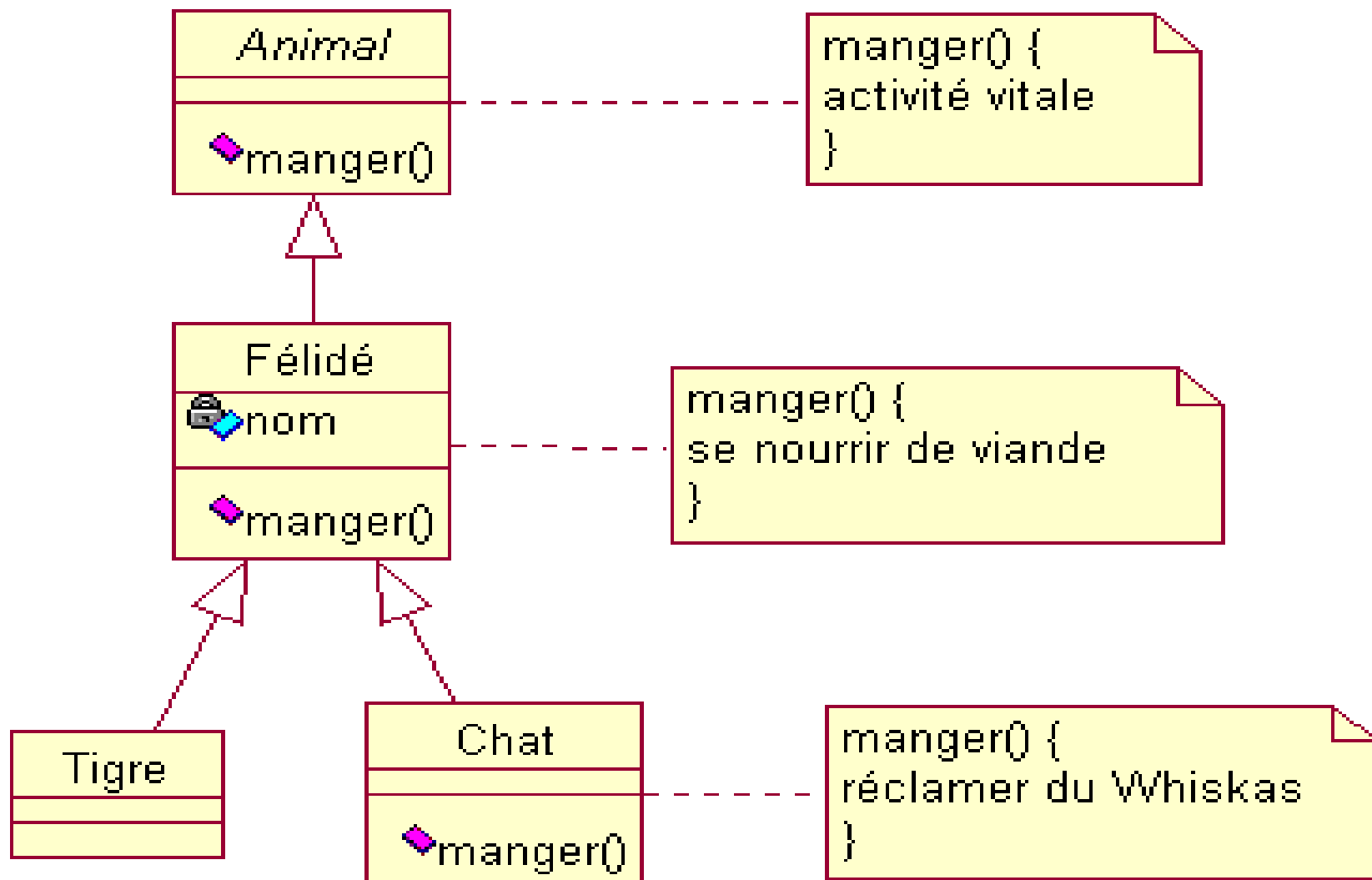
```
public void dormir() {  
    // Interface de parcours de liste  
    Enumeration parcoursAnimaux = animaux.elements();  
    // un animal de la liste - variable polymorphe  
    Animal unAnimal;  
    // tant qu'il y a des animaux dans la liste  
    while (parcoursAnimaux.hasMoreElements()) {  
        // récupération de chaque animal de la liste  
        unAnimal = (Animal)  
        parcoursAnimaux.nextElement();  
        // ordre de se présenter et de dormir à l'animal  
        unAnimal.tonNom();  
        unAnimal.dormir();  
    }  
}
```



- abstract class Animal {
- private String nom;
- // constructeur de la classe Animal
- protected Animal (String nom) {
- this.nom = nom;
- }
- // implémentation de la méthode tonNom()
- protected void tonNom() {
- System.out.print(getClass().getName() + " : " + nom + "
- ");
- }
- // déclaration de la méthode abstraite dormir()
- abstract protected void dormir();
- }

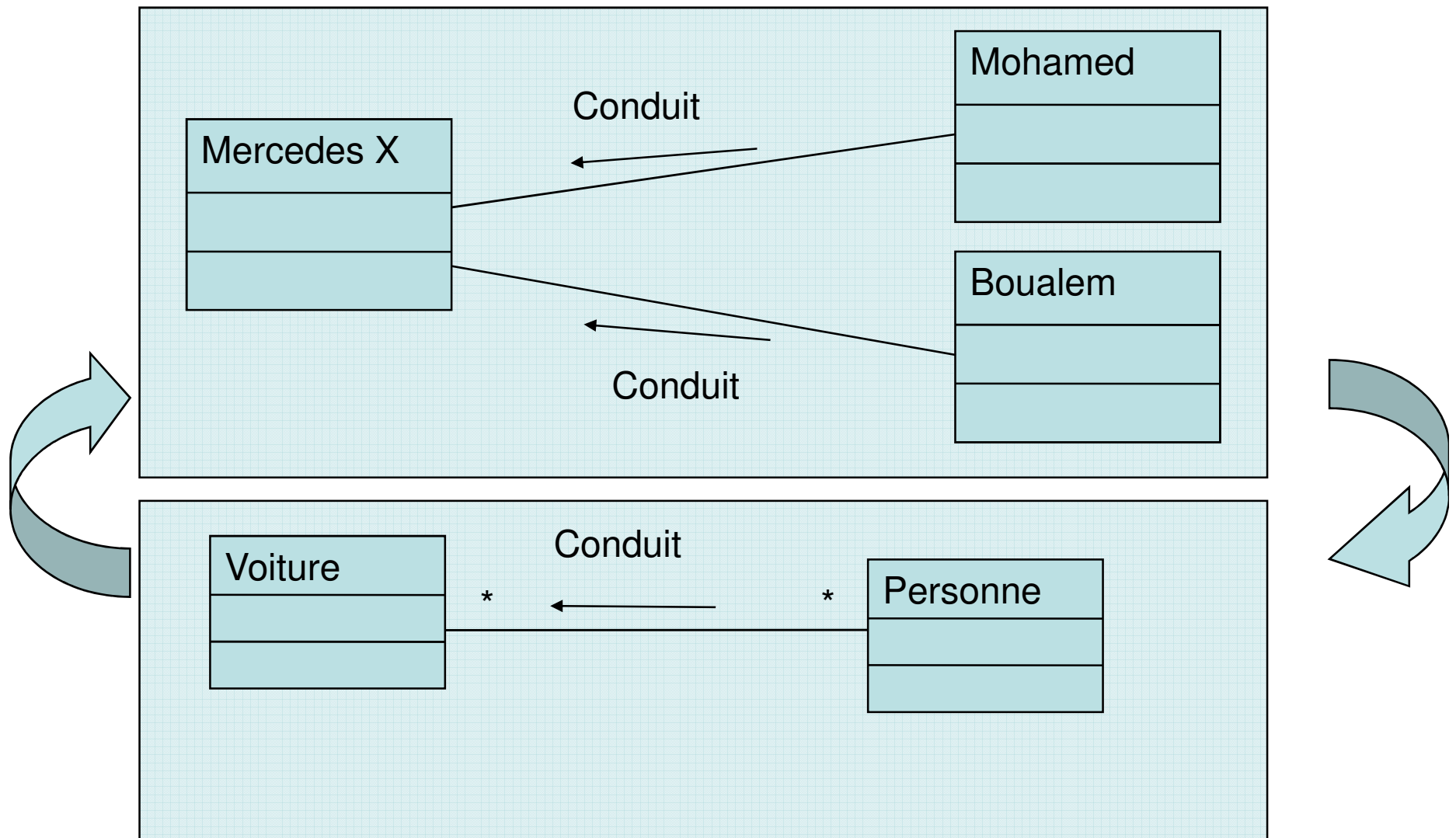
- public class Lion extends Animal {
- // Constructeur de la classe Lion
- public Lion (String nom){
- super(nom);
- }
- // réalisation de la méthode dormir()
- public void dormir() {
- System.out.println ("dort sur le ventre");
- }
- }

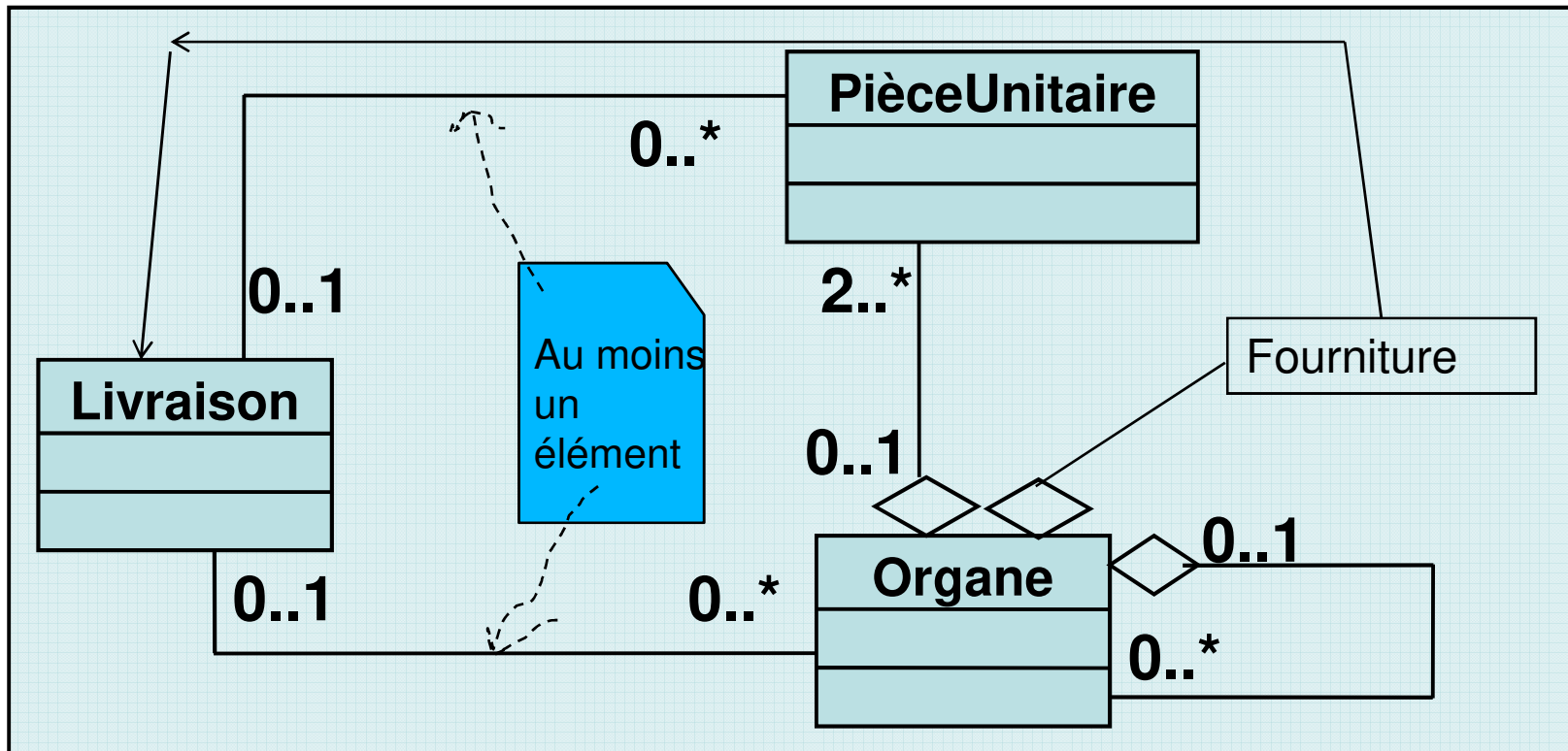
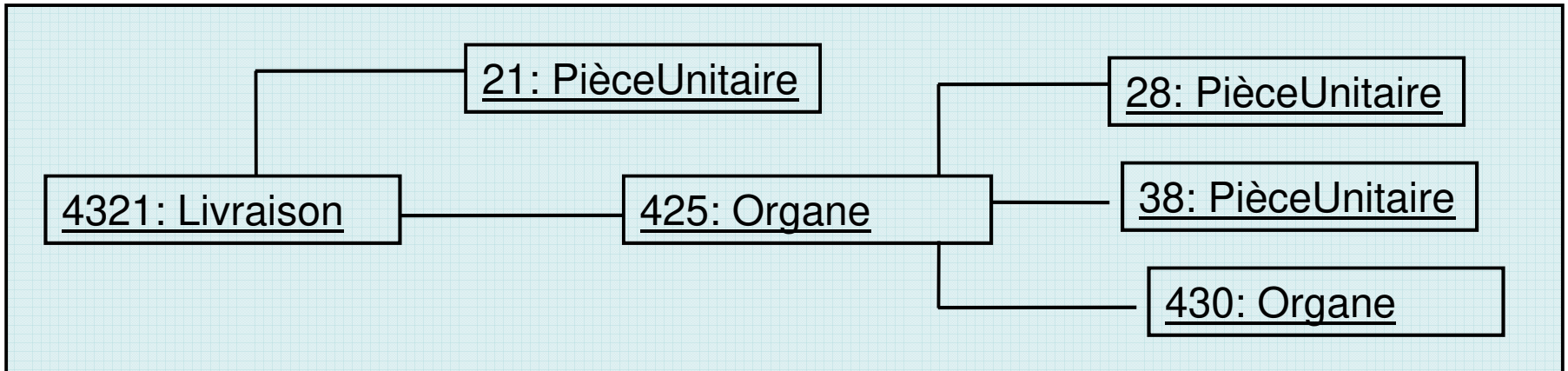
# Exemple



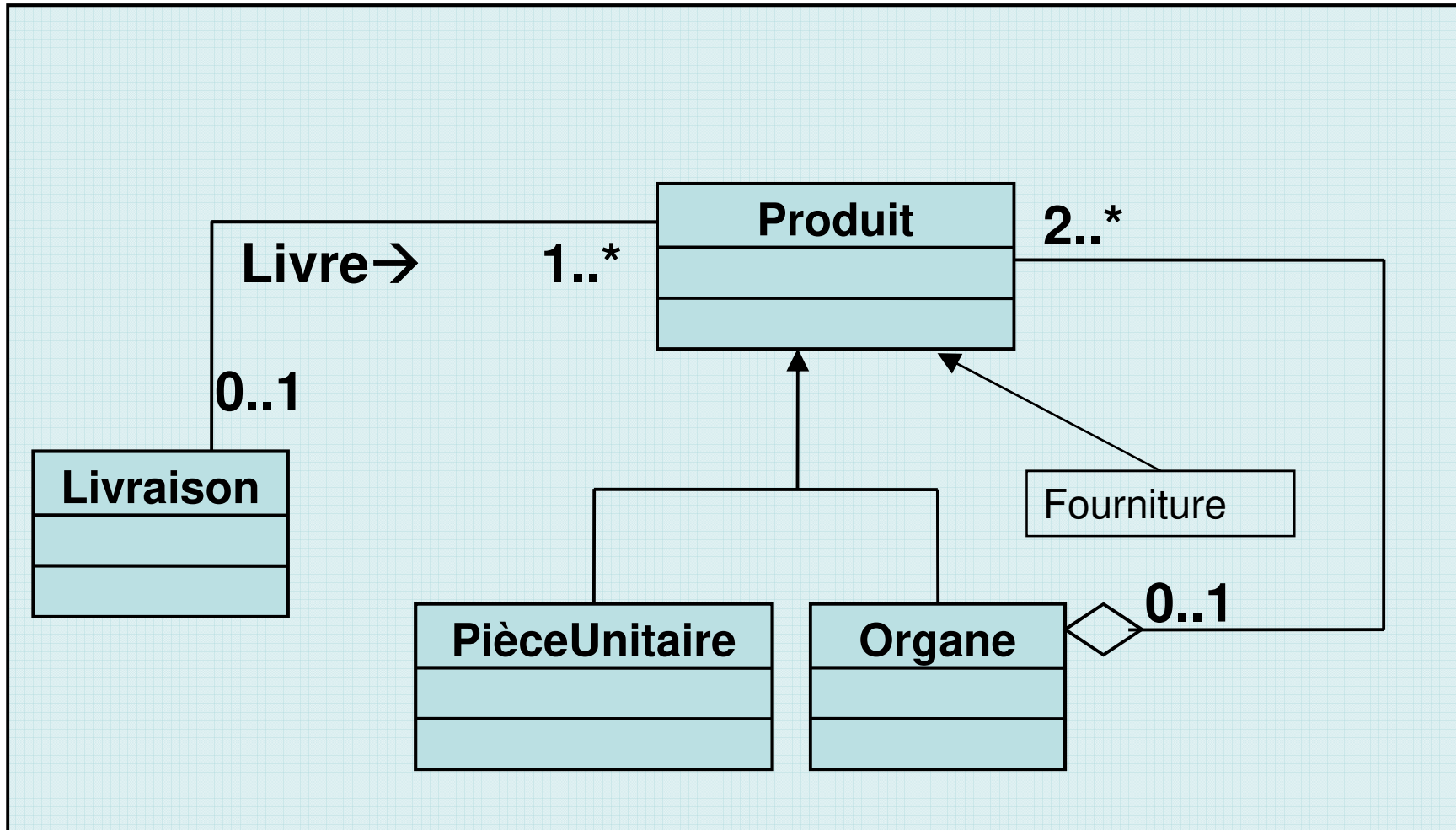
# Diagramme des Objets

- Ce diagramme est un outil de recherche et de test; Il peut aider à comprendre un problème ou à valider un diagramme de classes en représentant des exemples.
- Le diagramme des objets modélise des faits concernant des entités spécifiques, alors que le diagramme de classes modélise des règles concernant les types d'entités.
- Le diagramme des objets est constitué de deux éléments : des objets et des liens.





# Amélioration de la solution précédente



```
class Produit{ ref, nom, qté, ....}
class PU extends Produit{.....}
class Organe extends Produit{
    ArrayList<Produit> elements = new ....
    Void remplirElements(Produit elt){
    }
}
.....
Main(){
    Organe unOrgane = new Organe();
    PU p1 = new PU(); unOrgane.remplirTableau(p1);
    Organe p2=new Organe(); unOrgane.remplirTableau(p2);
}
```



## Exercice :

- Une école comporte plusieurs départements.
- Les enseignants peuvent enseigner divers modules dans plusieurs départements.
- Un même module peut être enseigné dans plusieurs départements.
- L'un des enseignants assure la direction d'un département.
- Chaque Section suit un ensemble de module
- UN étudiant appartient à une seule section au maximum