

Diagramme de classes

Diagramme d'objets

Yvan Maillot

1 Présentation

Le diagramme de classes est peut-être le diagramme le plus important dans le cadre de la conception orientée objet. Il définit la nature des objets d'un programme et leur lien.









Le diagramme de classes

- C'est un diagramme structurel fondamental pour le développement logiciel
 - Il définit les objets d'un programme.
 - Il définit les liens entre les objets.
- Parce qu'il exploite particulièrement le caractère « optionnel » d'UML, il est utilisé aussi bien :
 - depuis l'analyse du logiciel
 - jusqu'à sa conception détaillée,
- Il est donc très bien adapté à un développement incrémental.

2 Éléments de base d'un diagramme de classes

UML définit les éléments constitutifs d'un diagramme de classes. Ce sont les classes, les interfaces et les relations, discutées en détail dans la suite.


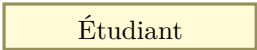
Les éléments d'un diagramme de classes :

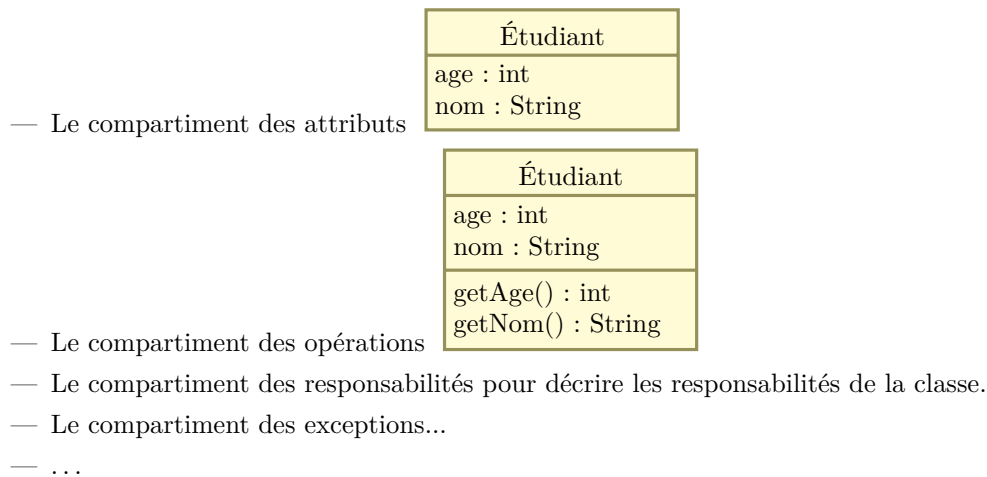
- Classes 
- Interfaces 
- Relations
 - Association 
 - Agrégation 
 - Composition 
 - Dépendance 
 - Réalisation 
 - Généralisation 

2.1 Les classes

La syntaxe des classes exploite bien le caractère optionnel d'UML.

La syntaxe d'une classe

- Elle se représente par un rectangle 
- avec éventuellement :
 - son nom 
Par convention l'initiale du nom est en majuscule
Le nom en italique indique que la classe est abstraite.
 - Des compartiments éventuels

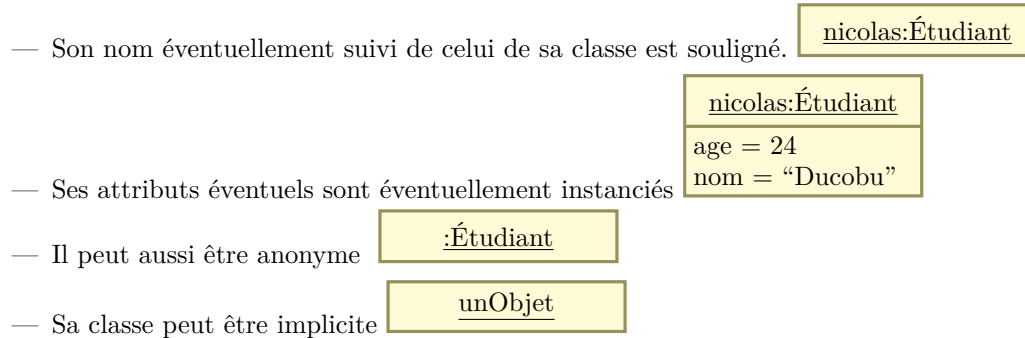


2.1.1 Instances de classe

Une classe définit les instances qu'elle peut engendrer. Une instance de classe s'appelle un objet et peut se dessiner en UML.

Une instance de classe est un objet

Il est représenté par un rectangle



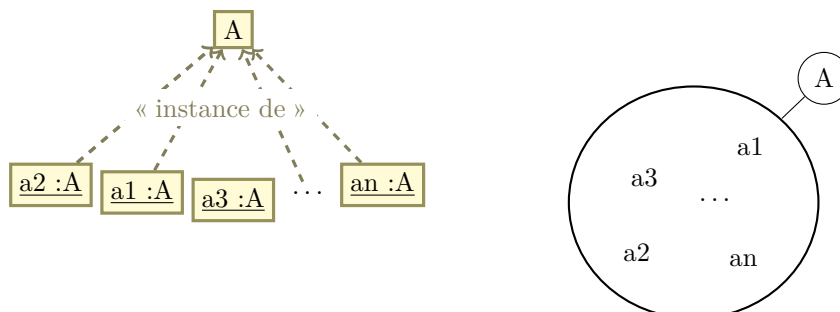
La définition d'une classe est analogue à la définition en compréhension d'un ensemble.

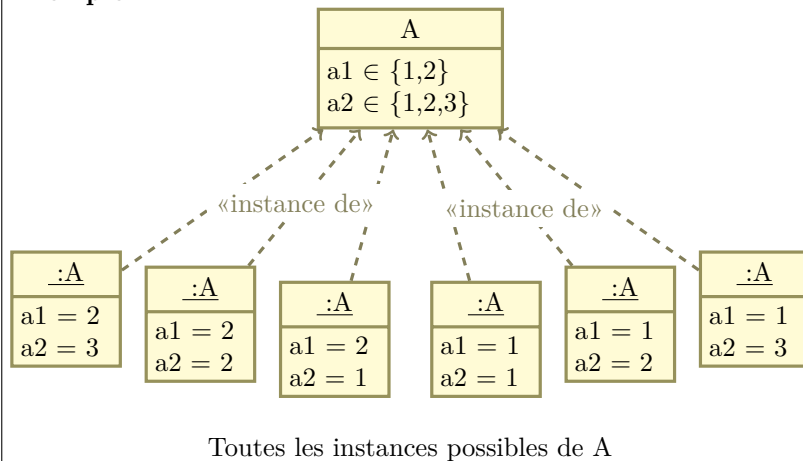
Analogie ensembliste

Une classe définit un ensemble d'objets

de façon un peu analogue à la définition en compréhension d'un ensemble.

$A = \{ :A \mid :A \text{ respecte des propriétés} \}$



Exemple**2.1.2 Les opérations d'une classe**

Une opération est une caractéristique comportementale d'une classe qui spécifie son nom, son type, ses paramètres et des contraintes pour invoquer un comportement associé.

Une opération peut être d'instance ou de classe (statique).

- Une opération d'instance exprime (avec plus ou moins de détails) le comportement des instances d'une classe.
- Une opération de classe (dont le nom est souligné) exprime (avec plus ou moins de détails) une « routine » (fonction ou procédure).

Une opération :

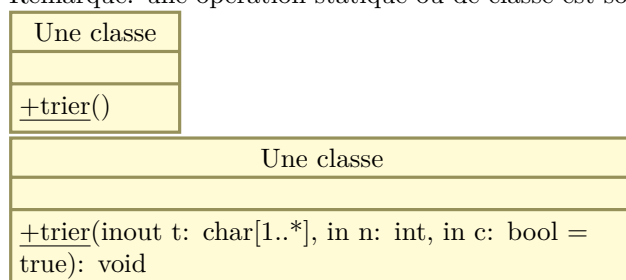
- précédée de « create » indique qu'elle crée une instance de cette classe,
 - précédée de « destroy » indique qu'elle se charge de détruire l'instance.
- La syntaxe permet de spécifier tout ou partie d'une opération.

Syntaxe des opérations d'une classe

- opération ::= visibilité nom(paramètres) : type-retourné {propriétés}
 - visibilité ::= - # ~+ (privé, protégé, amical, public)
 - paramètres ::= direction nom_paramètre : type[multiplicité] = valeur_par_défaut {propriétés}
 - direction : in, inout, out
 - nom_paramètre : nom significatif
 - multiplicité : unicité ou [*] ou [2..6] ou ?
 - valeur_par_défaut : valeur
 - propriétés : langage naturel ou formel comme OCL
- nom : en général un verbe significatif
- type-retourné : void ou type primitif ou classificateur

Tout est optionnel

Remarque: une opération statique ou de classe est soulignée.



Il est possible de spécifier la programmation par contrat grâce au pré et post-conditions.

Contraintes sur les opérations

Condition a priori (pré-condition)

- Elle précise l'ensemble des états du système admis pour le bon déroulement de l'opération.
- Elle s'écrit en OCL ou en langage naturel, entre accolades après le prototype et précédée du mot clé precondition.

Condition a posteriori (post-condition)

- Elle définit l'état du système après l'opération.
- Elle s'écrit entre accolades après le prototype et précédée du mot clé postcondition.

Une classe
<code>+trier(inout t : char[1..*], in n : int) : void</code> « pre-condition » $\{n > 0\}$ « post-condition » $\{i < j \Rightarrow t[i] \leq t[j]\}$

Condition de corps

C'est une contrainte s'appliquant sur la valeur retournée.

2.1.3 Les attributs d'une classe

Un attribut peut être de classe ou d'instance.

Description

- Les attributs de classe sont partagés par toutes les instances de cette classe et même par d'autres selon leur visibilité.
- Les attributs d'instance sont propres à une instance. Ensemble, ils définissent son état.

Syntaxe

Les attributs de classe sont soulignés contrairement aux attributs d'instance.

Point
<u>origineX</u>
<u>origineY</u>
x
y

Un attribut peut être déclaré avec un niveau de détail allant du plus simple ou plus précis.

Niveaux de détail d'un attribut

- Absent

Point
- Simplement nommé

Point
x
y
- Typé

Point
x : double
y : double
- Multiplicité définie

Polygone
p : Point[3..*]

- Contraints ou avec des propriétés
{ordonné}, {unique}, {lecture seule}, ...

Polygone
p : Point[3..*] {ordonné, unique}

- Initialisé

Point
x : double = 0 y : double = 0

- Calculé

Point
x : double y : double /r : double /a : double

- Visibilité définie

+ public
protégé
- privé
~ paquetage

Exemple
- a : int = 0 # b : double ~ c : boolean + d : String

- Contraintes exprimées en langage naturel ou dans une grammaire formelle comme OCL
 - {les portes doivent être fermées}
 - {hauteur = largeur}

Exemple de traduction en Java

Point
- x : double - y : double - <u>origineX</u> : double - <u>origineY</u> : double
« create » + Point(x : double, y : double) + getX() : double + getY() : double

Voici un exemple de traduction d'une classe UML en Java.

```

public class Point {
    private static double origineX ;
    private static double origineY ;
    private double x ;
    private double y ;
    public Point(double x, double y) {
        this.x = x ;
        this.y = y ;
    }
    public double getX() {
        return x ;
    }
    public double getY() {
        return y ;
    }
}

```

Les codes des méthodes ne sont pas spécifiés dans la classe UML parce qu'ils sont évidents ici. Mais ils auraient pu être expliqués avec plus ou moins de détails dans des annotations.

Attributs en ligne ou par relation

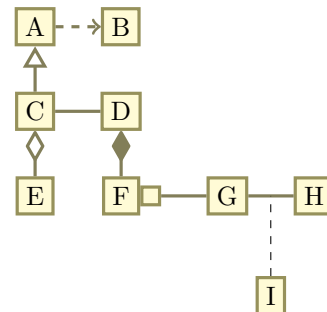
- Les attributs définis dans leur compartiment sont dits *en ligne*.
- Une autre façon de les définir est *par relation*.
Ce sont les attributs définis au travers des relations d'association, expliquées dans la suite.

2.2 Les relations

UML définit quatre types de relations entre classes : l'association, la dépendance, la généralisation et la réalisation.

Les relations définies dans un diagramme de classes

- Association
 - Agrégation
 - Composition
 - Association qualifié
 - Classe d'association
- Dépendance
- Généralisation



2.2.1 L'association

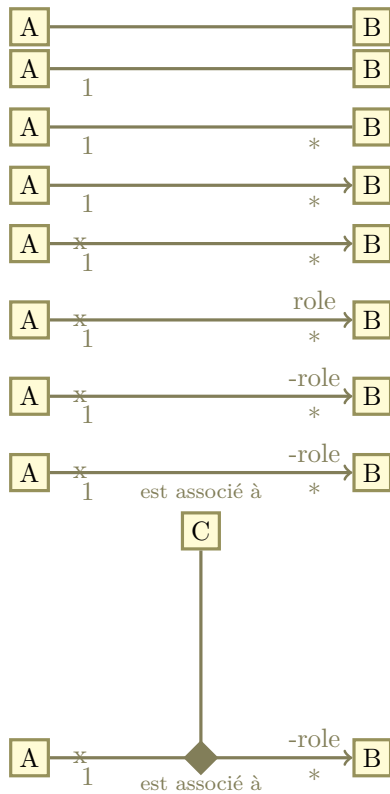
Une *association* est un classificateur. Elle définit donc des instances. Une instance d'association est un *lien*. Les associations expriment la nature des relations entre classes. Plus précisément, une association entre classes (disons A et B) exprime les couples d'instance de A et d'instance de B possibles.

Définition d'une association

- C'est une relation structurelle et/ou sémantique
 - Exprime comment les objets sont liés entre eux
 - Exprime des liens logiques
- Elle définit l'existence de liens entre des instances
 - Une instance d'association s'appelle un lien
 - Un lien est un tuple avec une valeur à chaque extrémité où chaque valeur est une instance
- Elle est
 - bidirectionnelle
 - éventuellement réflexive
 - le plus souvent binaire
 - éventuellement n-aire (d'arité n)

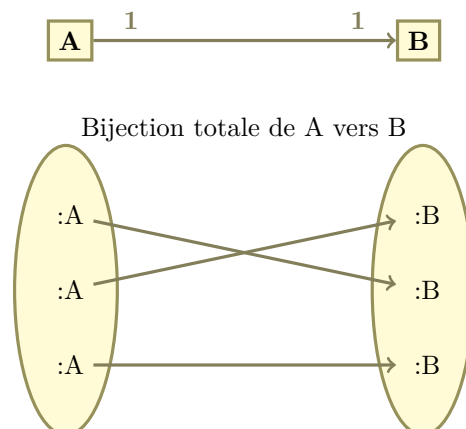
Propriétés d'une association

- Multiplicité
 - non spécifiée, 1, *, 2..*, 1..5, ...
- Navigabilité
 - non spécifiée, bidirectionnelle, unidirectionnelle, ...
- Rôles
- Nommage (verbe ou forme passive)
- Contraintes
- Arité



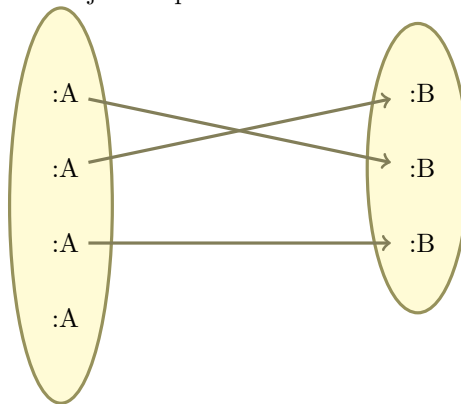
Puisqu'une classe s'apparente à la définition d'un ensemble (voir en page 2), une association est analogue à la définition d'une relation entre ensembles.

L'association 1—1

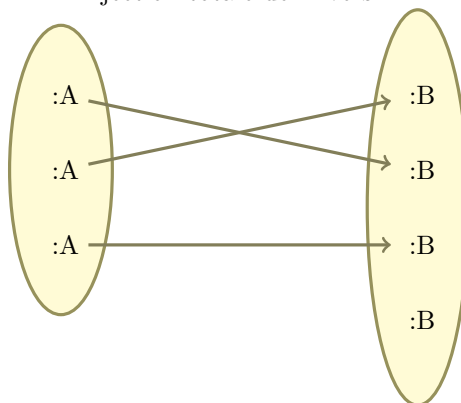


L'association 1—0..1

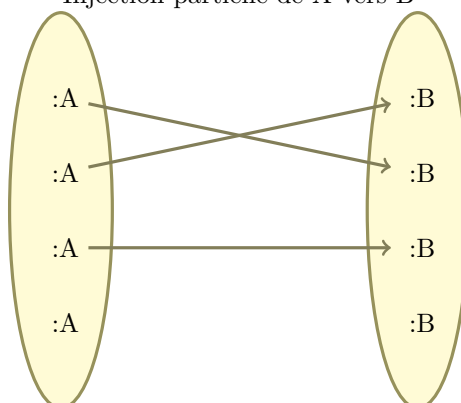
Bijection partielle de A vers B

**L'association 0..1—1**

Injection totale de A vers B

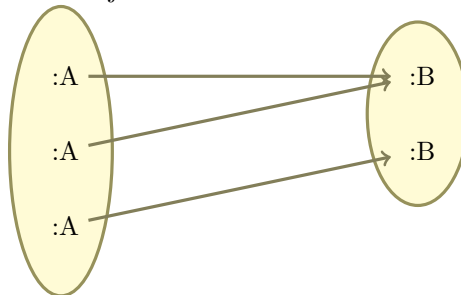
**L'association 0..1—0..1**

Injection partielle de A vers B

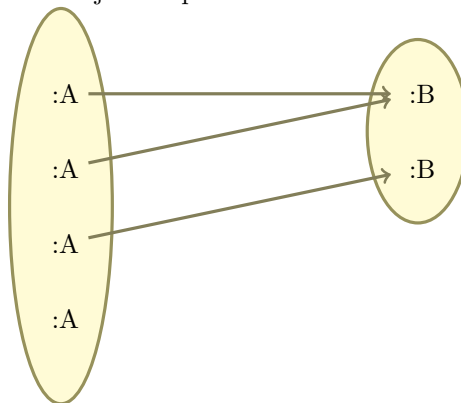


L'association 1..*—1

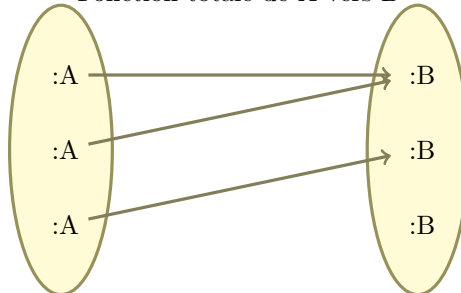
Surjection totale de A vers B

**L'association 1..*—0..1**

Surjection partielle de A vers B

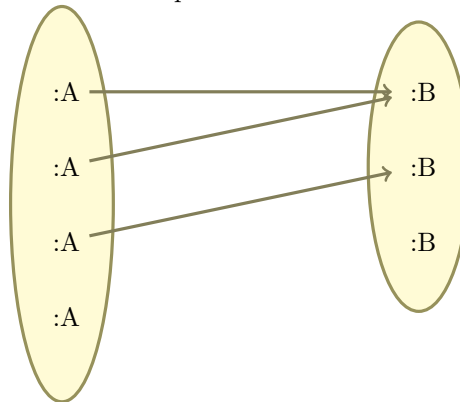
**L'association *—1**

Fonction totale de A vers B

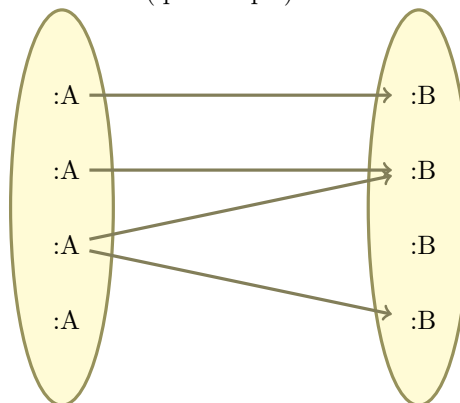


L'association *—0..1

Fonction partielle de A vers B

**L'association *—***

Relation (quelconque) de A vers B

**Agrégation et composition**

Agrégation



Aucune différence structurelle avec une association. La différence est sémantique. L'agrégation exprime une notion de hiérarchie. ici A est supérieur à B.

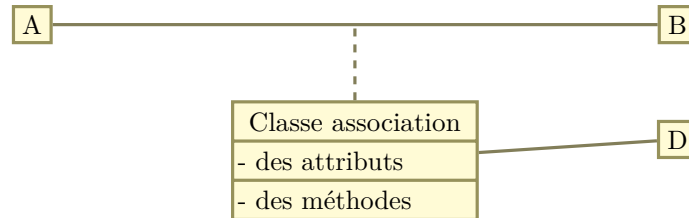
Remarque : L'agrégation est peu utilisée parce qu'elle est sujette à confusion. Certaines équipes l'utilisent pour exprimer des particularités de conception comme par exemple pour souligner l'utilisation de pointeurs.

Composition



Exprime une véritable différence structurelle avec une association. La durée de vie des objets B dépend de celle de l'objet A auxquels ils sont liés.

Classe association

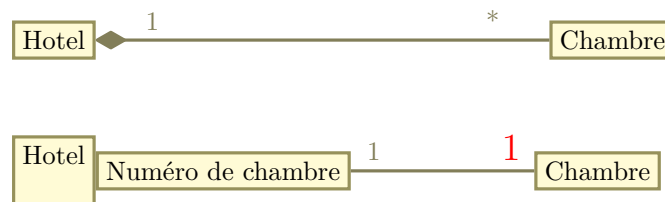


Une association peut être une classe à part entière

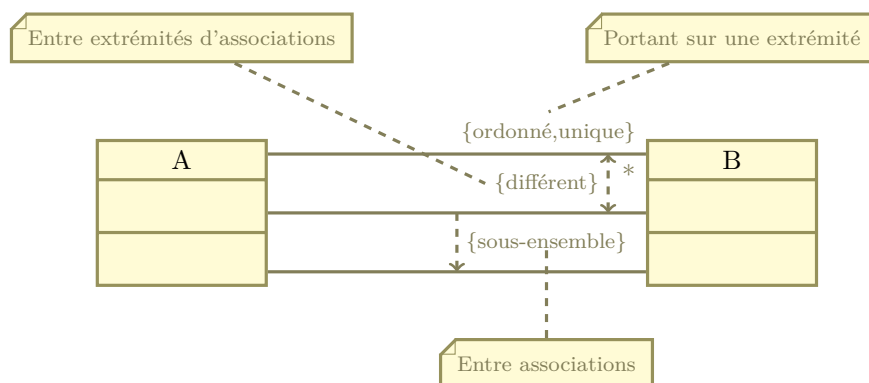
- Elle peut avoir des attributs et des méthodes.
- Elle peut être reliée à d'autres classes du diagramme.
- Chaque instance de cette classe définit un couple (A, B).

Association qualifiée

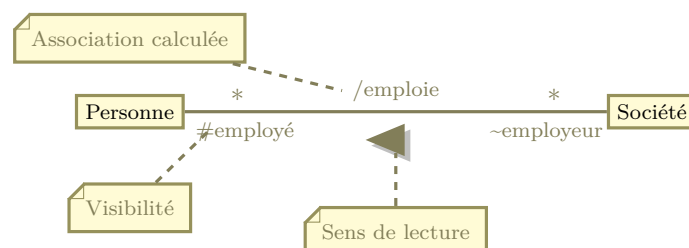
- Une association qualifiée permet de simplifier certaines association de multiplicité maximale supérieure à 1.
- Elle définit une indexation.



Contraintes



Autres décorations



Quelques exemples de traduction en C++



- La multiplicité « 1 » indique qu'à une instance de A est associée une et seule instance de B
- La multiplicité « 0..1 » indique qu'à une instance B est associée zéro ou une instance de A
- La navigabilité indique qu'une instance de A accède à son instance de B associée tandis que l'inverse n'est pas vrai.
- Le rôle « -b » indique que A a un attribut privé de type B nommé `b`

```

class B {}; // Une simple classe B
class A { // Une classe A avec un attribut privé de type B
private :
    B b; // L'accès à B depuis A et pas l'inverse.
}; // Il ne peut pas y avoir de A sans B.
  
```



La différence réside dans la multiplicité 0..1 qui devient 1. Cela signifie qu'à un A correspond un et un seul B et vice-versa. Un B ne peut pas être construit sans A.

```

class B { /* Il faut par exemple ne pas autoriser la construction publique d'un B. */
private :
    B(){}; // Le constructeur est dans la partie privée.
friend class A; /* Ainsi, seules les instances de A peuvent construire des instances de B */
};
class A {
private :
    B b;
};
  
```



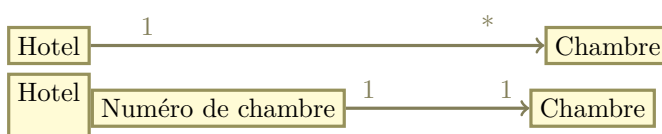
La double navigabilité fait qu'un A doit accéder à un et un seul B et inversement. Difficile de se passer de pointeurs.

```

class B { /* Seul A peut construire B */
friend class A;
private :
    A *a;
    /* Le constructeur est privé */
    B(A* p_a) : a(p_a) {}
};
  
```

```

class A {
private :
    B *b;
public :
    A(){ b = new B(this); }
    /* Ne pas oublier de détruire b */
    ~A() {delete b;}
};
  
```



```

class Chambre {};
class Hotel {
public :
    Chambre* chambre ;
    Hotel(int n) {
        chambre = new Chambre[n] ;
    }
    ~Hotel() {
        delete[] chambre ;
    }
};

```

2.2.2 La relation de dépendance

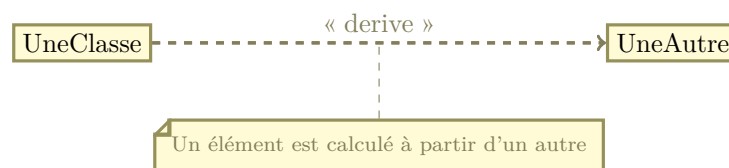
Dépendance

La plus faible Relation

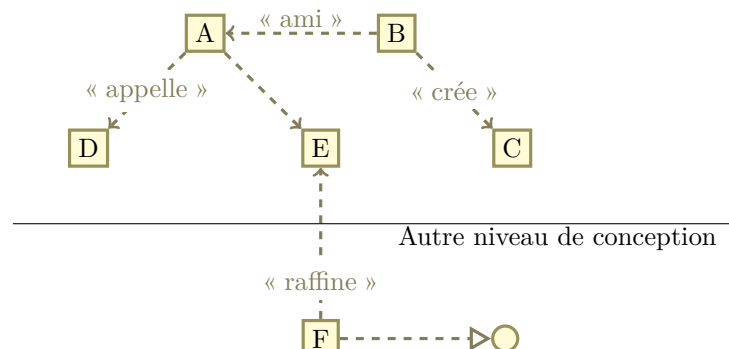
- Relation sémantique
- Relation unidirectionnelle (source vers cible) Changement cible \Rightarrow Changement source

UML en définit 4 types

- Abstraction
 - « dérive »
 - « raffine »
 - « réalise »
- Liaison
 - « lie »
- Permission
 - « ami »
- Utilisation
 - « utilise »
 - « crée »
 - « appelle »



Dépendance : exemple de représentation graphique



Dépendance entre objet et classe



3 Quelques exercices

3.1 Enseignement

Dessinez une classe qui exprime le fait qu'un enseignant a un nom, un prénom, une date de naissance et un âge.

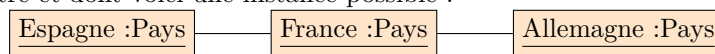
Dessinez une classe qui exprime le fait qu'un cours a un nom, un nombre d'heures de cours et un nombre d'heures de TD.

Dessinez un diagramme de classes qui exprime que plusieurs enseignants peuvent intervenir dans un même cours mais seul un enseignant en est responsable.

Écrivez un code java correspondant à ce diagramme.

3.2 Pays frontaliers

Réalisez un diagramme de classes dont les instances permettent de préciser quel pays est frontalier de quel autre et dont voici une instance possible :



3.3 Polygones

Réalisez un diagramme de classes qui exprime les relations entre un polygone et ses points. Un point a deux coordonnées x et y.

Argumentez vos choix de multiplicité.

Réalisez un diagramme d'objets pour deux triangles qui ont un côté en commun.

3.4 Le dîner des philosophes

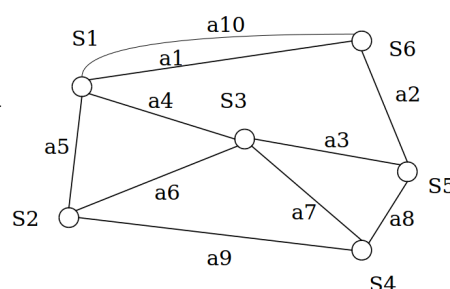
Des philosophes autour d'une table ronde partagent un repas. La particularité de la situation est qu'ils partagent aussi leur baguette. Chaque philosophe partage sa baguette de gauche (resp. droite) avec son collègue gauche (resp. droit).

Lorsqu'ils ne mangent pas, ils pensent. Pour manger, un philosophe a besoin de ses deux baguettes. Cette allégorie est exploitée pour simuler les problèmes de partage de ressources entre plusieurs processus concurrents. L'idée est de trouver un protocole d'actions qui permette aux philosophes de manger à leur faim et passer suffisamment de temps à penser sans jamais être bloqués.

Dessinez un diagramme des classes d'un programme permettant la simulation de différents protocoles.

3.5 Graphe

Dessinez un diagramme de classes qui exprime la structure d'un graphe



3.5.1 La relation de généralisation/spécialisation

C'est une relation qui permet de faire de la classification. Elle est mise en œuvre dans les langages orientés objet par l'héritage. Mais elle est plus générale.

Généralisation/Spécialisation

C'est une relation qui permet de définir une classification.

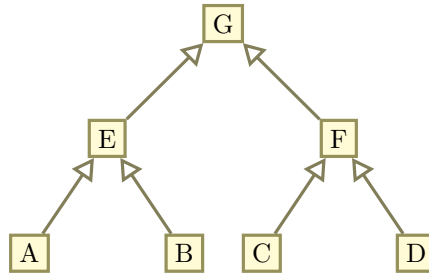
Elle favorise la conception par généralisation/spécification.

Généraliser (conception ou analyse ascendante)

Factoriser les propriétés communes de certaines classes.

Spécialiser (conception ou analyse descendante)

Particulariser une classe (ou plusieurs) à partir d'une autre.



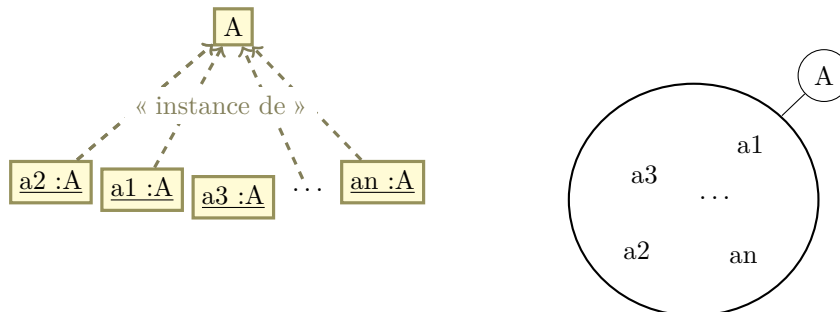
Deux techniques opposées mais utilisées conjointement.

Analogie ensembliste

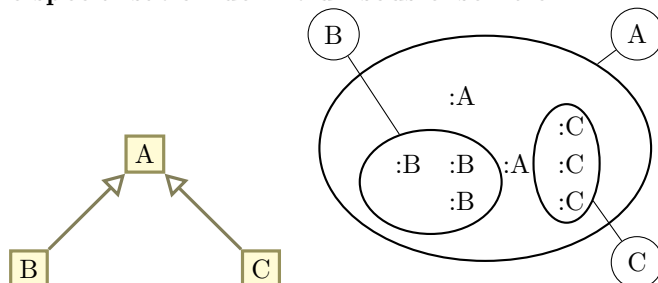
Une classe définit un ensemble d'objets

de façon un peu analogue à la définition en compréhension d'un ensemble.

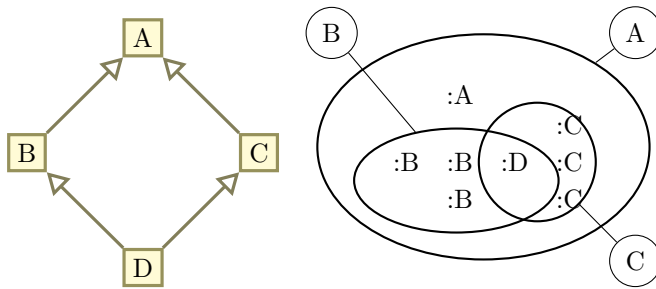
$A = \{ :A \mid :A \text{ respecte des propriétés} \}$



Une spécialisation définit un sous-ensemble



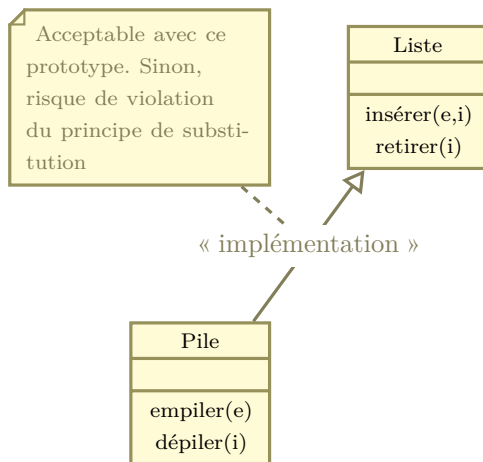
Une généralisation multiple exprime une intersection



Spécialisation pour la construction

L'héritage est le mécanisme offert par les langages orienté objet pour mettre en œuvre la généralisation/spécialisation.

- Il s'utilise sans précaution pour classifier
- Il peut s'utiliser pour construire, mais avec précaution.



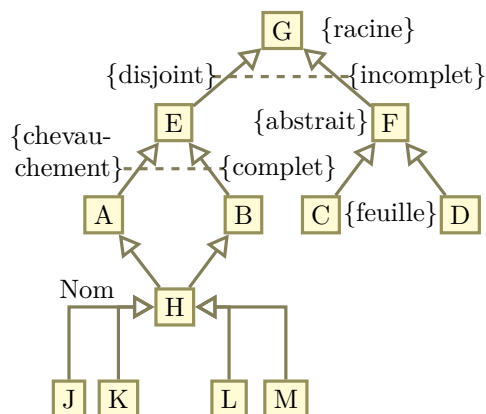
Contraintes et propriétés

Contraintes de généralisation

- Disjointe / Chevauchement
- Incomplet / Complet

Contraintes de classe

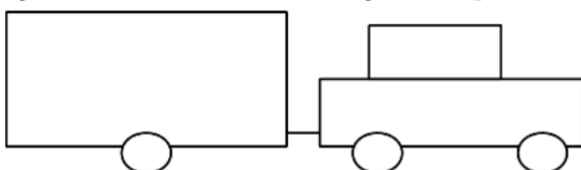
- Racine
- Feuille
- Abstrait



4 Quelques exercices

4.1 Figures géométriques

On aimerait écrire une application de conception de figures à bases de formes géométriques simples comme des lignes, des cercles et des rectangles. Elle permettra, par exemple, de dessiner la figure suivante :



Proposez un diagramme de classes où

- un cercle, un rectangle, une ligne est une figure simple et
- une figure composée est composée de figures (simples ou composées).

On veut de surcroît pouvoir ajouter et supprimer des figures à une figure composée.

Enfin, il doit pouvoir être facile de dessiner et de translater une figure.

Dessinez un diagramme d'objets pour l'exemple précédent.

4.2 Expressions arithmétiques

Dessinez un diagramme de classes pour modéliser des expressions avec des variables et des constantes afin de les évaluer.

Dessiner le diagramme d'objets pour l'expression $\frac{2x+3}{-x(y-2)}$

Écrire le code java correspondant, instancier le diagramme d'objet précédent pour lancer son évaluation.

4.3 Cri d'animaux

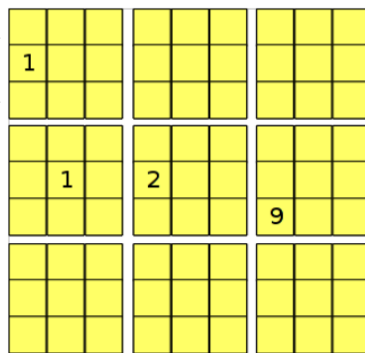
Une application simple a été réalisée pour apprendre aux enfants à associer la silhouette d'un animal à son cri et son nom. Seuls, le chat et le chien sont implantés dans cette première version. Devant le succès du produit, on souhaite le faire évoluer afin d'ajouter d'autres animaux et rendre aisé ce genre d'ajouts à l'avenir. Par ailleurs, on aimerait éviter de réécrire les classes existantes.

Proposer une solution à l'aide d'un diagramme de classes.

4.4 Sudoku

On veut réaliser une application capable d'assister un concepteur de grilles de sudoku.

Une case peut être modifiée. On doit notamment pouvoir aisément connaître quelles cases sont affectées par le changement d'une autre case. La modification d'une case affecte toutes les cases de la même ligne, de la même colonne et du petit carré dans lequel il se trouve.



4.5 Emplois

Une entreprise emploie bien souvent plusieurs employés. Mais un employé peut aussi avoir un poste dans plusieurs entreprises. En revanche, un employé occupe un et un seul poste au sein d'une entreprise. Le salaire d'un employé est lié aux postes qu'il occupe. Un employé peut être responsable d'autres, mais c'est en réalité le poste qu'il occupe qui permet de connaître quels employés sont sous ses ordres, au travers des postes qu'ils occupent.