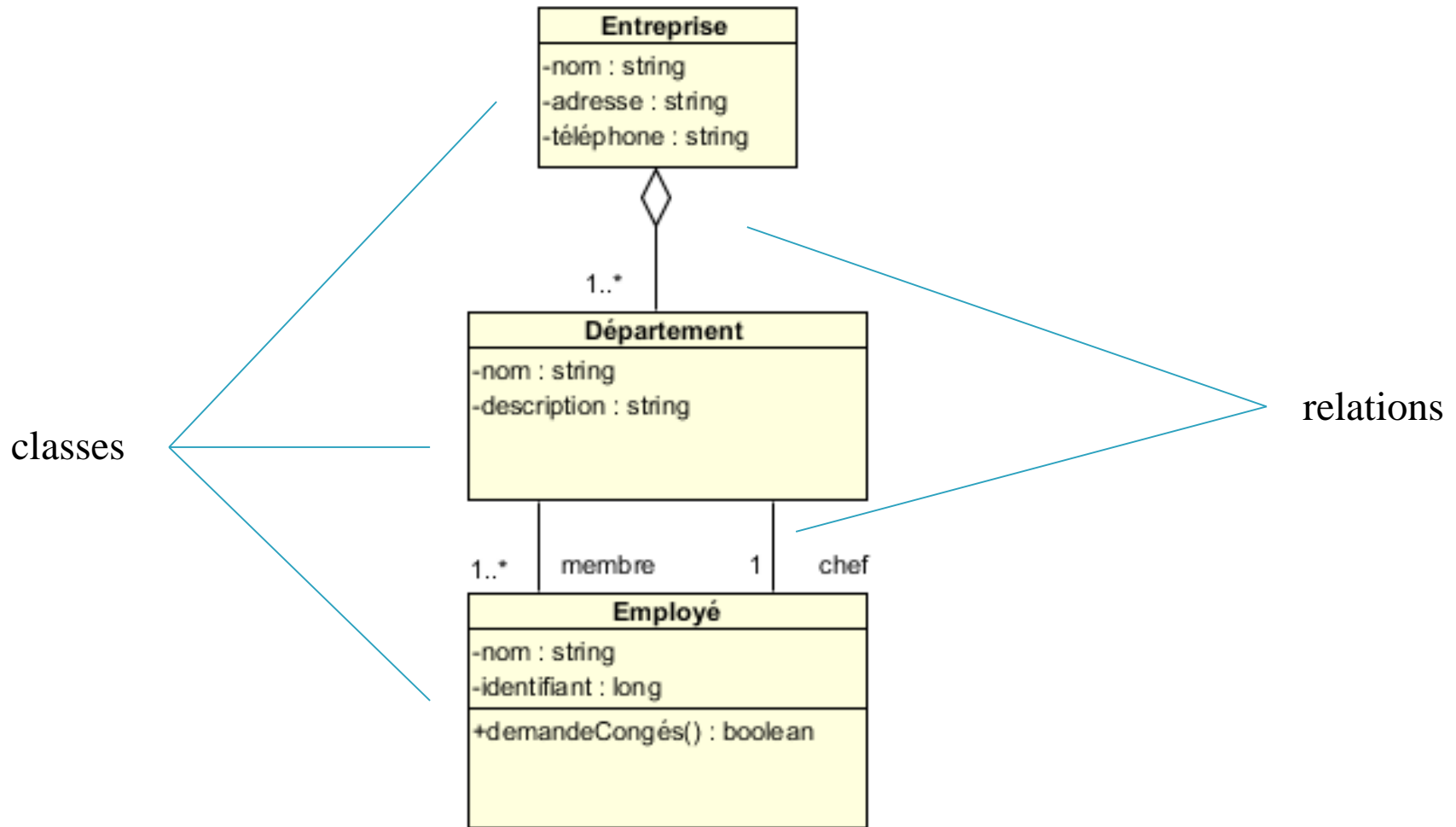


Les diagrammes de classes

Diagramme de classes

- ▶ Le plus important, étant l'objectif final de l'analyse/conception
- ▶ Décrit la **structure interne** du système, sous forme de:
 - **classes** (attributs + opérations)
 - **relations** entre classes
- ▶ Les classes représentent les objets qui réaliseront les cas d'utilisation
- ▶ Ne montre pas comment la fonctionnalité du systèmes, c'est une description **statique**

Exemple de diagramme de classes



Ex: Une entreprise est composée de plusieurs départements. Dans chaque département il y a au moins un employé et un seul chef de département

Concept de classe

- Les objets de la vie quotidienne ont des caractéristiques communes
 - Ex: toutes les clés USB ont une capacité de stockage et permettent d'enregistrer et d'effacer les données
 - Ex: les employés d'une entreprise ont un nom et un prénom, et peuvent demander des jours de congés
- Une **classe** est une description d'un ensemble d'objets qui ont une sémantique, des attributs, des méthodes et des relations en commun

Clé USB
-capacité : long
+enregistrer(string uneDonnée) +effacer()

Classes et objets

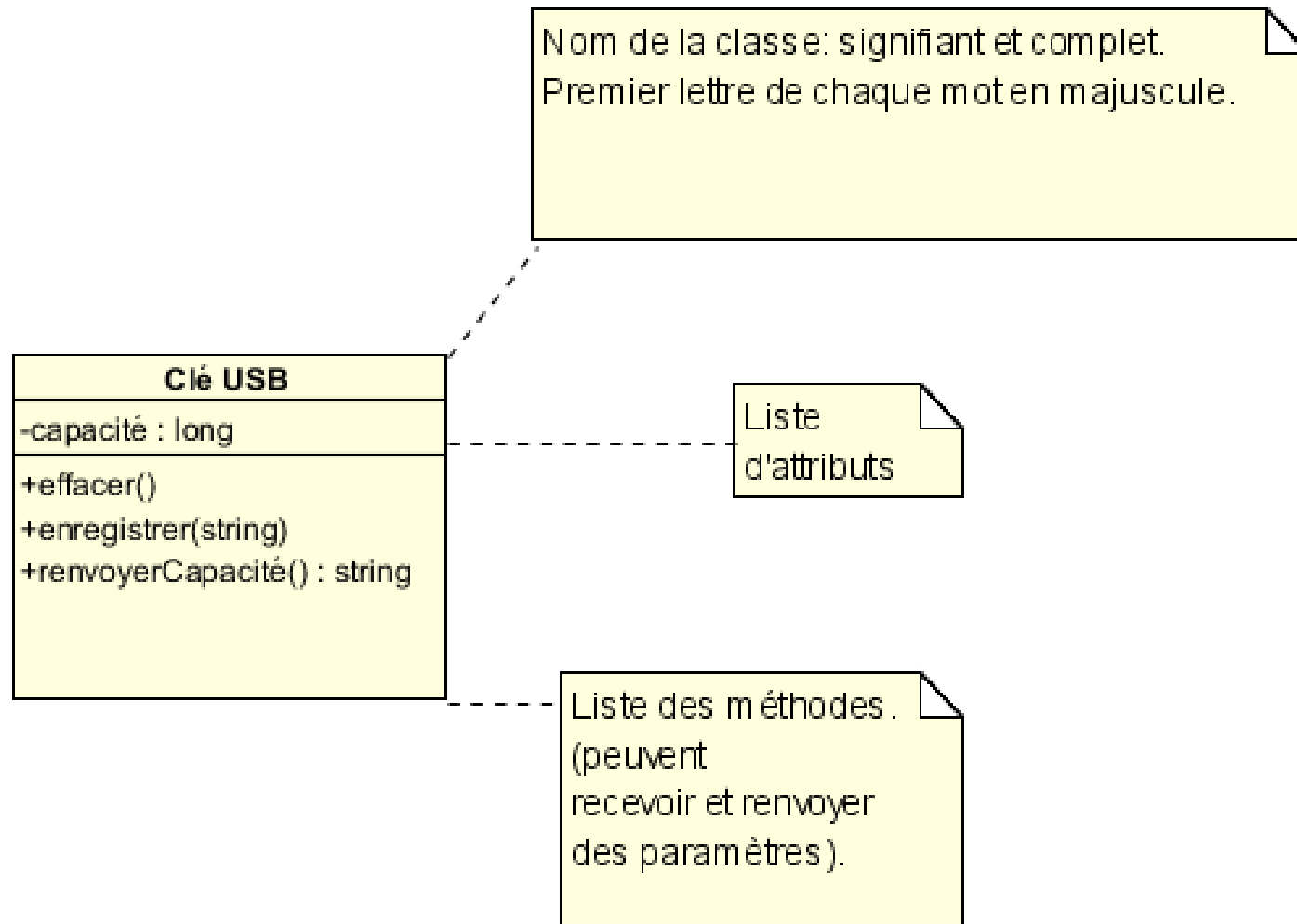
Une classe est un concept abstrait qui peut représenter:

- ▶ des éléments concrets (des personnes, des voitures)
- ▶ des éléments abstraits (des virements, des réservations)
- ▶ des composants d'une application (les boutons des boîtes de dialogue)
- ▶ des structures informatiques (un tableau d'entiers)
- ▶ des éléments comportementaux (un processus d'un logiciel, une exception dans un système)

Un objet est une instance d'une classe:

- Marie est un objet de la classe Personne, et Paul aussi
- Un virement fait aujourd'hui pour payer l'électricité est un objet, et un virement fait demain pour payer l'internet aussi. Ce sont tous les deux des objets (concrets) de la classe Virement

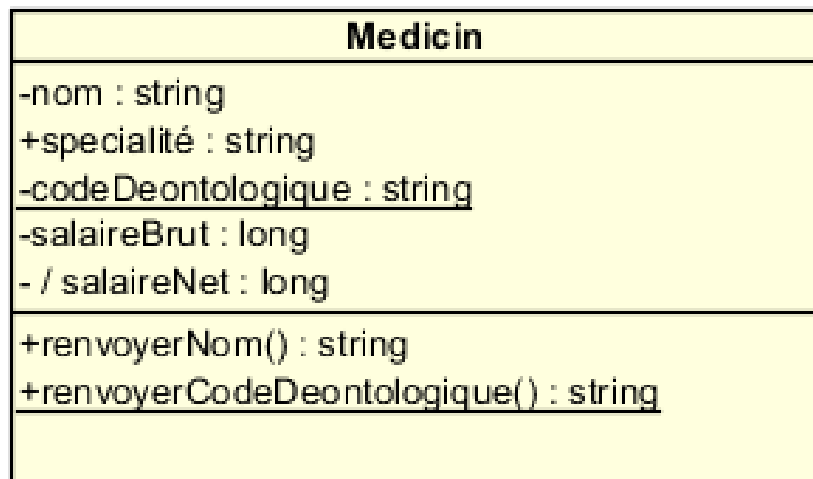
Notation des classes (I)



Notation des classes (II)

- ▶ Nombreuses notations supplémentaires (aux niveaux conception et implantation)
 - Indicateurs de visibilité des attributs et opérations
 - + public (visible par tous)
 - privé (visible dans la classe uniquement)
 - # protégé (visible dans la classe et ses sous classes)
 - Types des attributs et profils des méthodes

- opérations et méthodes **de classe (partagés par tous les objets de la classe)** sont soulignées
- les attributs calculés sont notés (ex: age)
/ attribut : type



Ce texte est commun à tous les médecins. C'est un **attribut de classe** (souligné)

(Scope: classifier en VP)

Relations

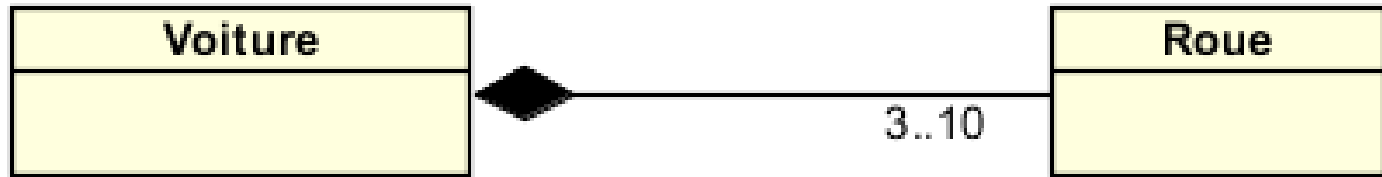
- Expriment les liens sémantiques ou structurels entre les objets des classes
- Les plus utilisées sont **l'association**, **l'agrégation**, la **composition**, la **dépendance** et **l'héritage**
- La **plupart** de relations sont **binaires** (lient deux classes uniquement)
- Plusieurs objets de chaque classe peuvent intervenir dans une relation grâce à la **multiplicité**

Ex: dans la relation binaire Employé-Entreprise il peut y avoir plusieurs objets Employés en relation avec plusieurs objets Entreprise. Marie et Joanne travaillent dans l'entreprise "Glaces Don Giovanni". Joanne travaille aussi pour "Glaces Don Peppone S.L".

Multiplicité

- Indique le nombre d'objets (instances de chaque classe) qui interviennent dans l'association:

Ex: une voiture est composée par d'au moins 3 roues et d'au plus 10 roues. On ajoute 3..10 de côté de la classe Roue



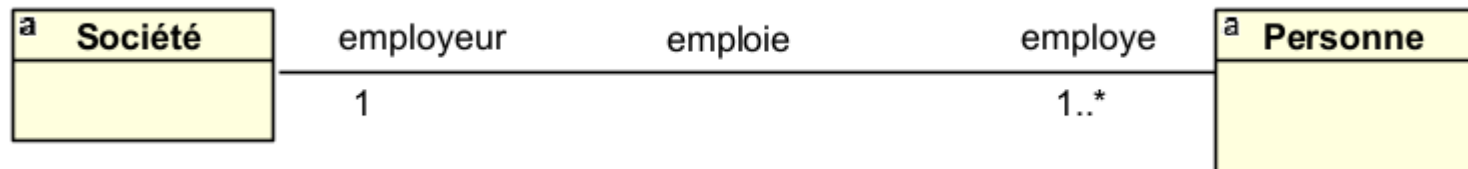
- Les multiplicités possibles sont:
 - 1 → 1 et un seul
 - * → plusieurs
 - 1..* → un à plusieurs
 - 0..1 → zéro à un
 - 5..10 → plusieurs à plusieurs (valeurs connues)

Associations

- Relation sémantique **entre les objets (instances de classe) des classes associées**
- Décrit un ensemble de liens (**instances de l'association**)
- Complémentée par un verbe à l'infinitif
- Chaque extrémité indique le rôle et la multiplicité

Ex: Une société emploie au moins une personne mais un employé travaille uniquement pour une société.

Les mots « employé » et « employeur » expriment le rôle de chaque entité. Le rôle est facultatif en général, mais très utile quand on a plusieurs associations entre les mêmes entités



Associations

- Une association peut être modélisée en utilisant un objet ou un tableau d'objets à l'intérieur des classes
 - Ex: Un employée est associée

```
public class Societe {  
    private Personne[] employes;
```

```
}  
public class Personne {  
    private Societe employeur;
```

```
}
```

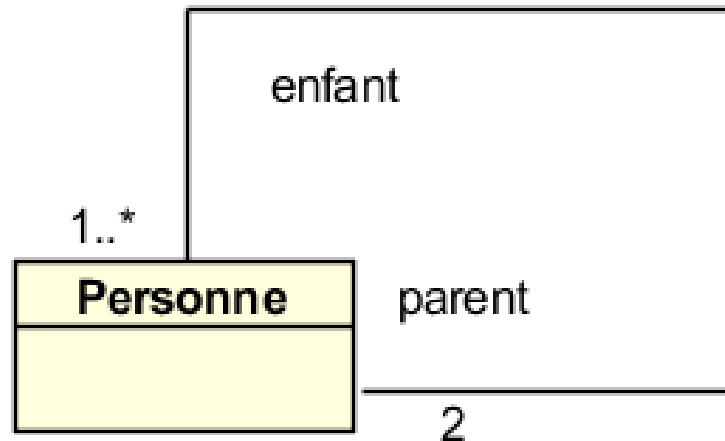
```
public class Polygon {  
    private Point[] sommet;
```

```
}  
  
public class Point {  
    // pas de référence vers Polygon. Un point ne peut pas accéder le polygon  
    // qui le contient  
}
```

Associations Réflexives

- Les objets d'une classe sont associés à des objets de la même classe

Ex: Créer une hiérarchie entre les objets d'une même classe



Associations Réflexives

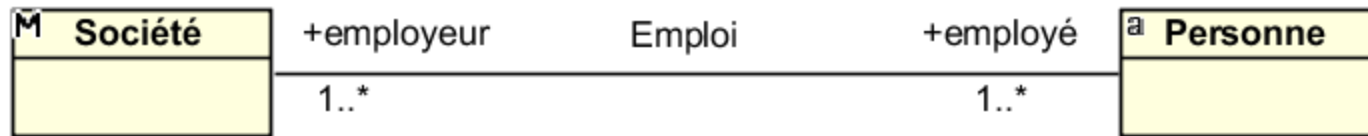
- Peuvent être modelées comme deux tableaux d'objets (un pour chaque rôle):

```
public class Personne {  
    private Personne[] parent;  
  
    private Personne[] enfant;  
}
```

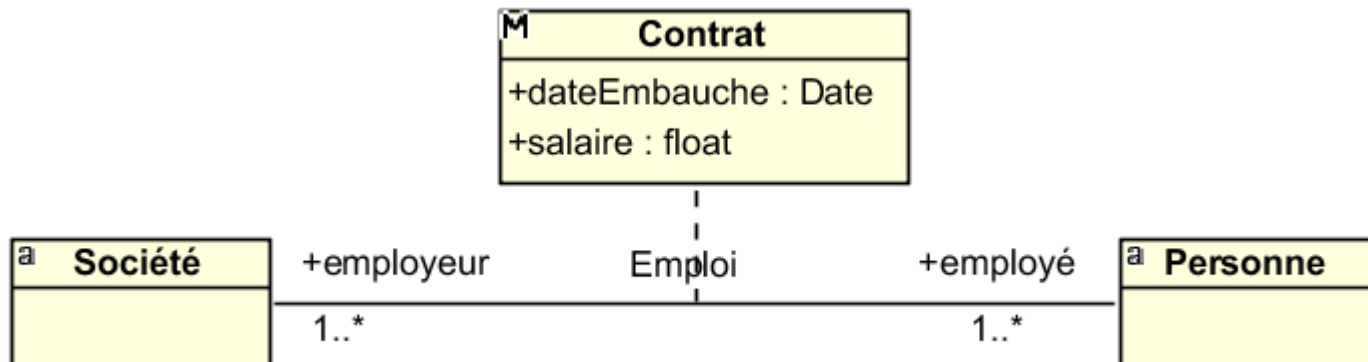
Classe Association (I)

- Association qui a ses propres propriétés, qui ne sont pas disponibles dans aucune des classes qu'elle lie
- L'association devient une classe-association et elle est traitée comme toutes les autres classes

Ex: Si une société a plusieurs employés et un employé peut travailler pour plusieurs sociétés... où peut-on stocker la date d'embauche et le salaire?

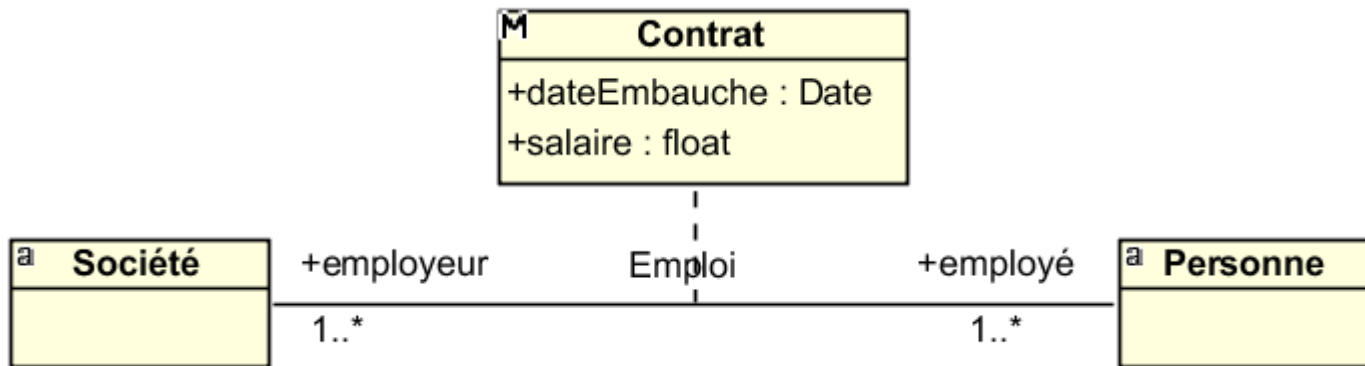


Solution: créer une **classe association** **Emploie**

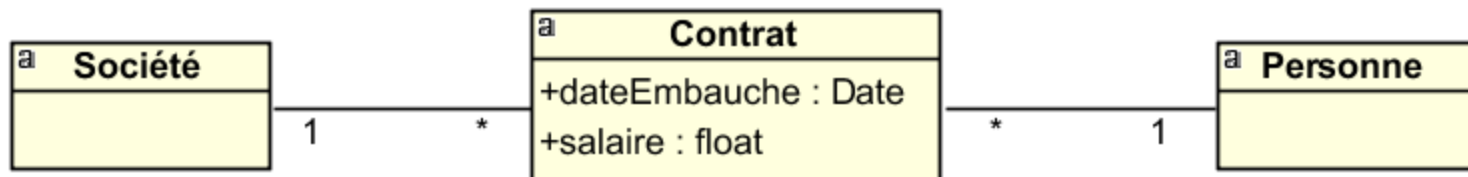


Classe Association (II)

Il y a une autre façon de modéliser la classe association. Nous pouvons remplacer cette représentation



par la suivante (attention aux cardinalités!)



Classe Association (III)

- Chaque objet de la classe association contient ses propres attributs et un objet de chaque classe associée:

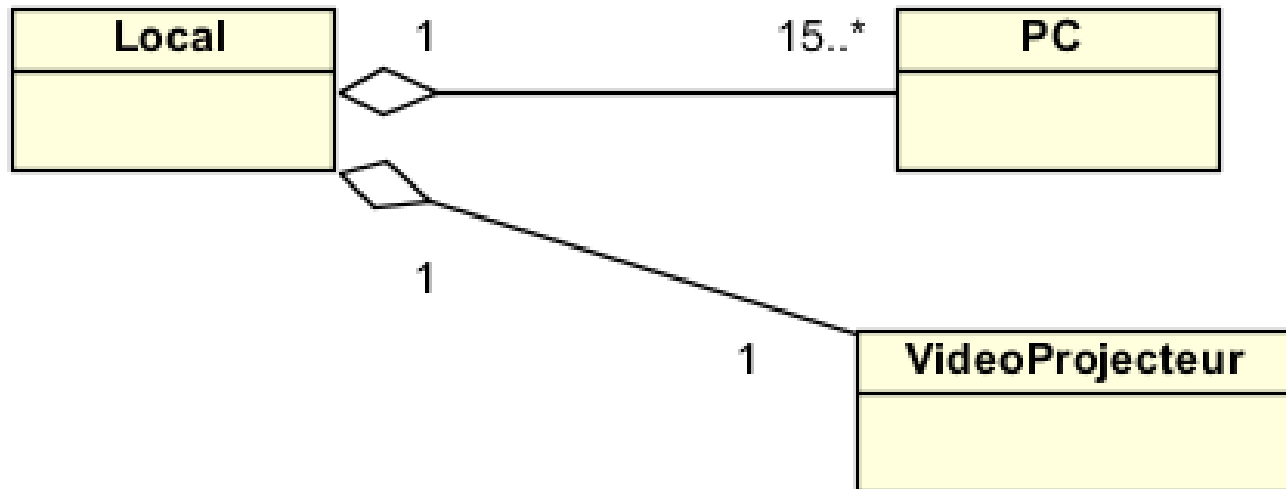
```
public class Contrat {  
    public Date DateEmbauche;  
    public float salaire;  
    private Personne unePersonne;  
    private Societe uneSociété;  
}  
  
public class Personne {  
    private Contrat[] listeContrats; // la personne accède à ses  
                                     //employeurs  
}  
  
public class Societe {  
    private Contrat[] listeContrats; // l'employeur accede à ses  
employés  
}
```

(les deux modèles du slide précédant génèrent le même code)

Agrégation

- Une forme particulière d'association
- Représente l'inclusion d'un élément dans un ensemble

Ex: dans un local d'interface 3 il y a au moins 15 PC et un vidéoprojecteur



- La **durée de vie** des éléments qui composent l'agrégat est **indépendante** de celle de l'agrégat

Agrégation

- L'agrégation peut être modélisée en utilisant un objet ou un tableau d'objets, selon les multiplicités

```
public class Local {
    private PC[] listePC;
    private VideoProjecteur videoProj;

    public Local (VideoProjecteur videoProjInit, PC[] listePCInit){
        this.videoProj=videoProjInit;
        // code pour copier ici chaque element du tableau de
        // listePCInit vers listePC
    }
}

public class PC { // code de la classe
    private Local leLocal;
}

public class VideoProjecteur {      // code de la classe
    private Local leLocal;
}
```

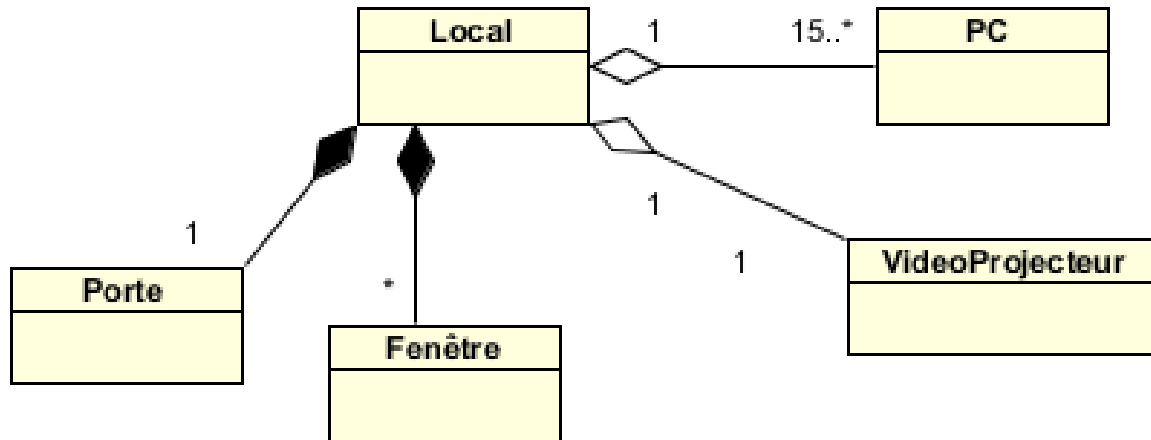
Note: Il y a plusieurs façons d'implémenter une aggregation, ici on voit qu'un exemple

Dans ce cas, les classes PC et VideoProjecteur doivent implémenter ICloneable pour pouvoir copier les objets correctement

Composition

- Une forme particulière d'agrégation ("composite")
- Décrit une contenance structurelle entre les instances: la **création, copie ou destruction** de l'élément agrégat **implique la même opération** sur les éléments qui le composent

Ex: dans chaque local il y a une porte et plusieurs fenêtres qui seront détruites si on décide de "détruire" le local



- La **durée de vie** des éléments qui composent l'agrégat est **dépendante** de celle de l'agrégat (\neq Agrégation)

Composition

- La composition peut être modelée aussi en créant l'objet à l'extérieur mais en faisant appel au destructeur du contenu dans le destructeur du container

```
public class Local {  
    Porte laPorte;  
    public Local() {  
        laPorte= new Porte();  
        // l'objet de la classe Porte est crée dans le  
        // constructeur du Local et devra être  
        // détruit ailleurs dans la classe  
    }  
    .  
    .  
}
```

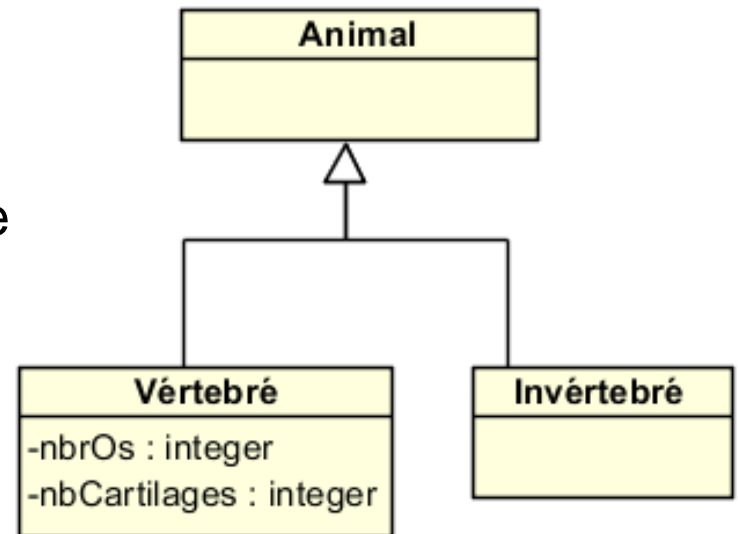
Note: Il y a plusieurs façons d'implémenter une composition, ici on voit qu'un exemple

Hiérarchisation des classes (I)

- ▶ La hiérarchisation (héritage) des classes permet de gérer la complexité: simplifie la modélisation
- ▶ Nous pouvons **généraliser** ou **spécialiser** les classes de notre système
- ▶ Nous avons la classe parent ou **superclasse** et la classe dérivée ou **sous-classe**

Spécialisation : adapter une classe trop générale à des cas particuliers.

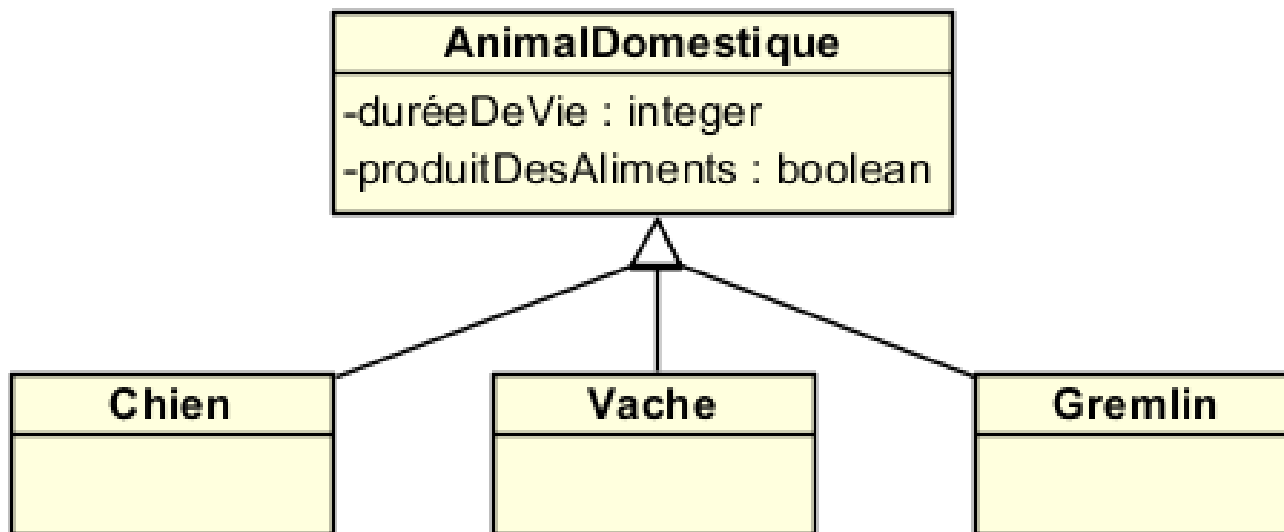
Ex: la classe Vertébré spécialise la classe Animal: on rajoute le nombre d'os et le nombre de cartilages



Hiérarchisation des classes (II)

Généralisation : factorisation des éléments communs de classes (attributs, opérations); favorise la réduction de la complexité.

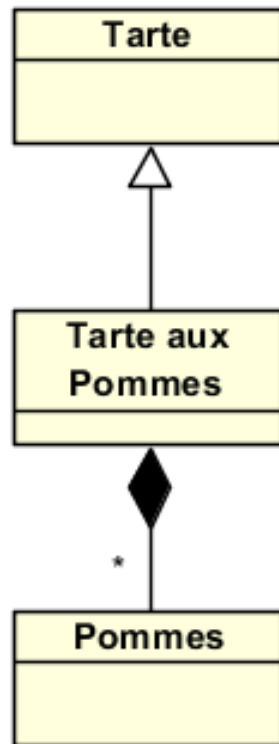
Ex: grouper toutes les propriétés communes des chiens, des vaches et des gremlins dans une superclasse AnimalDomestique



Règles de la Hiérarchisation

- ▶ La classe fille possède toutes les propriétés de ses parents mais elle ne peut pas accéder à ses membres privés
- ▶ Une sous-classe peut redéfinir une méthode d'une superclasse. La définition de la sous-classe sera utilisée.
- ▶ Toutes les associations s'appliquent aux sous-classes
- ▶ On peut utiliser toujours une instance d'une sous-classe là où on utilise une instance de sa classe parent (ex: toute opération acceptant un objet Animal doit accepter un objet Chien... pas à l'inverse!)
- ▶ Une classe peut avoir plusieurs classes parents (héritage multiple... et multiplication des problèmes dans l'implémentation aussi!)

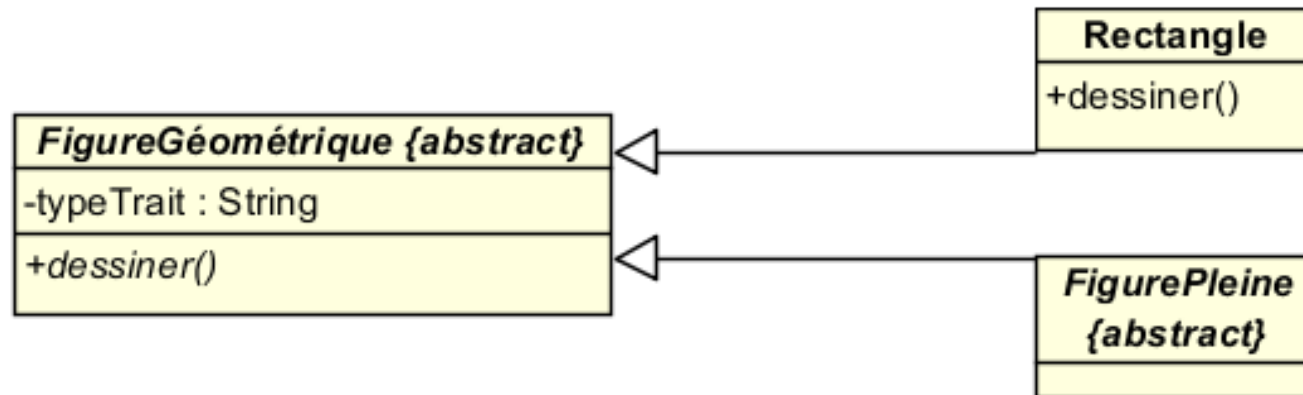
Ne pas confondre généralisation/spécialisation et agrégation ! Quand une classe est une spécialisation d'une autre elle est **de même nature** ce qui n'est pas le cas avec l'agrégation. Ces relations peuvent être associées.



Une pomme n'est pas de la même nature qu'une tarte !

Méthodes et Classes Abstraites

- ▶ **Méthode abstraite:** Une méthode dont l'implémentation n'est pas définie. Cette méthode sera implémentée dans une sous classe
- ▶ **Classe abstraite:**
 - 1) si elle possède au moins une méthode abstraite (italique!)
 - 2) si elle hérite d'une classe abstraite contenant une méthode abstraite qui n'a pas été encore réalisée



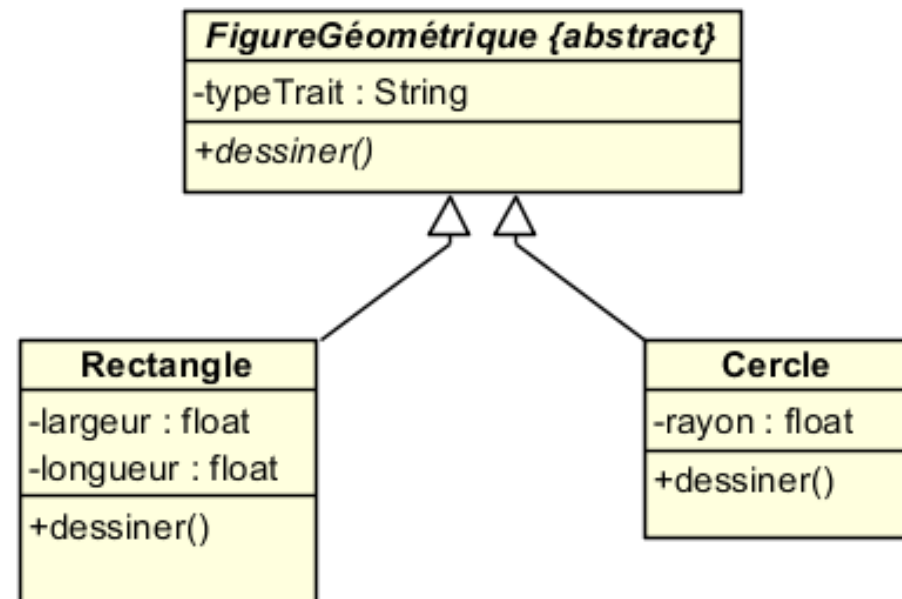
La classe *FigureGéométrique* est abstraite (1) et la classe *FigurePleine* aussi (2) – elle hérite mais n'implémente pas la méthode. La classe *Rectangle* n'est pas abstraite: elle implémente la méthode

Remarque: On ne peut pas créer des objets d'une classe abstraite!

La collaboration des objets

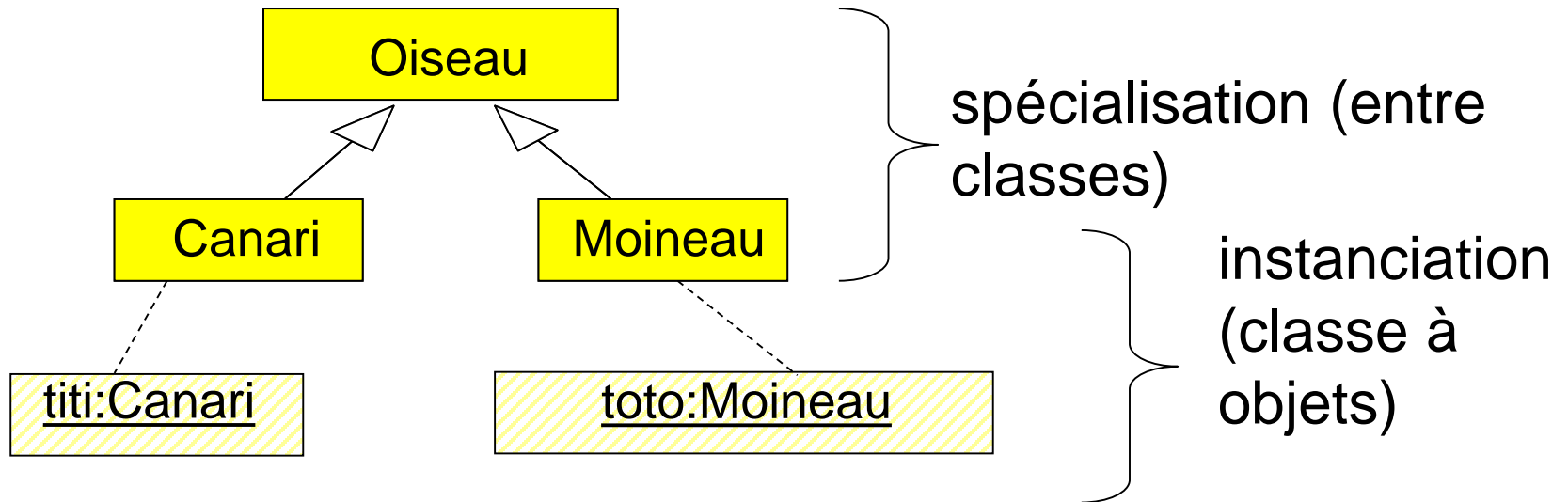
- ▶ Les objets collaborent en utilisant l'envoi « envoi de messages » (appel d'opération/méthode)
- ▶ Un même message peut être traité de manière différente par différents types de receveur (**polymorphisme**). L'émetteur n'a pas à connaître la classe du receveur

Ex : On peut appeler à la méthode **dessiner** de n'importe laquelle FigureGeometrique sans savoir s'il s'agit d'un Rectangle ou un Cercle: la méthode `dessiner()` correspondante sera appelée selon le **type effectif** de l'instance (ex: instance **c1** de **Cercle** ou **r3** de **Rectangle**). Si un nouveau type de figure est ajouté au système, on ne modifie rien dans le reste de classes



Remarques

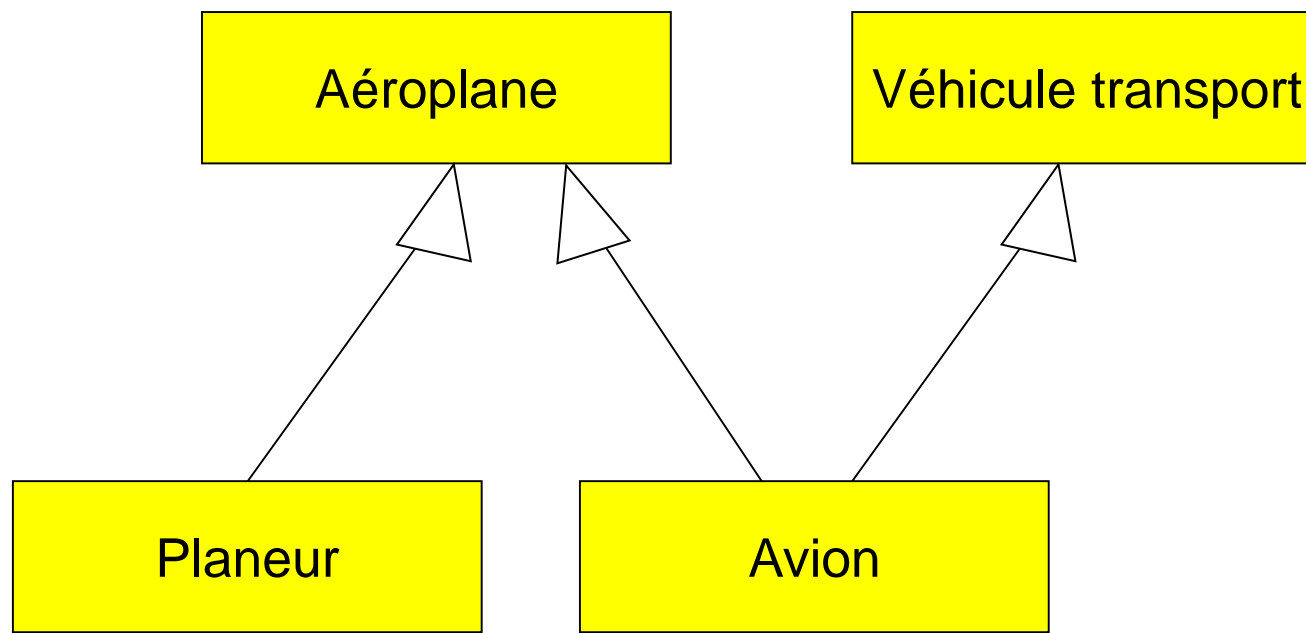
- ▶ Ne pas confondre spécialisation et instanciation !



- ▶ Notation UML des objets : identificateur:classe ou :classe (si on ne spécifie pas un objet = objet anonyme)
- ▶ Les objets de la classe spécialisée **héritent** de la description des attributs (variables) et des opérations (méthodes) de la superclasse
- ▶ Elles peuvent en **ajouter** d'autres et/ou en **redéfinir** certaines

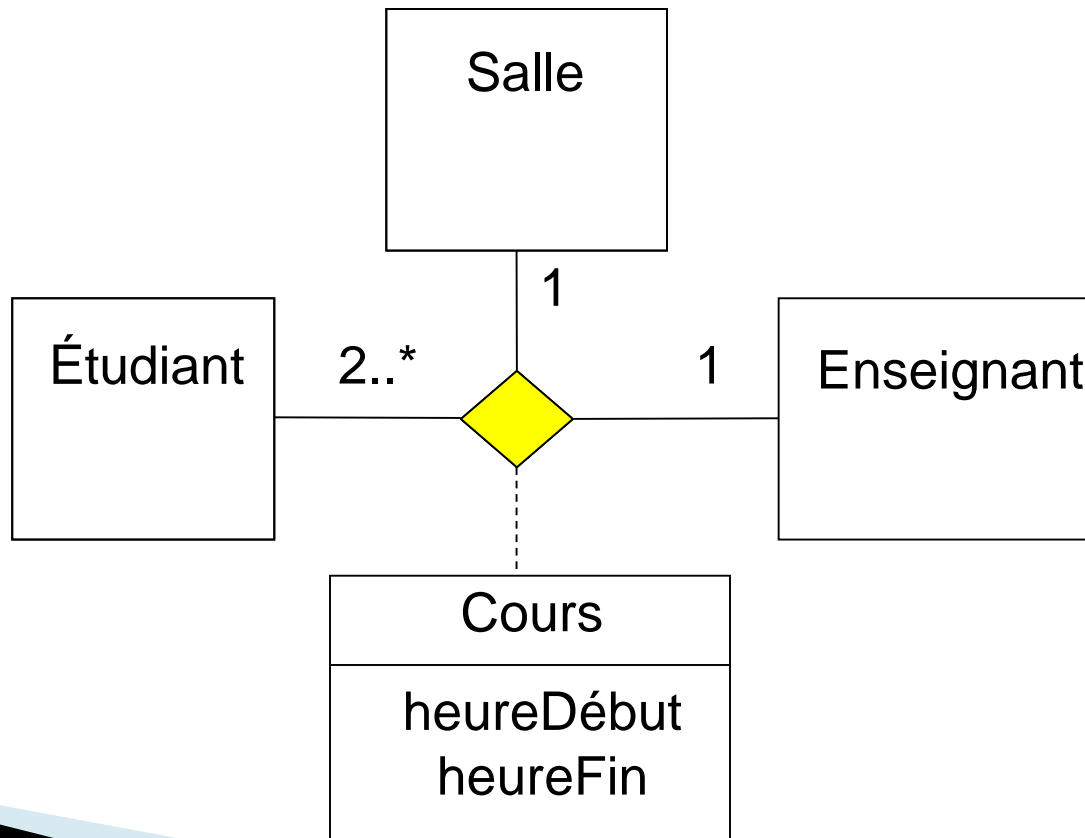
Héritage Multiple

- Une classe a plusieurs super classes
- Difficile à implanter: il y a souvent une manière de trouver une solution au modèle en utilisant l'héritage simple.



Association d'arité n

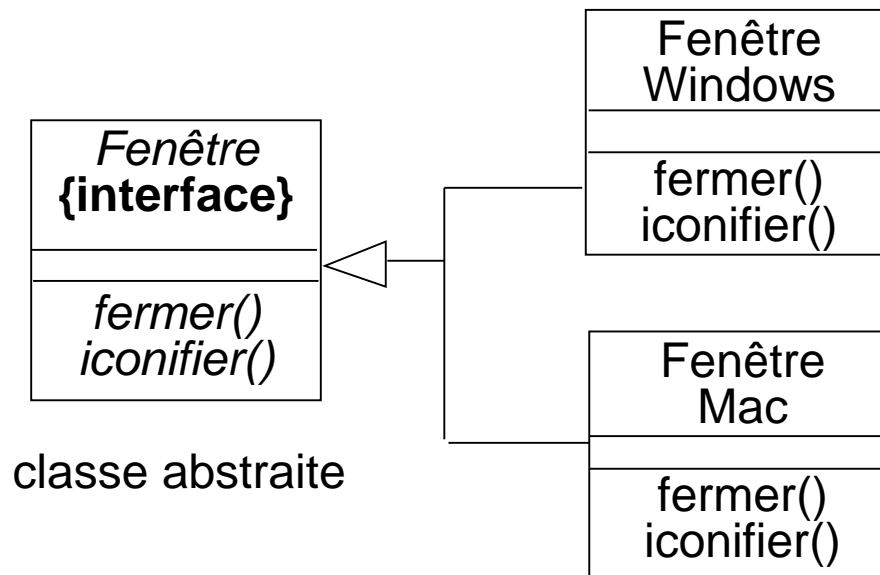
Représentée par un losange avec n 'pattes' auquel peut être associé une classe porteuse d'attributs et/ou d'opérations.



Interface

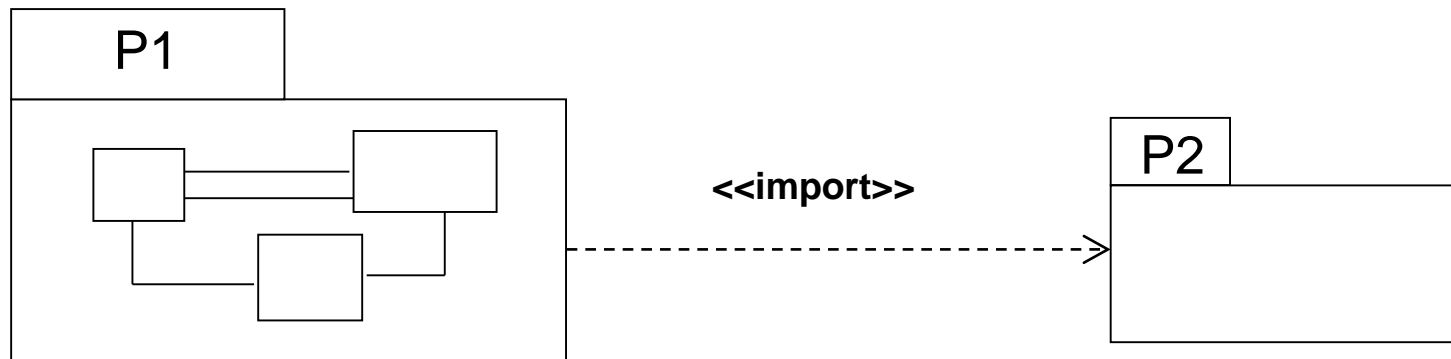
Interface : classe abstraite ne contenant que des opérations abstraites ;

Une interface sert à préciser les fonctionnalités (le «service») que les classe qui les implantent doivent fournir.



Paquetage

Paquetage (package) : notion qui peut apparaître dans tous les diagrammes pour spécifier le regroupement d'éléments au sein d'un sous modèle (cas, classes, objets, composants, autres paquetages...).



<<import>> : les éléments de P1 peuvent utiliser ceux de P2
(permet de décrire la structure générale d'une application)
P1 est dépendent de P2, pas à l'inverse.

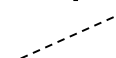
Contraintes

Contraintes : entre { }; exprimées dans le langage OCL (« Object Constraint Language »); portent sur les classes, les attributs, les méthodes, les associations etc. OCL n'est pas détaillé dans ce cours.

Exemple :

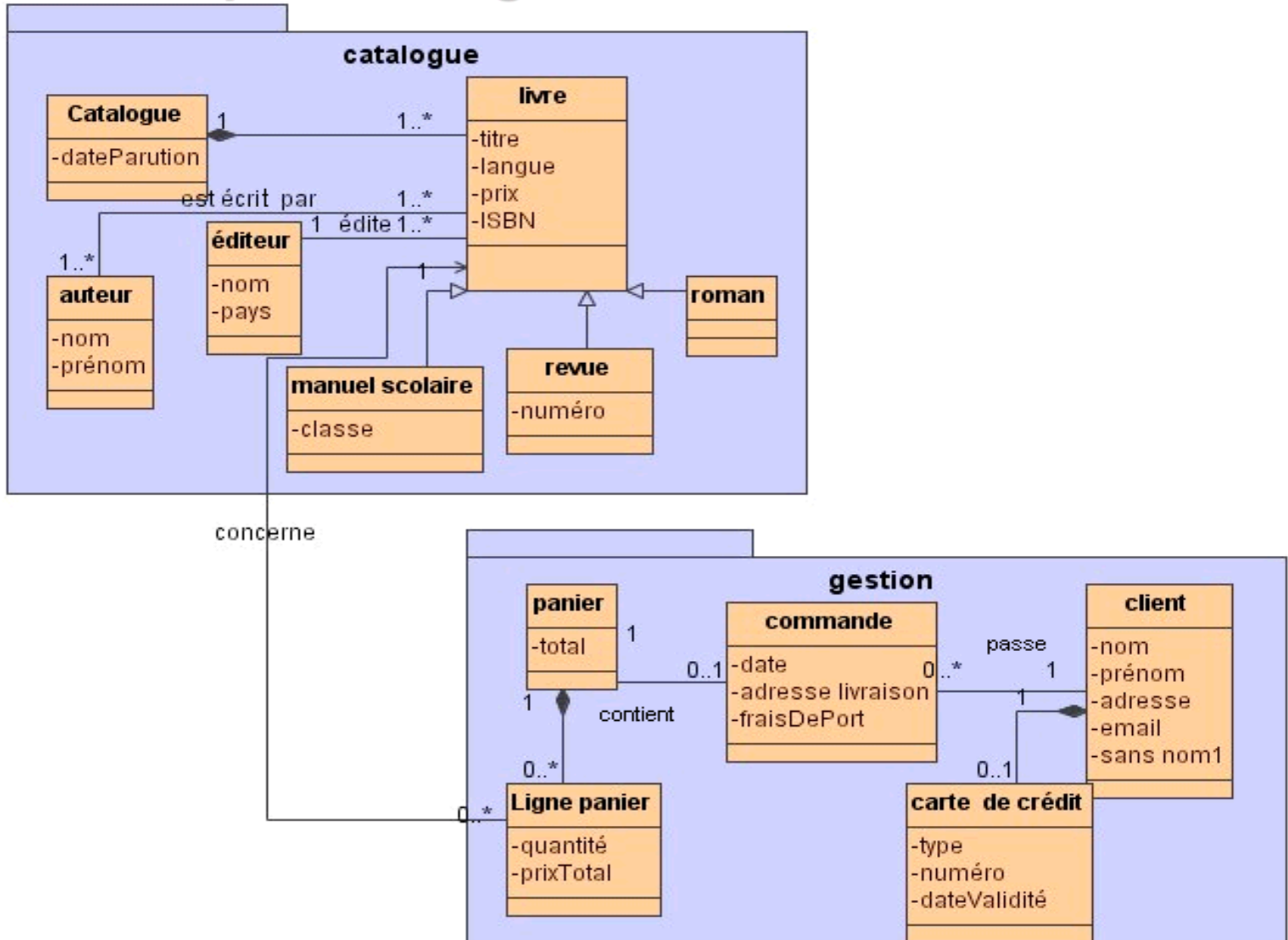
contrainte de classe et d'attribut :

{actif = passif}



Entreprise	
actif : float	{value >= 0}
passif : float	

Exemple de diagramme de classes



Méthodologie pour la création du modèle de classes

1. Trouver les classes
2. Trouver les associations
3. Trouver les attributs de chaque classe
4. Organiser et simplifier le diagramme

Itérer et affiner le modèle!

Trouver les classes

- substantifs du domaine
 - Ex: compte, appel
 - Eviter les opérations qui s'appliquent aux objets. Ex: payer, appeler
- éliminer les classes redondantes et superflues
- éviter les noms vagues ou porteurs de rôles
 - Utilisez "Pilot" ou "Mécanicien" au lieu de "EmployéDansÉcurieDeCourseAutomobile"
 - Utilisez "Pilot", pas PremierPilote et DeuxièmePilote
- si un substantif n'est pas décomposable, il s'agit normalement d'un attribut (ex: âge)

Trouver les associations

- **verbes mettant en relation** plusieurs classes
 - Ex: "est composé de", "travaille pour"
- éviter les associations qui mettent en jeu plus de deux classes
 - Ex: dans l'association "conducteur-assurance-voiture", l'assurance peut être un attribut de "conducteur-assurance"
- ne pas confondre une association avec un attribut
 - Ex: l'association "est plus lourd" entre les classes Avion et Voiture peut s'exprimer avec l'attribut "masse" dans chaque classe
- éviter les chemins redondants dans les associations

Trouver les attributs des classes

- ce sont de substantifs qui ne sont pas de classes
 - Ex: "la masse d'une voiture", "le montant d'une transaction"
- ses valeurs sont représentées par des adjectifs ou des expressions
 - Ex: "rouge", "50 euros"
- éviter la dépendance entre les attributs
 - Ex: si on a "datenaissance", éviter "âge"
- les attributs peuvent être ajoutés pendant toutes les étapes du cycle de vie d'un projet (implémentation comprise)

Organiser et simplifier le diagramme en utilisant l'héritage

- "vers le bas": spécialiser une classe en une ou plusieurs sous-classes
 - Ex: créer les classes "Programmateur", "Analyste" et "Responsable de projet" à partir de la classe Employé
- "vers le haut": prendre les éléments communs de plusieurs classes et créer une classe générale
 - Ex: créer la classe "Carte Bancaire" à partir de "Carte Crédit" et "Carte Débit"