UML 2 – Diagramme de classes

Laurent Audibert

Institut Universitaire de Technologie de Villetaneuse
Département Informatique

15 février 2011

Introduction

- **Langage de POO :** moyen spécifique d'implémenter le paradigme objet (pointeurs ? héritage multiple ? . . .)
- **Diagramme de classes :** permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier
- Diagramme le plus important de la modélisation objet
- Le seul obligatoire lors d'une modélisation objet

Introduction

- **Diagramme de cas d'utilisation :** système du point de vue des acteurs
- **Diagramme de classes :** structure interne
  $\rightarrow$ représentation abstraite des objets du système (conceps du domaine et concepts internes) qui vont interagir ensemble pour réaliser les cas d'utilisation
- Les cas d'utilisation ne réalisent pas une partition des classes du diagramme de classes
- Il s'agit d'une vue statique (on ne tient pas compte du facteur temporel)
- Principaux éléments : **classes** et leurs **relations**

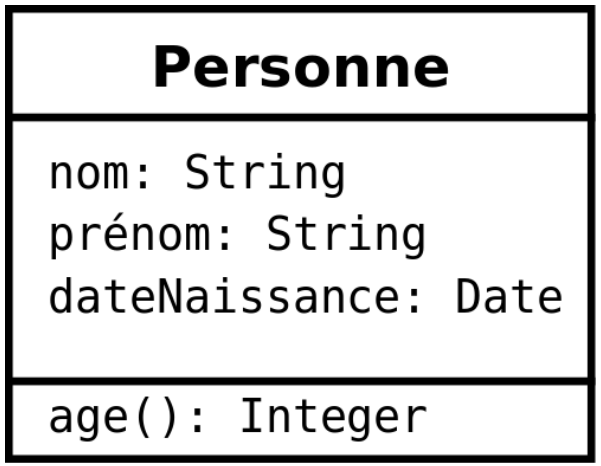
Notion d'instance

- **Une instance** est une concrétisation d'un concept abstrait
- Exemples :
  - Concept : voiture / Instance : la Ferrari *Enzo* qui se trouve dans votre garage
  - Concept : Amitié / Instance : l'amitié entre Jean et Marie
  - Concept : L'incivilité / Instance : l'automobiliste Pierre qui n'a pas laissé passer le pieton Paul pourtant engagé sur la chaussé
  - Concept : type *int* / Instance : variable i de type *int*

Notion de classe

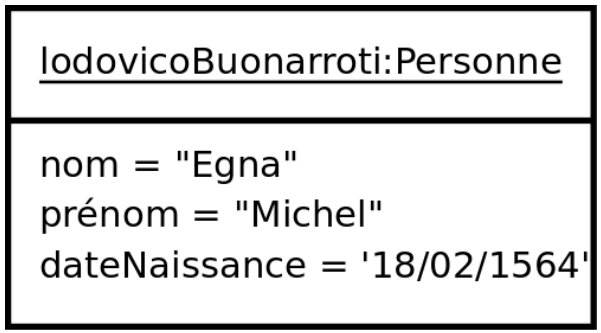
- Une classe est la description formelle d'un ensemble d'objets ayant :
  - une sémantique
  - et des caractéristiques (attributs, méthodes et relations)

  communes
- Une classe peut être instanciée
- L'instance d'une classe est un objet



Notion d'objet

- Un objet est une instance d'une classe
- C'est une entité discrète dotée :
  - d'une identité
  - d'un état
  - et d'un comportement que l'on peut invoquer

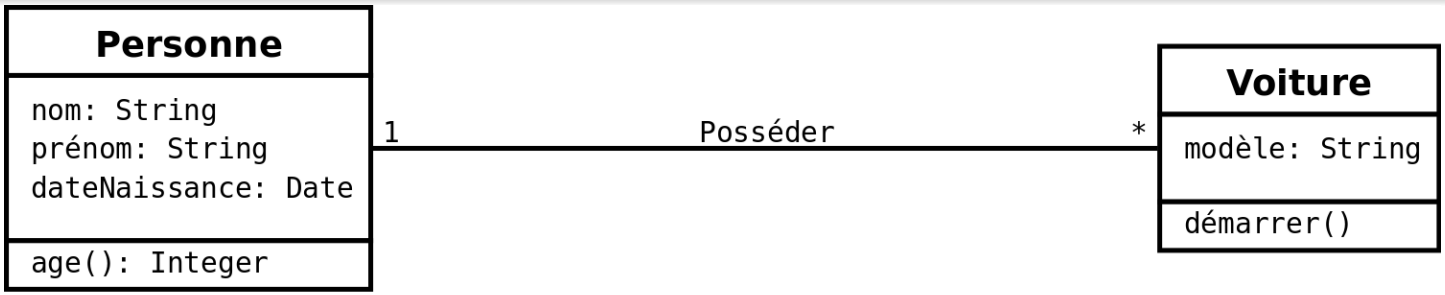


- Les objets sont des éléments individuels d'un système en cours d'exécution

Notions d'association

Association

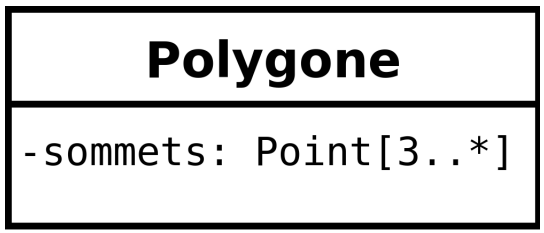
Une association est une relation entre des classes qui décrit les connexions structurelles entre leurs instance

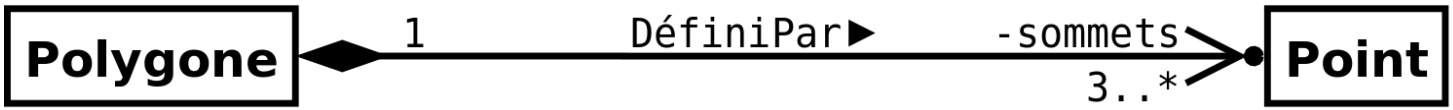


Comment une association doit-elle être modélisée ?

Notions d'association

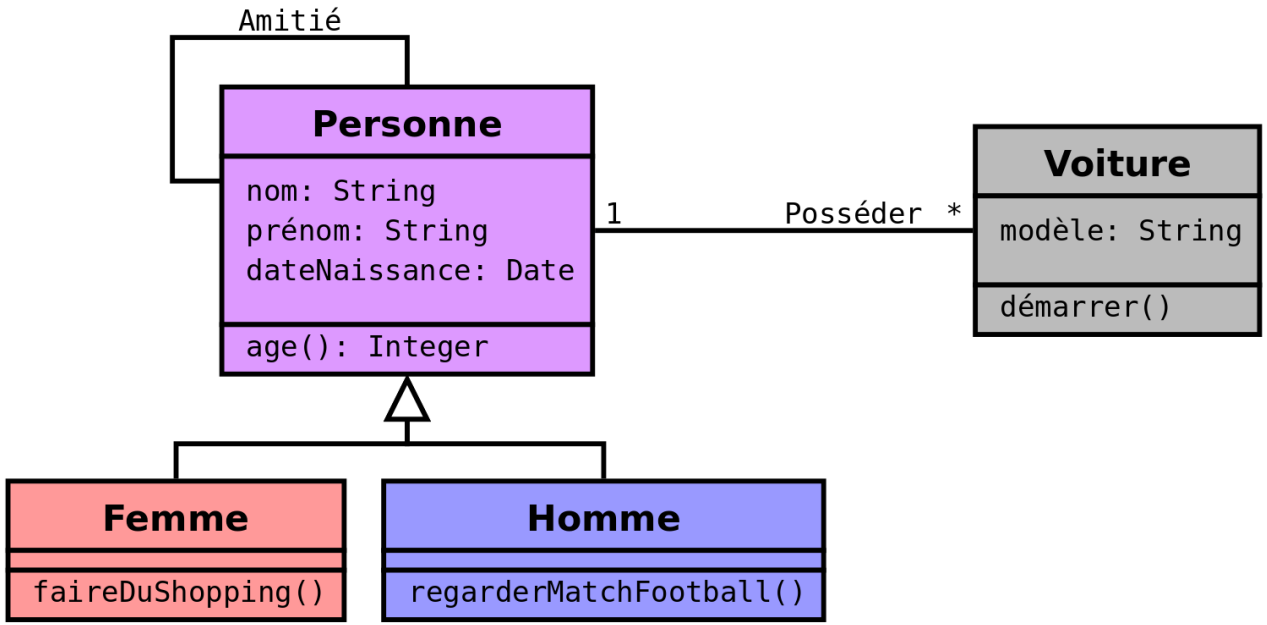
Terminaisons d'association et **Attributs** sont deux éléments conceptuellement très proches regroupés sous le terme de générique de **propriétés structurelles**





Pour UML, un attribut peut être considéré comme une association dégénérée dans laquelle une terminaison d'association est détenue par une classe

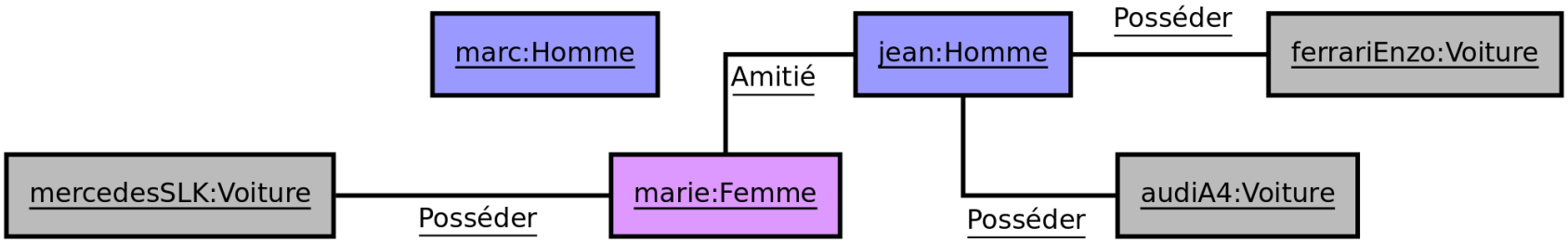
Notions de diagramme de classes



- Il modélise la structure statique d'un système
- Il représente graphiquement les classes interconnectées par des associations ou des relations de généralisation

Le diagramme de classes modélise les règles

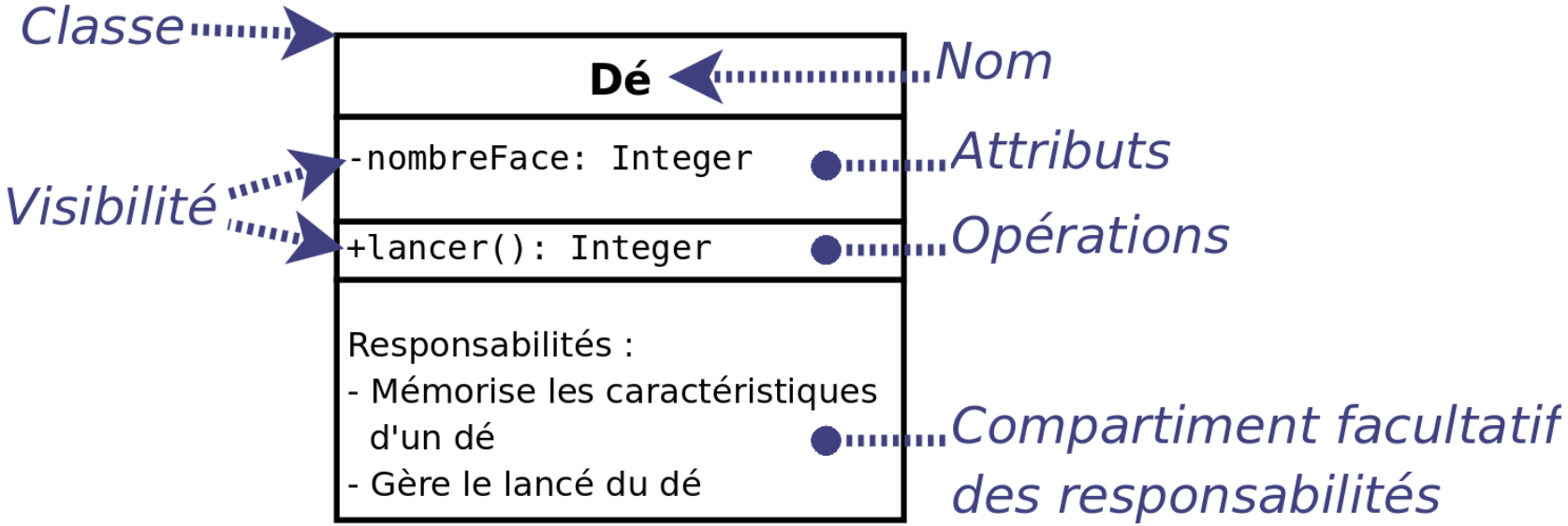
Notions de diagramme d'objets



- Un diagramme d'objets représente des objets (*instances de classes*) et leurs liens (*instances de relations*)
- Le type d'un objet est précédé d'un « : » et est souligné
- On peut ajouter l'identifiant de l'objet devant le « : »
- Le nom des associations est souligné

Le diagramme d'objets modélise des faits

Représentation graphique



- **Nom :** le nom de la classe doit évoquer le concept décrit par la classe, il commence par une majuscule

Encapsulation

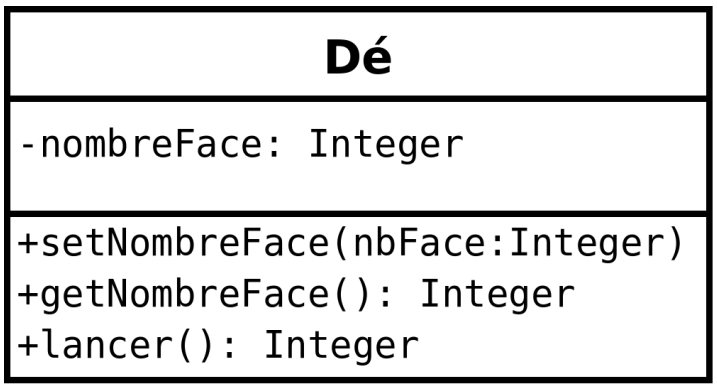
Consiste à masquer les détails d'implémentation d'une classe

Interface

Les services accessibles de la classe (sa vue externe)

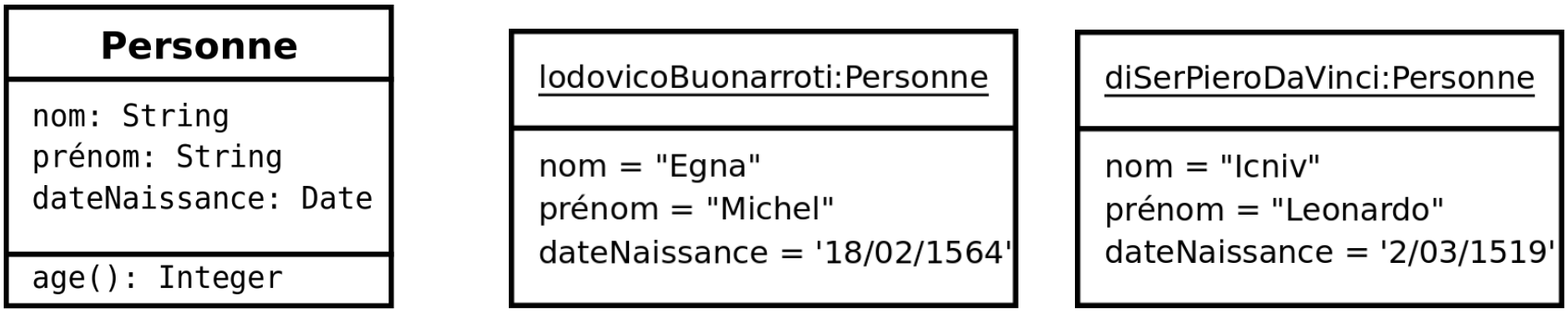
Visibilité

Degré d'accessibilité depuis un autre espace de noms



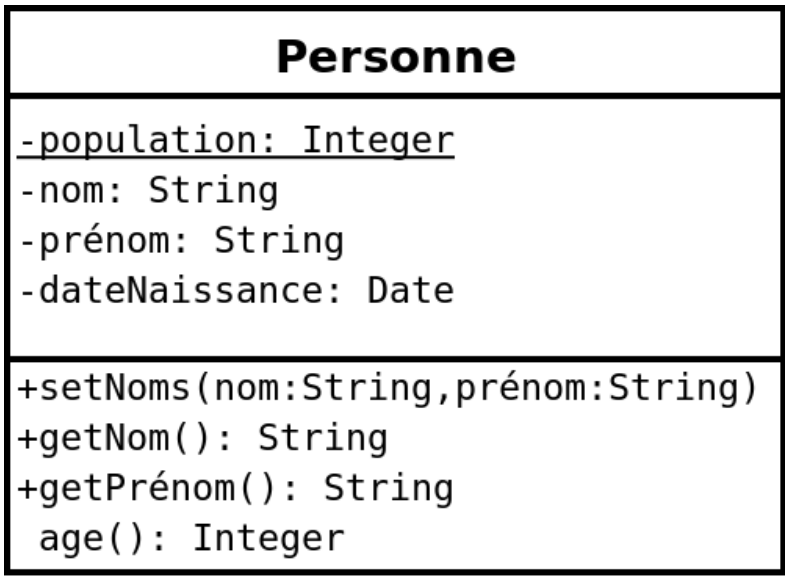
- $+$: visible partout
- $\#$: visible dans la classe et ses descendants
- $-$: visible uniquement dans la classe
- $\sim$: visible uniquement dans le paquetage (visibilité par défaut)

Attribut d'instance



- Les attributs définissent des informations qu'une classe ou un objet doivent connaître
- Chaque instance possède sa propre copie des attributs
  $\rightarrow$ leur valeur peut différer d'un objet à un autre
- Chaque attribut est défini par un nom, un type, une visibilité, une multiplicité et peut être initialisé

Attribut de classe



- Garde une valeur unique et partagée par toutes les instances
- Les instances ont accès à cet attribut mais n'en possèdent pas une copie
- L'accès à cet attribut ne nécessite pas l'existence d'une instance
- Pas propriété d'une instance
  $\rightarrow$ Propriété de la classe
- Un attribut de classe est souligné

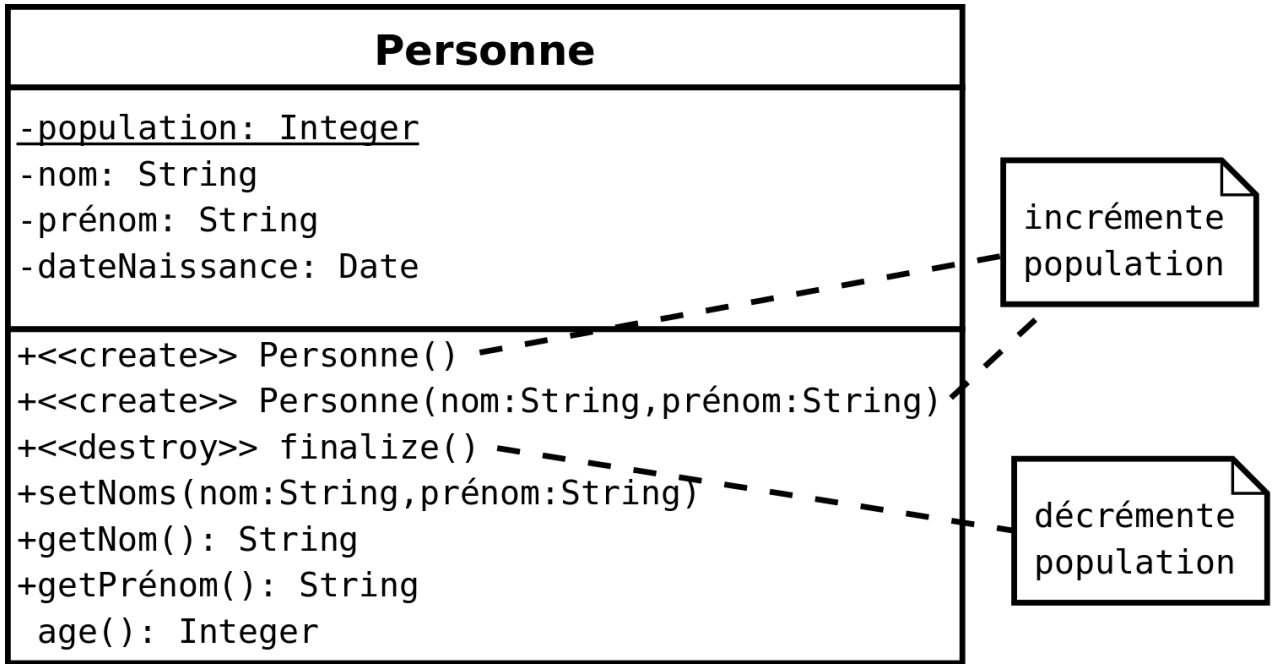
Attributs dérivés

- Peuvent être calculés à partir d'autres attributs et de formules de calcul
- Les attributs dérivés sont symbolisés par l'ajout d'un « / » devant leur nom
- Exemple : age d'une personne.

Opération

- Les opérations décrivent des services qui peuvent être invoqués au titre d'un objet pour déclencher un comportement
- Ce sont des fonctions qui peuvent prendre des valeurs en entrée et modifier les attributs ou produire des résultats
- L'implémentation d'une opération est appelée une méthode

Constructeurs et Destructeurs



- Un constructeur est une opération appelée lors de la création d'un objet
- Un destructeur est une opération appelée à la fin de la vie d'un objet

Opération de classe

Personne
-population: Integer -nom: String -prénom: String -dateNaissance: Date
+getPopulation(): Integer +<<create>> Personne() +<<create>> Personne(nom:String,prénom:String) +<<destroy>> finalize() +setNoms(nom:String,prénom:String) +getNom(): String +getPrénom(): String age(): Integer

- Ne peut manipuler que des *attributs de classe* et ses propres paramètres
- N'a pas accès aux attributs *de la classe*
- L'accès à une opération de classe ne nécessite pas l'existence d'une instance de cette classe
- Graphiquement, une méthode de classe est soulignée

```
Personne P1 ;
Personne P2("Jaques","Dupont") ;
P1.setNoms("Jean","Durand") ;
System.out.println(P2.getPrénom()) ;
System.out.println(Personne.getPopulation()) ;
```

Opération et classes abstraites

Opération abstraite

Opération dont la méthode associée n'est pas définie

Classe abstraite

Classe déclarée abstraite ou possèdant une méthode abstraite (qui peut être héritée)

- On ne peut instancier une classe abstraite
  $\rightarrow$ elle est vouée à se spécialiser
- Une classe abstraite peut contenir des méthodes concrètes

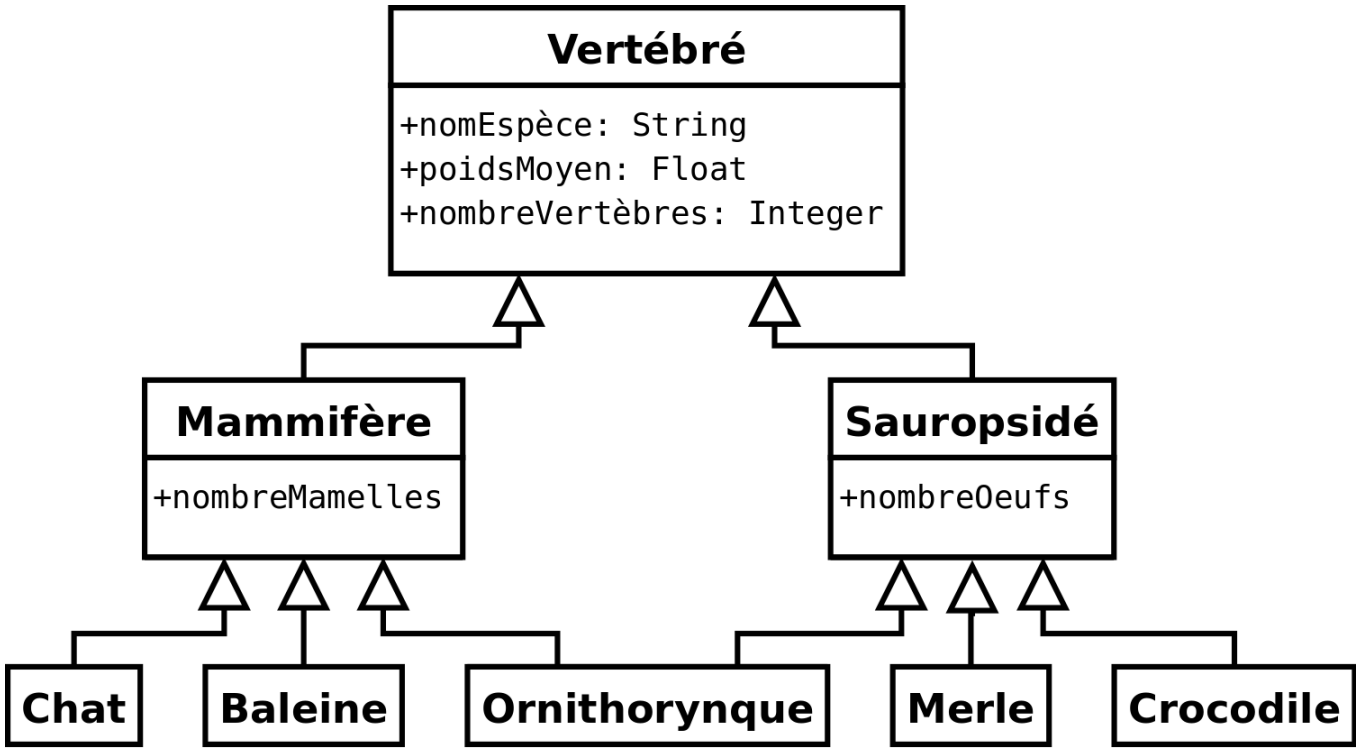
Généralisation et Héritage : définition

Relation de généralisation $\rightarrow$ concept d'héritage

Relation de généralisation

- Décrit une relation entre une classe générale (classe de base ou classe parent) et une classe spécialisée (sous-classe)
- La classe spécialisée est intégralement cohérente avec la classe de base (héritage), mais comporte des informations (attributs, opérations, associations) supplémentaires (spécialisation)

Généralisation et Héritage : exemple

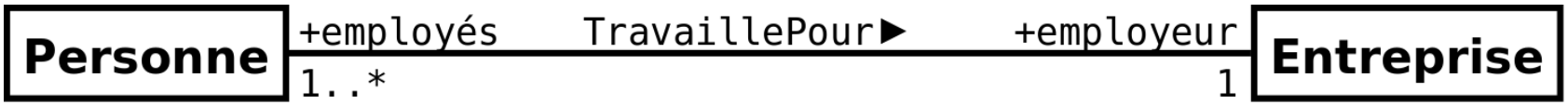


Graphiquement : flèche avec un trait pleins dont la pointe est un triangle fermé désignant le cas le plus général

Généralisation et Héritage : propriétés

- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue
- La classe enfant possède toutes les propriétés de ses classes parents (attention toutefois à la visibilité)
- Une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent
- Un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes
- Toutes les associations de la classe parent s'appliquent aux classes dérivées
- Une classe peut avoir plusieurs parents $\rightarrow$ on parle alors d'héritage multiple

Association : définition

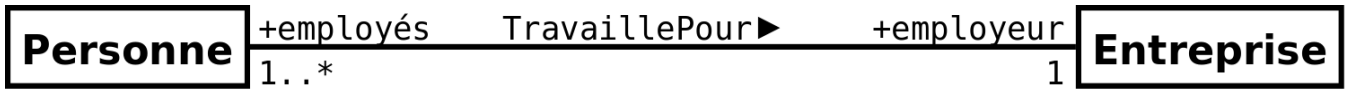


Association

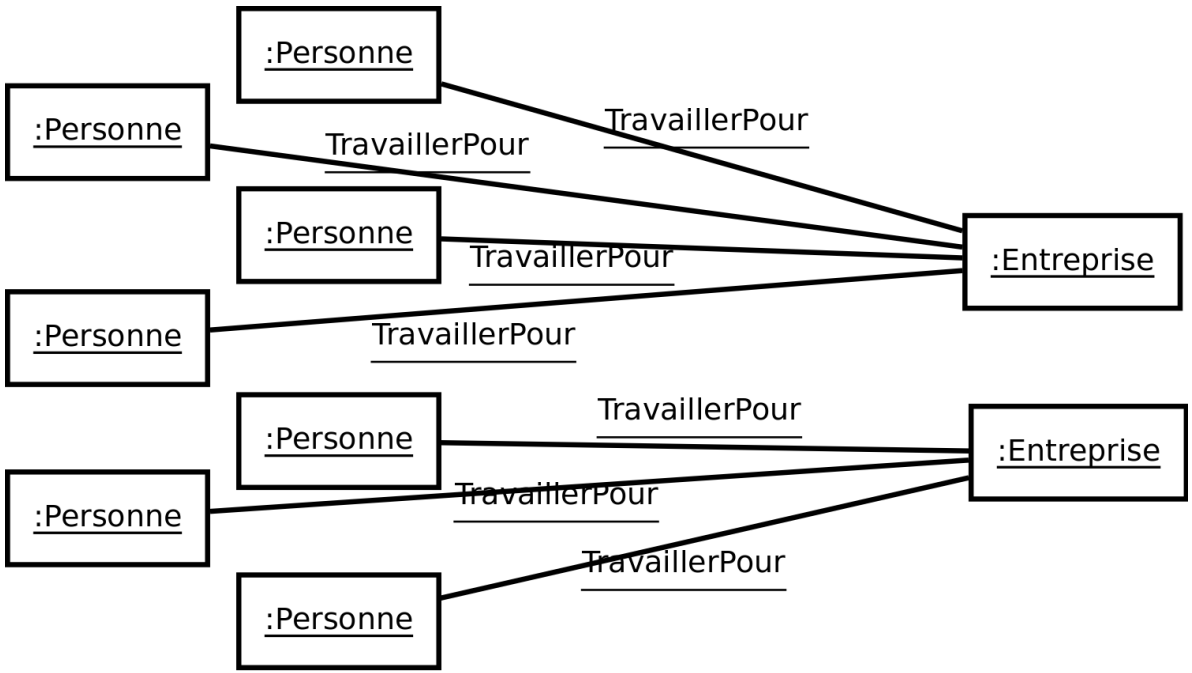
Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions entre leurs instances

- Objectif : permettre la navigation d'une instance vers l'autre
- Elle peut être ornée d'un nom et d'un sens de lecture (▶ ou ◀)
- Une association binaire est matérialisé par un trait plein entre les classes associée

Association : diagramme d'objets

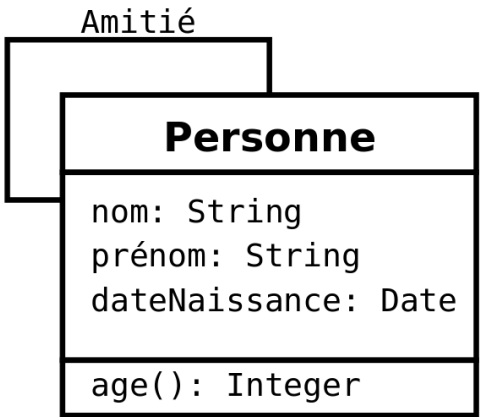


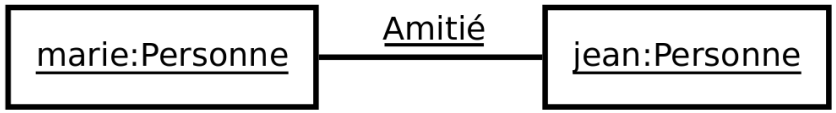
- Le diagramme de classes modélise les règles



- Le diagramme d'objets modélise des faits

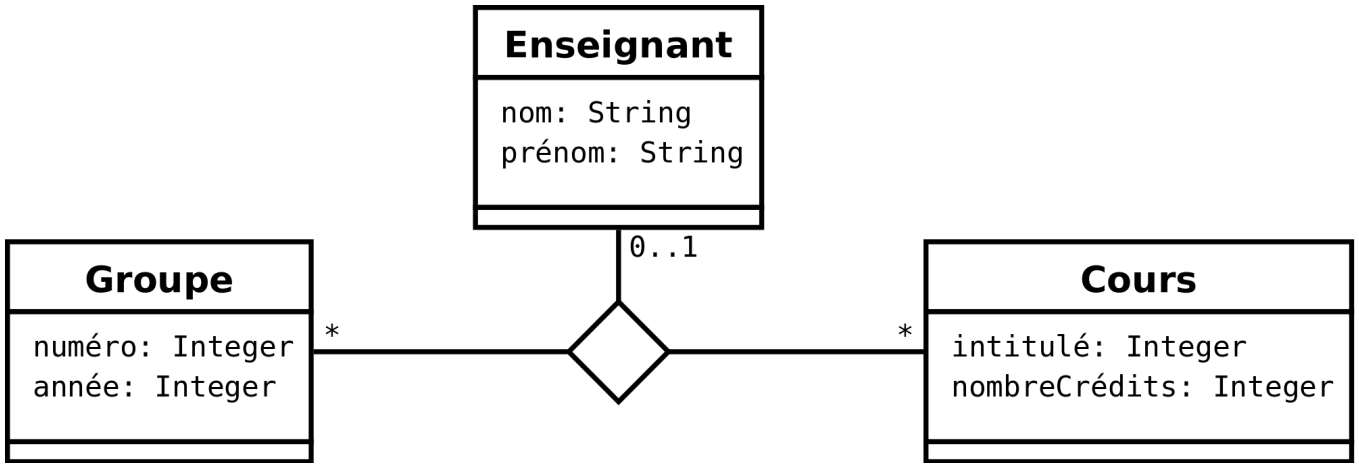
Association réflexive





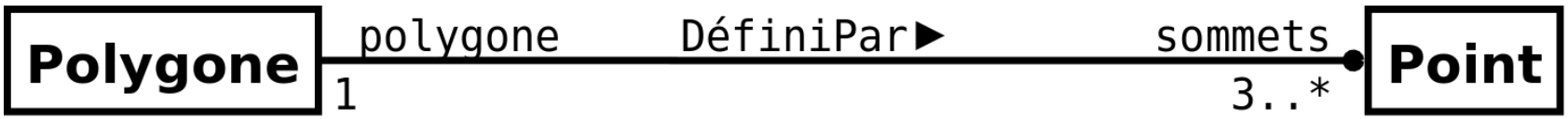
- Quand les deux extrémités de l'association pointent vers la même classe, l'association est dite réflexive
- La réflexivité de l'association n'implique donc pas que les liens correspondants soient également réflexifs

Association n-aire



- Une association n-aire lie plus de deux classes.
- On représente une association n-aire par un grand losange avec un chemin partant vers chaque classe participante, le nom de l'association peut apparaître à proximité du losange.

Multiplicité ou cardinalité : association binaire

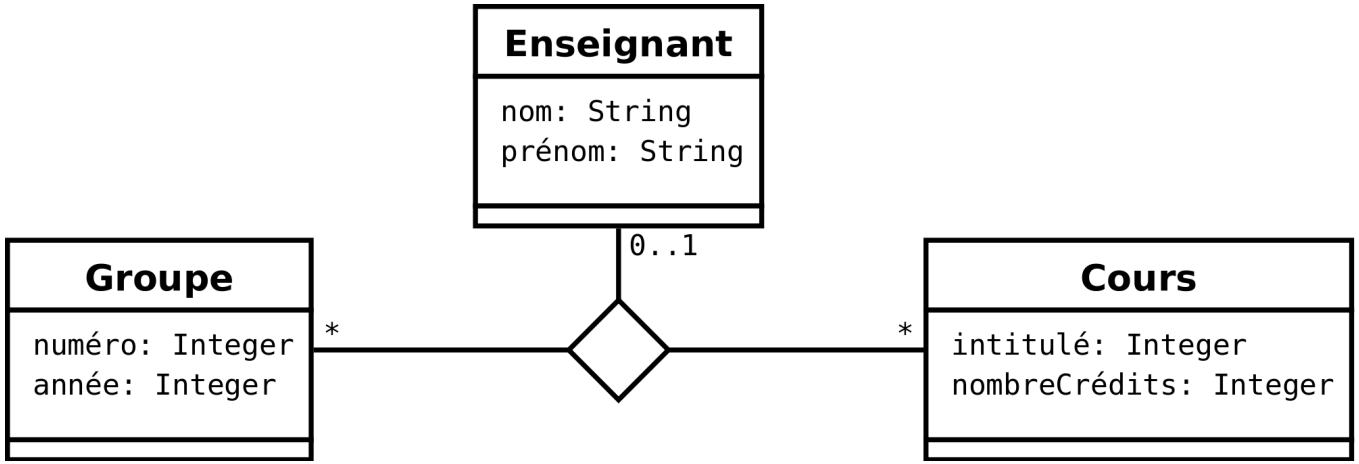


Multiplicité ou cardinalité

Dans une association binaire, la multiplicité sur la terminaison cible contraint le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association).

- exactement un : 1 ou 1..1
- plusieur : $*$ ou 0..$*$
- au moins un : 1..$*$
- de un à six : 1..6

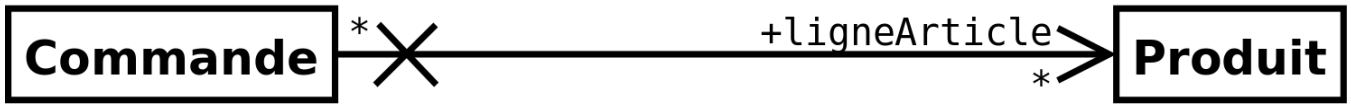
Multiplicité ou cardinalité : association n-aire



Multiplicité ou cardinalité

Dans une association n-aire, la multiplicité apparaissant sur une terminaison d'association doit être vérifiée pour toute combinaison d'instances de chacune des classes associées, à l'exclusion de la classe-association et de la classe située à la terminaison d'association considérée

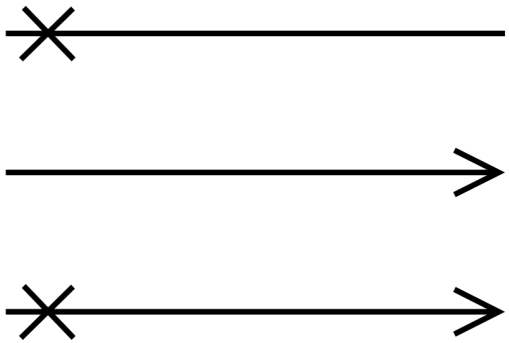
Navigabilité



- La navigabilité indique s'il est possible de traverser une association
- On représente graphiquement la navigabilité par une flèche du côté de la terminaison navigable
- On empêche la navigabilité par une croix du côté de la terminaison non navigable

Navigabilité

- Par défaut, une association est navigable dans les deux sens
- Ces trois notations de navigabilité veulent dire la même chose :



Agrégation

Agrégation

Association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble

- Graphiquement, on ajoute un losange vide ($\diamondsuit$) du côté de l'agrégat
- La signification de cette forme simple d'agrégation est uniquement conceptuelle

Composition

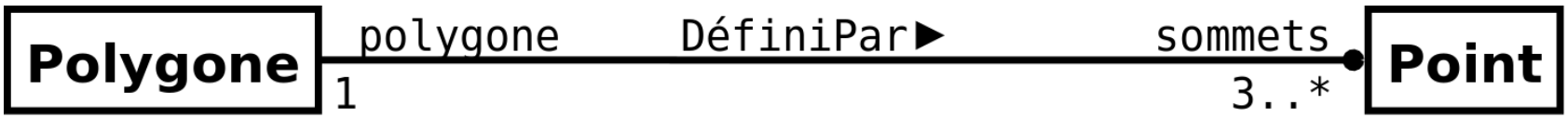
Composition

La composition, également appelée agrégation composite, décrit une contenance structurelle entre instances

- La destruction de l'objet composite implique la destruction de ses composants
- Une instance de la partie appartient toujours à au plus une instance de l'élément composite (multiplicité 1 ou 0..1)
- Graphiquement, on ajoute un losange plein (♦) du côté de l'agrégat

Propriétaire d'une terminaison d'association

- Une terminaison d'association est une propriété structurelle au même titre qu'un attribut
- *Mais qui est le propriétaire d'une terminaison d'association ?*
  - soit la classe située à l'autre extrémité de l'association
  - soit l'association



- Un petit cercle plein permet de préciser que c'est la classe située à l'autre extrémité de l'association
- Si le diagramme ne comporte aucune possession, la possession est ambigüe

Association vs. Attribut

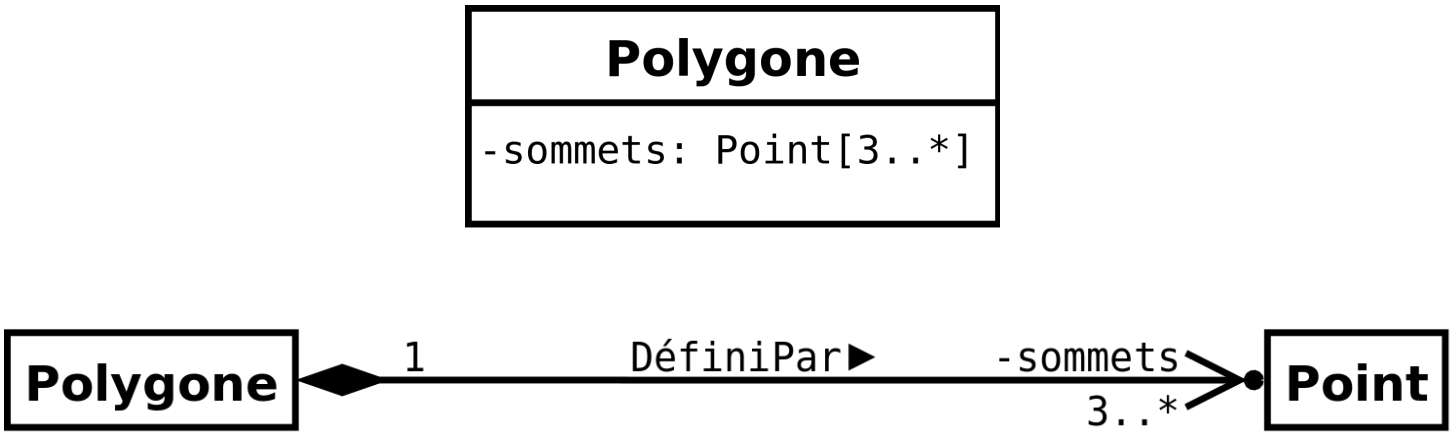
- Terminaisons d'associations et attributs sont deux éléments conceptuellement très proches : **propriétés structurelles**
- Les paramètres des terminaisons d'associations sont les mêmes que ceux des attributs (nom, visibilité, multiplicité. . .)

Association vs. Attribut

- Terminaisons d'associations et attributs sont deux éléments conceptuellement très proches : **propriétés structurelles**
- Les paramètres des terminaisons d'associations sont les mêmes que ceux des attributs (nom, visibilité, multiplicité. . .)

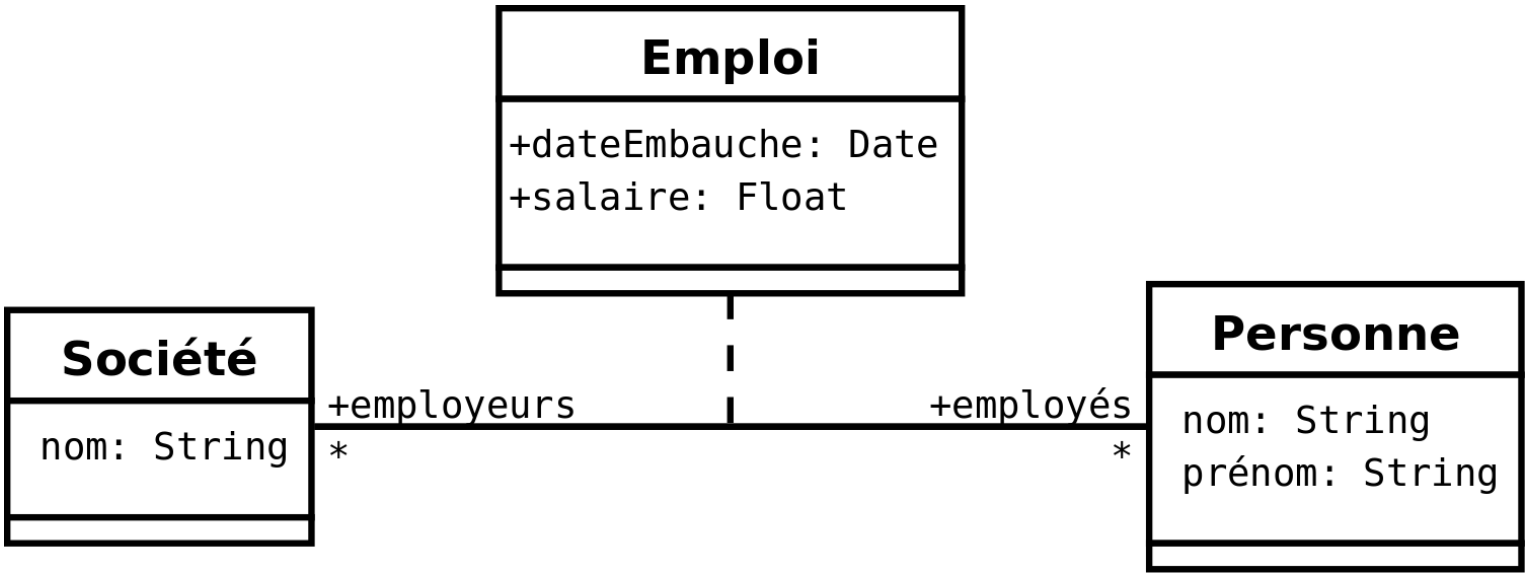
	Attribut	**Terminaison d'association**
Nom	obligatoire	*nom du rôle*, à proximité de la terminaison (facultatif)
Visibilité	devant le nom de l'attribut	devant le nom du rôle
Multiplicité	la multiplicité par défaut est 1	facultatif, la multiplicité par défaut est *non spécifiée*
Navigabilité	implicite, navigable depuis la classe vers l'attribut	paramètrable
Possession	implicite, la classe possède ses attributs	paramètrable

Équivalence association/attribut



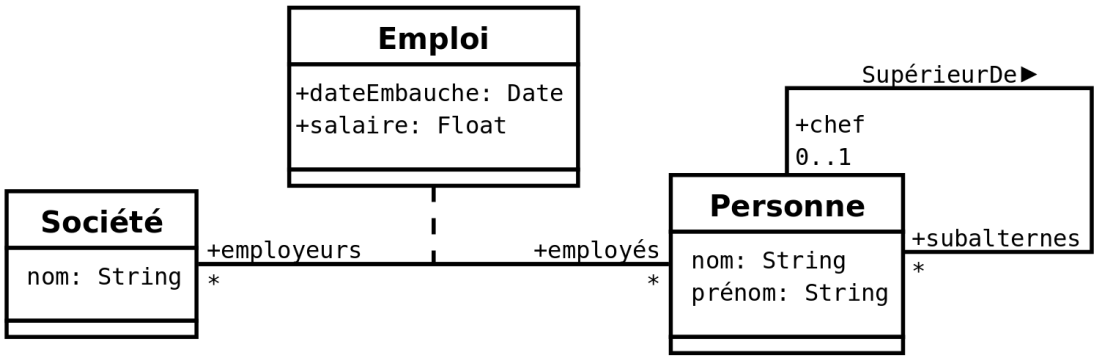
Un attribut est une association dégénérée dans laquelle une terminaison d'association est détenue par un classeur (généralement une classe). Le classeur détenant cette terminaison d'association devrait théoriquement se trouver à l'autre terminaison, non modélisée, de l'association. Un attribut n'est donc rien d'autre qu'une terminaison d'un cas particulier d'association.

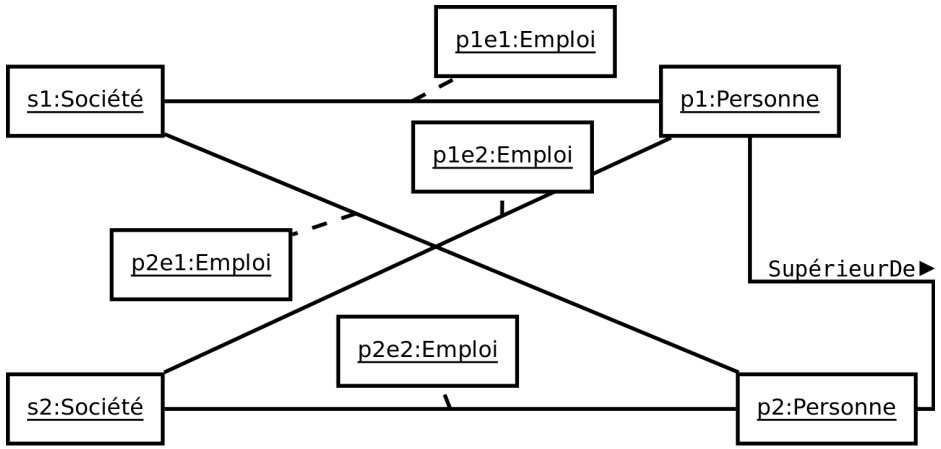
Classe-association



- Elle possède les propriétés des associations et des classes :
  - se connecte à deux ou plusieurs classes
  - possède des attributs et des opérations
- Elle est constituée d'un classeur relié par un trait discontinu à une association

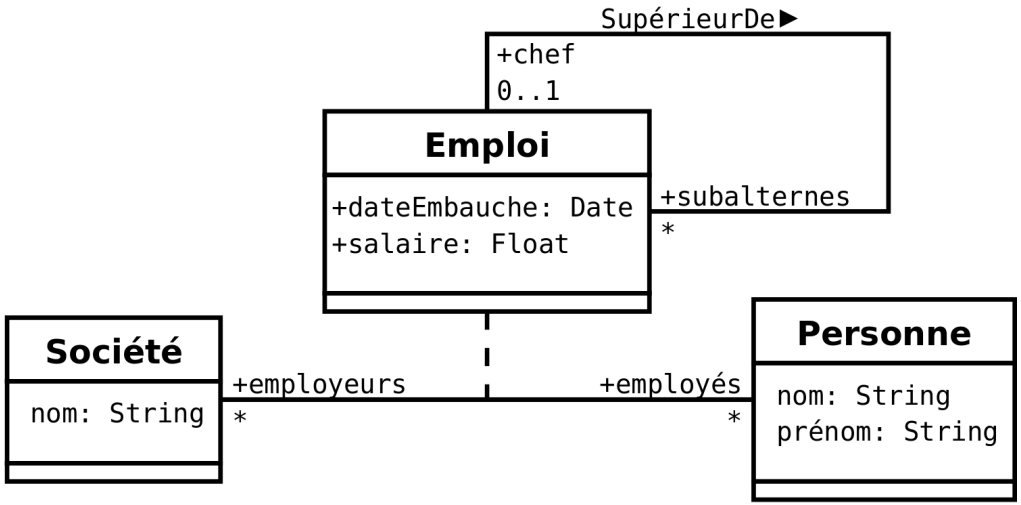
Auto-association sur classe-association : problématique

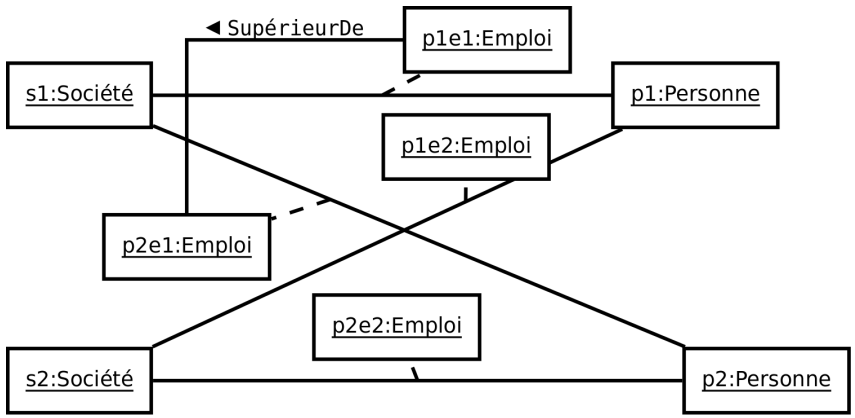




p1 est le supérieur de *p2* dans quelle société (*s1* ou *s2*) ?

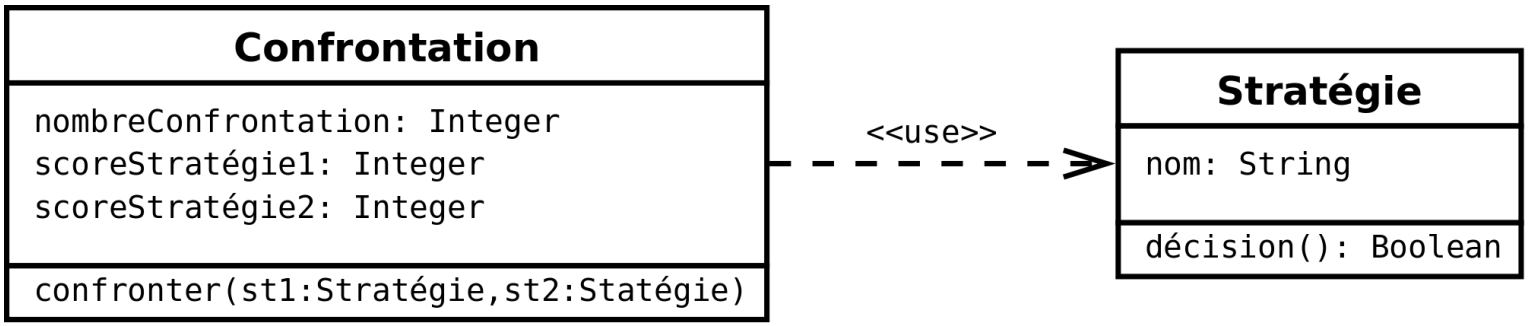
Auto-association sur classe-association





p1 est le supérieur de *p2* dans la société *s1*

Relation de dépendance

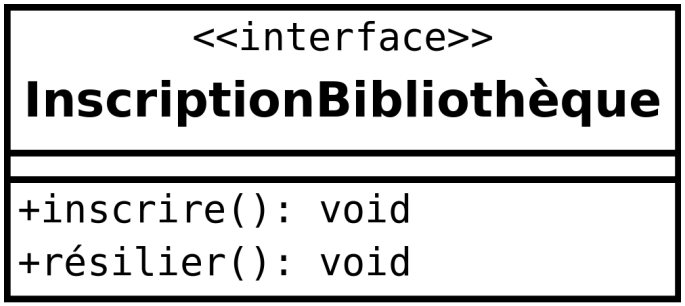


Dépendance

Relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle

- Elle indique que la modification de la cible peut impliquer une modification de la source
- Elle est représentée par un trait discontinu orienté
- Elle est souvent stéréotypée (≪`use`≫, ≪`instanceof`≫, ≪`include`≫, ≪`extend`≫...)

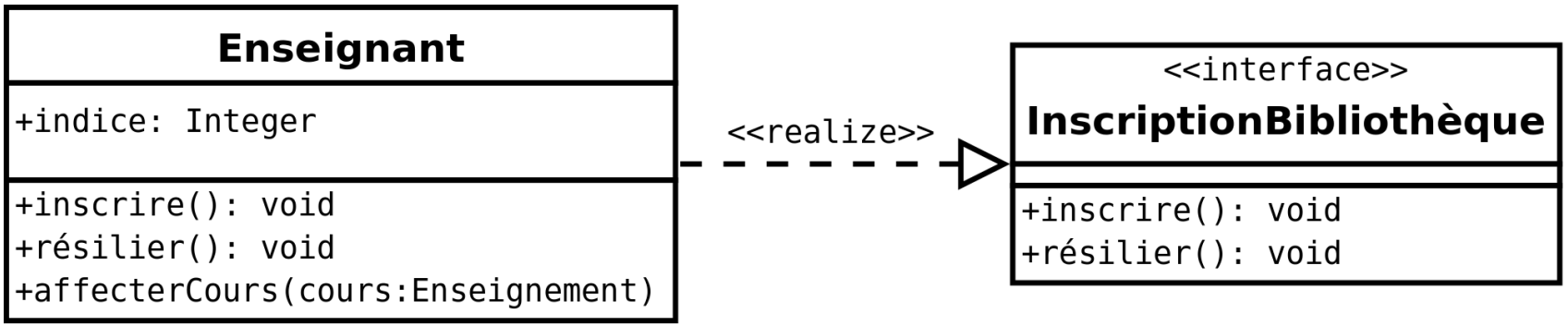
Interface



- Ce classeur permet de ne définir que des éléments d'interface
- Regroupe un ensemble de propriétés et d'opérations assurant un service cohérent
- Toutes les opérations sont abstraites
- Tous les attributs et les opérations sont publiques
- Représentation : comme une classe abstraite sans {`abstract`} mais avec un stéréotype «`interface`»

Relation de réalisation

- Au moins un élément d'implémentation doit être associé à chaque interface

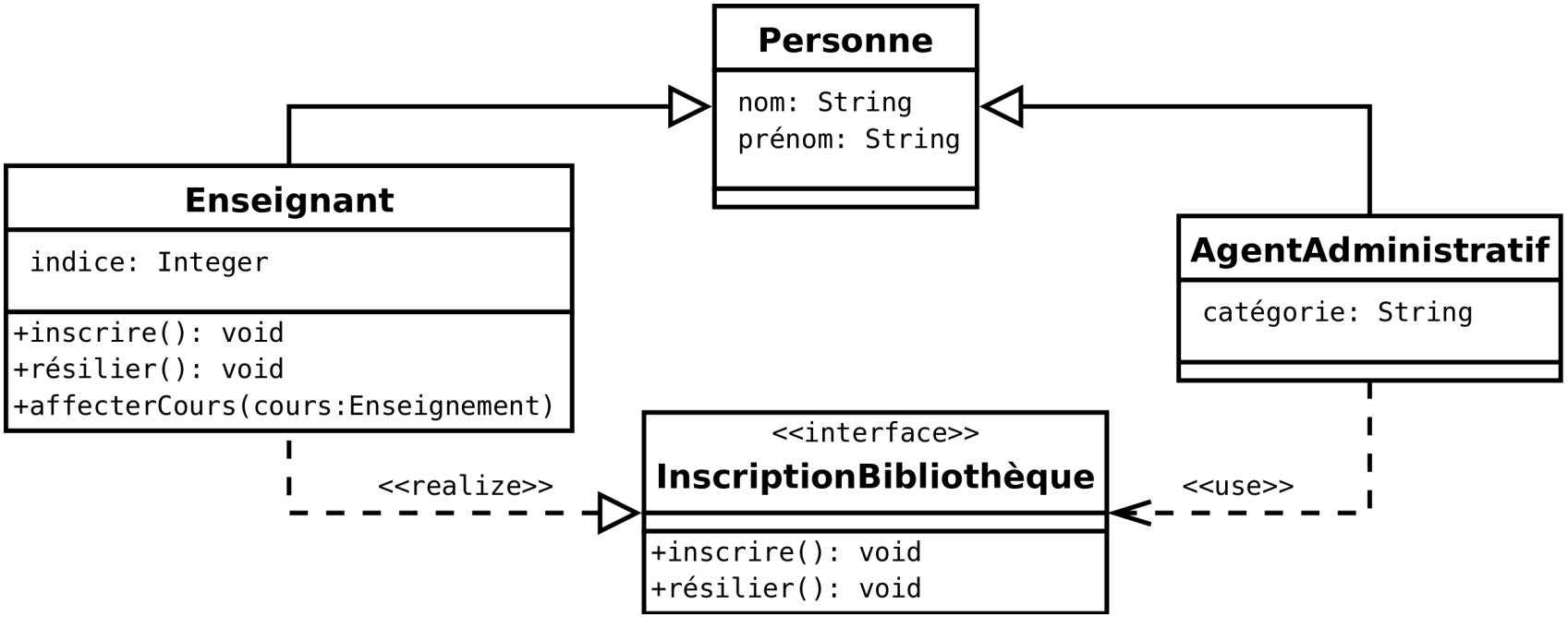


Relation de réalisation

Permet de mettre en relation un élément de spécification avec son implémentation

- Une classe peut réaliser plusieurs interfaces
- Une interface peut être réalisée par plusieurs classes

Mise en œuvre d'une interface



- Classe dépendante d'une interface (interface requise)
  → dépendance stéréotypée «use»
- Classe réalisant une interface (interface fournie)
  → relation de réalisation

Élaboration d'un diagramme de classes

Trouver les classes du domaine étudié en collaboration avec un expert du domaine (concepts ou substantifs du domaine)

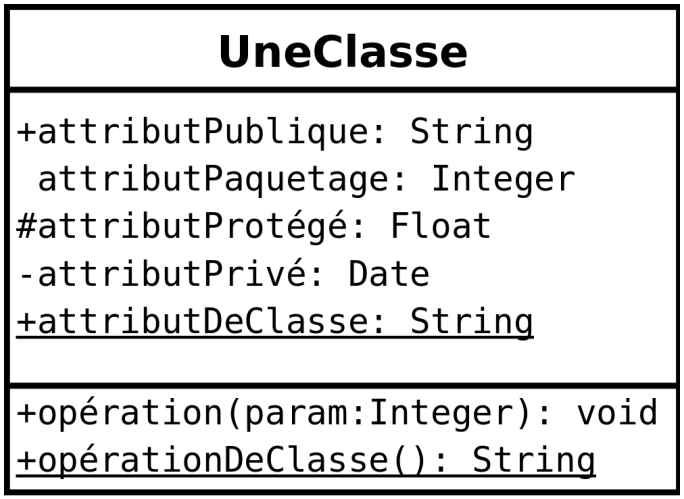
Trouver les associations entre classes : verbes, ou constructions verbales, mettant en relation plusieurs classes (≪*est composé de*≫, ≪*pilote*≫, ≪*travaille pour*≫...)

Attention, se méfier de certains attributs qui sont en réalité des relations entre classes

Trouver les attributs des classes : substantifs, ou groupes nominaux (≪*la masse d'une voiture*≫, ≪*le montant d'une transaction*≫...) ; on peut ajouter des attributs à toutes les étapes du cycle de vie d'un projet (implémentation comprise)

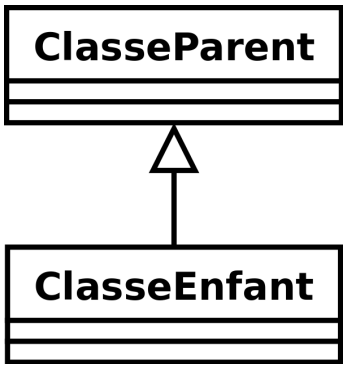
Organiser et simplifier le modèle en éliminant les classes redondantes et en utilisant l'héritage

Classe avec attributs et opérations




```
public class UneClasse {
  public String attributPublique ;
  long attributPaquetage ;
  protected double attributProtégé ;
  private Date attributPrivé ;
  public static String attributDeClasse ;
  public void opération(long param) {
    ...
  }
  public static String opérationDeClasse() {
    ...
  }
}
```

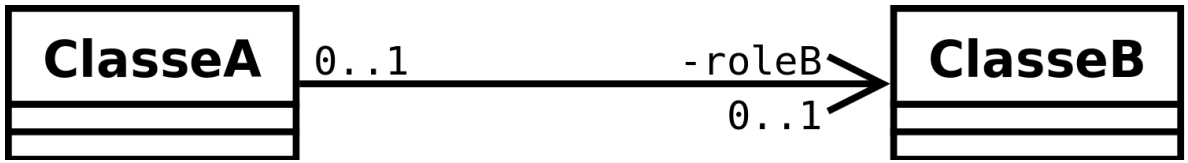
Héritage simple




```
public class ClasseParent {
   ...
}

public class ClasseEnfant extends ClasseParent {
   ...
}
```

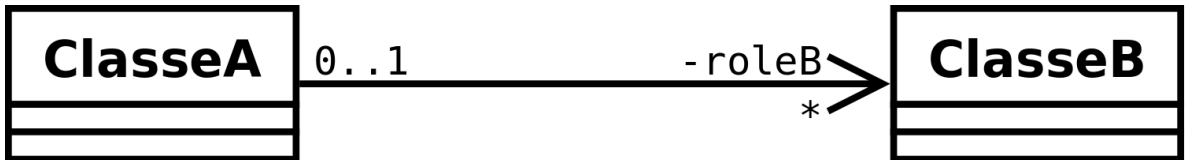
Association unidirectionnelle 1 vers 1




```
public class ClasseA {
  private ClasseB roleB ;
  ...
}
```

```
public class ClasseB {
 ... // ClasseB ne connaît pas l'existence de ClasseA
}
```

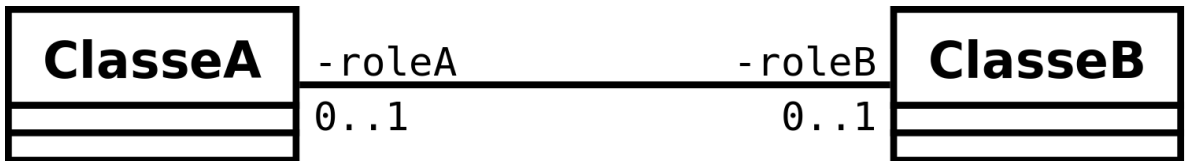
Association unidirectionnelle 1 vers plusieurs




```
public class ClasseA {
  private Set<ClasseB> roleB
    = new HashSet<ClasseB>() ;
  ...
}
```

```
public class ClasseB {
 ... // ClasseB ne connaît pas l'existence de ClasseA
}
```

Association bidirectionnelle 1 vers 1




```
public class ClasseA {
  private ClasseB roleB ;
  ...
}
```

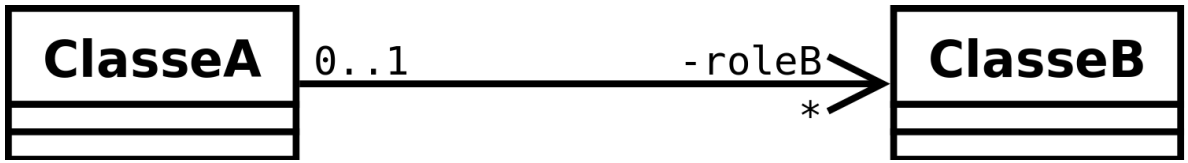
```
public class ClasseB {
  private ClasseA roleA ;
  ...
}
```

Association bidirectionnelle 1 vers 1 (suite)

```
public class ClasseA {
  private ClasseB roleB ;
  public void lierRoleB( ClasseB b ) {
    if( b != null && roleB != b){
      this.libérerRoleB() ;
      roleB = b ;
      roleB.lierRoleA( this ) ;
    }
  }
  public void libérerRoleB() {
    if (roleB != null) {
      ClasseB b = roleB ;
      roleB = null ;
      b.libérerRoleA() ;
    }
  }
}
```

```
public class ClasseB {
  private ClasseA roleA ;
  public void lierRoleA( ClasseA a ) {
    if( a != null && roleA != a){
      this.libérerRoleA() ;
      roleA = a ;
      roleA.lierRoleB( this ) ;
    }
  }
  public void libérerRoleA() {
    if (roleA != null) {
      ClasseA a = roleA ;
      roleA = null ;
      a.libérerRoleB() ;
    }
  }
}
```

Association bidirectionnelle 1 vers plusieurs




```
public class ClasseA {
  private Set<ClasseB> roleB
    = new HashSet<ClasseB>() ;
  ...
}
```

```
public class ClasseB {
  private ClasseA roleA ;
  ...
}
```