RAPPELS

ACSI = Analyse et Conception des Systèmes d'Information

Plusieurs démarches ont vu le jour :

- Modèle en cascade : analyse, conception, programmation, tests, exploitation

⇒Pour applications simples, ne prend pas en compte les applications complexes que l'on décompose en sous-sytèmes, …

→ Une erreur détectée à la fin oblige à tout reconcevoir => très coûteuse

Modèle en V : amélioration du modèle en cascade, prend en compte les sous-systèmes,
Les retours en arrière en cas d'erreur restent coûteux.
Modèle en Spirale : le développement se fait de manière itérative. On commence par faire un système avec peu de fonctionnalités : le prototype1, qui va suivre les étapes d'analyse, conception, codage, tests, …
Puis on augmente les fonctionnalités pour obtenir le prototype2, etc jusqu'à aboutir à un prototype complet qui représente tout le système (ou presque ….)

En parallèle, des méthode sont apparues : certaines Sont destinées pour une étape du cycle de développement d'un logiciel, d'autres pour 2 ou 3 étapes, …

Par exemple la méthode SADT est efficace pour décrire un logiciel (système) existant, du point de vue fonctionnel, ou de décrire les spécifications fonctionnelles d'un nouveau système, mais est peu efficace pour les autres étapes du cycle de vie.

Parmi ces méthodes :

-Les méthodes cartésiennes (structurées, descendantes, …). Exemple SADT.

- Les méthodes systémiques : considèrent le système dans sa globalité. Modélisation selon 3 axes :

- cycle de vie (étude préalable, analyse, conception, codage, ….),
- cycle d'abstraction (niveau conceptuel (MCD, MCT), niveau organisationnel (MOT, MLD), niveau physique (MPD, MPT) et
- cycle de décision (étapes où sont prises les grandes décisions lors du développement d'un logiciel)

Les méthodes Objets : nouvelle façon d'appréhender les logiciels (systèmes). Tout est objet et les objets d'un système communiquent par envoi de messages.

Chaque objet possèdes un aspect statique (ses caractéristiques, attributs) et un aspect dynamique (ses méthodes ou opérations).
Au début des années 1990, plus de 50 méthodes ont vu le jour : SysPO, AOO/COO, OMT, ..

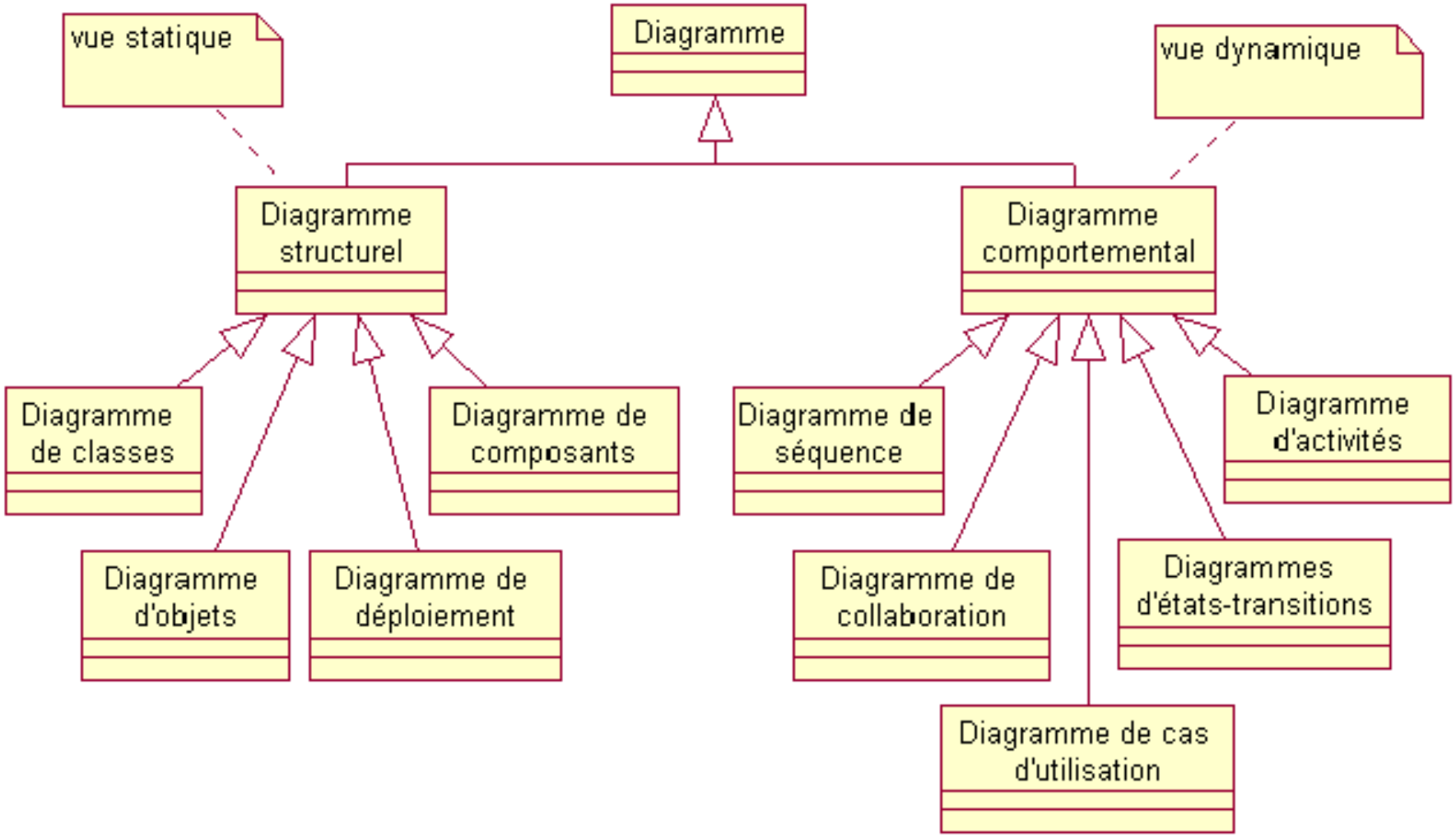
J. Rumbaugh (OMT), G. Booch (OOD) et I. Jacobson (OOSE)

Au milieu des années 1990, trois créateurs de méthodes Objet ont uni leurs efforts pour tenter d'obtenir une méthode Objet « standard ».

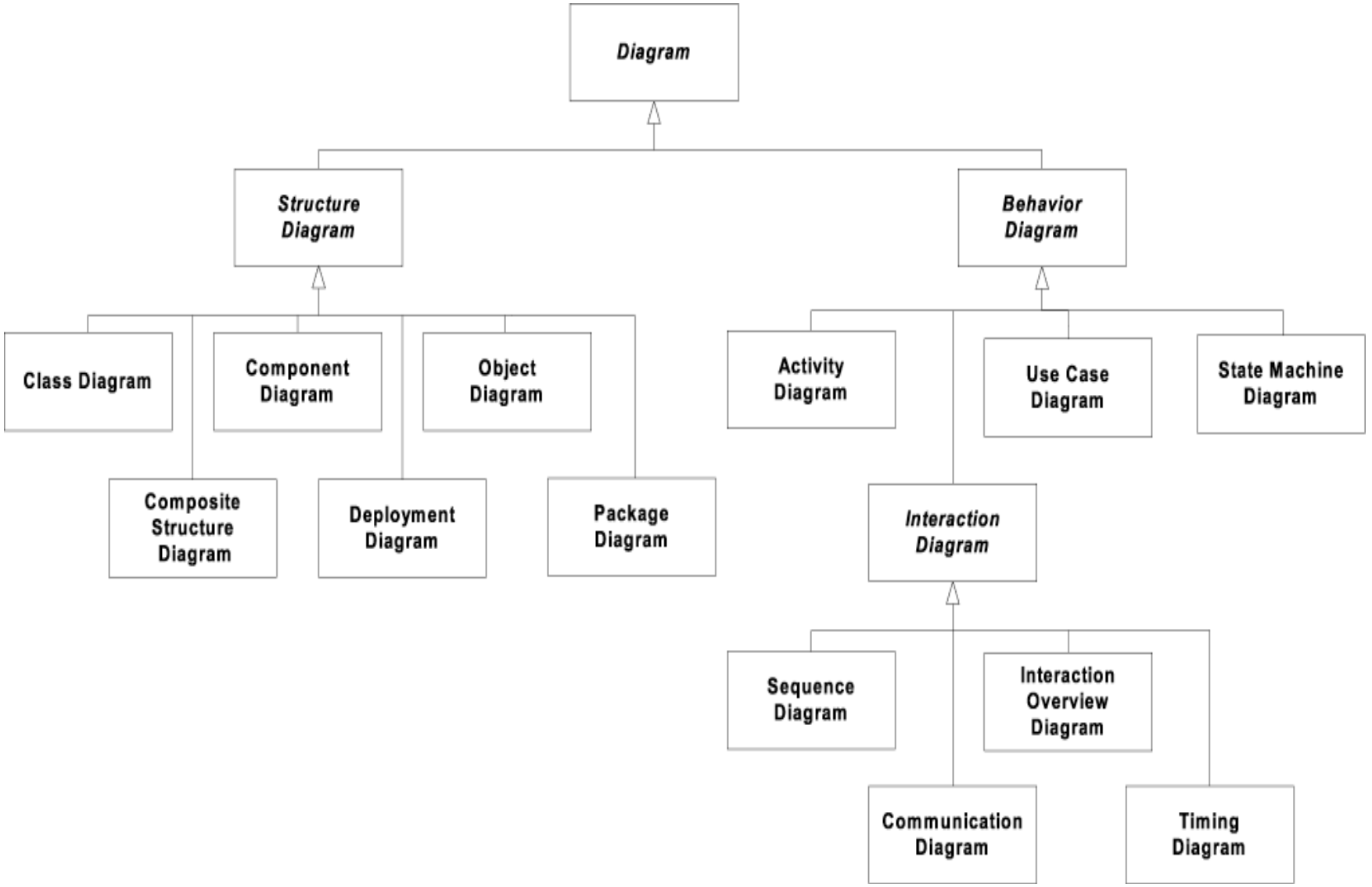
A la fin, ils ne sont pas arrivés à créer une méthode standard, mais plutôt un formalisme (une notation) standard pour toute démarche Objet : c'est la notation UML (Unified Modeling Language), standardisé par l'OMG (Object Management Group), organisme international constitué de professionnels de génie logiciel → plusieurs versions dont 2 principales :

- UML 1.x (1997 →): 9 diagrammes
- UML 2.x (2003-2004) : 13 diagrammes

Les 9 diagrammes (UML 1.x)



Les 13 diagrammes (UML 2.x)



Concepts objets

Les objets :

- ce sont des entités (unités de base) organisées en classes, qui partagent des caractéristiques communes (statiques : attribut ou dynamiques : méthodes ou procédures).

Un objet d'une classe est aussi appelé 'instance' ou 'occurrence' d'une classe.

- ce sont des entités du monde réel, des concepts de l'application ou du domaine étudié.

Illustration :

- Quelles sont les caractéristiques d'une personne? d'un enseignant ?
- Quelles sont les comportements génériques d'une personne? d'un enseignant ?
- Donner des exemples d'instances d'une personne. D'un enseigneant
- Donnez des exemples de sous classes de Personne

Une classe peut être :

1. Publique (+) : Les fonctions de toutes les autres classes peuvent accéder aux données ou aux méthodes d'une classe définie avec le niveau de visibilité *public*. Il s'agit du plus bas niveau de protection des données.

2. Protégée (#) : L’accès aux données est réservé aux fonctions des classes héritières, c'est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées.

3. Privée (-) : L'accès aux données est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé.

NOTE : On peut appliquer les mêmes attributs de visibilité pour les attributs et pour les méthodes : +, #, -

L’encapsulation : le fait que les structures de données et les détails de l’implémentation sont cachés aux autres objets du système.

Note : La seule façon d’accéder à l’état d’un objet est de lui envoyer un message qui va déclencher l’exécution de l’une de ses méthodes.

-L'abstraction

•C'est le fait d'appréhender un objet selon plusieurs points de vue (niveaux). Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise. Par exemple, au niveau conceptuelle, on n'a pas besoin d'exprimer certaines caractéristiques d'un objet liées à l'organisation.

•L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur pour une utilisation donnée

L'héritage :

• Une sous-classe peut hériter d'une classe (la super-classe). Chaque instance de la sous-classe possède ses propres caractéristiques (attributs+méthodes), mais aussi celles de son éventuelle super classe (elle hérite de sa classe-mère).

• L'héritage permet de décrire un type de lien qui unit certains objets.

Illustration : Illustrer l'héritage possible d'une classe Personne. Donner deux instances possibles des [sous]classes.

La modularité : C'est le fait de définir des frontières bien précises pour des fonctionnalités du système (et documentées) à l'intérieur. Les programmes correspondants sont également partitionnés dans l'objectif d'en réduire la complexité.

Le polymorphisme : C'est la possibilité d'utiliser la même expression pour dénoter différentes opérations. Par exemple une fonction à qui on donne deux arguments numériques renvoie leur somme. Si on lui donne deux arguments de type 'String', elle renvoie une chaîne résultat qui est la concaténation des 2 chaînes en entrée

- L'héritage est une forme particulière de polymorphisme caractéristique des systèmes orientés objet.

Langage UML
Méthode sous-jacent

UML n’est pas une méthode, c’est une notation (langage) pour décrire les différentes étapes d’une modélisation Objet d’un système (logiciel).

Mais, la réalisation des différents diagrammes UML dans un certain ordre suggère une méthodologie. En élaborant tous les diagrammes UML, on aura une vision globale et cohérente d’un système.

Pendant la phase d’analyse d’un projet, on recense les besoins des utilisateurs/clients, puis on décrit ce que doit faire le système (QUOI) et non pas COMMENT le faire.

On peut se servir durant cette phase des **diagrammes de cas d'utilisation**, qui décrivent les différentes fonctionnalités du système.

On peut lui associer un **diagramme de séquence et/ou un diagramme de communication** pour décrire la dynamique du système.

Durant cette phase, on peut commencer à élaborer un premier jet du **diagramme de classes** UML qui représente l'aspect statique du système : on recense les classes et les différentes interactions entre elles : ce sont des éléments relativement assez 'stables' dans le temps. On peut avoir recours aux **diagramme d'objets** pour modéliser un cas particulier d'objets du système avec leurs interactions.

Puis, on va doter ces classes de propriétés qui sont les attributs (aspects statique de la classe) et de méthodes (son aspect dynamique).

Au fur et à mesure qu'on passe d'une étape à la suivante dans le cycle de vie d'un système (logiciel) : analyse, conception globale, détaillée, …, on enrichit le diagramme de classes (d'autre attributs/méthodes peuvent apparaître).

Durant l’étape de conception (choix techniques), on décrit COMMENT on réalise les fonctionnalités du système que l’on a recensées dans les cas d’utilisation : c’est le rôle des **diagrammes d’activités et d’états-transitions**.

Le diagramme d’activité va décrire le cheminement des flots de données et de contrôle depuis les entrées du système jusqu’à l’obtention des résultats. Le diagramme d’états décrit les différents changements d’états que subit un objet, instance d’une classe.

Si le système que l'on développe est assez complexe, on aura recours aux **diagrammes de composants, et de paquetages**.

Parfois, on a besoin de prendre en compte des contraintes de temps : on utilise le **diagramme de temps (timing)**. Puis, on utilise le **diagramme de déploiement** pour modéliser l'implantation du système sur les éléments physiques.

Rmq :

1. Le processus de modélisation avec UML ne se fait pas de manière linéaire : on élabore les diagramme dans un certain ordre une fois pour toutes, mais on modélise le système de manière itérative : on revient souvent aux étapes précédentes pour corriger/améliorer, …

2. Il y a 2 ou 3 autres diagrammes pour une utilisation spécifique à certaines applications (structure composite, global d'interaction, …)

Notation UML

UML est une notation issue des formalismes utilisés dans les principales méthodes Objet (OMT, Booch et OOSE). Ie langage UML a été normalisé par l'OMG en 1997, et il est devenu rapidement un standard incontournable.

C’est un outil efficace pour modéliser les concepts Objet : il en en donne une définition plus standardisée et semi-fomelle et apporte la dimension méthodologique qui manquait à l'approche objet.

Trois points de vue de la modélisation

Avec les principaux diagramme d'UML :

- Point de vue fonctionnel : principal : diagramme cas d'utilisation (use case diagram) : ce que le système doit faire (le QUOI). Ce qui entre et ce qui sort du système (pas COMMENT).

- Point de vue statique : principal : diagramme de classes, le plus important dans le modèle.
  Contient les classes d'objets. Une classe = attributs + méthodes.
  -> Eléments assez stables dans le temps.

- Point de vue dynamique : principaux : diagrammes d'interaction, diagramme d'états-transition, diagramme d'activité. Montrent comment les instances des classes communiquent pour réaliser une fonctionnalité.
  c'est pour modéliser ces interactions, qu'UML propose plusieurs diagrammes

Diagramme des cas ’utilisation (Use Case Diagram)

Le développement d’un nouveau système ou l’amélioration d’un système existant doit comporter plusieurs fonctionnalités (répondre aux besoins). Il faut un moyen simple pour que la ou les personnes qui commandent le logiciel expriment les besoins et comprennent ce que les analystes préconisent. C’est le rôle du diagramme d’utilisation.

Les diagrammes de cas d'utilisation :

- permettent de structurer les besoins des utilisateurs et définir les objectifs du système.

- ils ne présentent pas des solutions d'implémentation, mais uniquement les fonctionnalités métiers du système représentant les besoins des utilisateurs.

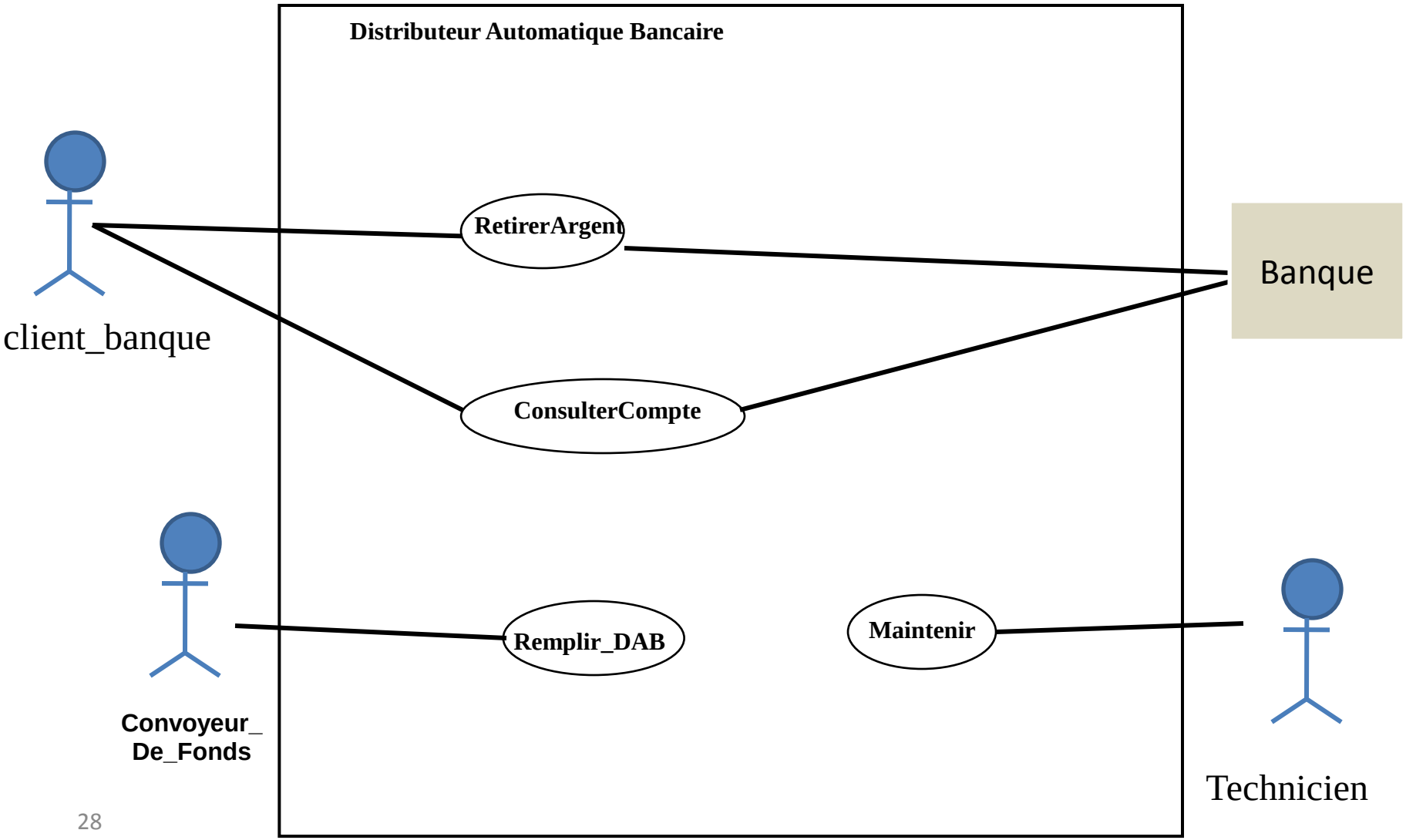
- permettent de clarifier les objectifs du système, pour faciliter et préciser les étapes suivantes de son développement.

Le diagramme des cas d'utilisation est la base du modèle UML.

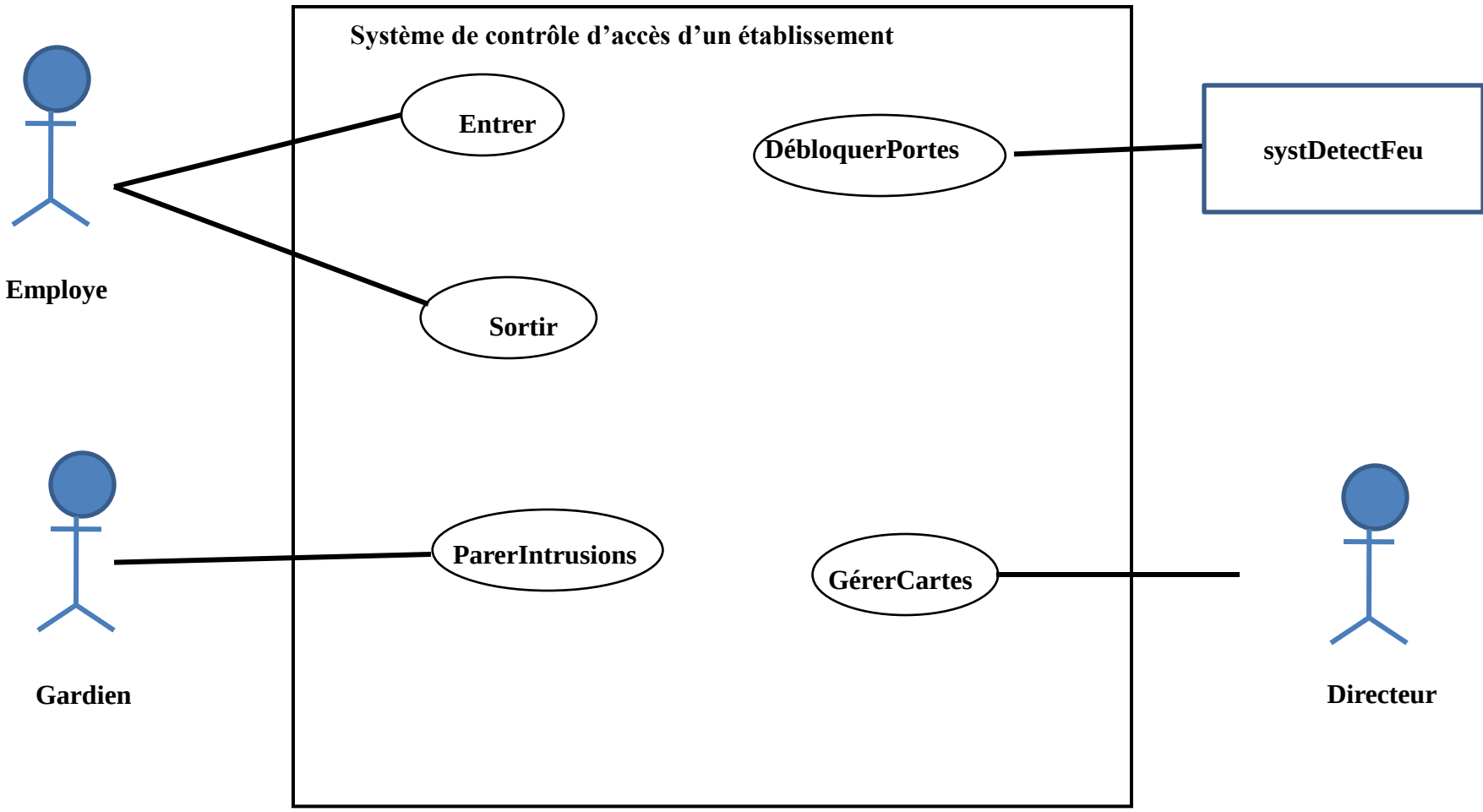
Il donne une vue du système à développer et de son environnement (le acteurs extérieurs qui interagissent avec le système mais qui n'en font pas partie).

On peu ajouter des 'notes' aux cas d'utilisation pour préciser une information ou expliciter un cas
=> Il s'agit d'un texte dans un rectangle :

Un exemple de diagramme de cas d'utilisation : utilisation d'un DAB



Autre exemple de diagramme de cas d'utilisation



Modèle des cas d'utilisation

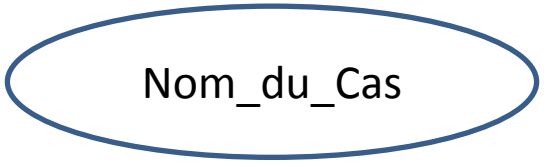
- Un **diagramme de cas d'utilisation** décrit :
  - les **acteurs**
  - les **cas d'utilisation**
  - le **système**

- Un **modèle de cas d'utilisation** peut être formé
  - de plusieurs diagrammes,
  - et/ou de descriptions textuelles.

Cas d'utilisation

- Cas d'utilisation →
  - une manière d'utiliser le système
  - une suite d'interactions entre un acteur et le système

    ex: le guichetier de la banque peut créer un nouveau compte



- Correspond à une fonction perceptible par l'utilisateur
- Permet d'atteindre un but pour un utilisateur
- Doit être suffisamment auto-explicatif.
- Le diagramme de use case n'exprime pas la temporalité (voir diagramme de séquence)

Remarques :

Le diagramme des cas d'utilisation est souvent utilisé dans l'étape d'analyse
Des besoins et pour la compréhension du système existant.

Il permet aux utilisateurs non (informaticiens nécessairement) de comprendre ce que le future système va faire : exprimé par des cas d'utilisation par les analystes.

Les cas d'utilisation servent à recueillir les besoins des Utilisateurs, et donc recenser les fonctionnalités du système (les fonctions qu'il doit effectuer, et non pas la manière dont il va les faire).

Le système

- Le système est un ensemble de cas d'utilisation
- Le système contient :
  - les cas d 'utilisation (use cases), avec leurs interaction

Note : les acteurs sont extérieurs au système.

- Un modèle de cas d 'utilisation permet de définir :
  - les fonctions essentielles du système et ses limites

Les acteurs

- Un Acteur est :
  - humain : Client  ou système : 
  - un élément externe au système et qui interagit lui.
  - un **rôle** qu'un utilisateur joue par rapport au système

    ex: un technicien, un client, un banquier, …
- Une même personne peut jouer plusieurs rôles

  ex: Marc est chef-garagiste, et peut être réparateur
- Plusieurs personnes peuvent jouer un même rôle

  ex: Jean, Alban et Pierre sont des clients d'une même banque
- Exemple d'acteur non humain

  un distributeur automatique bancaire (DAB) peut être vu comme un acteur

Les différents types d'acteurs

- Acteur principal : c'est l'acteur pour lequel le cas d'utilisation est observable. C'est lui qui déclenche le cas d'utilisation.

  ex: client ——— inserer Carte,

  mécanicien ——— réparer

  guichetier ——— ouvrirCompte

- Acteur secondaire : est sollicité pour aider à la réalisation d'un use case ou pour des informations complémentaires.

  ex: systemeBancaire ——————— verifierSolde

Description des acteurs

- On définit un acteur par un identificateur, représentatif de son rôle. Parfois, on donne une brève description textuelle

mécanicien

Un mécanicien est un employé d'un garage. Il est chargé de réceptionner les clients, effectuer des devis de réparation, conseiller les clients.

- La définition d’acteurs permet principalement de lister les cas d’utilisation, en se posant des questions sur son rôle.

Exemple : quel fonctionnalité est attendue :

- d’un mécanicien ? (réparer, faireDevis, conseiller)
- d’un client d’une banque ? (interrogerSolde, retirerLiquide, deposerCheque, …)

Créer un diagramme de cas d'utilisation

- Identifier les acteurs primaires, (puis secondaires) du système, c-à-d rechercher tous les rôles que des acteurs peuvent jouer dans le système.

- Définir les cas d'utilisation (fonctionnalités du système) ainsi que les interactions entre eux et avec quels acteurs.

- Décrire les scénarios d'utilisation : le scénario nominal (normal) et quelques scénarios d'exception (erreurs, alternatives, ..). On décrit ces scénarios de façon textuelle.

Description des cas d'utilisation

- Pour chaque cas d 'utilisation :
  - choisir un identificateur 'représentatif' du cas
  - donner une brève description textuelle
  - citer la fonction réalisée, qui doit être simple à comprendre
  - Ne pas trop détailler (rester à un 'bon ' niveau d'abstraction éviter les actions élémentaires)

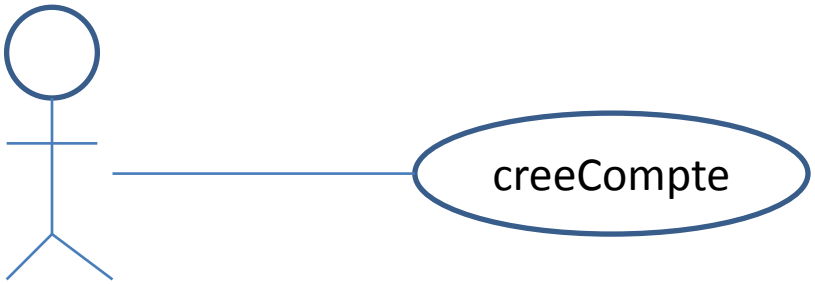
Exemple de description du cas d'utilisation : utilisation d'un DAB

Lorsqu'un *client* veut retirer du liquide, il se présente devant un DAB. Pour cela :

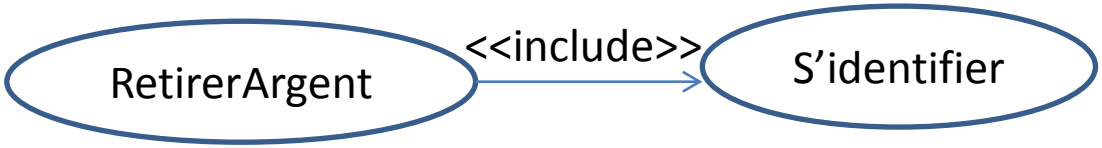
- le *client* insère sa carte bancaire dans le distributeur
- le *système* demande le code secret
- le *client* choisit le montant du retrait sur l'écran affiché
- le *système* vérifie que le compte correspondant à la carte est suffisamment crédité
- si oui, le *système* débite le compte du client de la somme indiquée et présente les billets
- le *client* prend les billets et retire sa carte

Relations possibles entre cas d'utilisation

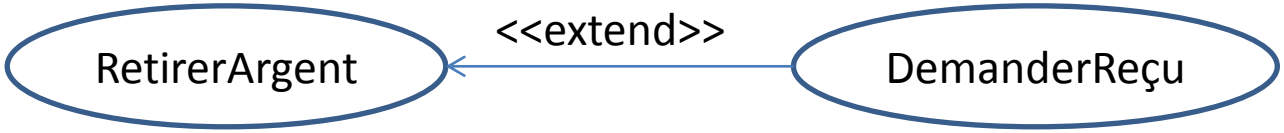
- Relation de communication



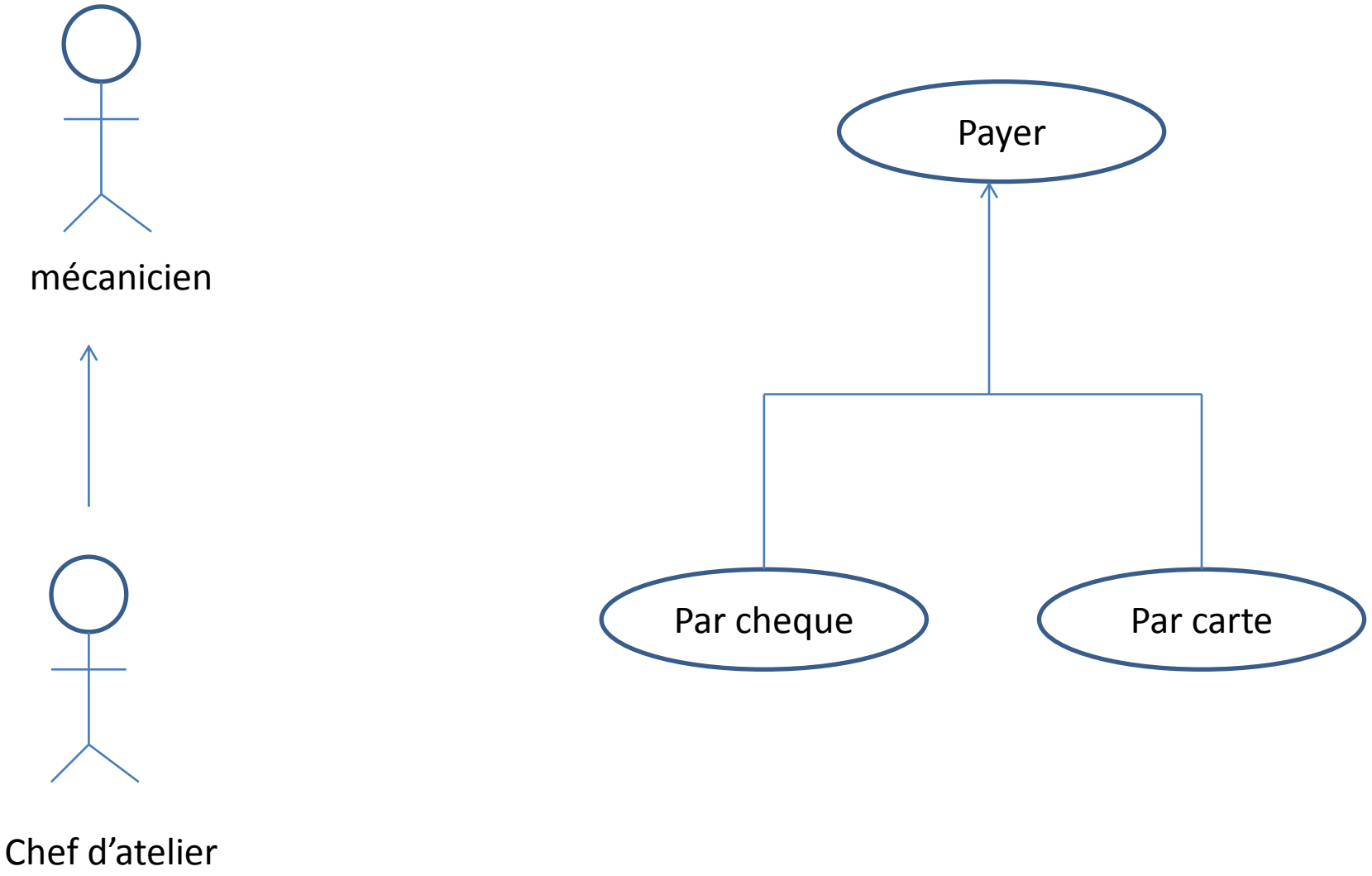
- Relation d'utilisation : <<utilise>> ou <<include>> : obligatoire



- Relation d'extension <<extend>> ou <<etend>> : optionnel



Relation d’héritage



Description du modèle de cas d'utilisation

- Un modèle de cas d'utilisation peut contenir
  - plusieurs diagrammes *use case*
  - et/ou plusieurs descriptions textuelles
- Permet d'avoir une vision globale du système
- …

Description détaillée
de chaque cas d'utilisation

- Chaque cas d 'utilisation doit être décrit en détail
- Commencer par les principaux use case
- La description détaillée 'semi'-formelle
  - langue naturelle la plus structurée possible, vocabulaire précis
  - diagramme d'états
  - ...

Contenu de la description textuelle

- pré-conditions : actions avant le début du cas
- post-conditions : quoi faire à la fin du cas
- Chemin correspondant au déroulement normal
- Variantes possibles et exceptions (cas d’erreurs)
- Interactions entre le système et les acteurs

Exemple : Utilisation d'un DAB cas : RetirerArgent

Précondition :
Le DAB est approvisionné. Il n'est pas panne. Il est en attente d'une opération,

Début : lorsqu'un client introduit sa carte bancaire dans le distributeur.

Fin : lorsque la carte bancaire et les billets sont sortis.

Postcondition :
Si Retrait d'une somme S, alors Somme restant = Somme avant opération – S
Sinon la somme restant = Somme avant opération.

Exemple de description détaillée d'un use case

Déroulement normal :

1. le *client* introduit sa carte bancaire
2. le *système* lit la carte et vérifie si la carte est valide
3. le *système* demande au client de taper son code
4. le *client* tape son code
5. le *système* vérifie que le code correspond à la carte
6. le *client* choisit une opération de retrait
7. le *système* demande le montant à retirer
8. …

Variantes :

(A) *Carte invalide* : au cours de l 'étape (2) si la carte est jugée invalide, le système affiche un message d 'erreur, rejette la carte et le cas d 'utilisation se termine.

(B) *Code erroné* : au cours de l 'étape (5), retour à l'étape (3) trois fois, puis fermeture guichet (avaler carte ou non) ...

Exemple de description détaillée d ’un cas : RetirerArgent

Contraintes non fonctionnelles :

(A) Performance : le système doit réagir dans un délai inférieur à x secondes.

(B) Résistance aux pannes : La transaction en cours doit être annulée « correctement », si une coupure de courant ou une autre défaillance survient et l ’argent ne sera pas distribué. Le système doit pouvoir redémarrer automatiquement dans un état cohérent et sans intervention humaine.

(C) Résistance à la charge : le système doit pouvoir gérer un certain nombre de transactions de retrait simultanément

...

Scénario

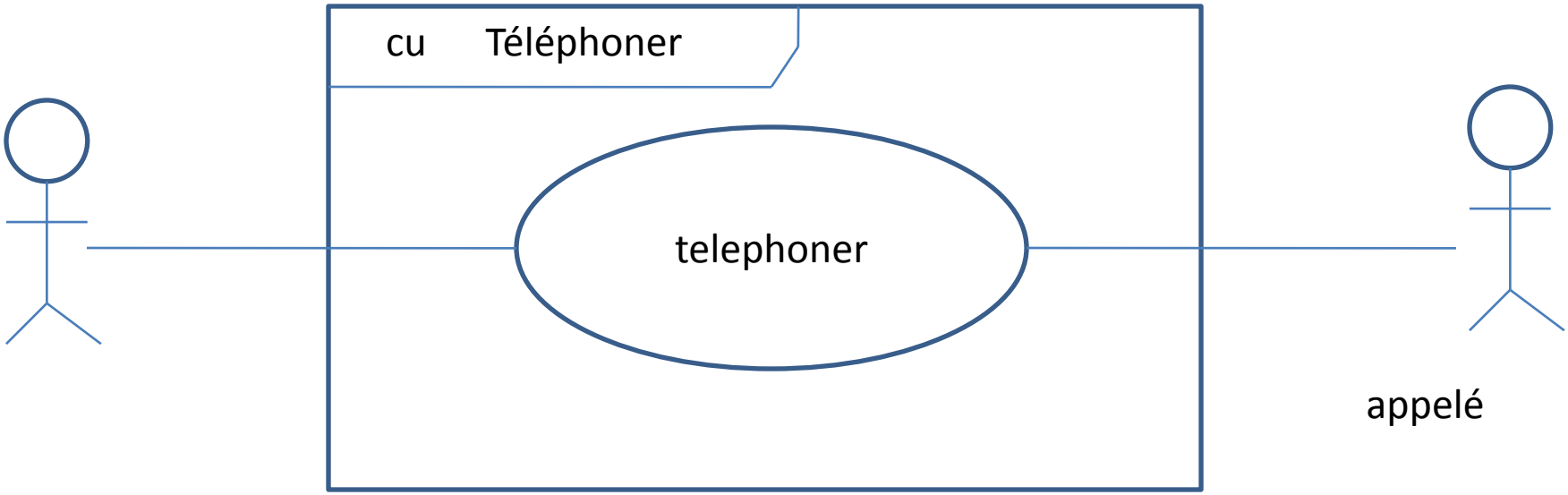
- Pour décrire ou valider un cas : les **scénarii**
- Un scénario est un exemple :
  - une manière particulière d ’utiliser le système …
  - … par une personne particulière …
  - … dans un contexte particulier.
- cas d’utilisation = ensemble de scénarios
- scénario = une exécution particulière d’un cas

Exemple de scénario

SCENARIO 4

1. Paul insère sa carte dans le distributeur numéro d10
2. Le système accepte la carte et lit le numéro de compte
3. Le système demande le code
4. Paul tape ‘ 1234 ’
5. Le système indique que ce n ’est pas le bon code
6. Le système affiche un message et propose de recommencer
7. Paul tape ‘ 6622’ (si erreur, aller à 3 si nb tentatives < 3, sinon exit)
8. Le système affiche que le code est correct
9. Le système demande le montant du retrait
10. Paul tape 70
11. Le système vérifie si le compte est approvisionné
12. ...

Autre exemple



Scénario nominal

1. l'appelant décroche le téléphone
2. la tonalité se déclenche
3. l'appelant tape un chiffre
4. la tonalité s'arrête
5. l'appelant tape 10 chiffres
6. le téléphone appelé commence à sonner, en même temps que se fait entendre cette sonnerie chez l'appelant
7. l'appelé décroche en un nombre de sonneries <10
8. le téléphone de l'appelé cesse de sonner et la tonalité de sonnerie cesse dans l'appelant
9. les téléphones sont connectés. La conversation commence
10. à la fin, l'un des 2 raccroche le téléphone, la connexion est rompue
11. L'appelant raccroche le téléphone

Diagrammes d'interaction :

→ modéliser les interactions entre objets (instances, occurrences), dans le temps.

Deux principaux :

- Diagramme de séquences
- Diagramme de communication

Diagrammes de séquences (1)

- Exprime la temporalité (déroulement dans le temps, représente des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages.)

- Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation, pour décrire un scénario

- L'ordre d'envoi des messages est déterminé par leurs positions sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe.

Diagrammes de séquences (2)

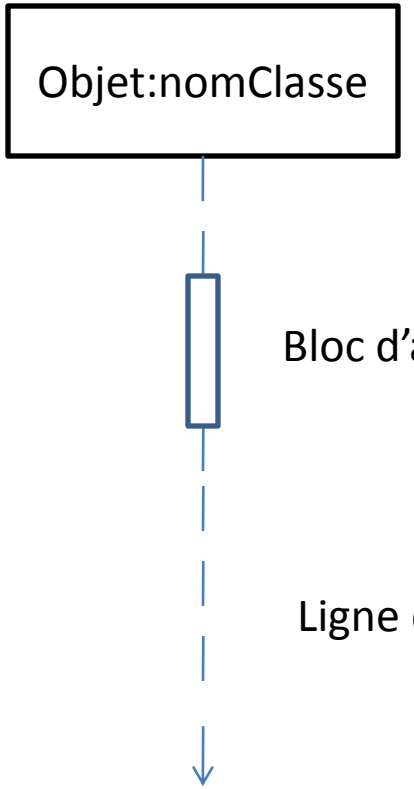
- Diagramme de séquences :
  - L'une des notations UML, une notation générale
  - Peut être utilisée dans de nombreux contextes
  - Permet de décrire une séquence des messages échangés entre différents objets
  - Différents niveaux de détails
- Concepts : L'objet (= instance d'une classe)

  La ligne de vie : verticale, en pointillé

  L'objet en exécution (rectangle)

  Message échangé (sur une flèche horizontale) : → souvent = méthode de l'objet récepteur

Syntaxe graphique



ou acteur humain

Bloc d'activation (exécution)

Ligne de vie (dans le temps)

Rmq : Le nom de la classe est optionnel

-Informations contenues dans un diagramme de séquence : messages échangés entre des objets, présentés dans un ordre chronologique.

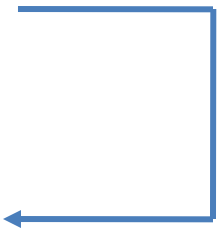
- Contrairement au diagramme de communication, on représente le temps explicitement par une dimension (la dimension verticale) et s'écoule de haut en bas, sur la **ligne de vie**.

- Ligne de vie = depuis la création de l'objet jusqu'à sa destruction (fin) : ligne verticale en pointillé, terminée.

Plusieurs types de messages :

- envoi d'un signal, d'un message
- invocation d'une opération
- création d'un objet
- destruction d'un objet
- message de retour : retourne une valeur à l'appelant

Rmq :Un objet peut s'envoyer un message pour invoquer une méthode locale



2 types de messages : asynchrone et synchrone

Message asynchrone

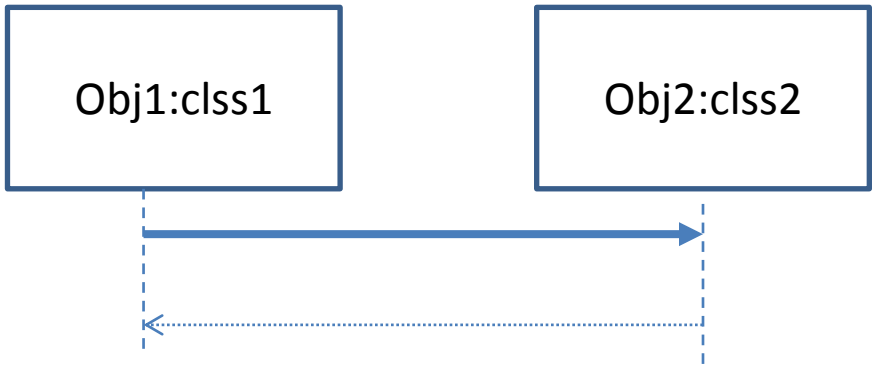
L'émetteur n'attend pas de réponse. Exemple : un signal (une interruption ou un évènement quelconque). Ex. envoyerLettre()

Graphiquement :

Message synchrone

L'émetteur attend la réponse. Exemple : l'invocation d'une méthode (opération) -> le plus utilisé en programmation objet. L'invocation peut être asynchrone ou synchrone. Dans la pratique, la plus part des invocations sont synchrones, c-à-d que l'émetteur reste alors bloqué pendant la durée de l'exécution de l'opération.

Graphiquement :



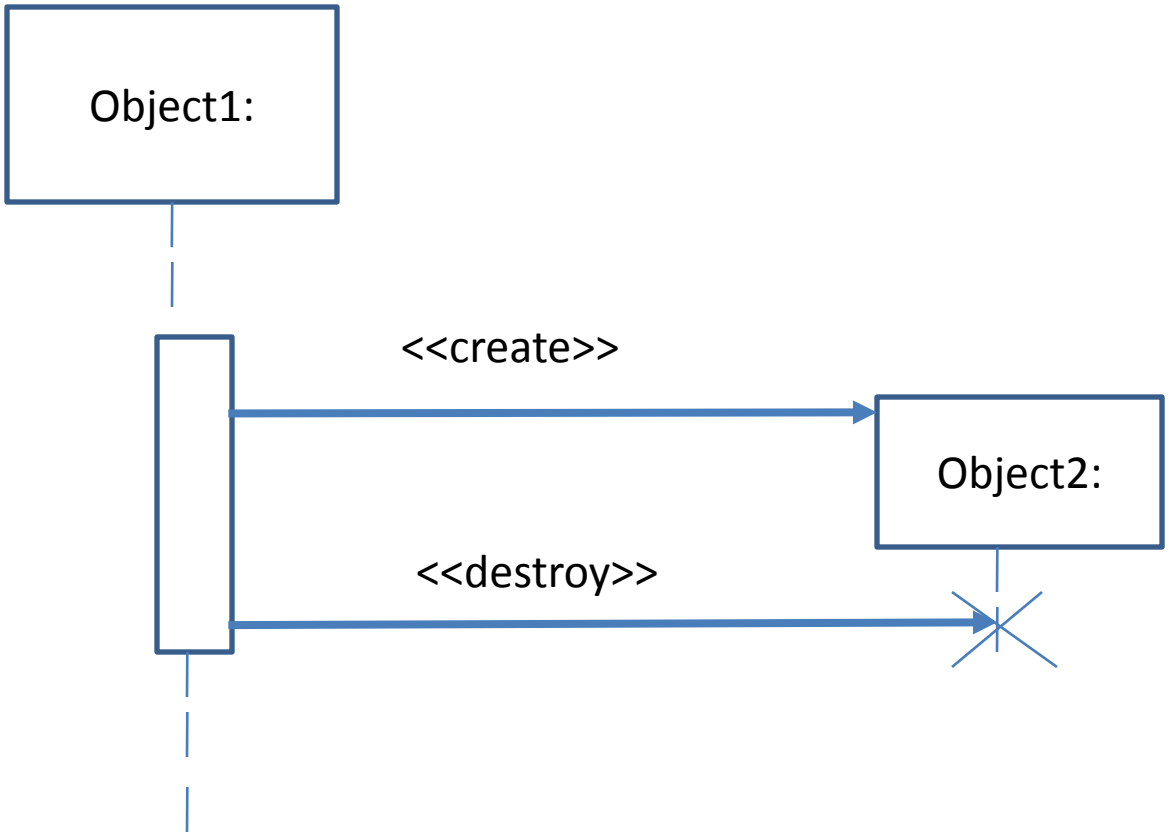
Parfois, ce message est suivi d'une réponse (une flèche en pointillé). Ex. DemanderSomme() message envoyé

TaperSomme (montant) message de retour

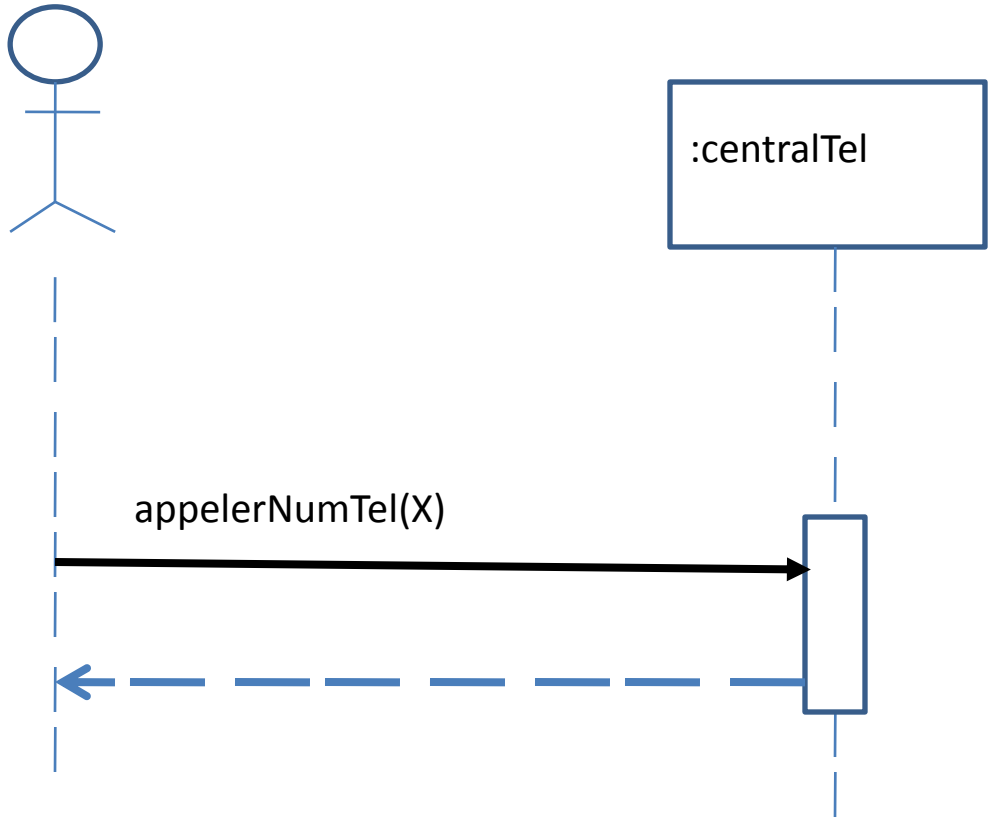
Synchronisation

- La synchronisation est le mode par défaut de transmission des messages (message d'appel, de retour, de création, de destruction)
- Pendant l'envoi et l'exécution par l'appelé, l'appelant est suspendu.
- Le contrôle est rendu à l'appelant à la fin de l'opération.

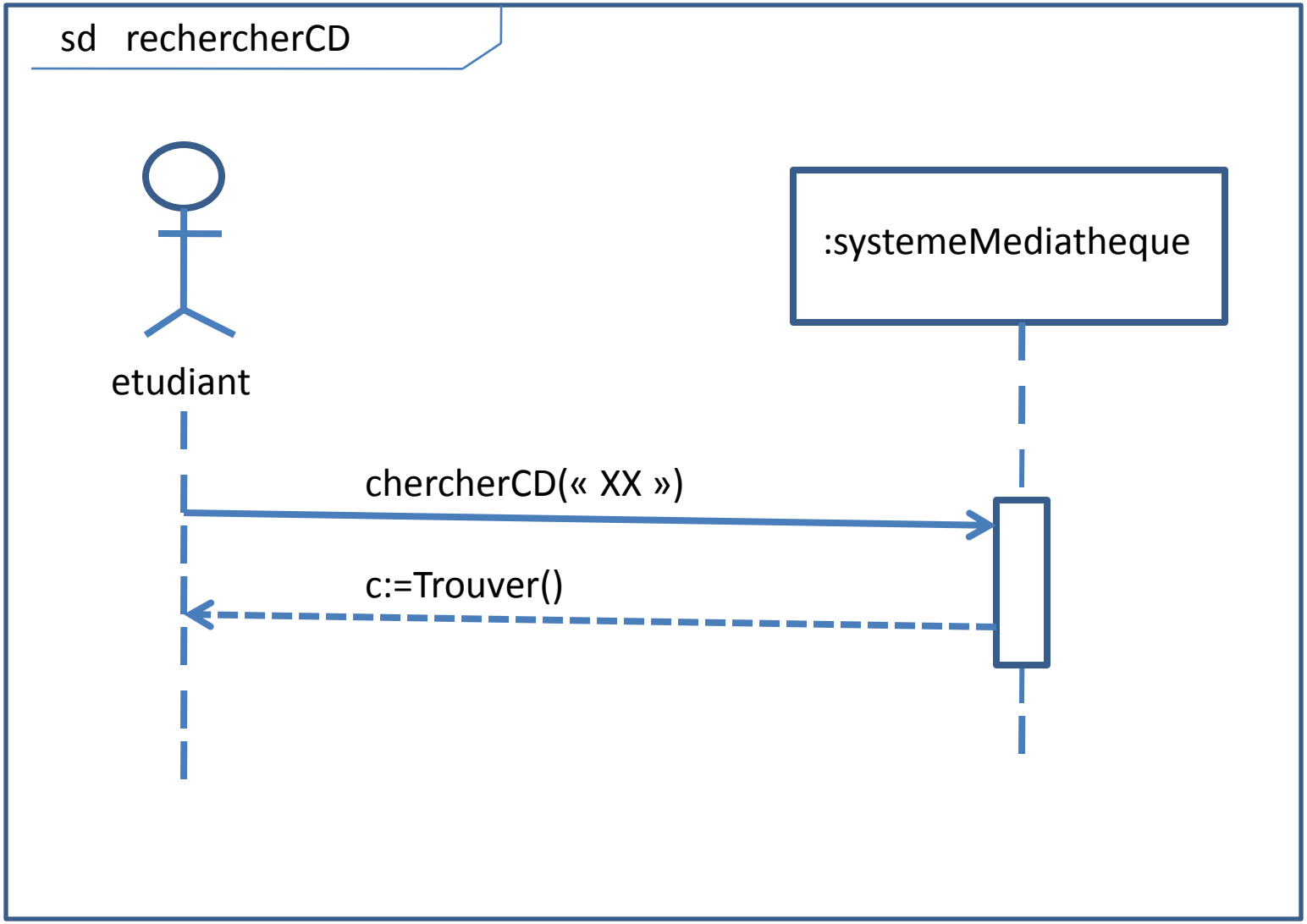
Création/Destruction d'un objet (d'instance)



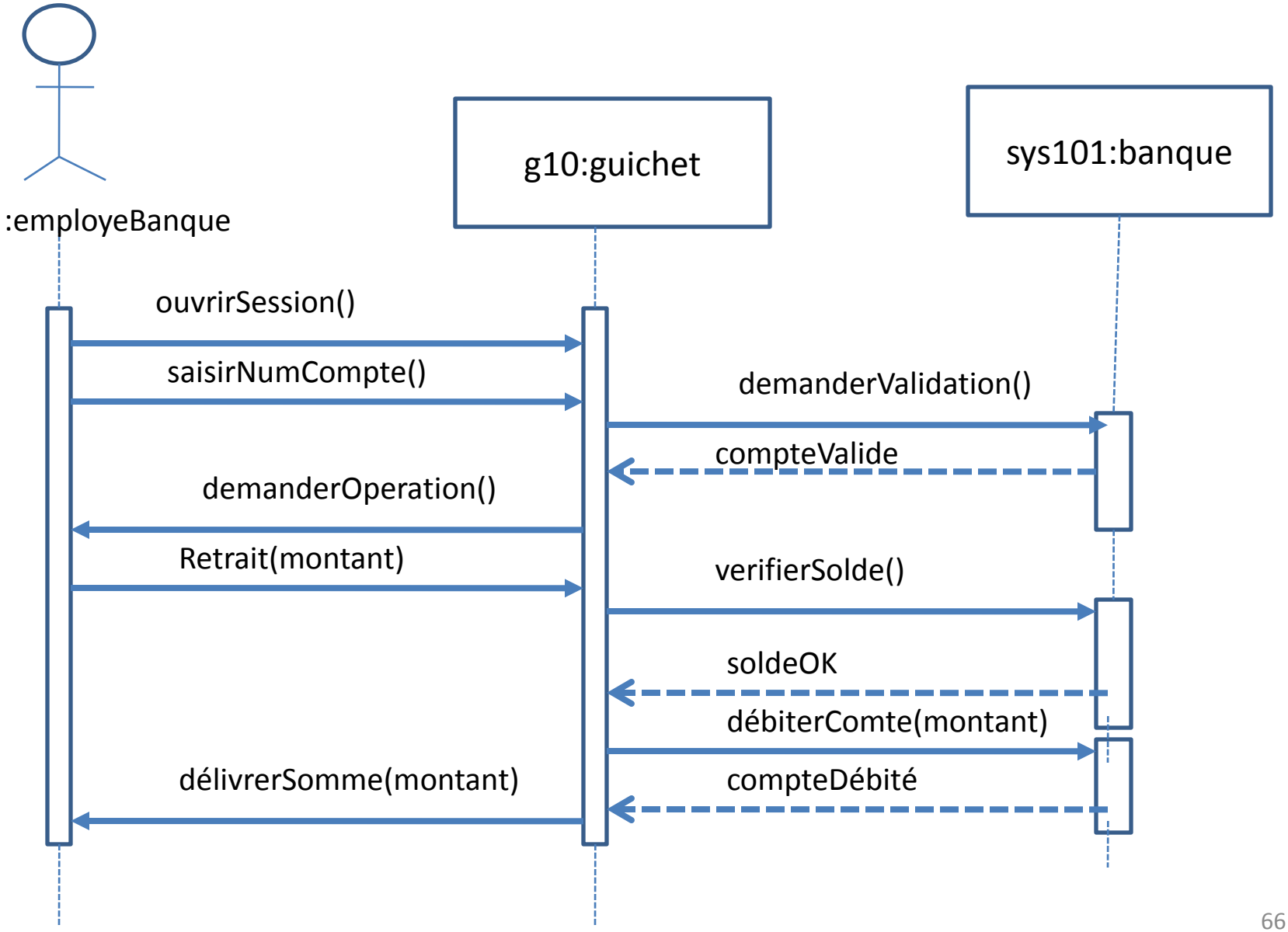
Exemple 1



Exemple 2


sd rechercherCD
:systemeMediatheque
etudiant
chercherCD(« XX »)
c:=Trouver()

Exemple 3


:employeBanque
g10:guichet
sys101:banque
ouvrirSession()
saisirNumCompte()
demanderValidation()
compteValide
demanderOperation()
Retrait(montant)
verifierSolde()
soldeOK
débiterComte(montant)
délivrerSomme(montant)
compteDébité

Fragments d'interaction combinés

Un fragment combiné permet de simplifier la représentation des diagrammes de séquences. Il décompose une interaction complexe en fragments. Il est défini par un opérateur et des opérandes. Il existe plusieurs opérateurs définis dans la notation UML 2

Un fragment combiné regroupe des ensembles de messages pour montrer un flot conditionnel.

Les conditions de choix des opérandes sont données par des expressions booléennes entre crochets « [ ] »

Liste des opérateurs

Opérateurs de choix : **alt (ernative)** équivalent au **SI ..ALORS … SINON**

opt(ion) équivalent au **SI … ALORS (une seule branche)**

break

Boucle : **loop** répétition d'un traitement **(n fois** ou **condition, …)**

Envoi en parallèle de messages : **parallel** et **critical region**

L’opérateur **alt** représente des choix mutuellement exclusifs entre deux ou plusieurs séquences de messages,

L’opérateur **opt** : pour modéliser une séquence qui se produira sous une certaine condition

Reference **ref** : pour référencer un autre diagramme de séquence.

break : pour modéliser la gestion des exceptions (sortie de boucle, par ex.)

parallel **par** : activités exécutées en parallèle

Ignore et **consider** : pour que l’opérateur ignore (ou considère) certains messages si nécessaire.

Diagramme de séquence : Fragment alternative

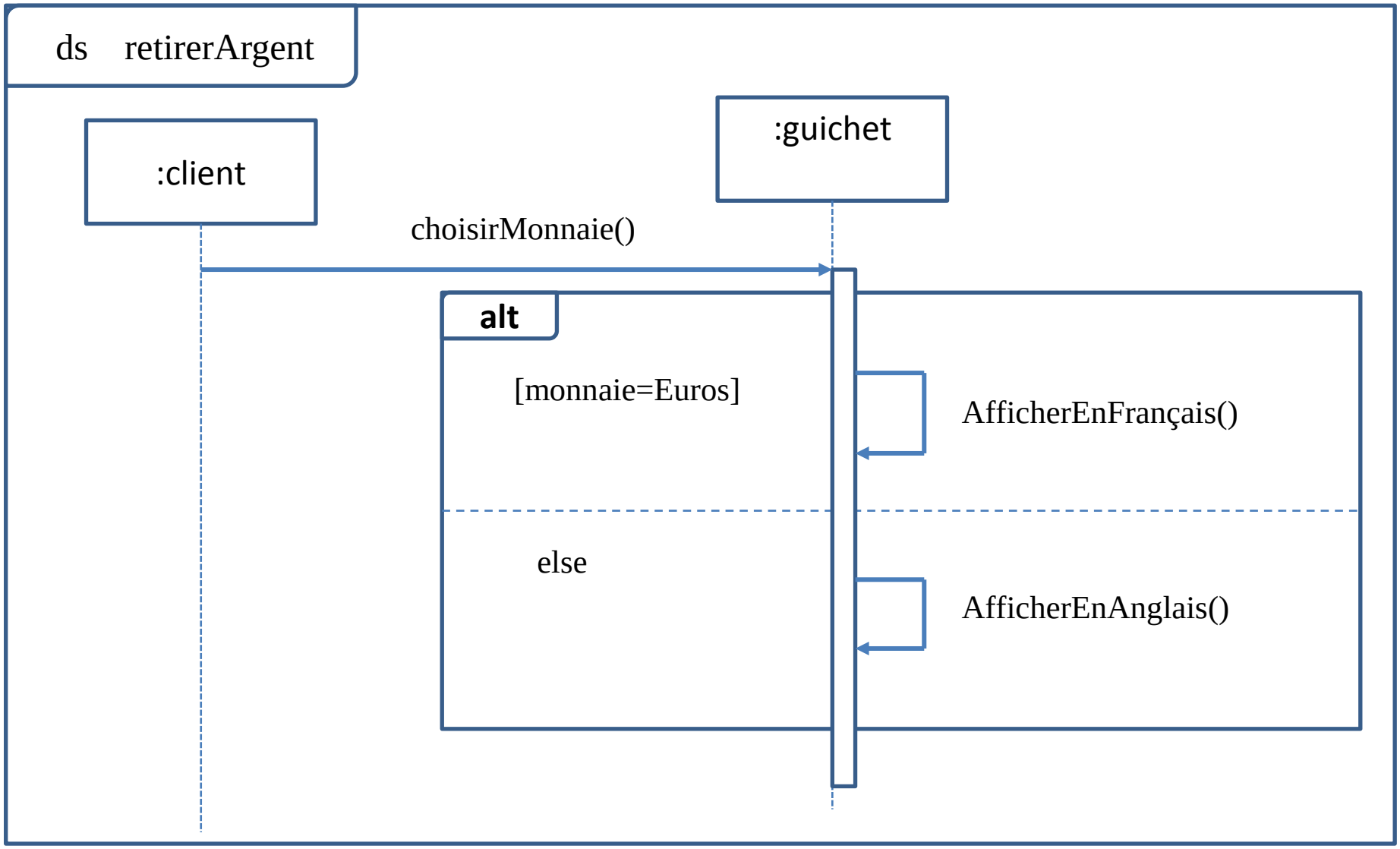


Diagramme de séquence : Fragment boucle

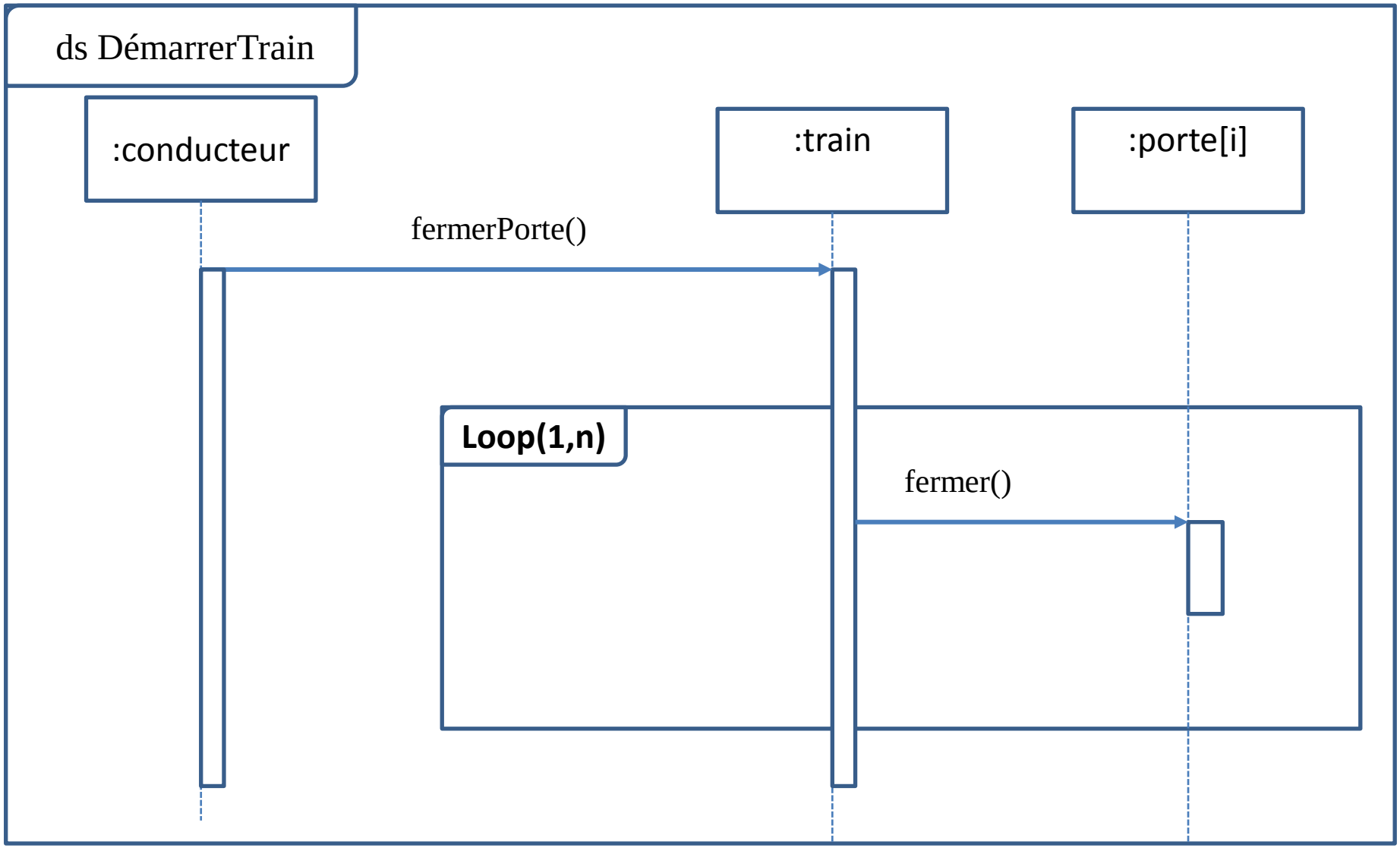
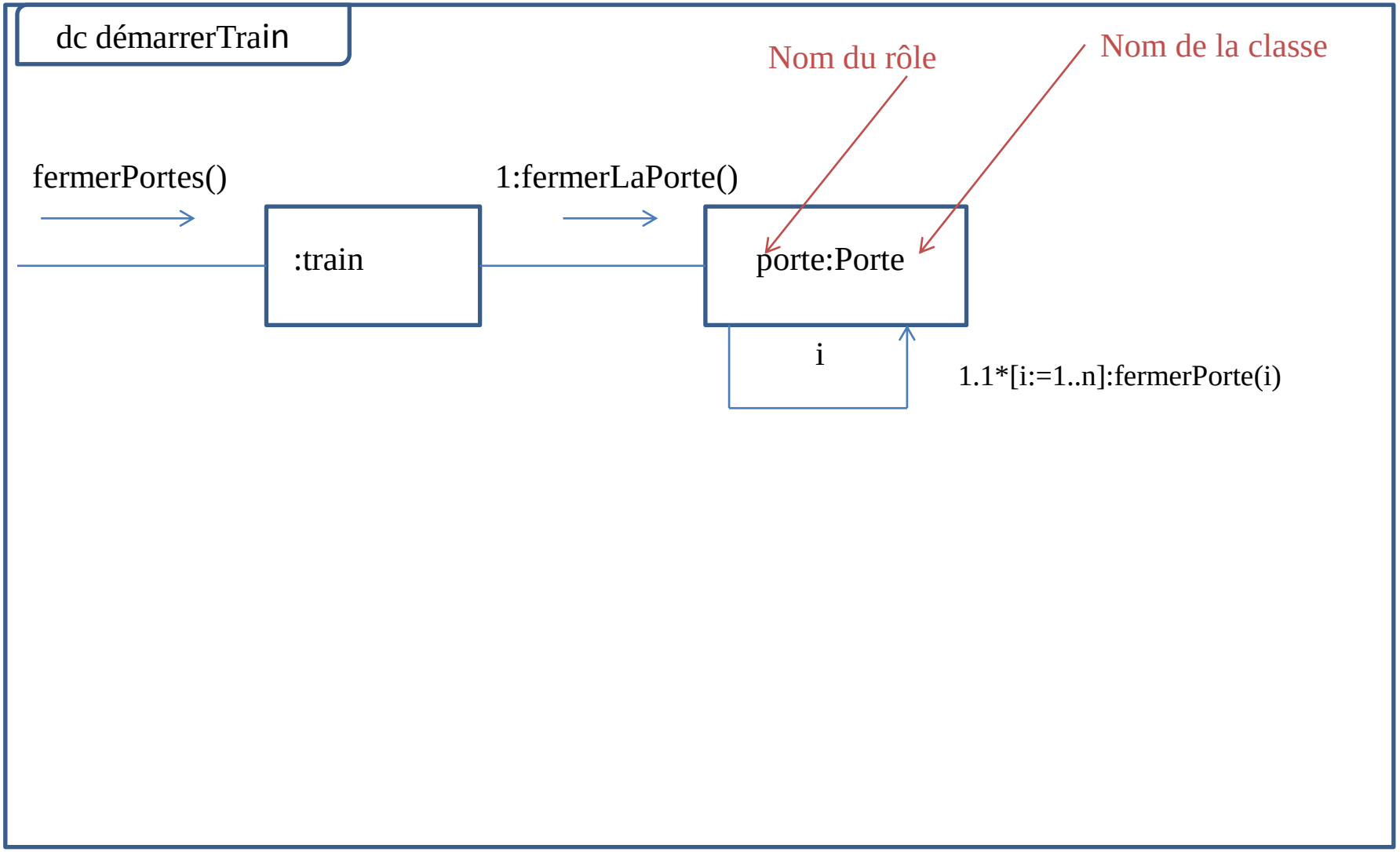


Diagramme de communication : démarrerTrain



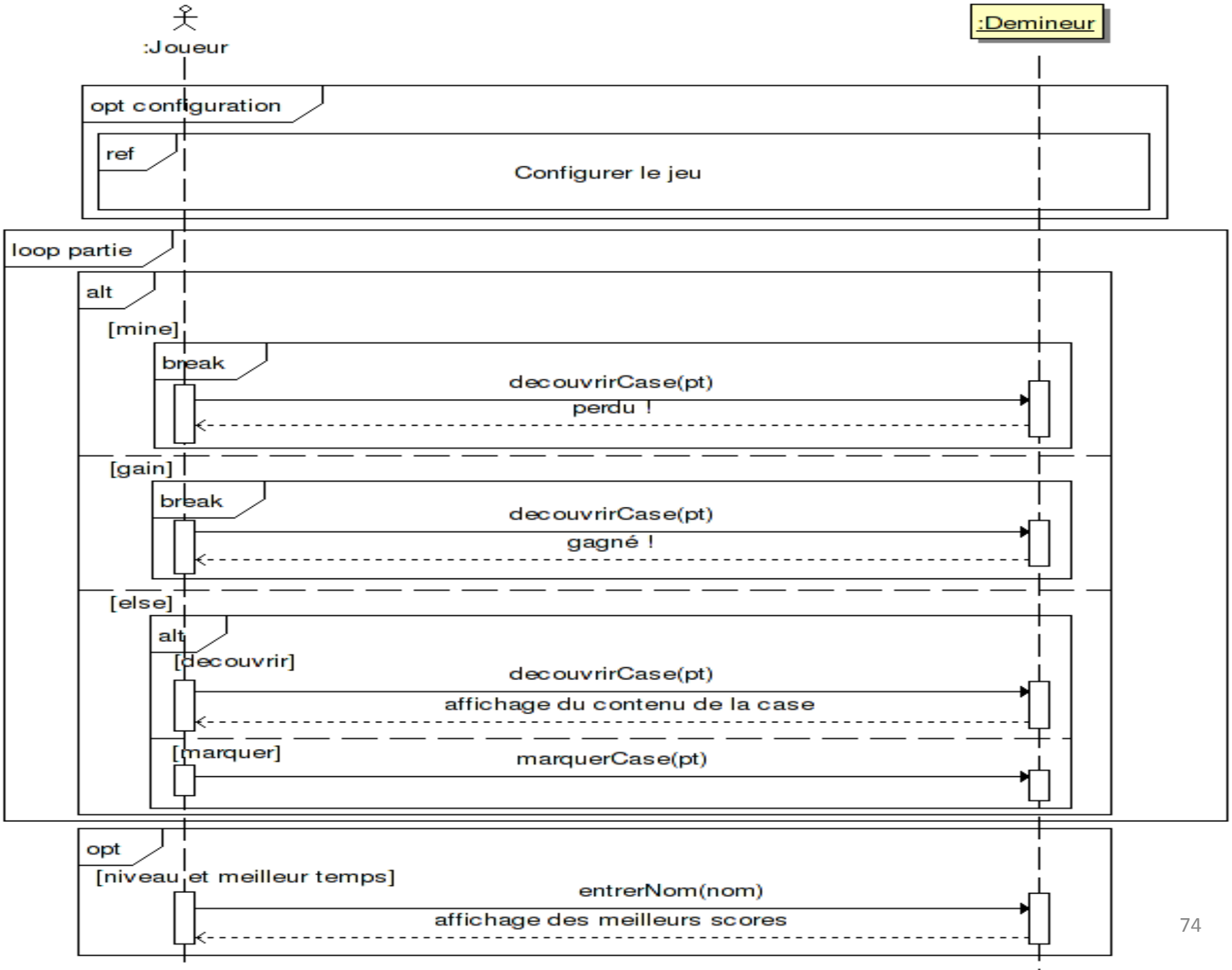
Exemple : jeu du démineur

Au début, on configure le jeu, puis on boucle :

- soit on tombe sur une mine, on a perdu, on sort la partie est finie
- soit on découvre une case libre, on a gagné, on sort la partie est finie
- soit on continue : 2 cas : - on découvre la case dont le contenu est affiché
  - soit il marque les cases

Dans les 2 cas, la partie continue jusqu'à gagné ou perdu (sortie de la boucle avec break)

A la fin, on rentre le nom, il affiche le meilleur score dans le jeu.


:Joueur
:Demineur
opt configuration
ref
Configurer le jeu
loop partie
alt
[mine]
break
decouvrirCase(pt)
perdu !
[gain]
break
decouvrirCase(pt)
gagné !
[else]
alt
[decouvrir]
decouvrirCase(pt)
affichage du contenu de la case
[marquer]
marquerCase(pt)
opt
[niveau et meilleur temps]
entrerNom(nom)
affichage des meilleurs scores

Autre exemple : 2 boucles

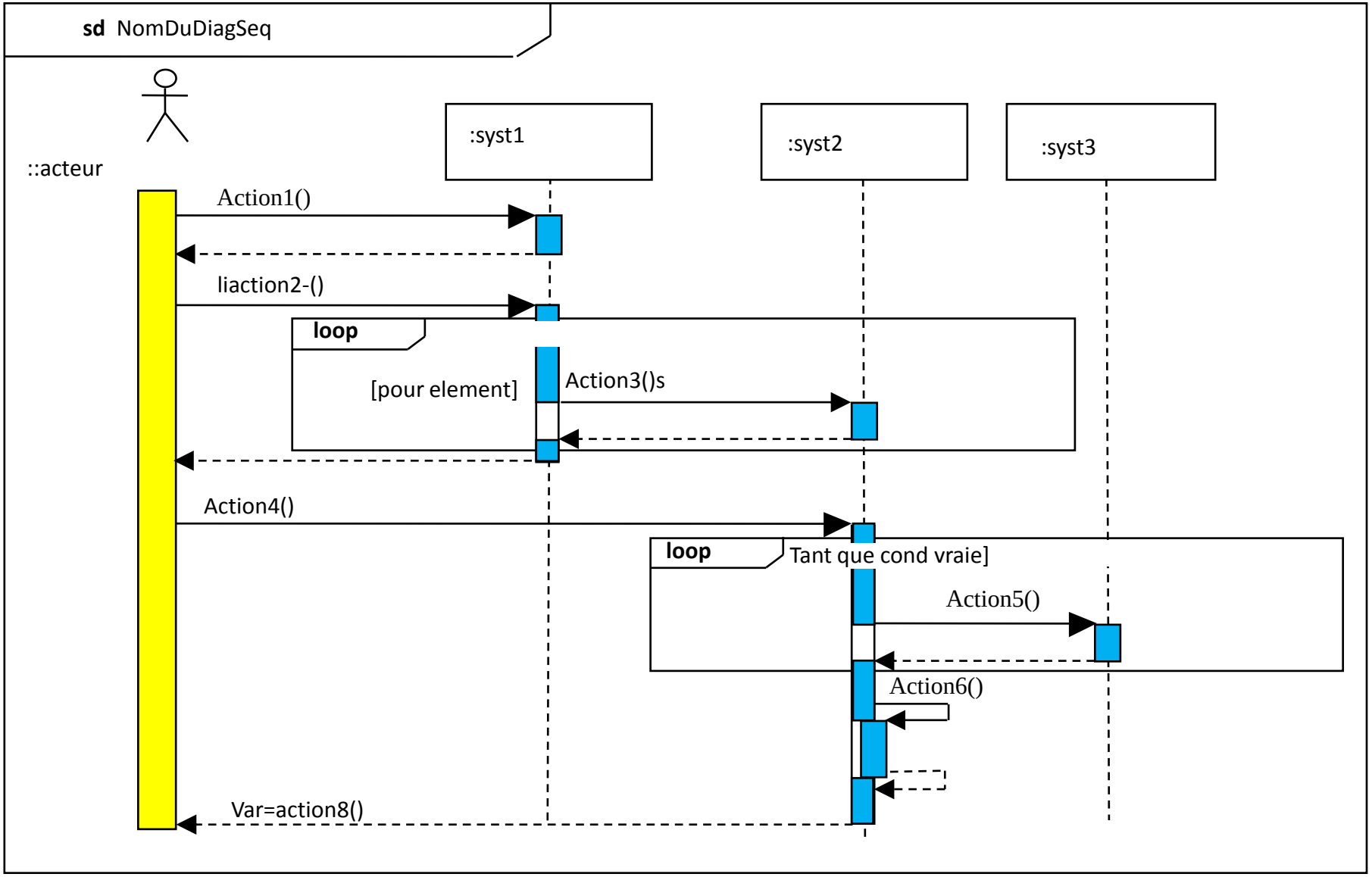


Diagramme de communication

Donne une représentation spatiale des objets avec leurs interactions, contrairement au diagramme de séquence où l'aspect temporel prévaut. Donc, il représente en quelque sorte un diagramme d'objets avec les communications.
Les messages échangés sont étiquetés de la manière suivante :

Dans un diagramme de communication, les messages sont généralement ordonnés selon un numéro de séquence croissant.
Un message est, habituellement, spécifié sous la forme suivante:

[pré « / »] ['['<cond>']' [<séq>] [* [||] ['['<iter>']']] :] [<var> :=] <msg> ([<par>])

<pré> liste des numéros de messages prédécesseurs. Indique que le message courant n'est envoyé qu'après l'envoi de ses prédécesseurs.

<cond> garde, exp booléenne, conditionne l'envoi du message.

<séq> est le numéro de séquence du message. On numérote les messages par envoi et sous-envoi désignés par des chiffres séparés par des points : ainsi l'envoi du message 1.4.4 est postérieur à celui du message 1.4.3, tous deux étant des conséquences (*i.e.* des sous-envois) de la réception d'un message 1.4. La simultanéité d'un envoi est désignée par une lettre : les messages 1.6*a* et 1.6*b* sont envoyés en même temps.

<iter> spécifie (en langage naturel, entre crochets) l'envoi séquentiel (ou en parallèle, avec ||) de plusieurs message. On peut omettre cette spécification et ne garder que le caractère * (ou *||) pour désigner un message récurrent envoyé un certain nombre de fois.

<var> valeur de retour du message, qui sera par exemple transmise en paramètre à un autre message.

<msg> nom du message.

<par> paramètres (optionnels) du message.

Rmq : Cette syntaxe permet de décrire précisément l'ordonnancement et la synchronisation des messages entre les objets du diagramme de communication. Le sens de transmission d'un message est spécifié par une flèche pointant vers objets destination du message. Les objets sont reliés par un trait continu (connecteur).

Exemples de messages transmis

1.4.5 * : ouvrir () → message OUVRIR() envoyé de manière séquentielle plusieurs fois

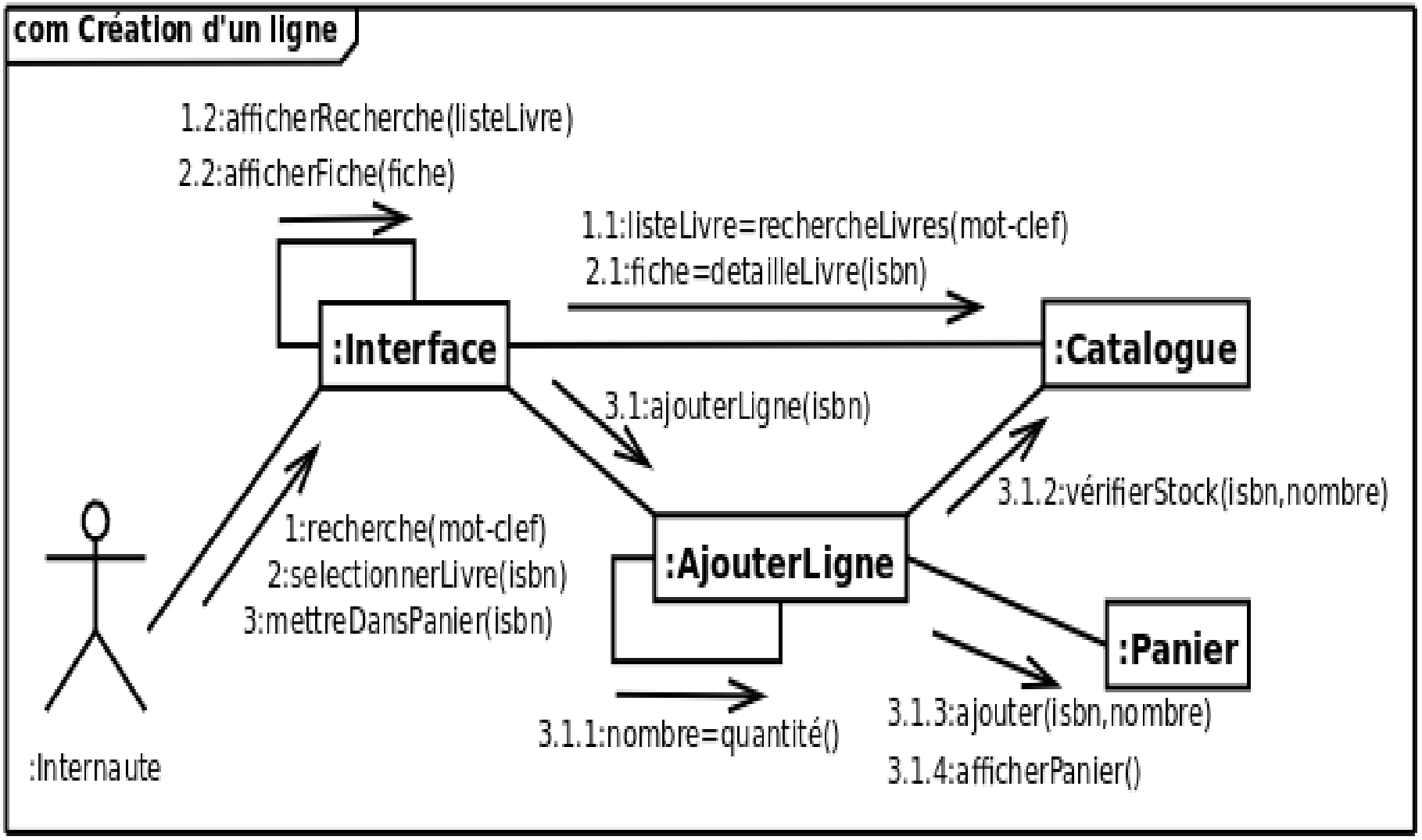
[heure = 17] 1 : saisirInfo(n, p, a) → message saisirInfo (nom, pren, adr) est effectué à 17h

1.3, 2.1 / [h > 10h] 2.5 : age := demanderAge(n, p) →
message num 2.5 ne sera envoyé qu'après l'envoi des message 1.3 et 2.1 et après 10h

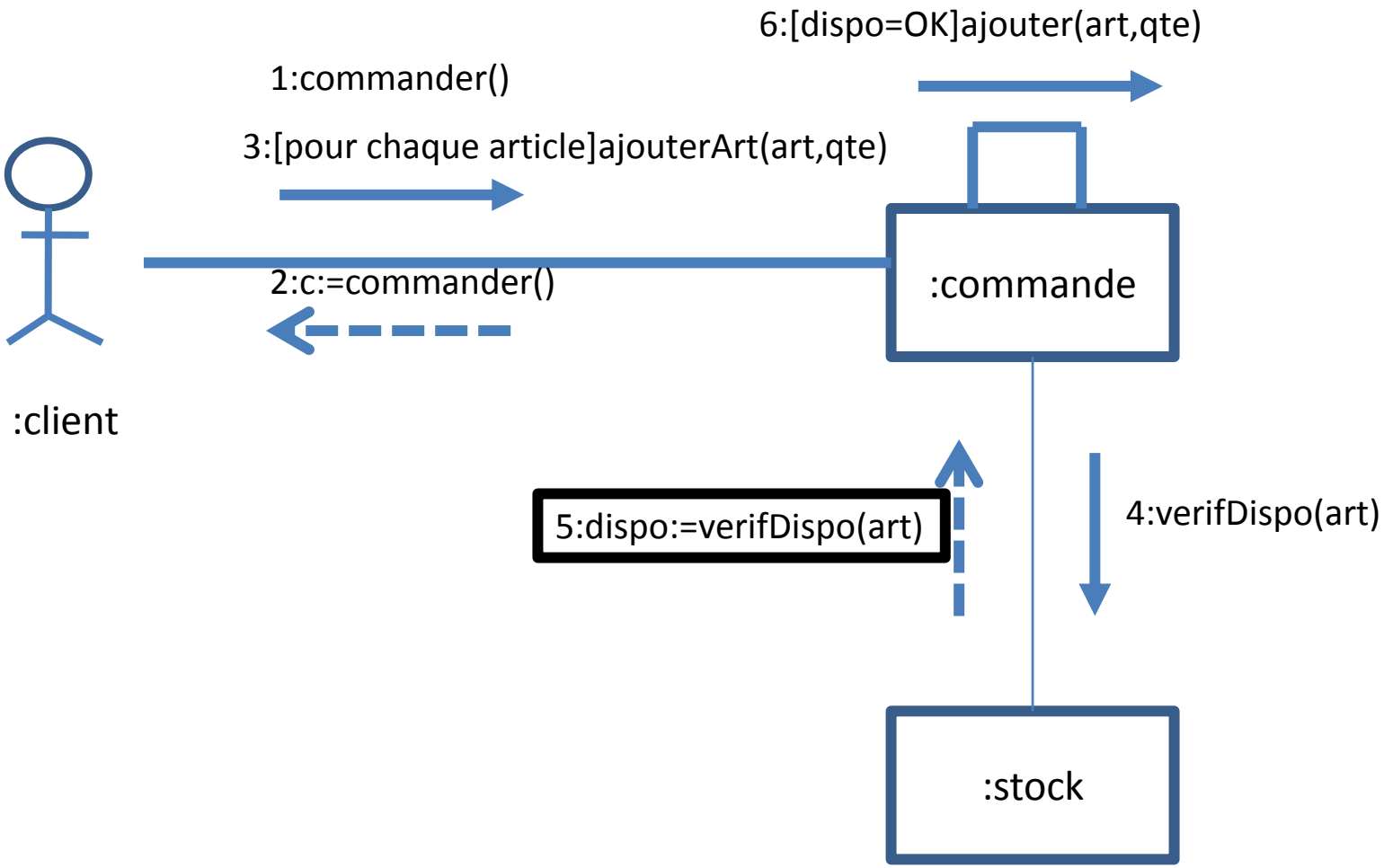
1.3 / [disk full] 1.7.a deleteTempFiles()

1.3 / [disk full] 1.7.b reduceSwapeZone() → Les messages 1.7.a et 1.7.b ne seront envoyés qu'après le message 1.3 et que «disk full» est réalisée. 1.7.a peut être envoyé en parallèle avec le message 1.7.b

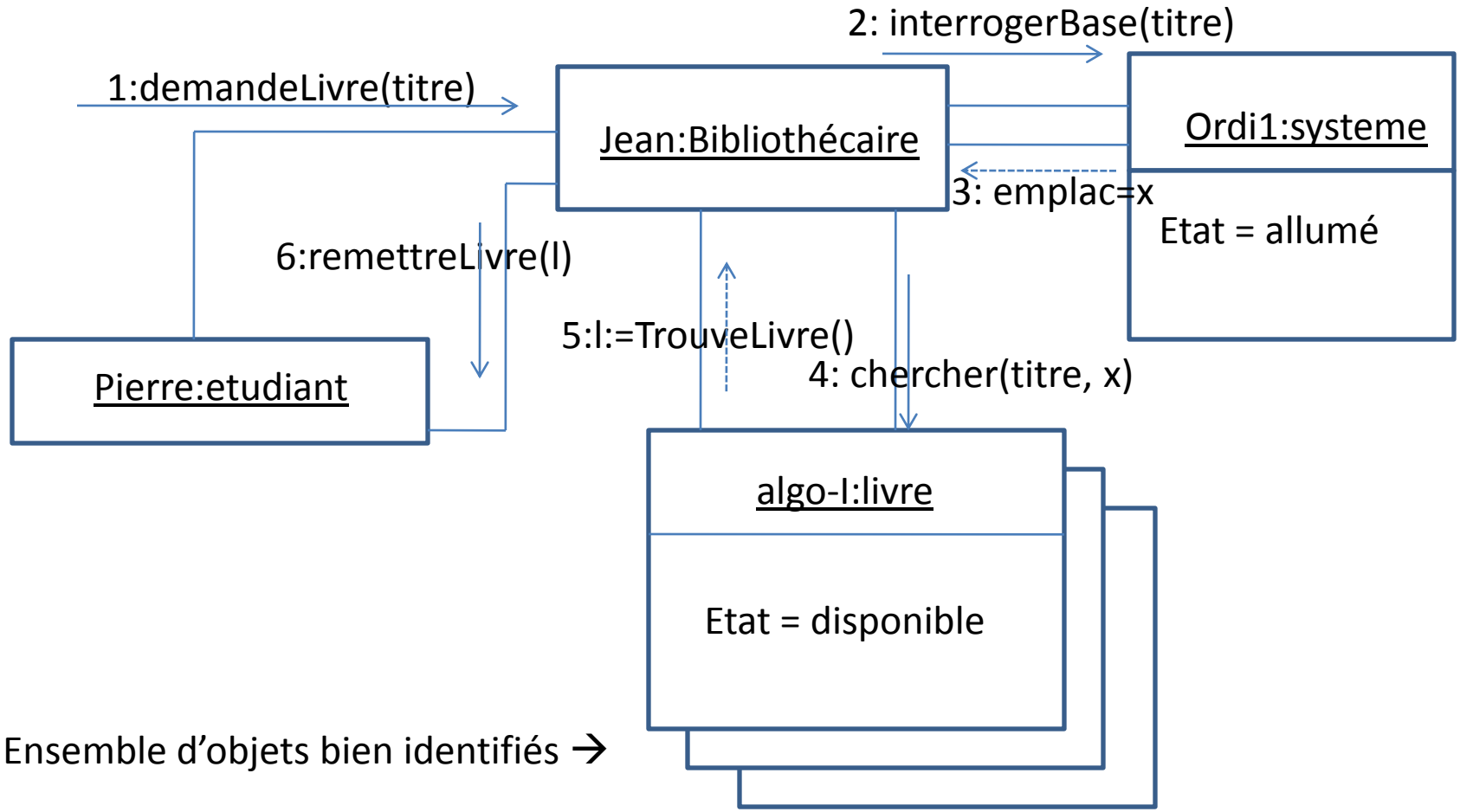
Exemple 1 : Création d'une ligne de commande



Exemple 2 : Gérer une commande



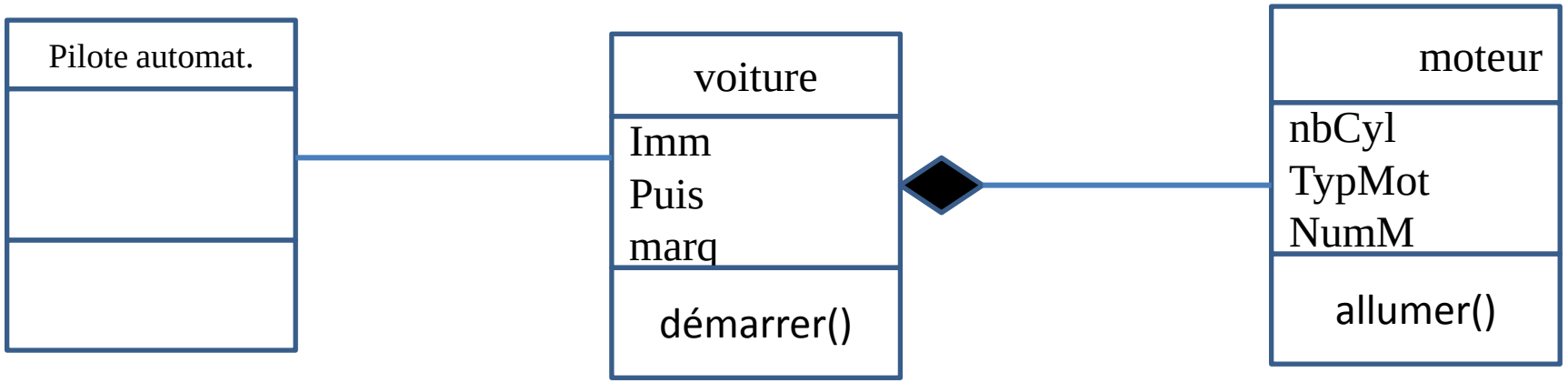
Exemple 3 : Diagramme Objet «Prêt de livre»



Ensemble d'objets bien identifiés →

Pierre sollicite Jean, le bibliothécaire pour emprunter un livre. Jean interroge la base de données Il prend le livre et le remet à Pierre.de données (dans l'ordi)

Exemple : Diagramme de classe et diagramme de communication correspondant



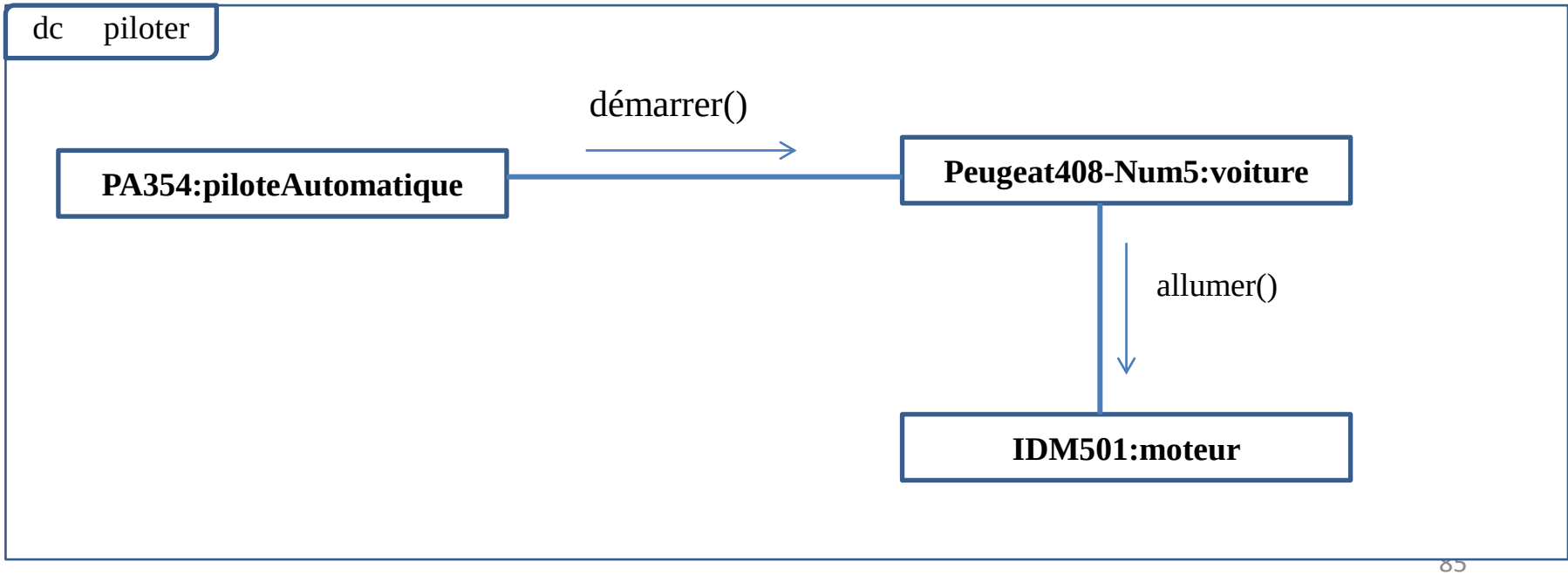


Diagramme de séquence correspondant

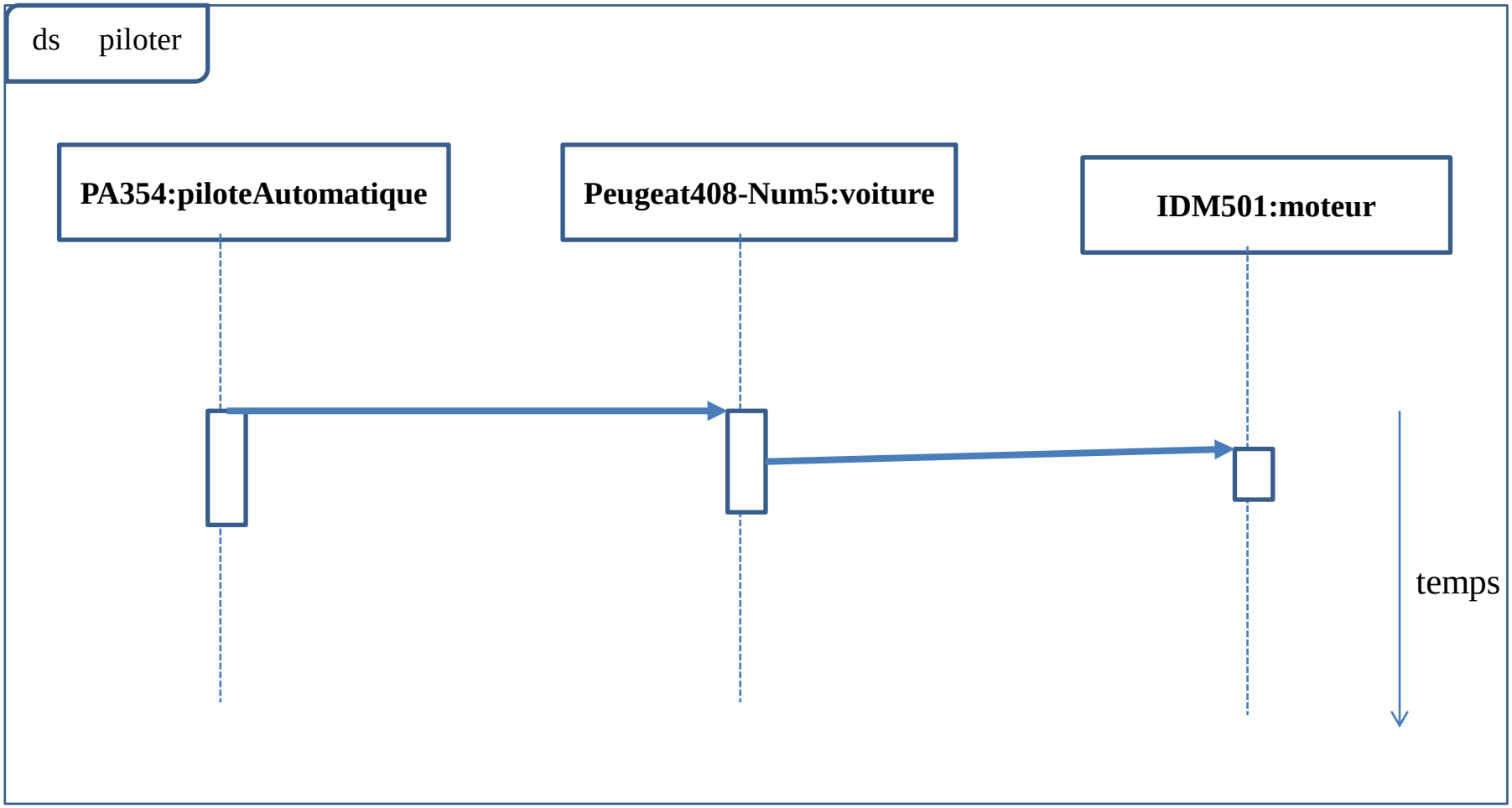


Diagramme de séquence : venteImmobilière

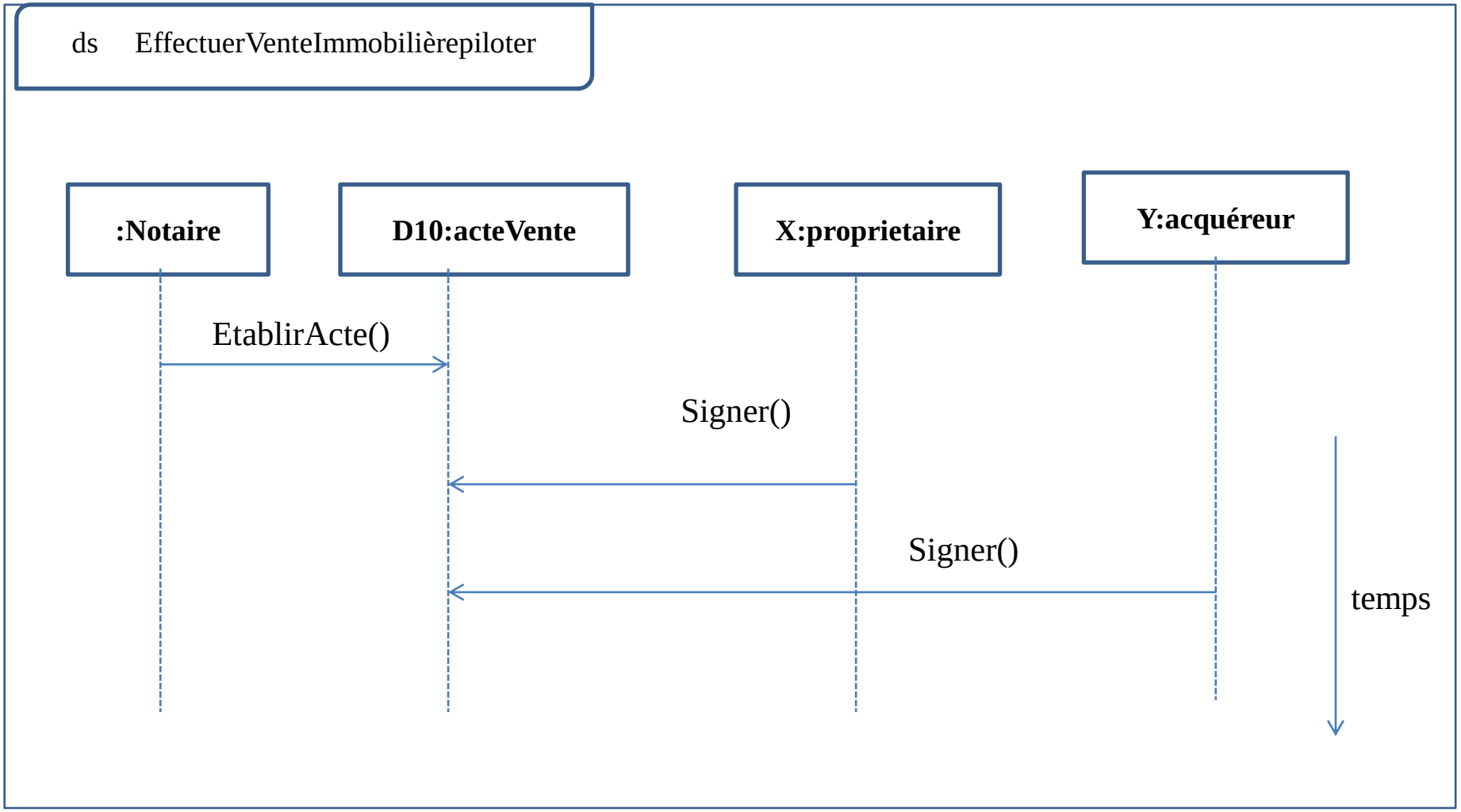
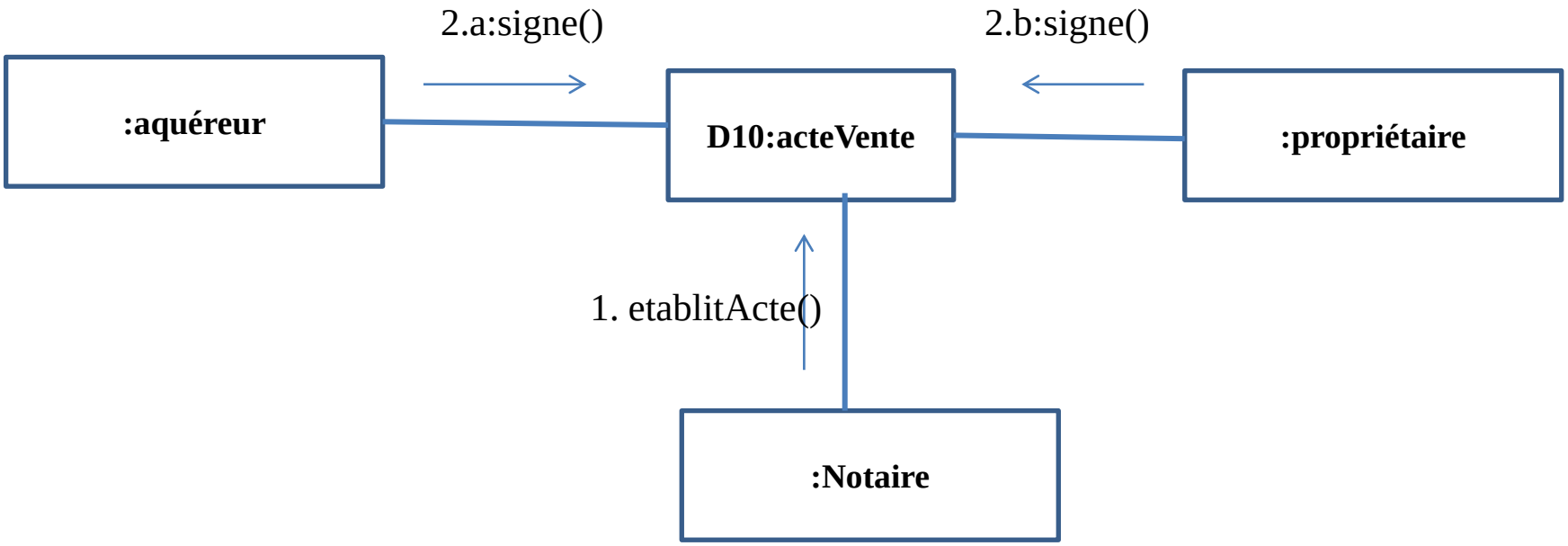


Diagramme de communication : venteImmobilière



UML 2 cours

Diagramme de classe,
Diagramme d’objets

- Diagrammes de cas d'utilisation (USE CASE)

- **Diagrammes de classes**
- **Diagrammes d'objets**

- Diagrammes de séquence
- Diagrammes de communication
- Diagrammes d'états/transition
- Diagramme d'activités

Diagrammes de classe UML

Vue statique du système

- diagrammes de cas d'utilisation (fonctionnalités de système)
- diagrammes de classes
- diagrammes d'objets
- diagrammes de composants
- diagrammes de déploiement
- …

Vue dynamique du système

- diagrammes de séquence
- diagrammes de communication
- diagrammes d'états-transitions
- diagrammes d'activités
- …

Diagramme de classe

- Il s’agit de l’un des diagramme les plus importants. Il représente la structure statique du système en terme de classes et de relations entre ces classes (~entités et associations)

- Une classe est caractérisée par ses propriétés : attributs (aspect statique) et méthodes (aspect dynamique) qui sont communes à un ensemble d'objets et qui permettent de créer des objets ayant ces propriétés.

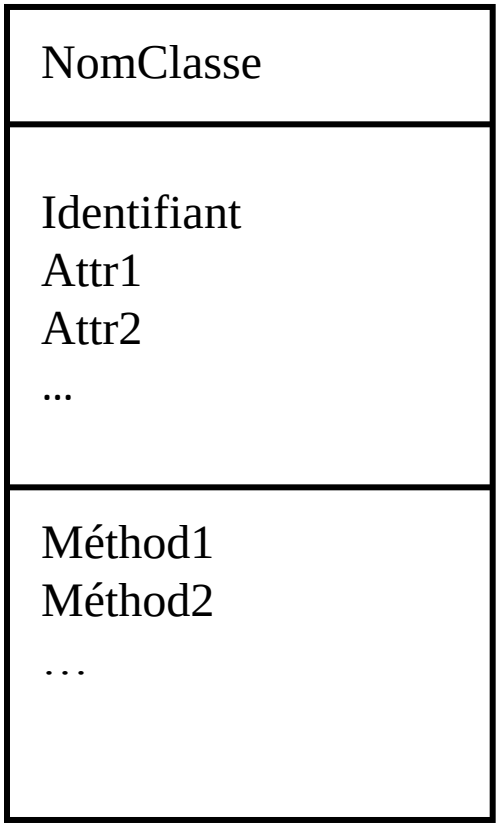
- Exemple : la classe ETUDIANT, ayant les attributs : num-etud, nom-etud, group-etud, dipl-prep, et comme méthodes : s’inscrire-a-iut(), s’inscrire-au-sport(), s’absenter().

Diagramme de classe

- Dans certains cas, on ne représente pas les attributs ou les méthodes d'une classe sur un diagramme (ou on ne les représente pas tous), pour des raisons de lisibilité ou de visibilité (pour représenter la classe avec un certain niveau d'abstraction).

Représentation Graphique d'une classe UML

Ou bien

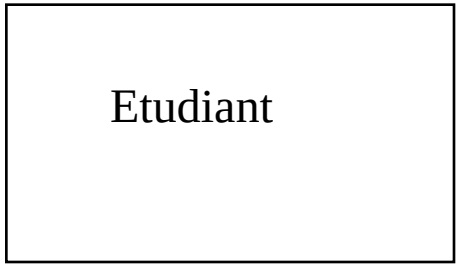


Ou bien plus complet :

- Attribut :
  - nom-attr : type = valeur_par_défaut
- Méthode :
  - nom_method (arg : type = valeur_par_defaut, …) : type_de_retour
- Types d'accès :
  - + : (publique), Public, visible de toutes les classes
  - # : (protégé), Protected, visible de la classe et ses sous-classes (ses descendants)
  - - : (privé), Private, visible dans la classe uniquement

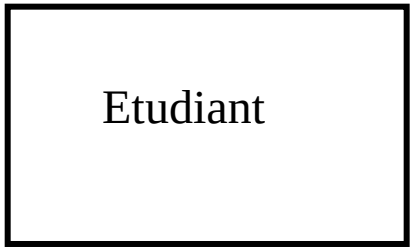
Niveaux d’abstraction (1)

→ Une certaine vue sur une classe



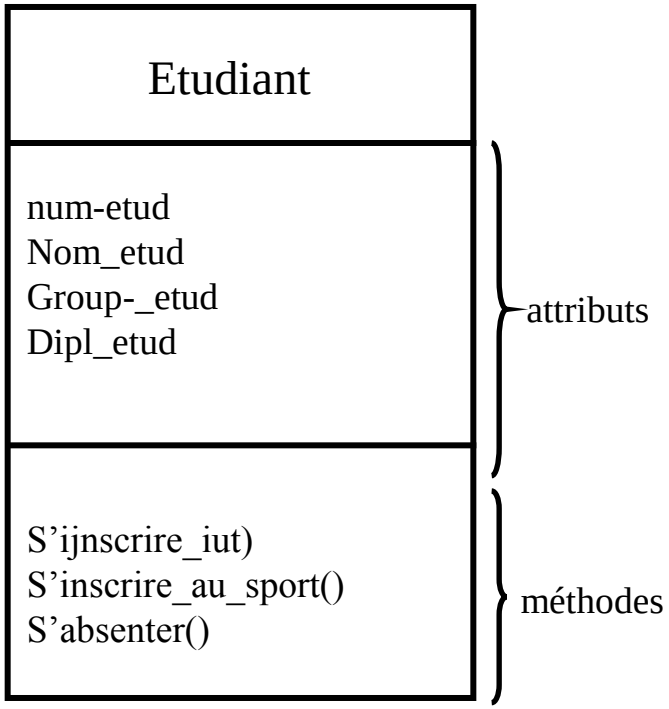
Classe non documentée

Niveaux d’abstraction (2)



Classe non documentée

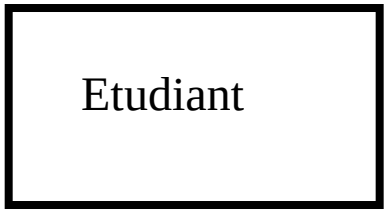
et



Classe documentée

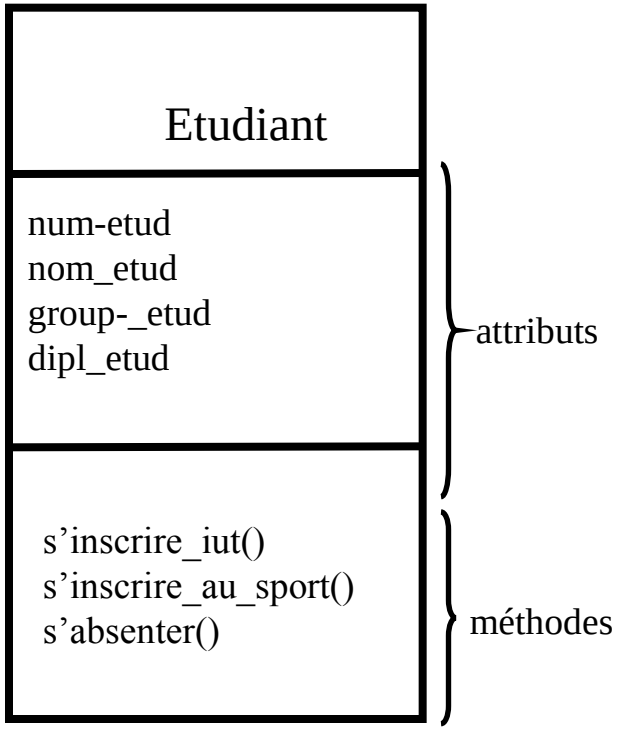
Niveaux d'abstraction (3)

(I)



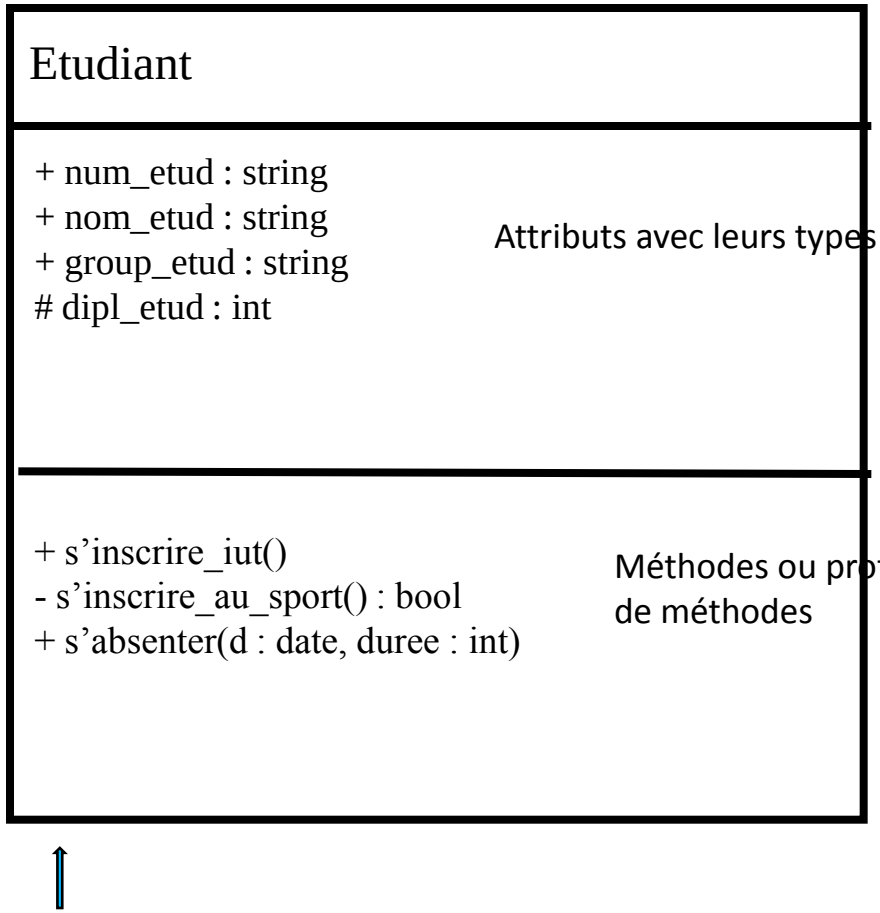
Classe non documentée

(II)



Classe documentée

(III)



Types d'accès

Classe détaillée

Principaux concepts

Objet (occurrence d'une classe, instance)
Classe :
attributs (caractéristiques) et
méthodes (opérations)

Association
généralisation/spécialisation (« héritage »)
composition (composé/composants)
agrégation (ensemble/éléments)
simple

Différentes sémantiques pour un diagramme de classe

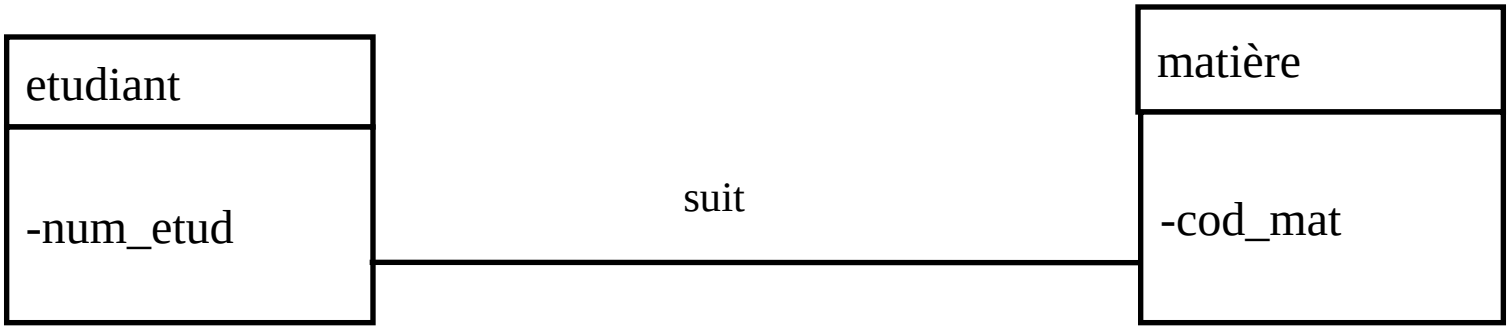
- Un diagramme de classes comprend des éléments de modélisation statiques comme les classes ou les paquetages....
- Il représente la décomposition structurelle d'un modèle.

- Parfois, on peut avoir recours à plusieurs diagrammes de classes pour modéliser un système complexe. On peut se focaliser sur :
  - les classes qui participent à un cas d'utilisation.
  - les classes impliquées dans la réalisation d'un scénario d'un cas d'utilisation.
  - les classes qui composent un paquetage
  - etc ...

Association ou relation

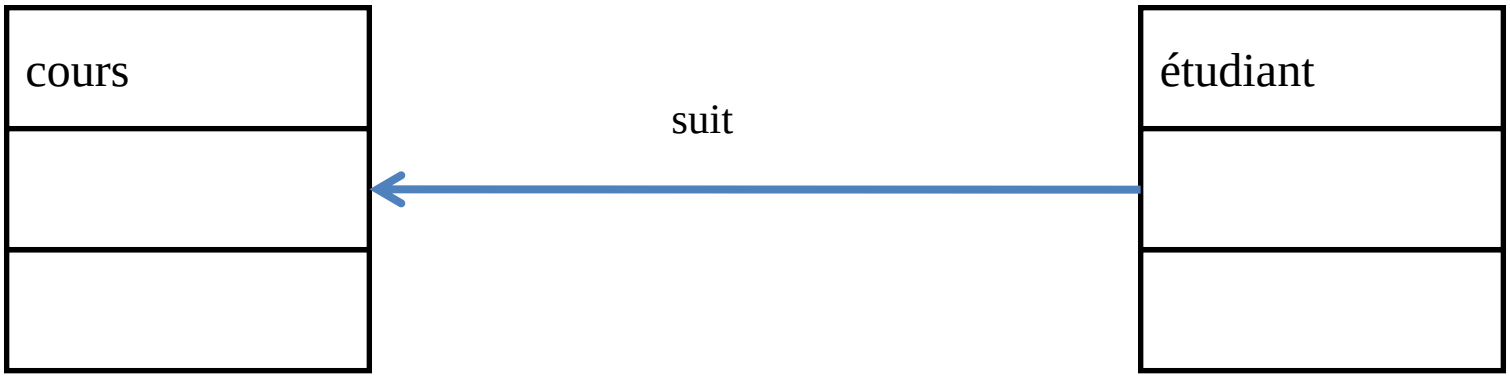
- Une association exprime une connexion (un lien) sémantique entre deux classes, qui est ensuite instanciée dans un diagramme d'objets ou de communication, sous forme de liens entre les objets (les occurrences, les instances) issus des classes concernées.

Association simple

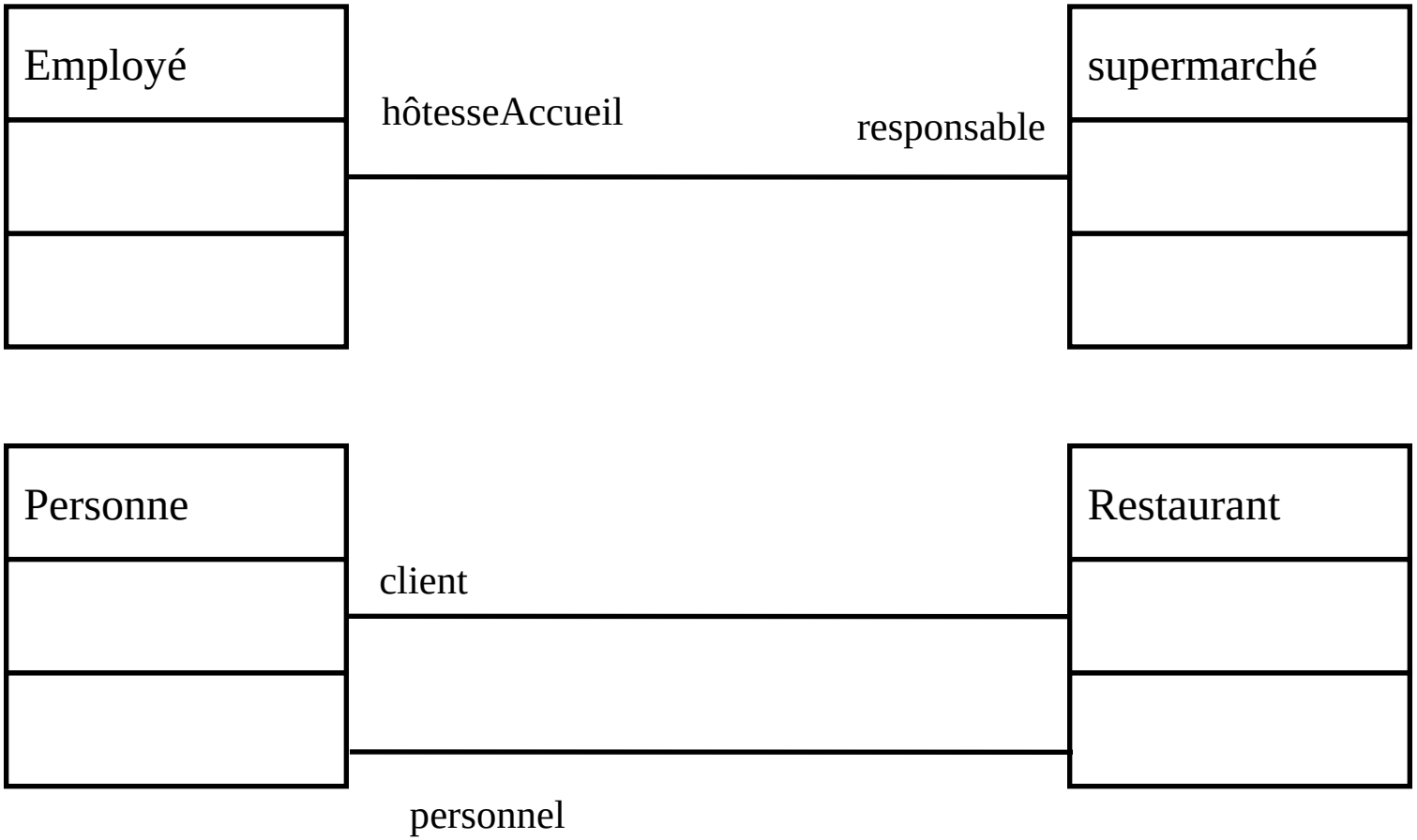


Association unidirectionnelle

→ Elle précise le sens de la lecture



Rôles d’une association

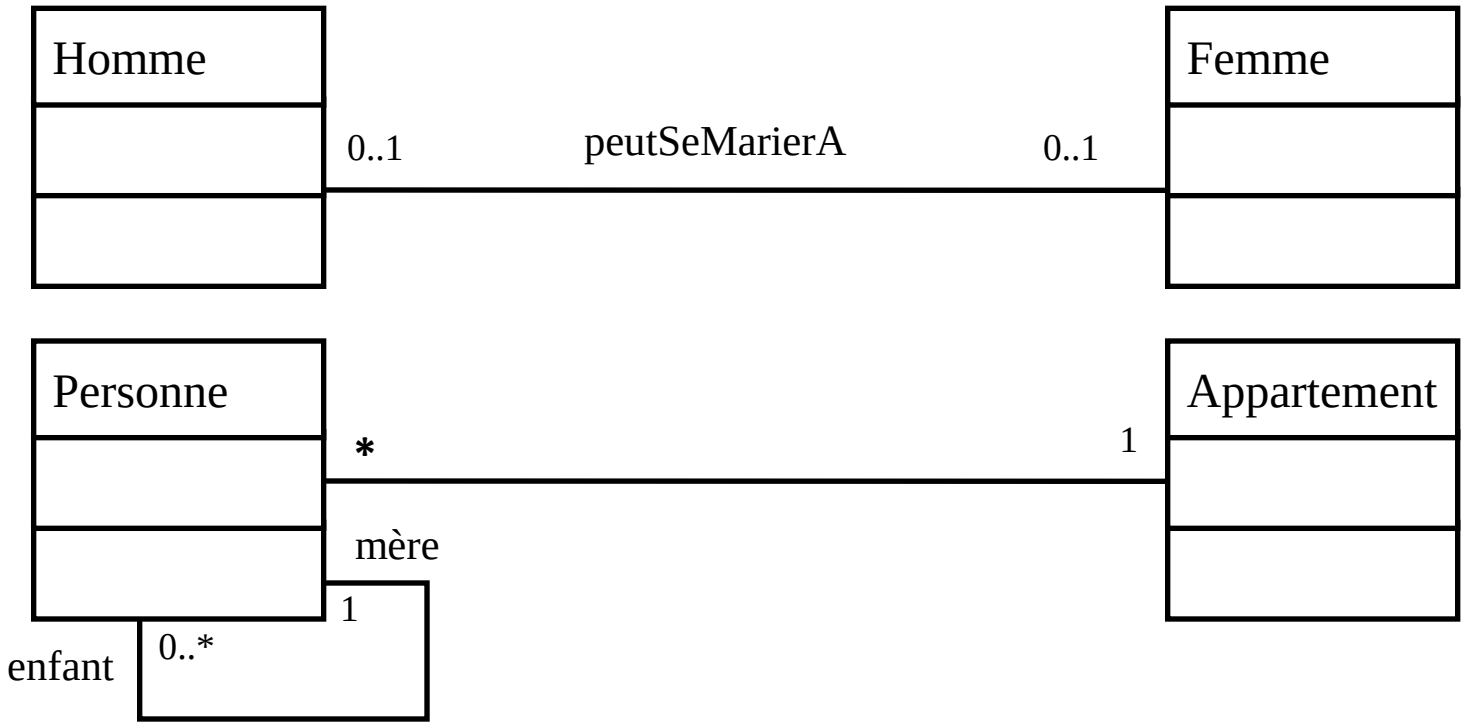


Cardinalité (multiplicité)

- = Nombre d'occurrences (instances) participant à l'association
- **n :** exactement *n* (entier naturel > 0)
  - exemples : 2, 8, 25
- **m..n :** de *m* à *n* , avec *m* > *n* (= entiers naturels ou variables)
  - exemples : 1..5, 7..30, 5..50
- *** :** plusieurs (équivalent à *0..n ou et 0..**)
- **n..* :** n ou plusieurs (rare en pratique)
  - Rmq : souvent n=0, c-à-d *0..**

Exemples

Un homme peut être marié à au plus une femme .
Une femme peut être mariée à au plus un homme .



Sens de la lecture

⇉ Remarque : lecture inverse des cardinalités dans le modèle entités/associations

→Dans les exemples précédents :

Un homme peut ne pas se marier (0 femme) ou se marier avec une seule femme (1)

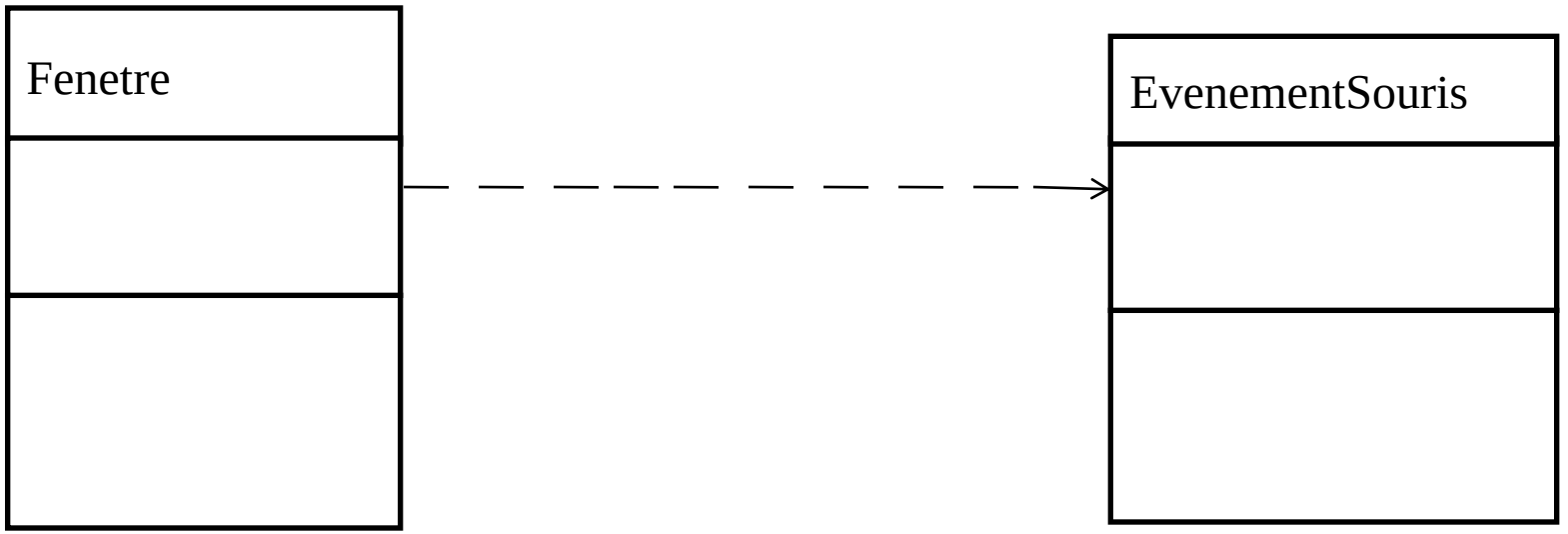
Une femme peut ne pas se marier (0 homme) ou se marier avec un seul homme (1)

Une personne habite un et un seul appartement (1) et
un appartement peut être habité par 0 ou plusieurs personnes (*).

Une personne possède une seule mère (1) et
une personne possède 0 ou plusieurs enfants (0..*) → association réflexive

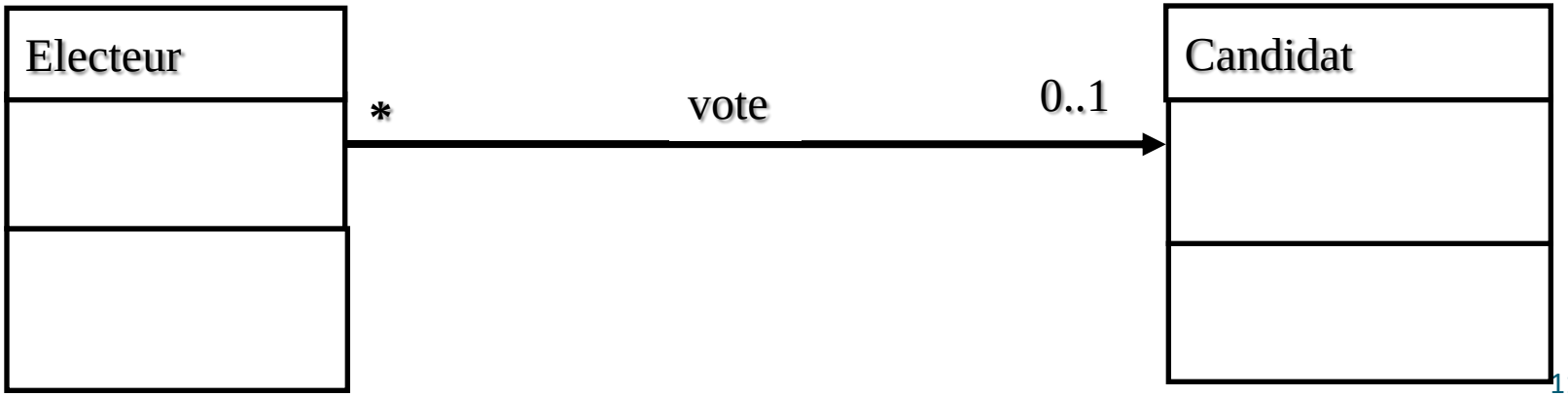
Relation de dépendance entre classes

- Orientée : une relation d'utilisation dans un seul sens
- => une modification de la classe CL1 dont dépend une classe CL2, nécessite de mettre à jour l'élément dépendant (classe CL2).
- Exemple: en mettant à jour une fenêtre d’écran, on met à jour également la position de la souris.



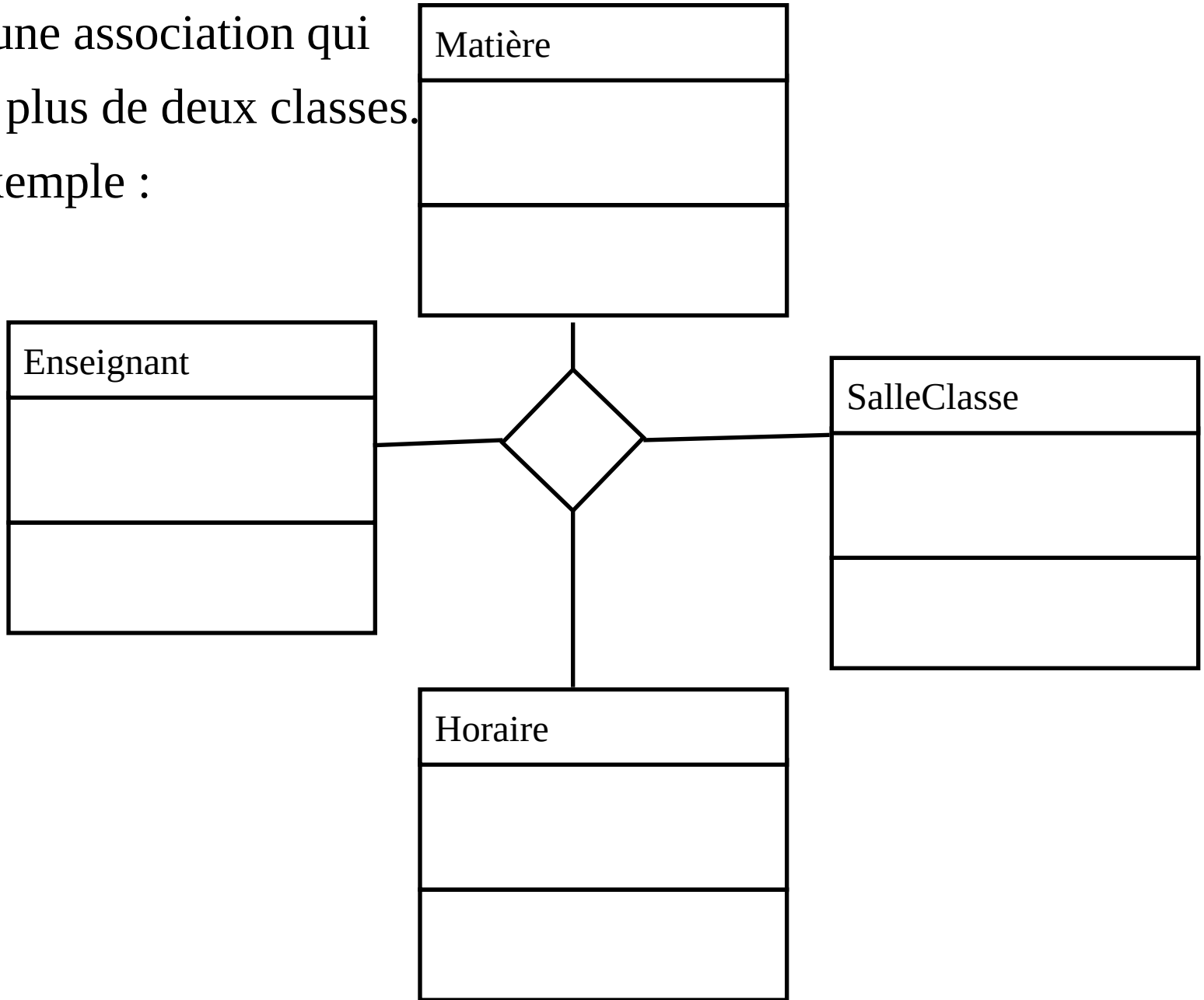
Navigabilité restreinte pour une association

- Par défaut, une association est navigable dans les deux sens
- La réduction de la portée de l'association peut être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre (elles ne les "voient" pas).
- Exemple : un électeur vote pour 0 ou pour un candidat (qu'il connait). Le candidat ne connait pas nécessairement l'électeur.



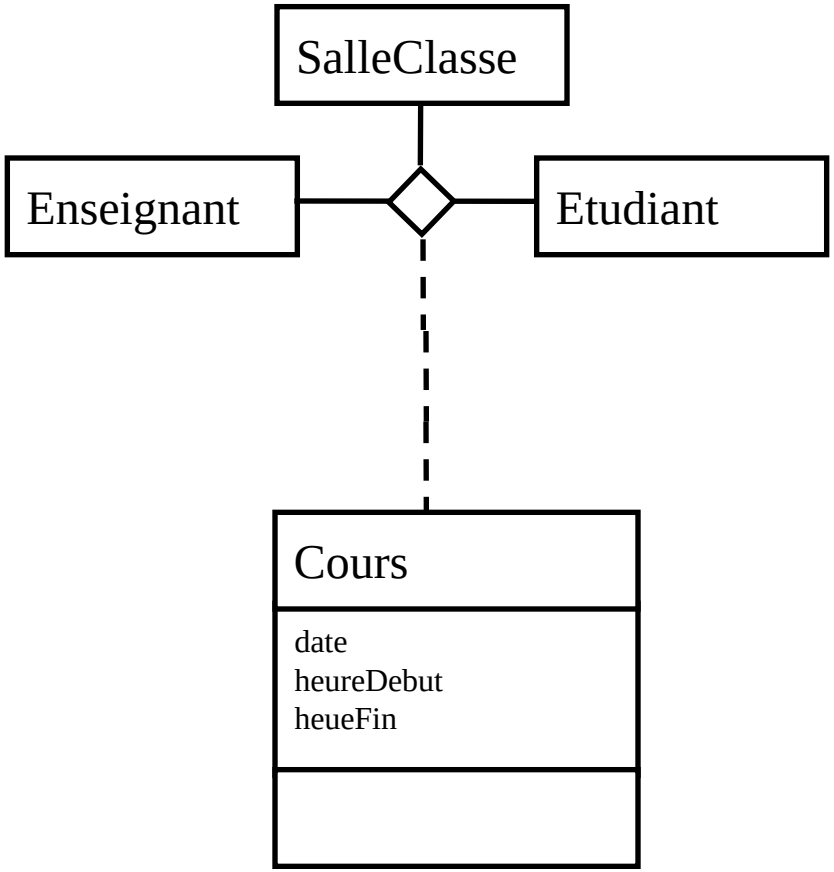
Association n-aire

- = une association qui relie plus de deux classes.
- Exemple :



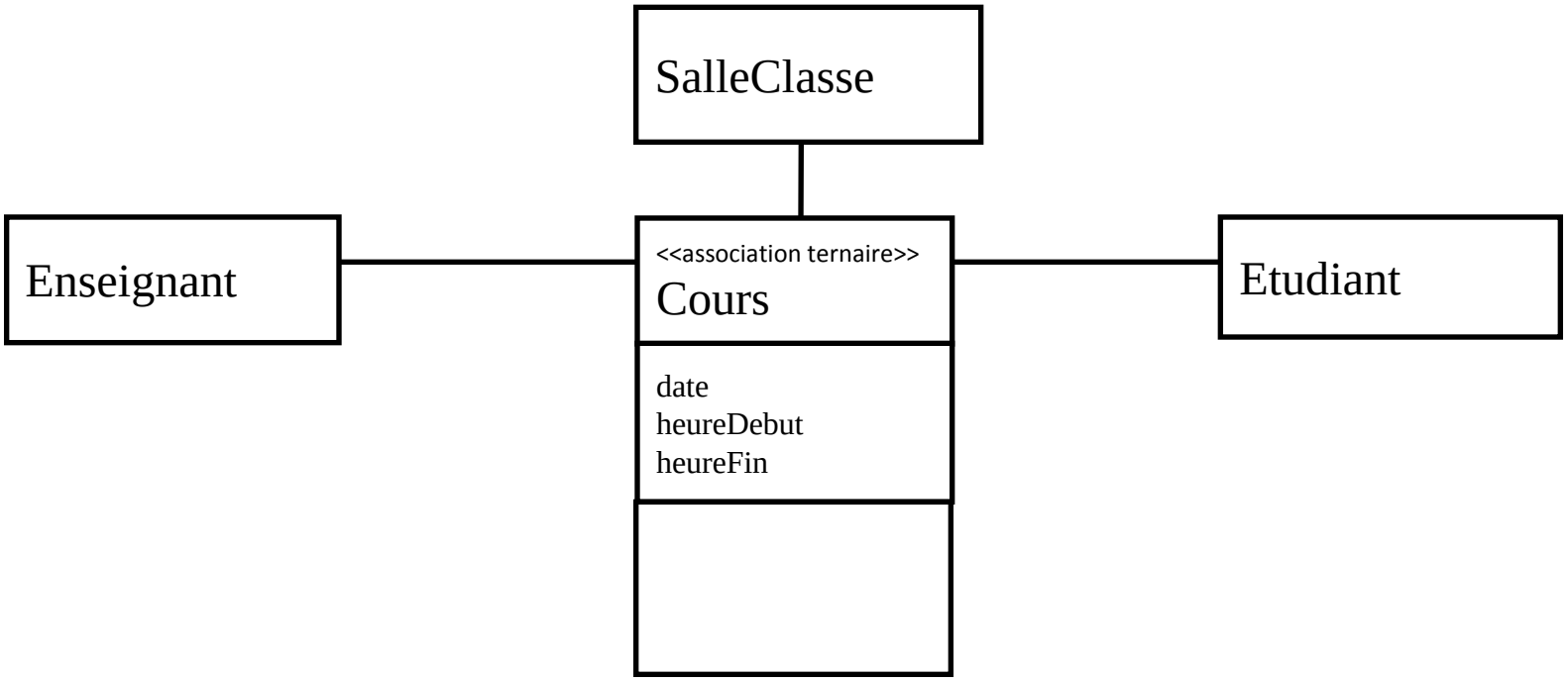
Classe-association

- = une classe qui réalise la navigation entre les instances d'autres classes. Elle est souvent porteuse d'attributs.
- Exemple :

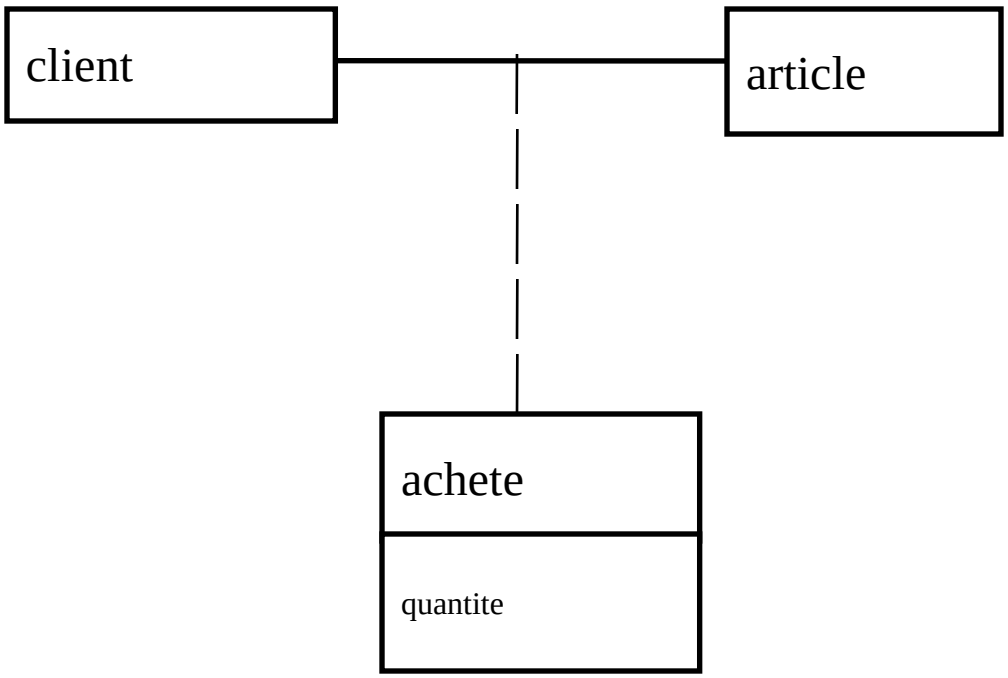


Classe-association

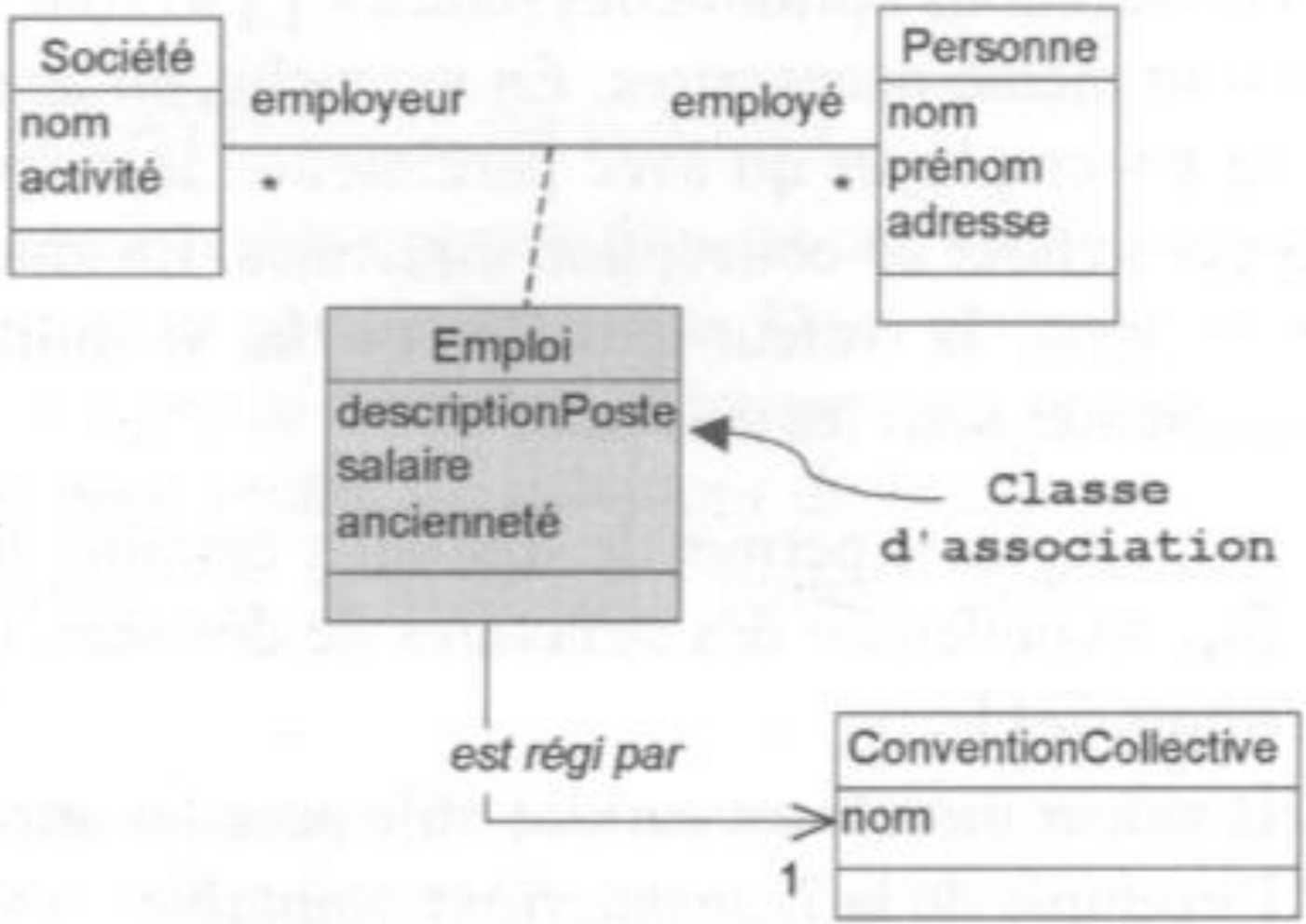
Équivalent à l'association ternaire :



Autre exemple :

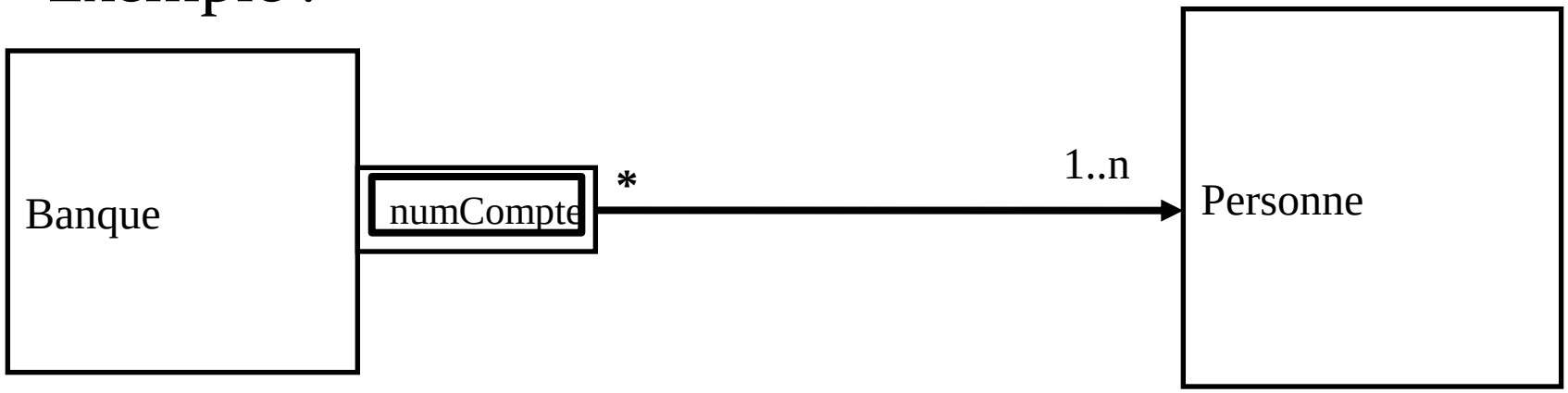


Autre exemple :



Association qualifiée

- Elle met en relation deux classes relativement à un attribut spécifique, appelé « clé ».
- Exemple :



Une banque peut avoir 0 ou plusieurs clients. Ce lien (s'il existe) se fait à travers le numéro de compte du client. Un client peut avoir 0 ou plusieurs comptes.

Contraintes sur les associations

- Les contraintes sont des expressions écrites dans un certain formalisme. Elles précisent le rôle ou la portée d'un élément de modélisation, c-à-d qu'elles permettent d'étendre ou de préciser sa sémantique.

- Par exemple, elles peuvent restreindre le nombre d'instances visées dans une association,
- Le formalisme des contraintes peut être le langage naturel, ou le langage graphique (texte entre accolades).

UML n’impose pas le langage dans lequel la contrainte doit être écrite. L, on utilise syntaxe appelée OCL est proposée par UML. Le langage OCL (Object Constraint Language) est partie prenante d'UML et permet d'exprimer des contraintes sous forme d'expressions booléennes qui doivent être vérifiées par le modèle, mais son utilisation n'est pas imposée.

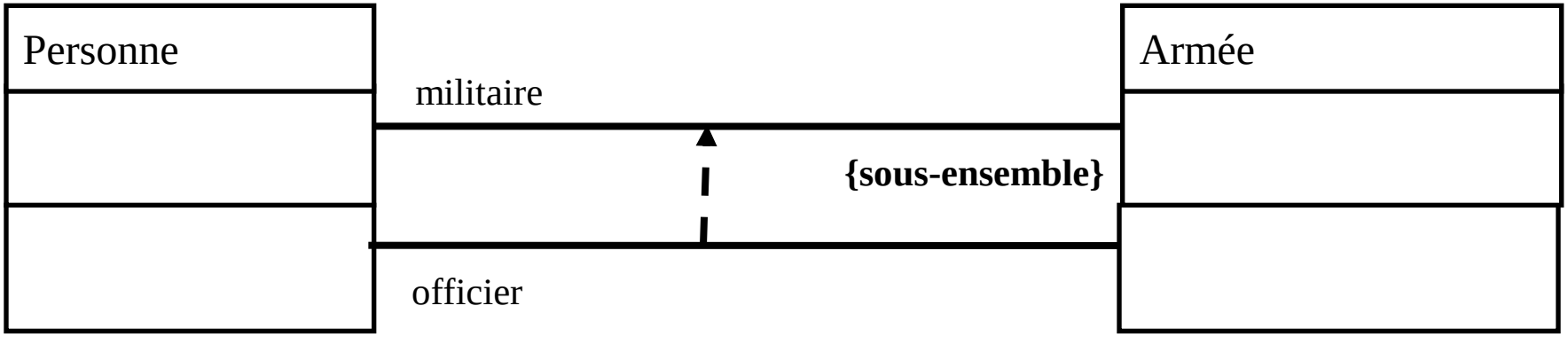
- ➔ tendre vers l'exhaustivité du code lors d'une génération automatique.

Sur les graphiques UML, une contrainte est représentée par un texte entre accolades ({}).

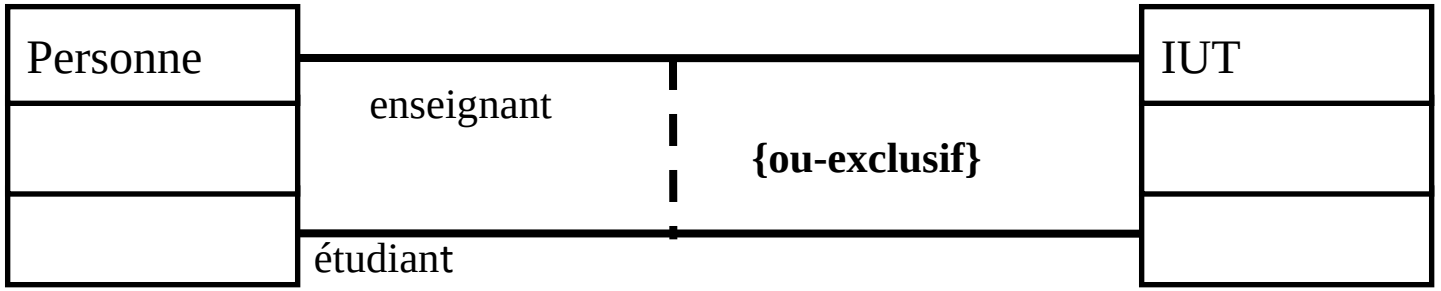
Quelques contraintes

1. Contrainte {ordonnée} : c'est une relation d'ordre qui décrit les objets
2. Contrainte {sous-ensemble, incomplete} : c'est une collection qui est incluse dans une autre collection
3. Contrainte {ou-exclusif, disjoint} : pour un objet donné, une seule association est valide

Exemples

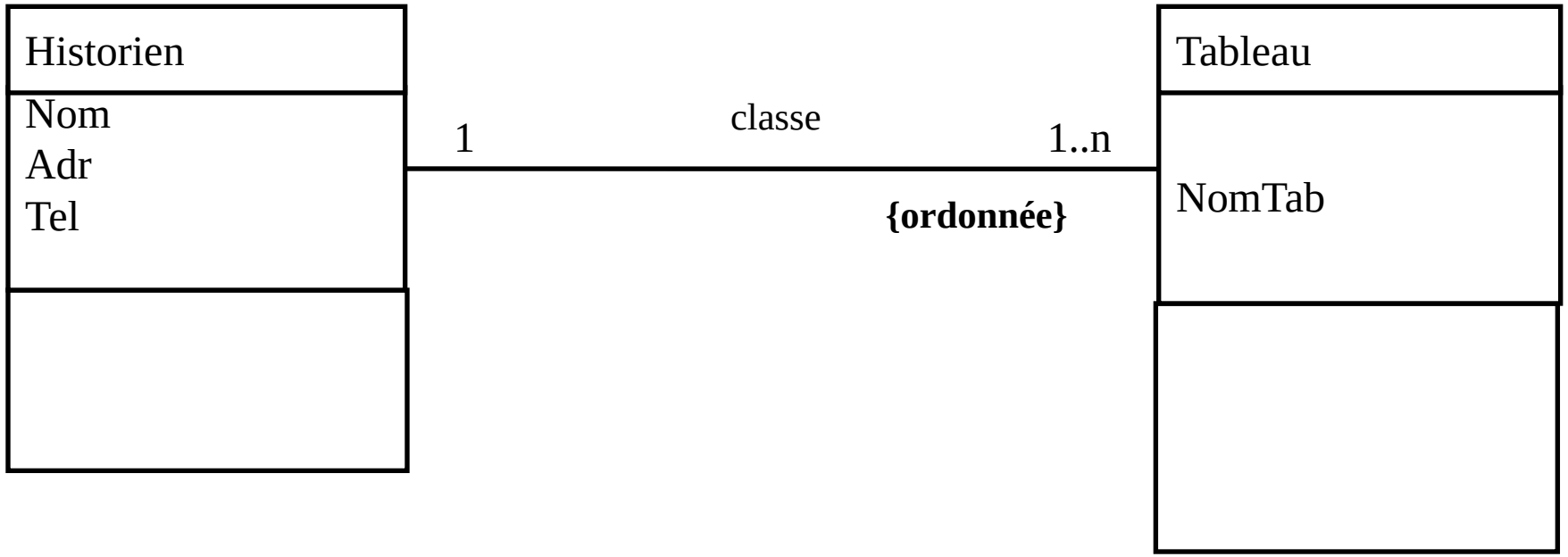


Une armée est formée de militaires. Parmi ces militaires, il y a des officiers (sous-ensemble de militaires).

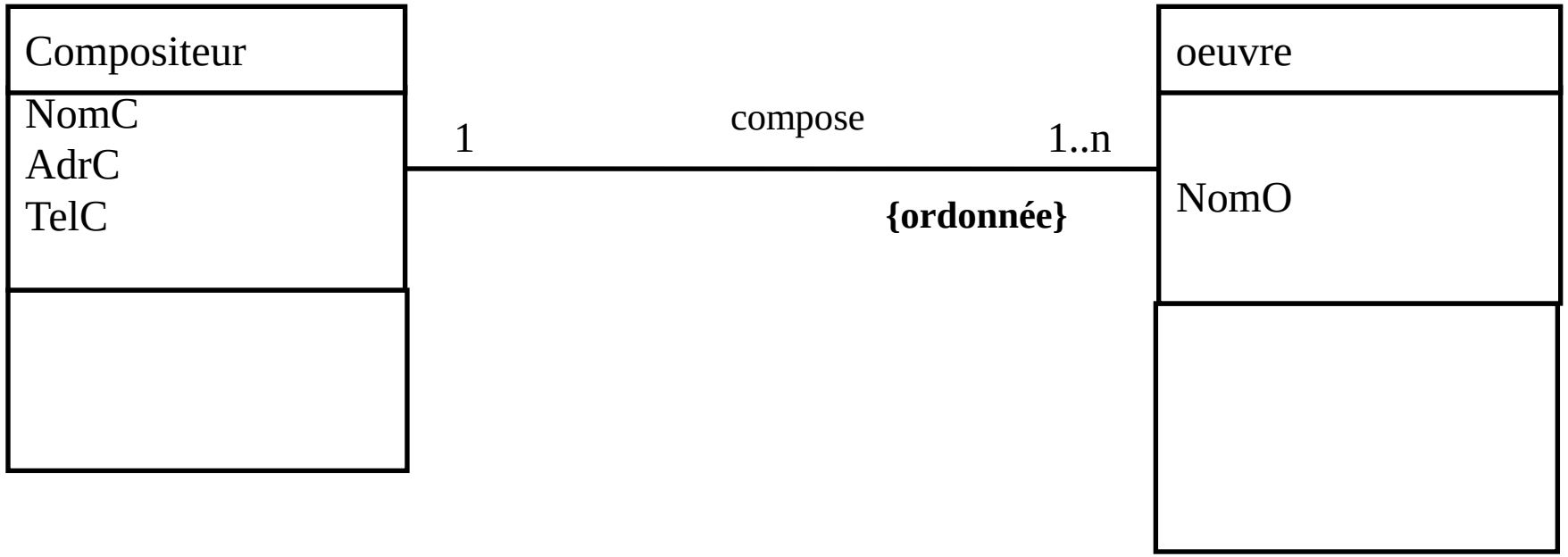


Une personne est dans un IUT avec pour rôle, soit étudiant, soit enseignant

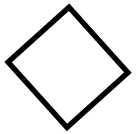
Un historien classe des tableaux de peintures, de manière ordonnée



On souligne que les œuvres sont composées dans l'ordre.



Deux associations particulières : agrégation et composition

L ’agrégation : représentée graphiquement par un losange vide 

- L'agrégation est une association **non symétrique**, qui exprime un couplage fort et une relation de subordination entre un composé et ses composants .
  - Elle représente une relation de type **ensemble /éléments**"

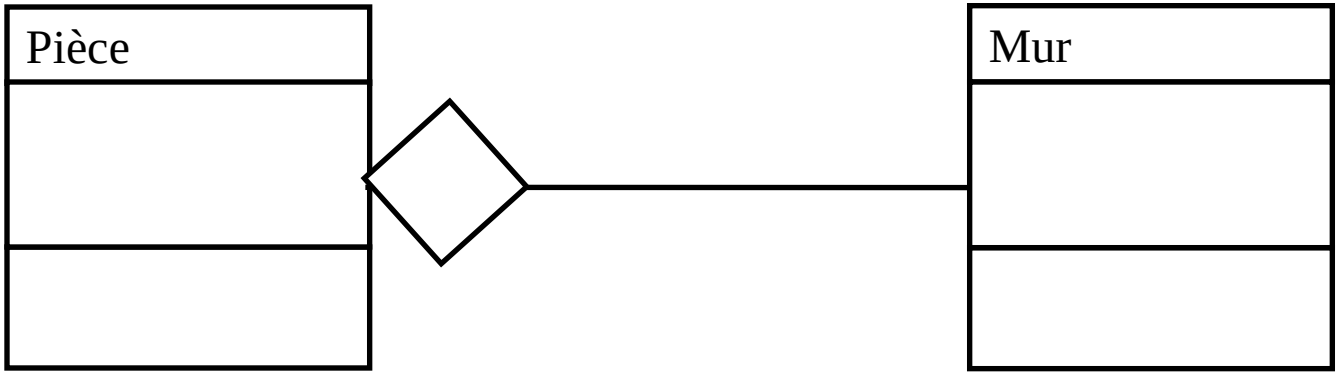
- A un même moment, une instance de l’élément agrégé peut être associée à plusieurs instances d'autres classes, c-à-d que **l'élément agrégé peut être partagé**

- Les cycles de vies de l’élément agrégé et les agrégats sont indépendants, c-à-d qu’une instance «ensemble» ou «agrégé» peut exister sans élément (et inversement).

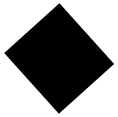
- Une agrégation peut *notamment* exprimer le fait :
  - qu'une classe (un "élément") fait partie d'une autre ("l'ensemble"),
  - ou qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre (l'élément),
  - ou qu'une action sur une classe, entraîne une action sur une autre (l'élément).

Exemple

Une pièce est composée de murs. Si on détruit la pièce, il peut toujours rester des murs (qui appartiennent à une autre pièce, par exemple)

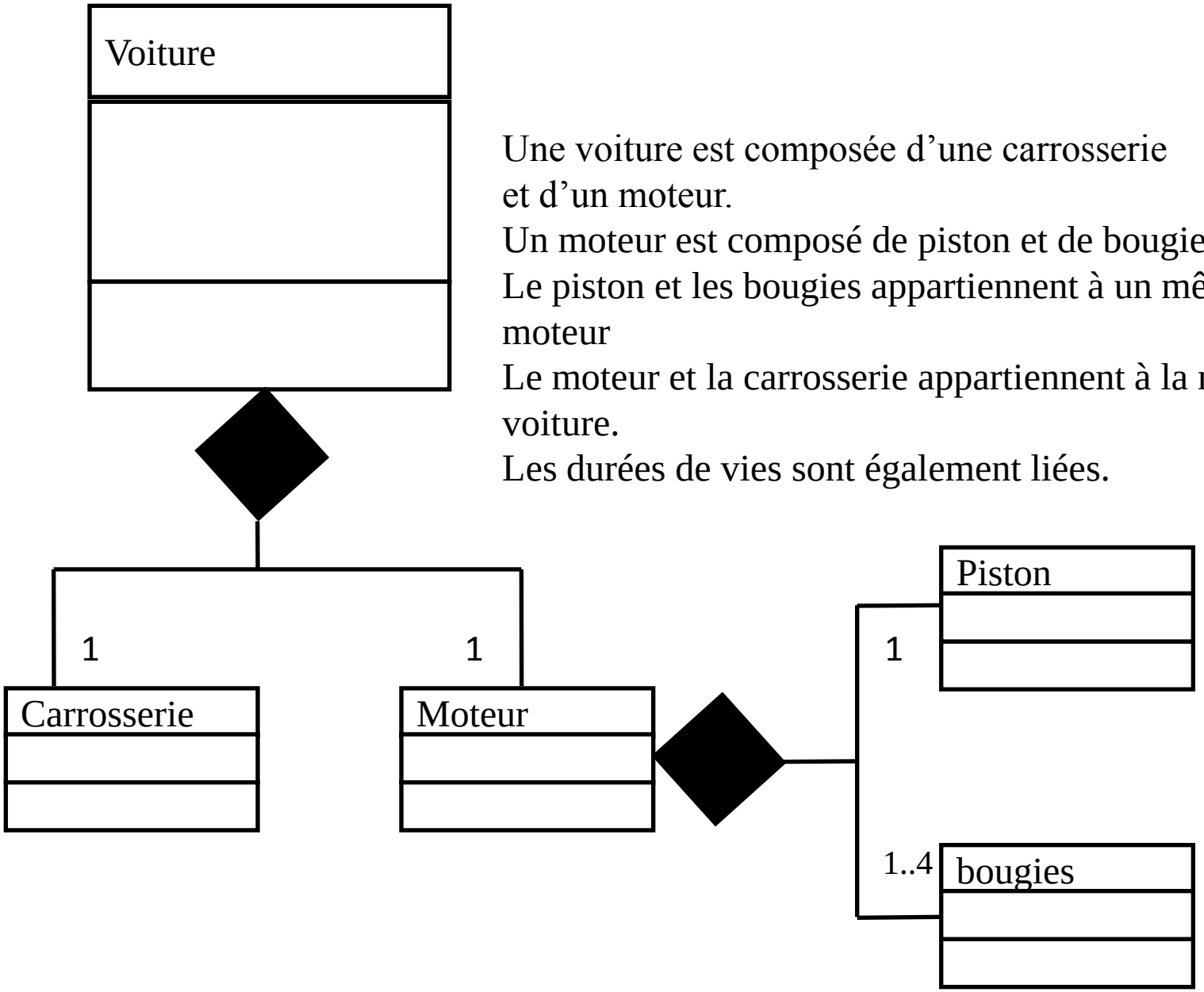


Composition : graphiquement, c’est un losange plein



1. Derrière cette notion, est sous-jacente la notion de **contenance physique** (exemple : Voiture avec moteur et roues)
2. Elle représente une relation de type "**composé / composant**"
3. Les **cycles de vies** des composants et du composé sont **liés** : si le composé est détruit (ou copié), ses composants sont détruits (ou copiés) également.
4. A un même moment, une instance de composant **ne peut être liée qu'à un seul composé**

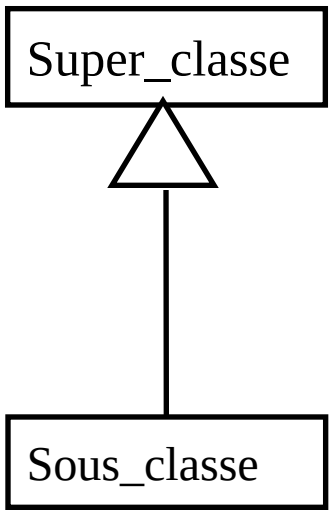
Exemple



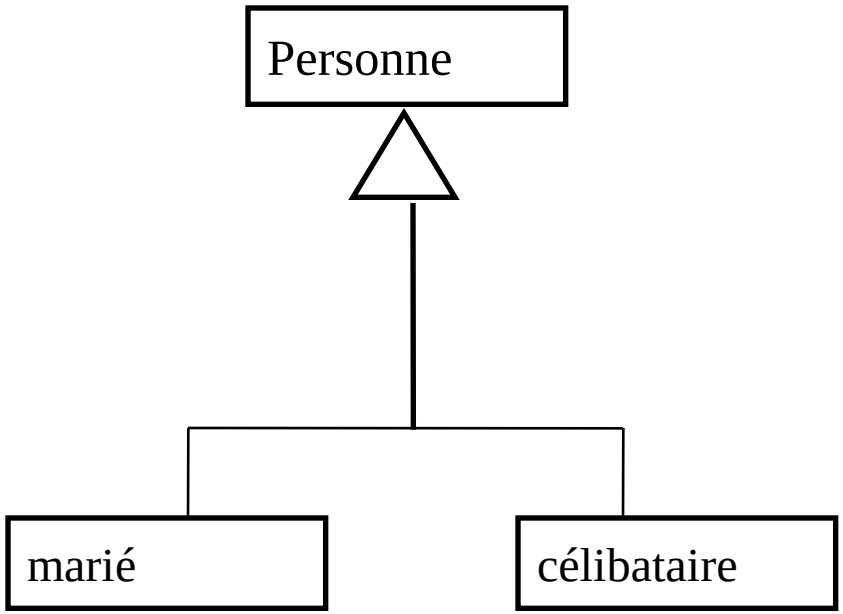
Une voiture est composée d'une carrosserie et d'un moteur.
Un moteur est composé de piston et de bougies.
Le piston et les bougies appartiennent à un même moteur
Le moteur et la carrosserie appartiennent à la même voiture.
Les durées de vies sont également liées.

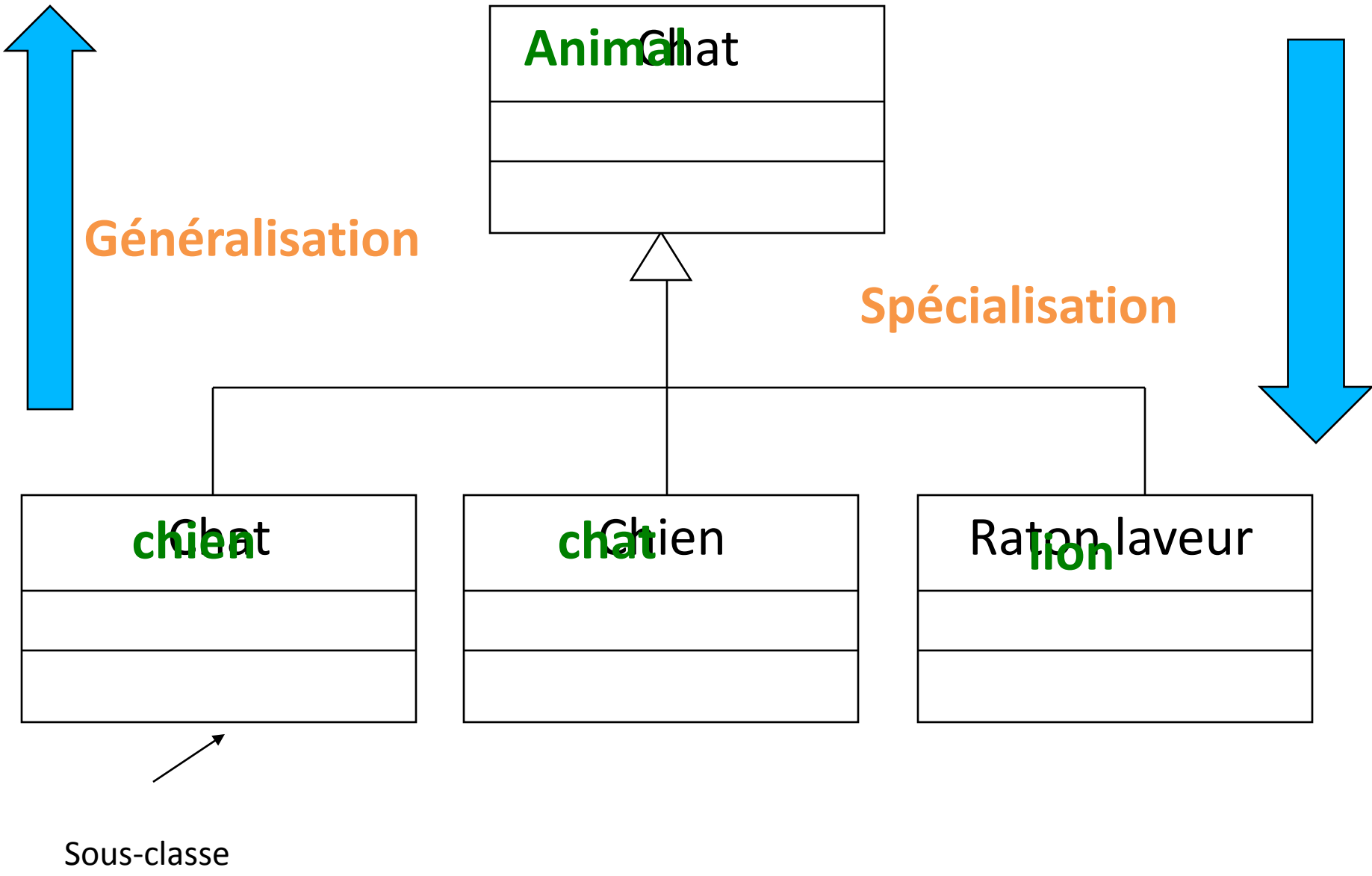
Héritage simple

Graphiquement :

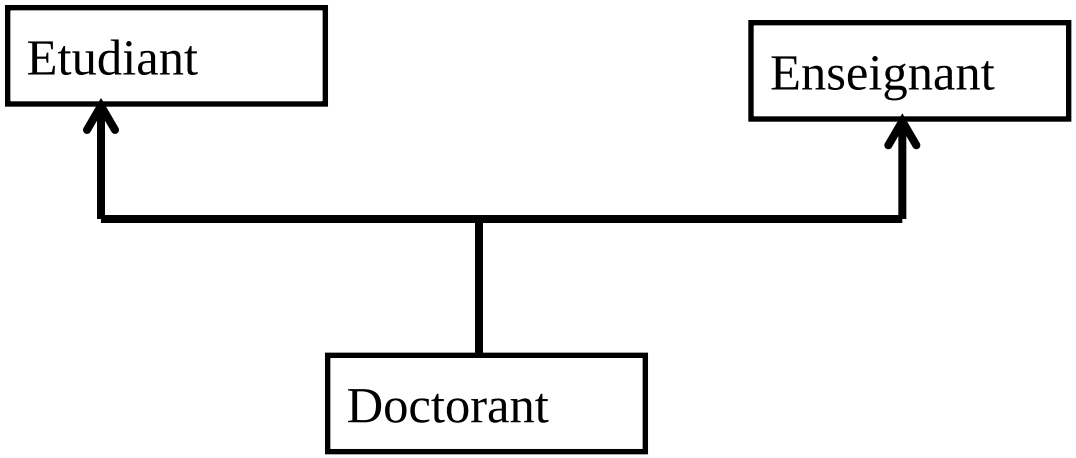


Exemple :



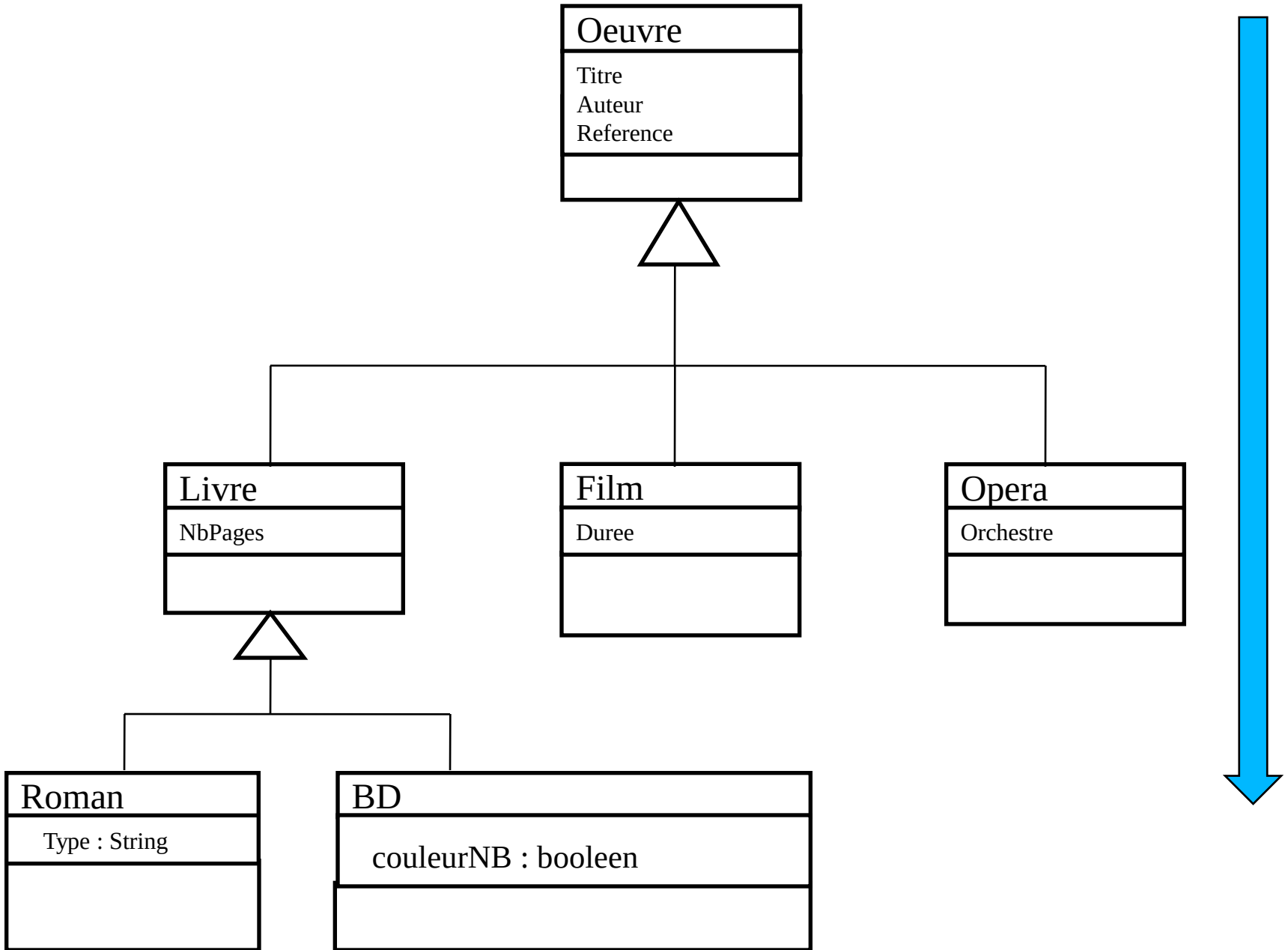

Animal
Chat
Généralisation
Spécialisation
chien
Chat
chat
Chien
lion
Raton laveur
Sous-classe

Héritage multiple

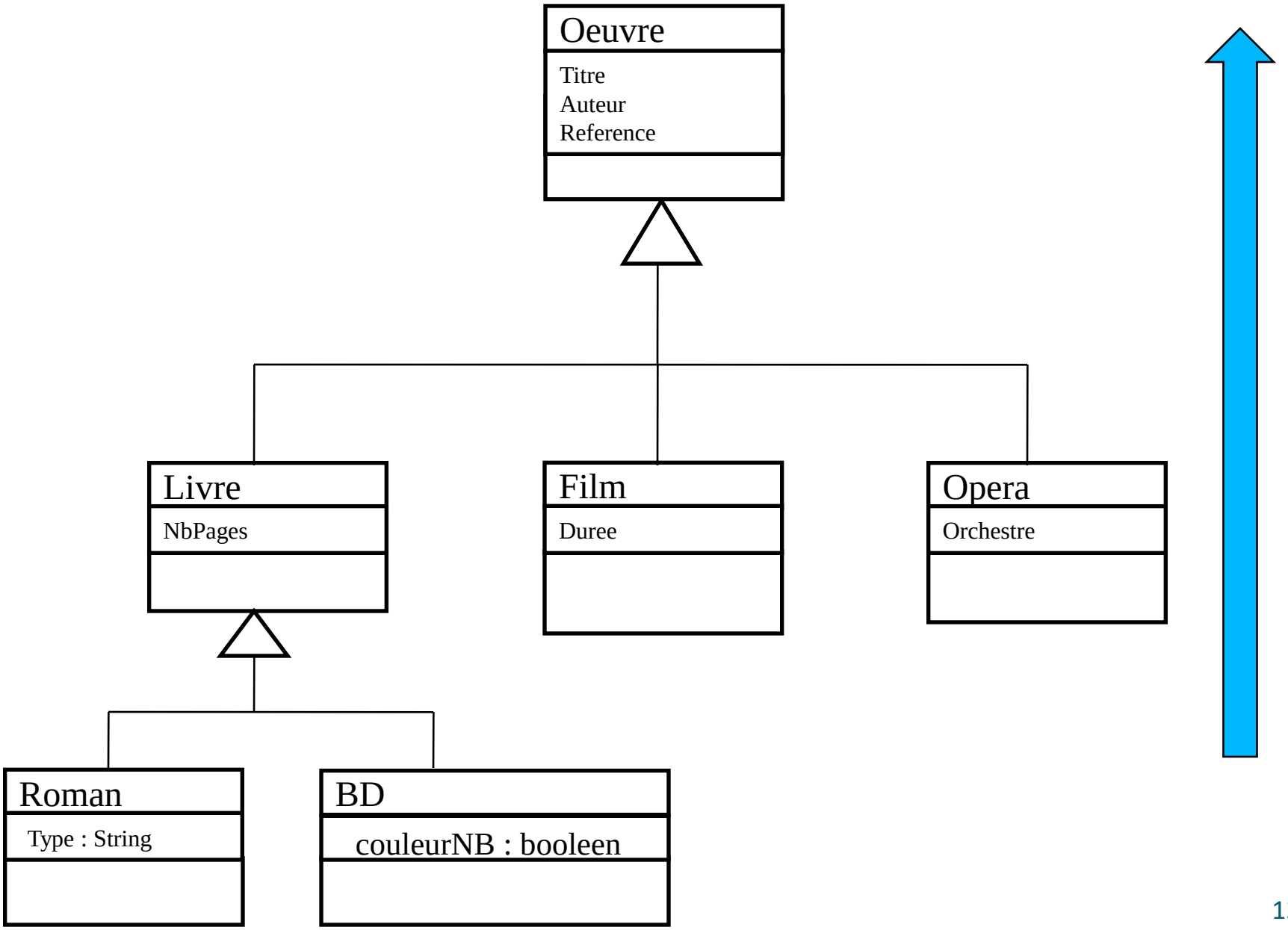


Un doctorant est un étudiant, mais peut également être un enseignant, donc la classe doctorant hérite des 2 classes.

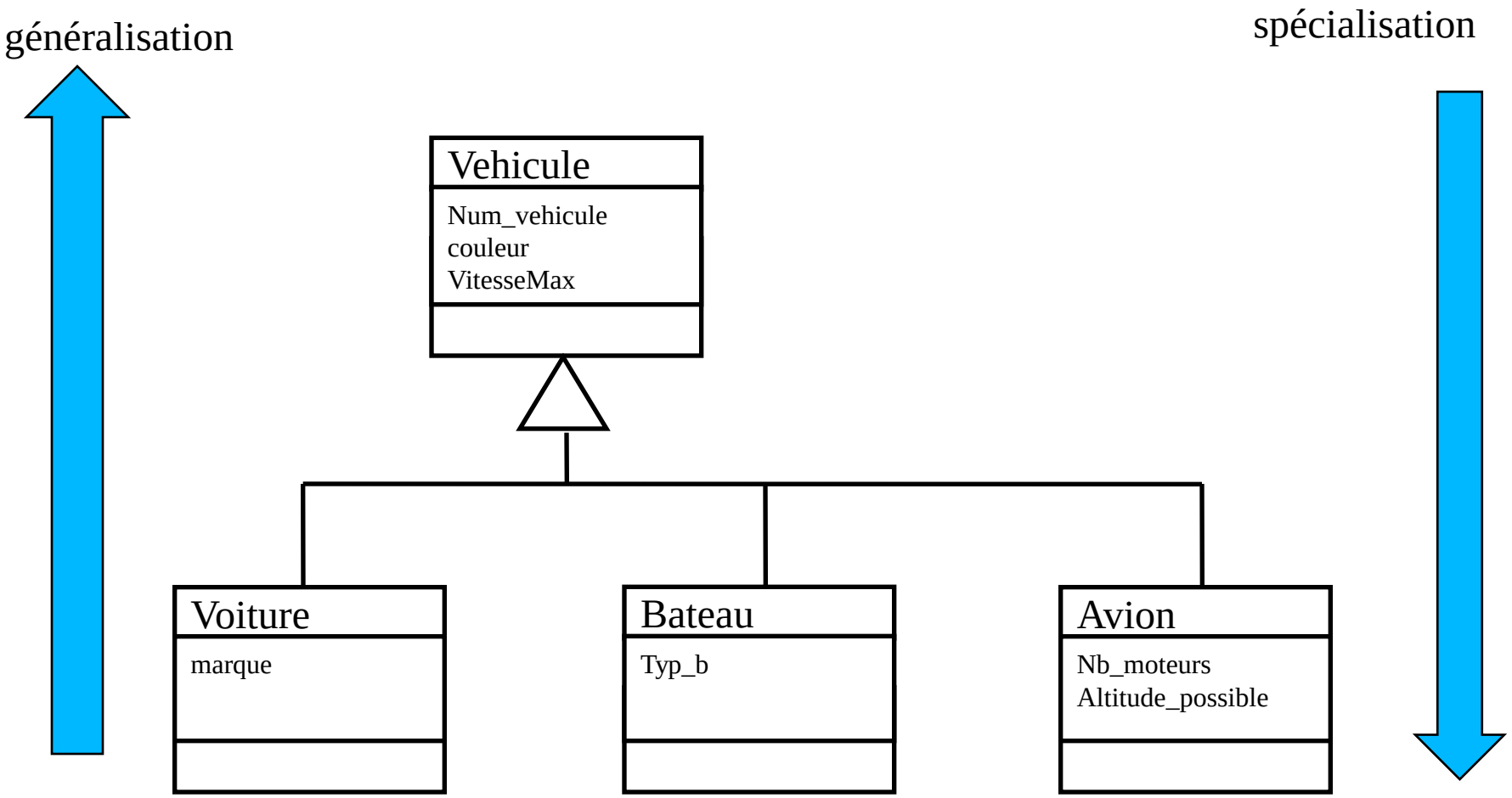
Spécialisation



Généralisation



Autre exemple : généralisation-spécialisation



Diagrammes d’objets

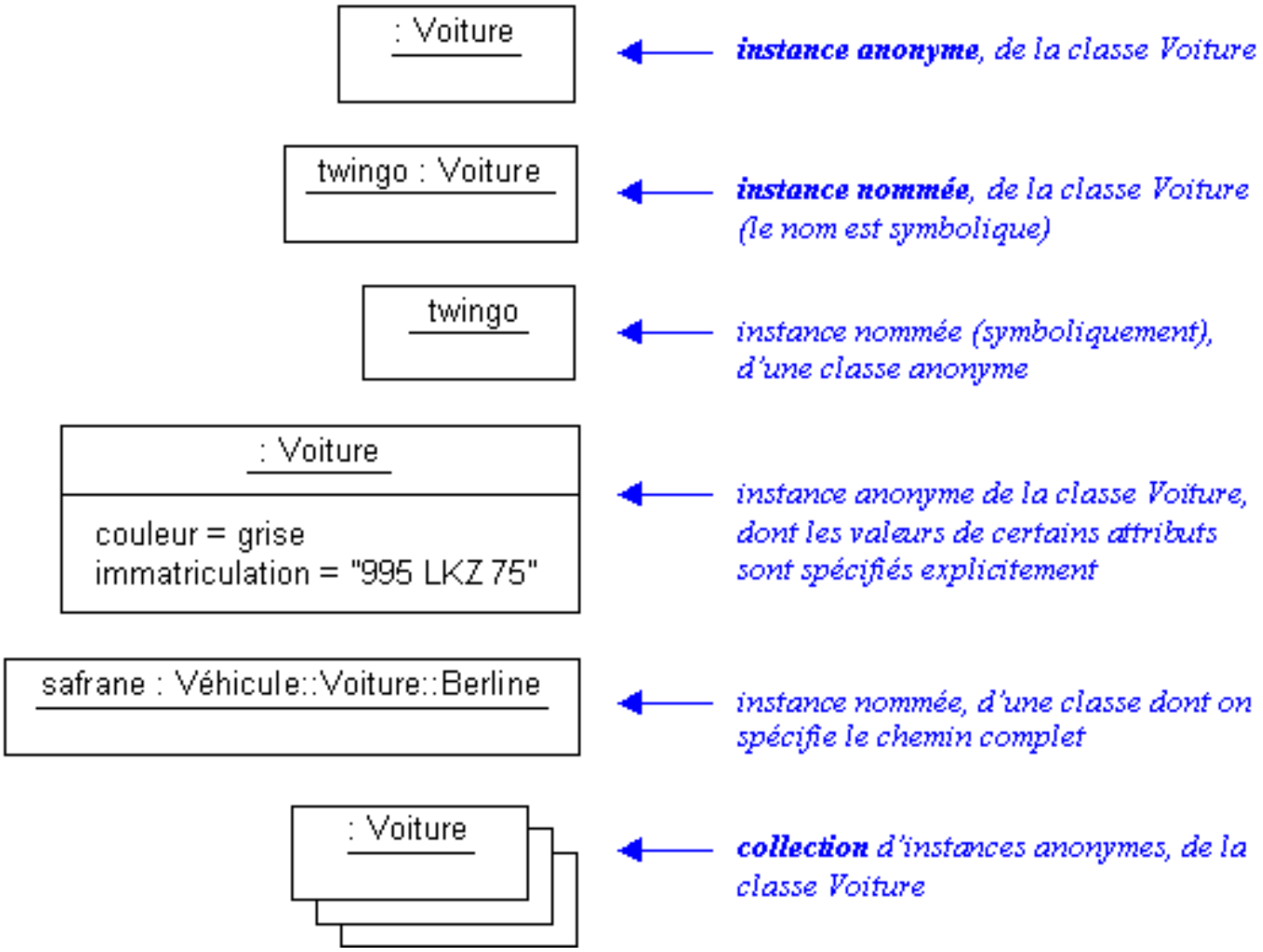
Diagramme d'objets

- Ils représentent un ensemble d’objets et leurs liens
- Ils représentent des cas particuliers d’occurrences de classes avec leurs liens.
- Ils présentent la vue de conception d’un système, à partir de cas réel ou de prototypes.
- C’est une instance (une réalisation) d’un diagramme de classes.

Différentes représentations des objets

:Voiture	Est une instance anonyme de la classe voiture
Renaul 5:Voiture	Instance nommée de la classe voiture
Renault 5	Instance nommée d'une classe anonyme
Renault 5:Voiture N_imm = 'AV 434 CC' Puissance = 8	Instance nommée de la classe voiture et Spécification des attributs
golf:Vehicule::Voiture::Berline	Spécification du chemin complet
:Voiture	Ensemble d'instances de la classe voiture

Autre exemple



Exemple

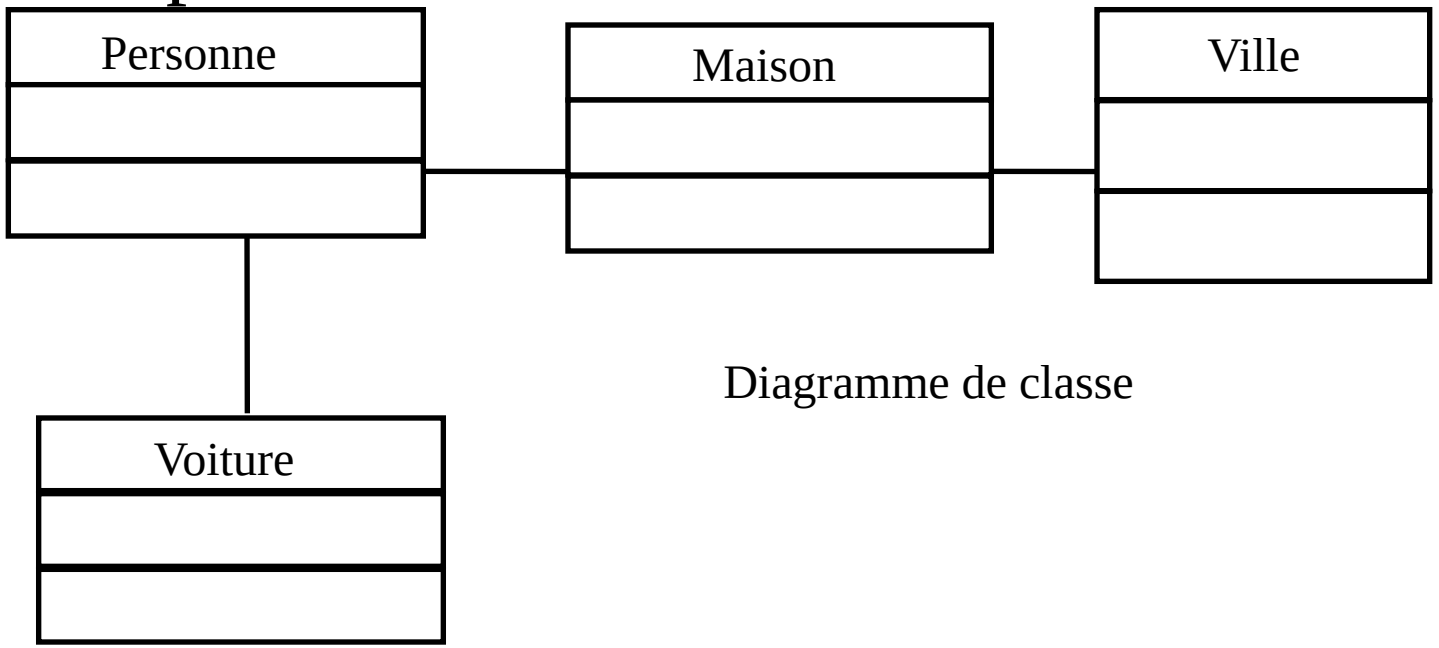


Diagramme de classe

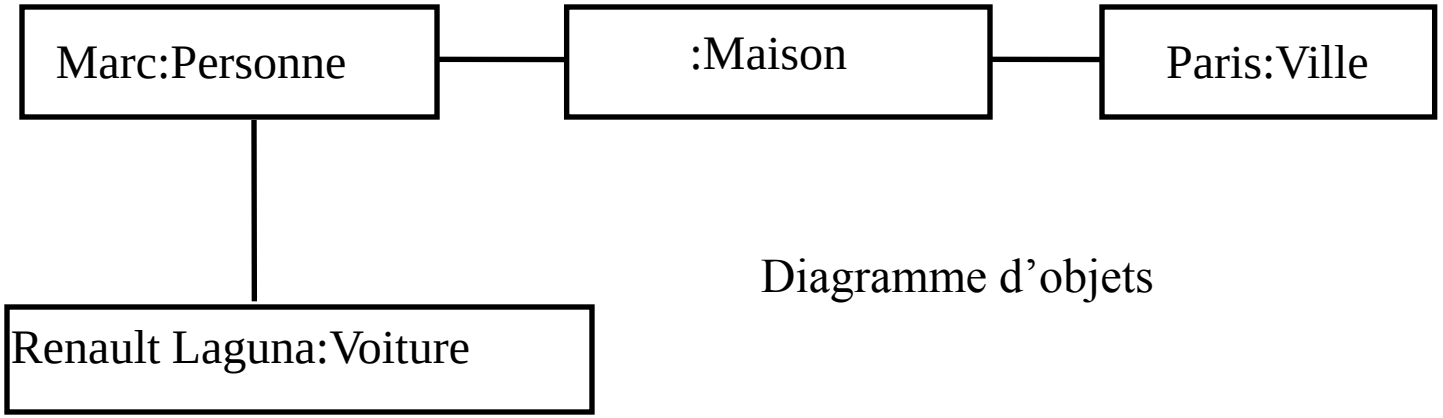


Diagramme d'objets

Diagrammes d’objets

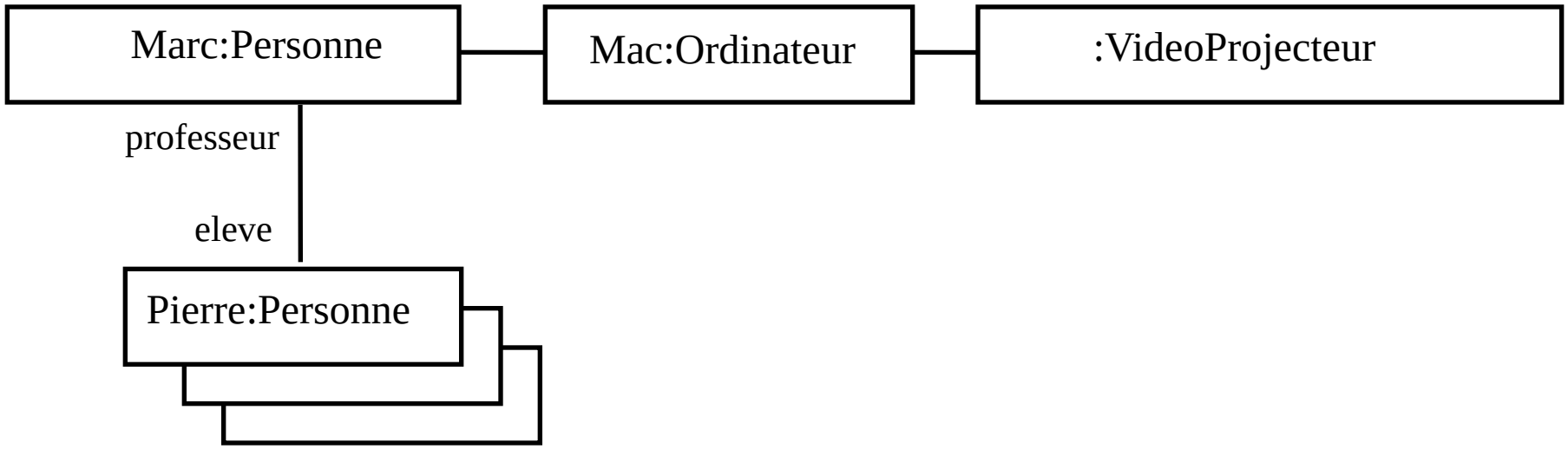
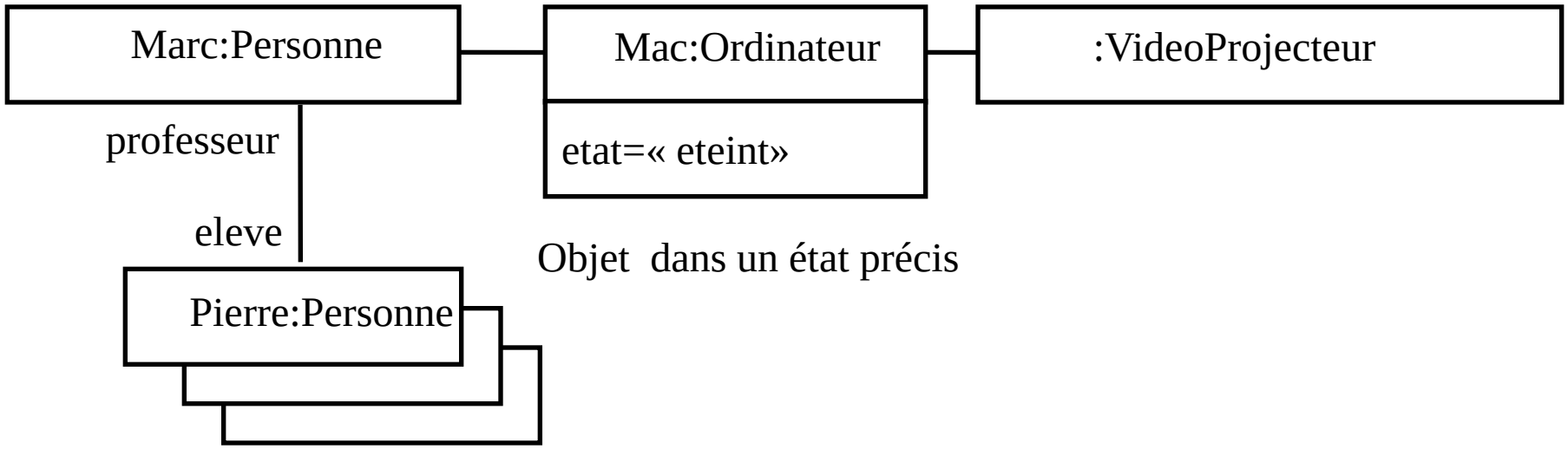
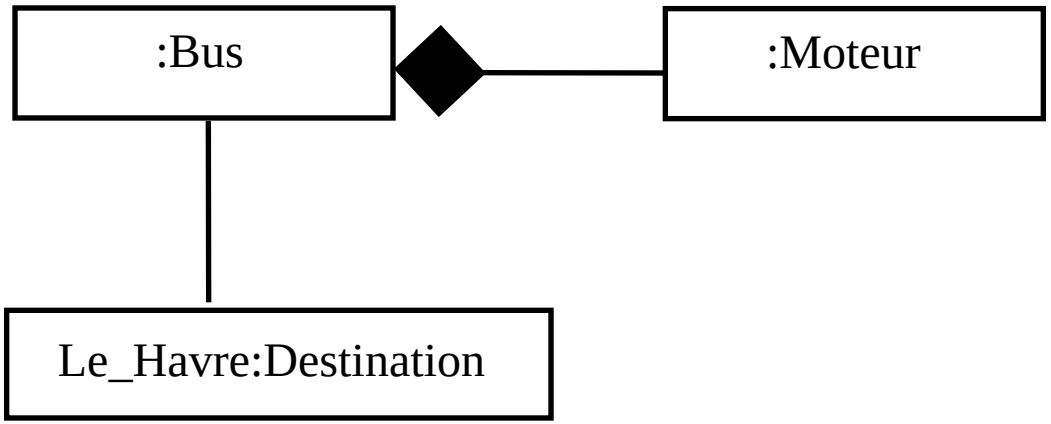


Diagramme d’objets



Objet dans un état précis

Diagramme d’objets



Diagrammes d’activités