

# Conception orientée-Objets avec UML

Rémi Bastide

IRIT – CUFR J.F. Champollion

<http://ihcs.irit.fr/bastide>

[Remi.Bastide@irit.fr](mailto:Remi.Bastide@irit.fr)



# Plan

- Présentation d'UML
- COO avec UML
  - Diagrammes de classe
  - Diagrammes d'objets
  - Diagrammes d'interaction
- Conception par Objets
  - Méthode des CRC Cards
  - Transition vers UML
  - Mapping UML ► Java

# Historique d'UML

- Fin des années 80 : compétition des méthodes d'analyse et de conception OO
  - Booch : particulièrement adaptée au design et à l'implémentation
  - OOSE (Jacobson) : expression des besoins
  - OMT-2 (Rumbaugh) : analyse et applications orientées-données
- 1994 : Rumbaugh rejoint Booch chez Rational
- 1995 : Jacobson rejoint Rational
- 14 novembre 1997 : UML adopté par l'OMG

# Qu'est-ce qu'UML ?

- « UML est un langage pour visualiser, spécifier, concevoir et documenter les artefacts d'un système à base logicielle »
  - Langage : lexique (graphique), syntaxe (diagrammes), sémantique
  - Visualiser : représentation graphique
  - Spécification : précis, complet, non-ambigu
  - Construction : translation vers des langages de programmation
  - Documentation : des besoins aux tests

# Le Langage

- langage = lexique + syntaxe + sémantique
  - Lexique : les mots du dictionnaire
    - Dans UML : lexique graphique (symboles)
  - syntaxe = Règles par lesquelles les éléments du lexique (e.g., mots) sont assemblées en expressions (e.g., phrases, clauses)
  - sémantique = Règles par lesquelles on donne un sens aux expressions syntaxiques
- *UML Notation Guide*
  - définit la syntaxe graphique d'UML
- *UML Semantics*
  - définit la sémantique d'UML

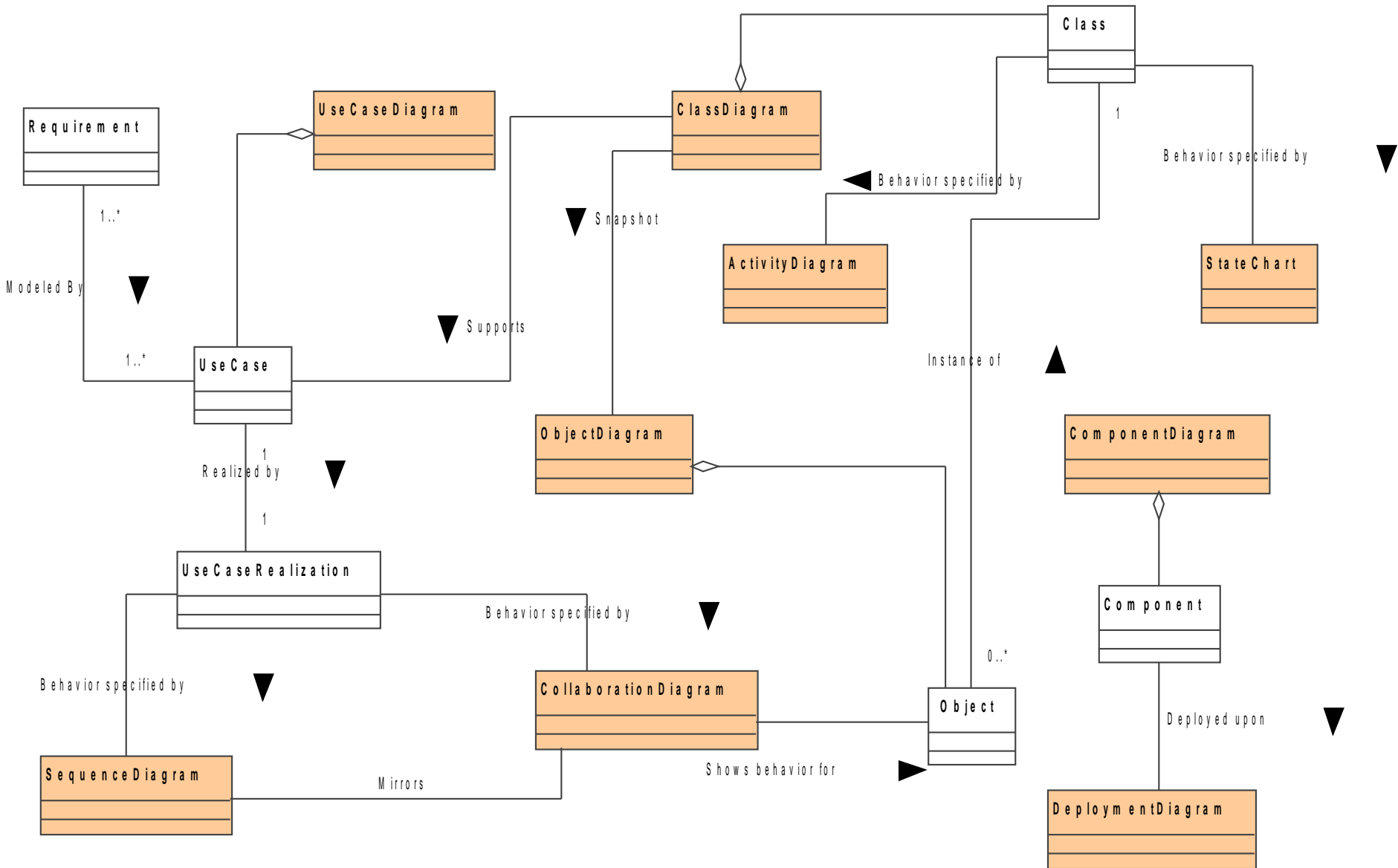
# Les briques de base de l'UML

- Des choses...
  - Structurelles
    - Classe, Interfaces, Collaborations, Use Cases...
  - Comportementales
    - Messages et machines à états
- Des relations entre les choses
  - Dépendances, Associations, Généralisation, Réalisation
- Des diagrammes

# Les diagrammes de l'UML

- Diagramme de Classe
  - Diagramme d'Objets
  - Diagramme Use Case
  - Diagrammes d'interactions
    - Diagramme de Séquence
    - Diagramme de Collaboration
  - StateCharts
  - Diagramme d'Activité
  - Diagrammes de Composants
  - Diagramme de Déploiement
- } isomorphes

# UML par lui-même





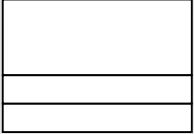
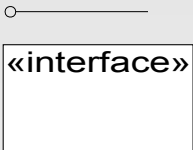
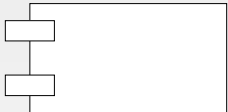
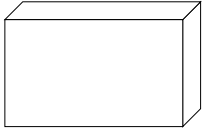
# Fonction des diagrammes

- Diagrammes prescriptifs : décrivent le système tel qu'il doit être ou se comporter à tout moment
  - Classe, StateCharts, Use Cases, Activités, Composants, Déploiement
- Diagrammes descriptifs : illustrent un état ou un comportement possible et typique du système
  - Objet, Séquence, Collaboration

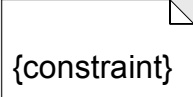
# Modèle structurel

- Une vue d'un système qui met l'accent sur la structure des objets, avec leurs classificateurs, leurs relations, leurs attributs et leurs opérations

# Éléments de modélisation structurelle

Élément	Description	Syntaxe
<b>classe</b>	Description d'un ensemble d'objets qui ont même attributs, opérations, méthodes, relations et sémantique.	
<b>interface</b>	Ensemble nommé d'opérations qui caractérisent le comportement d'un élément.	
<b>composant</b>	Partie physique et remplaçable d'un système qui encapsule une implémentation et fournit la réalisation d'un ensemble d'interfaces.	
<b>Nœud</b>	Objet physique qui représente une ressource de calcul	

# Éléments de modélisation structurelle (suite)

Élément	Description	Syntaxe
<b>contrainte<sup>1</sup></b>	Condition ou restriction sémantique.	


<sup>1</sup> Mécanisme d'extension.

# Modélisation structurelle: Relations

Relation	Description	Syntaxe
association	Relations entre 2 classifieurs ou +, qui implique des connexions entre leurs instances.	—————
agrégation	Forme d'association qui spécifie une relation « tout / partie » entre agrégat (tout) et composant (partie).	—————◇ —————◇
généralisation	Relation taxonomique entre un élément général et un élément plus spécifique.	—————▷
dépendance	Relation entre 2 éléments, dans laquelle un changement sur un élément (indépendant) peut affecter l'autre (dépendant).	----->

# Modélisation structurelle: Relations

(suite)

<b>Relation</b>	<b>Description</b>	<b>Syntaxe</b>
<b>réalisation</b>	Relation entre spécification et implémentation.	

# Diagrammes structurels

- Montrent la structure statique d'un modèle
  - Les entités qui existent (e.g., classes, interfaces, composants, nœuds)
  - Leur structure interne
  - Leurs relations avec d'autres entités
- Ne montrent pas
  - Des informations temporelles ou dynamiques
- Catégories
  - Diagrammes structurels statiques
    - diagramme de classe
    - diagramme d'instances
  - Diagrammes d'implémentation
    - diagrammes de composants
    - diagrammes de déploiement

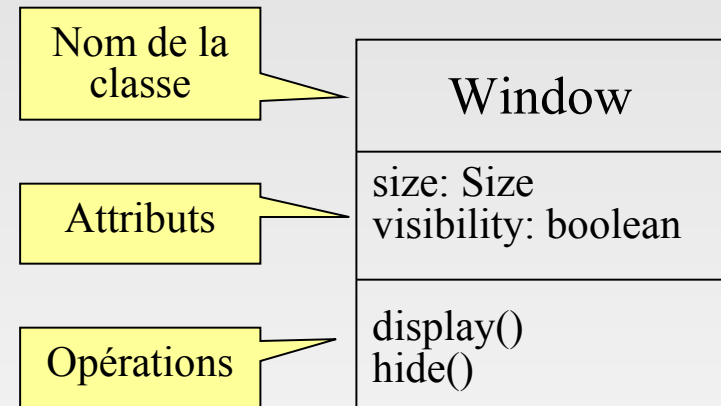
# Diagrammes de classes

- Les *diagrammes de classes* représentent un ensemble de classes, d'interfaces et de collaborations, ainsi que leurs relations. Ce sont les diagrammes les plus fréquents dans la modélisation des systèmes à objets. Ils présentent la vue de conception statique d'un système.



# Les classes

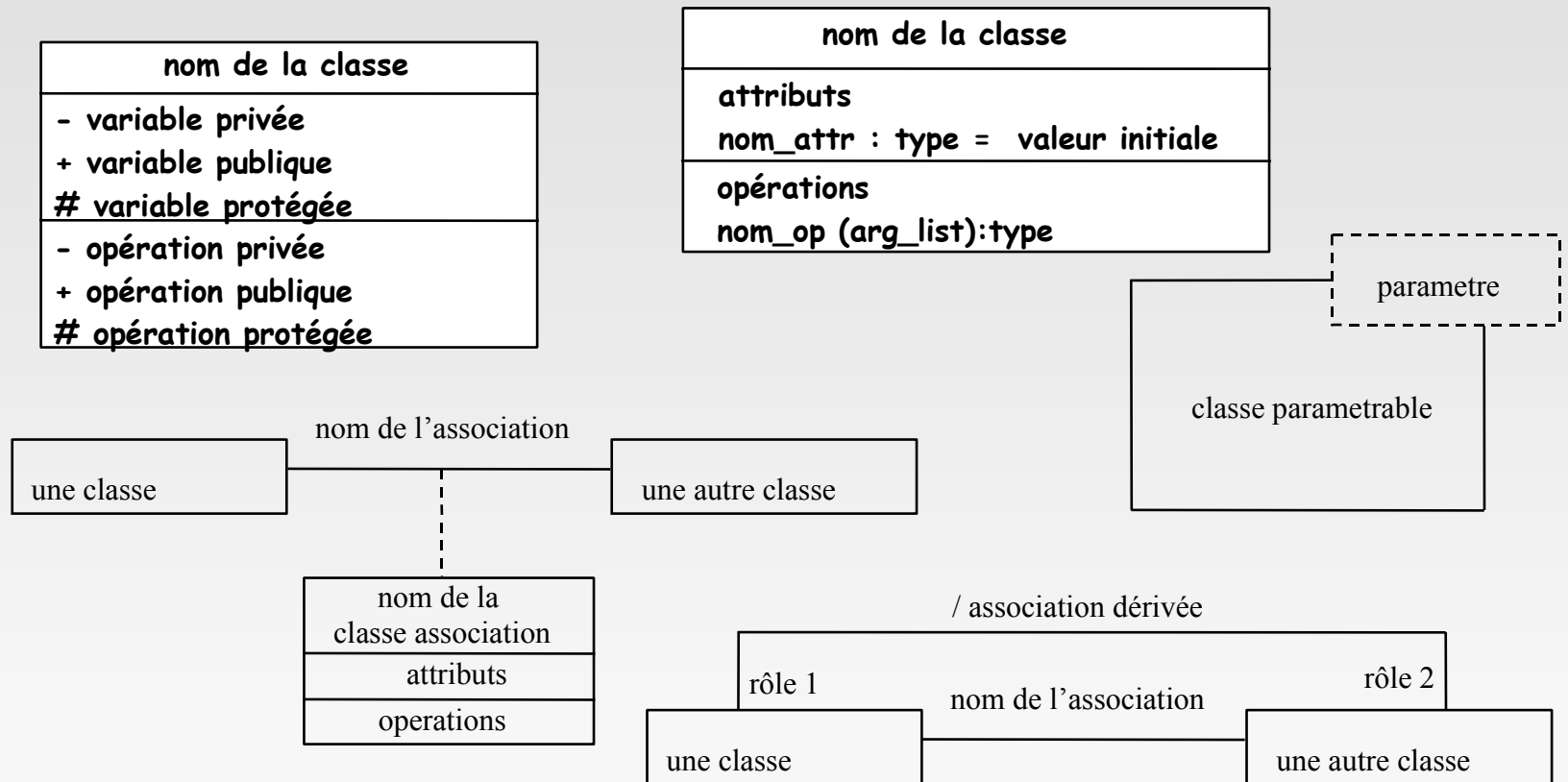
- La classe est une description abstraite d'un ensemble d'objets
- La classe peut être vue comme la factorisation des éléments communs à un ensemble d'objets
- La classe décrit le domaine de définition d'un ensemble d'objets



# Définition d'une classe

- Description d'un ensemble d'objets qui ont même structure et même sémantique
  - Nom
  - Attributs
  - Opérations
  - Responsabilités
    - Exprimées en langage naturel

# Diagrammes de classe.



# Réprésentation des Classes

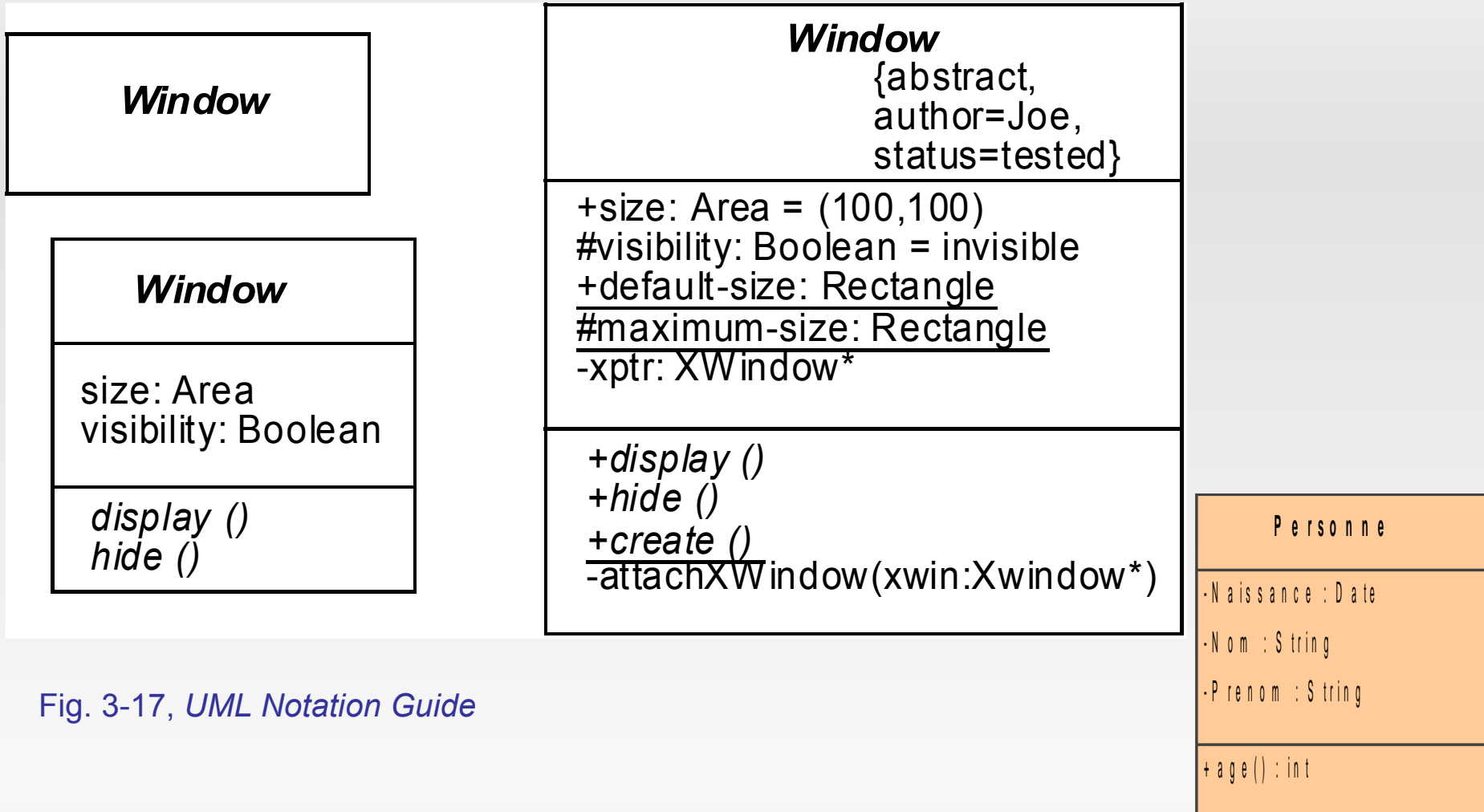


Fig. 3-17, UML Notation Guide

# Visibilité des attributs / opérations

- Public (+)
  - Visible et utilisable par toute autre classe (utilisation très limitée pour les attributs)
- Protected (#)
  - Visible et utilisable par toute spécialisation de la classe
- Private (-)
  - Visible uniquement par la classe elle-même
- Sinon
  - Même sémantique que dans java (package)

# Opérations (méthodes)

- Implémentation d'un service offert par l'objet, correspondant à une partie de ses responsabilités
  - Accesseurs : une opération qui renvoie une information sur l'état de l'objet (fonction)
  - Modifieur : une opération qui modifie l'état de l'objet (procédure)
  - Constructeur : une opération de la classe qui permet d'initialiser une nouvelle instance
    - Les propriétés de classe sont soulignées
      - mot-clé “static” en java

# Attributs/opérations: Syntaxe

- **Attributs :**

- `nomAttribut : type = valeur initiale`

- **exemples :**

- `prenom : String`

- `size : int = 100`

- **Opérations :**

- `nomOpération(param1 : type1, param2 : type2...):typeResultat`

- **exemples**

- `getName() : String`

- `setName(newName : String)`

# Utilisation des attributs

- Utilisation plus limitée que dans une modélisation de type entité-association
  - La définition d'un attribut contraint l'implémentation future de la classe
  - Préférer :
    - des accesseurs / modifieurs (getX, setX)
    - des associations (plus riche sémantiquement)
    - A réserver à des types primitifs (entier, réel) ou des « structures de données » simples (String, Date)
    - Ne pas utiliser des structures de type tableau, liste, ... Sauf exception
- Conception « centrée sur les responsabilités » plutôt que « centrée sur la structure »



# Interfaces

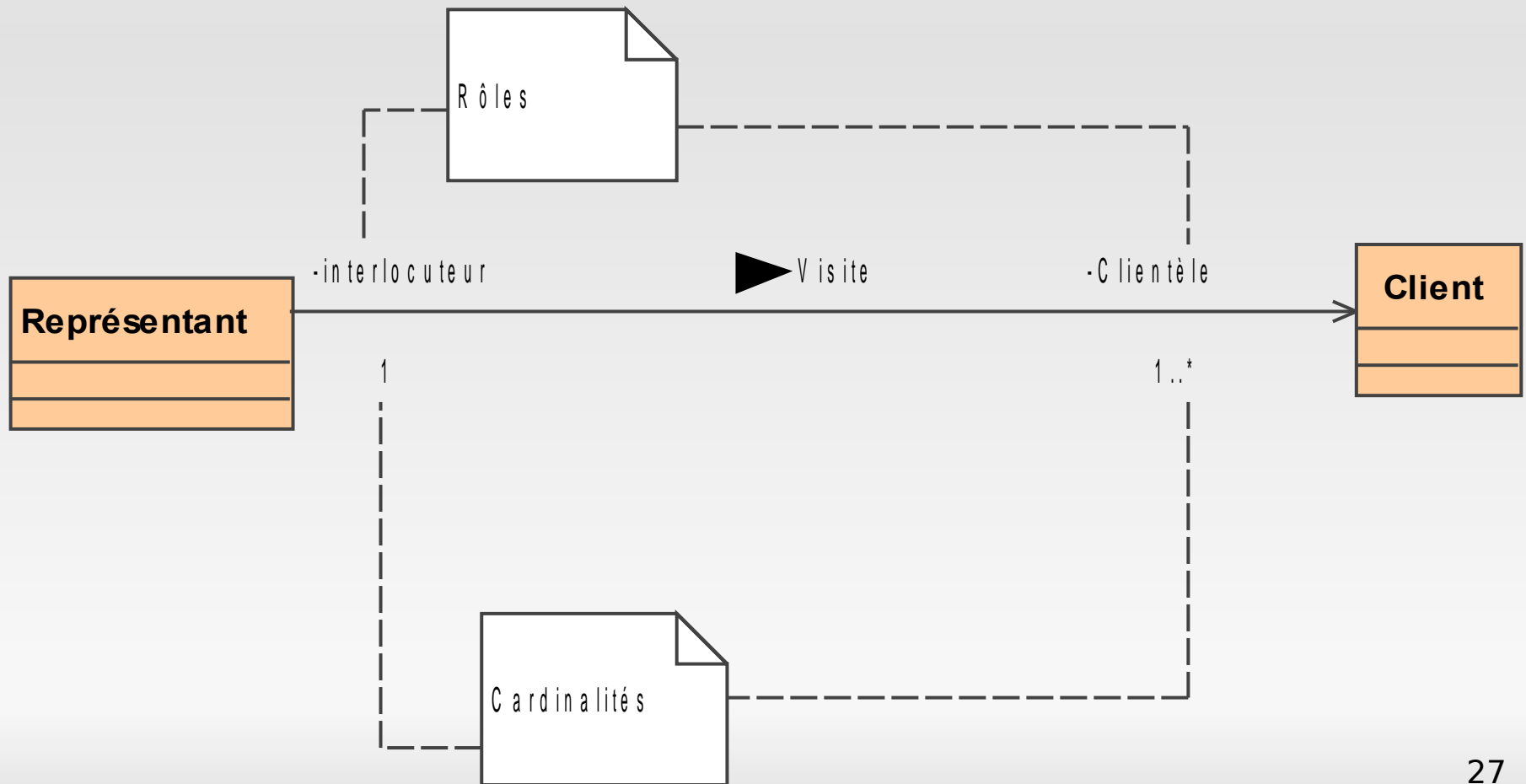
- Spécification des opérations visibles
  - Classes, paquetage, composants
- Contrat sans implémentation
  - Pas d'attributs ni d'associations
  - Peut être cible d'une association monodirectionnelle



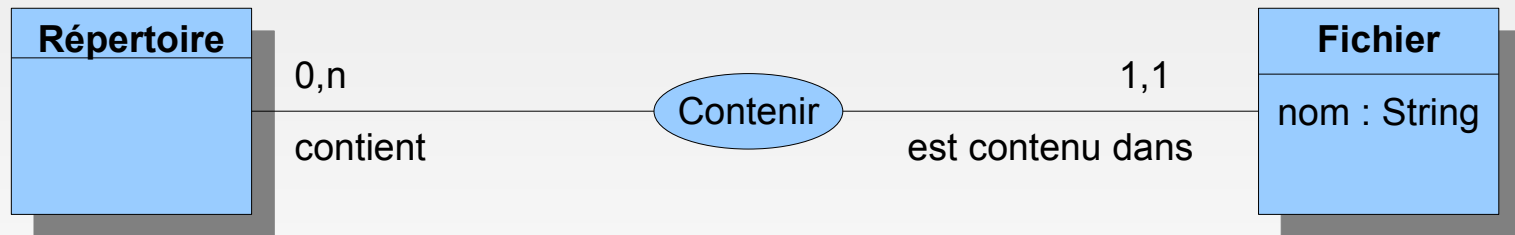
# Associations

- Expriment une relation permanente entre instances de 2 ou plusieurs classes
  - Nom de l'association
  - Sens de lecture
  - Cardinalités
  - Rôles
  - Visibilité des rôles
  - Navigabilité

# Associations

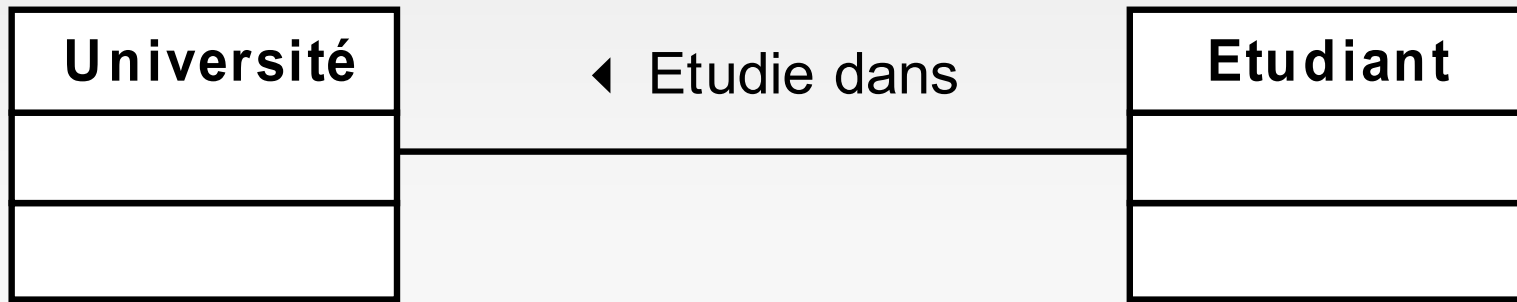


# UML / Entité-Association



# Nommage des associations

- Indication du sens de lecture



# Navigabilité des associations

- Par défaut, la navigabilité est bidirectionnelle  
A partir d'un objet d'une des 2 classes, on peut atteindre les objets de l'autre classe
- On peut restreindre la navigabilité en ajoutant un flèche à un extrémité
  - Navigation uni-directionnelle

# Navigabilité

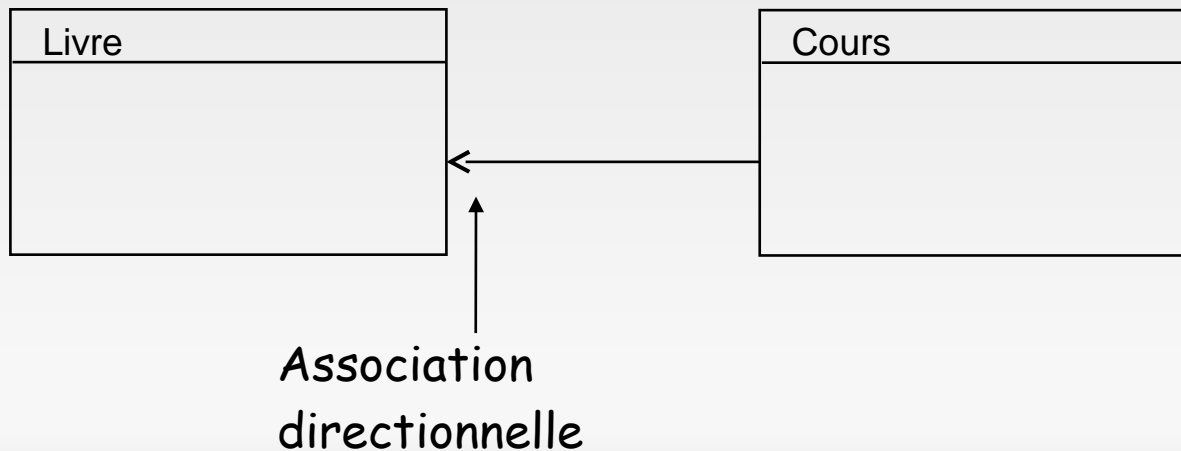
- étant donnée une association non décorée entre deux classes, on peut naviguer d'un type d'objet vers un autre type d'objet.
- par défaut une association est navigable dans les deux sens.
- une indication de navigabilité suggère en général qu'à partir d'un objet à une extrémité on peut directement et facilement atteindre l'un des objets à l'autre extrémité.
- lors d'une mise en œuvre programmée, ceci peut suggérer par exemple qu'un objet source mémorise une référence directe aux objets cible.

- Exemple :
  - étant donné un utilisateur, on désire pouvoir accéder à ses mots de passe
  - étant donné un mot de passe, on ne souhaite pas pouvoir accéder à l'utilisateur correspondant



# Association directionnelle

- Un cours connaît les livres qui lui sont nécessaires mais pas le contraire.
- Si la flèche est absente, la navigation est bi-directionnelle



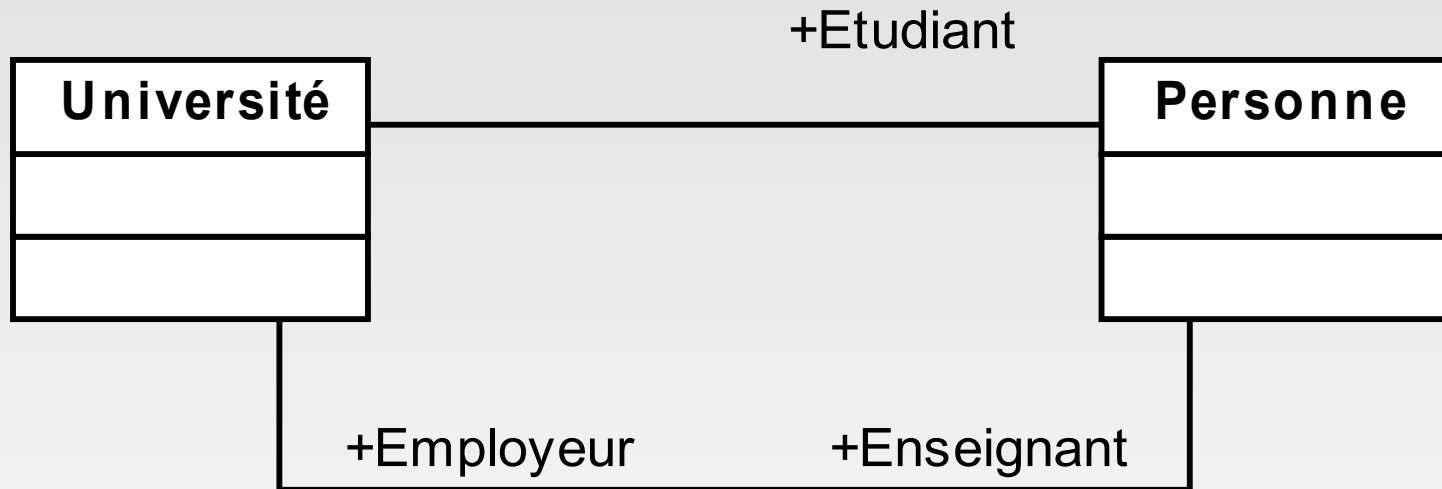


# Directionnalité

- Un « détail d'implémentation » qui n'est pas toujours nécessaire dans les phases initiales de conception.
- Impact important sur les techniques d'implémentation et sur les performances.

# Nommage des rôles

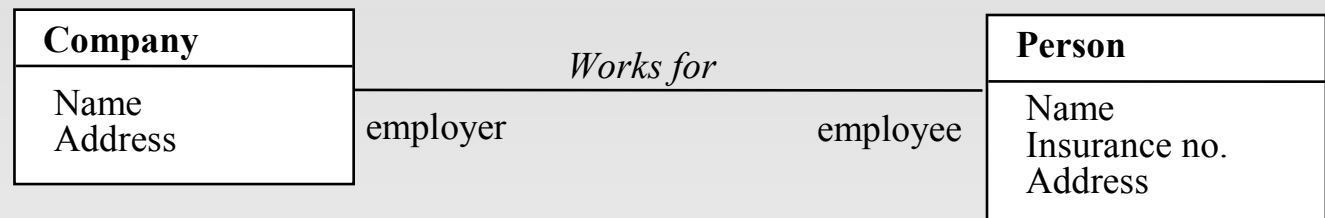
- Le rôle décrit une extrémité d'une association



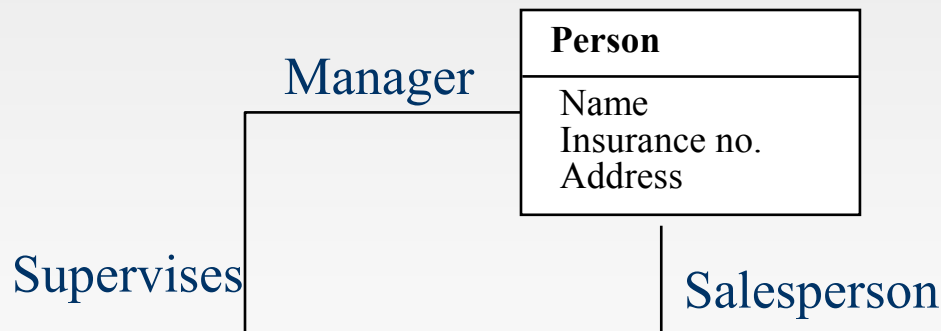
- Une personne peut être considérée comme étudiant ou enseignant **du point de vue de l'université**

# Noms des rôles

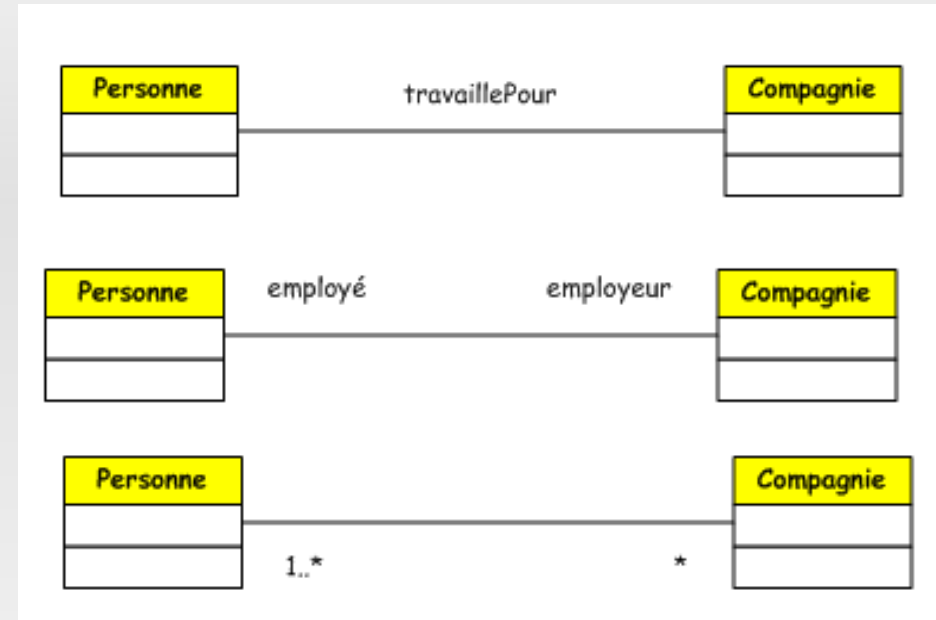
- Rôle : identifie l'extrémité d'une association



- Les noms de rôles sont obligatoires pour les associations réflexives



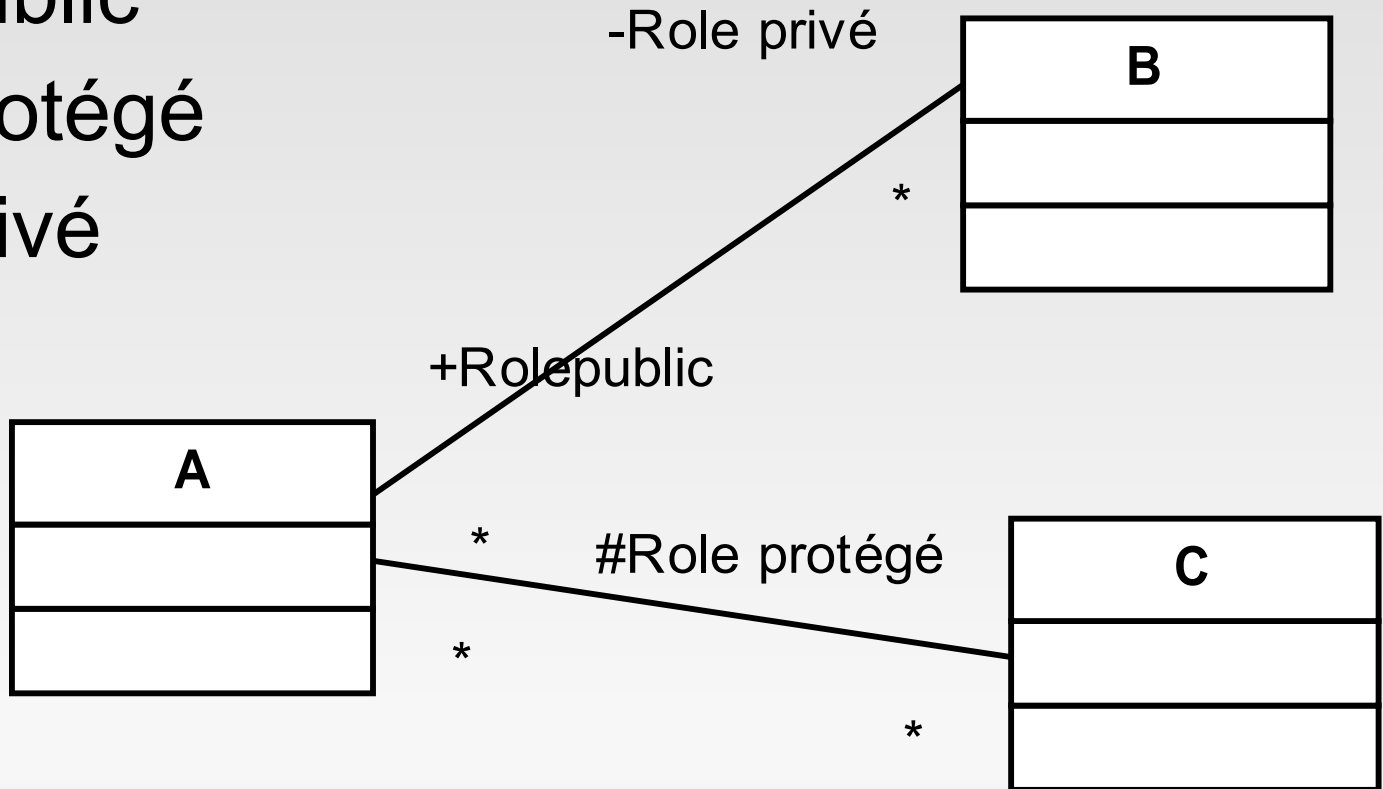
# Multiplicité des rôles



- 1 Un et un seul
- 0..1 Zéro ou un
- M .. N De M à N (entiers naturels)
- \* Un nombre quelconque
- 0 .. \* Un nombre quelconque
- 1 .. \* un ou plusieurs

# Visibilité des rôles

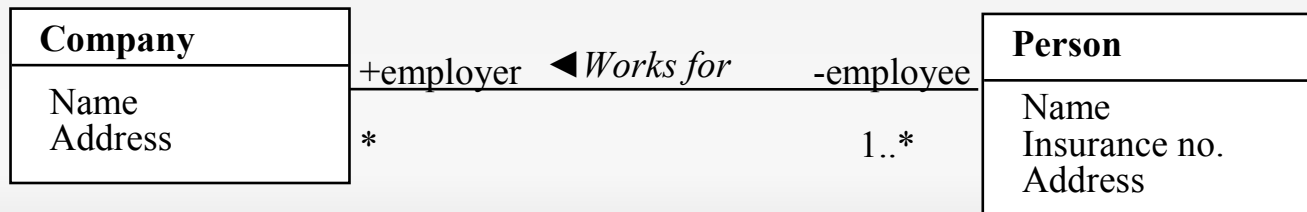
- Public
- Protégé
- Privé



# Les rôles définissent des responsabilités

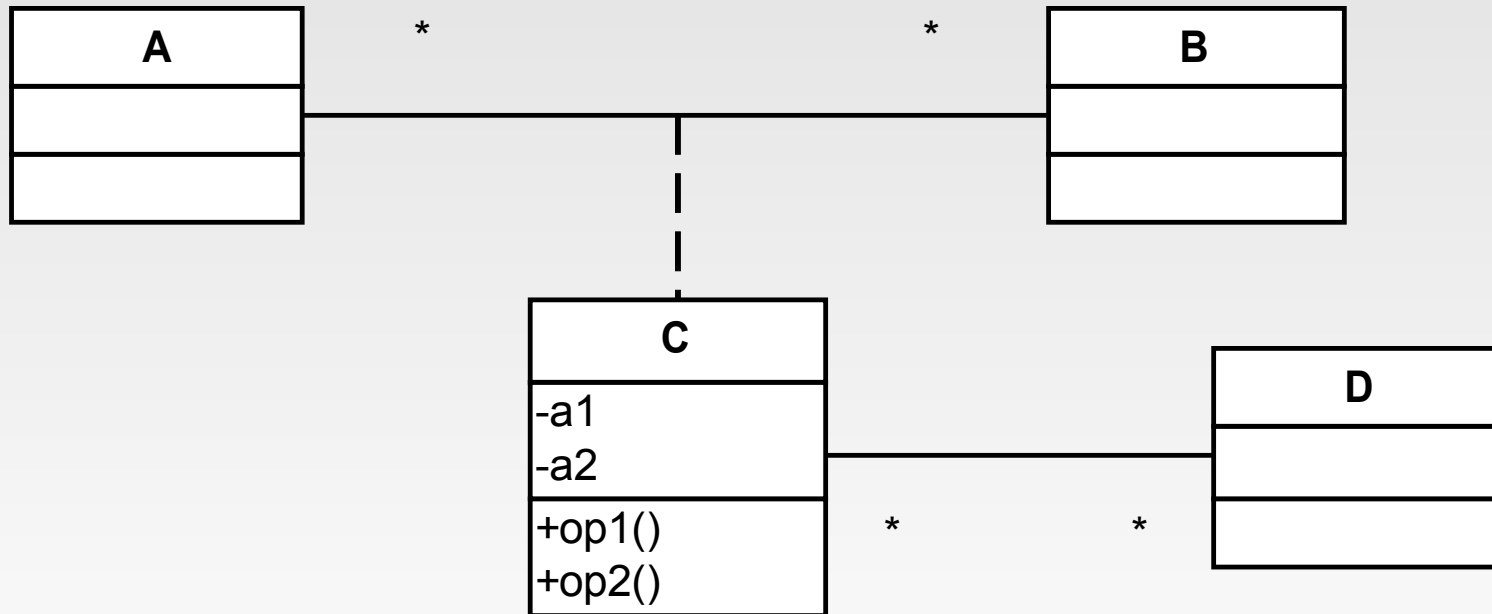
- Connaissant une personne, on doit pouvoir retrouver ses employeurs
  - Il doit y avoir une méthode publique dans `Person` qui renvoie une collection d'instances de `Company`
  - exemple java :

```
public Collection<Company> getEmployers();
```

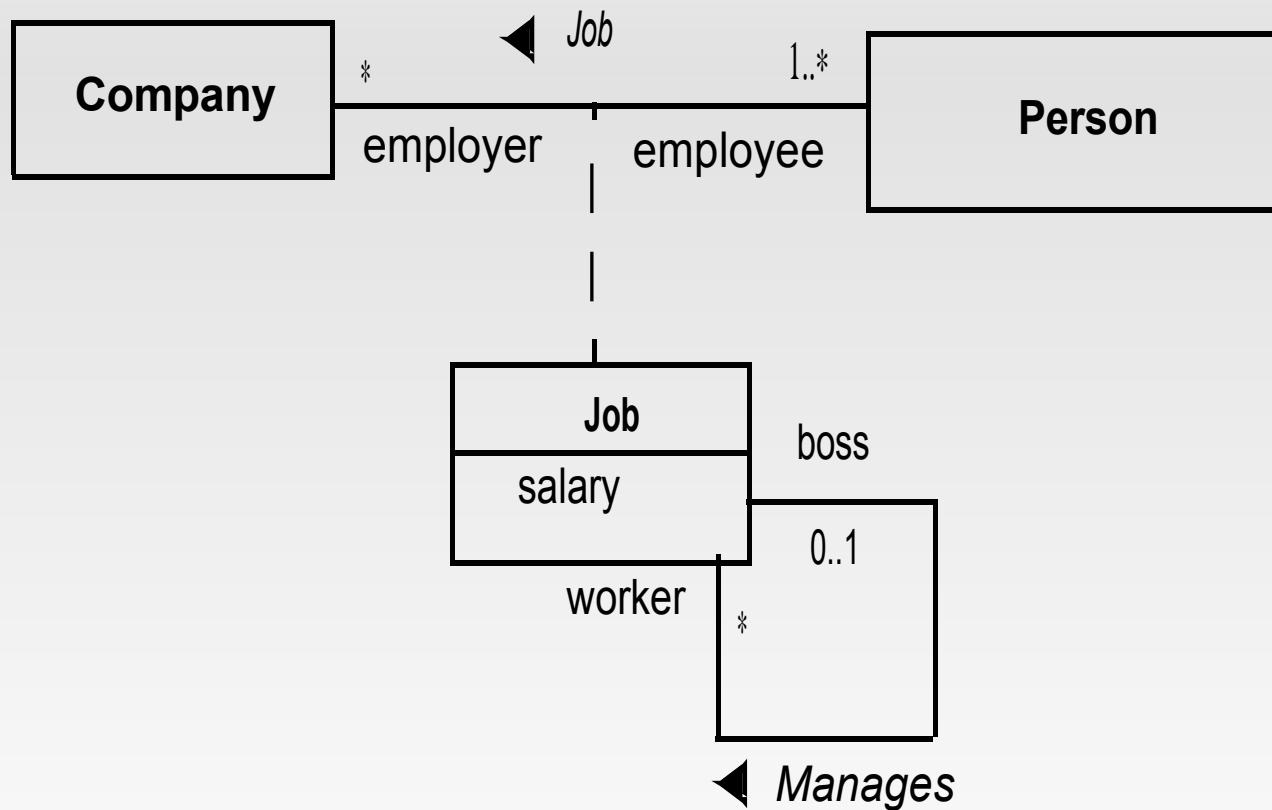


# Les classes-associations

- L'association est porteuse de propriétés et/ou d'opérations

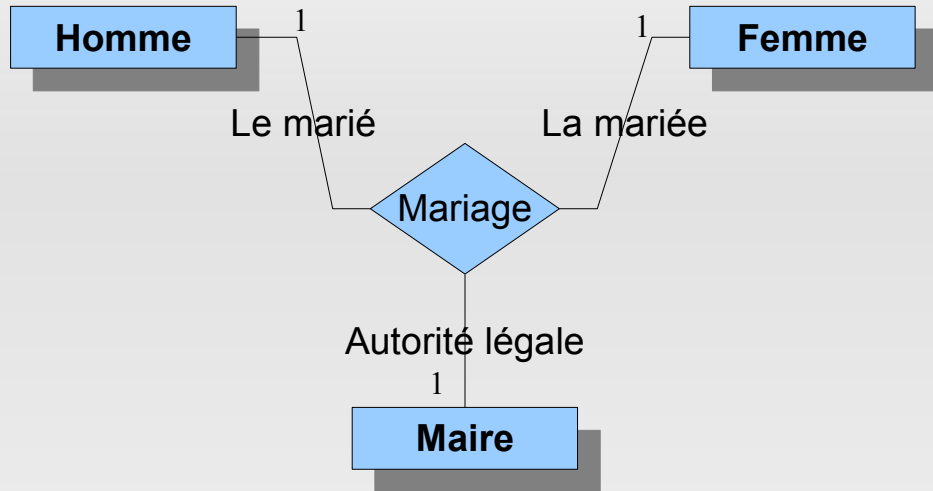


# Associations





# Associations ternaires

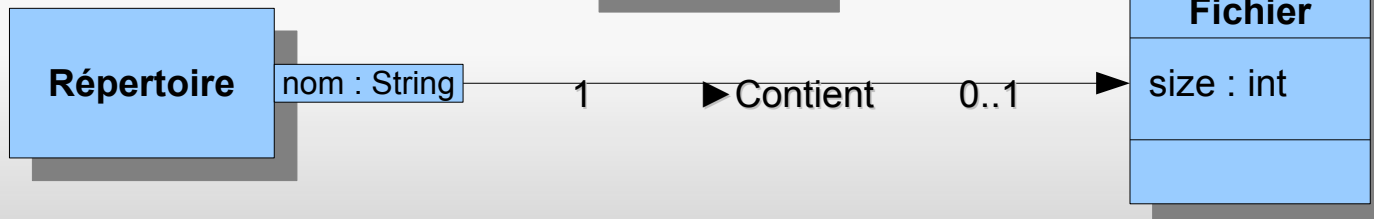
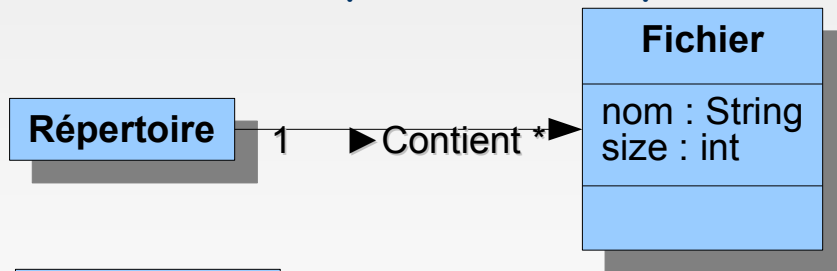
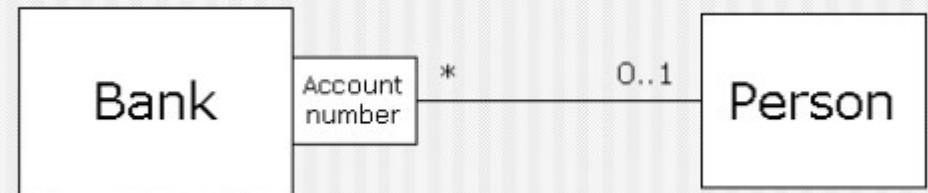


# Association qualifiée

- Une association qualifiée met en relation deux classes sur la base d'une clef.
- La technique associée de mise en œuvre est souvent un tableau associatif ou un dictionnaire.

Une personne peut être associée à plusieurs banques.

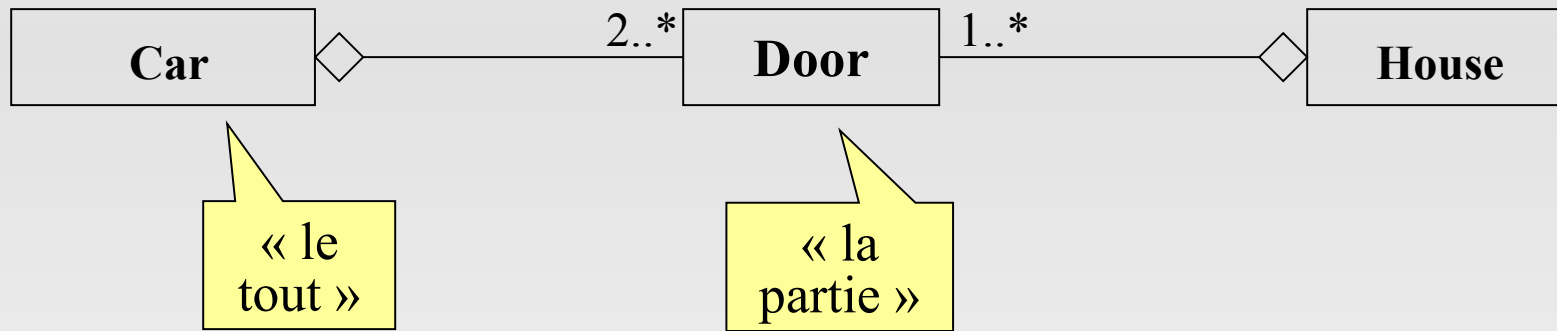
Étant donné une banque et un numéro de compte, il y a au plus une personne ayant ce numéro de compte à cette banque.



# Agrégation et composition

- Connexions sémantiques bidirectionnelles antisymétriques
- Forme de relation qui exprime un couplage plus fort entre les instances
- Représentation des relations de type :
  - maître et esclaves
  - tout et parties
  - composé et composant.

# Agrégation : exemple

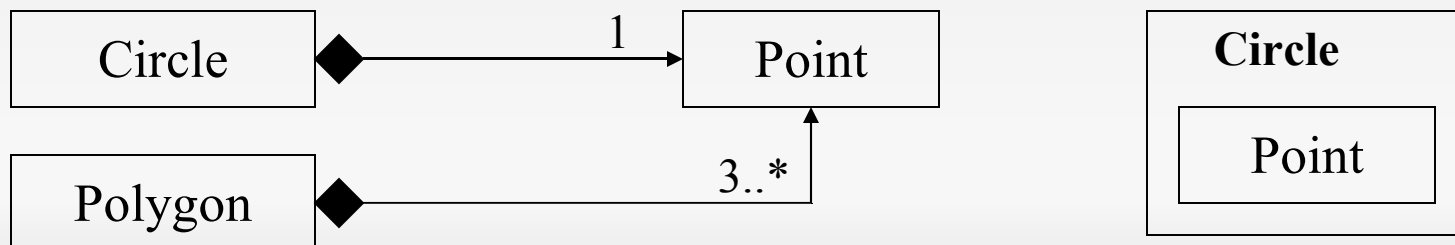


# Agrégation (suite)

- Quand utiliser l'agrégation ?
  - Si on utilise « est une partie de » dans la description en langage naturel
    - La porte fait partie de la voiture
  - Si des opérations sur « le tout » sont automatiquement appliquées à « la partie »
    - Si on déplace la voiture, on déplace la porte
  - Si des propriétés sont automatiquement propagées du « tout » à « la partie »
    - La voiture est bleue, donc sa porte est bleue.
  - Si il y a une asymétrie intrinsèque dans la relation
    - La porte est une partie de la voiture, la voiture n'est pas une partie de la porte

# Composition

- Une forme plus forte de l'agrégation
  - Le « tout » est seul possesseur de « la partie ».
    - La partie appartient à un seul tout
  - Le cycle de vie de la partie dépend de celui du tout
    - Le composé gère la création et la destruction de ses composants



# Composition = Attributs

« Une fenêtre est composée d'un titre, d'un corps et de 2 scrollbars »

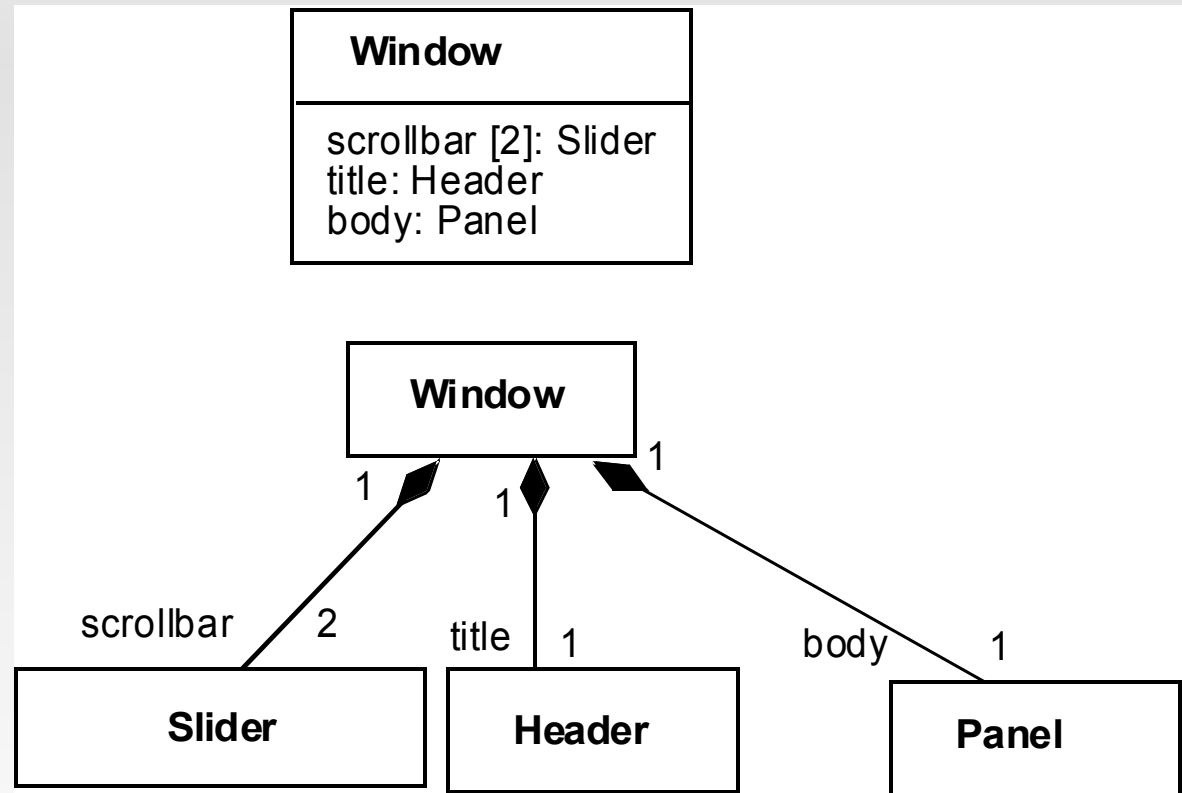
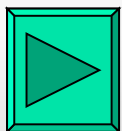
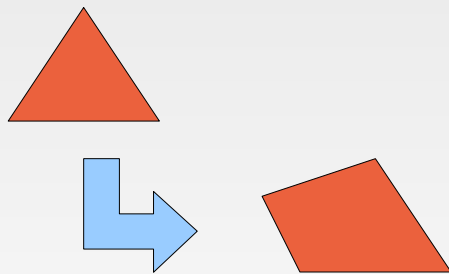
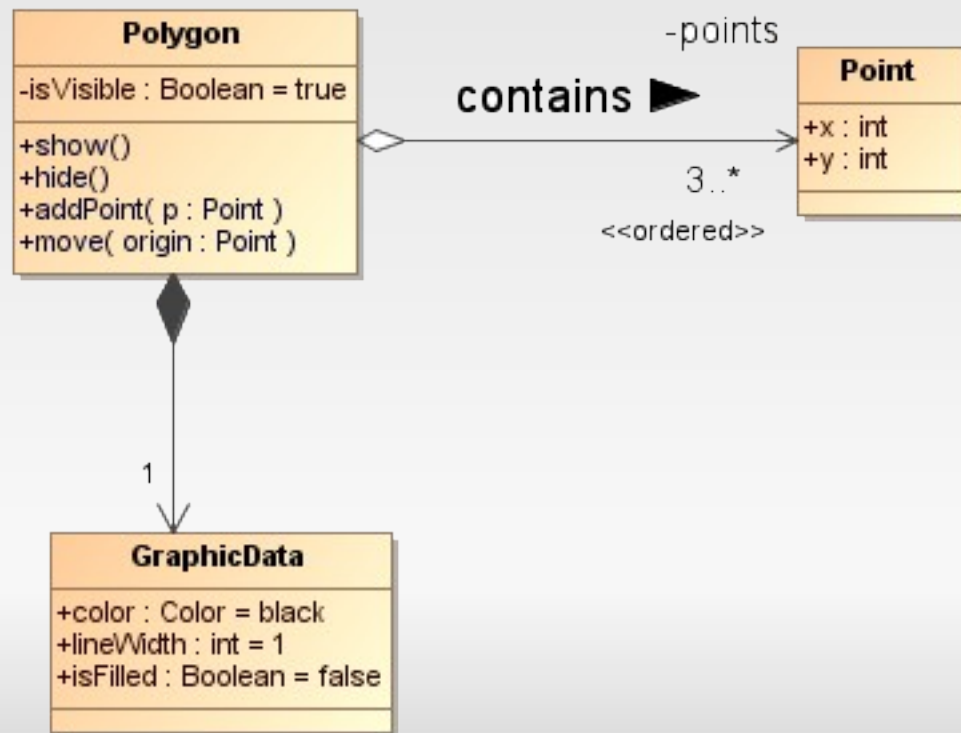


Fig. 3-36, UML Notation Guide

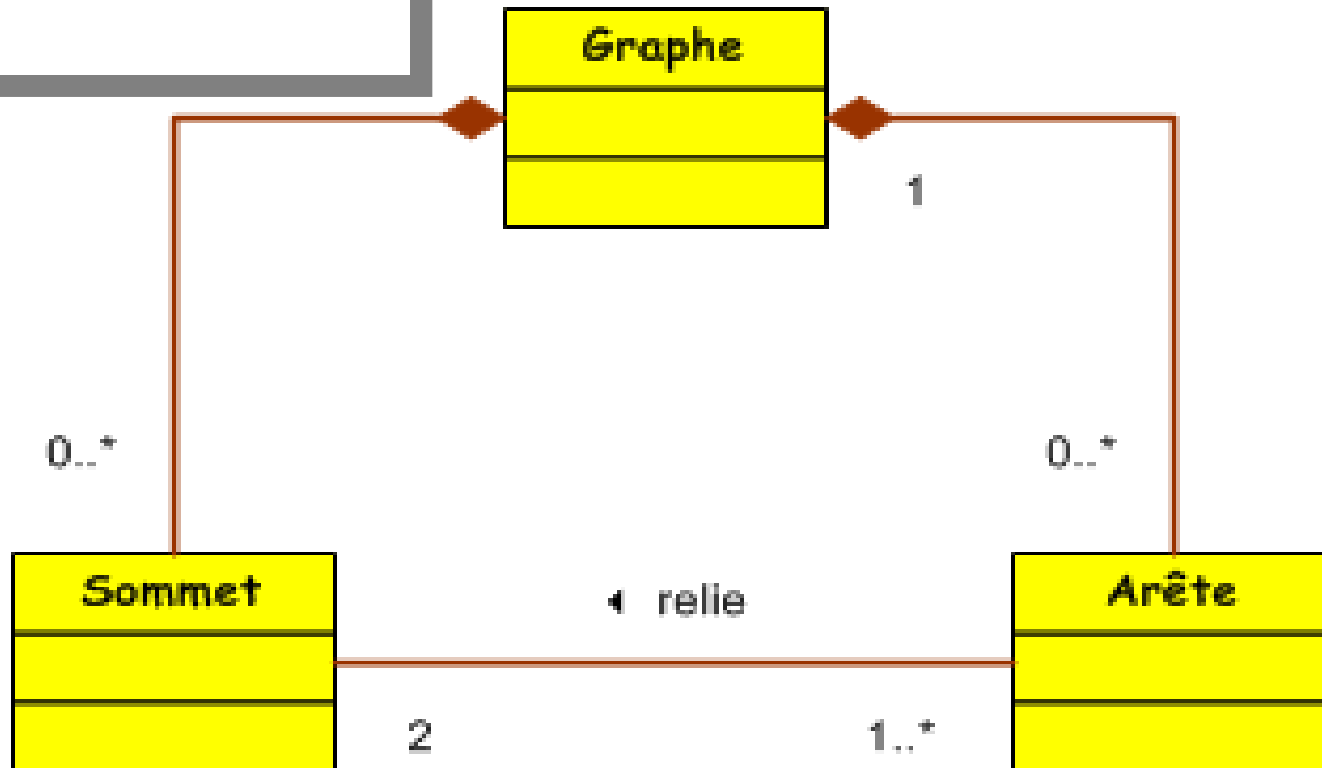
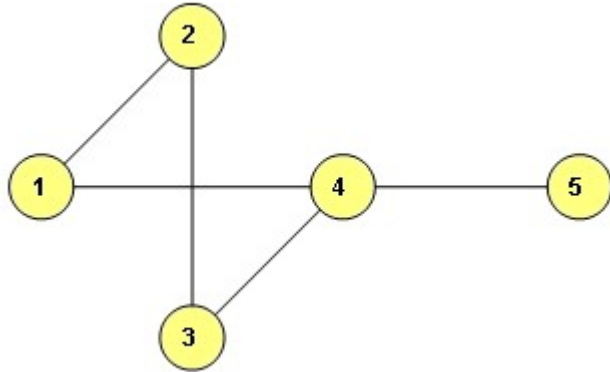
# Agrégation / Composition

- Polygon ► Point : Agrégation
  - On peut rajouter des points pendant la durée de vie du Polygone
- Polygon ► GraphicData : Composition
  - GraphicData est créé ou détruit en même temps que le polygone

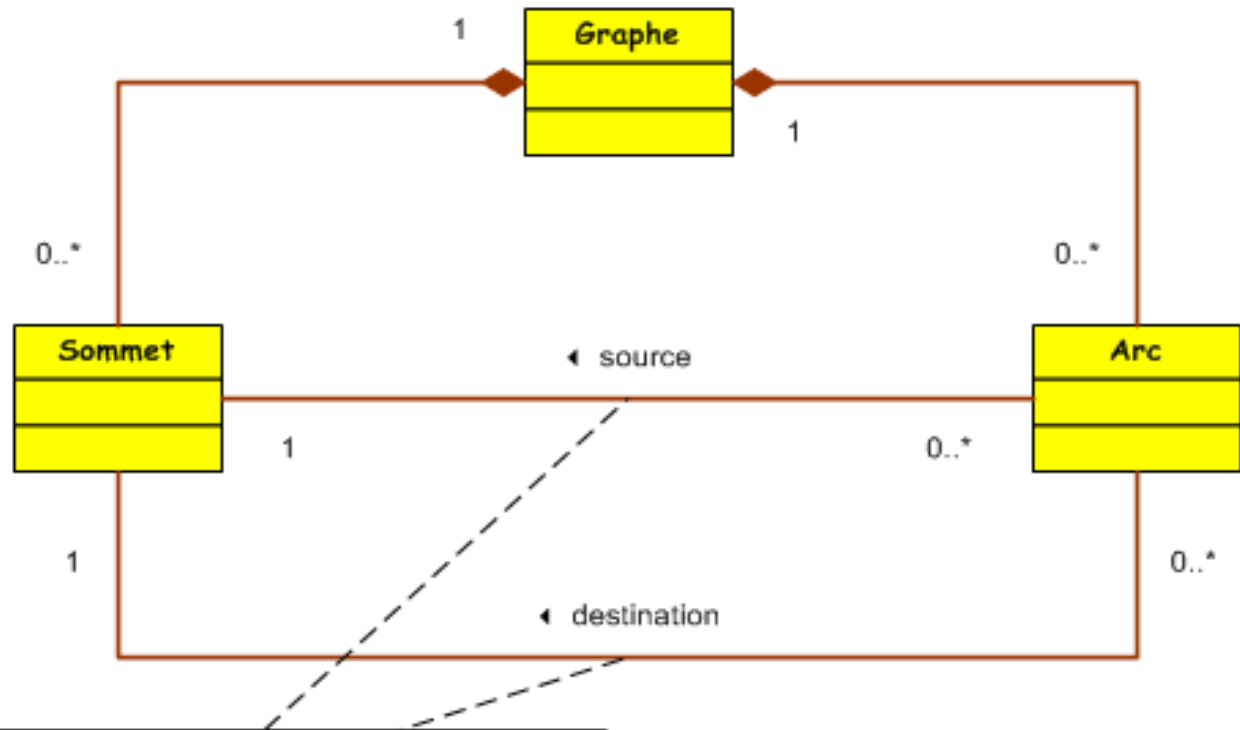
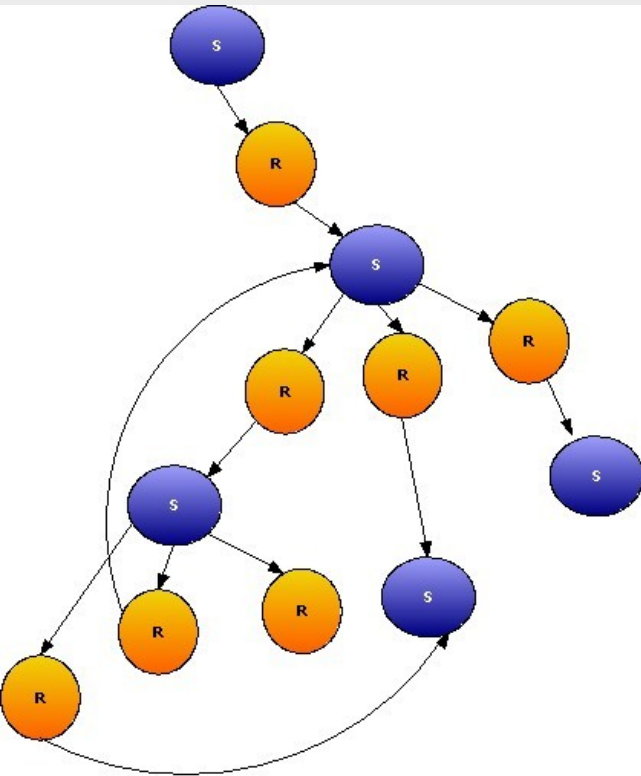




# Exemple : graphe non orienté

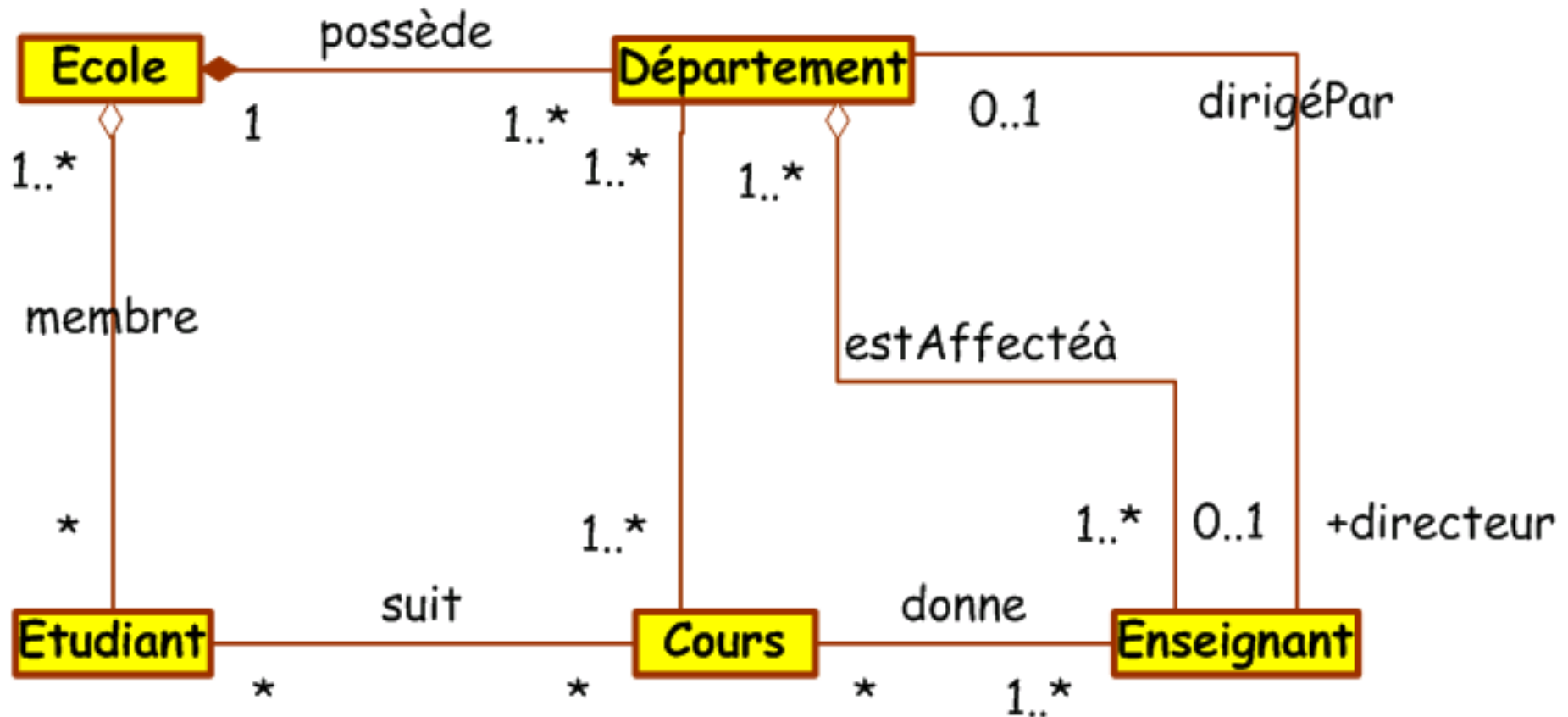


# Exemple : graphe orienté



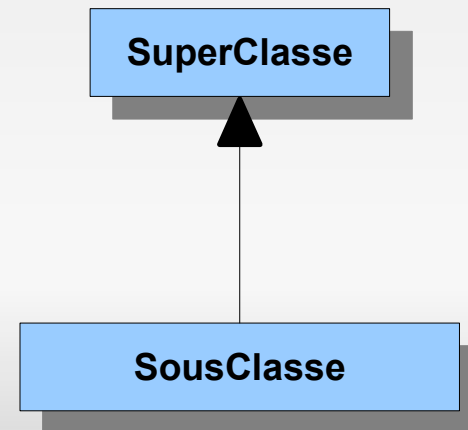
Tout sommet doit être relié au moins à un autre sommet.

# Exemple



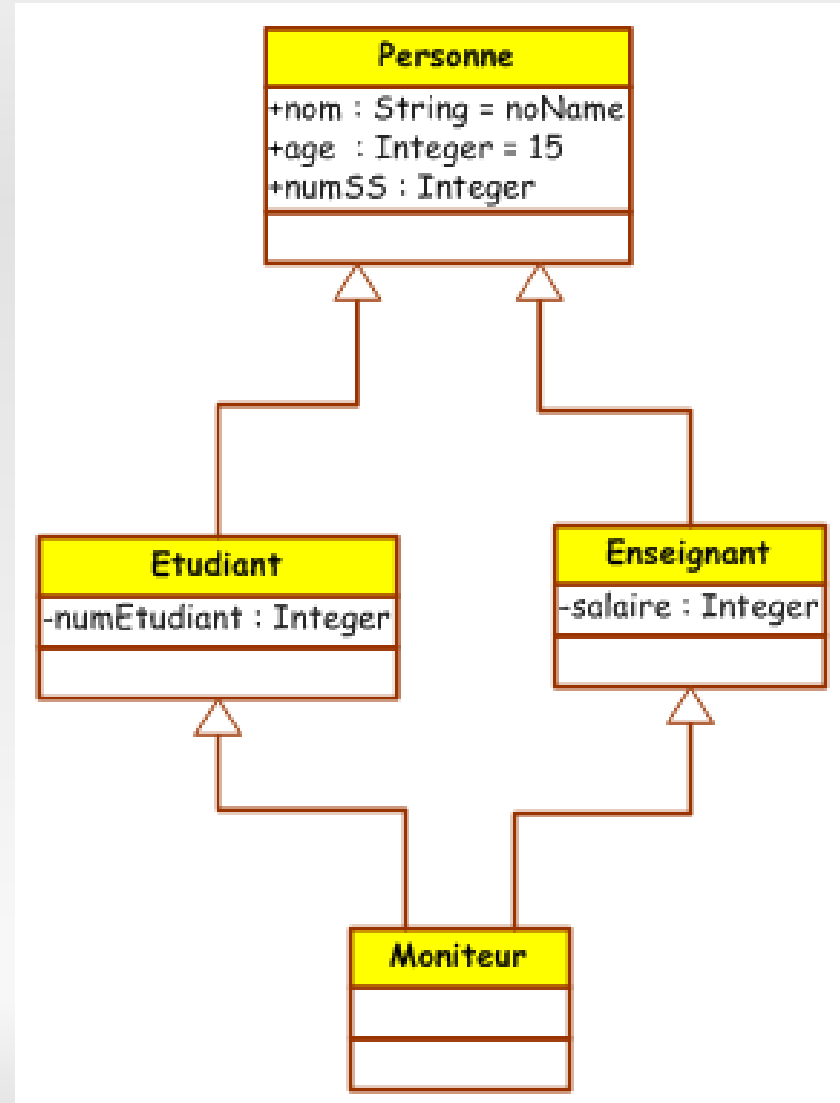
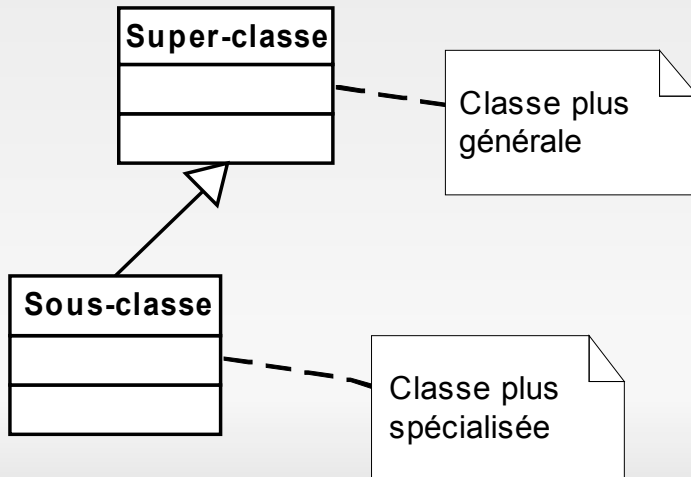
# Relation de Généralisation

- Relation de spécialisation (est-un, is-a-kind-of) entre classes
  - La classe spécialisée (sous-classe) hérite des méthodes des attributs et des relations de la classe-générale (super-classe)
  - Elle peut ajouter des attributs / méthodes
  - Elle peut redéfinir le comportement des méthodes (mais pas les attributs !)



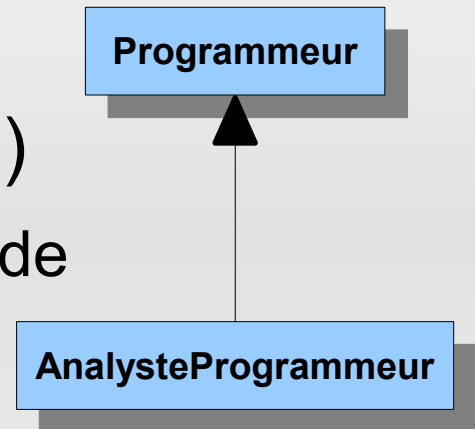
# Hiérarchies de généralisation

Arborescences  
de classes  
d'abstraction  
croissante



# Principe de substituabilité

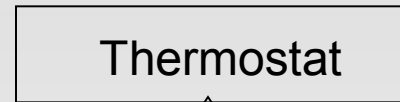
- Barbara Liskov, Jeannette Wing (1993)
  - Si **A** est une sous-classe de **P**, les objets de type **P** dans un programme peuvent être remplacés par des objets de type **A** sans remettre en cause le bon fonctionnement du programme
  - Si je confie une tâche à un **Programmeur**, je peux le remplacer par un **Analyste-programmeur** et la tâche sera aussi bien réalisée (voire mieux !)
  - Un **AnalysteProgrammeur** **est-un** **Programmeur** (le contraire n'est pas vrai !)



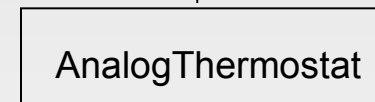
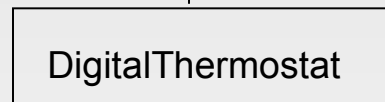
(Prof. Barbara Liskov, M.I.T.)

# Héritage du comportement

Super-classe

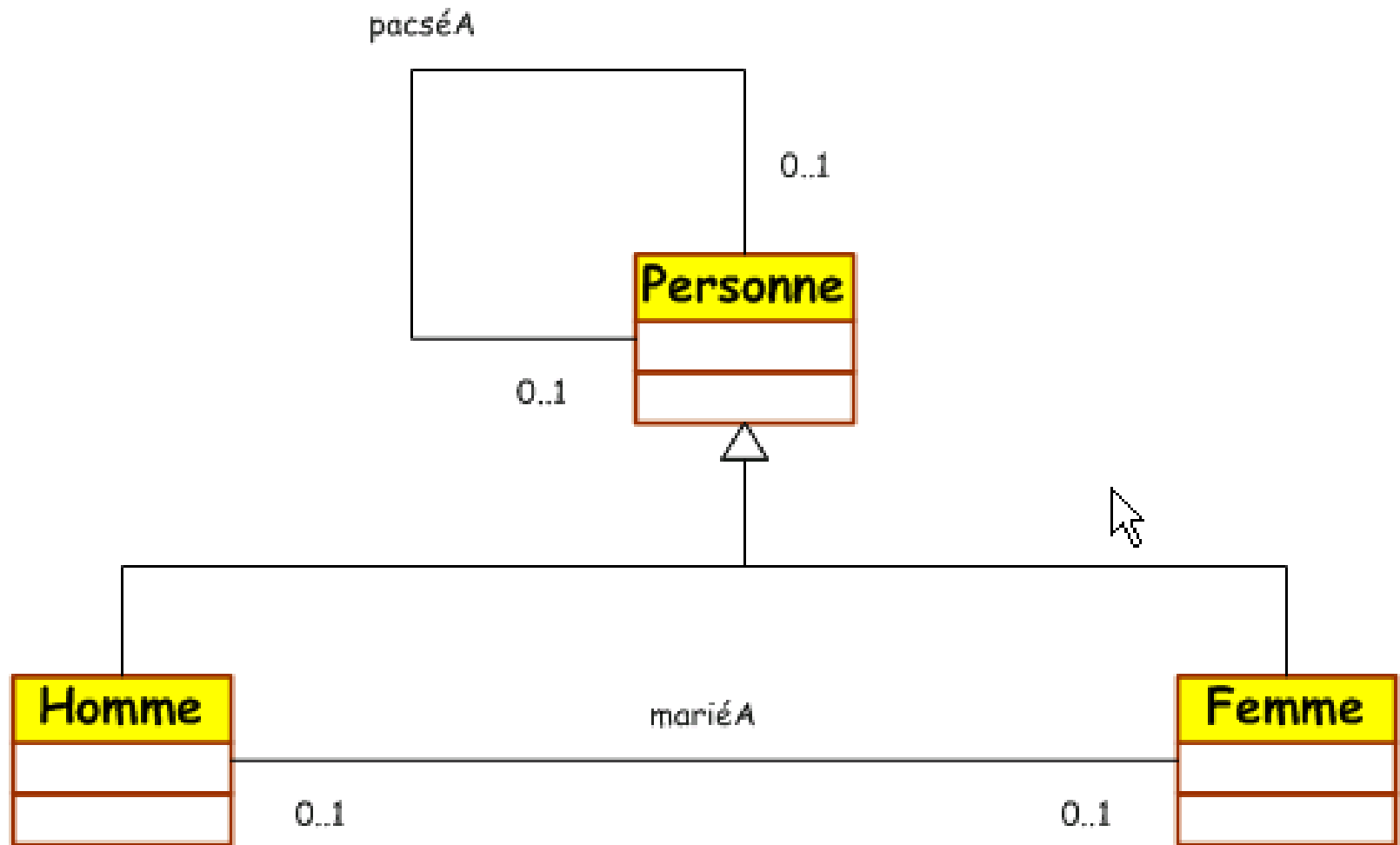


Sous-classe



Anything a superclass can do its subclasses can do.  
These are known as “is-a” relationships  
(*a DigitalThermostat is a Thermostat*)

# Exemple





# Généralisation : notation

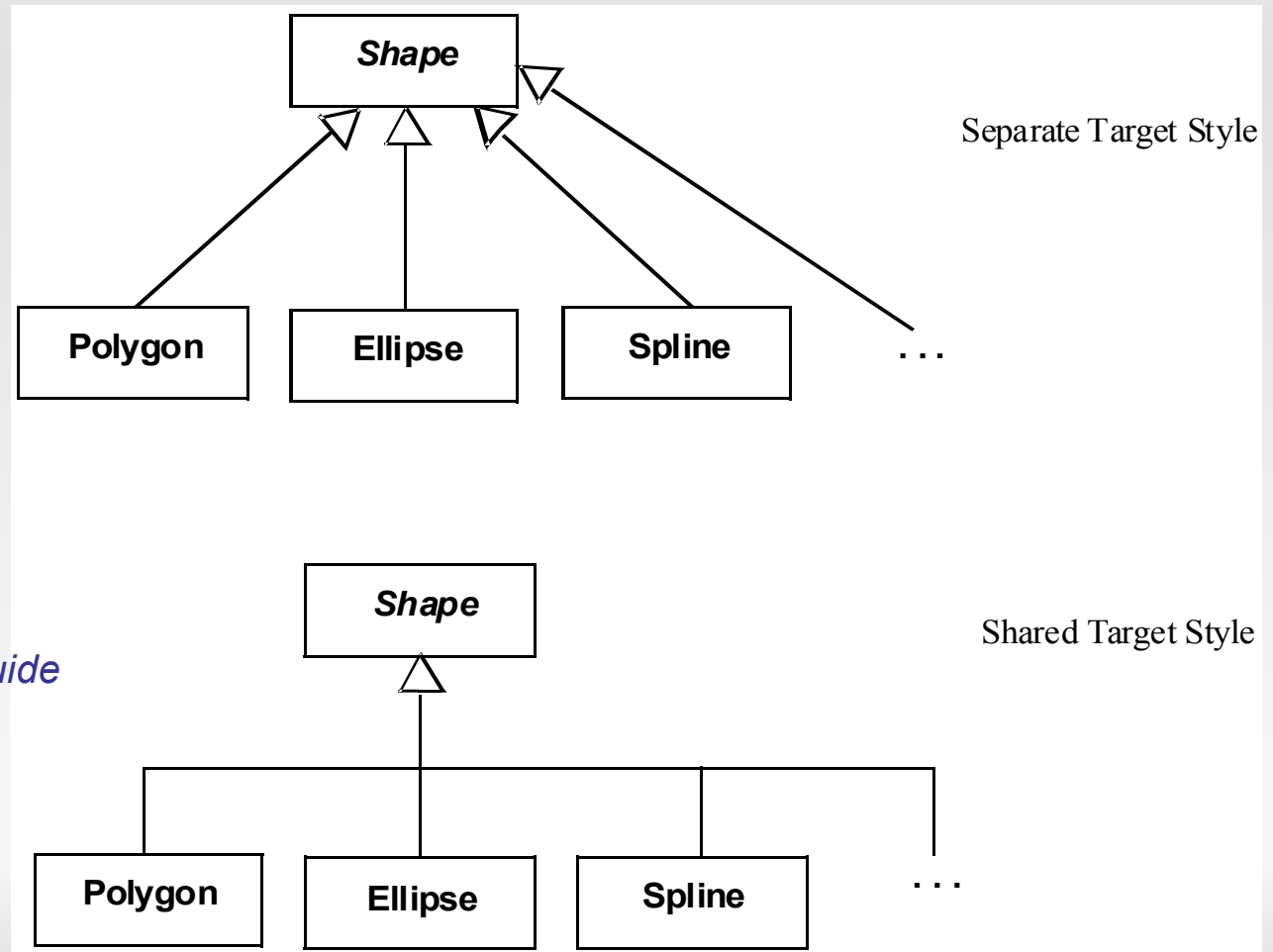
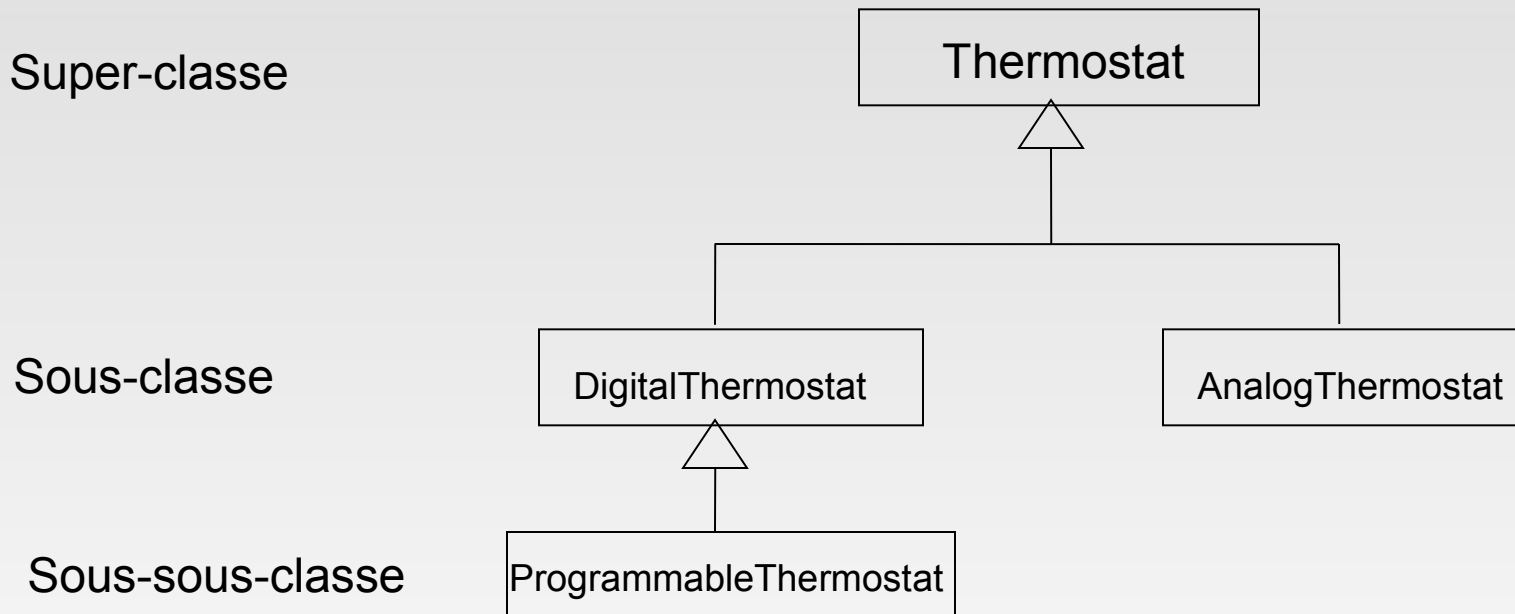


Fig. 3-38, *UML Notation Guide*

# Hiérarchies d'héritage



Inheritance allows us to organize classes into hierarchies based on their inheritance relationships.

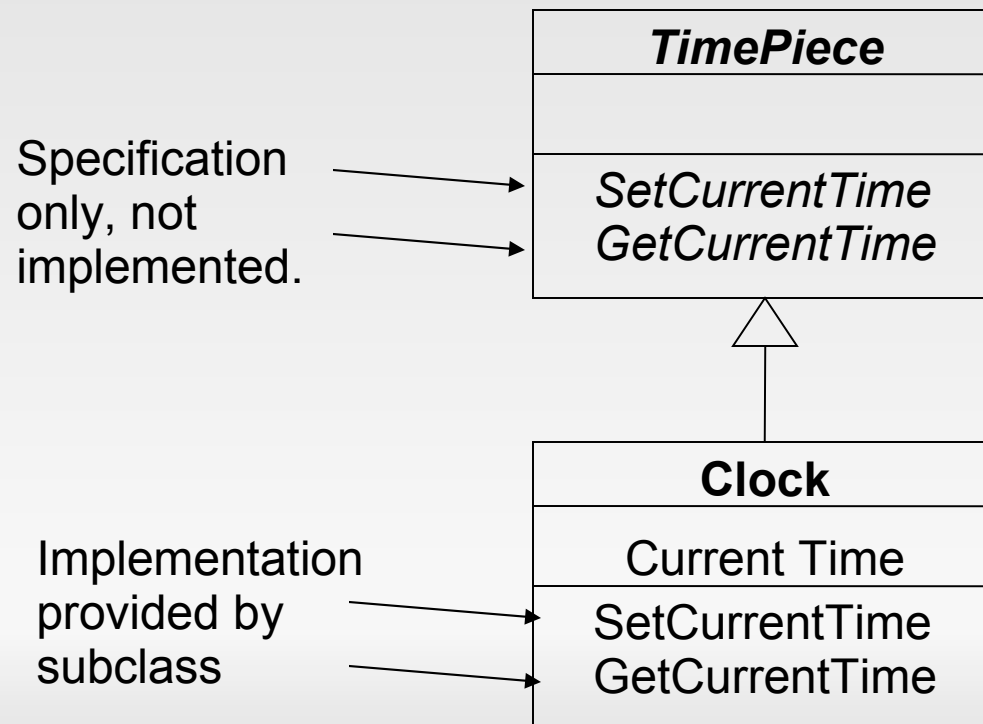
Inheritance is transitive – subclasses inherit state and behavior from superclass.

# Formes d'héritage

- Héritage de spécification
  - La super-classe spécifie un comportement qui sera implémenté dans les sous-classes, mais pas dans la super-classe. Ca permet de garantir que toutes les sous-classes implémentent un comportement similaire
- Héritage de spécialisation
  - La sous-classe est une forme spécialisée de la super-classe, mais respecte ses spécification
- Héritage d'extension
  - La sous-classe ajoute de nouvelles fonctionnalités à celles héritées de la super-classe
- Héritage de limitation
  - La sous-classe restreint l'usage d'un comportement hérité de la super-classe
- Héritage de combinaison
  - La sous-classe hérite des caractéristiques de plusieurs super-classes (héritage multiple)

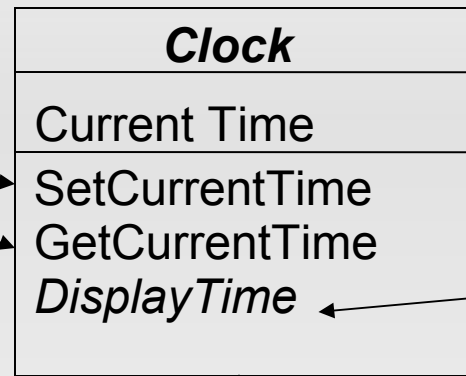
# Héritage de spécification

La super-classe spécifie des comportements requis (méthodes) sans les implémenter. Les sous-classes héritent d'implémentations vides, et doivent fournir une implémentation adéquate. La spécification peut être une classe abstraite ou une interface.

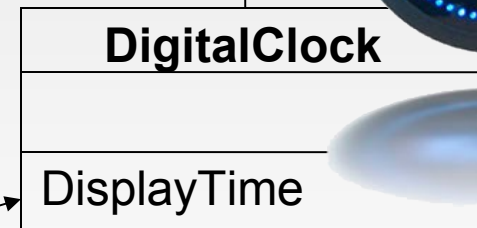
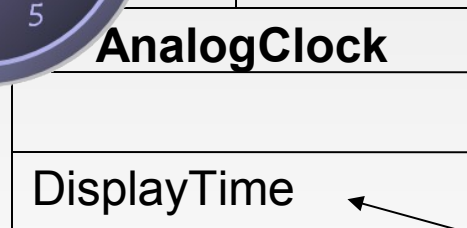


# Héritage de spécialisation

Implémentation fournie par la super-classe, héritée par les sous-classes



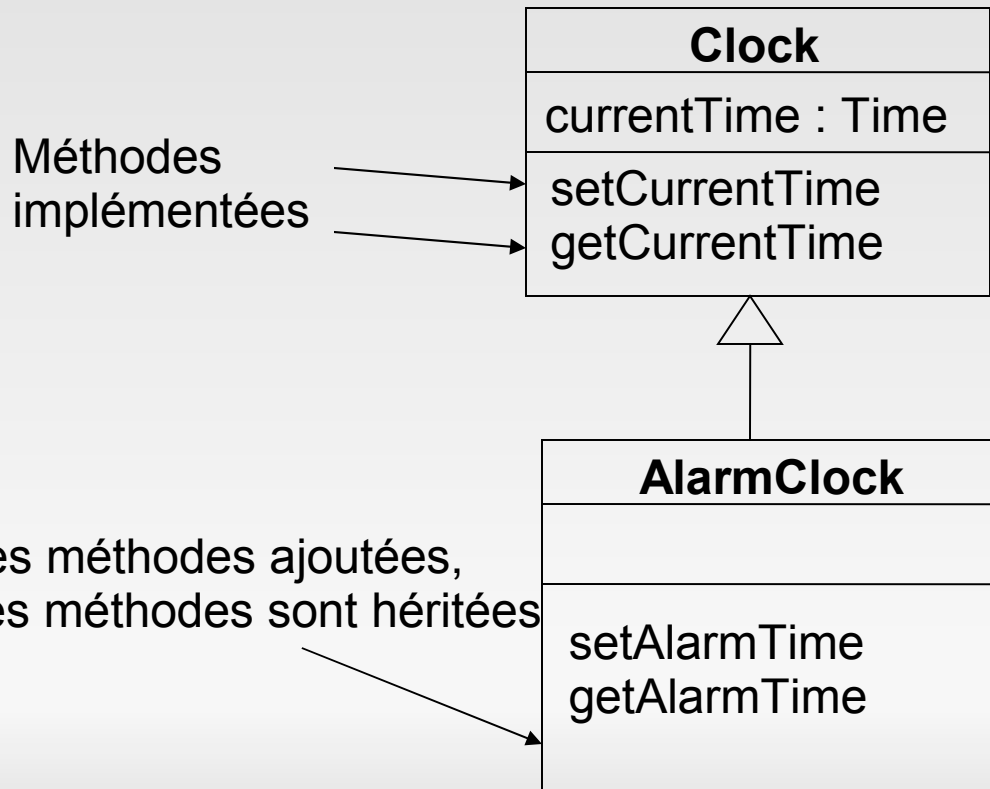
Uniquement la spécification pas d'implémentation  
► classe abstraite



L'implémentation de cette opération est fournie par chacune des sous-classe.  
L'implémentation des autres opérations est héritée

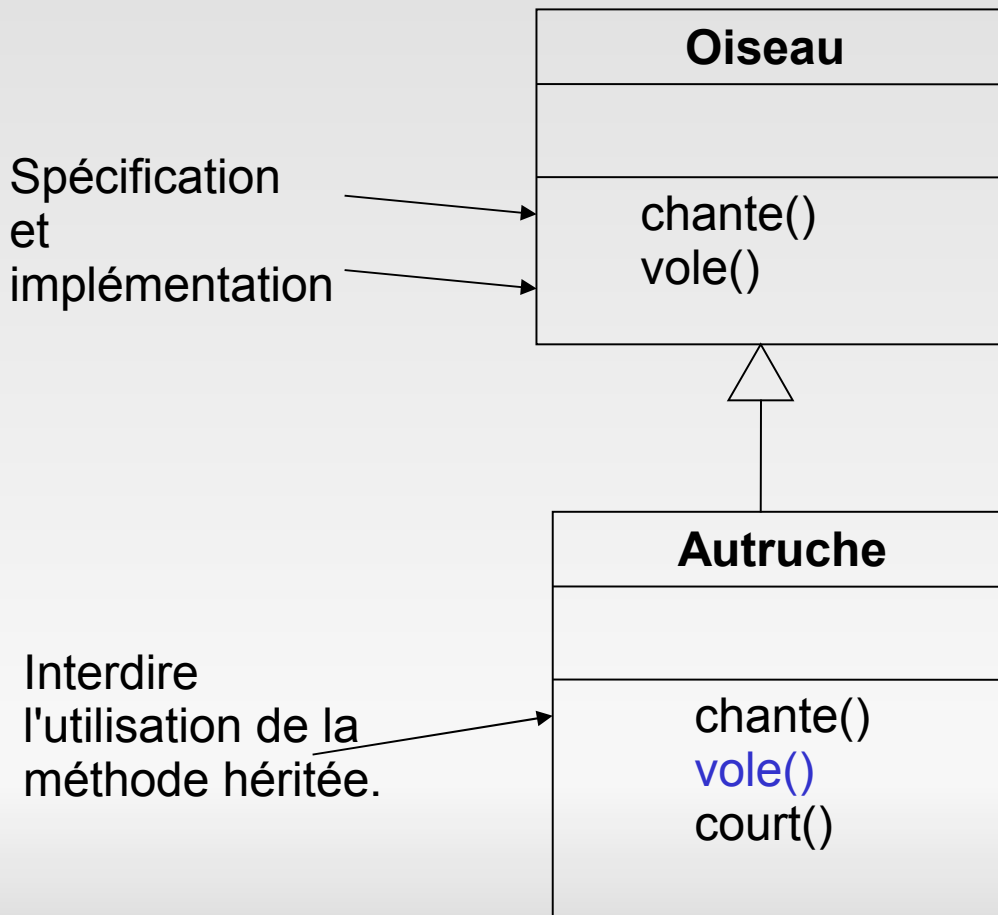
# Héritage d'extension

L'héritage permet de rajouter des nouvelles méthodes dans la sous-classe.



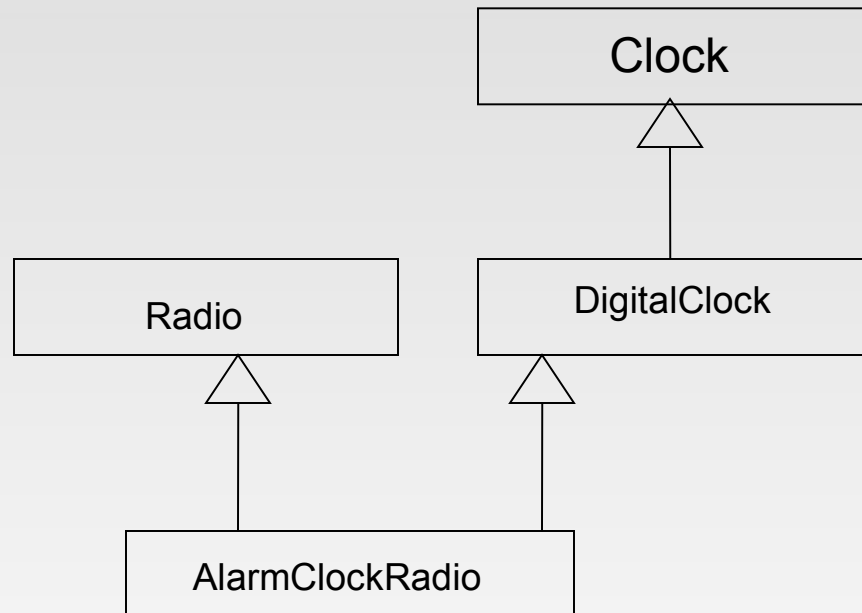
# Example: Limitation

L'héritage peut être utilisé pour limiter l'utilisation de méthodes héritées, par exemple en levant une exception `UnsupportedMethodException`



Principe de substituabilité de Liskov ?

# Héritage multiple



Une classe hérite des caractéristiques de deux ou plusieurs classes différentes



# Héritage multiple

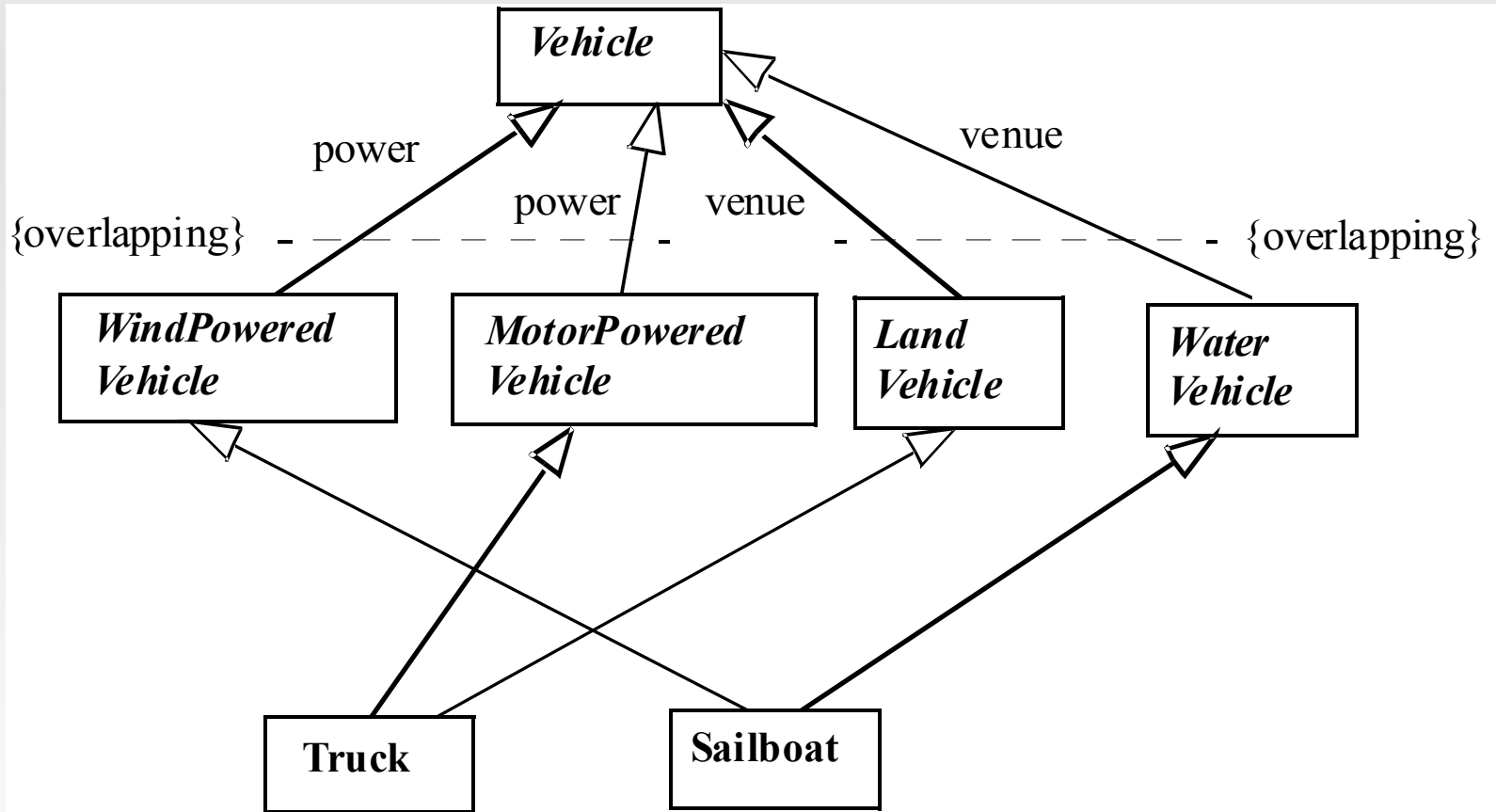


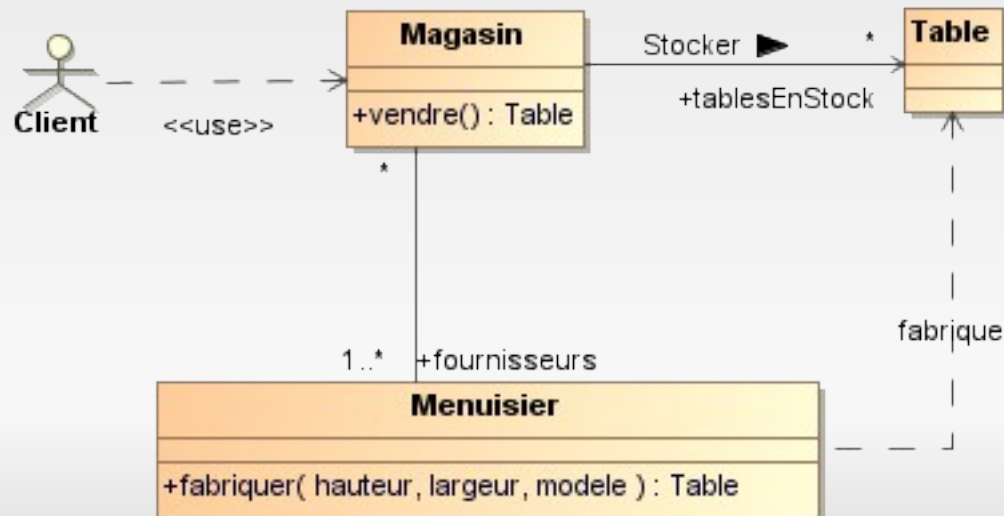
Fig. 3-39, UML Notation Guide

# Dépendance

- Une relation transitoire entre classes, qui n'est pas représentable par association ou composition
  - Un objet sert à créer des objets d'une autre classe (factory object)
  - Un objet utilise un autre sans qu'il fasse partie de ses attributs ou associations
    - Il peut recevoir un objet en tant que paramètre ou valeur de retour d'une méthode
- Une classe dépend d'une autre si elle ne peut pas être réutilisée sans réutiliser également l'autre

# Dépendance : exemple

- Le magasin stocke des tables
- Le menuisier fabrique des tables quand le magasin le lui demande, mais ne les stocke pas (il ne conserve pas de relation avec les tables qu'il a fabriquées)
- On ne peut pas utiliser la classe Menuisier sans disposer également de la classe Table
  - ➔ Il y a une dépendance entre Menuisier et Table



# Exercice: Donner le diagramme de classes correspondant à la formation ISIS en utilisant les concepts suivants

Salle

Bâtiment

Université

Enseignant

Département

Diplôme

Cours

Major

EmploiDuTemps

Etudiant

Rajouter d'autres classes si nécessaire, identifier les attributs associations, associations qualifiées, composition, agrégation, généralisation, cardinalités, rôles

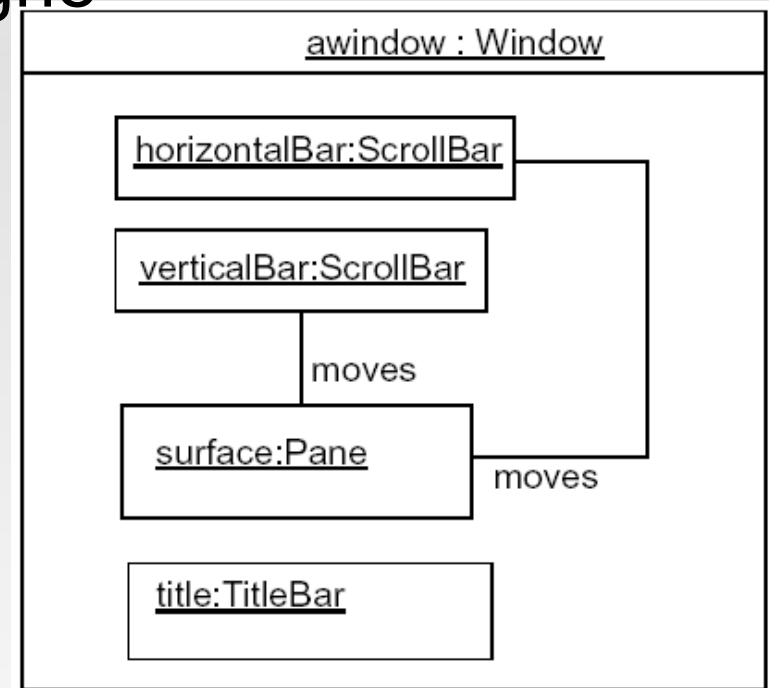
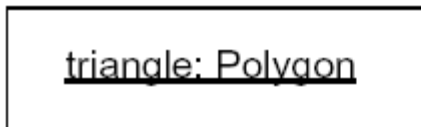
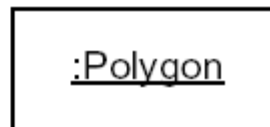
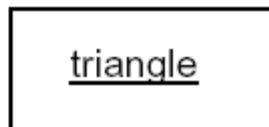
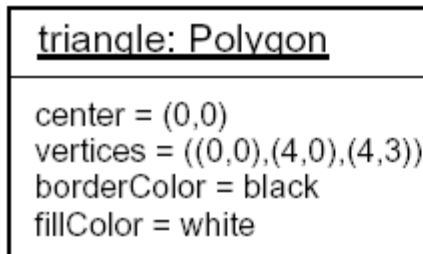
# Diagrammes d'objets

# Diagrammes d'objets

- Les *diagrammes d'objets* représentent un ensemble d'objets et leurs liens. Ce sont des vues statiques des instances des éléments qui apparaissent dans les diagrammes de classes. Ils présentent la vue de conception d'un système, exactement comme les diagrammes de classes, mais à partir de cas réels ou de prototypes.
- Un diagramme d'objet est une instance d'une diagramme de classes.

# Représentation graphique

- Le nom d'un objet est souligné
  - Nom : Classe
  - Nom
  - :Classe

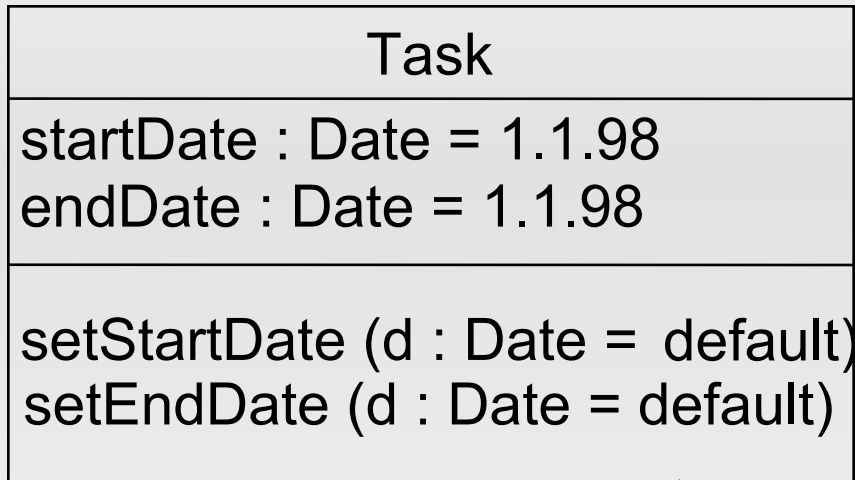


# Suite de l'exercice

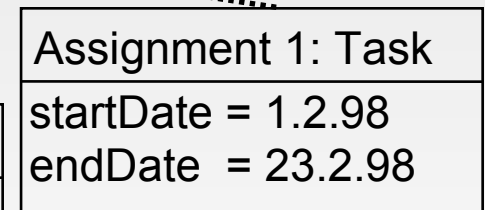
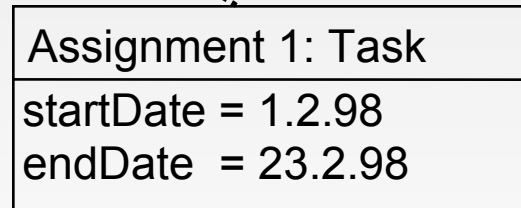
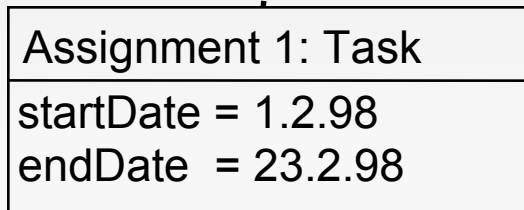
- Créer un diagramme d'objets représentant les informations suivantes :
  - ISIS est un diplôme du département d'ingénierie du CUFR Jean-François Champollion
  - En ISIS 1<sup>o</sup> année, il y a un cours « Réseaux » et un cours «UML»
  - Georges Untel est major de la 1<sup>o</sup> année ISIS en 2007-08
  - Le cours « UML » du lundi 17 mars 2008 (8h15-10h15) est assuré par Rémi Bastide en salle ISIS-1 du bâtiment GCE de l'Université Paul Sabatier



# Objets et classes

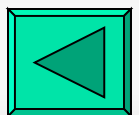
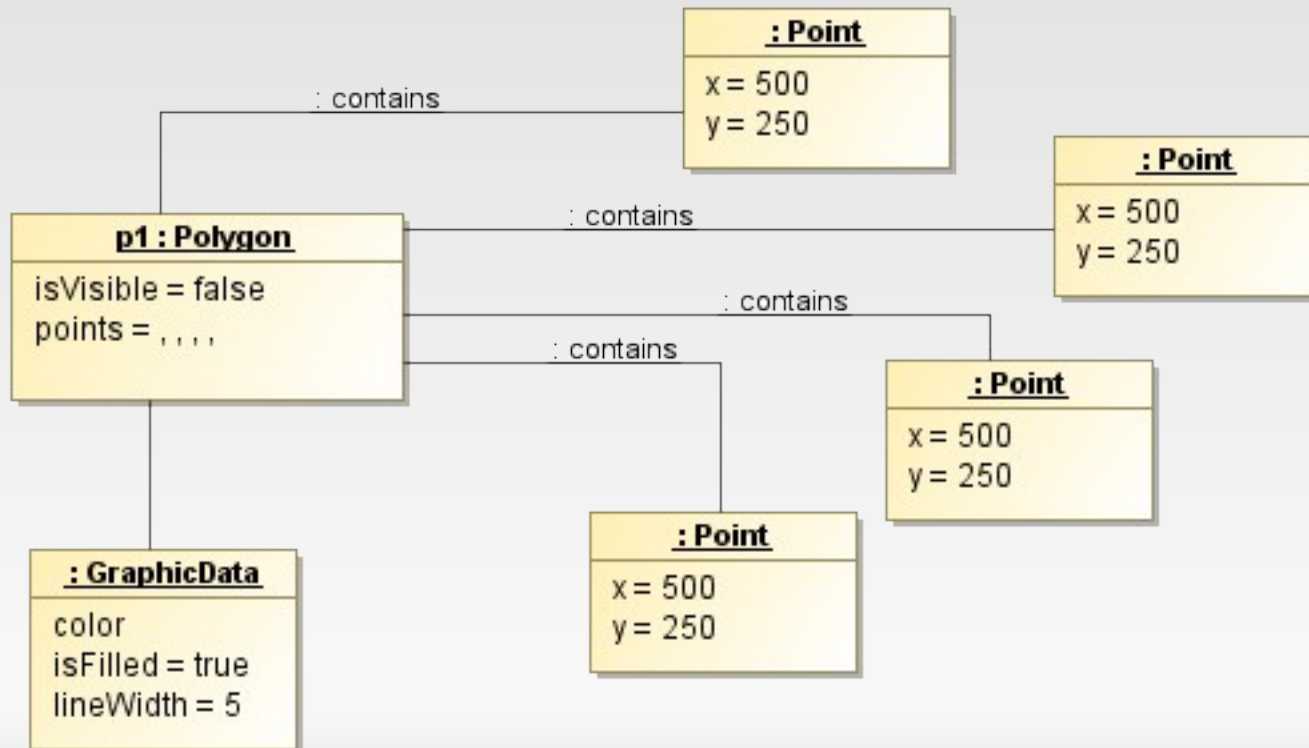


- Les objets montrent
  - Le nom de l'objet
  - Le nom de la classe
  - La valeur des attributs

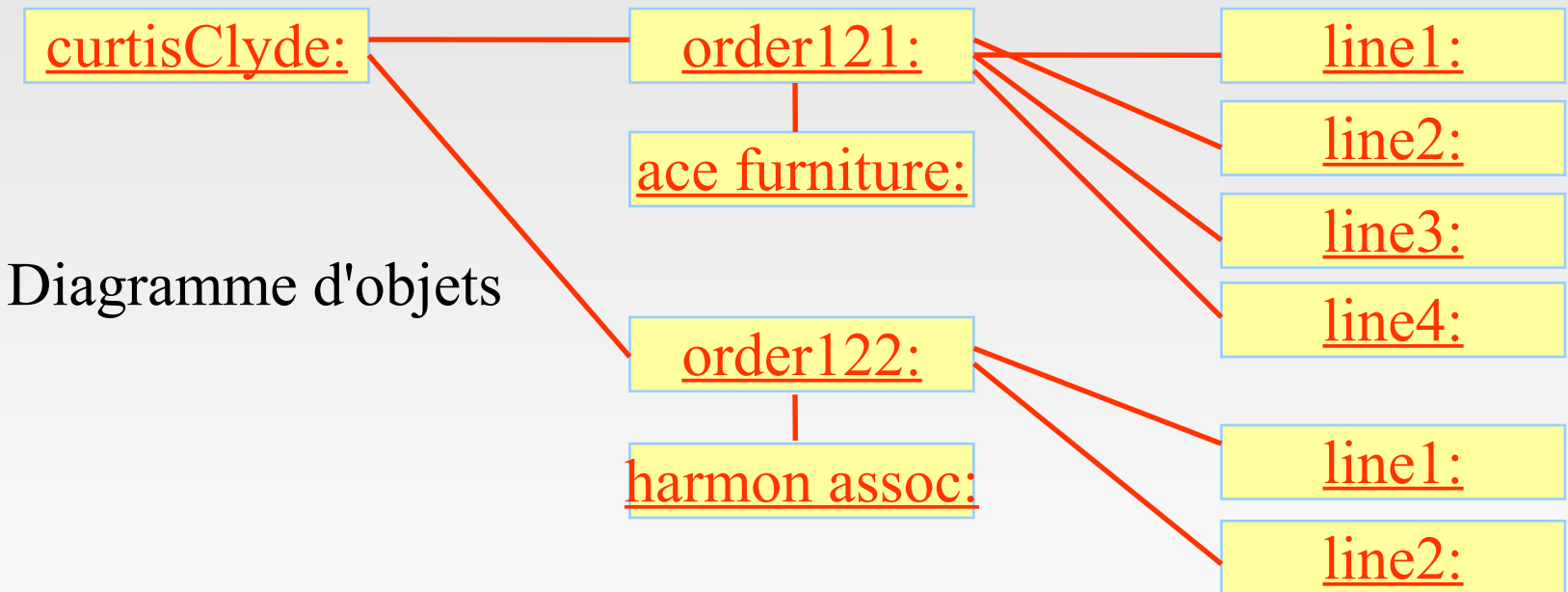
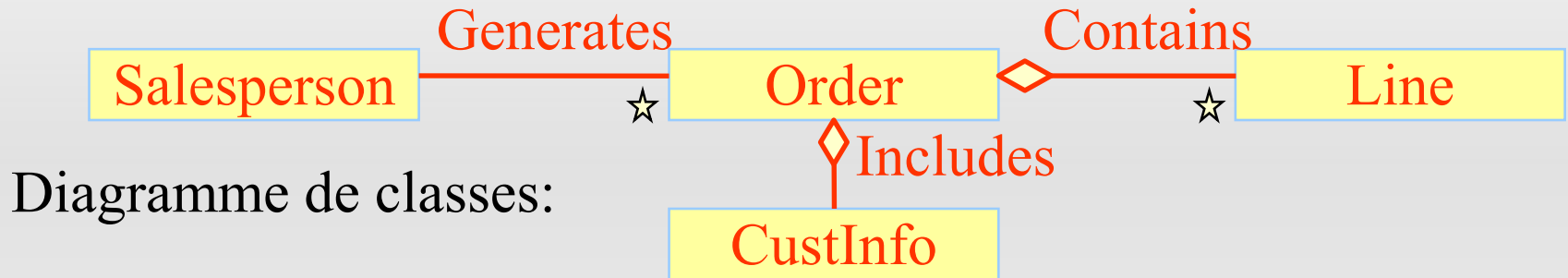


# Diagramme d'objets

- Cf. diagramme de classes « Composition / agrégation »



# Example



# 3 types de diagrammes avec des objets

- Diagrammes d'objets (point de vue statique)
- Diagrammes d'interaction (point de vue dynamique)
  - Diagramme de séquence
  - Diagramme de collaboration

# Modélisation de la dynamique

- Modélisation des interactions entre objets de plusieurs classes
  - Diagrammes d'interaction
    - Diagramme de séquence
    - Diagramme de collaboration
- Modélisation du comportement des objets d'une classe
  - StateCharts

# Interactions

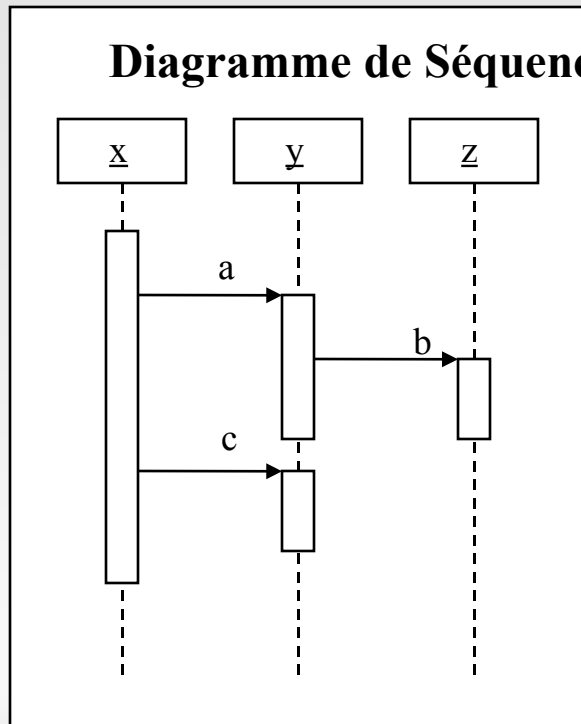
- Interaction:
  - Un ensemble de communications entre instances
    - Appel de méthodes
    - Création /Destruction
- Partiellement ordonné dans le temps

# Diagrammes d'interaction

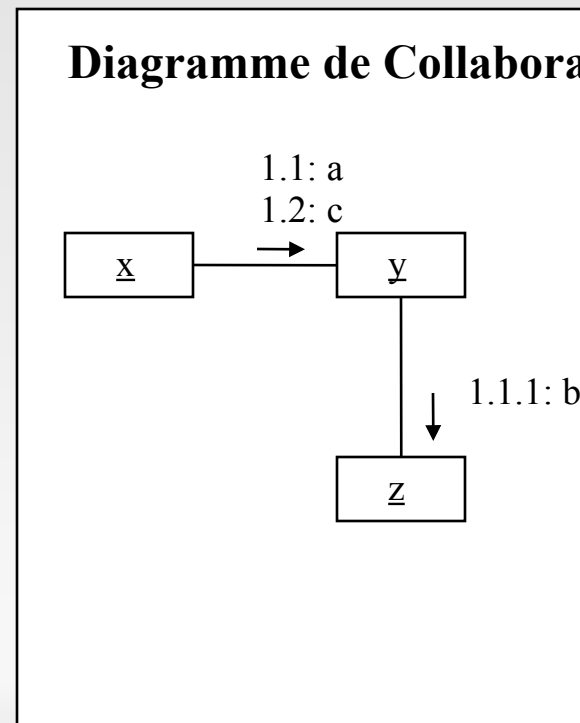
- Montre les interactions entre instances dans un modèle
  - graphe d'instances et de stimuli
  - Instances pré-existantes
  - création et destruction d'instances
- Types
  - Diagramme de séquence (point de vue temporel)
  - Diagramme de collaboration (point de vue structurel)

# Diagrammes d'interaction

**Diagramme de Séquence**

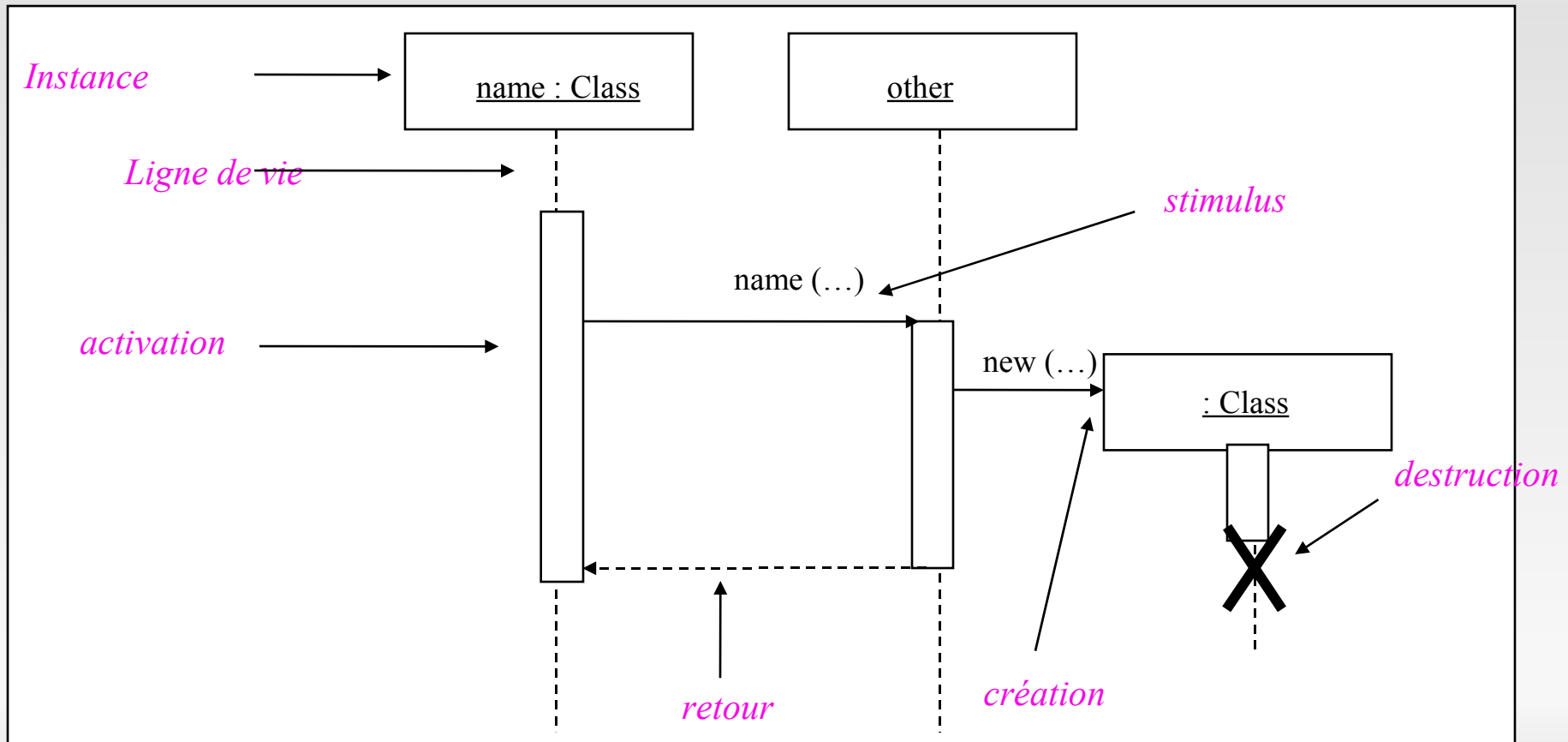


**Diagramme de Collaboration**

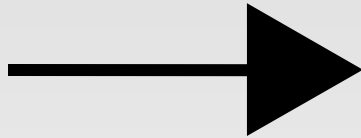




# Diagramme de Séquence



# Types de flèches



Flôt de contrôle  
imbriqué



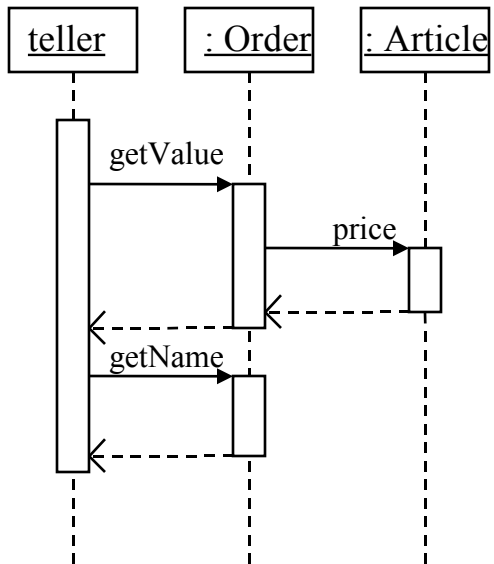
Flôt de contrôle  
asynchrone



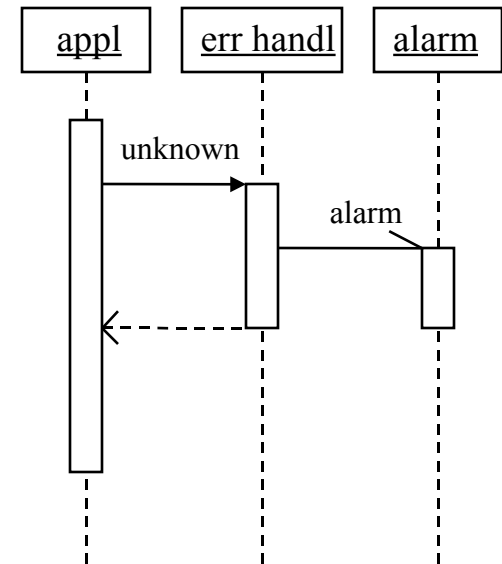
Retour

# Exemples

## Flôt imbriqué



## Flot asynchrone



# CRC Cards

- Classe, Responsabilités, Coopérations
  - Une manière "légère" (papier/crayon) d'identifier les classes d'un système
  - Approche informelle, préalable à une formalisation en UML
  - Conception "centrée sur les responsabilités" plutôt que sur les attributs ou méthodes

# CRC Cards : exemples

