COURS 5 : DIAGRAMME DE CLASSES

Samia BOULKRINAT

(Basé sur le cours de Assia HACHICHI)

Les quatre principes de l'Orienté Objet

1. La classe et l'objet
2. Encapsulation,
3. Héritage,
4. Polymorphisme…

La classe et l’objet

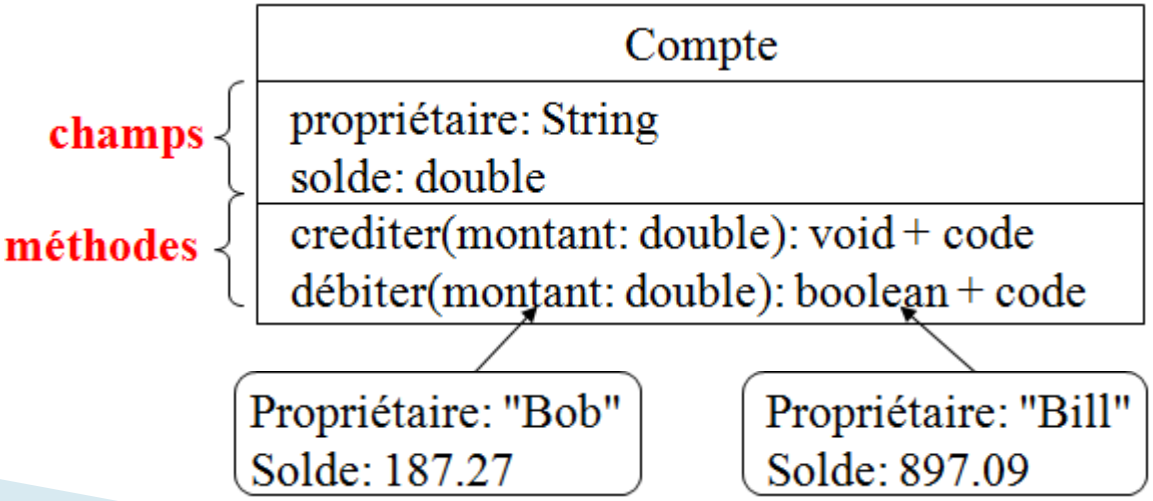
- **<u>La classe</u>** : "Abstraction d'un type de donnée caractérisée par des propriétés (attributs et méthodes) communes à des objets, et permettant de créer des objets ayant ces propriétés".

- **<u>L’objet</u>** : "une instance de la structure de données ainsi que du comportement définit par la class de l'objet. Chaque Objet a ses propres valeurs pour les variables d'instances de sa classe et répond aux méthodes définies par celle-ci".

La classe et l'objet

- **Classe** : élément de conception modélisant une entité du problème à résoudre, contient
  - Les données de l'entité (variable)
  - Les méthodes manipulant ces données (code)
  - **But** : regrouper dans une même entité les données et leurs méthodes. Seules les méthodes sont visibles.

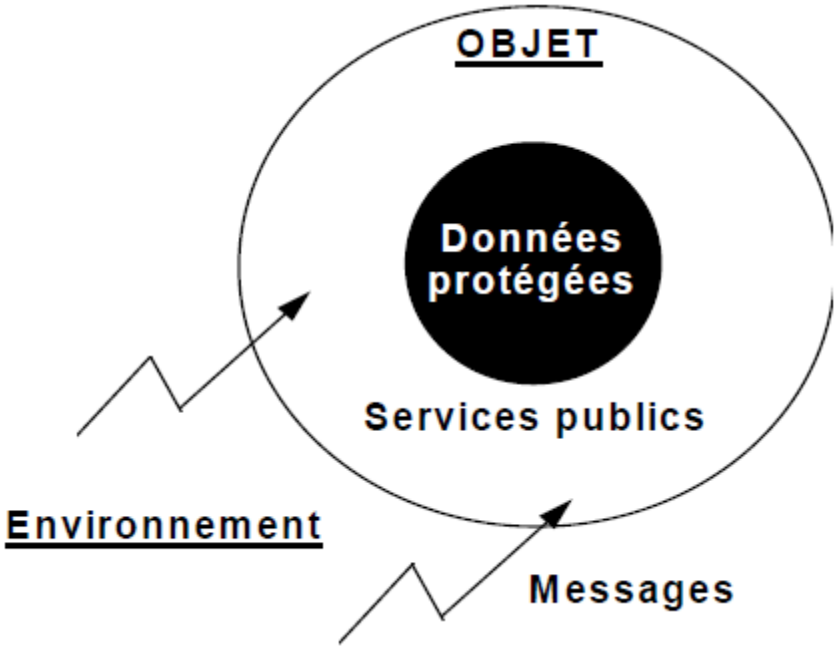
- **Objet** : instance d'une classe
  - Élément d'exécution possédant les propriétés de la classe



La classe

Locomotive
- modele - puissance - vitesse - couleur
+ allerEnAvant( ) + allerEnArriere( ) + demarrer( ) + stopper( ) + accelerer( )

L'objet


OBJET
Données protégées
Services publics
Environnement
Messages

Concepts de base

- Instanciation et gestion de la mémoire

- Instancier une classe X : appel **new X(arguments)**
  - Allocation de l'espace mémoire de X
  - Appel de la méthode spéciale X.X(arguments) appelée constructeur
  - Retourne une référence vers l'instance de X
  - Une référence n'est pas un pointeur (pas d'arithmétique)

- Supprimer une instance de X : **automatique**
  - Suppression automatique d'une référence dès qu'elle n'est plus référencée
  - **Aucun contrôle sur l'instant de suppression!**
  - Appel automatique de X.finalize() si définie

Variables et Méthodes de classe.

- 2 types de champs et de méthodes

- Champs et méthodes de classe :
  - Mot clé : static
  - Partagés par toute les instances

- Champs et méthodes d'instance :
  - Mot clé : aucun (défaut)
  - Liés à une instance

Classe X
static int a = 22;

x1.a = X.a = x2.a

Instance x1 de X
int b = 37;

Instance x2 de X
int b = 42;

- **Remarque** :
  - Une méthode d'instance peut manipuler des champs de classe et d'instance
  - Une méthode de classe ne peut manipuler que des champs de classe

Exemple

```
public class Compte {
  private String proprietaire; // private ⇒ invisible hors de la classe
  private double solde;

  public Compte(String proprietaire) {
                    // Constructeur, appelé lors de la création
    this.proprietaire = proprietaire; this.solde = 0;
                              // this référence notre instance
  }

  public void crediter(double montant) { solde += montant; }

  public boolean debiter(double montant) {
                    // public : visible en dehors de la classe
    if(solde >= montant) { solde -= montant;  return true;
    } else return false;
  }}
```

Exemple

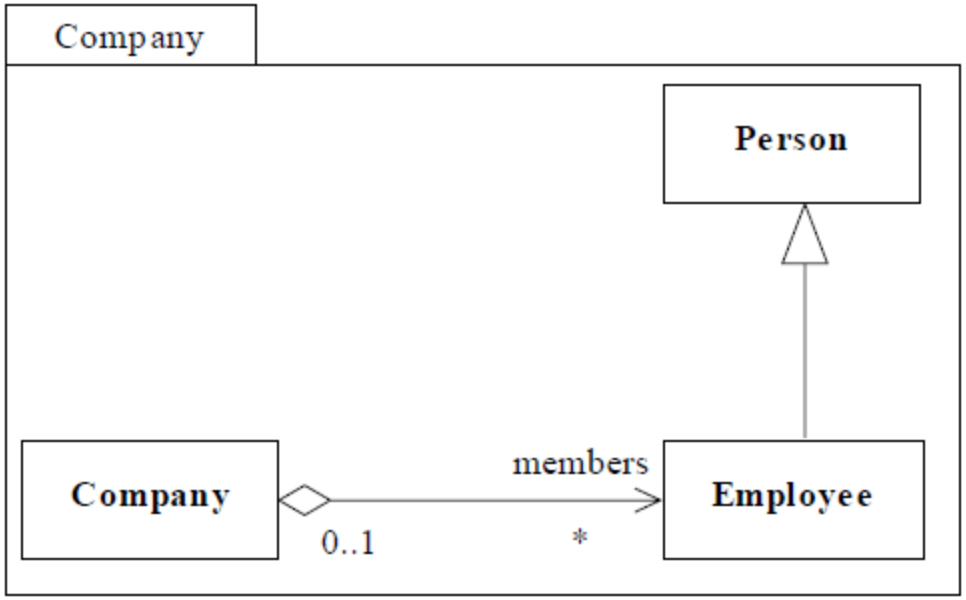
```
public class TestCompte {
  private static Compte bob;
      // static ⇒ champs partagé entre toutes les instances
  private static Compte bill;

  public static void main(String args[]) {
                                // type [] = tableau
    bob = new Compte("Bob");
                        // création du Compte de Bob
    bill = new Compte("Bill");
                        // création du compte de Bill
    bob.credite(187.12);
  }
}
```

Diagramme de classes

- Un diagramme de classes est un graphe d'éléments connectés par des relations.

- Un diagramme de classes est une vue graphique de la structure statique d'un système.



Objectif

- Les diagrammes de cas d'utilisation modélisent à QUOI sert le système.

- Le système est composé d'objets qui interagissent entre eux et avec les acteurs pour réaliser ces cas d'utilisation.

- Les diagrammes de classes permettent de spécifier la structure statique d'un système, en termes de classes et de relations entre ces classes.

La Classe

- Une classe représente la structure commune d'un ensemble d'objets.
- Une classe est représentée par un rectangle qui contient une chaîne de caractères correspondant au nom de la classe
  - Ce rectangle peut être séparé en trois parties (nom, attributs, opérations).
  - Le nom de la classe doit commencer par un caractère alphabétique et ne pas contenir le caractère '::'

La classe

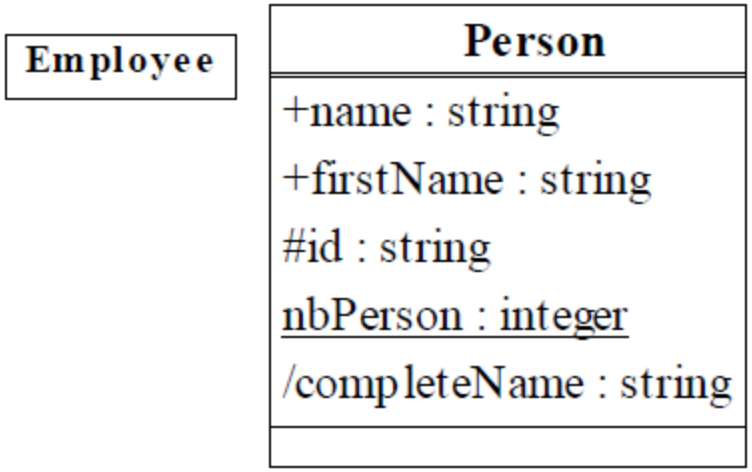
Identité + Etat + Comportement

- Une identité
  - deux objets différents ont des identités différentes
  - on peut désigner l'objet (y faire référence)
- Un état (attributs)
  - ensemble de propriétés/caractéristiques définies par des valeurs
  - permet de le personnaliser/distinguer des autres objets
  - peut évoluer dans le temps
- Un comportement (méthodes)
  - ensemble des traitements que peut accomplir un objet (ou que l'on peut lui faire accomplir)

La classe

Objet	États	Comportements
Chien	Nom, race, âge, couleur	Aboyer, chercher le baton, mordre, faire le beau
Compte	N°, type, solde, ligne de crédit	Retrait, virement, dépôt, consultation du solde
Téléphone	N°, marque, sonnerie, répertoire, opérateur	Appeler, Prendre un appel, Envoyer SMS, Charger
Voiture	Plaque, marque, couleur, vitesse	Tourner, accélérer, s'arrêter, faire le plein, klaxonner

CLASSES


Employee
Person
+name : string
+firstName : string
#id : string
nbPerson : integer
/completeName : string

Attributs

- Une classe peut contenir des attributs
- La syntaxe d'un attribut est :

  visibilité nom : type
- La visibilité est:
  - '+' pour public
  - '#' pour protected
  - '-' pour private
- UML définit son propre ensemble de types
  - Integer, real, string, …
- Un attribut peut être un attribut de classe, il est alors souligné.
- Un attribut peut être dérivé, il est alors préfixé par le caractère '/'

Attributs

Représentation d'un **attribut** :

visibilité nom : type [multiplicité] = valeur_initiale

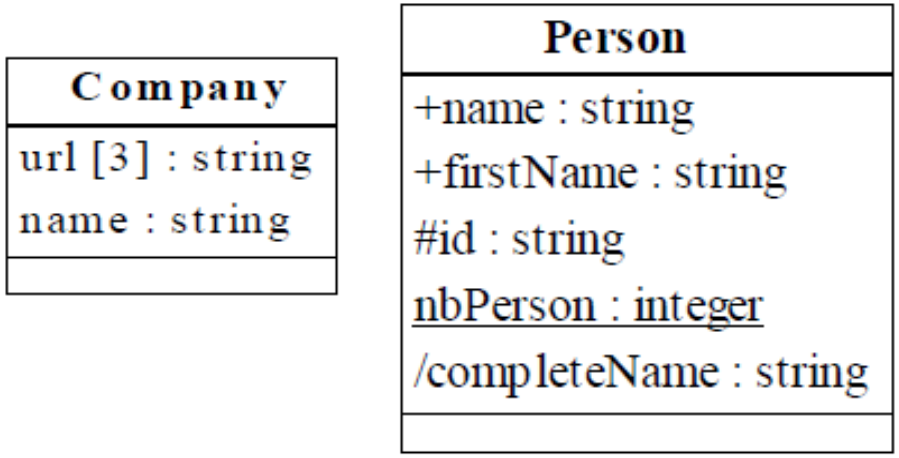
public +
privé -
protégé #

facultatif
mais impératif
pour l'implémentation

facultatif
par défaut: 1
[0..1] élément pouvant être nul
[4] tableau de 4 éléments
[2..*] tableau dynamique
d'au moins 2 éléments

facultatif

Attributs


Company
url [3] : string
name : string
Person
+name : string
+firstName : string
#id : string
nbPerson : integer
/completeName : string

Opérations

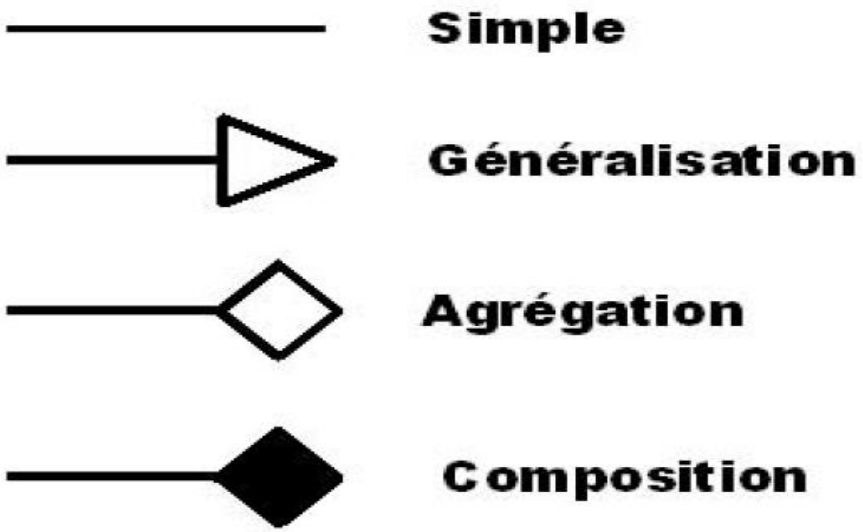
- Une opération est un service qu'une instance de la classe peut exécuter
- La syntaxe d'une opération est :
  - *visibility name(parameter):return*
- La syntaxe des paramètres est :
  - *kind name : type*
- Le kind peut être :
  - in, out, inout

Opérations

Company
url [3] : string name : string
+makeProfit():real +getWorkingEmployee(): [*] Employee

Employee
+stopWork():boolean +startWork(In work:string):boolean

Types d'Association

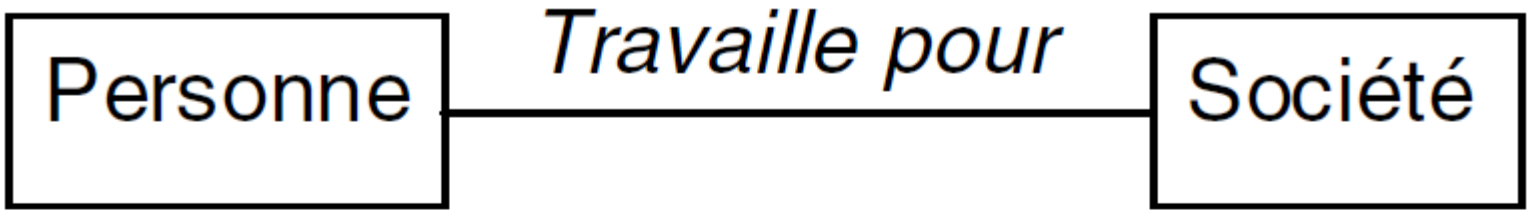


Associations

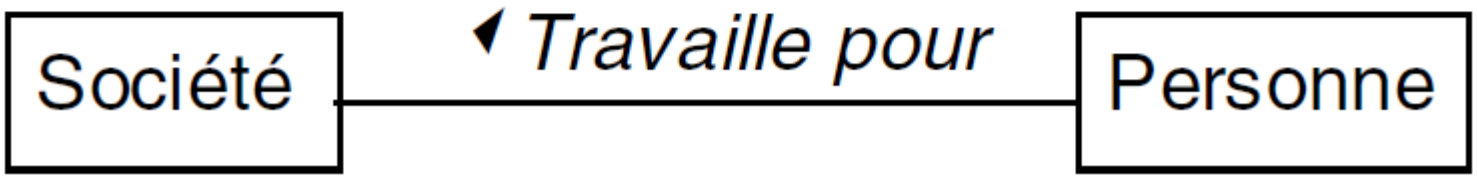
- Les associations binaires connectent deux éléments entre eux
- Une association binaire est composée de deux associations.
- Une association end est paramétrée par:
  - Un nom (le rôle joué par l'entité connectée)
  - Un genre d'agrégation (composite, agregation, none)
  - De plusieurs propriétés: isNavigable, isChangeable

1	Un et un seul
0..1	Zéro ou un
M..N	De M à N (entiers naturels)
*	De zéro à plusieurs
0..*	De zéro à plusieurs
1..*	De un à plusieurs
N	Exactement N (entier naturel >0)

Nommage des associations

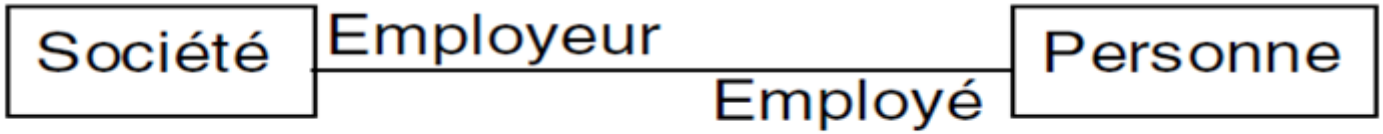


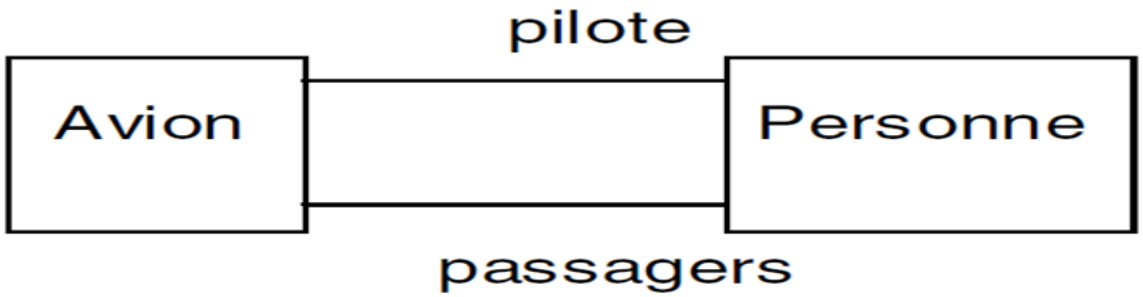
On peut ajouter un sens de lecture :



Noms d'extrémité d'association - Rôle

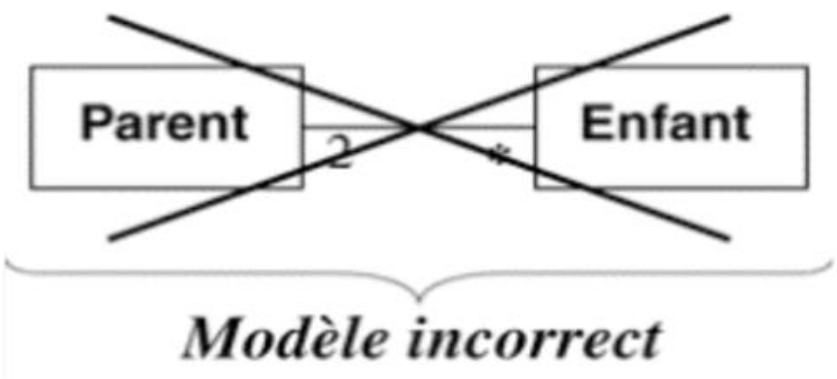
- Chaque **extrémité d'association** peut être nommée.

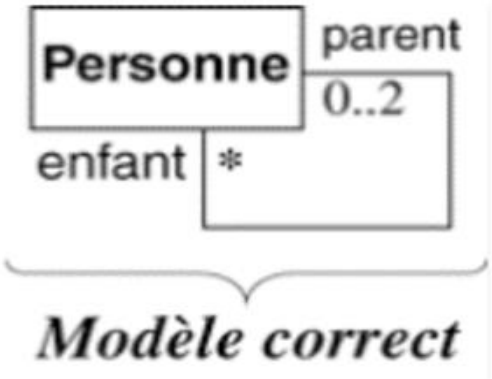




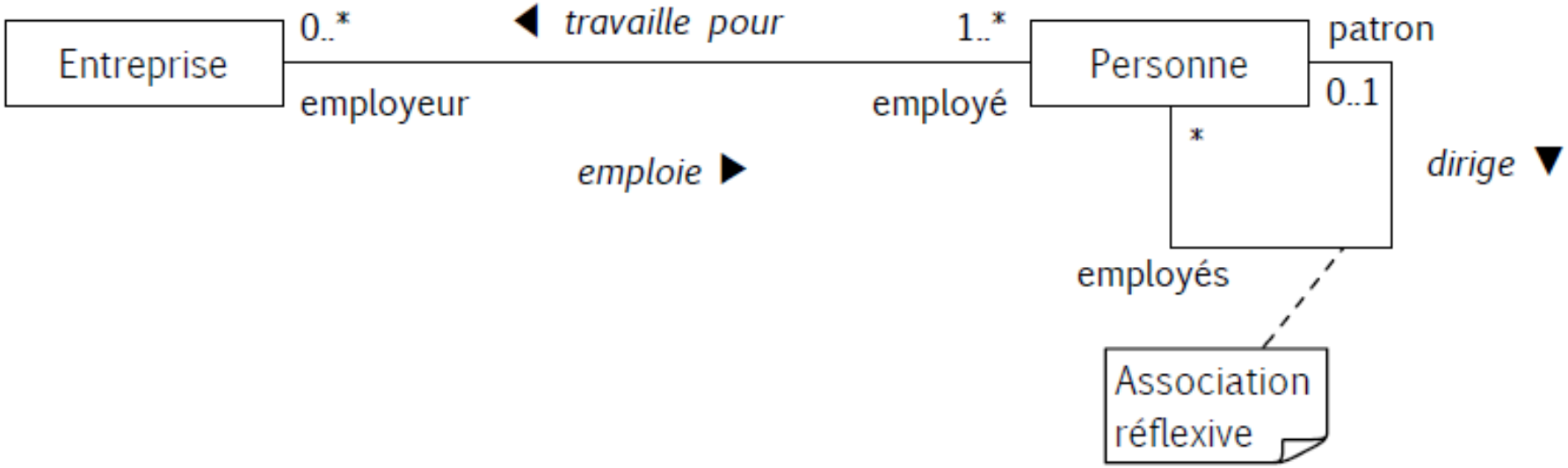
Noms d'extrémité d'association - Rôle

- Nommez les extrémités pour modéliser plusieurs références à la même classe.





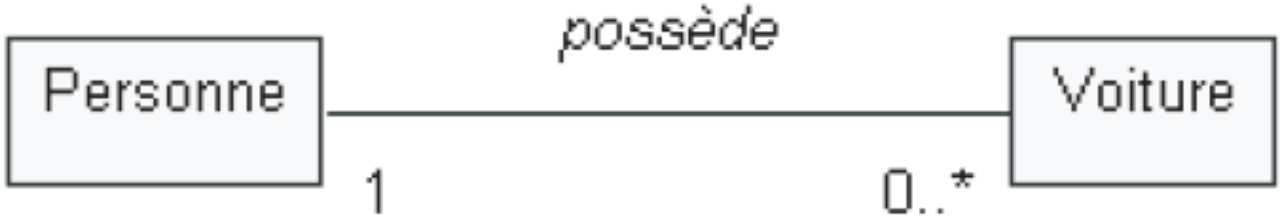
Exemple récapitulatif



Multiplicité des associations

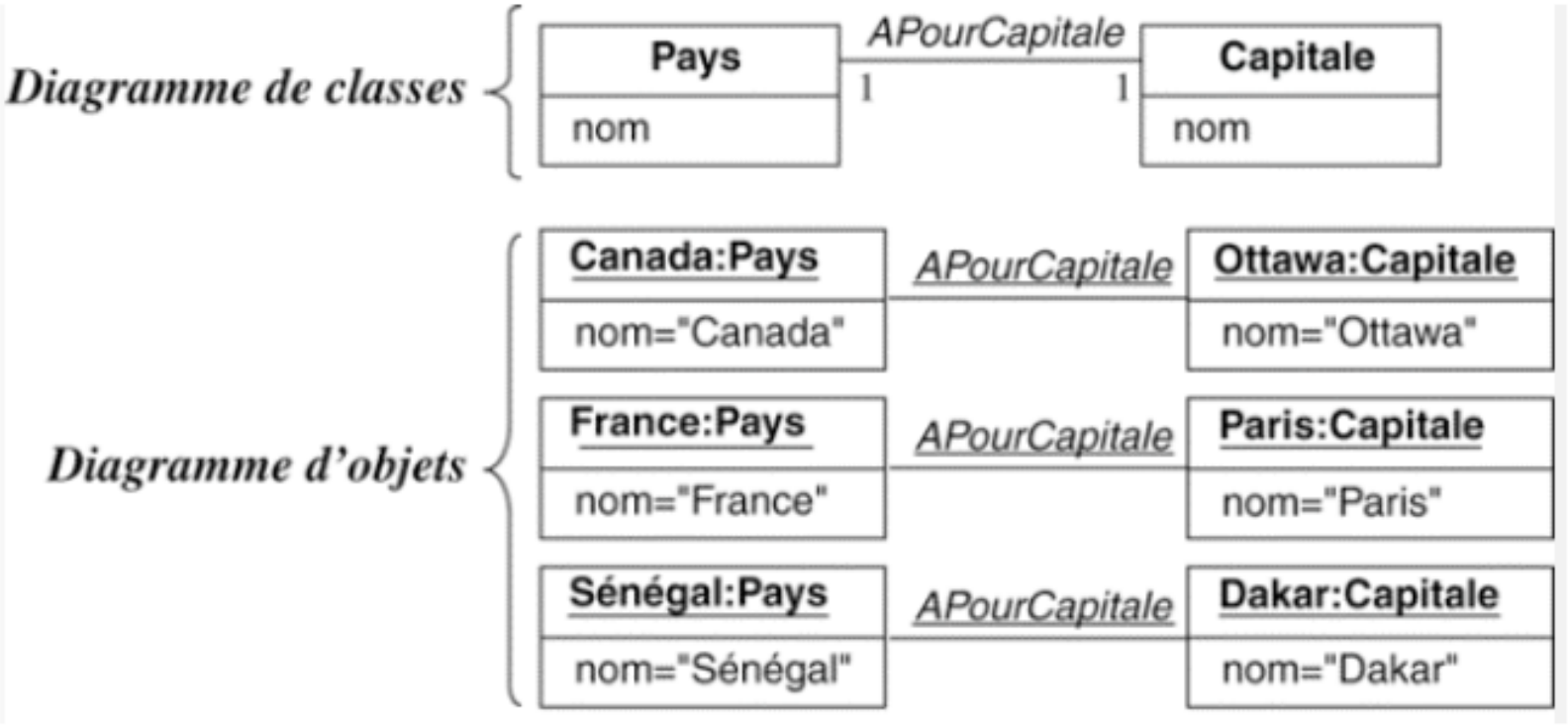
- La multiplicité spécifie le nombre d'instances d'une classe pouvant être liées à une seule instance d'une classe associée. Elle contraint le nombre d'objets liés.

- Exemple : une personne peut posséder plusieurs voitures (entre zéro et un nombre quelconque) ; une voiture est possédée par une seule personne.



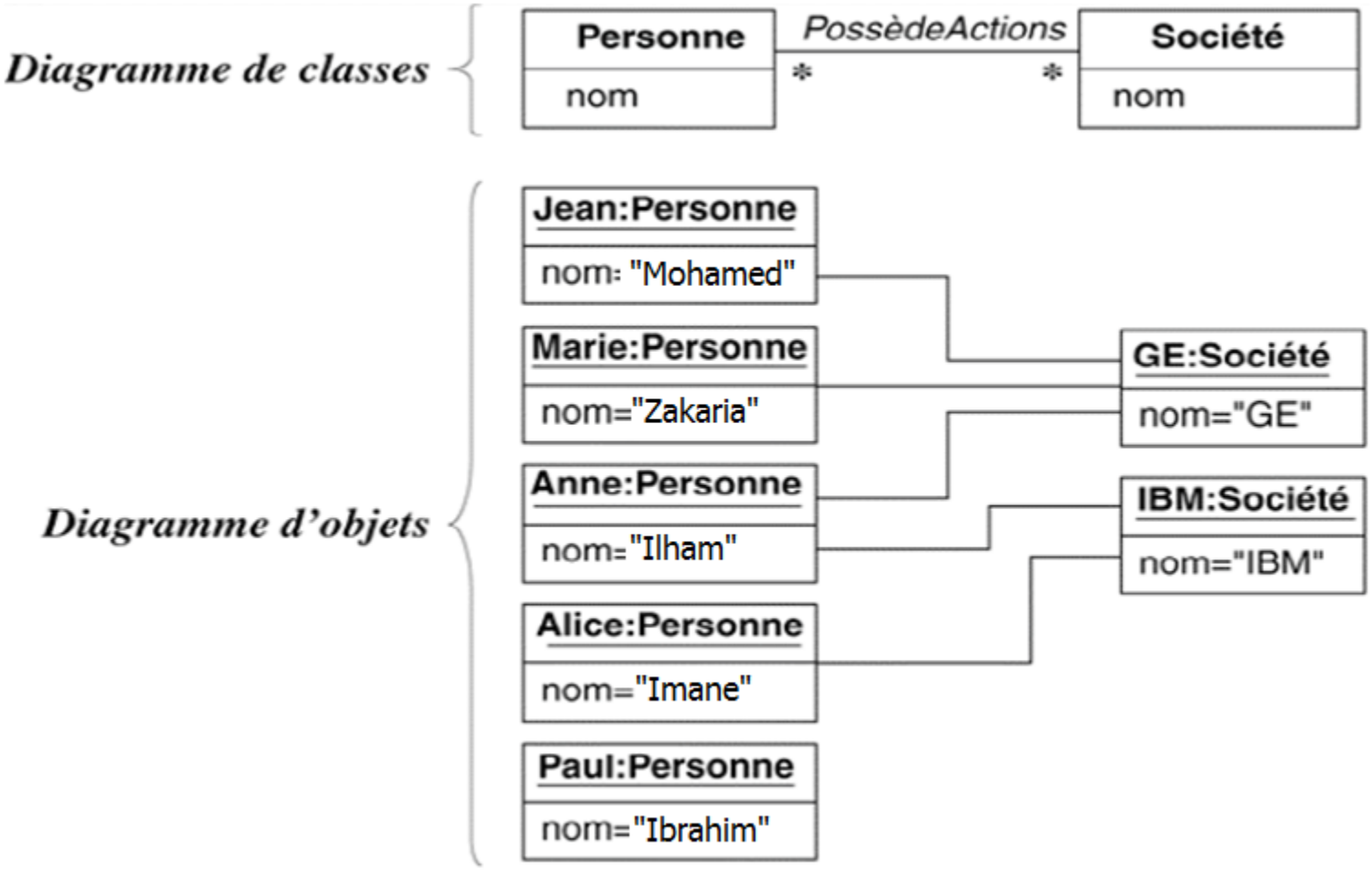
Multiplicité des associations

- Exemple :



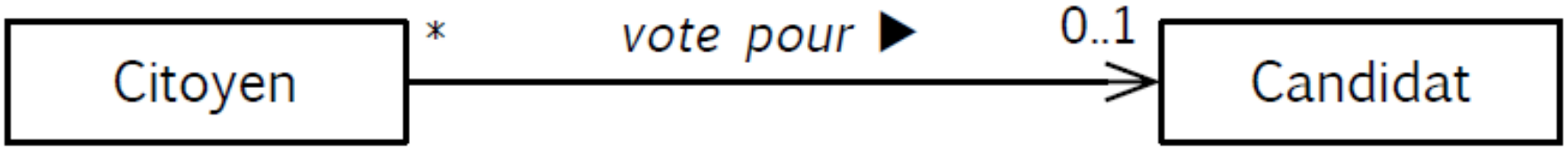
Multiplicité des associations

- Exemple :

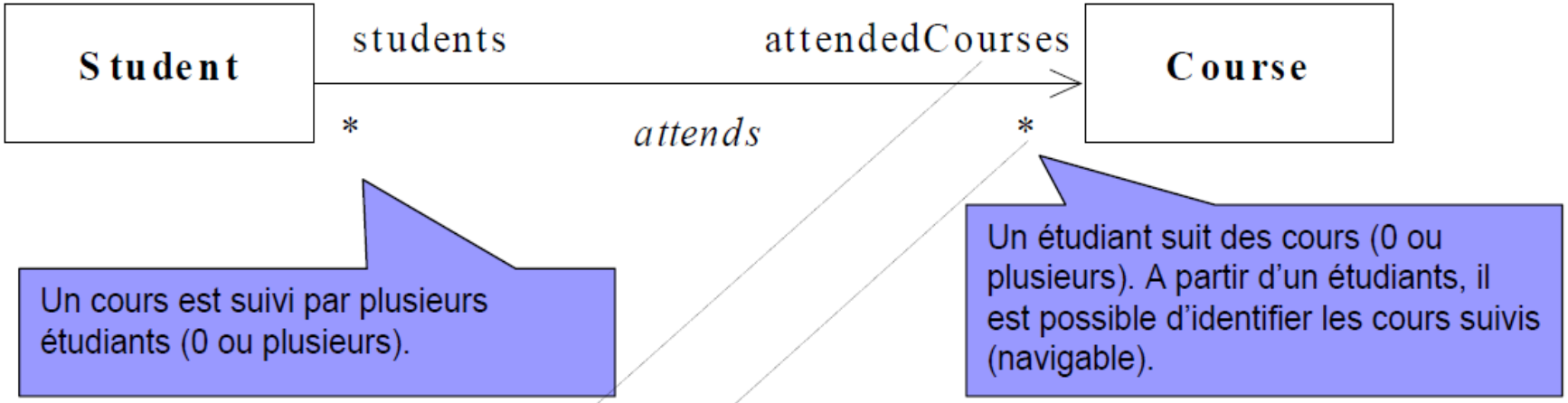


Navigabilité d'une association

- Par défaut, les associations sont navigables dans les deux directions.
- la navigation peut être restreinte à une seule ditrection : **les instances d'une classe ne "connaissent" pas les instances d'une autre**.
- On restreint la navigabilité d'une association à un seul sens à l'aide d'une flèche.



Associations




```
public class Student
{
   public Course attendedCourses[];

   public Student()
   {
   }
}
```

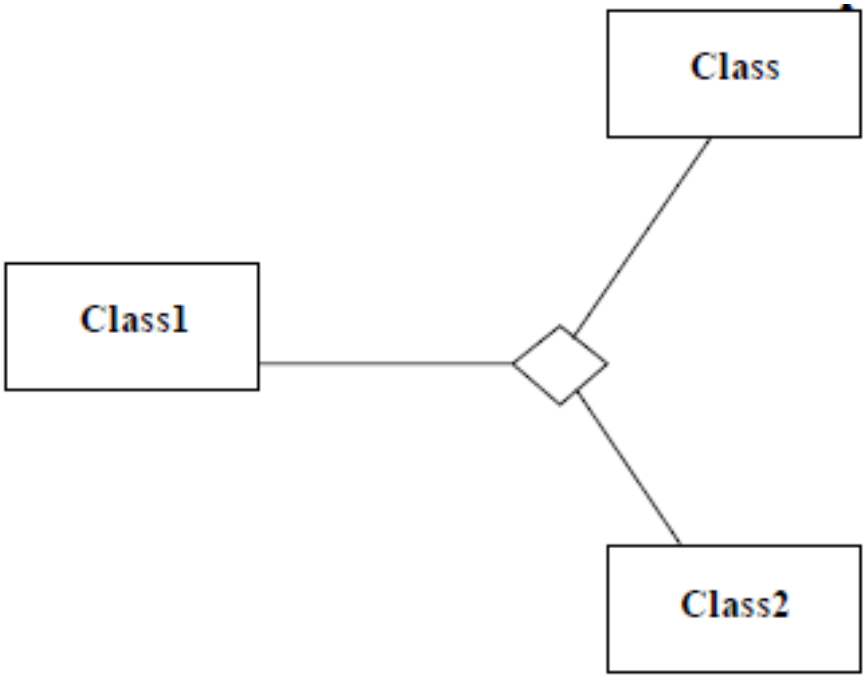
```
public class Course
{

   public Course()
   {

   }
}
```

Associations

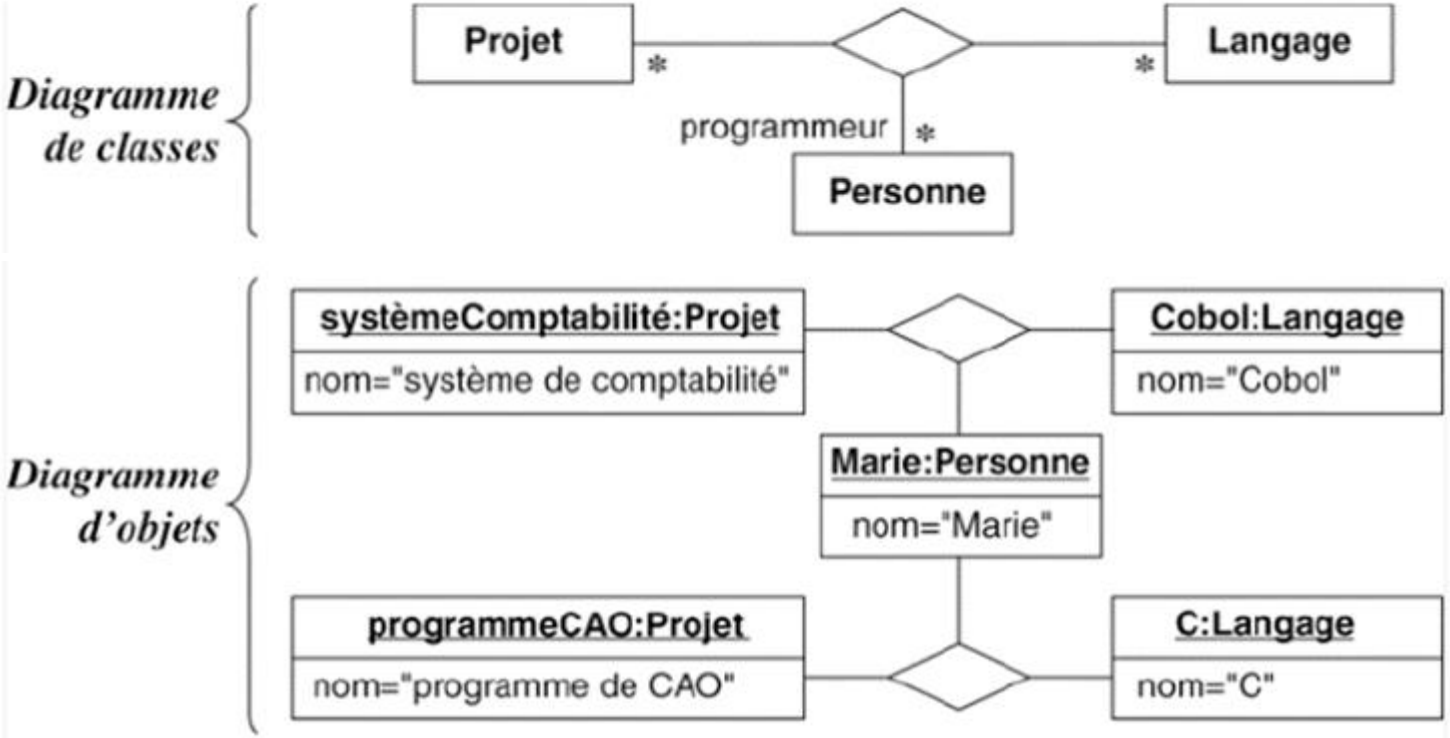
- Les associations N-aires connectent plusieurs éléments entre eux.
- Les associations N-aire sont très peu utilisées.



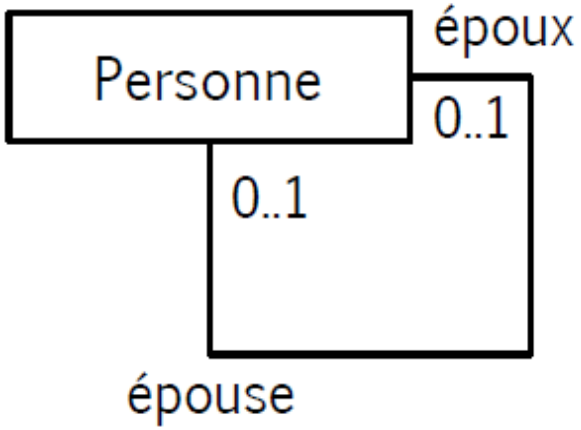
Associations N-aires

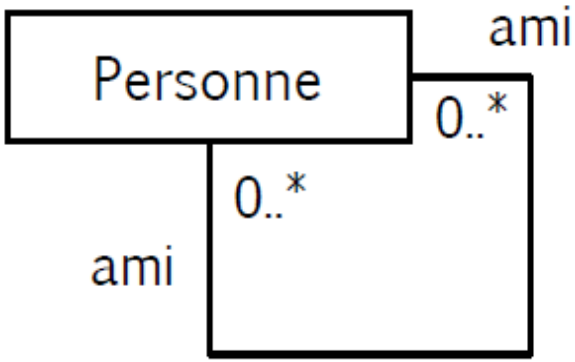
- En général, les associations sont binaires
- **N-aires** : au moins trois instances impliquées

A n'utiliser que lorsqu'aucune autre solution n'est possible !



Les associations réflexives





Agrégation

- Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance d'un élément dans un ensemble.
- Les agrégations n'ont pas besoin d'être nommées : implicitement elles signifient « contient », « est composé de ».
- On représente l'agrégation par l'ajout d'un losange vide du côté de l'agrégat (l'ensemble).

Agrégation - Exemples

Composition

Une composition est une agrégation plus forte impliquant que :

- un élément ne peut appartenir qu'à un seul agrégat composite (agrégation non partagée).
- la destruction de l'agrégat composite (l'ensemble) entraîne la destruction de tous ses éléments (les parties).
- le composite est responsable du cycle de vie des parties.

Composition - Exemples

- Une partie constituante ne peut appartenir à plus d'un assemblage ;
- Une fois une partie constituante affectée à un assemblage, sa durée de vie coïncide avec ce dernier.

Agrégation vs. composition

Quand mettre une composition plutôt qu'une agrégation ?

- Pour décider de mettre une composition plutôt qu'une agrégation, on doit se poser les questions suivantes :
  - Est-ce que la destruction de l'objet composite (du tout) implique nécessairement la destruction des objets composants (les parties) ? C'est le cas si les composants n'ont pas d'autonomie vis-à-vis des composites.
  - Lorsque l'on copie le composite, doit-on aussi copier les composants, ou est-ce qu'on peut les «réutiliser», auquel cas un composant peut faire partie de plusieurs composites ?

- Si on répond par l'affirmative à ces deux questions, on doit utiliser une composition.

Associations

```
public class School  //En Java
{
   public Student theStudent[];

   public School(){ }
}
```

Composition

School

has

Department

1

1..*

Aggrégation

1..*

member

*

```
public class Department
{
public Department() {}
}
```

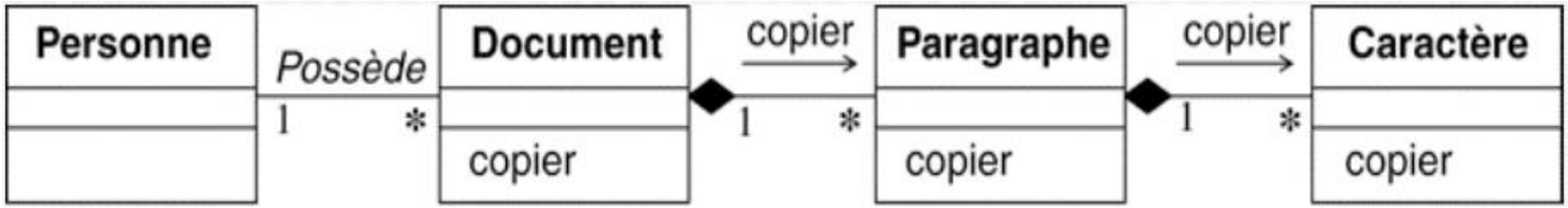
Student

```
public class Student
{
 public Student(){}
}
```

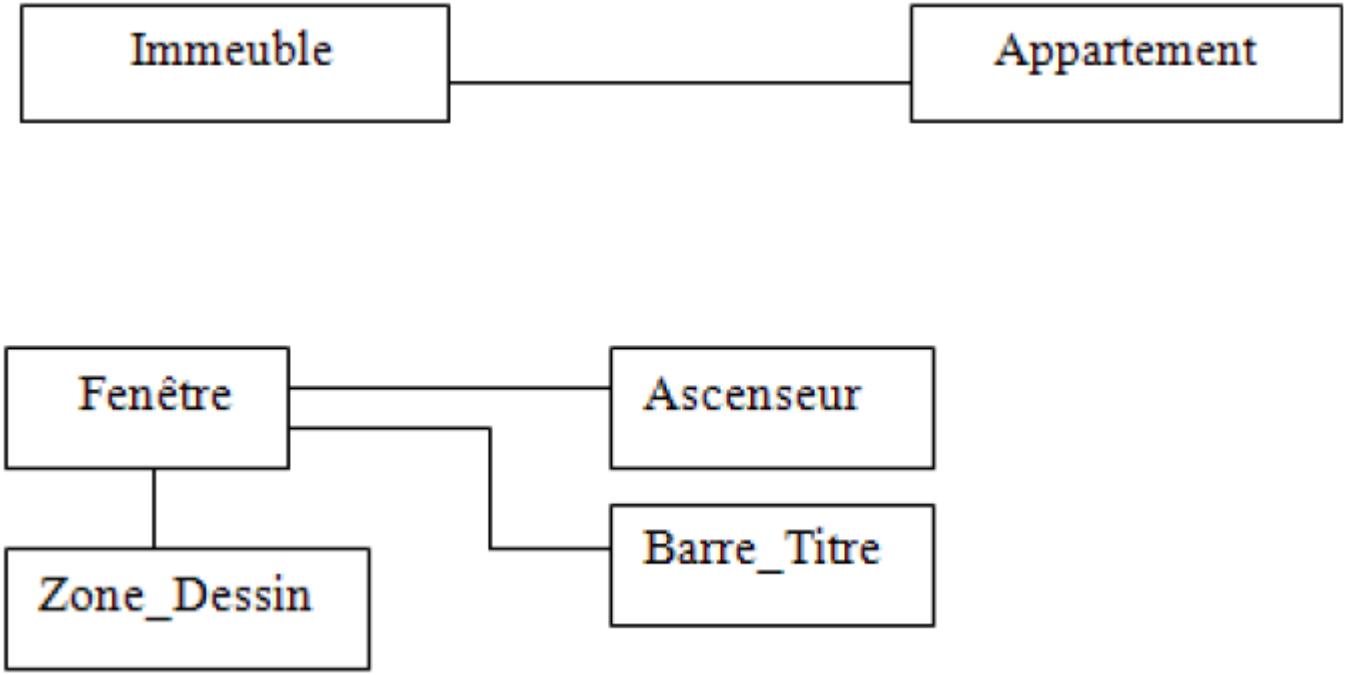
Propagation (ou déclenchement)

- Application automatique d'une opération à un réseau d'objets à partir d'un objet initial quelconque.

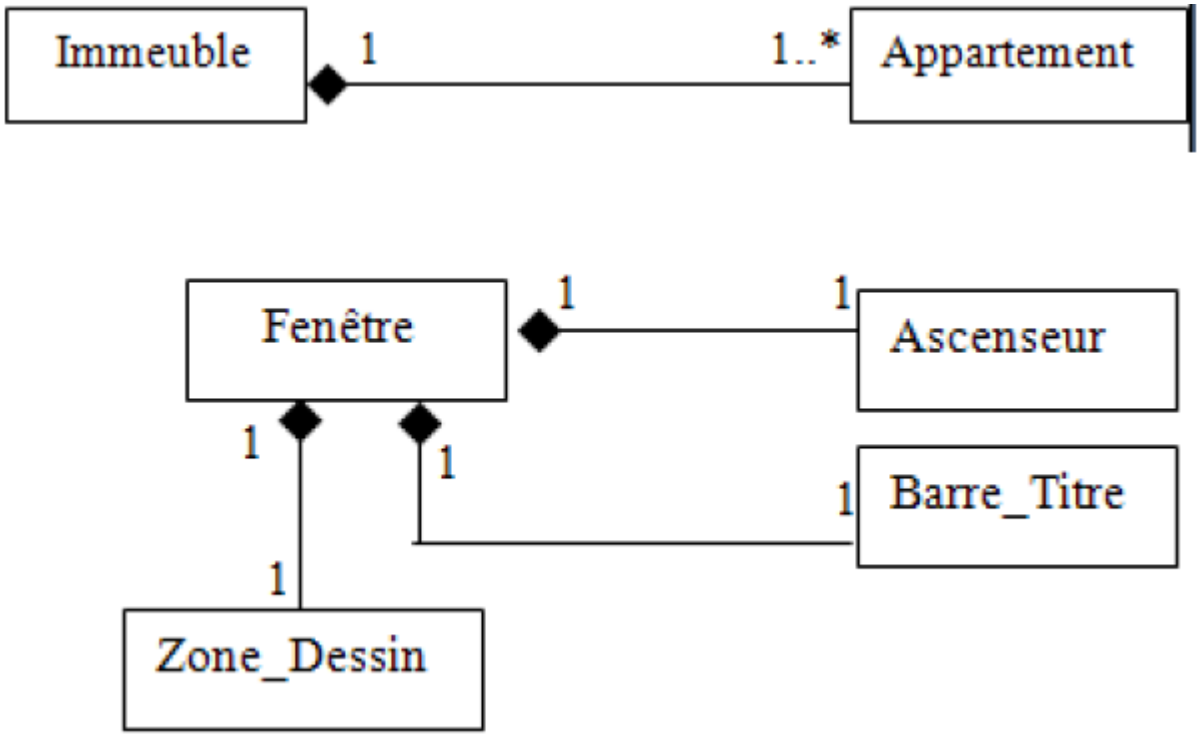
- L'agrégation permet un mécanisme de délégation d'opérations : l'opération *Document.copier*() peut être déléguée à l'opération *Paragraphe.copier*() en l'appliquant à toutes les instances de paragraphe qui composent le document. Celle-ci peut à son tour être déléguée dans les mêmes conditions à *Caractère.copier*().



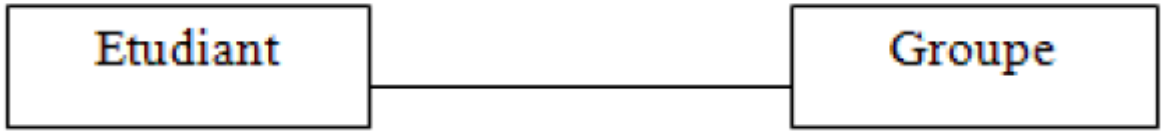
Agrégation vs. composition

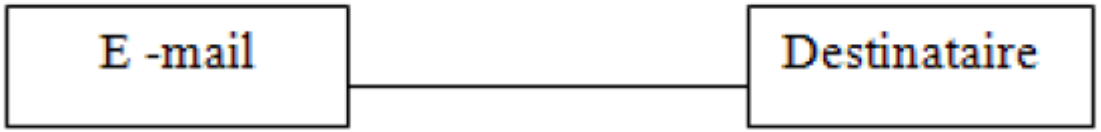


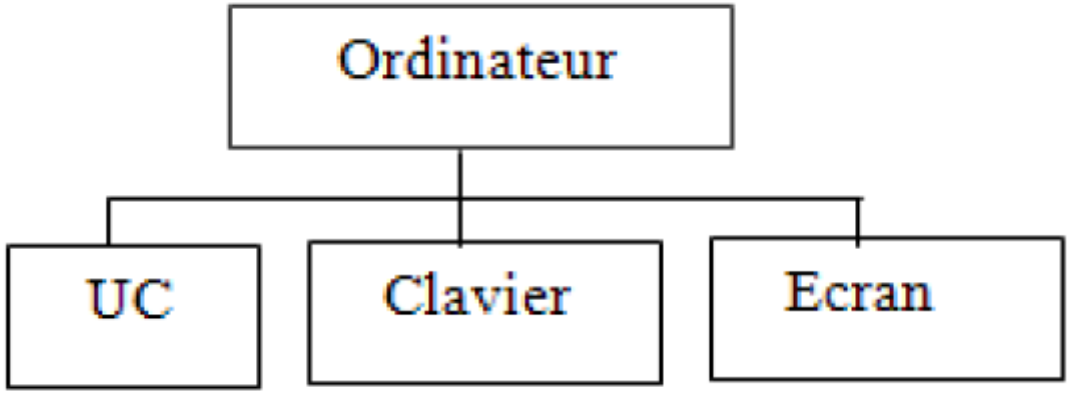
Agrégation vs. composition



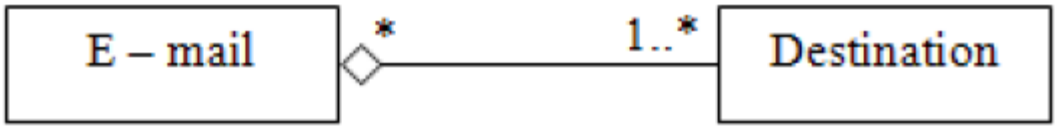
Agrégation vs. composition

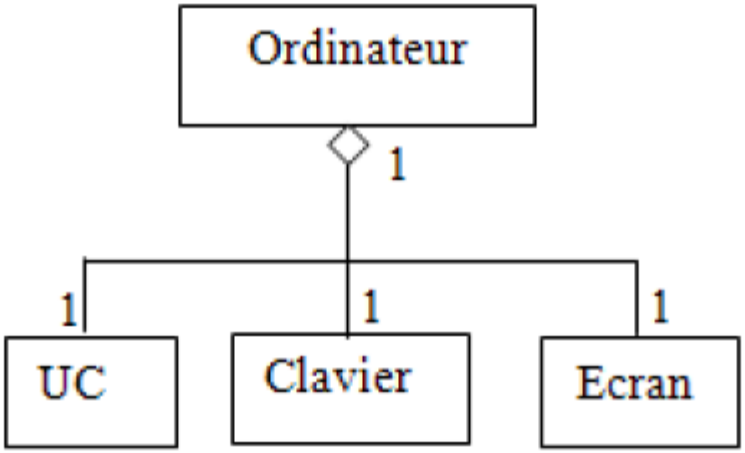






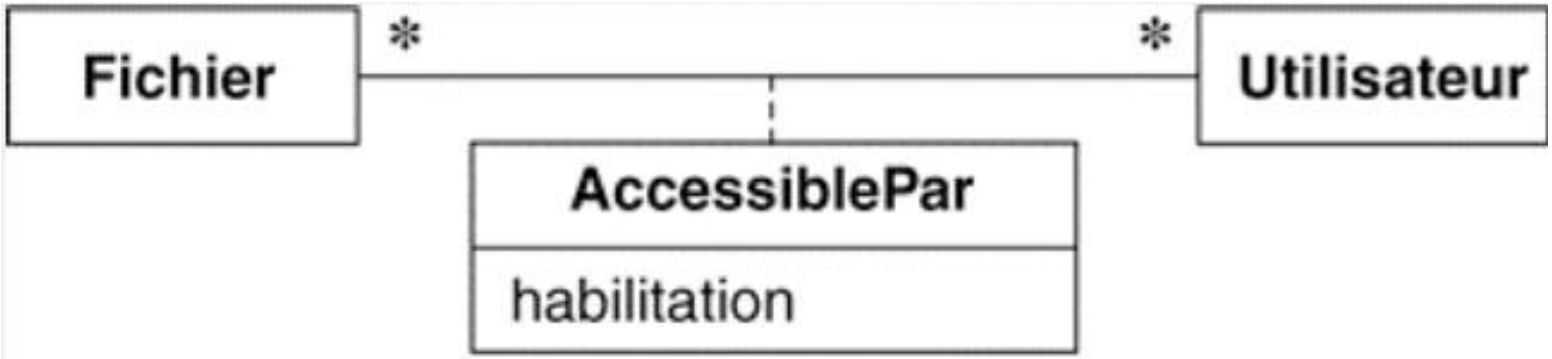
Agrégation vs. composition





Classe d'Association

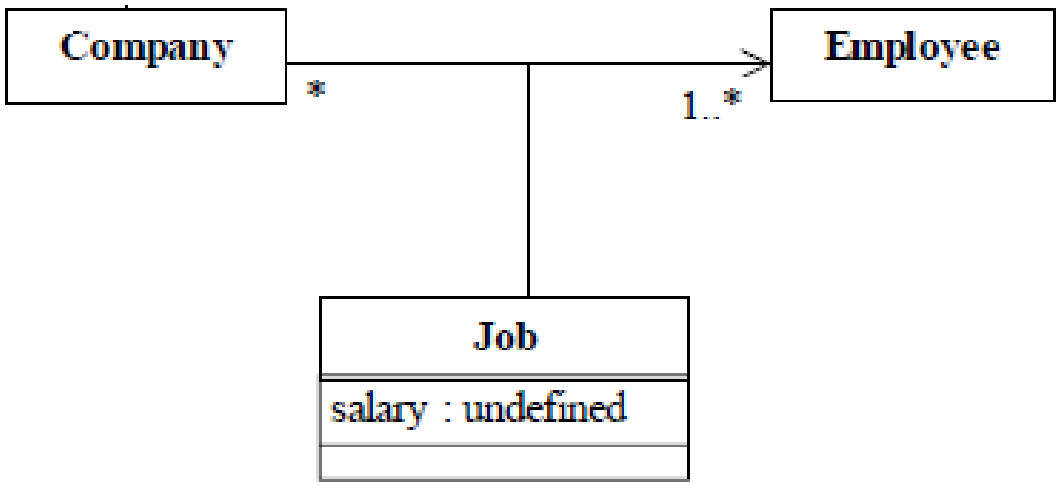
- association qui est aussi une classe.



/etc/termcap	lecture	Paul Lenoir
/usr/doe/.login	lecture/écriture	Marie Lebrun
/usr/lenoir/.login	lecture/écriture	Paul Lenoir

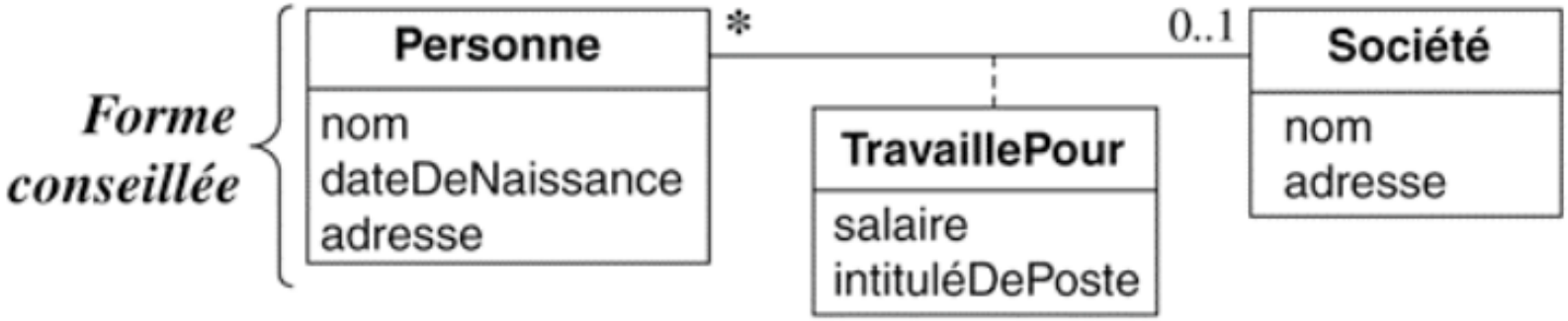
Classes - Associations

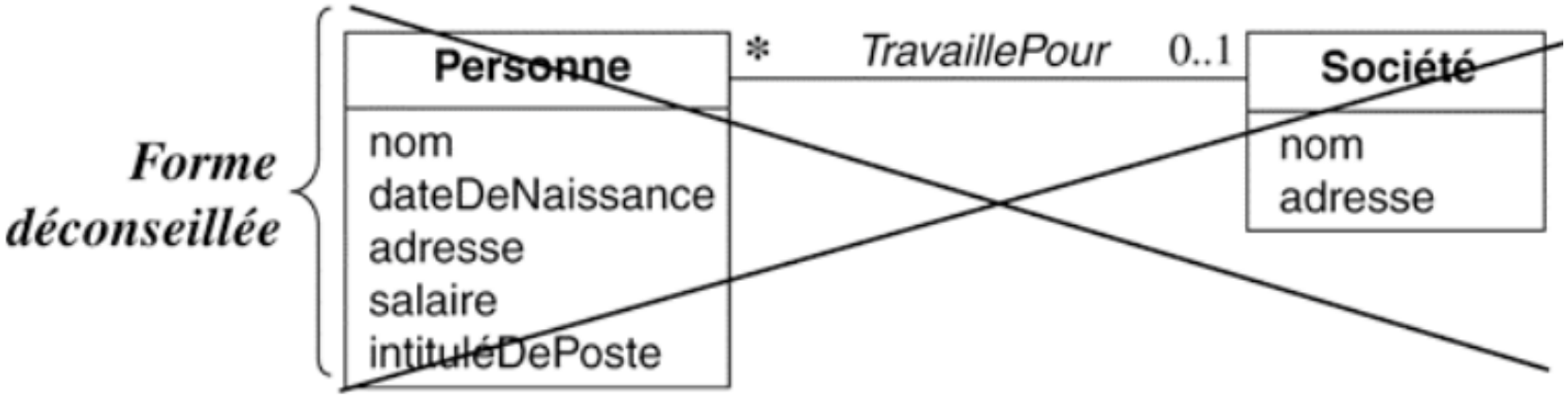
- Une classe-association est une association qui est aussi une classe.
- Les classes-associations sont utilisées lorsque les associations doivent porter des informations
- Il est toujours possible de se passer des classes-associations.



Classe d'Association

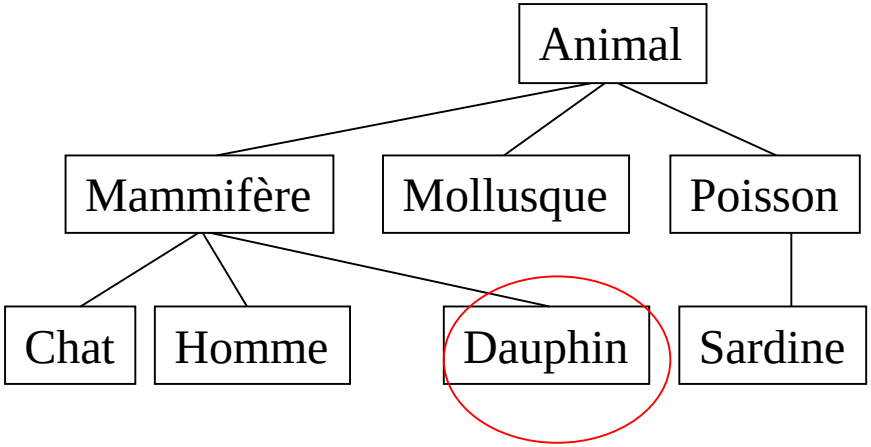
- Ne placez pas les attributs d'une association dans une classe.





Concepts de base

- **Héritage** : relation de spécialisation d'une classe
  - Une classe fille hérite d'une classe parente
  - Hérite des données et des méthodes de son parent
  - Mot clé : **extends**



Pas d'héritage multiple en Java

Concepts de base

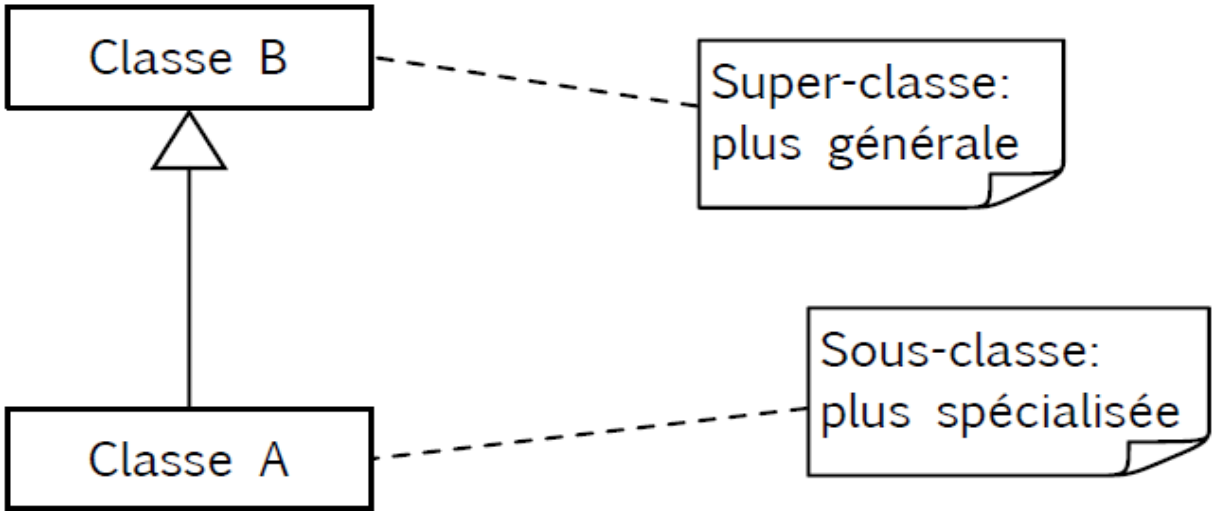
- **Interface** : définition abstraite d'une classe
  - Ne possède que des méthodes
  - Découple la définition d'une classe de son implantation
  - Lien avec la notion d'API
  - Une interface peut être implantée (mot clé implements) par plusieurs classe
  - Une classe peut posséder plusieurs interfaces

- **Interface : notion fondamentale pour séparer client et serveur**

Héritage (Généralisation / Spécialisation)

- **L'héritage** une relation de **spécialisation/généralisation**.

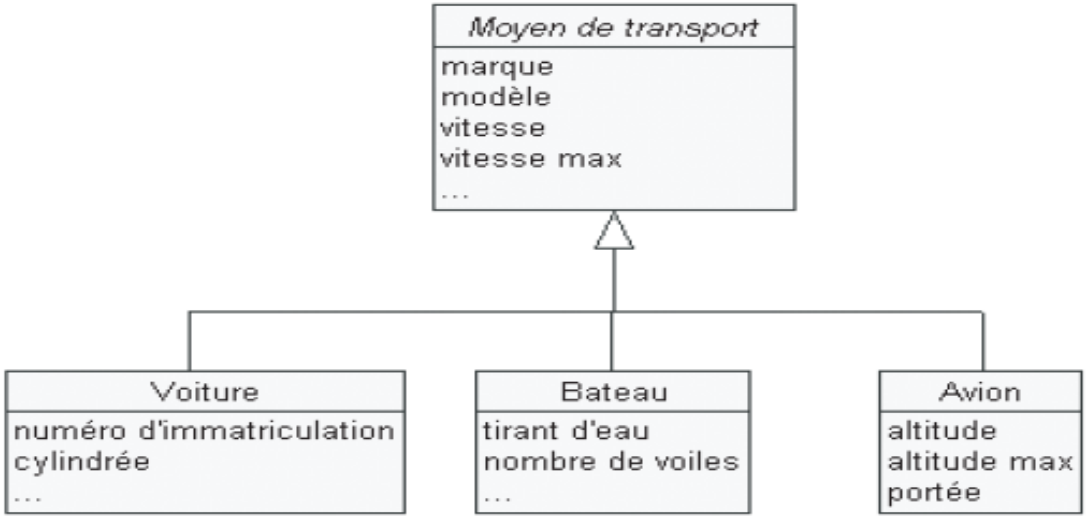
- Les éléments spécialisés héritent de la structure et du comportement des éléments plus généraux (attributs et opérations).



Héritage

Pour que ça fonctionne :

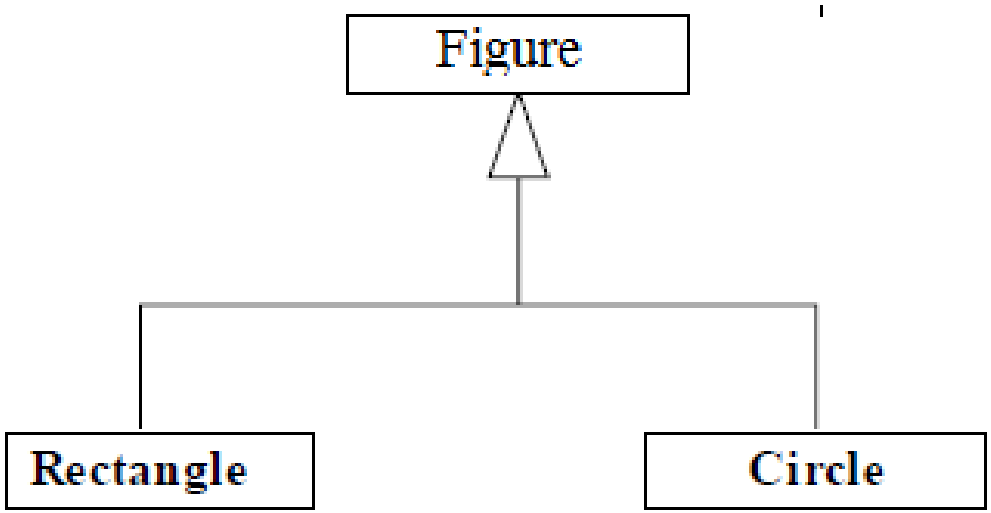
- Principe de substitution : toutes les propriétés de la classe parent doivent être valables pour les classes enfant.
- Principe du « A est un B » ou « A est une sorte de B » : toutes les instances de la sous-classe sont aussi instances de la super-classe. Par exemple, toute opération acceptant un objet d'une classe Animal doit accepter tout objet de la classe Chat (l'inverse n'est pas toujours vrai).



Héritage

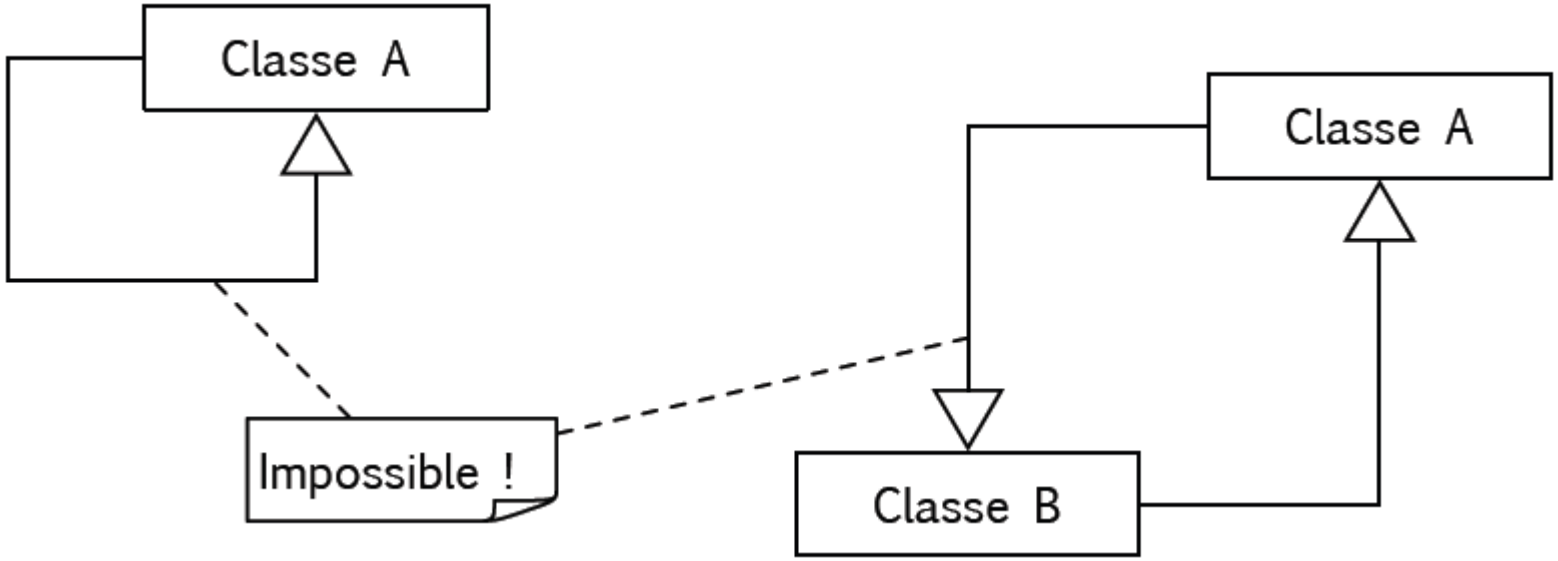
- L'héritage est une relation entre un élément plus général et un élément plus spécifique. L'héritage existe entre des classes, des packages, …

- L'héritage multiple est possible en UML



Héritage

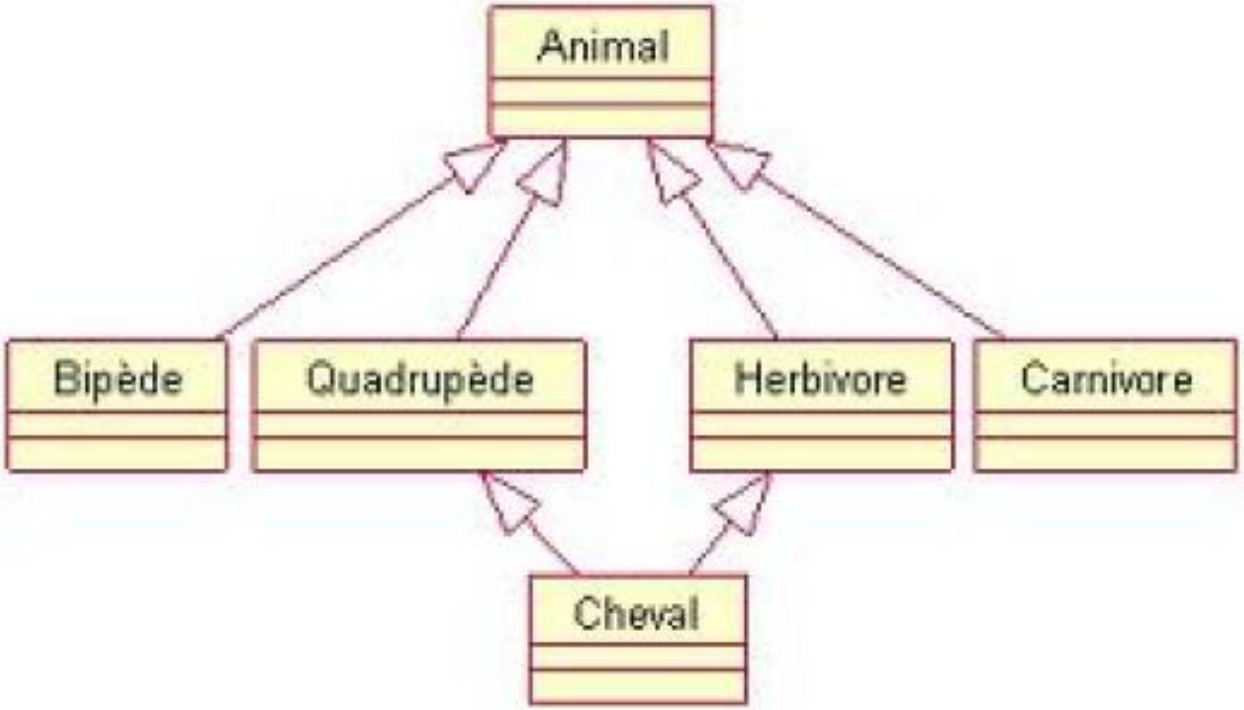
- Relation non-réflexive, non-symétrique !



Héritage Multiple

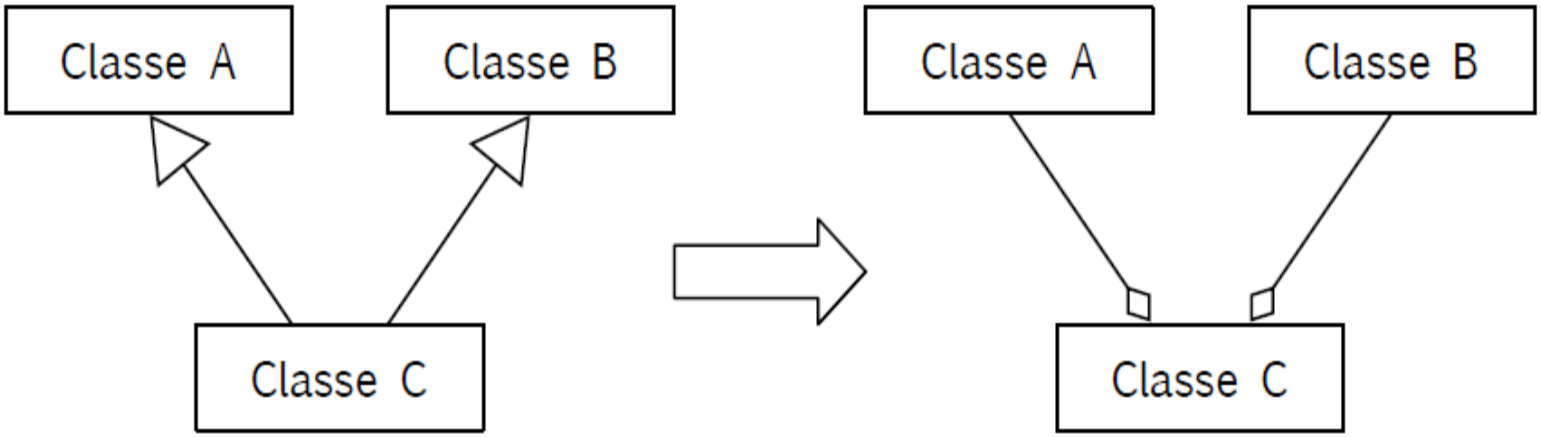
- Une classe peut avoir plusieurs classes parents. On parle alors d'héritage multiple.

- Exemple :



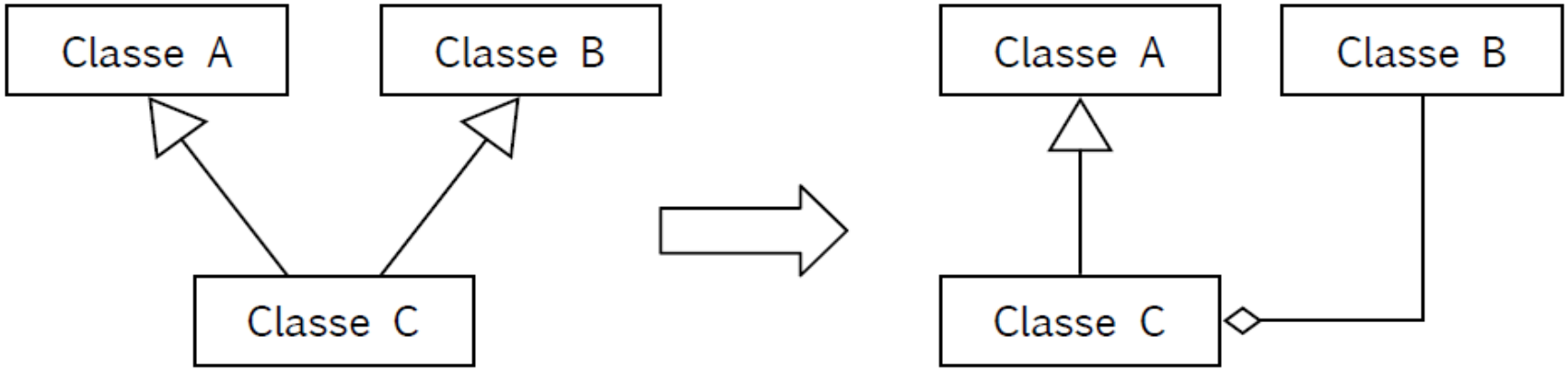
Héritage Multiple

- Comment éviter l'héritage multiple ?
- Première solution : déléguer



Héritage multiple

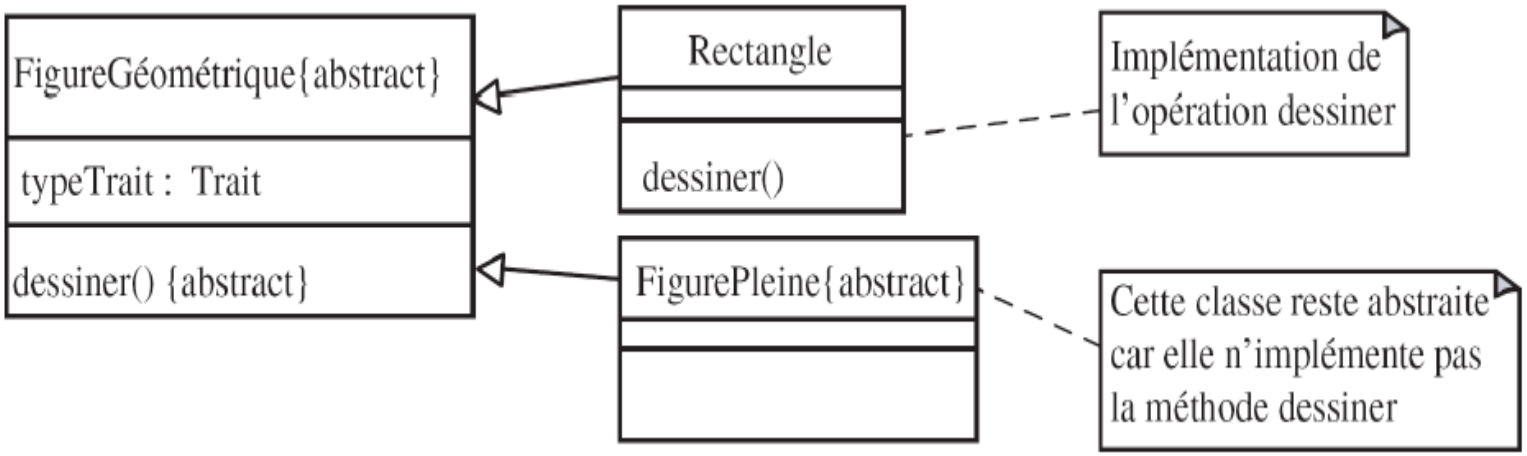
- Comment éviter l'héritage multiple ?
- Deuxième solution : hériter de la classe la plus importante et déléguer les autres



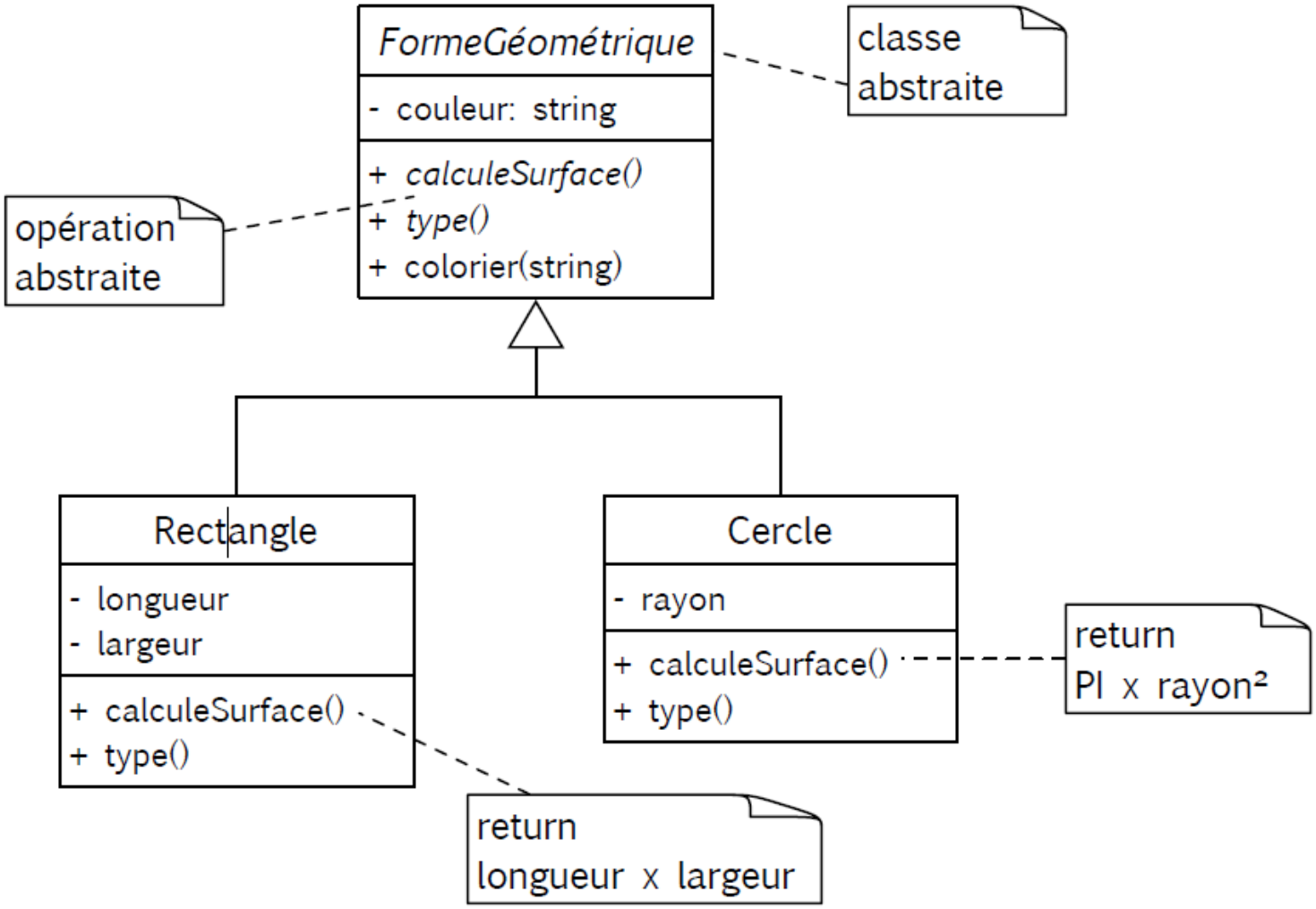
Classes abstraites

- Une **méthode** est dite **abstraite** lorsqu'on connaît son entête (signature) mais pas la manière dont elle peut être réalisée. Il appartient aux classes enfant de définir les méthodes abstraites.

- Une **classe** est dite **abstraite** lorsqu'elle définit **au moins** une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.

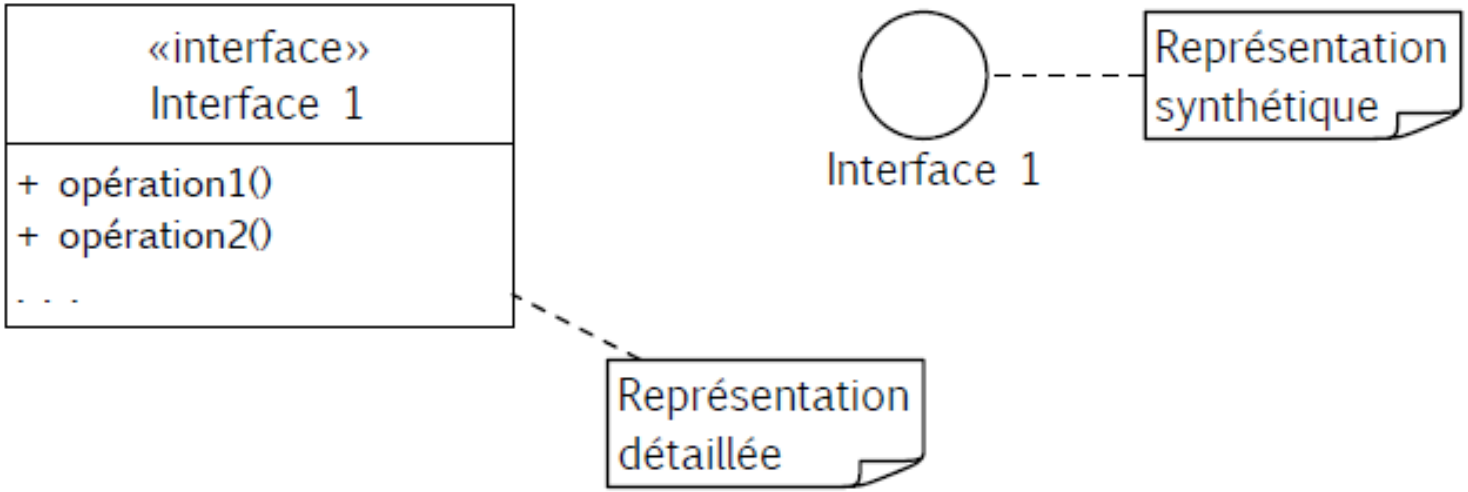


Classes abstraites - Exemple



Les interfaces

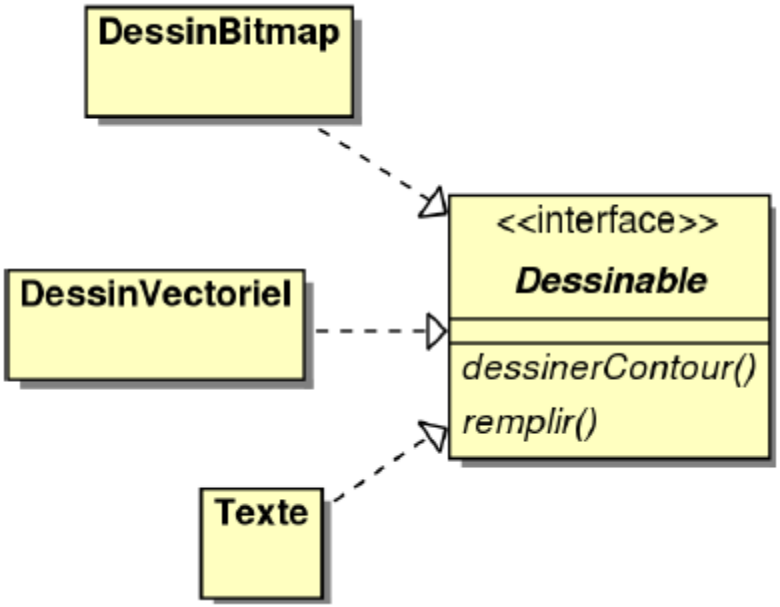
- Une interface spécifie un ensemble d'opérations (comportement)
- C'est un contrat :
  - Les classes liées s'engagent à respecter le contrat
  - elles doivent mettre en œuvre les opérations de l'interface



Les interfaces

- On utilise une relation de type réalisation entre une interface et une classe qui l'implémente.

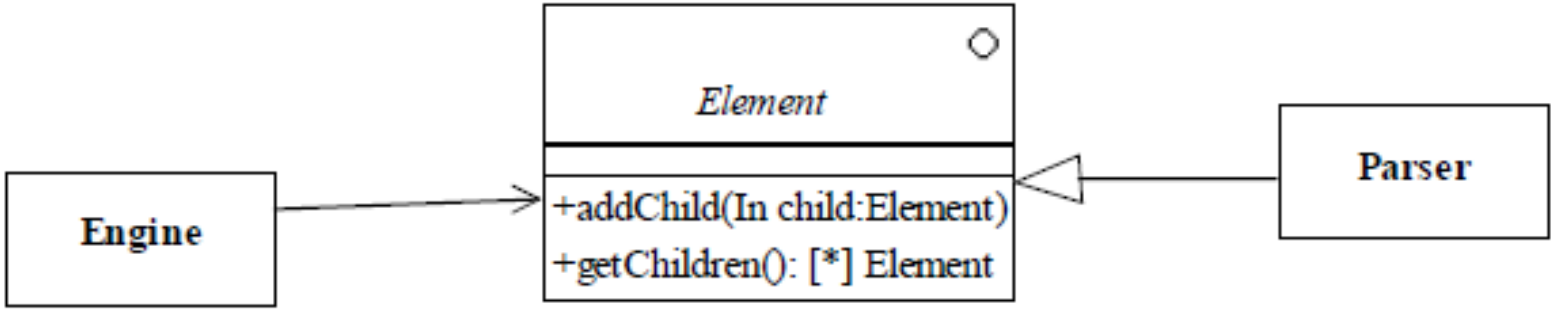
- Les classes implémentant une interface doivent implémenter toutes les opérations décrites dans l'interface

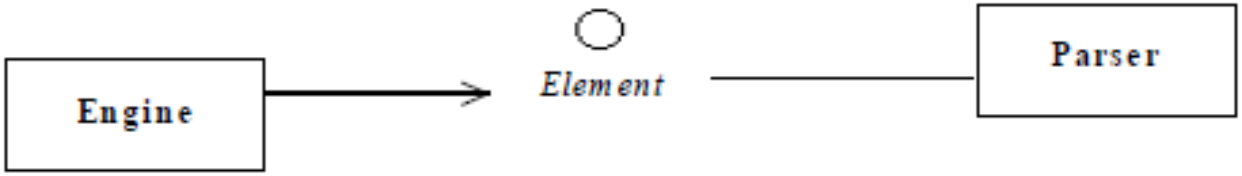


Interfaces

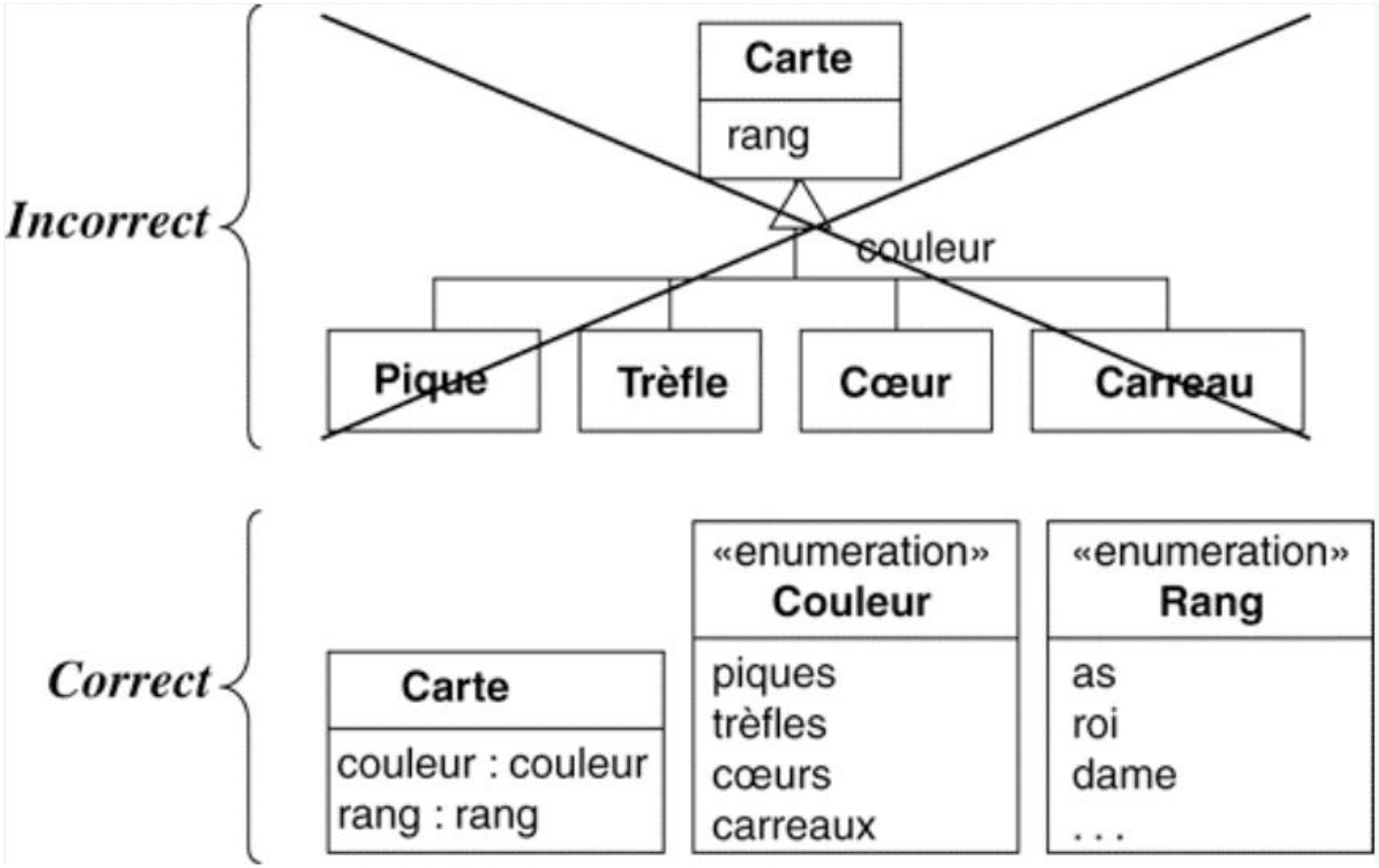
- Une interface est la spécification externe (en terme d'opérations) d'une classe.
- Une interface peut donc contenir des opérations
- Une classe réalise une interface si elle est capable d'exécuter toutes les opérations de l'interface
- On utilisera une relation de dépendance pour exprimer le fait qu'une classe est cliente d'une interface.

Interfaces


Element
+addChild(In child:Element)
+getChildren(): [*] Element
Engine
Parser


Engine
Element
Parser

Enumération



Etablir un diagramme de classes

- Réaliser un premier diagramme de classes qui contiendra l'ensemble des classes retenues et qui mettra en évidence des associations entre les classe. Les associations seront de simples traits banalisés : pas de nom, pas de multiplicité.

- Raffiner le diagramme précédent
  1. Renseigner les associations :
     - nom, multiplicité, agrégation, navigabilité, rôles des classes associées
  2. Renseigner les classes
     - attributs, classes associations, classes abstraites , héritage

- Vérifier que le diagramme permet de remplir les différents cas d'utilisation et leurs scénarii associés.

Exercice

- Donnez le code Java correspondant au diagramme de Classe suivant:

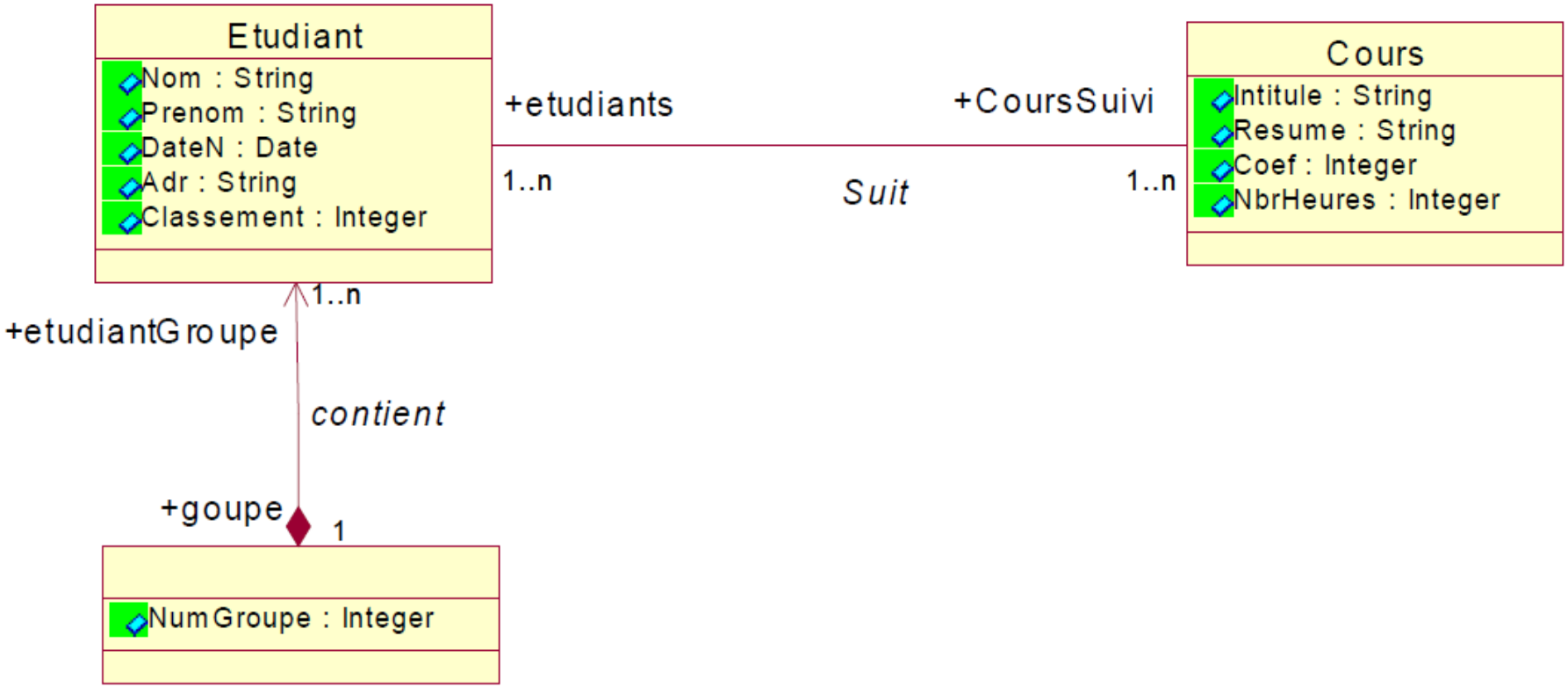


Diagramme de classe

- Les diagrammes de classes sont les diagrammes les plus utilisés
  - Ils permettent la décrire des programmes objet
  - Ils permettent de décrire le schéma logique de bases de données
  - Ils permettent de décrire des relations de concepts (modèle métier)

- Les diagrammes de classes peuvent être de différents niveaux d'abstraction

Conception de qualité

Pour qu'un logiciel soit extensible et réutilisable, il faut qu'il soit découpé en modules

- ***faiblement couplés*** : ainsi chaque entité peut être modifiée en limitant l'impact du changement au reste de l'application. et

- ***à forte cohésion*** *:* Il faut réunir ce qui est impacté par une même modification (« qui se ressemble s'assemble »)

- Comment peut-on réduire l’écart entre les besoins réels et les besoins exprimés dans un cahier de charge?
- Quels sont les quatre principes de l’orienté objets?
- Quelle est la différence entre la phase de validation et la phase de vérification?
- Soit l’application de l’université USTHB, modélisez à l’aide de diagramme de classe les étudiants, enseignants?