# NFP121 — Programmation Avancée

—

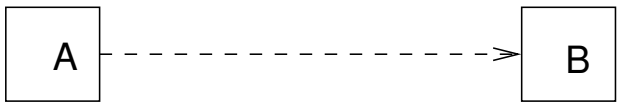
# Relations entre classes

Xavier Crégut
<Prénom.Nom@enseeiht.fr>

ENSEEIHT
Télécommunications & Réseaux

# Relations entre classes

On dit qu'il y a relation de dépendance entre une classe A et une classe B si la classe A fait référence à la classe B dans son texte.




Cette relation peut être momentanée si B apparaît comme

- un paramètre d'une méthode ;
- une variable locale.

Cette relation est structurelle si elle dure (plus longtemps qu'un appel de méthode). C'est généralement le cas quand B est un attribut.

**En UML**, on fait apparaître une relation entre les classes qui peut être graduée :

- association
- agrégation
- composition

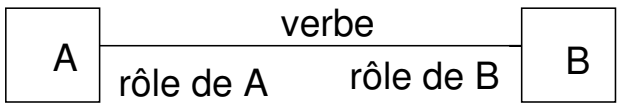
**En Java**, nous n'avons pas les mêmes nuances :

- les objets sont manipulés par des poignées
- donc par référence, avec partage par défaut

# Relation entre classes

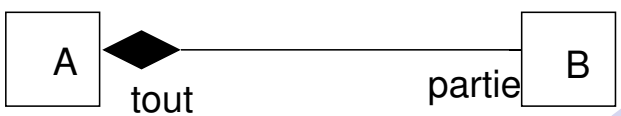
Une application est composée de plusieurs classes dont le couplage est caractérisé par des relations dites d'utilisation (ou de délégation) :

- **association :** couplage faible correspondant à une relation symétrique entre objets relativement indépendants (durées de vie non liées) ;

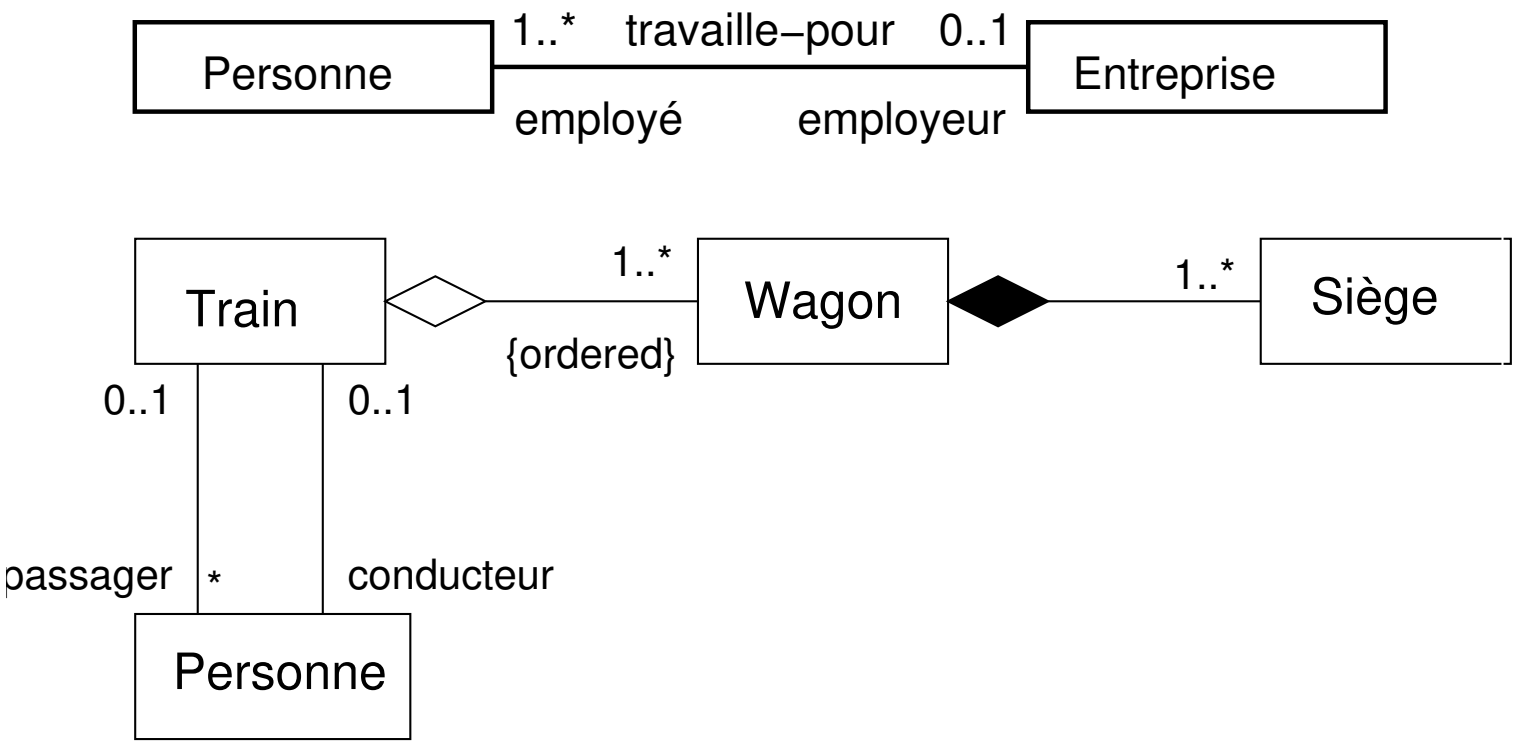



- **agrégation :** association non symétrique avec couplage plus fort
  - relation tout et parties
  - relation de subordination : propagation des opérations de A vers B
  - on ne peut pas parler de A sans parler de B


- **composition :** agrégation forte (par valeur) :
  - La durée de vie des objets « partie » est liée à celle du « tout »
  - Pas de partage possible.



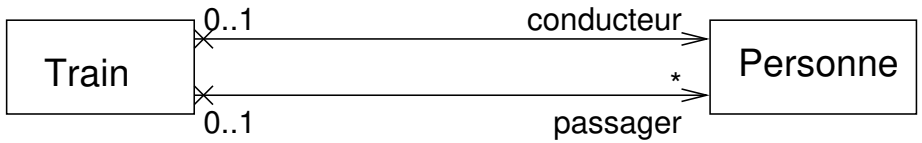
# Relations entre classes : exemples




**Exercice 1** Dessiner un diagramme de classes faisant apparaître un site web, des pages HTML et un « webmaster ».

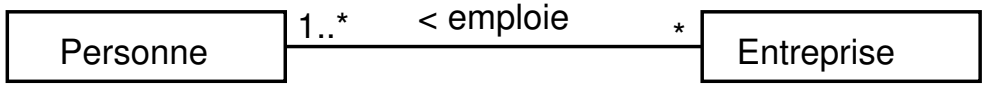
# Principales caractéristiques d'une relation d'utilisation

- Elle est bidirectionnelle :
  - à partir d'un objet d'une extrémité, on peut atteindre les objets de l'autre extrémité.
  - Il est possible de l'orienter et donc de supprimer un sens de navigation.

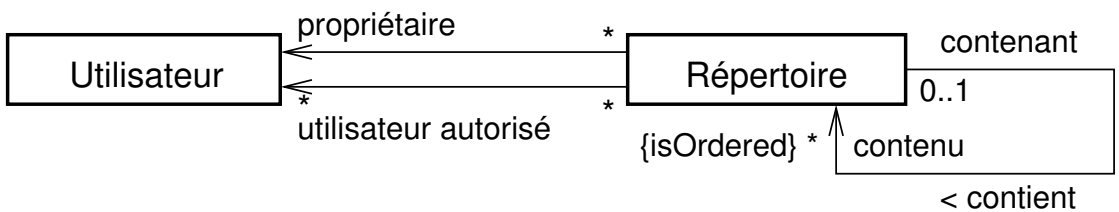



    - d'un objet train on peut obtenir le conducteur ou les passagers
    - d'une personne on ne peut pas retrouver le train (Train n'apparaît pas dans Personne)

- Il faut préciser sa signification :
  - en lui donnant un nom (verbe, au centre de la relation, avec sens de lecture)




  - en précisant le rôle que jouent les objets dans la relation (extrémité de la relation)

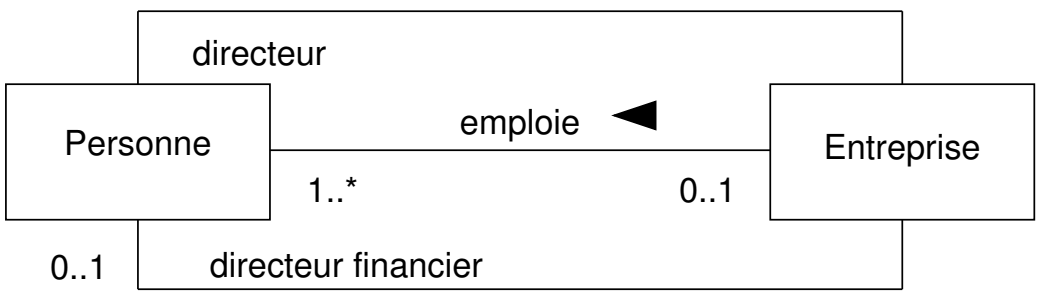



    - Les rôles peuvent avoir des droits d'accès (comme les attributs).

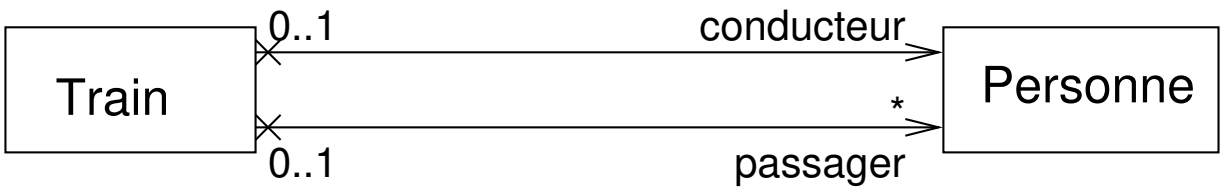
# Multiplicité (ou cardinalité)

**Objectif :** indiquer combien d'objets peuvent/doivent prendre part à la relation :

- forme générale : min..max
  - exemple : 1..4
  - le nombre d'objet est compris entre min et max inclus
  - la valeur de max peut être '*' : nombre quelconque (« illimité »)
- un entier (ex : 4) : le nombre exact d'objets, simplification d'écriture
- Caractère optionnel : `0..1`
- $*$ ou `0..*` : un nombre quelconque y compris 0
- `1..*` : un nombre quelconque mais au moins 1
- La multiplicité par défaut est 1



# Traduction en Java




```
1  public class Train {
2      private Personne conducteur;
3          // Pour une multiplicité de 1 ou 0..1 (null ?)
4
5      private Personne[] passagers;
6          // Au lieu d'un tableau, il est préférable
7          // d'utiliser une structure de données adaptée.
8      ...
```

- En UML, on ne met pas toujours la croix quand la flèche est à l'autre extrémité.
- En Java, pas de distinction entre association, agrégation et composition :
  - c'est toujours une poignée
  - C'est au programmeur de le gérer !
  - Plusieurs stratégies possibles pour la composition...

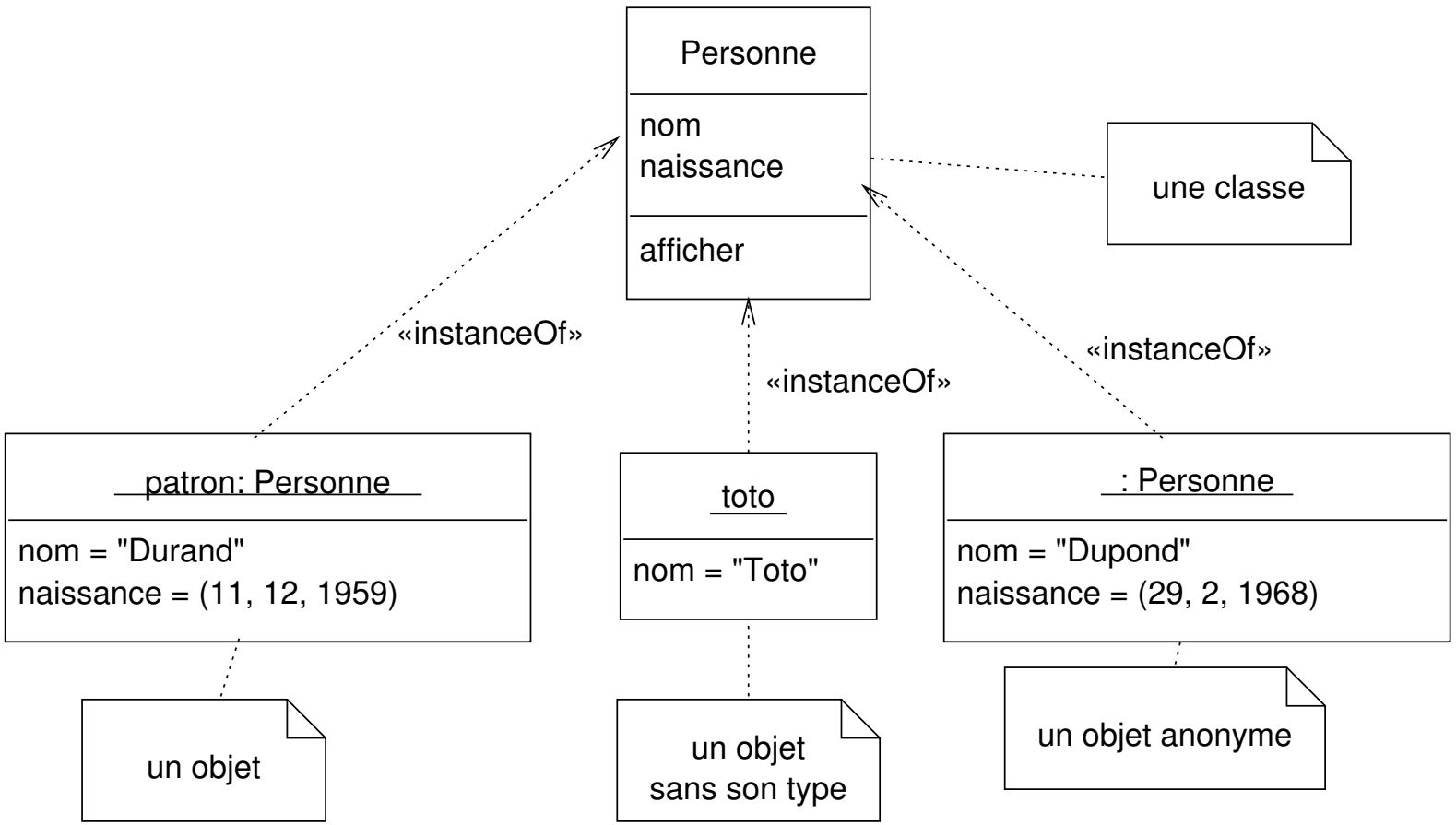
# Exercices

**Exercice 2** Indiquer quelles relations il serait possible de définir entre :

- Livre et Page
- Itinéraire et Gare
- Segment et Point

**Exercice 3** Que penser d'attributs de types primitifs ?
Et plus généralement de la rubrique « attribut » d'UML.

# Représentation des objets



# Diagramme d'objet

**Diagramme d'objet :** Une configuration possible respectant un diagramme de classe.

Exemple de diagramme de classe :

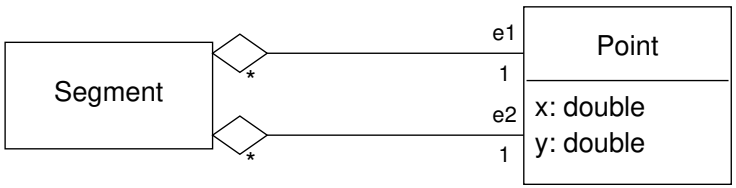
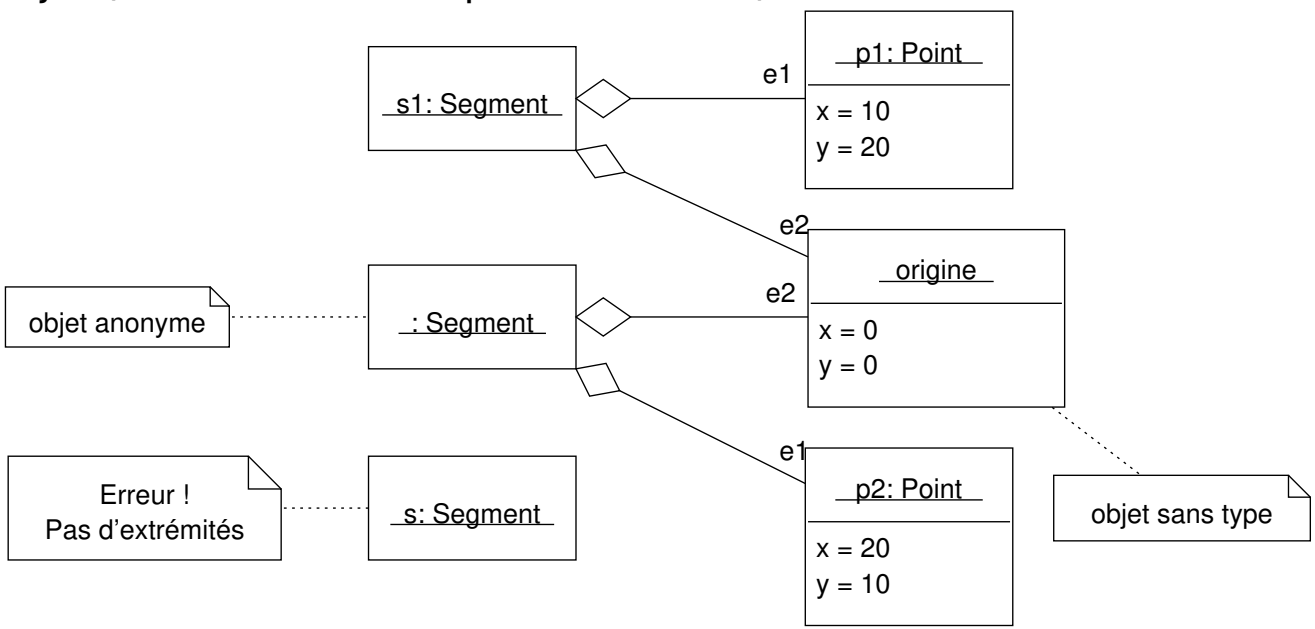



Diagramme d'objet (attention, s n'est pas conforme !) :



# Conformité

Un diagramme d'objet est conforme à un diagramme de classe ssi :

- tout objet du diagramme d'objet est instance d'une classe du diagramme de classe
  - chaque attribut de l'objet correspond à un attribut de la classe
  - la valeur des attributs de l'objet respecte le type de l'attribut dans la classe
- tout lien du diagramme d'objet est instance d'une relation du diagramme de classe
- le nombre de liens respecte les multiplicités exprimées sur le diagramme de classe
- la relation de composition est respectée :
  - un objet ne peut apparaître extrémité que d'un seul lien de composition !

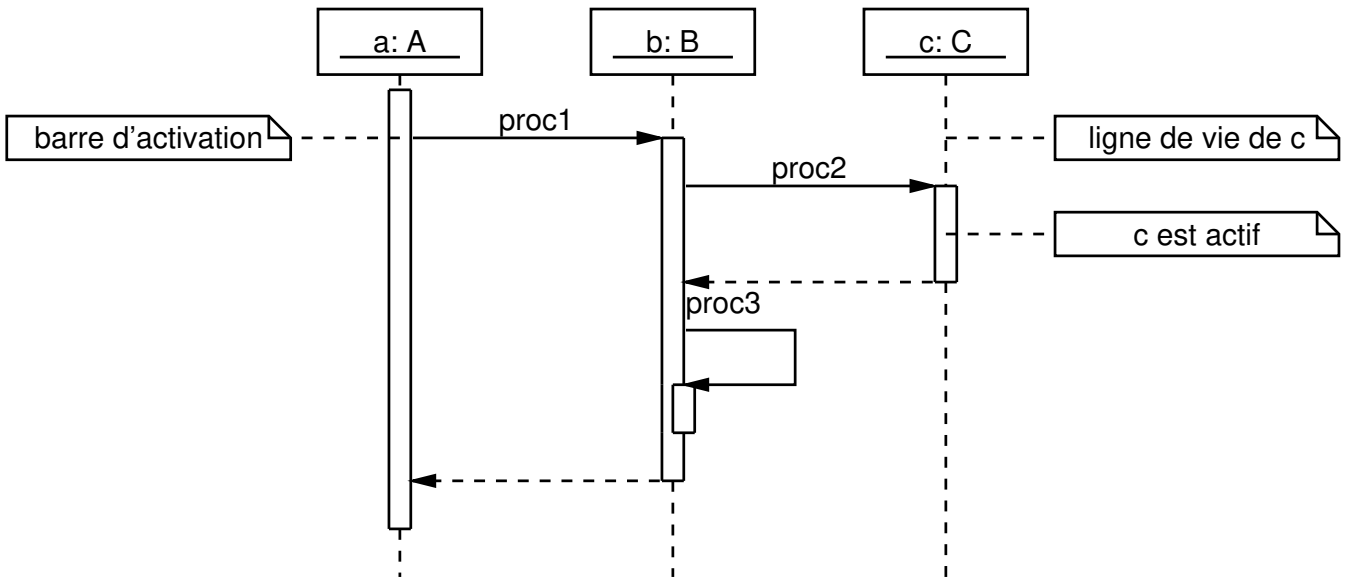
Quelques remarques sur le diagramme d'objet :

- Généralement, on ne fait pas apparaître les losanges (agrégation, composition)
- Les rôles sont souvent omis (si pas d'ambiguïté)

# Diagramme de séquence

**Objectif :** Décrire les interactions entre objets en privilégiant l'aspect temporel (chronologie des envois de messages, de haut en bas).

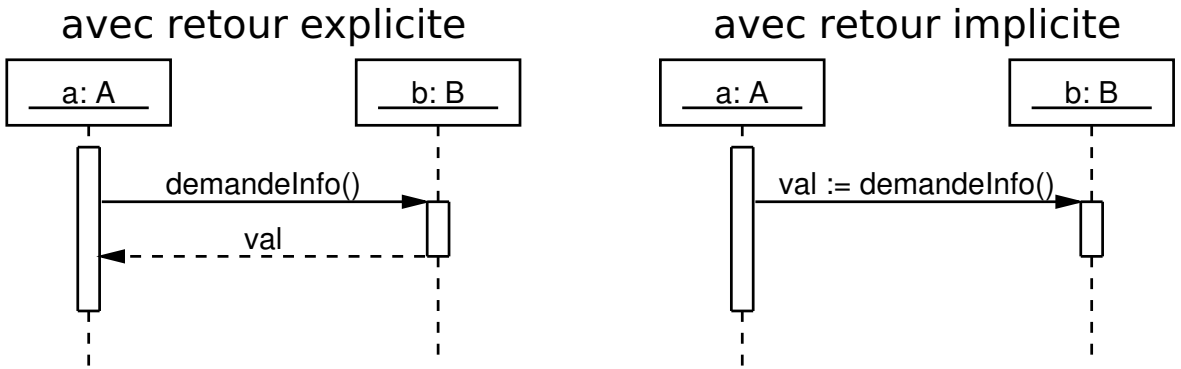
**Principaux constituants :**



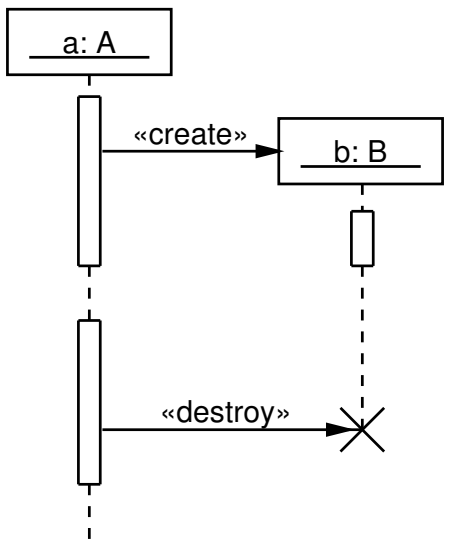

- Le temps s'écoule du haut vers le bas
- La barre d'activation montre que du code de l'objet est en train de s'exécuter
- On se limite aux messages synchrones (il existe aussi des messages asynchrones)

# Diagramme de séquence (2)

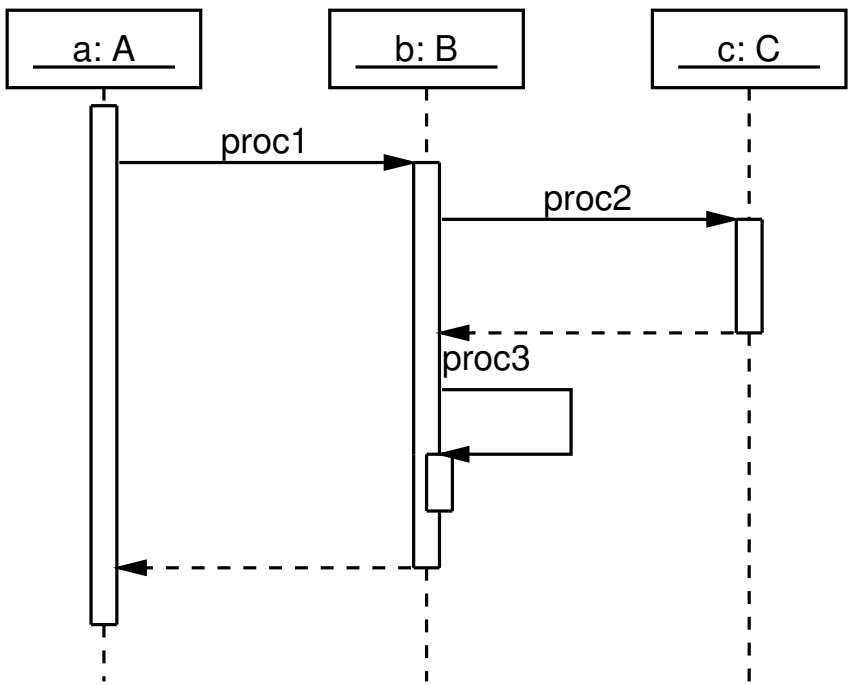
## Retour de méthode




## Création et destruction d'objet



# Diagramme de séquence et code Java




```
...
public class B {
    ...
    ... proc1() {
        ...
        c.proc2()
        ...
        this.proc3()
        ...
    }
    ... proc3() {
        ...
}   }

public class C {
    ...
    ... proc2() {
        ...
}   }
```

- Souvent, pas de correspondance parfaite entre diagramme de séquence et code Java.
- Un diagramme de séquence est un exemple, une simplification.

# Des raffinages aux diagrammes de séquence

```
Comment « Arbitrer un jeu entre un devin et un maître » ?
    Demander au maître de choisir un nombre
    Indiquer au devin les limites du jeu
    Répéter
        Demander au devin de faire une proposition
        Demander au maitre une indication
        Donner au devin l'indication
    JusquÀ nombre trouvé
    Féliciter le devin
```

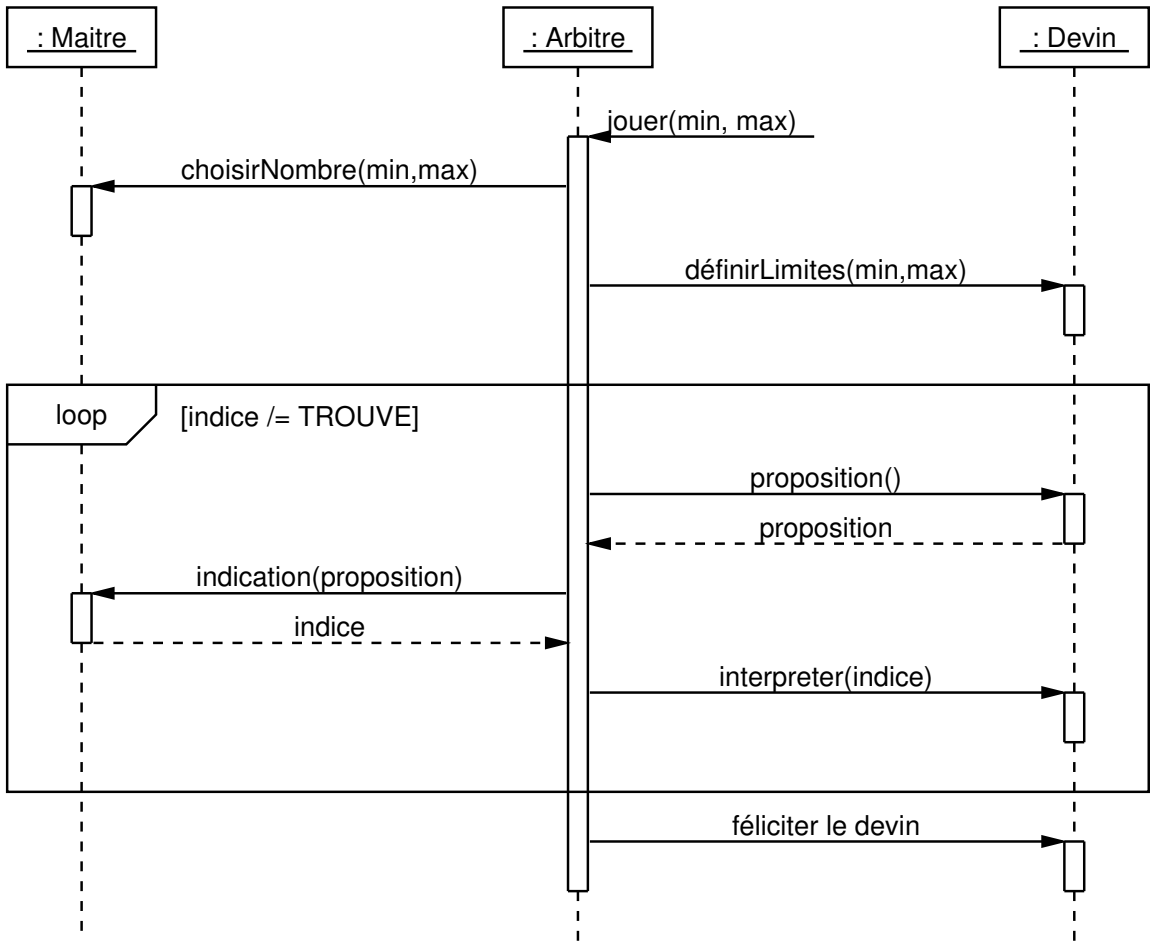
- Les participants sont explicités : devin, maître... et arbitre
- La règle « une étape est un verbe à l'infinitif » est souvent remplacée par « une étape est de la forme : sujet verbe complément »

```
L'arbitre demande au maître de choisir un nombre
L'arbitre indique au devin les limites du jeu
...
```

Comment représenter les structures de contrôle sur un diagramme de séquence ?

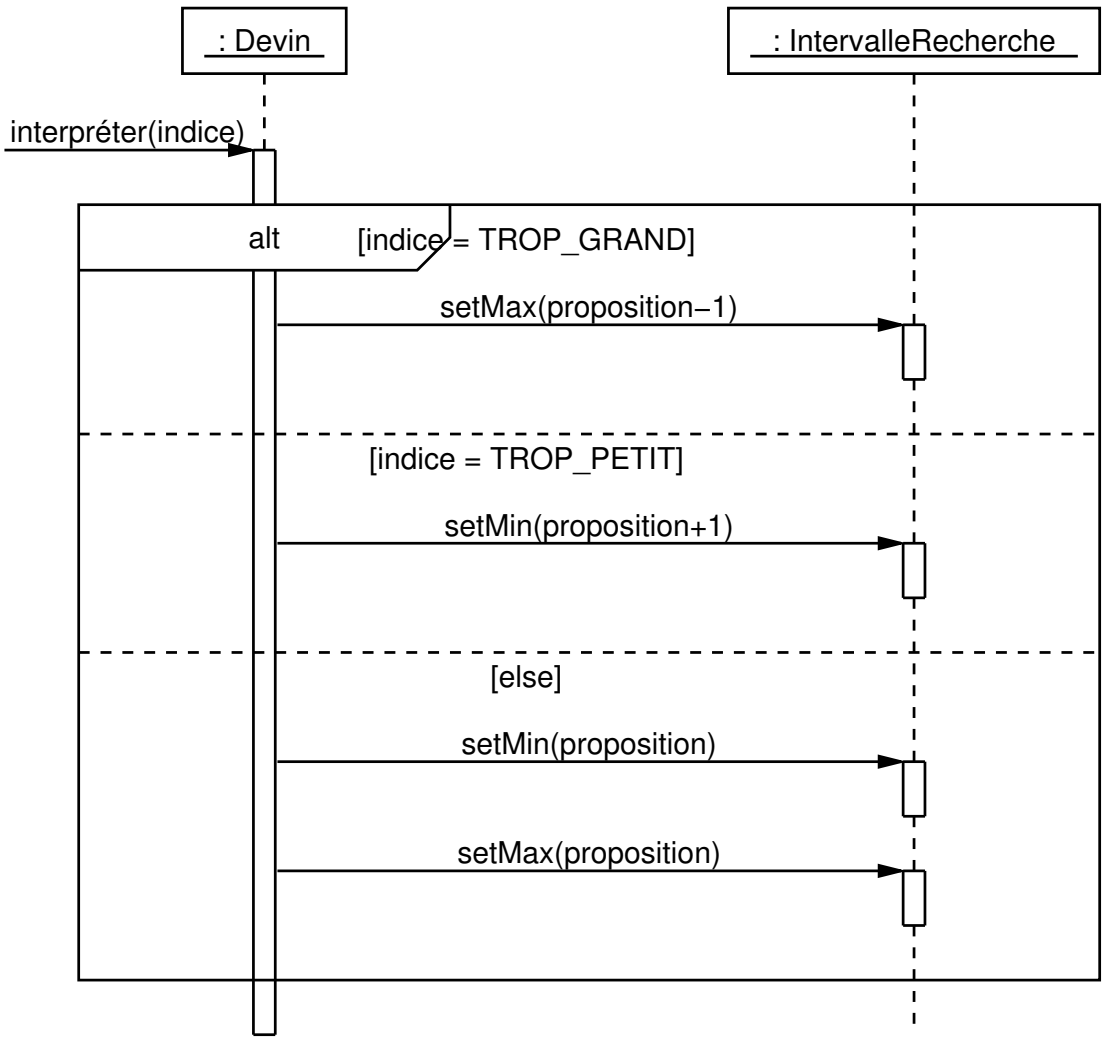
# Les cadres d'interaction (frame), depuis UML 2

## Vers l'expression d'un algorithme



# Les cadres d'interaction (frame), depuis UML 2 (2)

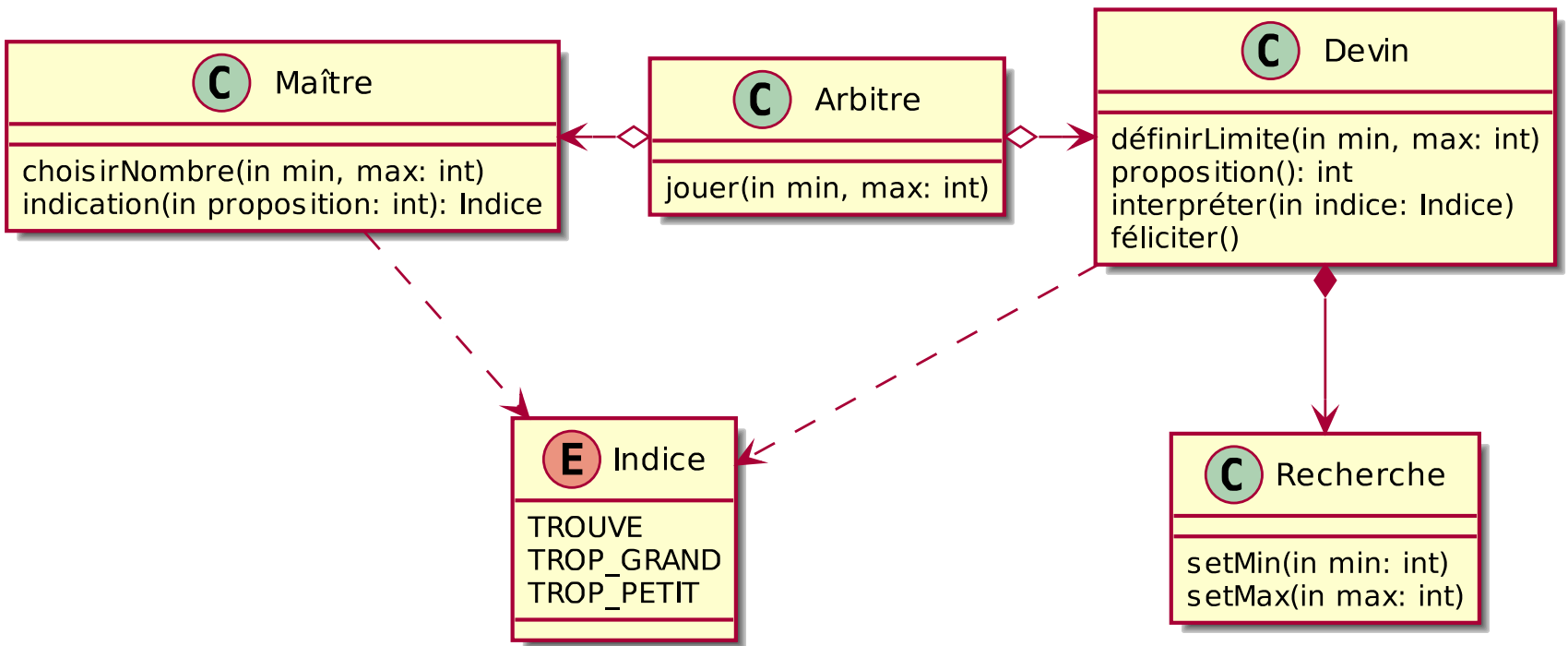
## Vers l'expression d'un algorithme



# Diagramme de classe correspondant

Du diagramme de séquence, on peut déduire des éléments du diagramme de classe :

- les classes : type des objets du diagramme de séquence
- les méthodes : message sur une interaction (dans la classe cible)
- les dépendances entre classes (mais pas le type de relation)



# Bilan sur les diagrammes de séquence

- Cadres d'interaction : ajoutés dans la dernière version majeure (UML 2)
- Étiquettes possibles pour les cadres d'interaction :
  - alt : alternatives
  - loop : répétition (while ou foreach)
  - opt : optionnel (if sans else)
  - par : exécution parallèle
  - negative : une interaction invalide
  - ...
- Ils s'éloignent de l'objectif initial des diagrammes de séquence :
  - on ne montre pas simplement une exécution particulière
  - mais plusieurs exécutions sur un même diagramme
  - ceci répond à un souci de factorisation... parfois au détriment de la lisibilité
  - voir les diagrammes d'activité d'UML
- Les diagrammes de séquence sont longs à dessiner (même avec un outil dédié)
- Un diagramme de séquence explicite de nombreux choix : objets, méthodes...
  - souvent trop tôt quand on en est à comprendre le problème
  - un raffinage (décomposition informelle peut alors être préférable)
  - à utiliser surtout pour expliquer un algorithme dans des conditions particulières