Cours #8 : Diagramme de Classes

Samia BOULKRINAT

(*Basé sur le cours de Assia HACHICHI*)

16/06/2021

Plan

1. Principes de l'Orienté Objet
2. Diagramme de classes

1. Principes de l'Orienté Objet

Quatre **principes de l'Orienté Objet** :

1. Classe et Objet
2. Encapsulation,
3. Héritage,
4. Polymorphisme…

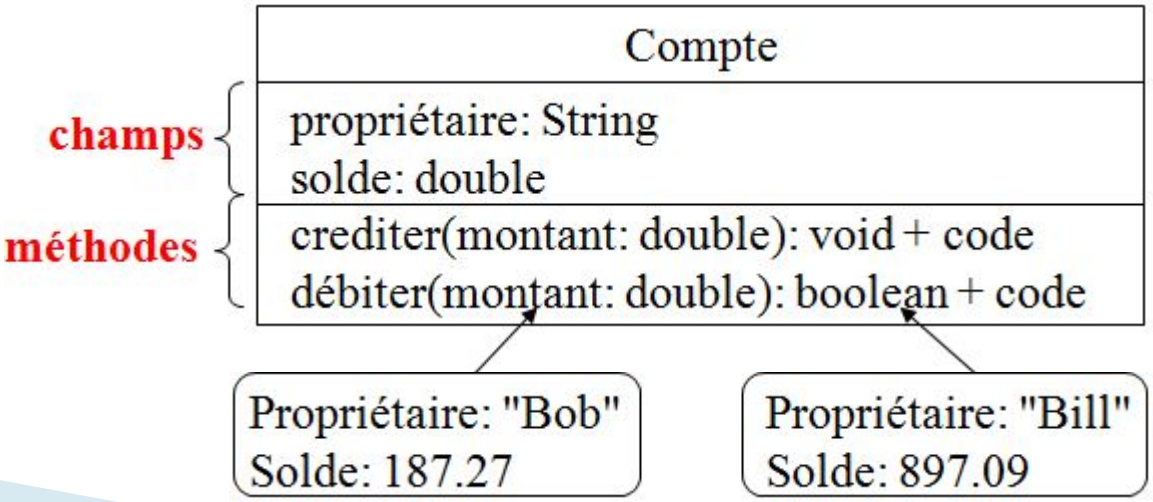
Classe vs Objet

- **La classe** : "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".

- **L'objet** : "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".

Classe vs 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

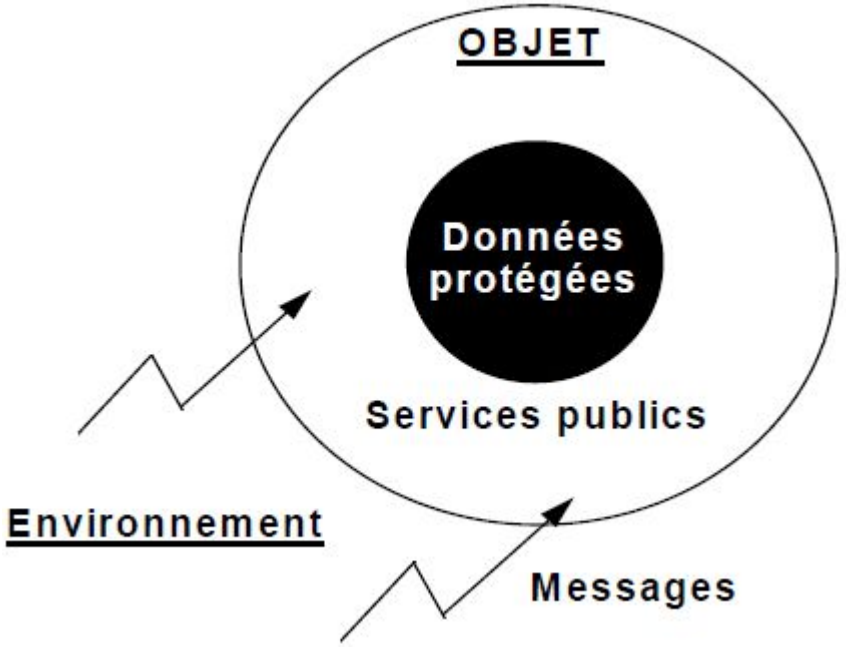


Classe vs Objet

Classe

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

Objet



Classe vs Objet

- 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

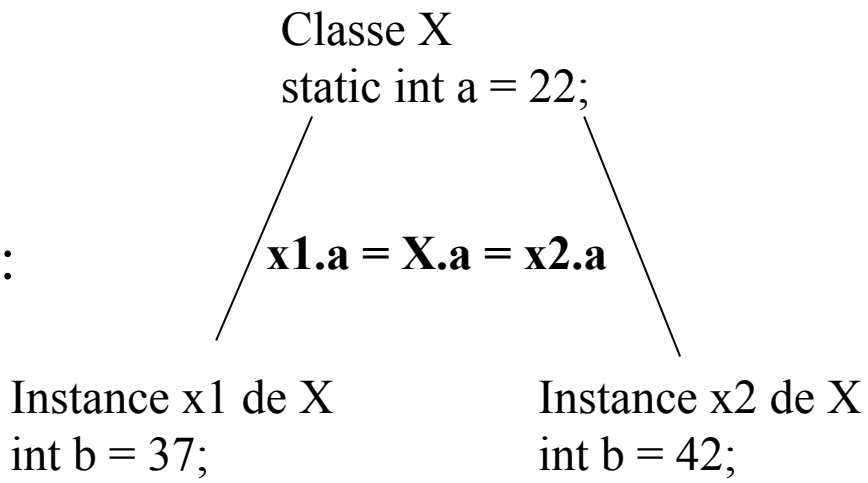
Classe vs Objet

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



- **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

Classe vs Objet

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;
            }}
```

Classe vs Objet

Exemple

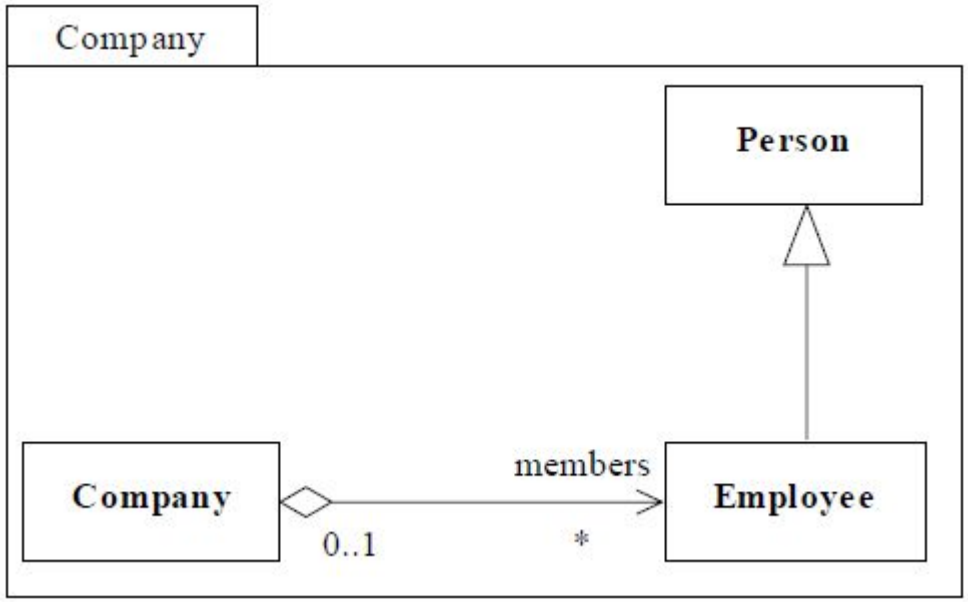
```
public class TestCompte {
    private static Compte bob;
    // static ⇒ champs partagé entre toutes les instances
    private static Compte bill;

    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);
    }
  }
```

2. 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.



2. Diagramme de classes

2.0 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.

2. Diagramme de classes

2.1 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 '::'

2. Diagramme de classes

2.1 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 (méthode) :

- Ensemble des traitements que peut accomplir un objet (ou que l'on peut lui faire accomplir)

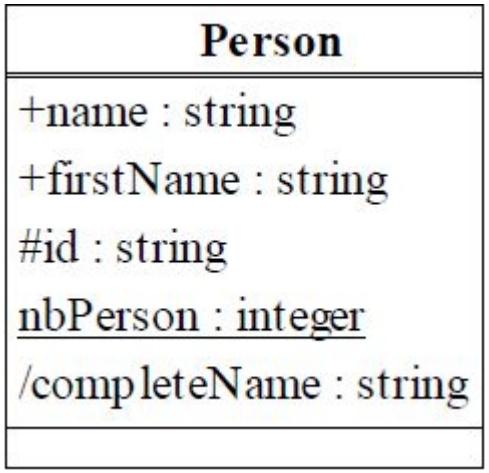
2. Diagramme de classes

2.1 Classe

Objet	Etats	Comportements
Chien	Nom, race, âge, couleur	Aboyer, chercher le bâton, mordre, faire le beau
Compte	N°, type, solde, ligne de crédit	Retrait, virement, dépôt, consultation du solde
Téléphone	N°, marque, sonnerie, répertoire, operateur	Appeler, prendre un appel, envoyer sms, charger
Voiture	Plaque, marque, couleur, vitesse	Tourner, accélérer, s'arrêter, faire le plein, klaxonner

2. Diagramme de classes

2.1 Classe



2. Diagramme de classes

2.2 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 de classe est souligné.
Un attribut peut être dérivé, préfixé par le '/'

2. Diagramme de classes

2.3 Représentation d'un Attribut

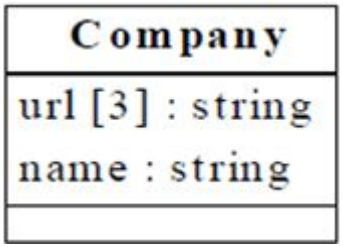
visibilité nom : type [multiplicité] = valeur_initiale

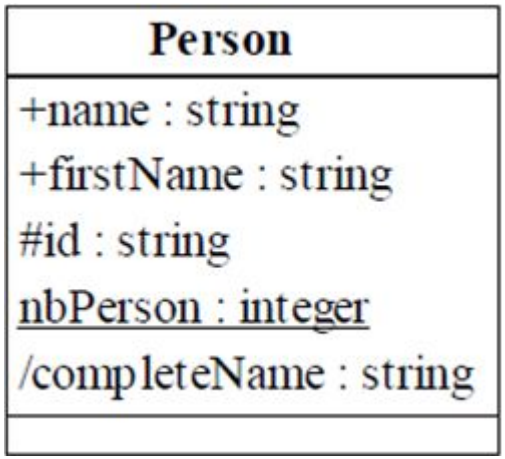
public +
privé -
protégé #

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





2. Diagramme de classes

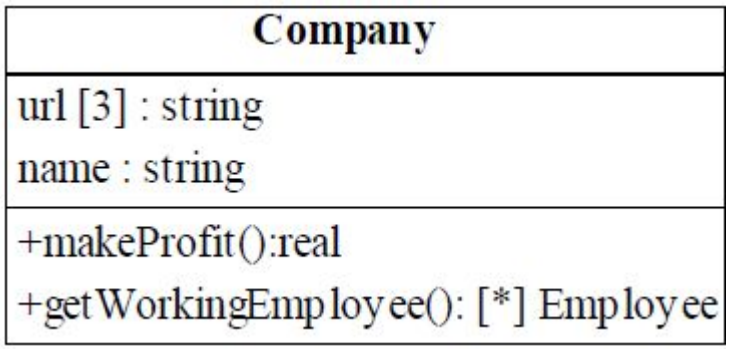
2.4 Opérations

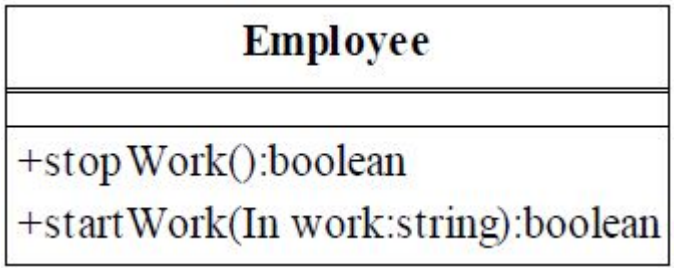
Une opération est un service qu'une instance de la classe peut exécuter

La syntaxe d'une opération est : ***visibility name(parameter):return***

La syntaxe des paramètres est : ***kind name : type***

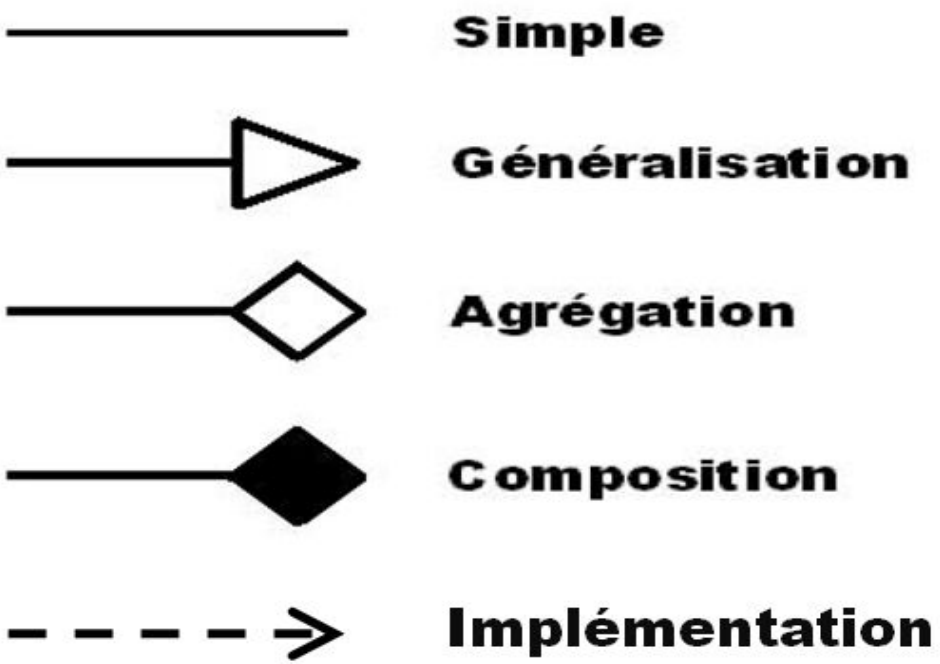
Le kind peut être : **in, out, inout**





2. Diagramme de classes

2.5 Types d'Association



2. Diagramme de classes

2.6 Associations

Les associations binaires connectent deux éléments entre eux

Une association binaire est composée de deux associations.

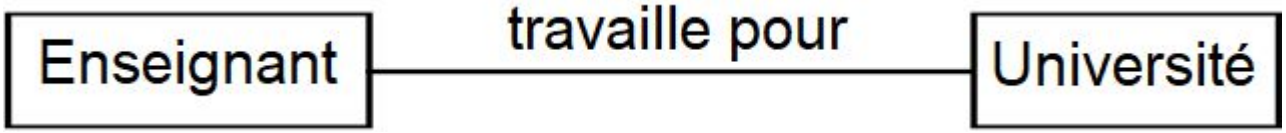
Une association end est paramétrée par:

- Un nom (le rôle joué par l'entité connectée)
- Un genre d'agrégation (composite, agrégation, none)

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)

2. Diagramme de classes

2.7 Nommage des associations

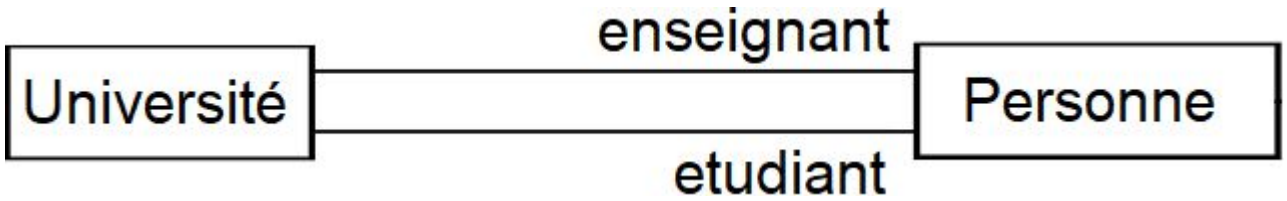


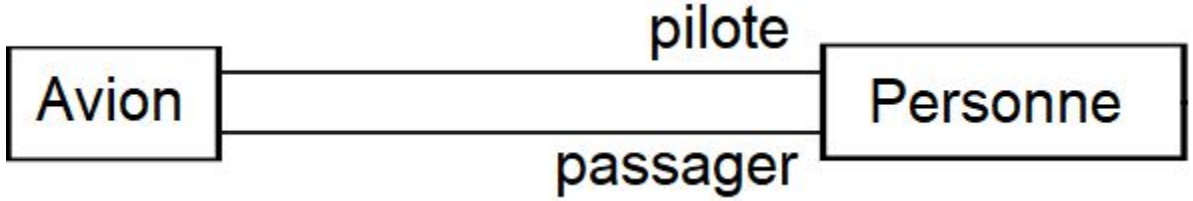
· On peut toujours ajouter un sens de lecture :

2. Diagramme de classes

2.8 Noms d'extrémité d'association - Rôle

- Chaque extrémité d'association peut être nommée.

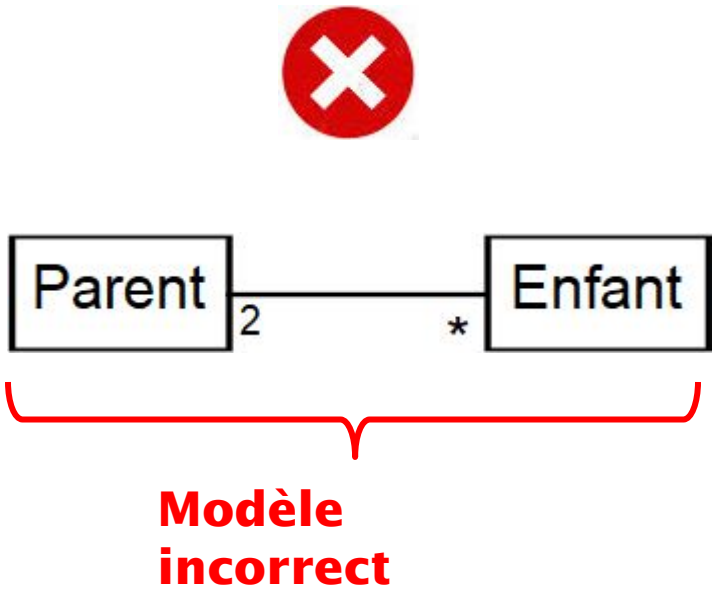




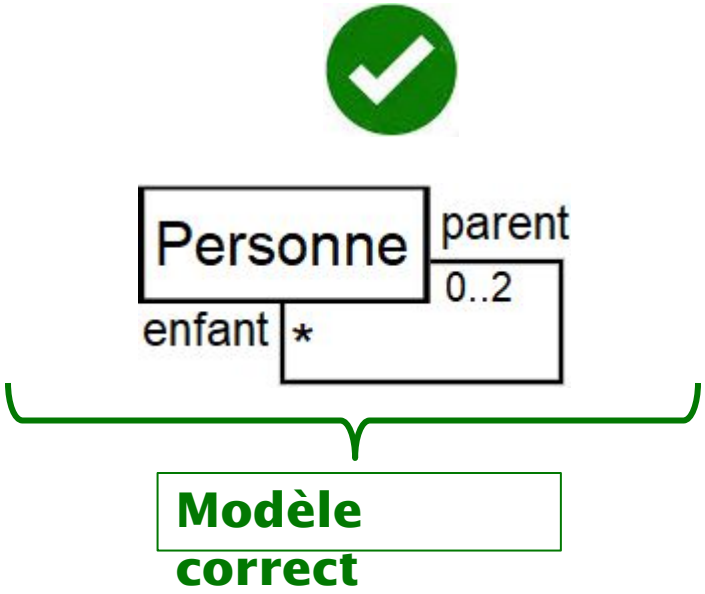
2. Diagramme de classes

2.9 Noms d'extrémité d'association - Rôle

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



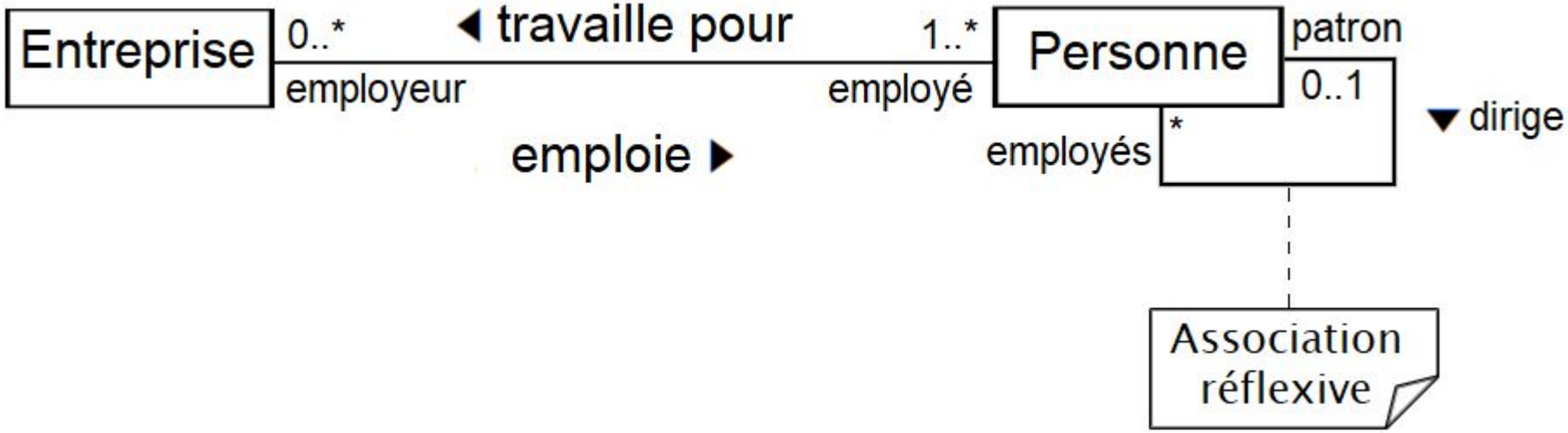
Modèle incorrect



Modèle correct

2. Diagramme de classes

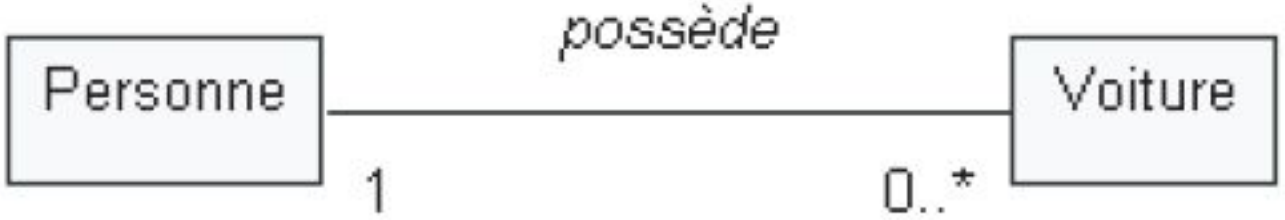
2.10 Exemple récapitulatif



2. Diagramme de classes

2.11 Multiplicité des associations

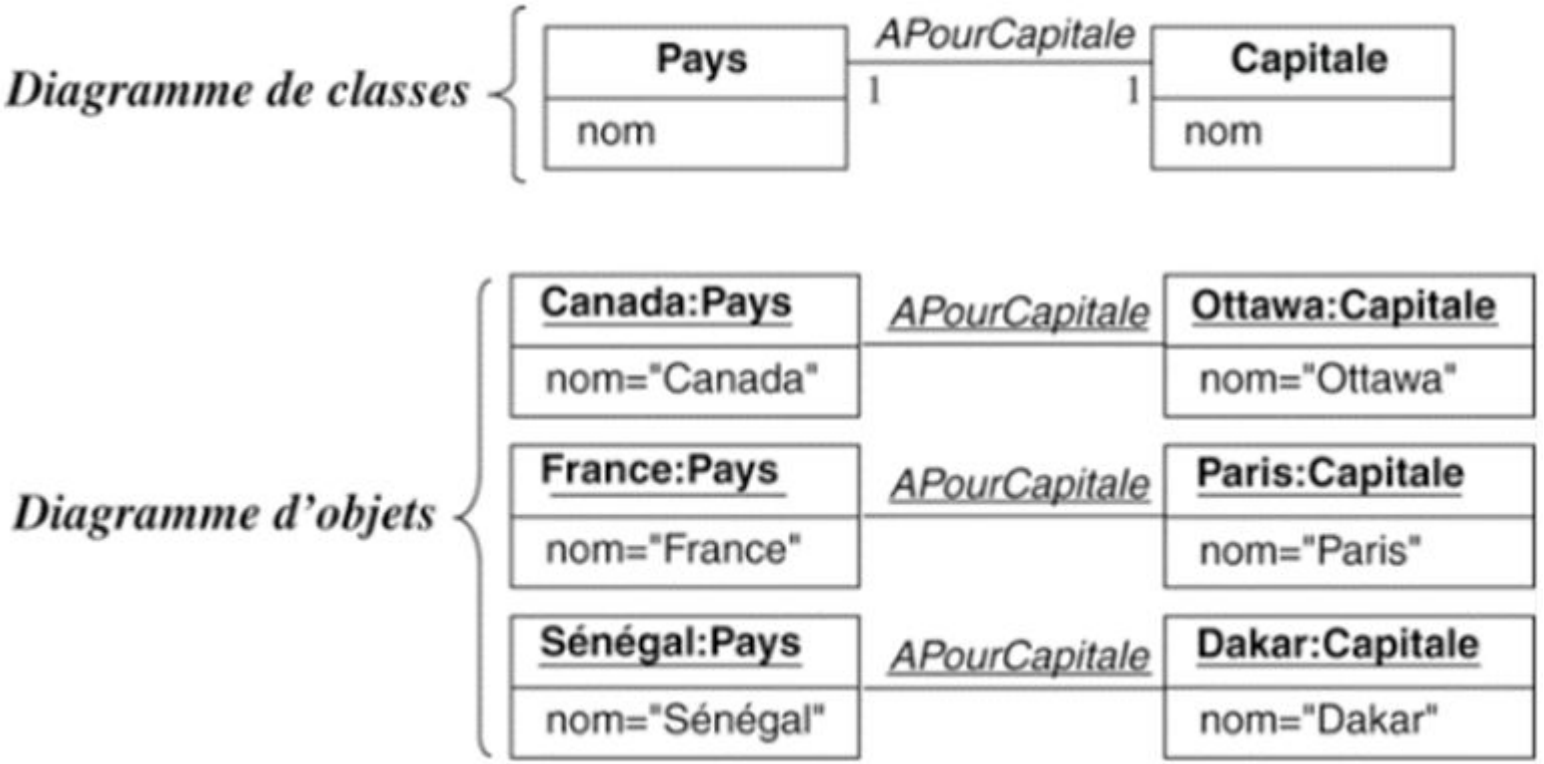
- Spécifie le nombre d'instances d'une classe pouvant être liées à une seule instance d'une classe associée.
- Elle contraint le nombre d'objets liés.



2. Diagramme de classes

2.11 Multiplicité des associations

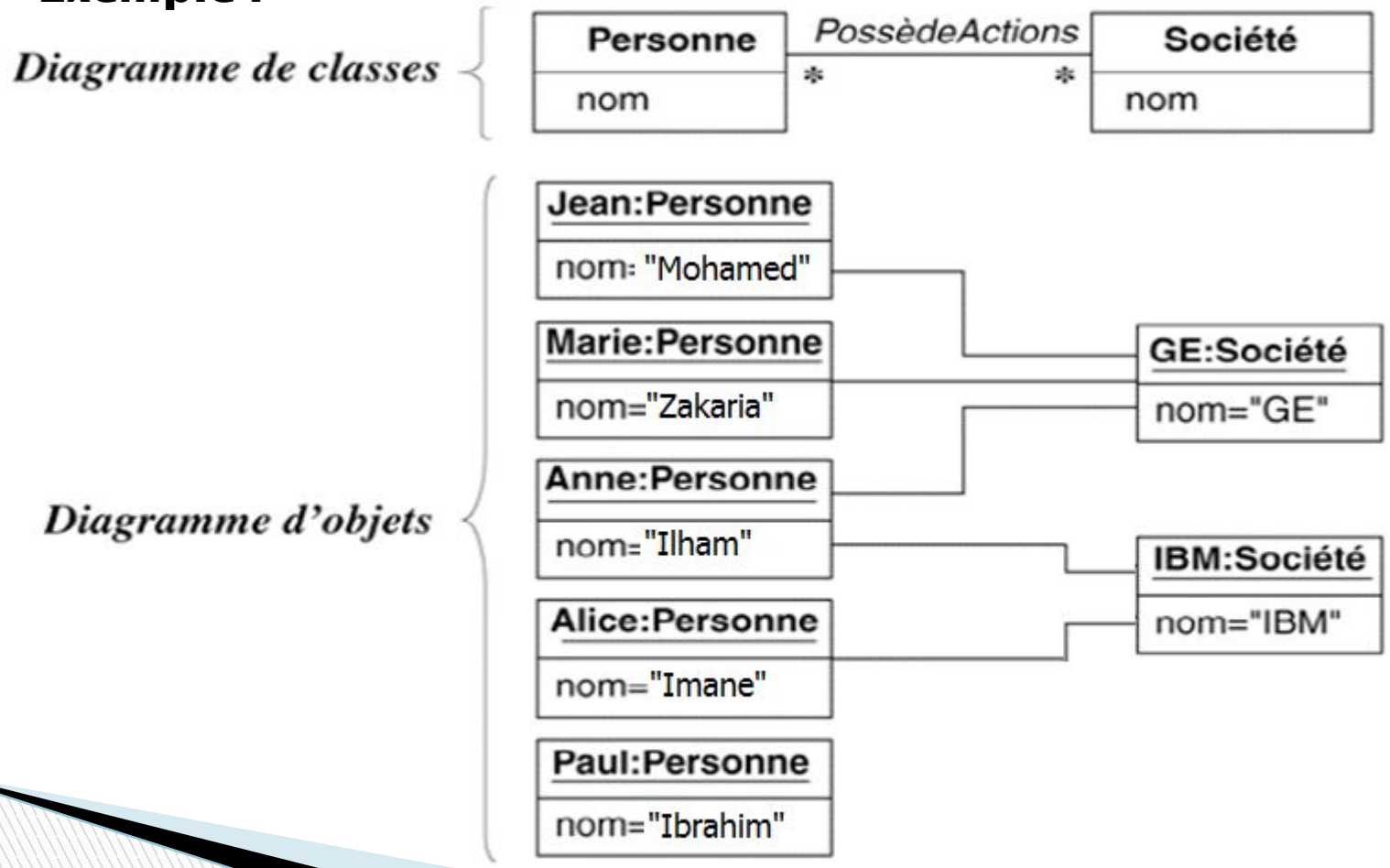
- **Exemple :**



2. Diagramme de classes

2.11 Multiplicité des associations

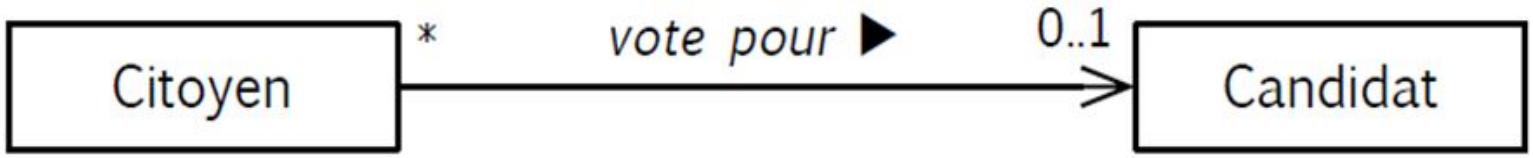
- **Exemple :**



2. Diagramme de classes

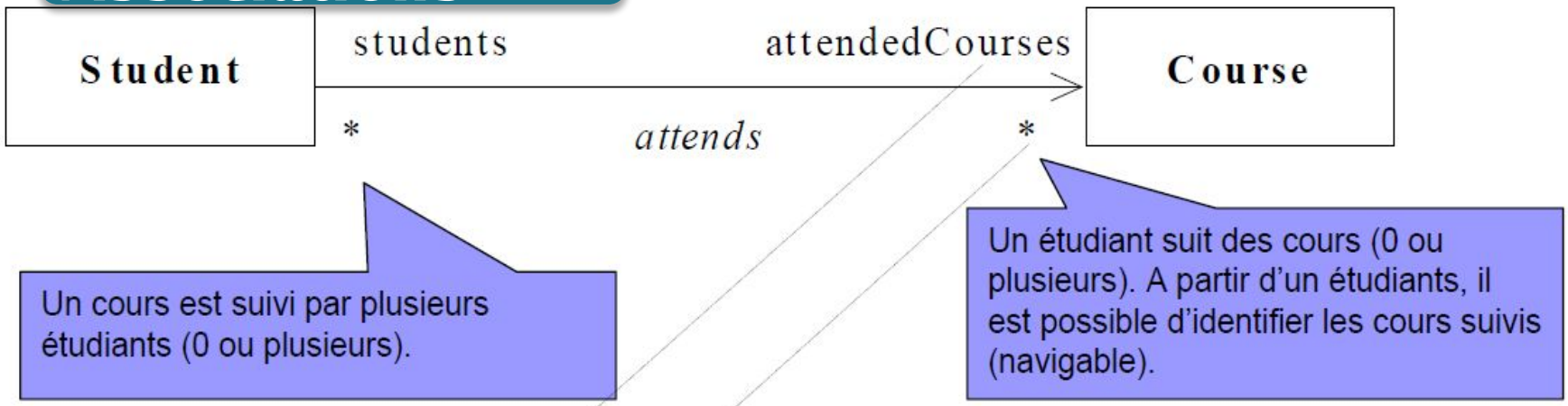
2.12 Navigabilité d'une association

- Les associations sont par défaut navigables dans les deux sens
- La navigation peut être restreinte à une seule direction : les instances d'une classe ne "connaissent" pas les instances d'une autre classe.
- On restreint la navigabilité d'une association à un sens à l'aide d'une flèche.



2. Diagramme de classes

2.13 Associations




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

   public Student()
   {
   }
}
```

```
public class Course
{

   public Course()
   {

   }
}
```

GOOD luck

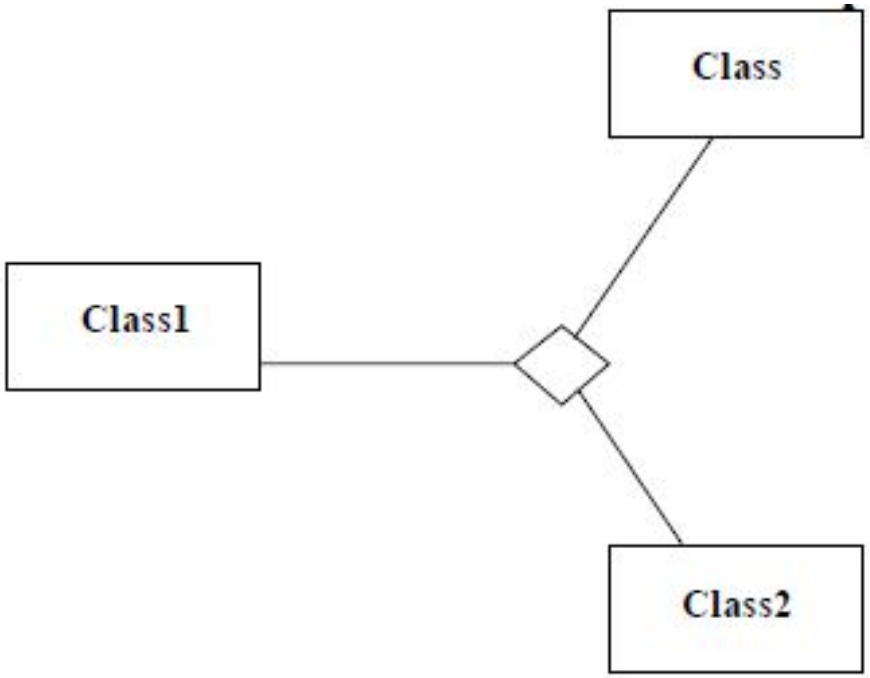
keep going

dédicace l aymen lgang
sans oublier sami la street
dedicace l akma lboss

2. Diagramme de classes

2.14 Associations N-aires

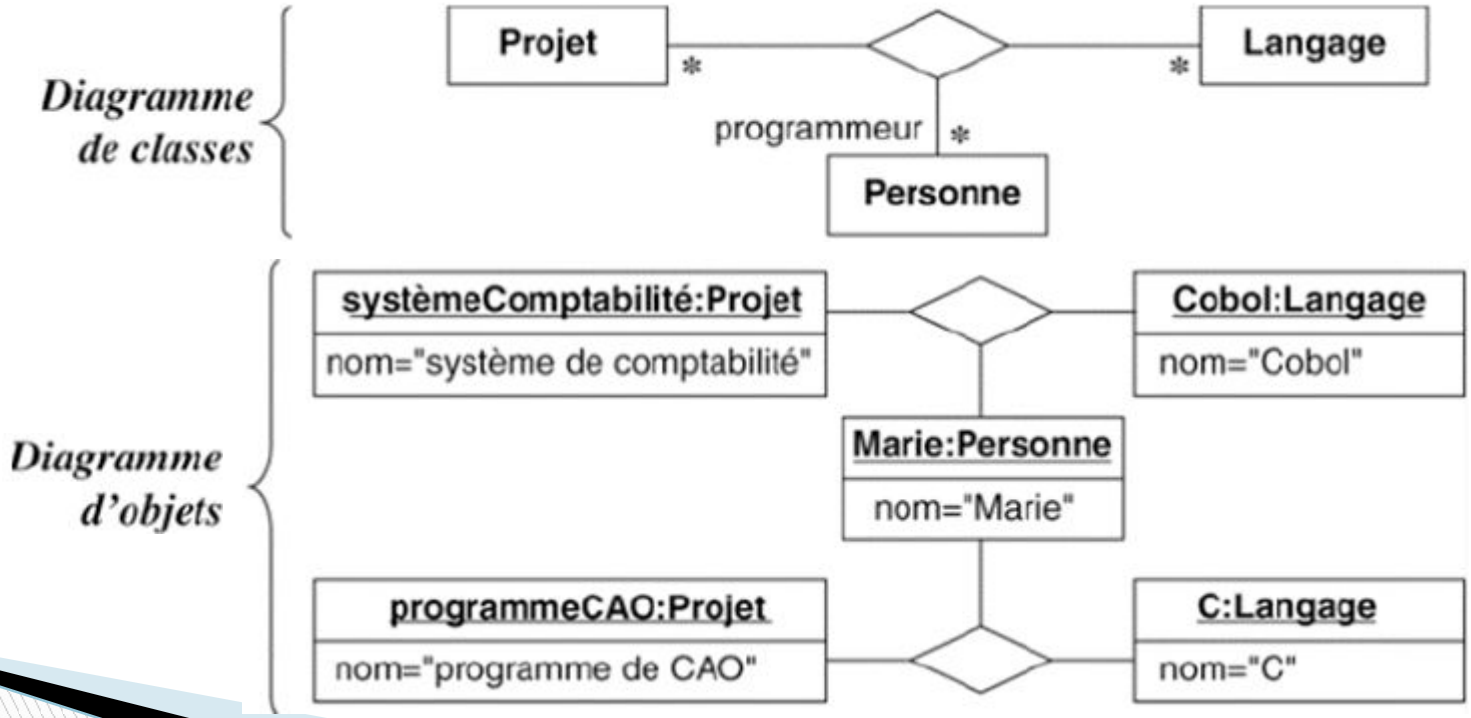
- Les associations N-aires connectent plusieurs éléments entre eux.
- Les associations N-aires sont très peu utilisées.



2. Diagramme de classes

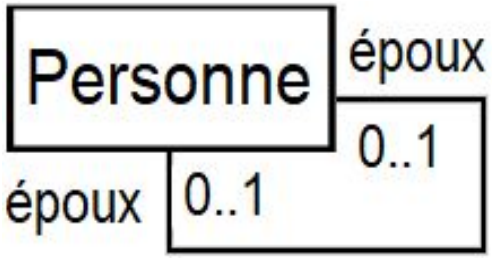
2.14 Associations N-aires

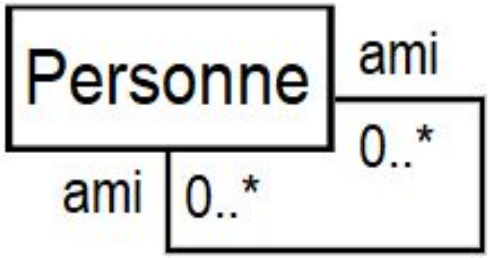
- En général, les associations sont binaires
- A-aires : au moins trois instances impliquées
- A n'utiliser que lorsqu'aucune autre solution n'est possible



2. Diagramme de classes

2.15 Associations réflexives





2. Diagramme de classes

2.16 Agrégation

Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance d'un élément dans un ensemble

Les agrégations ne sont pas nommées : implicitement elles signifient « contient », « est composé de ».

L'agrégation est représentée par l'ajout d'un losange vide du côté de l'agrégat (l'ensemble).

2. Diagramme de classes

2.16 Agrégation - Exemples

2. Diagramme de classes

2.17 Composition

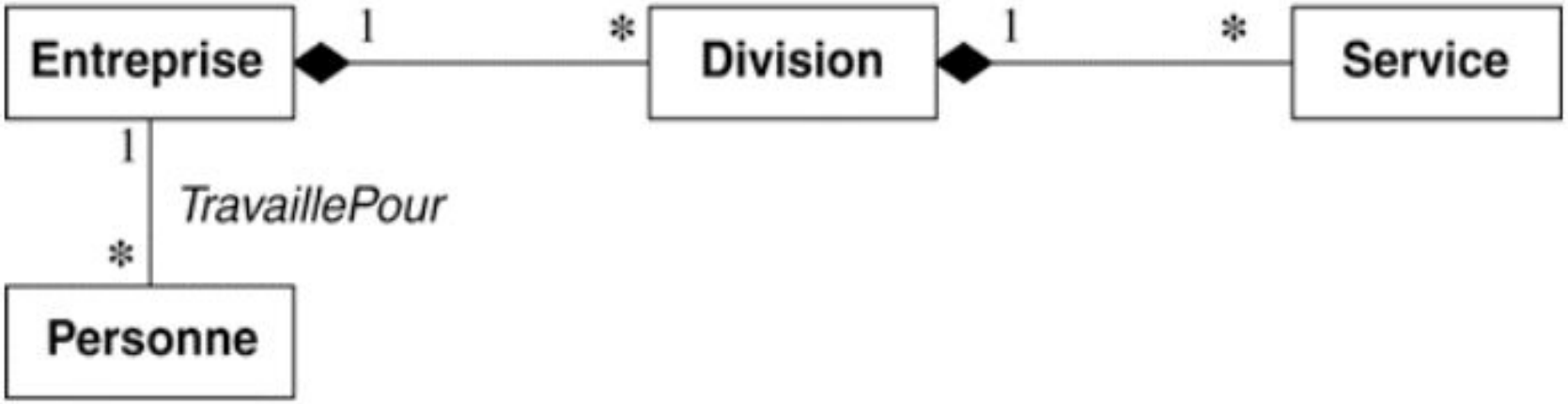
Une composition est une agrégation plus forte impliquant que :

- 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.

2. Diagramme de classes

2.17 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.



2. Diagramme de classes

2.18 Agrégation vs Composition

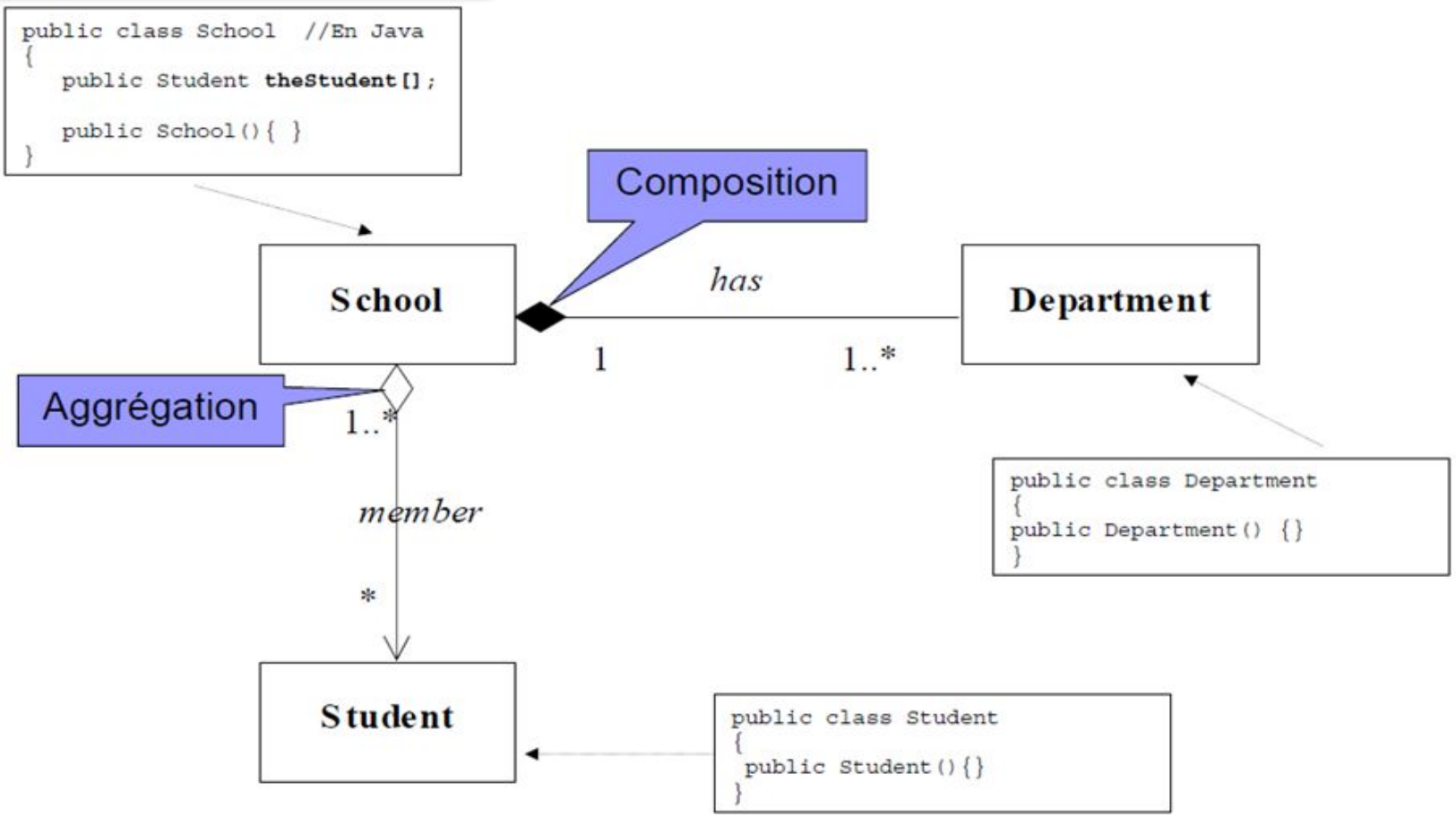
Quand mettre une composition plutôt qu'une agrégation ?

- 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.

2. Diagramme de classes

2.19 Associations

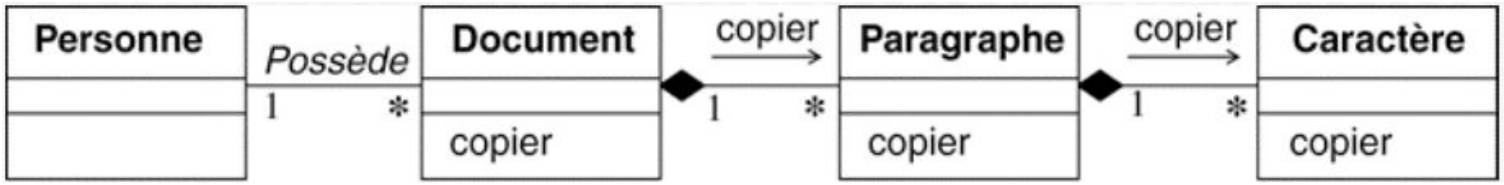


2. Diagramme de classes

2.20 Propagation (ou déclenchement)

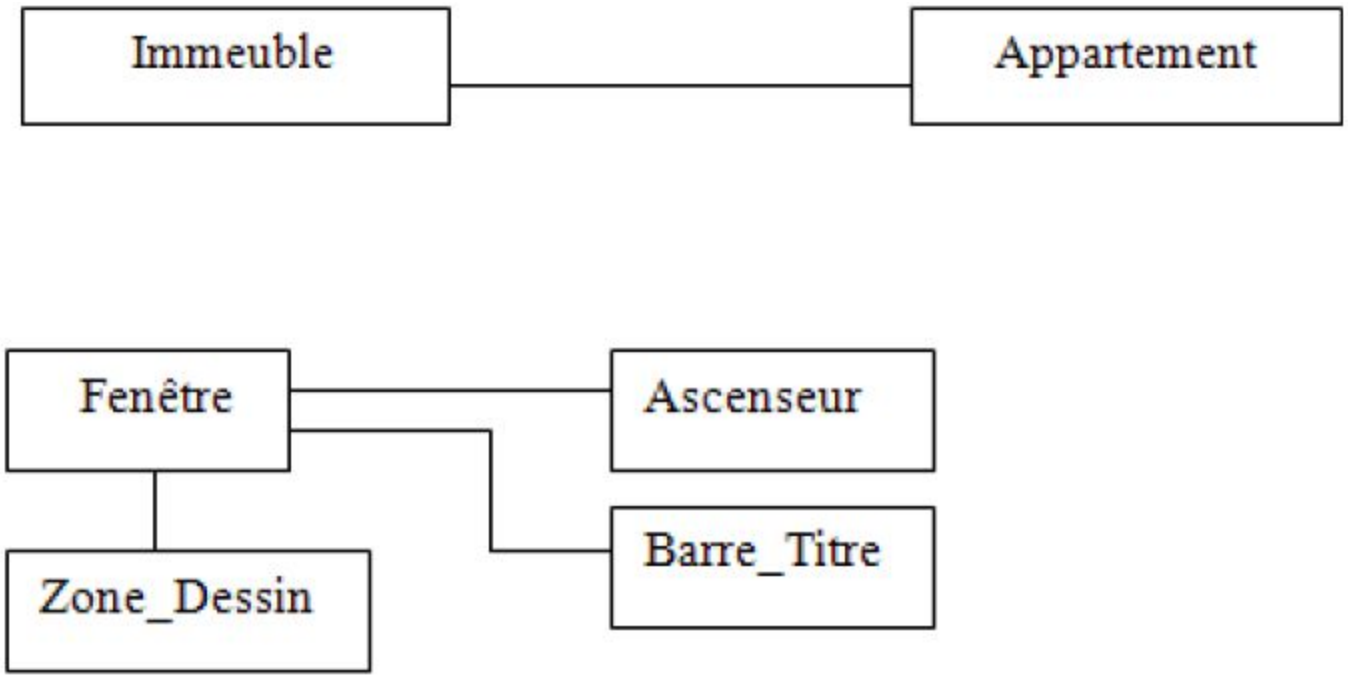
- Application automatique d'une opération à un réseau d'objets à partir d'un objet initial quelconque.

- 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*().



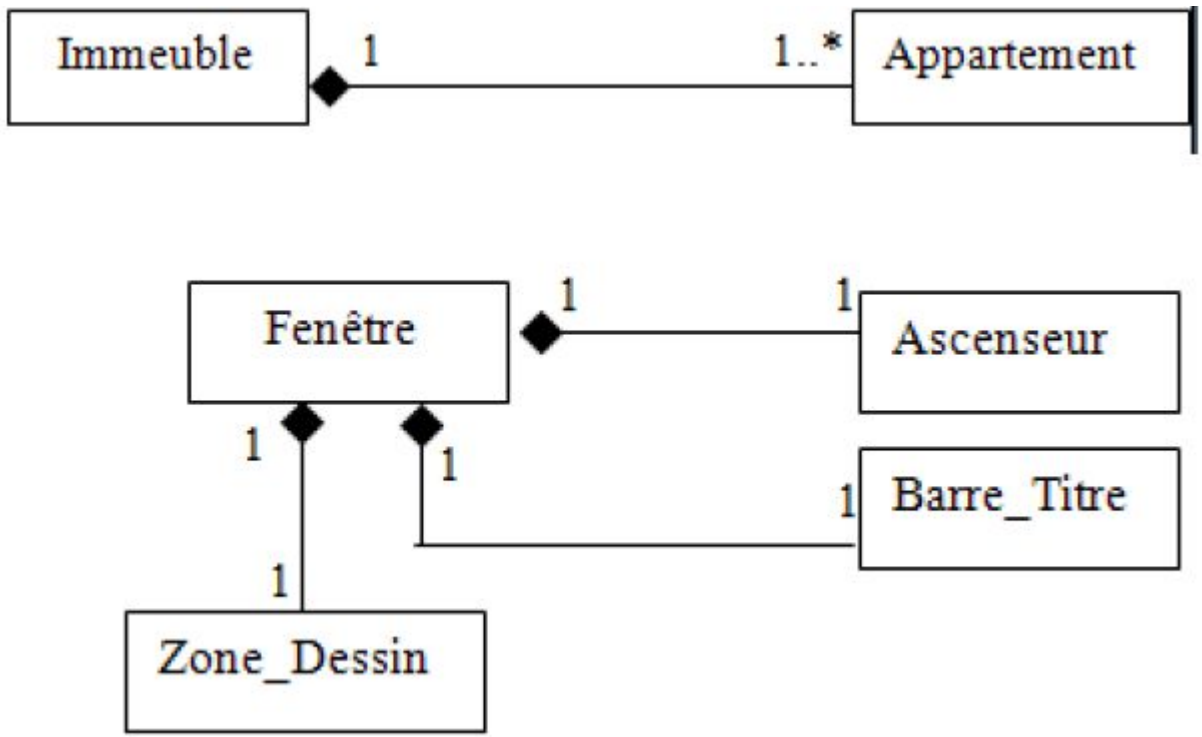
2. Diagramme de classes

2.21 Agrégation vs Composition



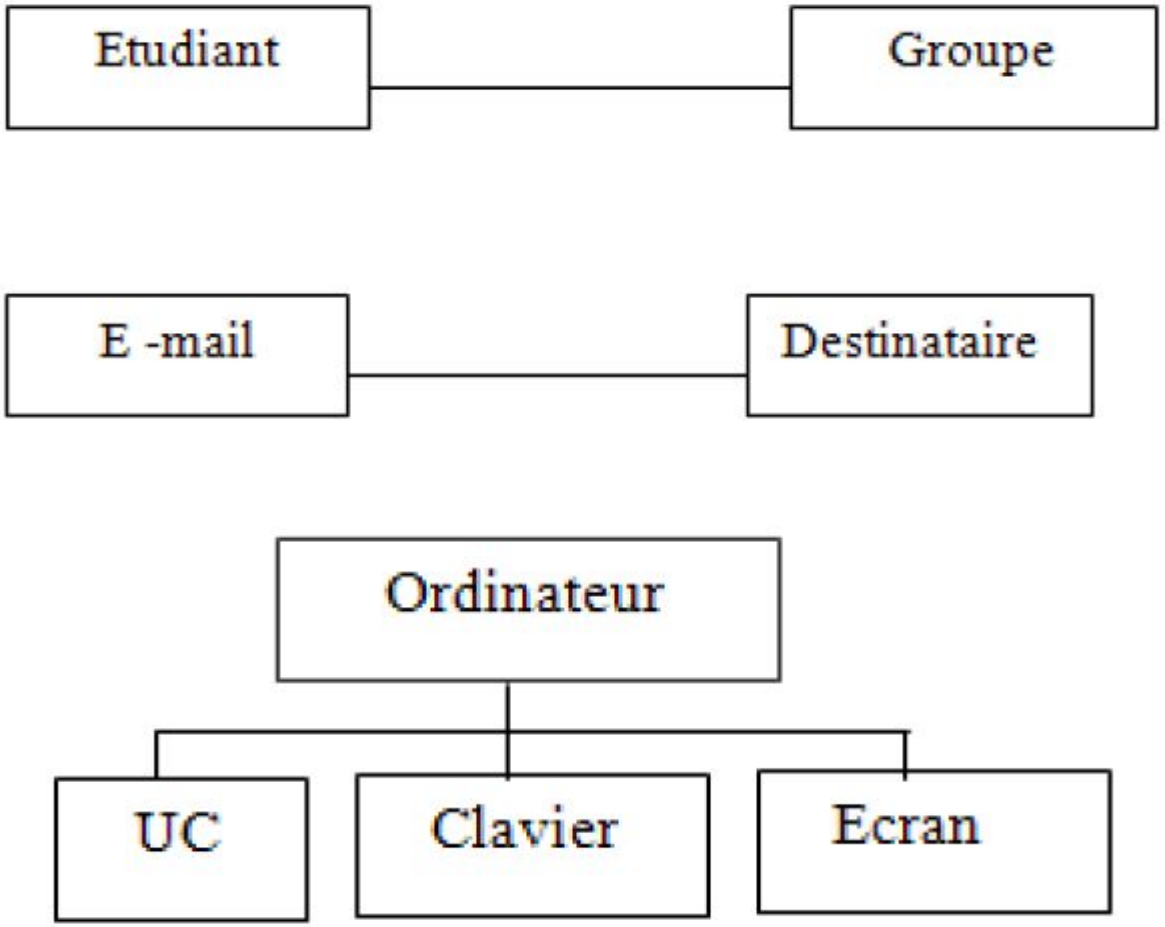
2. Diagramme de classes

2.22 Agrégation vs Composition



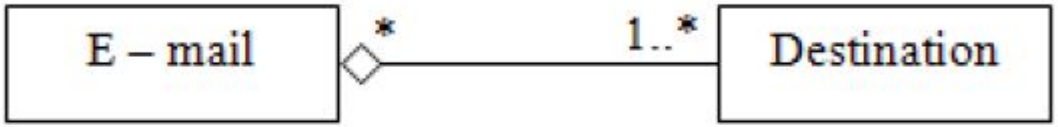
2. Diagramme de classes

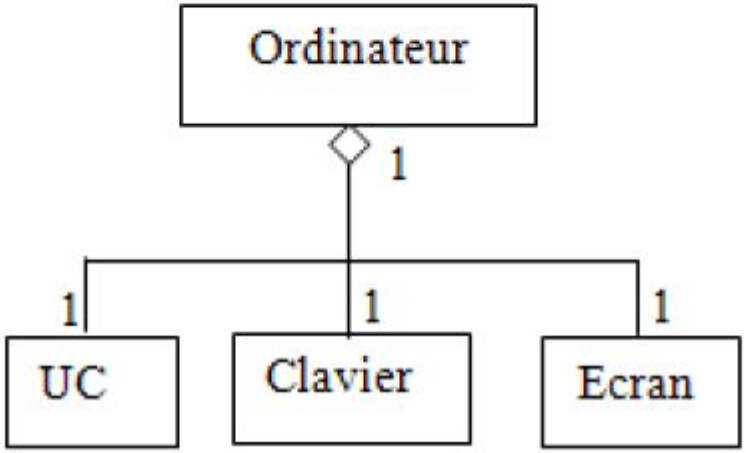
2.22 Agrégation vs Composition



2. Diagramme de classes

2.22 Agrégation vs Composition

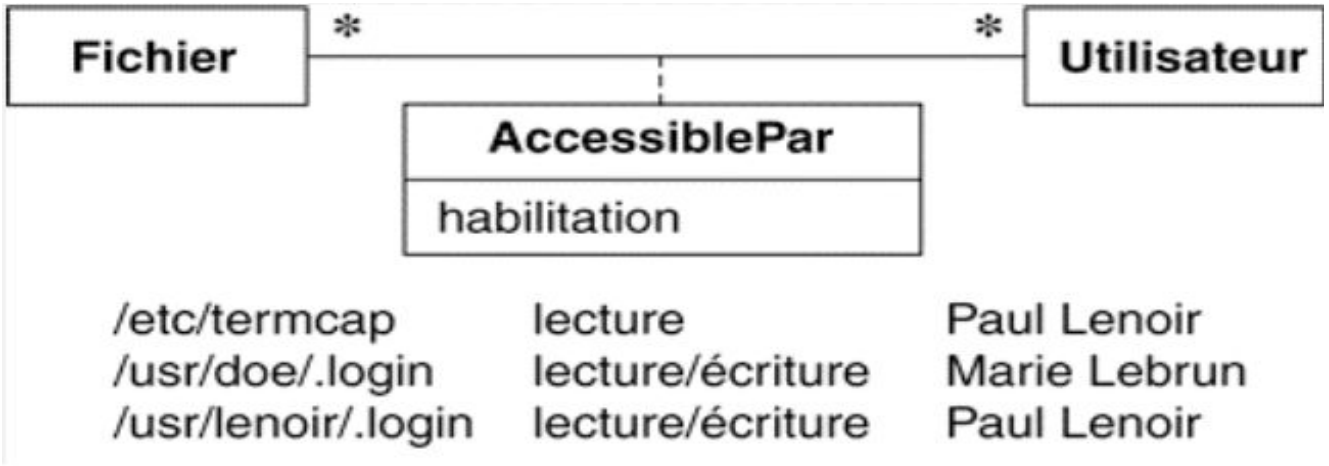




2. Diagramme de classes

2.23 Classe d'Association

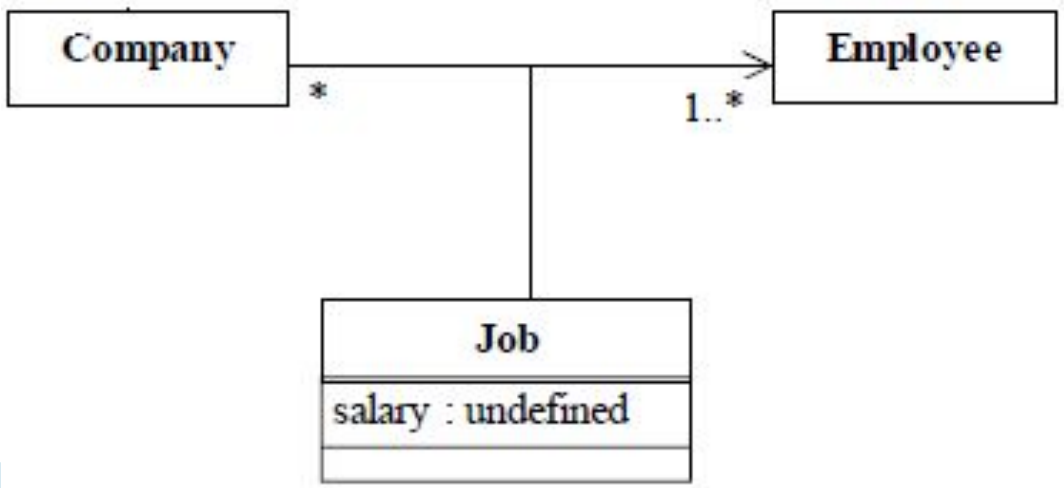
- association qui est aussi une classe.



2. Diagramme de classes

2.23 Classe d'Association

- 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.

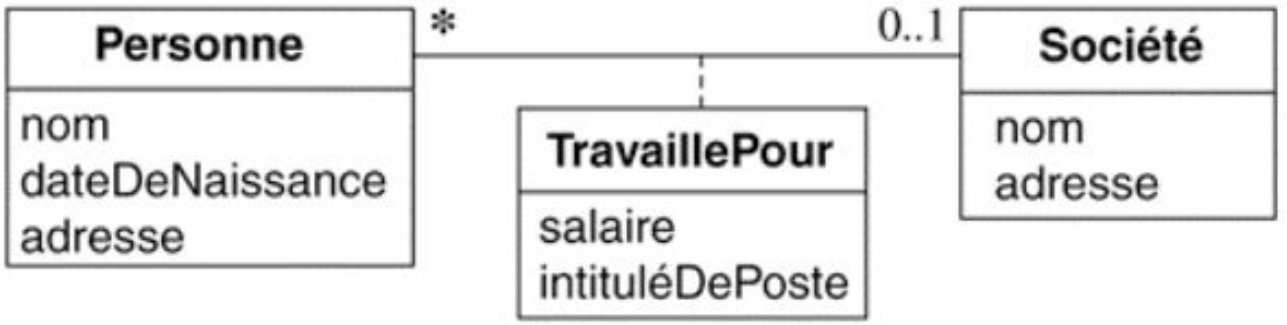


2. Diagramme de classes

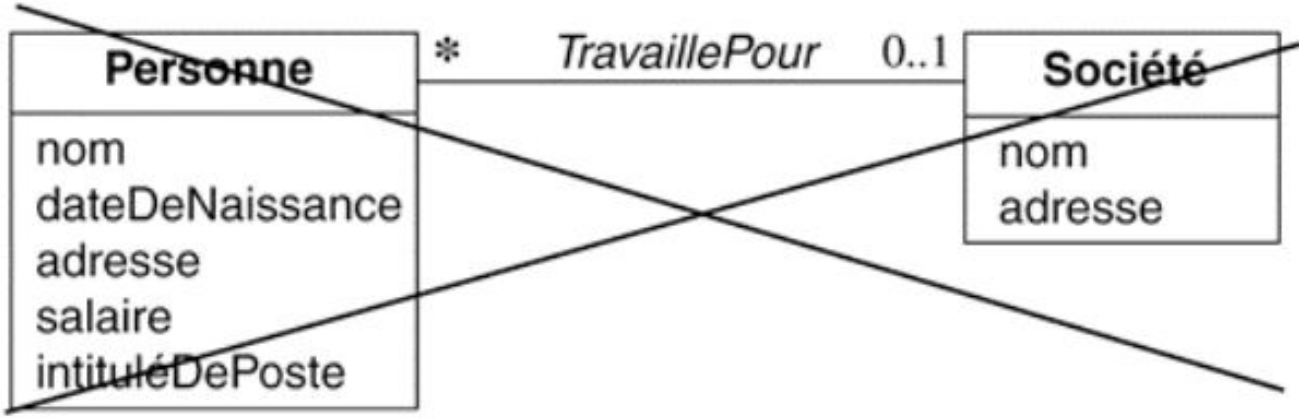
2.23 Classe d'Association

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

Forme conseillée



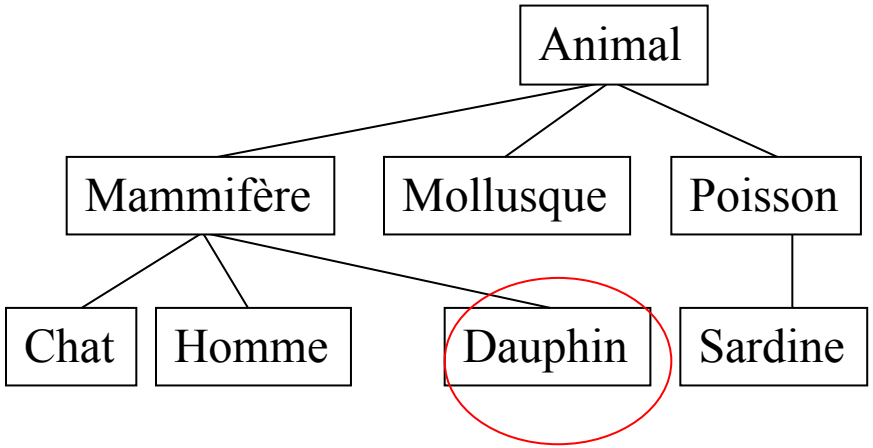
Forme déconseillée



2. Diagramme de classes

2.24 Héritage

- **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

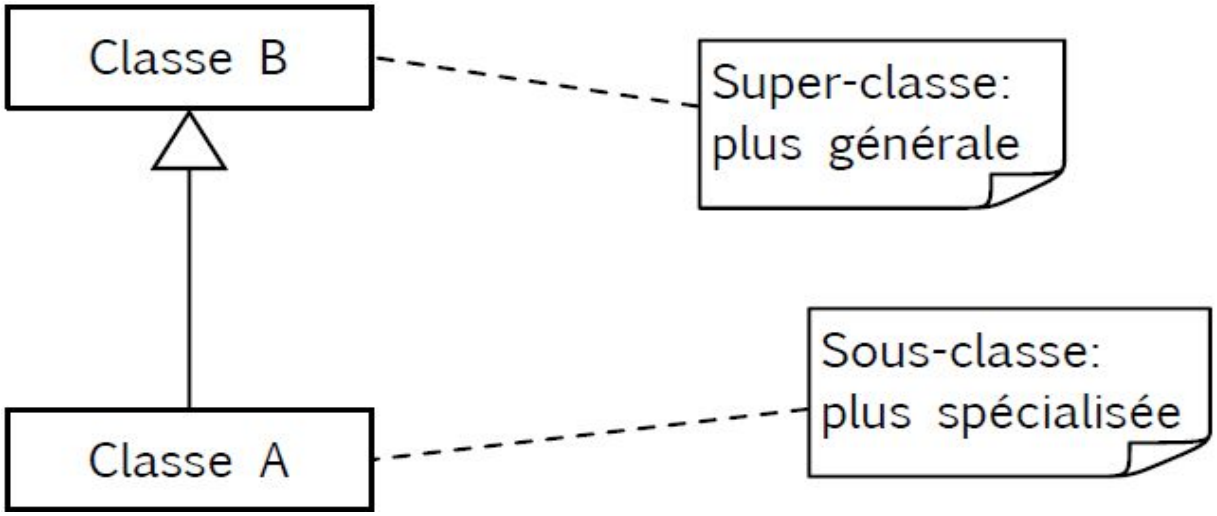
Héritage multiple possible en UML

2. Diagramme de classes

2.24 Héritage

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

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

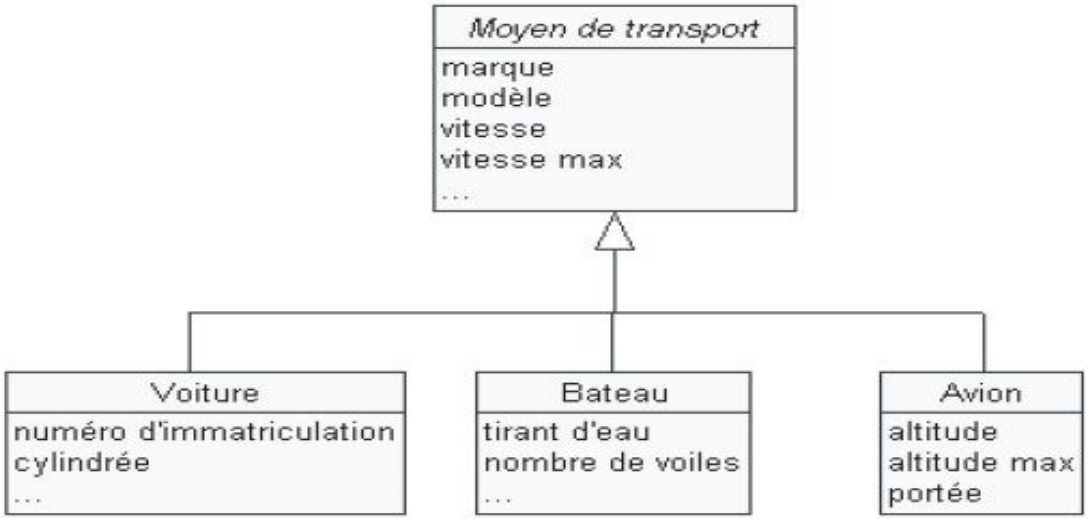


2. Diagramme de classes

2.24 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).

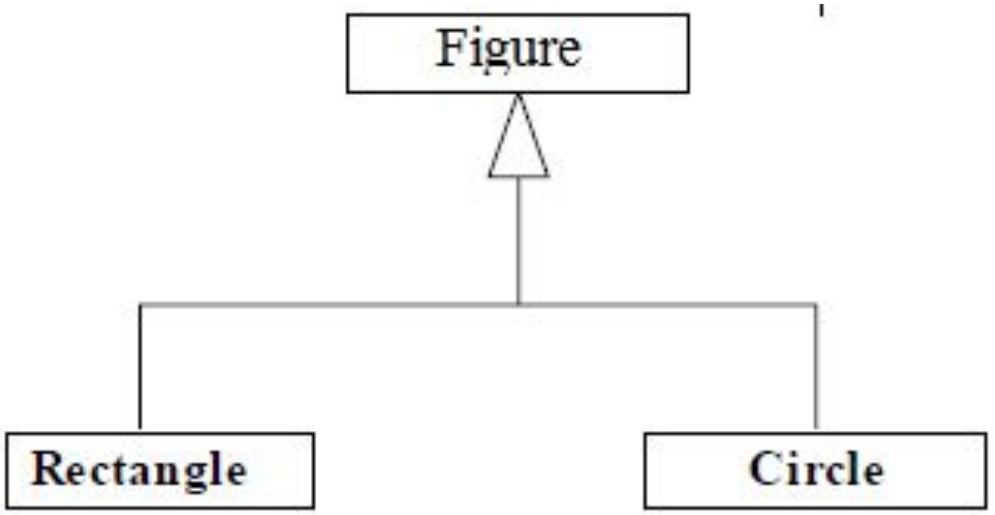


2. Diagramme de classes

2.24 Héritage

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

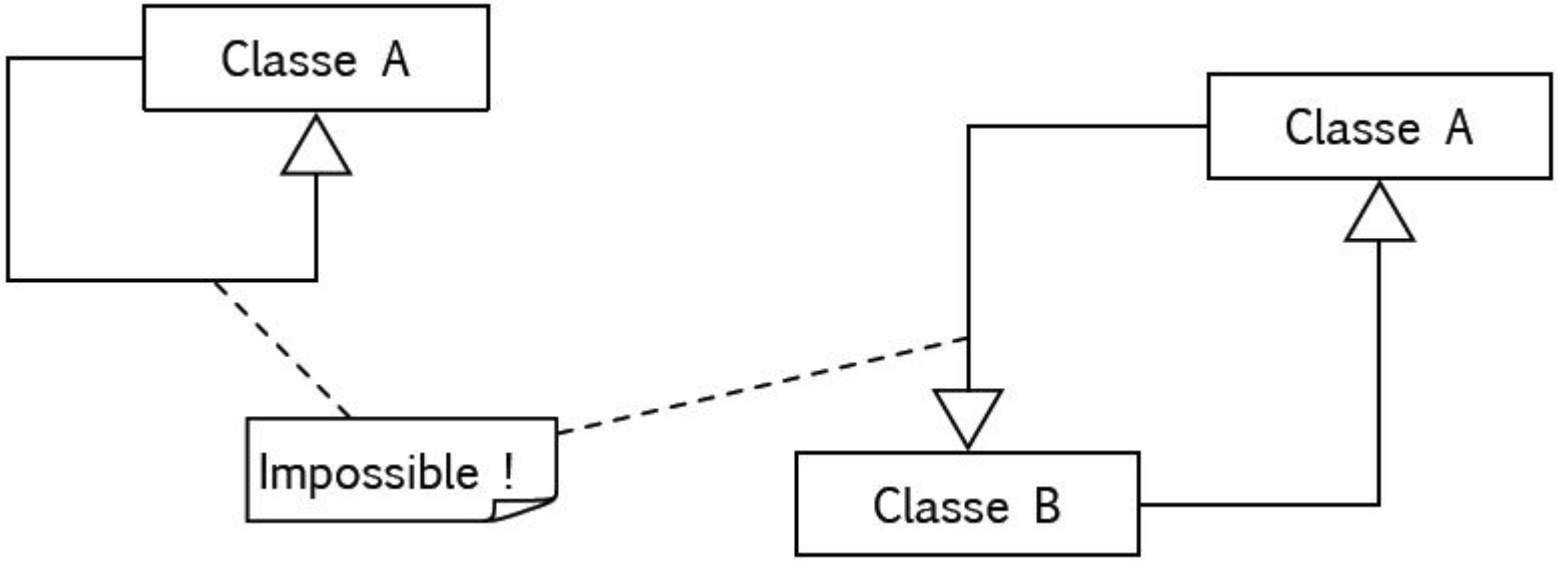
- L'héritage multiple est possible en UML



2. Diagramme de classes

2.24 Héritage

- Relation non-réflexive, non-symétrique !

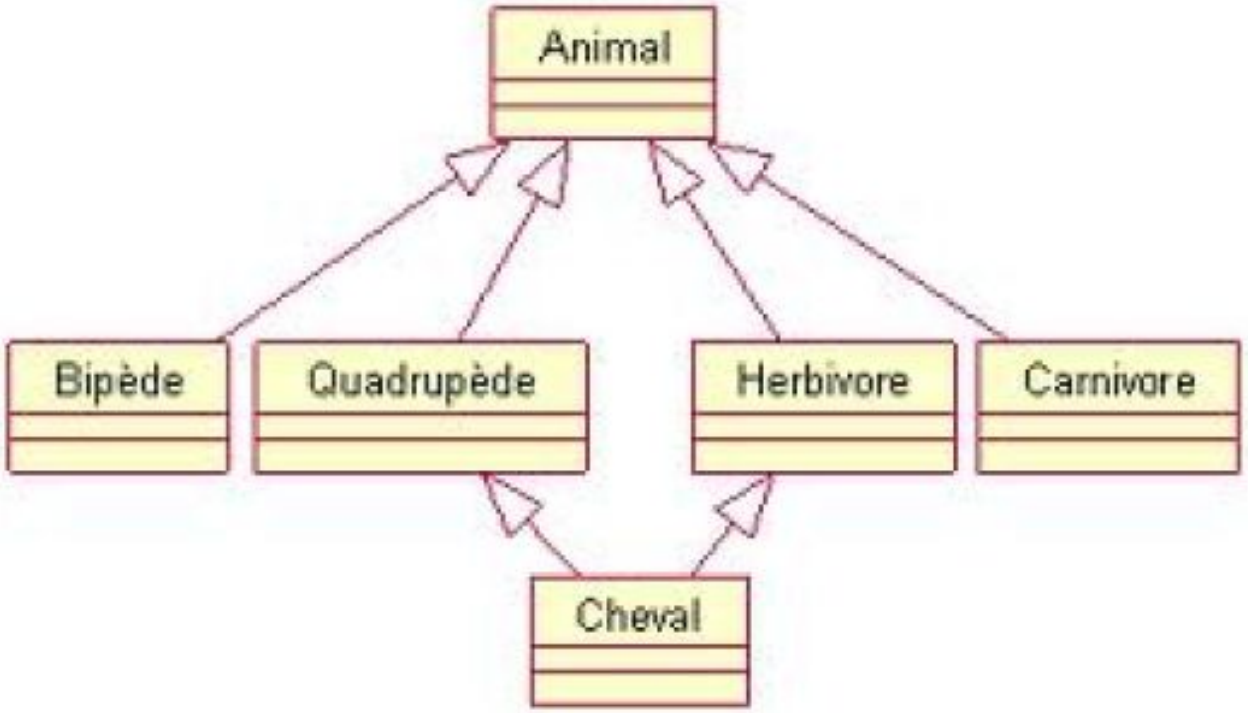


2. Diagramme de classes

2.25 Héritage multiple

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

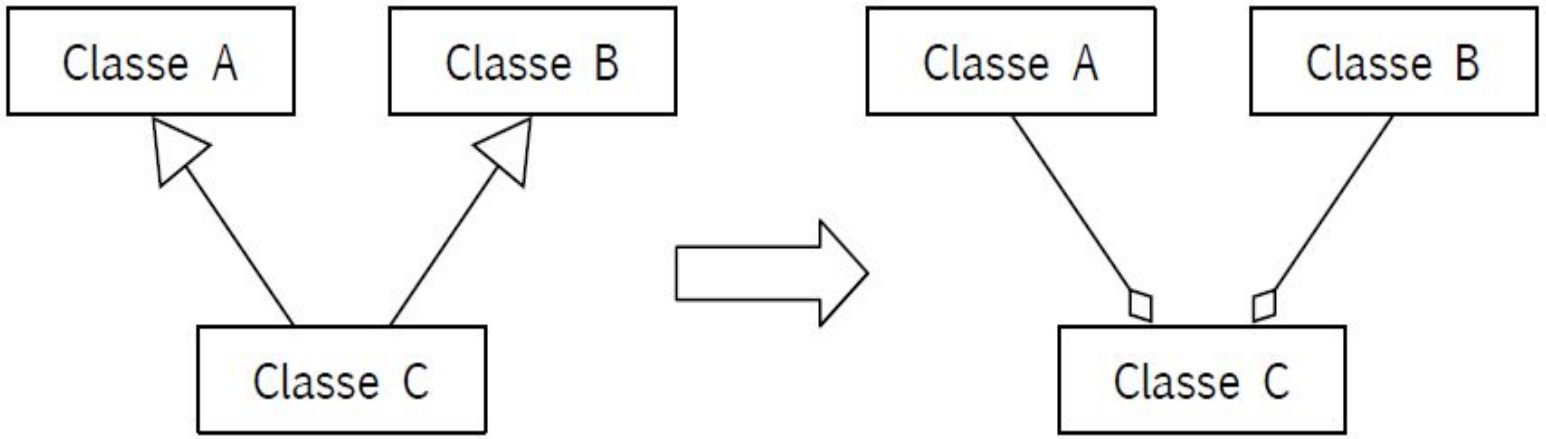
- Exemple :



2. Diagramme de classes

2.25 Héritage multiple

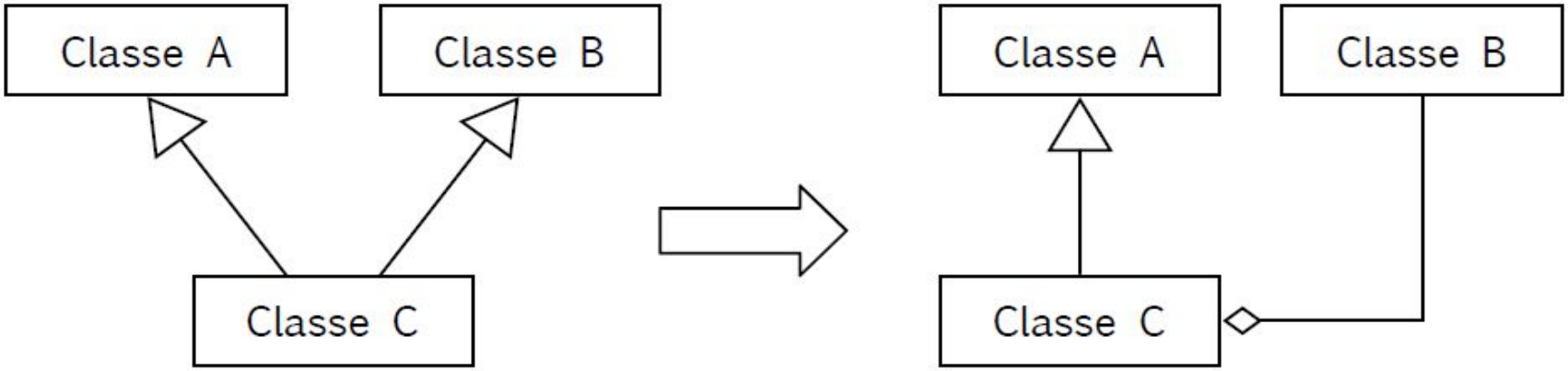
- Comment éviter l'héritage multiple ?
- Première solution : déléguer



2. Diagramme de classes

2.25 Héritage multiple

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

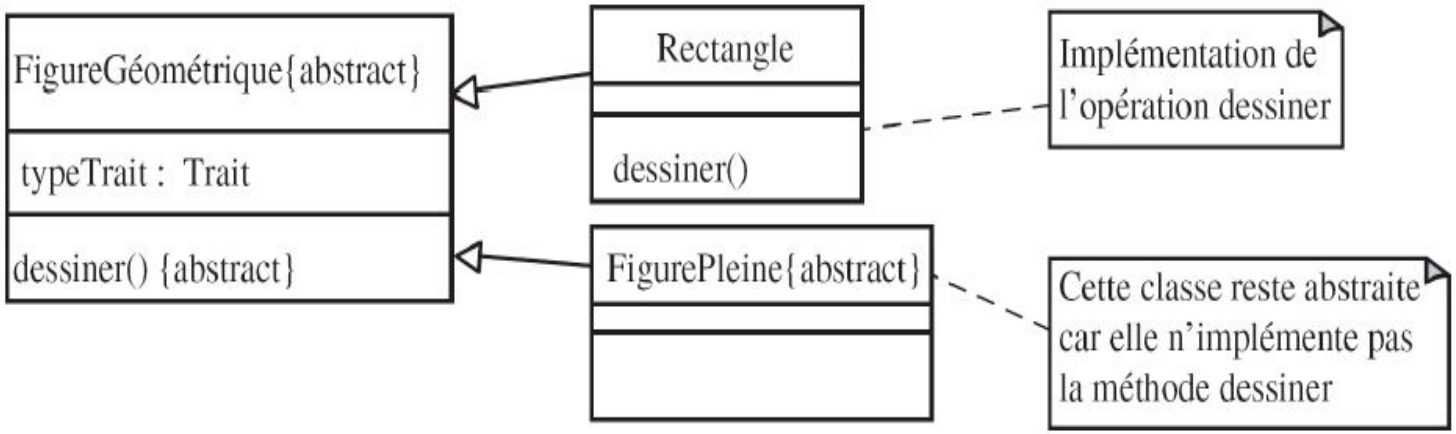


2. Diagramme de classes

2.26 Classes abstraites

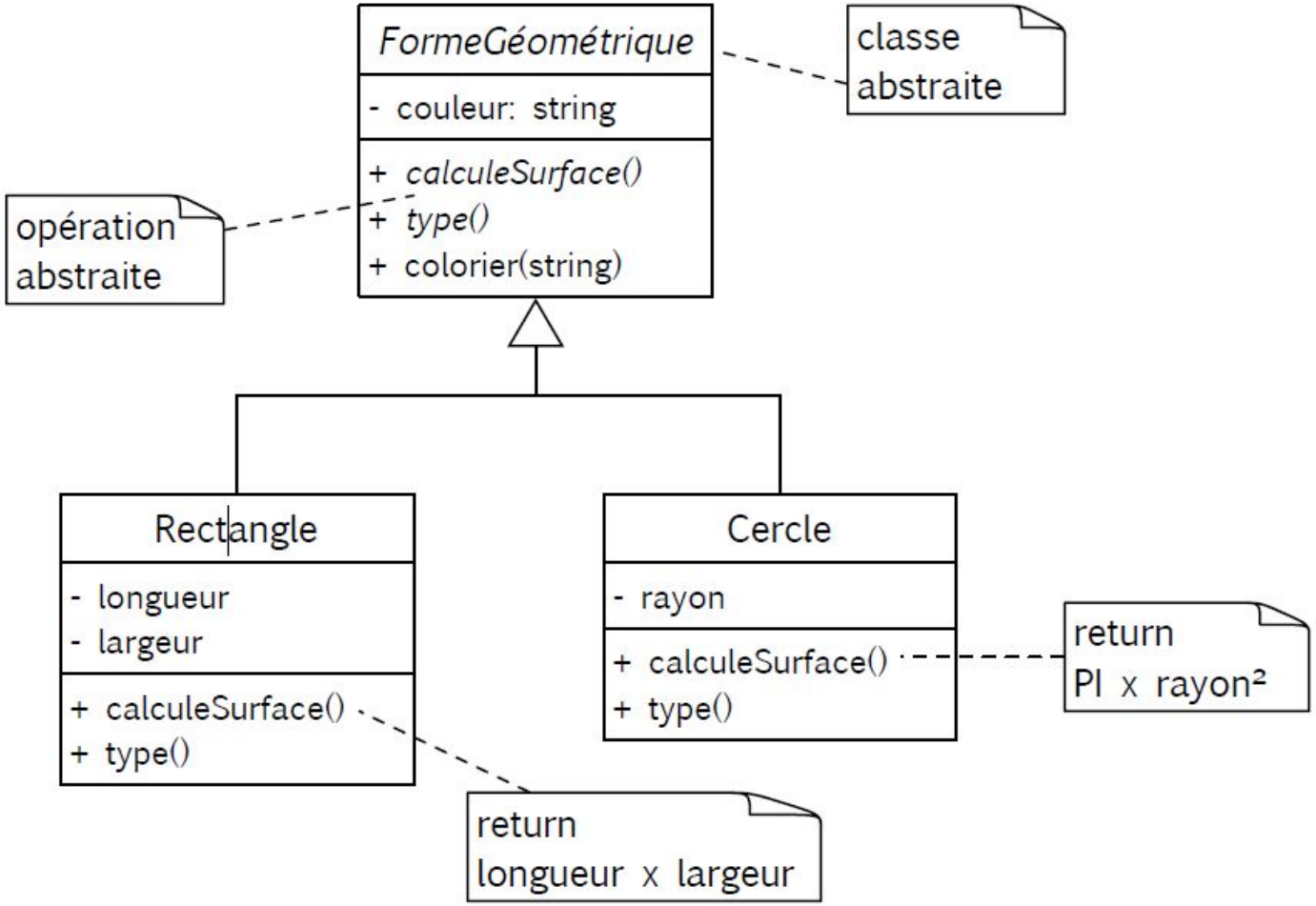
- Une **méthode** est dite **abstraite** lorsqu'on connaît son entête (signature) mais pas la manière dont elle peut être réalisée. Il appartient aux classes enfant de définir les méthodes abstraites.

- 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.



2. Diagramme de classes

2.26 Classes abstraites - Exemple



2. Diagramme de classes

2.27 Interfaces

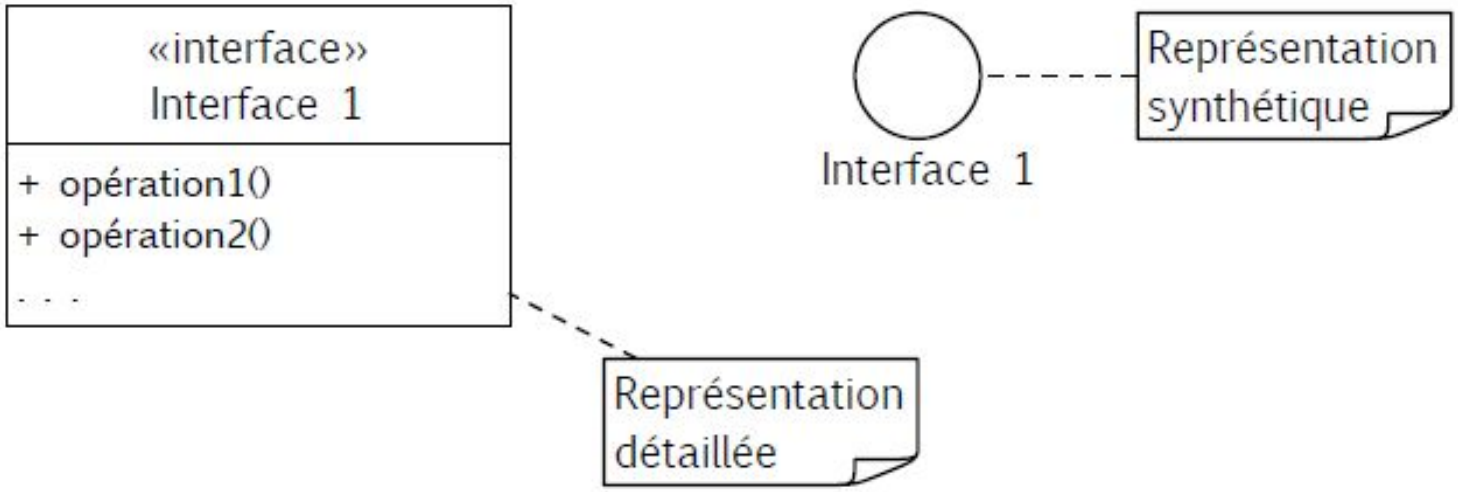
- **Interface** : définition abstraite d'une classe
  - Ne possède que des méthodes
  - Découple la définition d'une classe de son implantation
  - Lien avec la notion d'API
  - Une interface peut être implantée (mot clé implements) par plusieurs classes
  - Une classe peut posséder plusieurs interfaces

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

2. Diagramme de classes

2.27 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

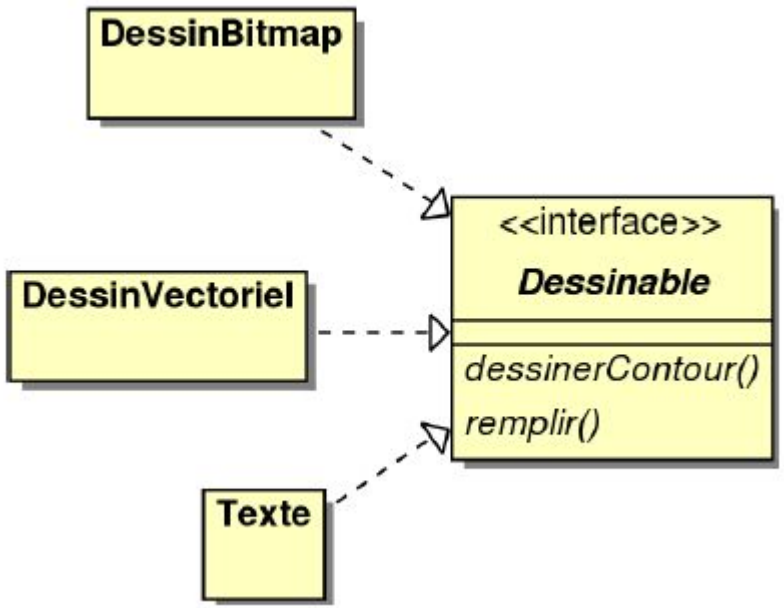


2. Diagramme de classes

2.27 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



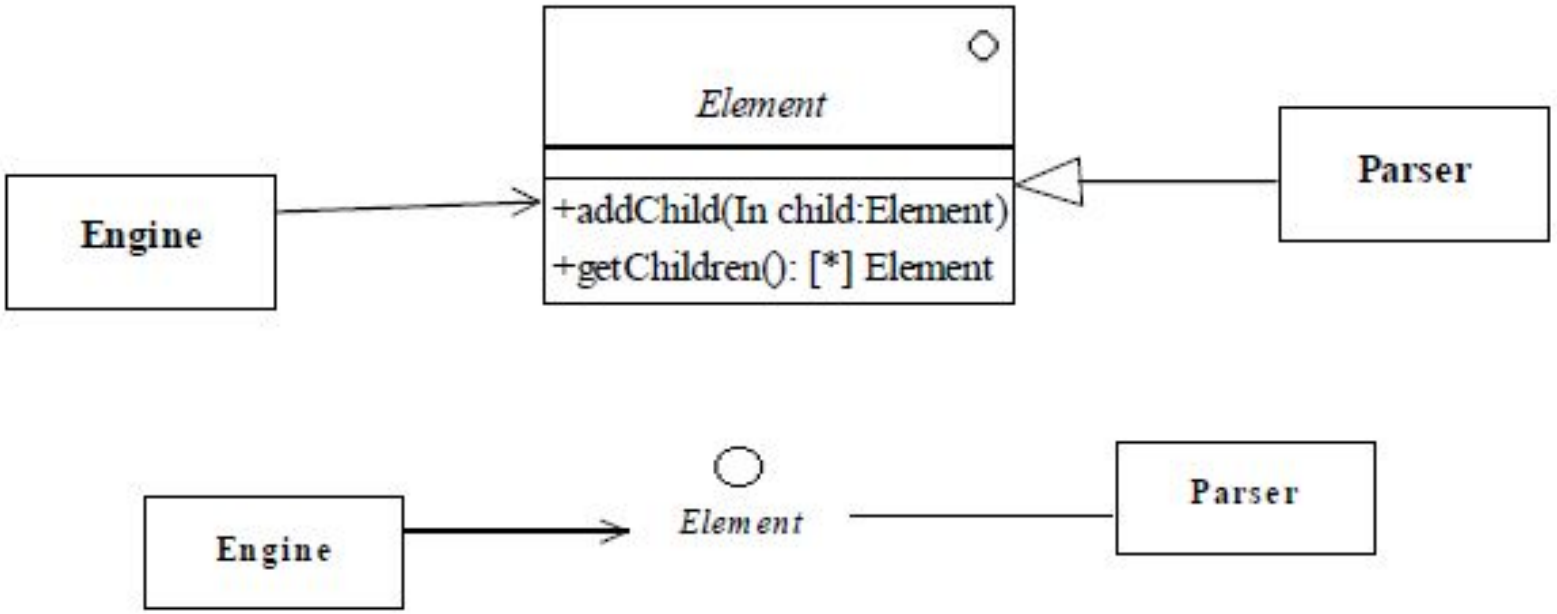
2. Diagramme de classes

2.27 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.

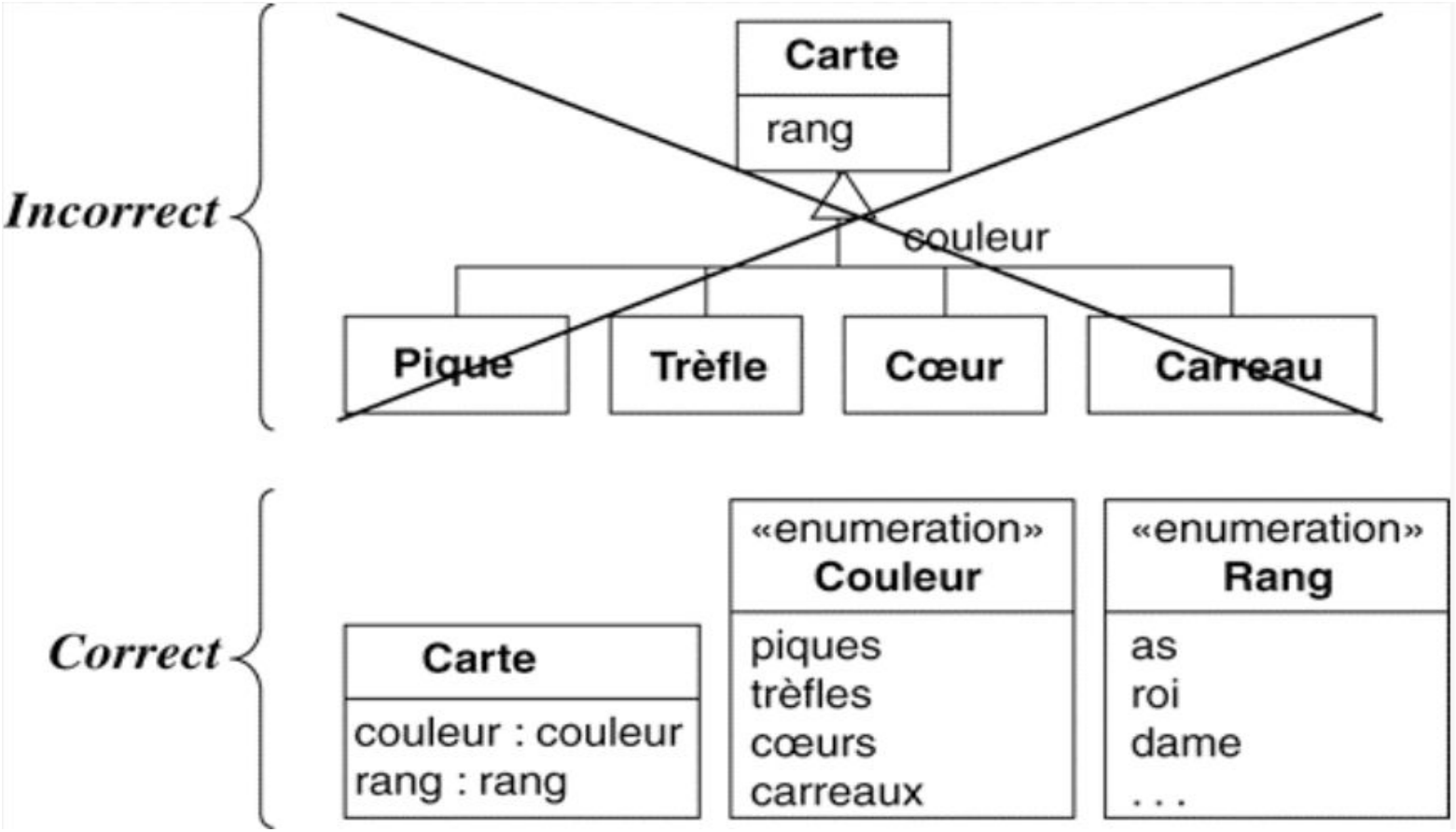
2. Diagramme de classes

2.27 Interfaces



2. Diagramme de classes

2.28 Enumération



2. Diagramme de classes

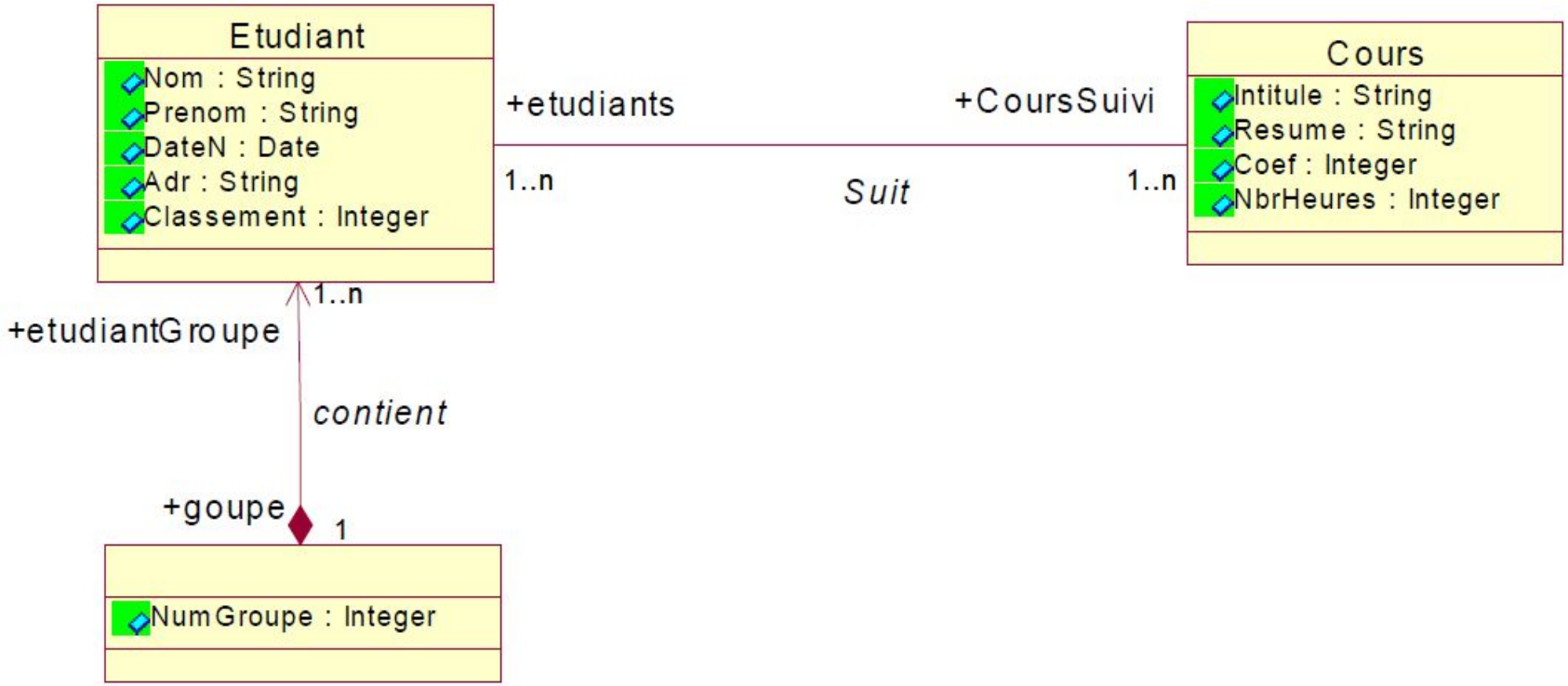
2.29 Etablir un diagramme de classes

- 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.

2. Diagramme de classes

2.30 Exercice

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



2. Diagramme de classes

2.31 Conclusion

- 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