Génie logiciel

Diagramme de classes

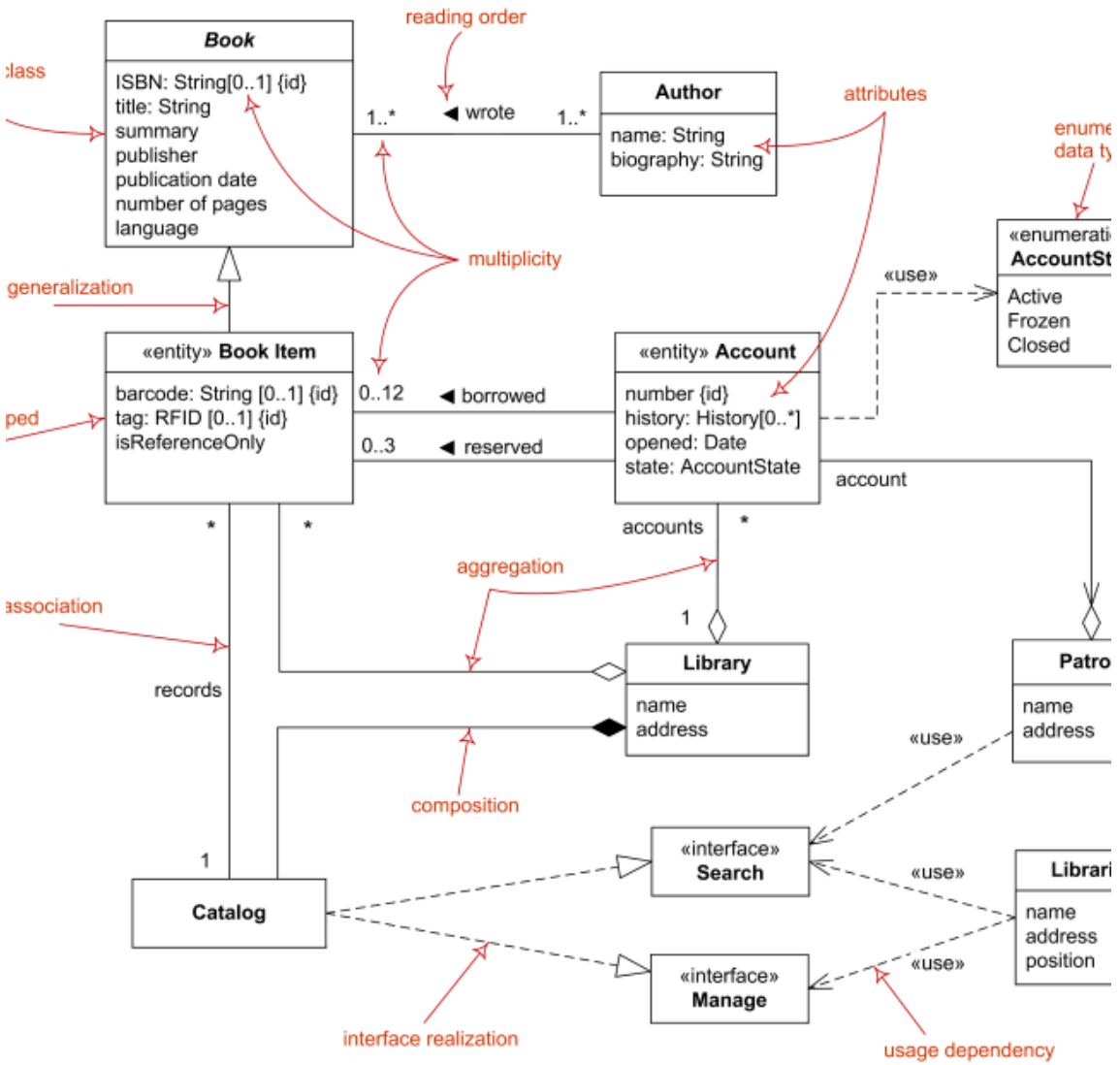
Louis-Edouard LAFONTANT

geodes

Université de Montréal

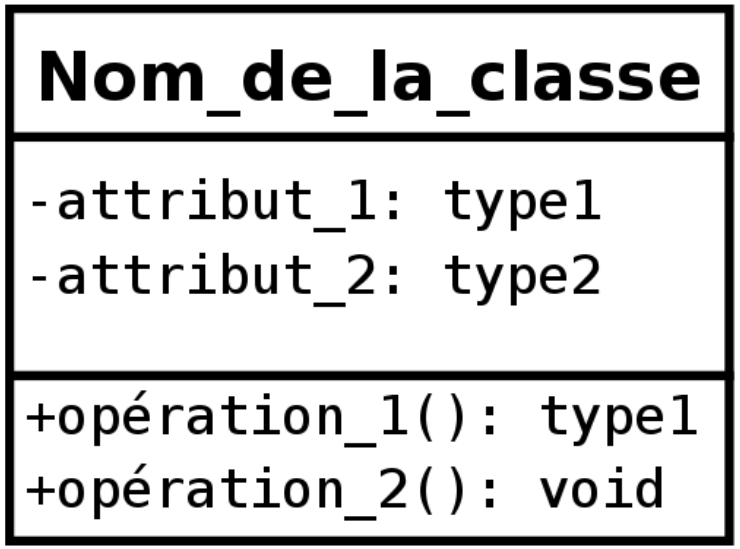
Diagramme de classes UML

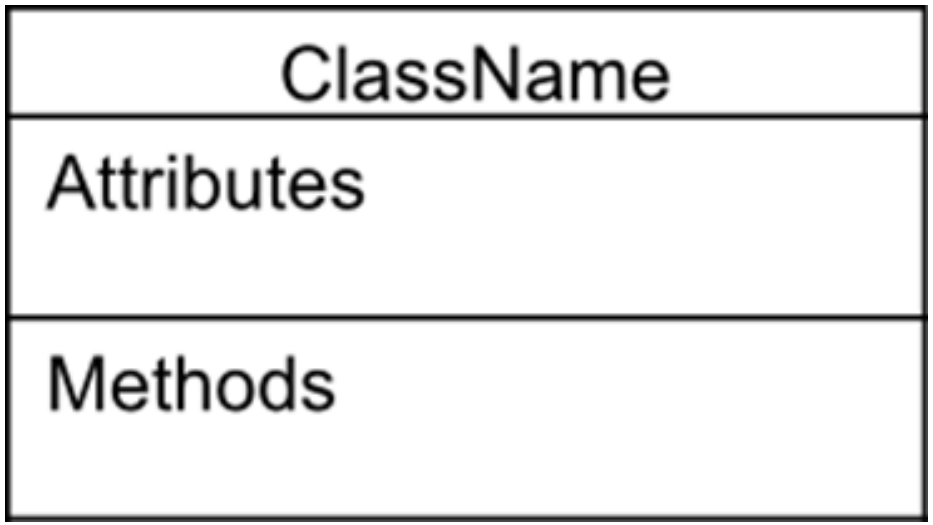
- Permet de décrire la structure statique du logiciel
- Indique les contraintes sur les objets et leurs relations
- **Classes** : attributs, opérations
- **Relations** : association, agrégation, composition, généralisation/spécialisation



Représentation des classes

- Compartiment du **Nom**
  - Unique
  - Noms répétées - *alias*
- Compartiment des **Attributs**
  - Caractéristiques des objets
- Compartiment des **Operations**
  - Comportement des objets





Paramètres

- **Attributs**

  [visibilité] [portée] <nom> : <type> [= <valeur_initiale>]

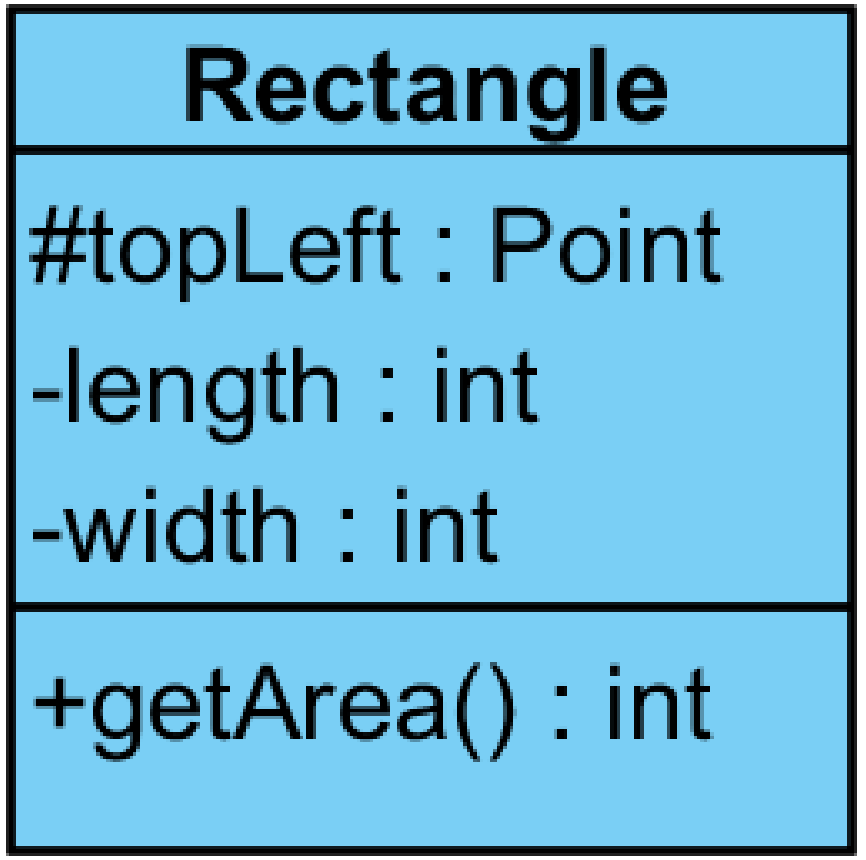
- **Opérations**

  [visibilité] [portée] <nom> (<params>) : <type_resultat>

Visibilité

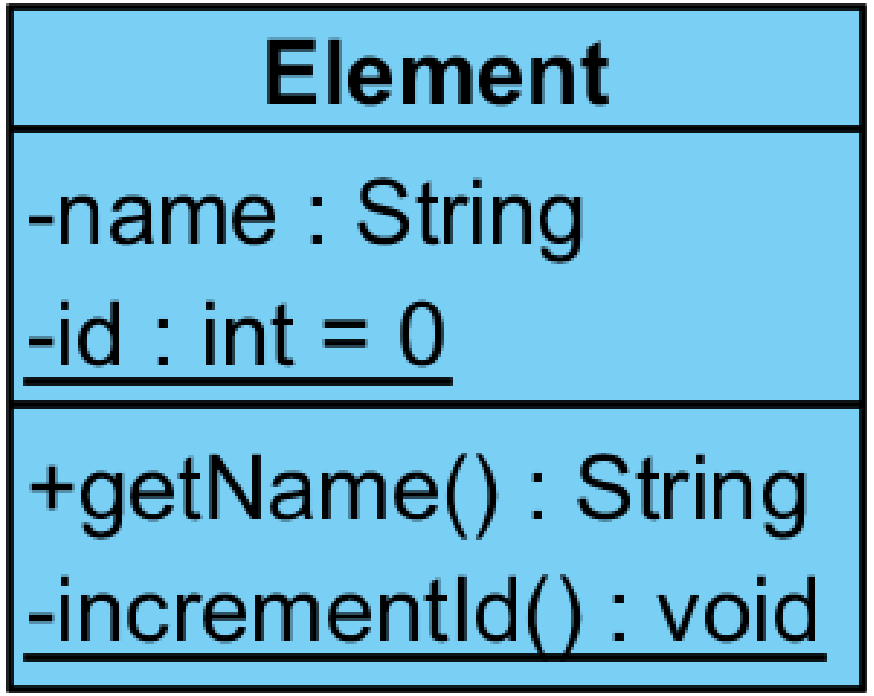
Ensemble de préfixes pour les attributs et les opérations

- + {Public}: élément visible par toutes les instances de toutes les classes
- # {Protected}: élément visible par toutes les instances de la classe et de ses sous-classes
- - {Private}: élément visible que par les instances de la classe
- ~ {Package}: élément visible par les classes du même paquet



Portée statique

- **Attribut d'instance**: chaque instance a ces propres valeurs d'attribut
- **Attribut de classe**: toutes les instances de la classe partagent la même valeur de l'attribut
- Opération de classe: opération appliquée à tous les éléments de la classe
  - Opération statique ne dépend que d'attributs et d'opérations statiques
- Représenté par nom souligné ou avec {static}
- Portée par défaut est d'instance



Type

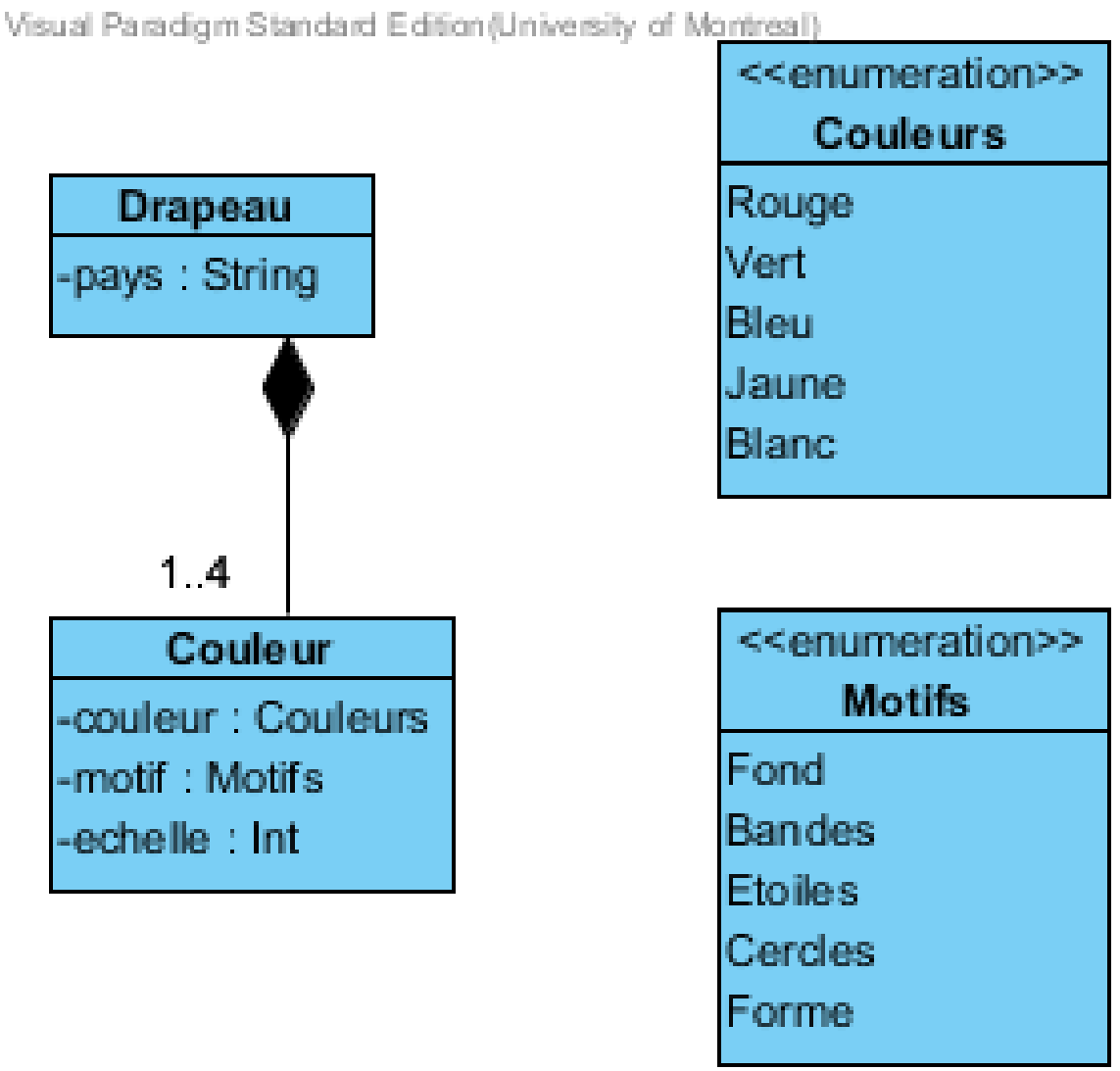
- Primitif : **boolean, real, integer, string, date**
- Enumération: **Enum**

- Types définis par une classe ou interface

- Favoriser l’utilisation d’association si l’attribut a pour type une autre classe

Énumération

Déclare des types discrets ou un ensemble fini de valeurs possibles



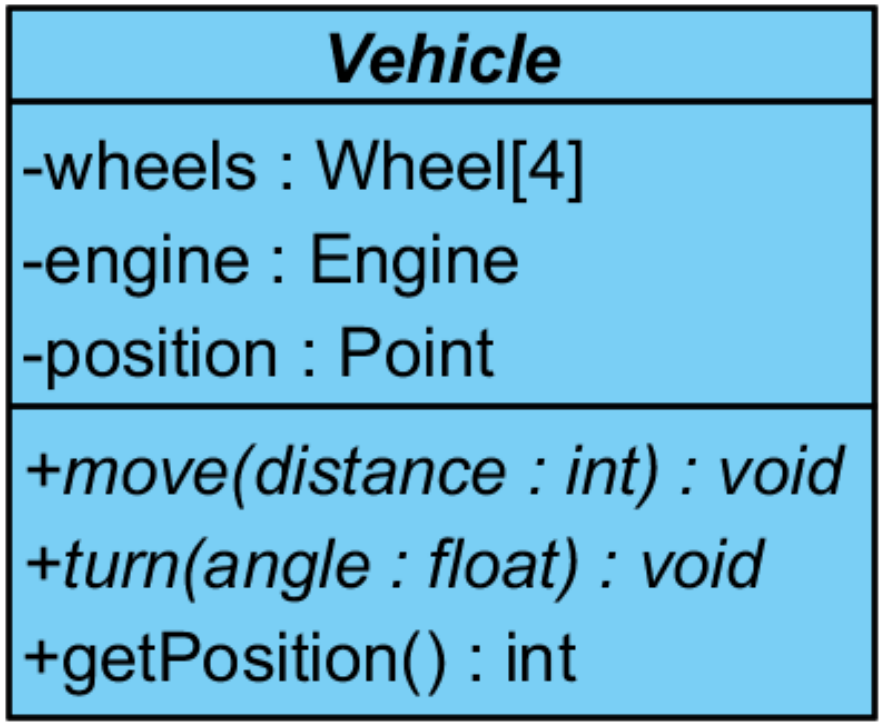
Paramètres des opérations

- <direction> <nom> : <type> [= valeur_initiale]

- <direction> := in | out | inout
  - Par défaut IN

Classe abstraite

- Les classes abstraites ne sont pas instanciables
- Les opérations abstraites ne disposent pas d'une implémentation
  - Une classe avec au moins une opération abstraite doit être abstraite
- Pour être utile, une classe abstraite doit être spécialisée
  - les classes concrètes sont forcées d'implémenter les opérations abstraites héritées
- Représenté avec *Italic* ou {abstract}
  - Utiliser {abstract} quand écrit à la main



Association

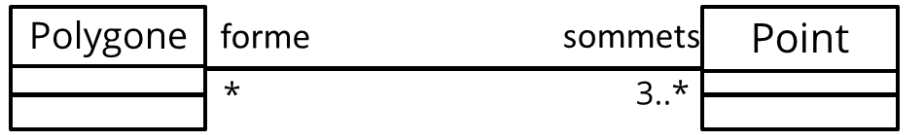
- Lien / connexion entre deux instances
  - Habituellement implémentée à l'aide d'attributs d'instances
  - Représentée visuellement par une ligne entre deux classes

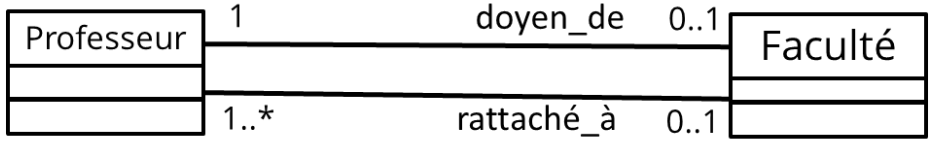
- Spécification
  - Cardinalité
  - Rôle(s)

Cardinalité

1	exactement une instance
0..1	optionnel
k	exactement *k* instances
ou 0..*	Plusieurs optionnel
k..*	au moins *k* instances
0..k	optionnel jusqu'à k
k..j	entre *k* et *j* instances
k,j	ou *k* ou *j* instances

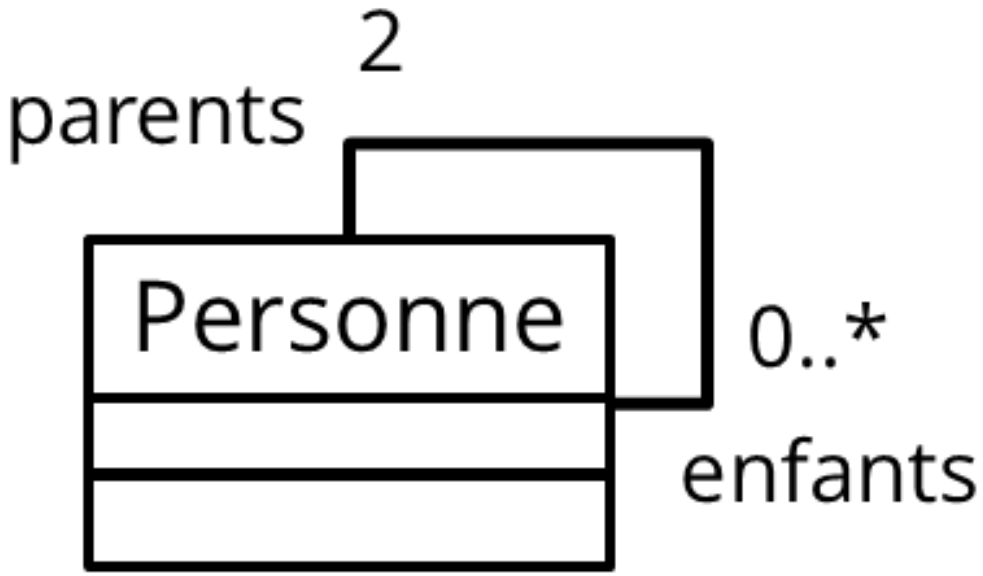
- Précise le nombre d'instances participantes
- min..max
  - *min* et *max* sont des entiers naturels
  - *max* peut être non borné (∗)
  - si le min et le max sont égaux, utilise un seul nombre





Rôle

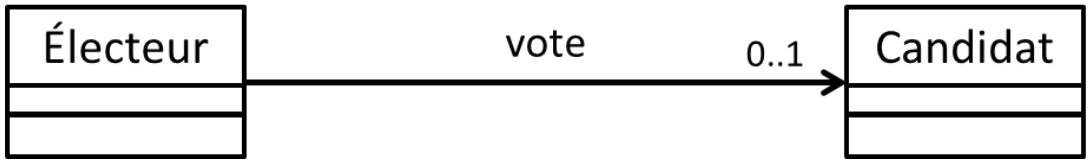
- Fonction sémantique que joue une classe dans une association
  - Chaque extrémité peut être annotée avec un rôle optionnel
- Information indispensable pour les associations réflexives



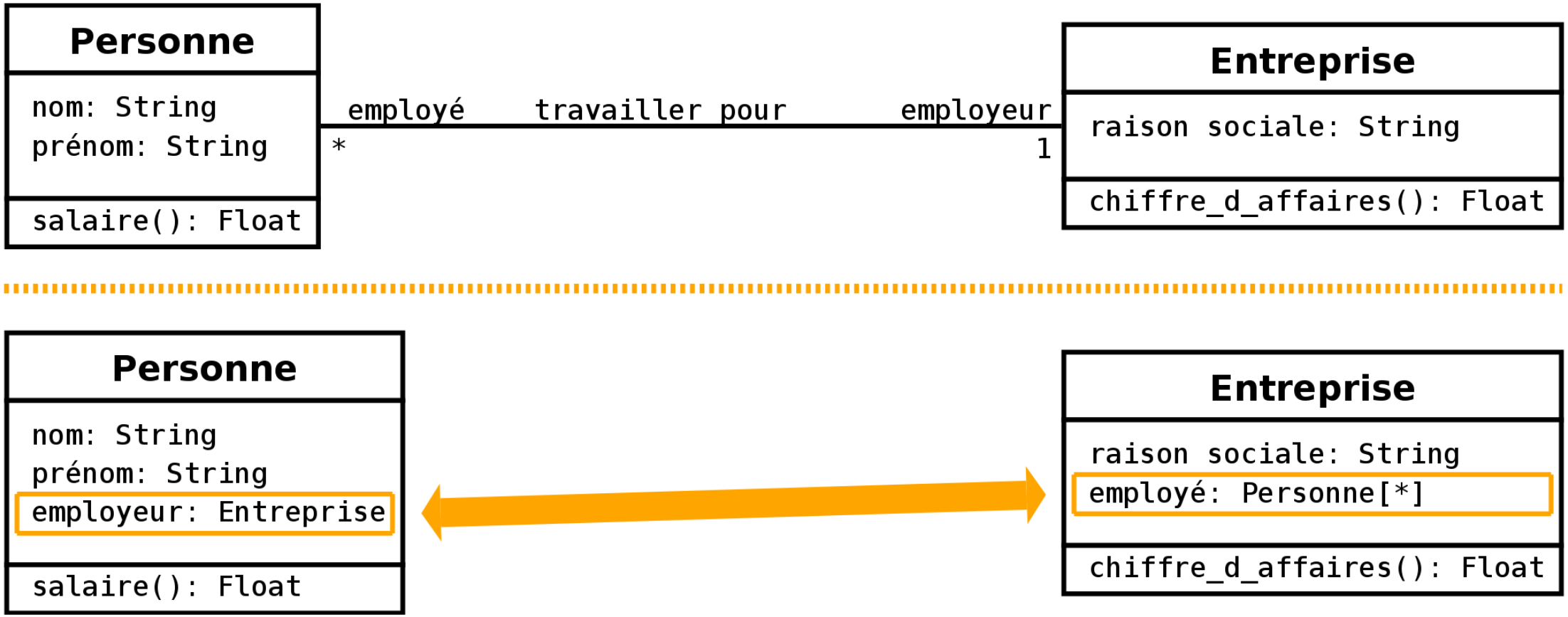
Navigabilité restreinte

- Par défaut, une association est bidirectionnelle
- Restreindre la navigabilité de l'association
- Association unidirectionnelle: classe cible ne peut pas identifier la source

À partir d'un électeur, on peut directement identifier le candidat pour lequel il a voté. À partir d'un candidat, on ne peut pas retrouver directement les électeurs qui ont voté pour lui.



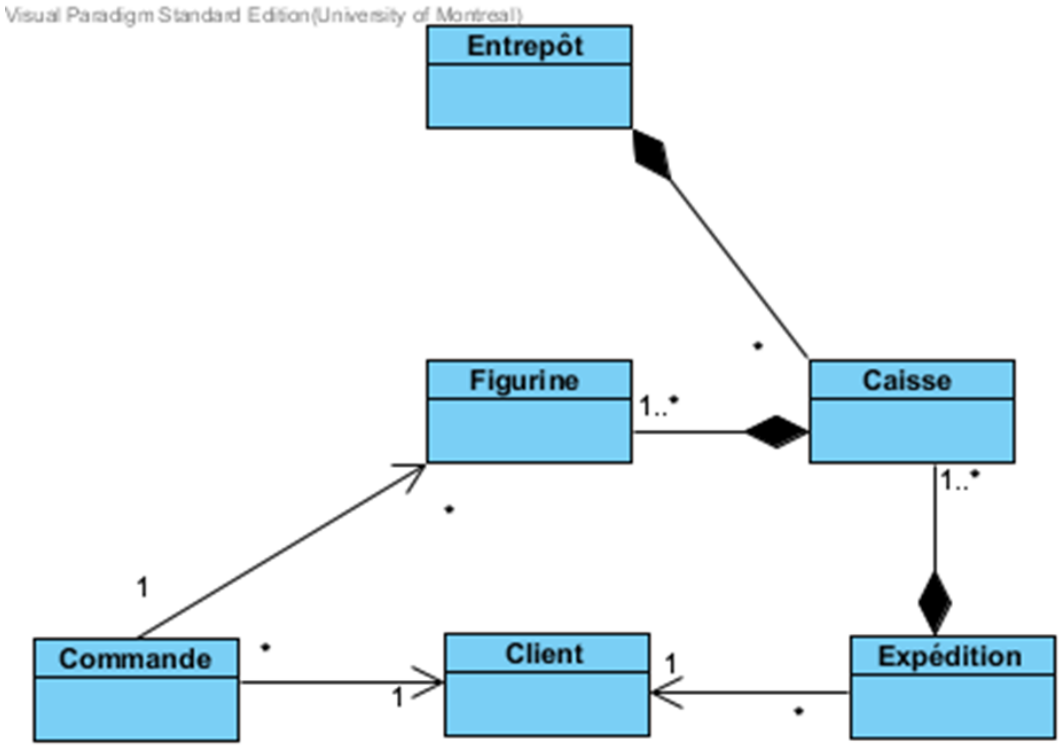
Équivalence avec les attributs



Composition

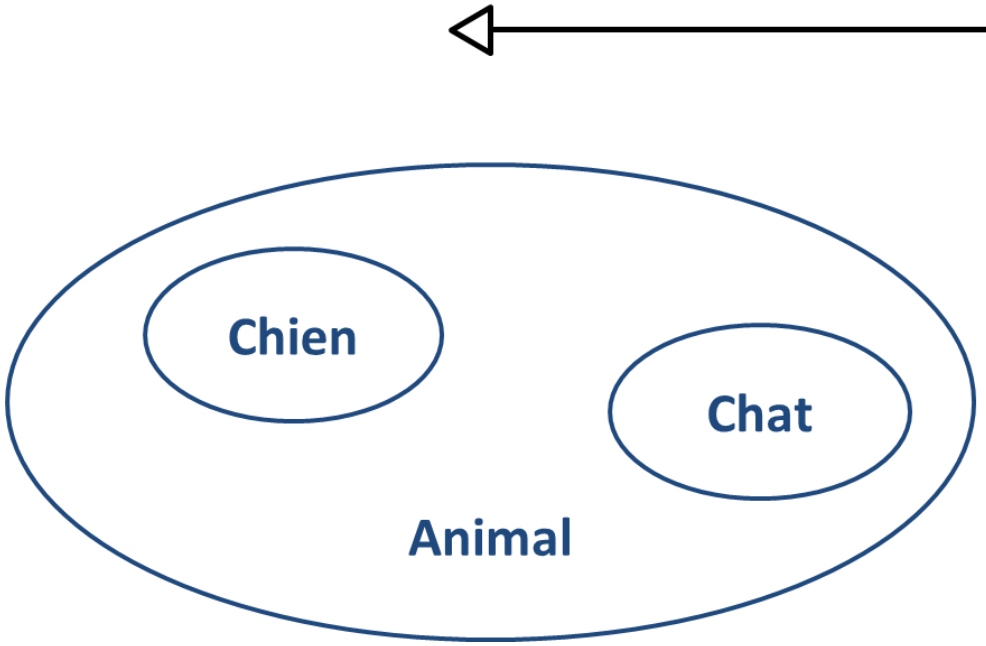
- Agrégation forte entre deux instances (composite et composant)
- Représente une relation « est dans » ou « fait partie de »
- **Chaque composant est une partie d'un seul composite**
- Si le composite est détruit (ou copié), ses composants le sont aussi

Représenté par un losange noir du côté composite



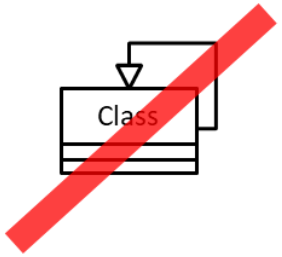
Généralisation

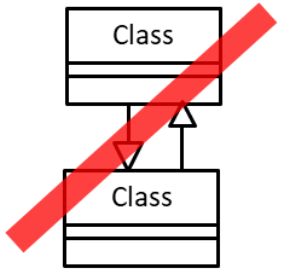
- Relation d'héritage entre un élément de description générale et un élément plus spécifique
- Représente la relation « est un(e) »
- Ex:
  - Un chien *est un* animal
  - Un chat *est un* animal

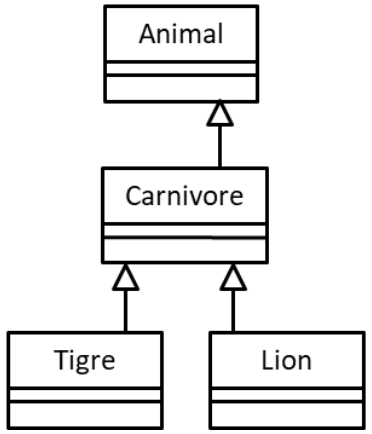


Généralisation

- Relation
  - Non réflexive
  - Non symétrique
  - **Transitive**

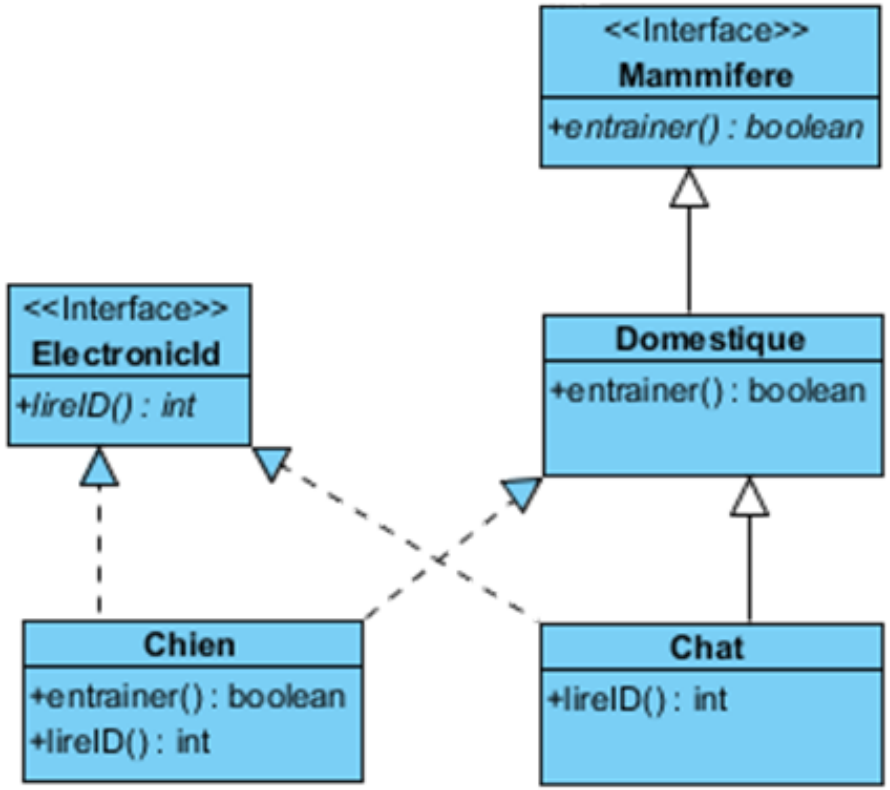






Interface/Implémentation

- Les interfaces ne peuvent avoir aucune implémentation
- Une classe qui implémente une interface doit fournir une implémentation de toutes les méthodes de cette interface
- Les interfaces peuvent être multiplement héritées,
  - Pas les classes abstraites



Contraintes + OCL

- Contraintes du diagramme de classes
  - Structurelles : associations, héritage, références…
  - De type : typage des attributs et signatures des méthodes
  - De visibilité, d'implémentation (abstrait ou non) etc…
- D'autres contraintes ne peuvent pas être exprimer sur le diagramme de classes

⇒ **OCL**

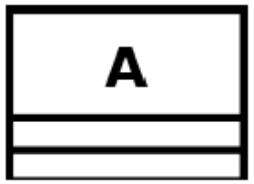
Object Constraint Language (OCL):

```
Context Personne inv:
self->closure(p : Personne | p.mère )->excludes(self)
```

Correspondance en code

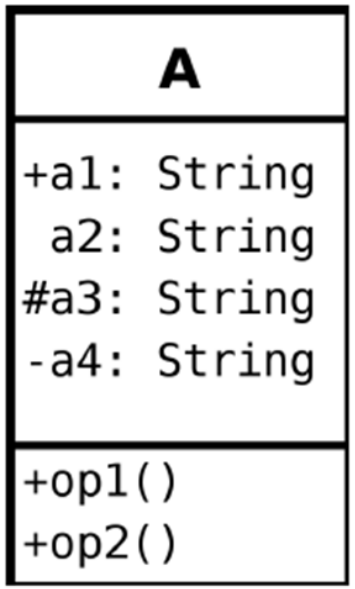
Diagramme de classes → Code JAVA

Classe vide




```
public class A {
  public A() {
    ...
  }
  protected void finalize() throws Throwable {
    super.finalize();
    ...
  }
}
```

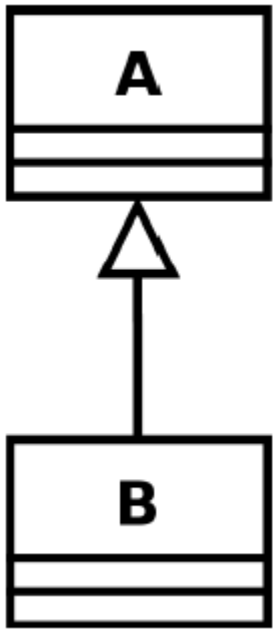
Classe avec attributs et opérations




```
public class A {
  public    String a1;
  package   String a2;
  protected String a3;
  private   String a4;
  public void op1() {
     ...
  }
  public void op2() {
     ...
  }
}
```

Classe abstraite et relation de généralisation

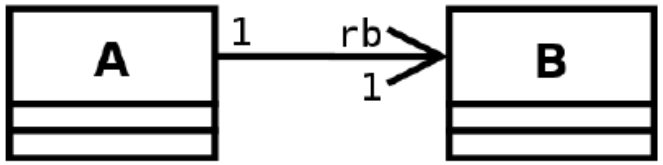

```
public abstract class A {
   ...
}
```




```
public class A {
   ...
}

public class B extends A {
   ...
}
```

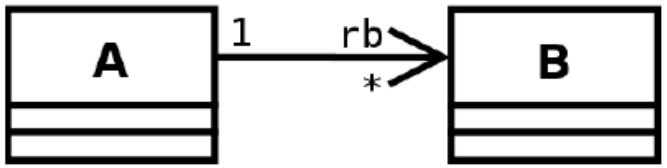
Relation unidirectionnelle




```
public class A {
  private B rb;
  public void addB( B b ) {
    if( b != null ) {
      this.rb=b;
    }
  }
}

public class B {
 ... // La classe B ne connaît pas l'existence de la classe A
}
```

Relation avec multiplicité




```
public class A {
  private ArrayList <B> rb;
  public A() { rb = new  ArrayList<B>(); }
  public void addB(B b){
    if( !rb.contains( b ) ) {
      rb.add(b);
    }
  }
}

public class B {
 ... // B ne connaît pas l'existence de A
}
```

Exemple

Modélisation d’un diagramme de classe

Spécifications

1. Un hôtel a les caractéristiques suivantes : une adresse, un nombre de pièces et une catégorie. Il est composé d'au moins deux chambres. Chaque chambre dispose d'une salle d'eau : douche ou bien baignoire. Une chambre est caractérisée par le nombre et de lits qu'elle contient, son prix et son numéro.
2. Un hôtel héberge des personnes. Il peut employer du personnel et il est impérativement dirigé par un directeur. On ne connaît que le nom et le prénom des employés, des directeurs et des occupants. Certaines personnes sont des enfants et d'autres des adultes (faire travailler des enfants est interdit).
3. On veut pouvoir savoir qui occupe quelle chambre à quelle date. Pour chaque jour de l'année, on veut pouvoir calculer le loyer de chaque chambre en fonction de son prix et de son occupation (le loyer est nul si la chambre est inoccupée). La somme de ces loyers permet de calculer le chiffre d'affaires de l'hôtel entre deux dates.

Modèle proposé