GÉNIE LOGICIEL ORIENTÉ OBJET (GLO-2004)
ANALYSE ET CONCEPTION DES SYSTÈMES ORIENTÉS OBJETS (IFT-2007)

Automne 2016

Révision examen 1
Module 1 à 11 (inclusivement)

Martin.Savoie@ift.ulaval.ca

Bachelier Génie logiciel, Chargé de cours,
département d'informatique et de génie logiciel

UNIVERSITÉ LAVAL

Avertissement

- Ceci est une révision pour l'examen 1
  - L'examen porte sur les modules 1 à 11 inclusivement
- Ceci ne couvre pas l'entièreté des éléments vue durant la session

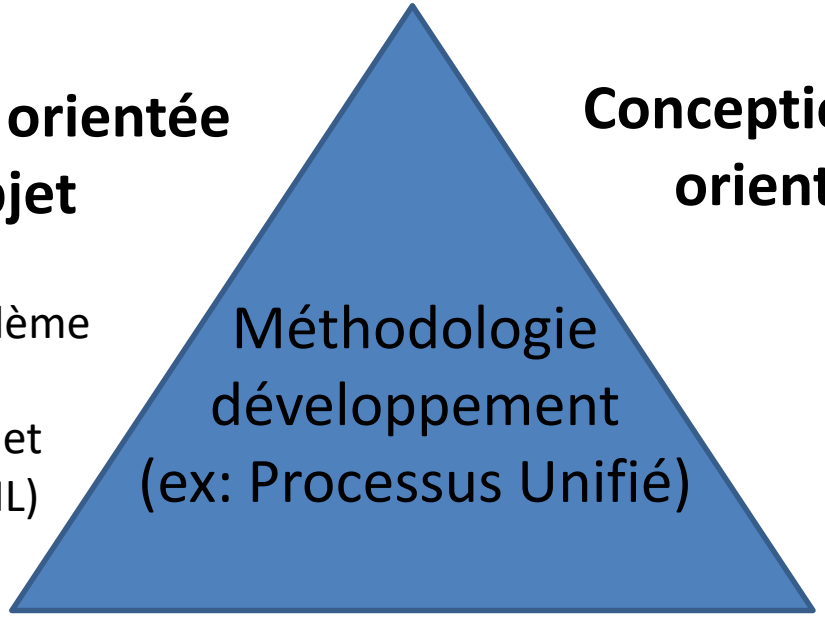
Module 2

- Méthodologie
- Processus de développement

Génie logiciel orienté objet

Analyse orientée objet

- Comprendre le problème
- Décrire la situation à l'aide de documents et diagrammes (ex: UML)



Conception (design) orienté objet

- Concevoir une solution informatique
- Tracer des plans (plus ou moins détaillés) sous la forme de documents et diagrammes (ex: UML)

Programmation orientée objet

Mettre en œuvre la solution à l'aide d'un langage (ex: Java)

UNIVERSITÉ LAVAL

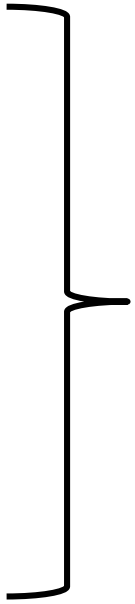
Méthodologie / processus de développement logiciel

- Précise une méthodologie à suivre pour développer un logiciel
- Spécifie les étapes / activités à réaliser et les livrables (« artefacts ») à produire
- Décrit notamment quand et comment procéder à l'analyse, à la conception, à la programmation, etc..

Le processus **doit permettre de gérer**

- la ***complexité*** du ***problème***;
- la ***complexité*** de gestion du ***processus*** de conception du logiciel par une équipe (plusieurs solutions sont possibles et il faut choisir la meilleure);
- la ***complexité*** associée à la ***flexibilité*** offerte par le logiciel;

- Plusieurs approches de développement de logiciel existent:
  - Waterfall (“en cascade”) – Méthode d’ingénierie classique
  - Spirale
  - Unified Process (UP)
  - Méthodes agiles (Agile Software Development Methods)
  - Méthodes “extrêmes” (Extreme Programming Methods)

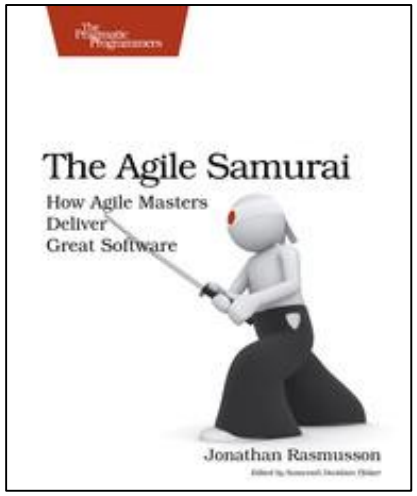


Waterfall

- Approche classique d'ingénierie formée d'étapes successives
  - Analyse, Conception, Implantation, Integration, Test, Transition
- Cette méthode est très rigide et exige que toutes les spécifications soient complètes avant que la conception et l'implantation commencent
- Elle est applicable avec succès pour de petits projets, mais mène à des échecs cuisants sur des projets d'envergure s'étalant sur de longues périodes de temps

Méthodes itératives

- Approche impliquant de courtes itérations (ex: deux semaines) incluant toutes les étapes de développement logiciel:
  - Requirements, design, implantation, intégration, test et acceptation par le client
- Favorise le travail en équipe en mettant l'accent sur la collaboration entre les membres de l'équipe plutôt que sur la documentation
- Favorise la participation étroite du client au développement du produit
- Exemple: Méthode agile



http://books.openlibra.com/pdf/AgileSamurai.pdf

Avantages d'une méthode itérative

- Réduction des risques
  - Identifier plus tôt les problèmes potentiels
  - Évaluer rapidement la vitesse de progression
- Développer une architecture robuste
  - Évaluation rapide de l'architecture: amélioration, correction possible
- Gérer des besoins évolutifs (et de façon évolutive)
  - Les utilisateurs (le client) donne une rétroaction constante
  - Répondre à ces rétroactions nécessite un changement incrémental (plutôt qu'une refonte totale du système).
- Permettre le (les) changement(s)
  - Adapter le système aux problèmes, aux besoins, aux requis
- Apprendre tôt dans le processus de développement
  - Tout le monde obtient une compréhension du domaine (des processus relatif au domaine) tôt.

Processus Unifié (PU)

- Processus de développement itératif reposant sur UML et proposé à l'origine par Booch, Rumbaugh et Jacobson (livre publié en 1999)
- De nouvelles versions (Rational Unified Process) ont été publiées depuis (propriété de IBM)
- Le processus original (UP) peut être vu comme un tronc commun à plusieurs méthodes itératives

UP: Plusieurs itérations, à chaque itération on fait du travail relié à plusieurs disciplines

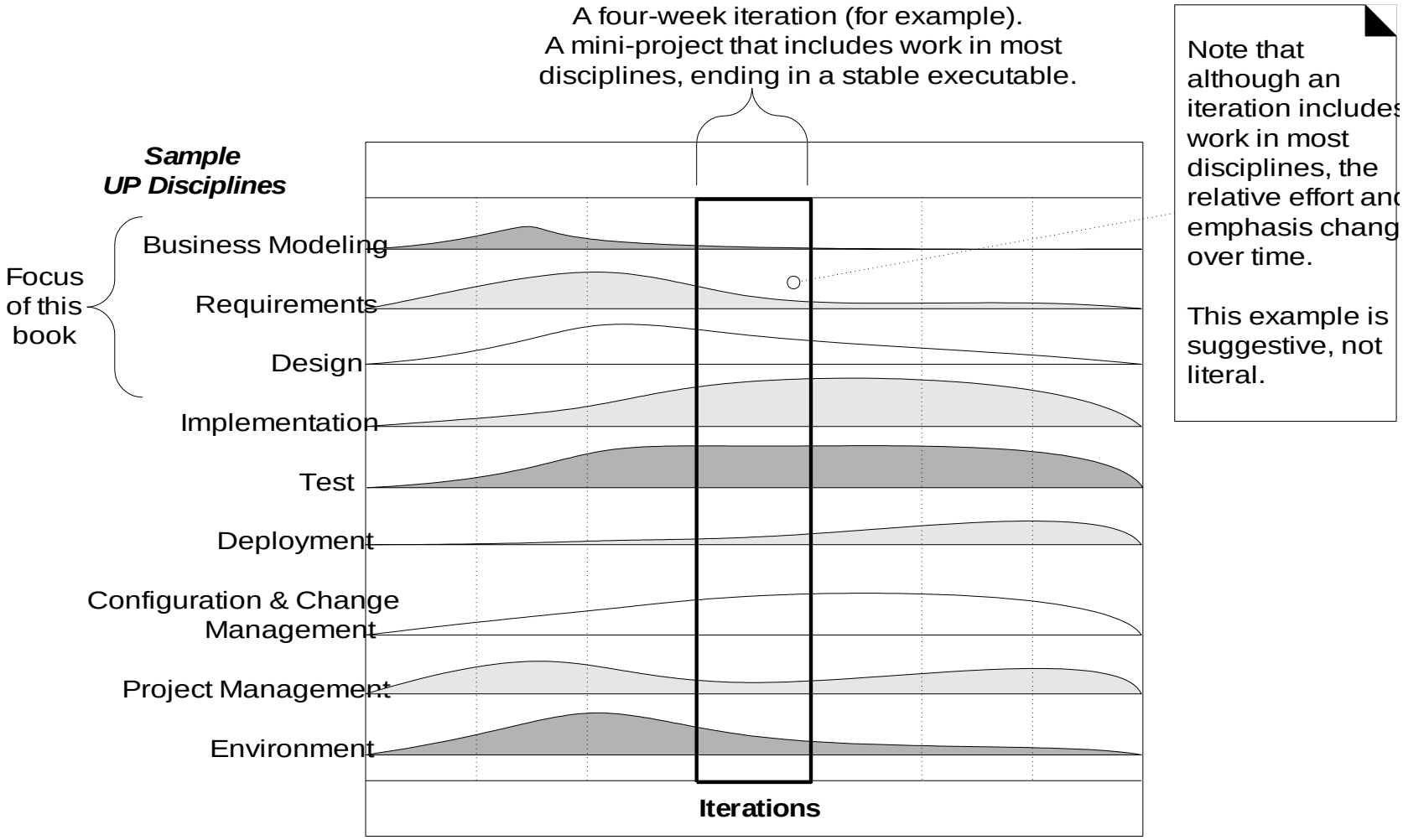


Fig. 2.7

Dans UP, les itérations sont regroupées en grandes phases

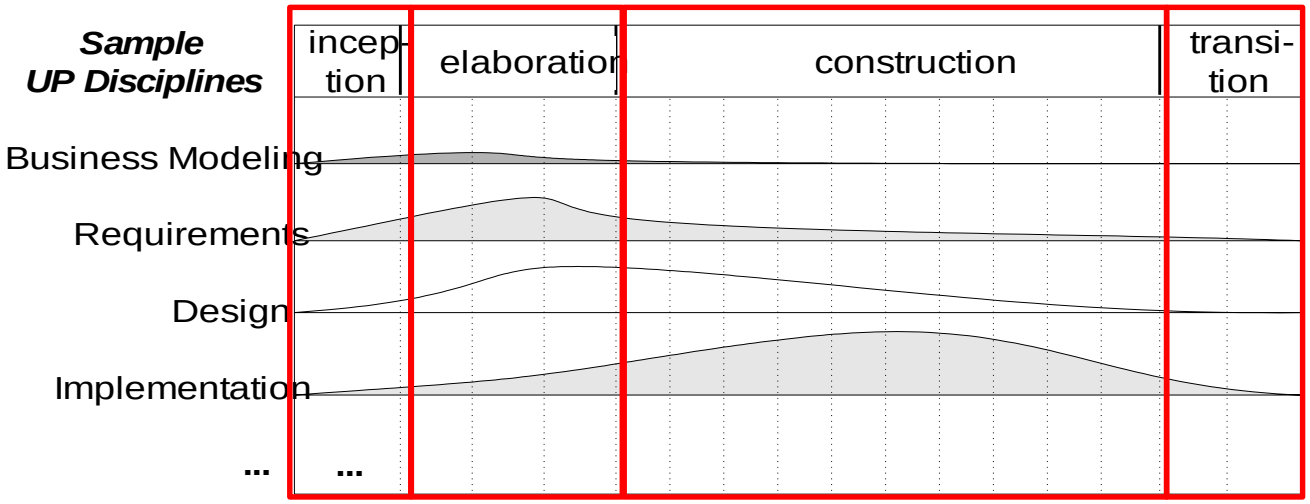
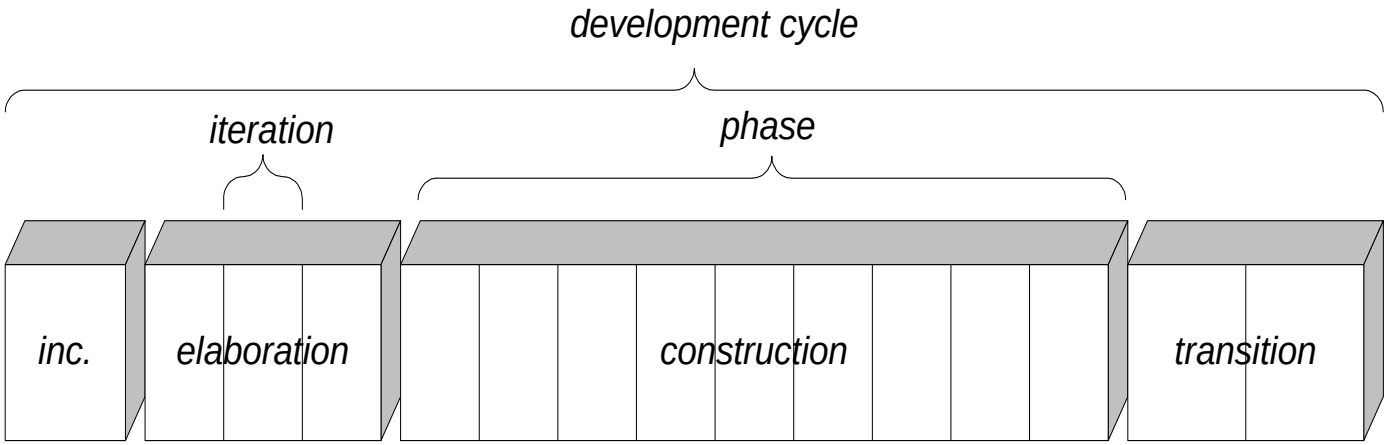


Fig. 2.6



4 grandes *phases*

Chaque phase est composée d'*itération**S.***

Dans chaque itération, on réalise du travail associé à plusieurs *activités* / disciplines.

Les *artefacts* sont les livrables produits.

Discipline	Artifact Iteration	Incep. Il	Elab. EL.En	Const. CL.Cn	Trans. T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	*s*	r		
	Vision	*s*	r		
	Supplementary Specification	*s*	r		
	Glossary	*s*	r		
Design	Design Model SW Architecture Document Data Model		s s s	r r	
Implementation	Implementation Model (code)		s	r	r

S: start
R: refine

	Activités (appelées *disciplines* dans le Processus Unifié)	**Modèles** et artefacts générés
Analyse	Modélisation domaine d'affaires / Business modeling / Modélisation métier *synonymes!*	**Modèle du domaine**: (1) diagramme de classe « conceptuel », (2) parfois un diagramme d'activités
	Analyse des besoins / Exigences / Requirements	(3) Énoncé de vision
		Modèle de cas d'utilisation / Use-case model : (4) diagramme des cas d'utilisation, (5) texte des cas d'utilisation, (6) diagramme de séquence système
		(7) Spécifications supplémentaires
		(8) Glossaire
	Design / Conception	**Modèle de conception / Design model** : (9) diagrammes de classes, (10) diagrammes d'interaction, (11) tout autre diagramme UML pertinent selon le contexte
	Implémentation	(12) Code
	…	

Module 3

- Phase de conceptualisation / inception
- Modèle des cas d’utilisation

Phase de conceptualisation / inception

- **But:**
  - Établir une définition du projet, les objectifs
  - Établir la faisabilité
  - Décider si on doit pousser plus loin (décider si on va réaliser le projet)
- **Autrement dit:**
  - Identifier le besoin
  - Rédiger un document décrivant la vision du projet / modèle d'affaires
  - Déterminer si le projet est-il réalisable
  - Peut-on acheter le système plutôt que le construire?
  - Estimer grossièrement les coûts
  - Échéancier approximatif
  - Go/no Go

Artefacts de la phase de conceptualisation/inception

Discipline	Artifact Iteration	Incep. Il	Elab. EL.En	Const. CL.Cn	Trans. T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	*s*	r		
	Vision	*s*	r		
	Supplementary Specification	*s*	r		
	Glossary	*s*	r		
Design	Design Model SW Architecture Document Data Model		s s s	r r	
Implementation	Implementation Model (code)		s	r	r

S: start
R: refine

	Activités (appelées *disciplines* dans le Processus Unifié)	Principaux **modèles** et artefacts générés
Analyse	Modélisation domaine d'affaires / Business modeling / Modélisation métier	**Modèle du domaine**: (1) diagramme de classe « conceptuel », (2) parfois un diagramme d'activités
	Analyse des besoins / Exigences / Requirements	(3) Énoncé de vision
		Modèle de cas d'utilisation / Use-case model : (4) diagramme des cas d'utilisation, (5) texte des cas d'utilisation, (6) diagramme de séquence système
		(7) Spécifications supplémentaires
		(8) Glossaire
	Design / Conception	**Modèle de conception / Design model** : (9) diagrammes de classes, (10) diagrammes d'interaction, (11) tout autre diagramme UML pertinent selon le contexte
	Implémentation	(12) Code
	…	

Besoins / requirements

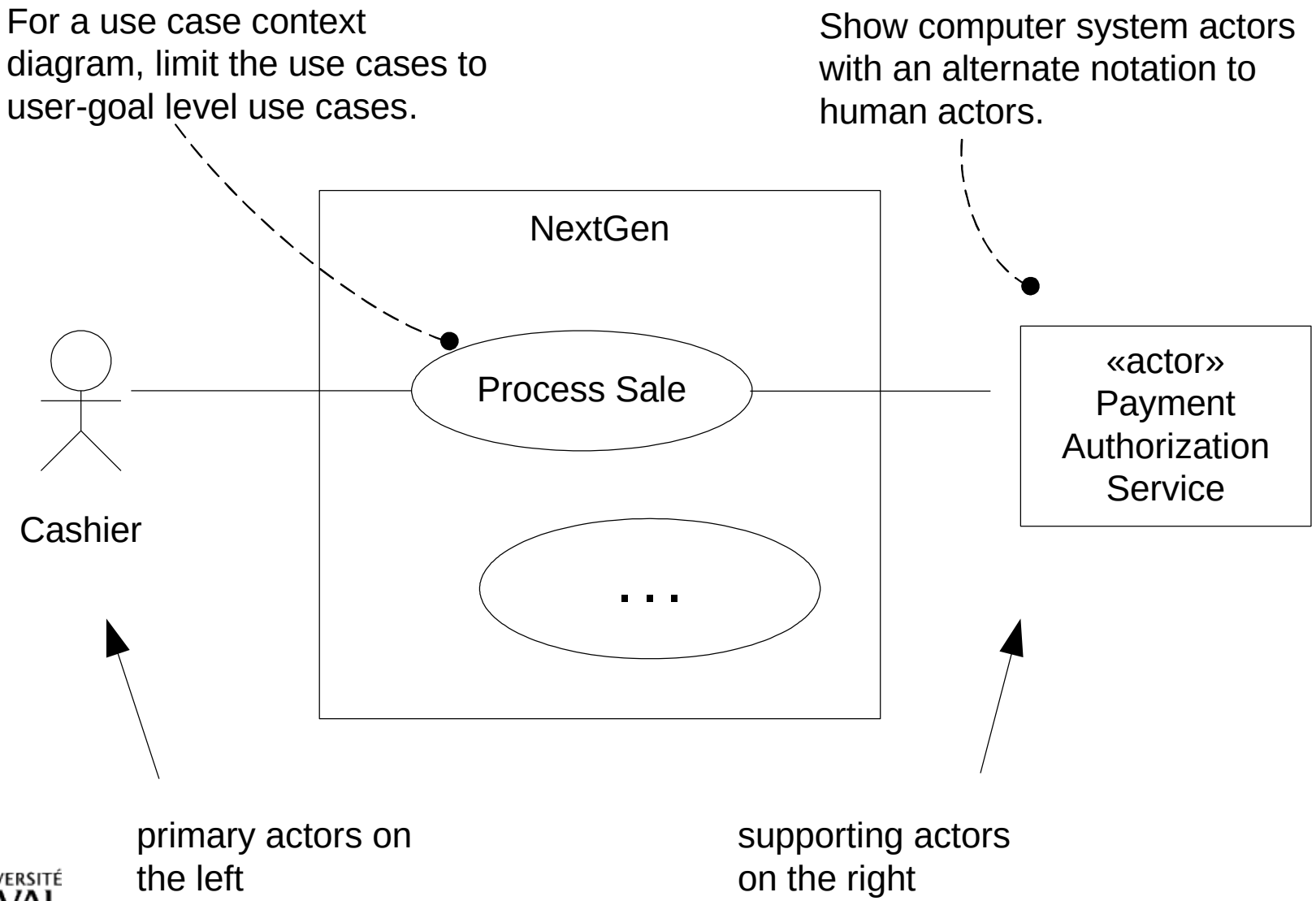
- **Besoins**: Conditions auxquelles un système (ou projet) doit satisfaire

- **Besoins fonctionnels**
  - Fonctionnalités
  - Répertoriées et décrites dans le **modèle des cas d'utilisation**

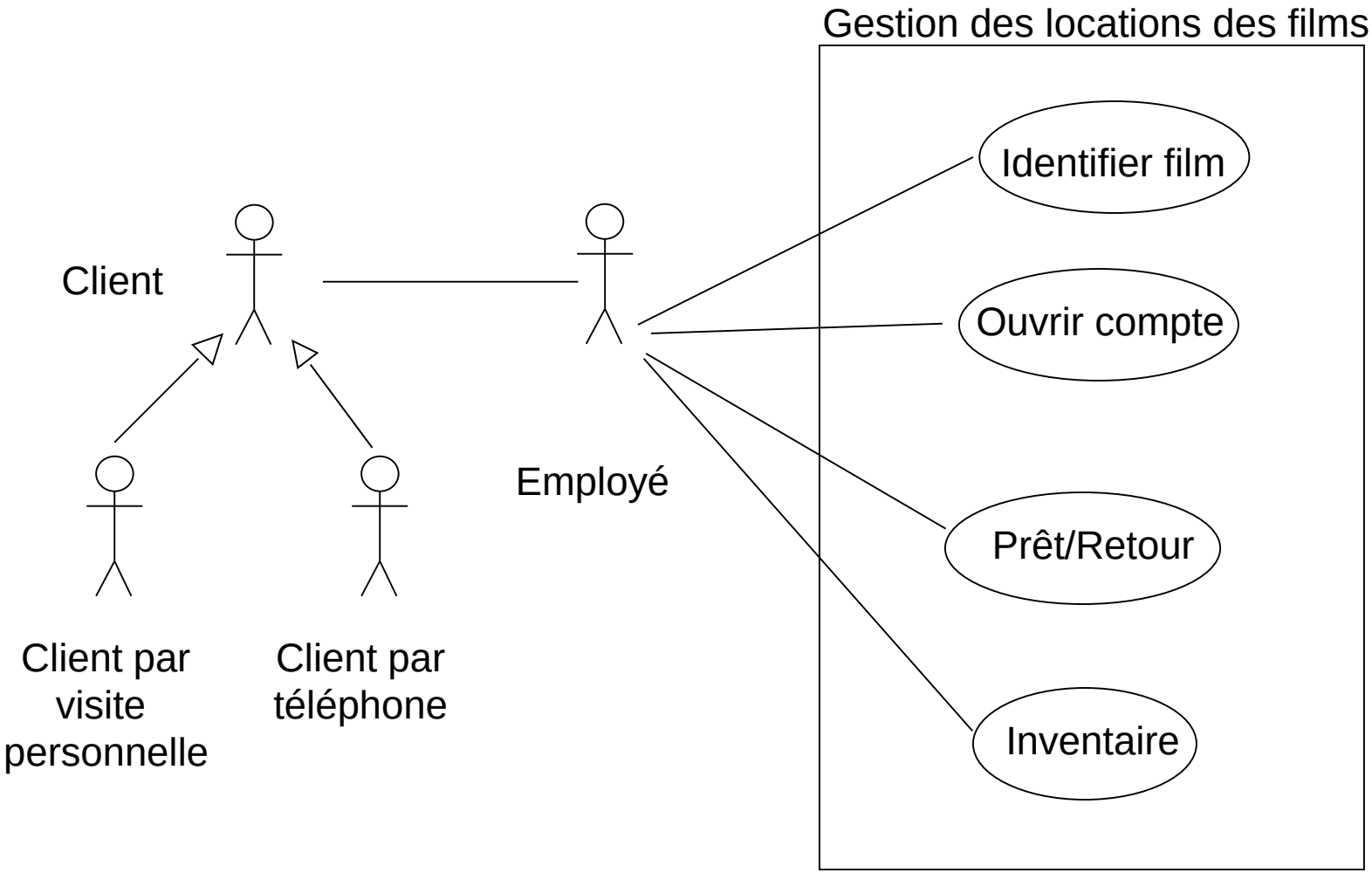
- **Besoins non fonctionnels (« URPS »)**
  - **U**sability (Help, documentation, …), **R**eliability (Frequency of failure, recoverability, …), **P**erformance (Response times, availability, …), **S**upportability (Adaptability, maintainability, …)

- **Rappel**: le processus unifié suppose une notion évolutive des besoins

Représenter les besoins fonctionnels à l'aide du diagramme des cas d'utilisation



Représenter les besoins fonctionnels à l'aide du diagramme des cas d'utilisation



Identifier les cas d'utilisation…

- Pour chaque acteur (primaires et secondaires), identifier ses buts (on suppose qu'une fonctionnalité du système est liée à chacun de ces buts)
- Définir un "cas d'utilisation" pour chacun de ces buts
- Réfléchir aux frontières du système
- Identifier les composantes / acteurs externes

Identifier les cas d'utilisation… pertinents

- Répondant à un but d'un utilisateur

- Permettant à l'utilisateur d'accomplir une tâche quelque chose ayant de la valeur dans le contexte de l'entreprise ("test du patron")

Exemple – Include & Exclude

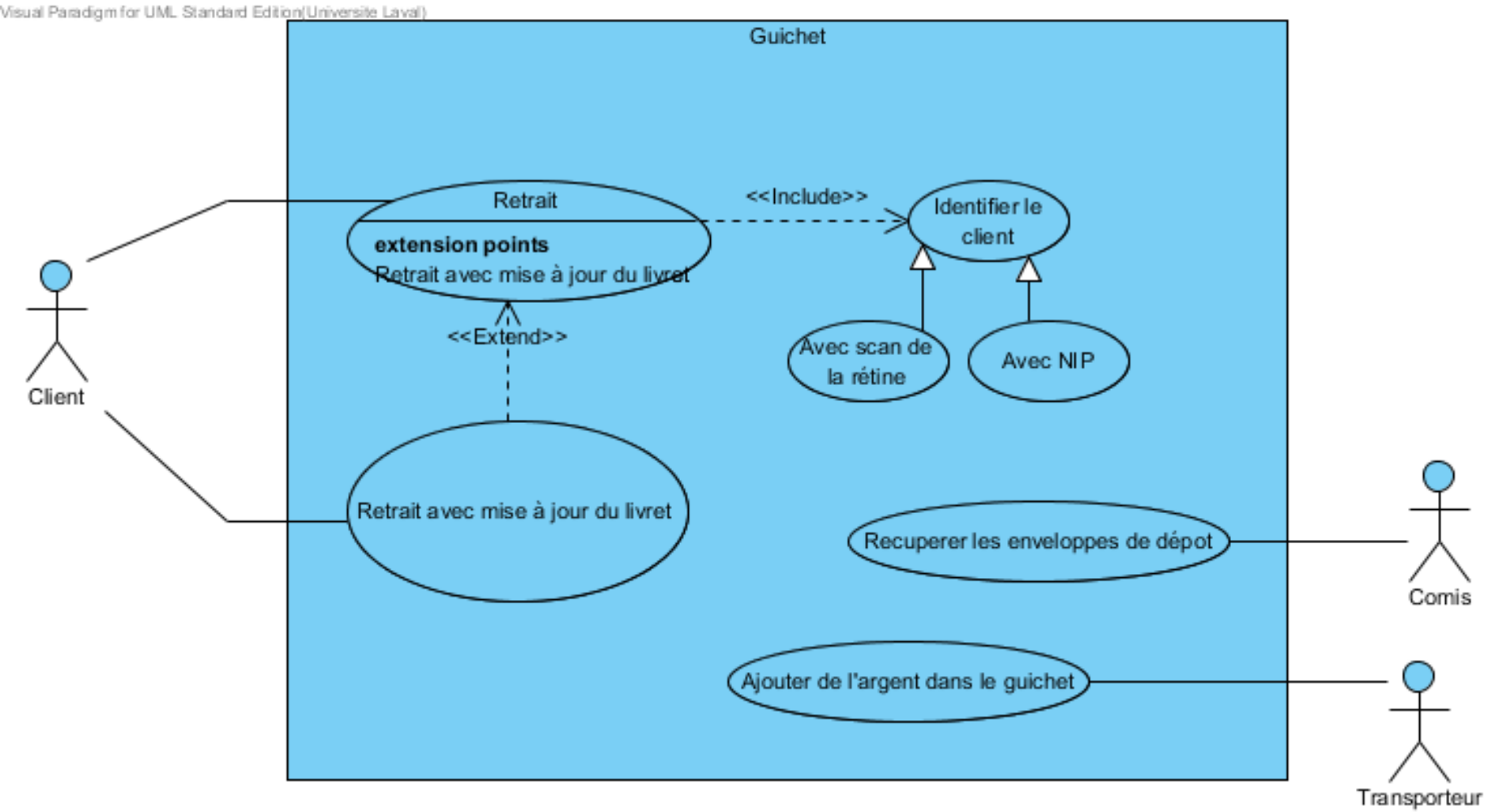


Diagramme des cas d'utilisation

≠

Cas d'utilisation

- **Diagramme des cas d'utilisation**: identifie **les** cas d'utilisation et les place dans un contexte
- Chaque **bulle** dans le diagramme identifie/nomme/réfère à un cas d'utilisation
- Mais **le cas d'utilisation est décrit par un texte** « à part ».
- Ce texte peut décrire un ou plusieurs « **scénarios** » associés à ce use case.
  - Un scénario principal/primaire
  - Des scénarios alternatifs

Description du cas d'utilisation

- Possibilité de le décrire avec différents niveaux de détail:
  - **Abrégé**: simple paragraphe décrivant le scénario principal.
  - **Informel**: On décrit en plus les scénarios alternatifs.
  - **Détaillé**: On décrit en détail chaque scénario. On spécifie en plus les détenteurs d'intérêt, on en plus des préconditions et garantie de succès, etc.

Abrégé

Cas d'utilisation:	**Effectuer un retrait**
Acteur(s):	Client
Type:	Primaire
Description:	Un client se présente au guichet automatique, s'identifie, retire un montant d'argent et quitte avec son argent.

Détaillé

Cas d'utilisation:	**Effectuer un retrait**
Système:	Guichet automatique
Acteurs:	Client
Parties prenantes et intérêts:	Client: Il désire repartir avec son argent. Gestionnaire de la banque: Il veut que le système enregistre la transaction.
Préconditions:	Le client a un compte bancaire actif.
Garanties en cas de succès:	Le client a reçu son argent et son compte a été débité.
Scénario principal:	1. Un client se présente au guichet automatique 2. Il s'identifie. 3. …
Scénarios alternatifs:	…

Détaillé en version deux colonnes

1. Un client se présente au guichet	
2. Le client s'identifie	3. Présentation des options
4. Le client choisit de faire un retrait.	5. Sollicitation du compte à débiter
6. Le client choisit le compte	7. Sollicitation du montant du retrait.
8. Le client entre le montant à retirer.	9. Validation de la disponibilité des fonds.
	10. Débite le compte
	11. Remet l'argent
12. Le client prend l'argent et quitte.	

Module 5

- Phase d’élaboration
- Diagramme de séquence système (DSS)

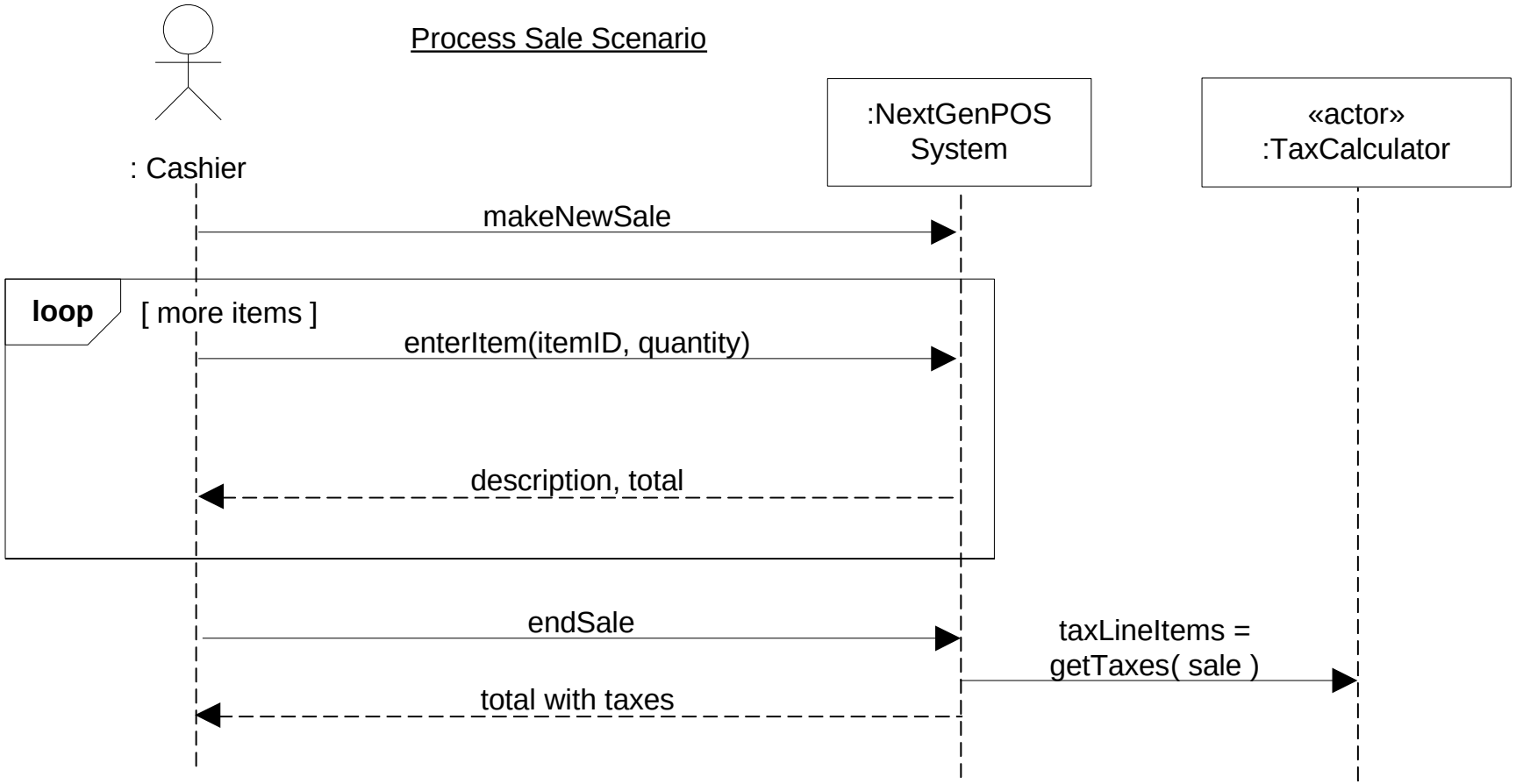
Artefacts de la phase d'élaboration

Discipline	Artifact Iteration	Incep. Il	Elab. EL.En	Const. CL.Cn	Trans. T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	*s*	r		
	Vision	*s*	r		
	Supplementary Specification	*s*	r		
	Glossary	*s*	r		
Design	Design Model SW Architecture Document Data Model		s s s	r r	
Implementation	Implementation Model (code)		s	r	r

S: start
R: refine

	Activités (appelées *disciplines* dans le Processus Unifié)	**Modèles** et artefacts générés
Analyse	Modélisation domaine d'affaires / Business modeling / Modélisation métier	**Modèle du domaine**: (1) diagramme de classe « conceptuel », (2) parfois un diagramme d'activités
	Analyse des besoins / Exigences / Requirements	(3) Énoncé de vision
		Modèle de cas d'utilisation / Use-case model : (4) diagramme des cas d'utilisation, (5) texte des cas d'utilisation, (6) diagramme de séquence système
		(7) Spécifications supplémentaires
		(8) Glossaire
	Design / Conception	**Modèle de conception / Design model** : (9) diagrammes de classes, (10) diagrammes d'interaction, (11) tout autre diagramme UML pertinent selon le contexte
	Implémentation	(12) Code
	…	

Concepts avancés – Systèmes externes



Module 6

- Phase d’élaboration
- Modèle du domaine

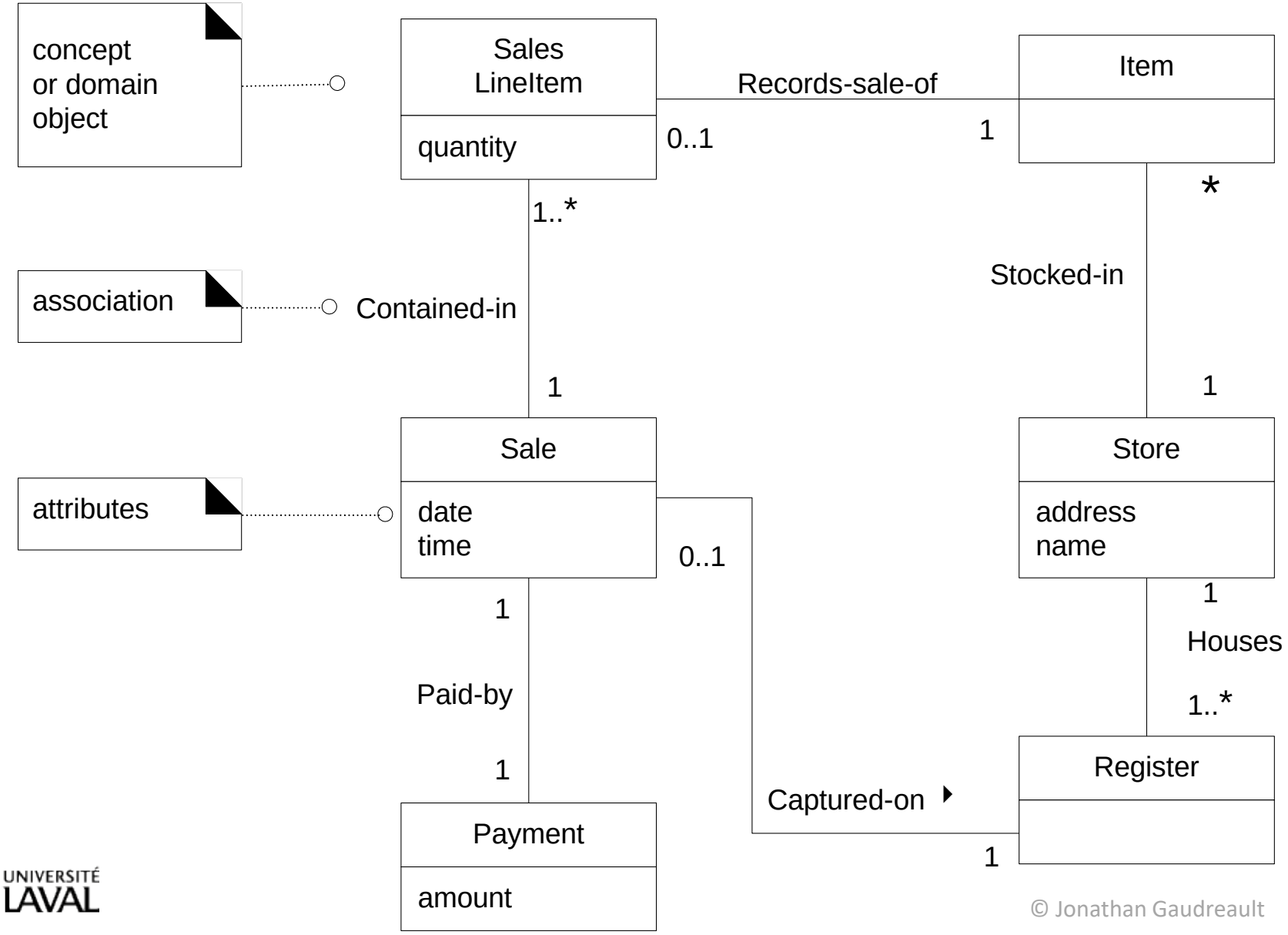
Modèle du domaine - Motivation

- Définir le « vocabulaire » du domaine
- S'assurer que l'on comprends le « monde » (**domaine**) dans lequel va évoluer notre application (rappelez-vous quel est le principal objectif de l'analyse: comprendre /décrire le problème)
- Établir communication avec les détenteurs d'intérêt

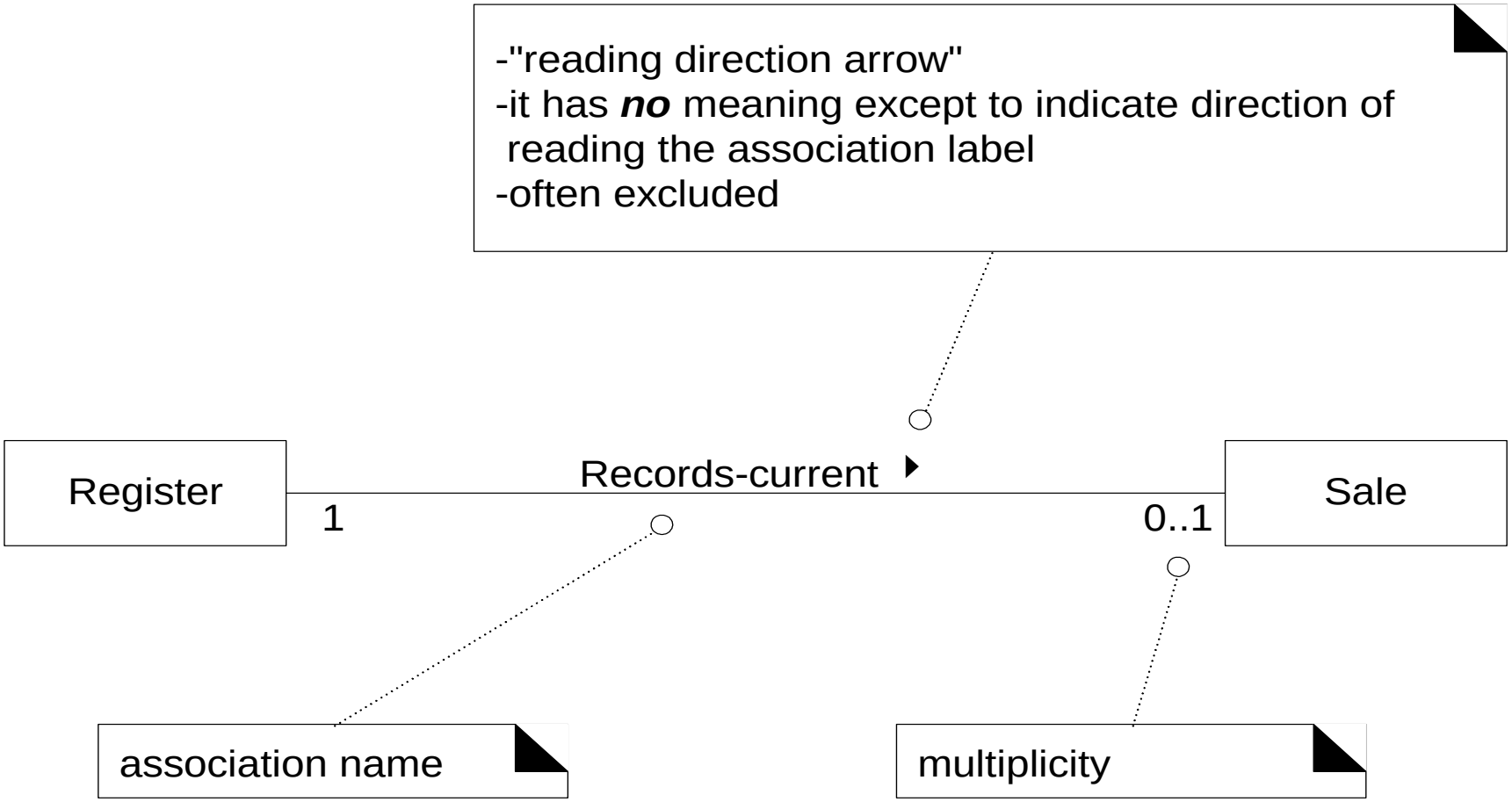
Modèle du domaine

- Le **modèle du domaine** est une représentation visuelle des classes conceptuelles (ou si vous préférez, des objets du monde réel) pour un domaine d'application / domaine d'affaires / domaine métier donné.
- Synonymes:
  - Modèle du domaine
  - Modèle objet du domaine
  - Modèle objet d'analyse
  - Diagramme conceptuel
  - Diagramme des classes conceptuelles

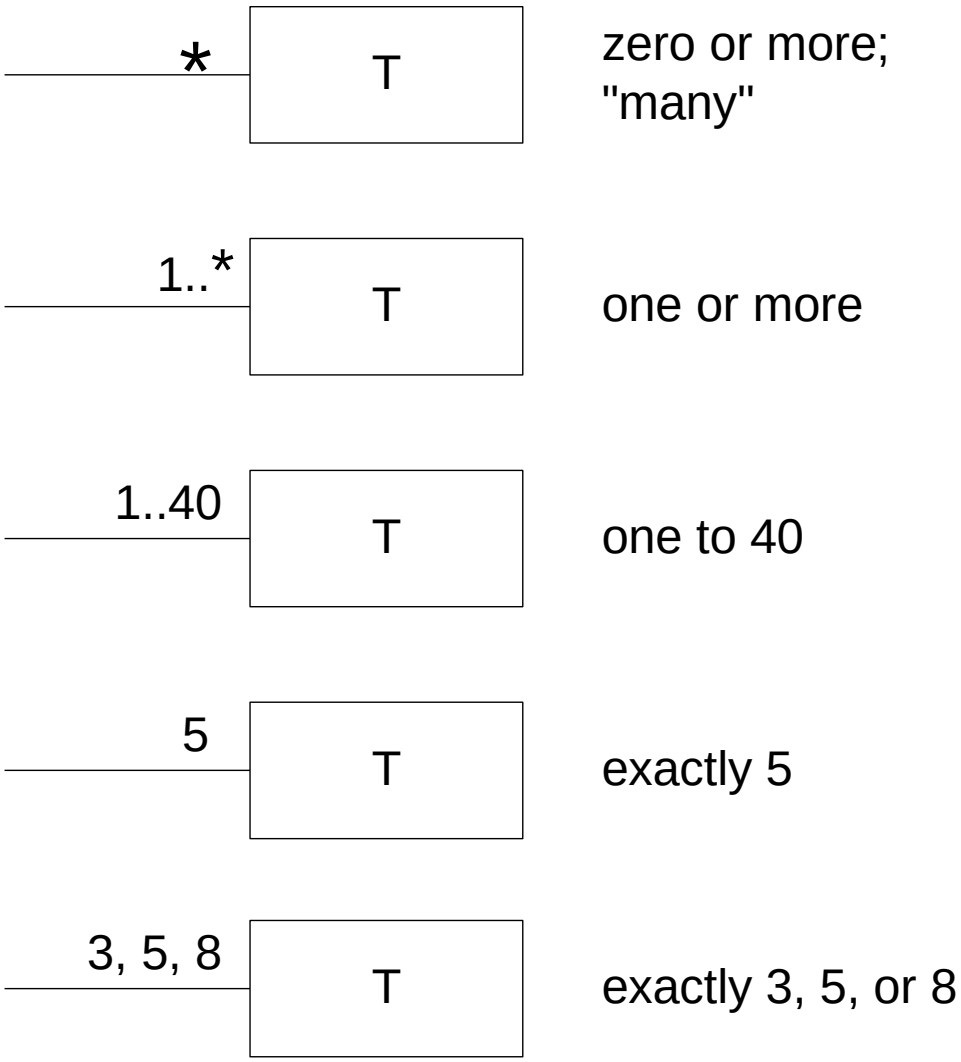
Exemple Nexgen POS (premier jet)



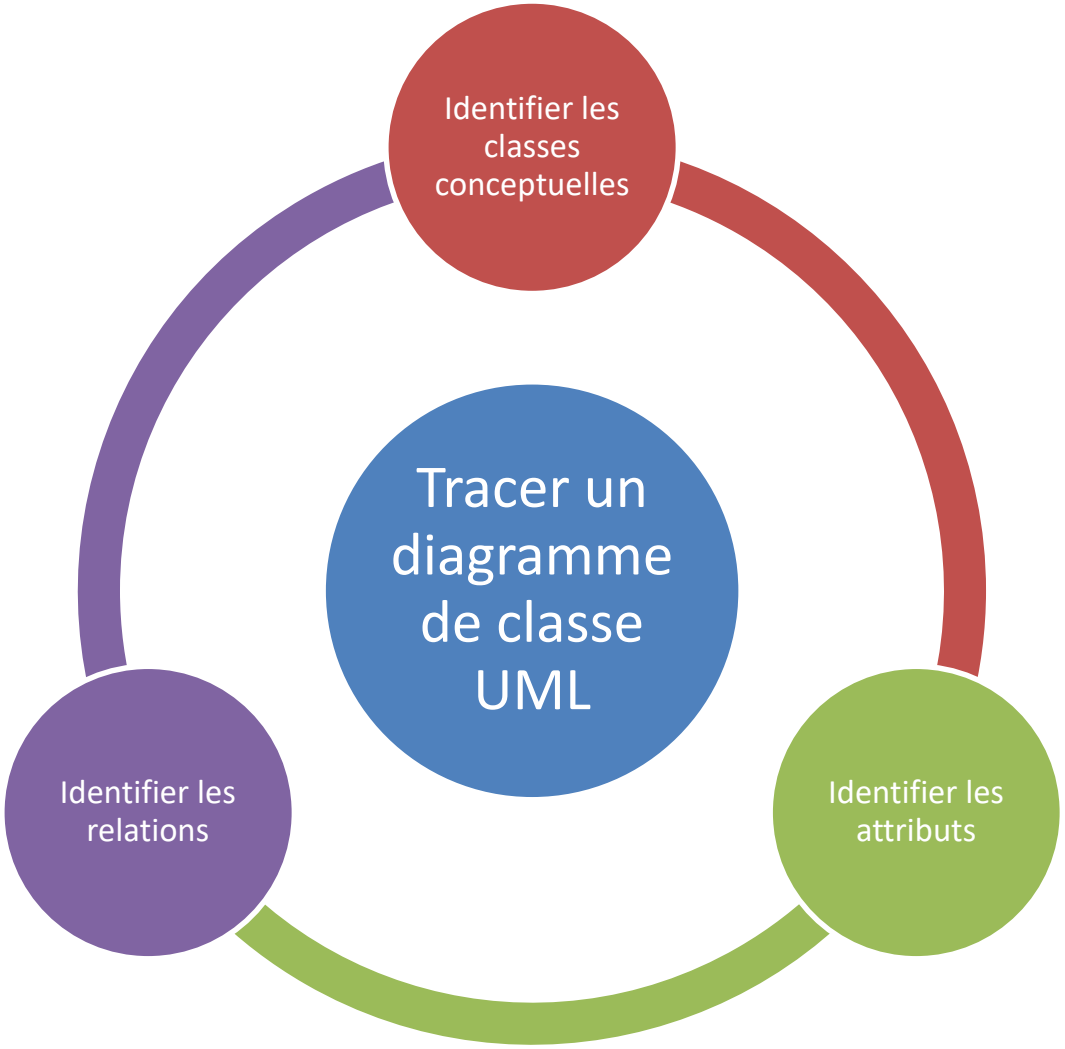
Direction de lecture



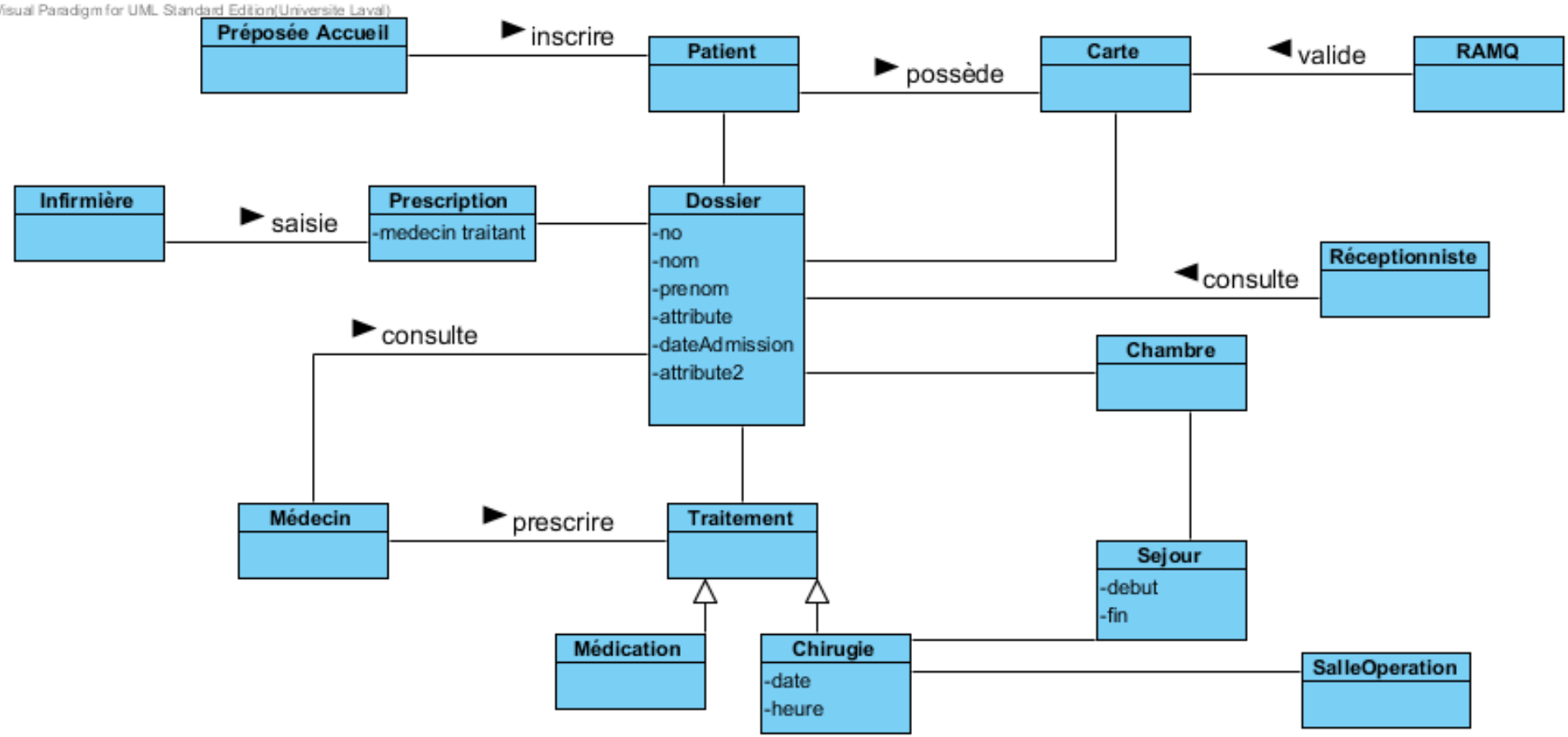
Cardinalité



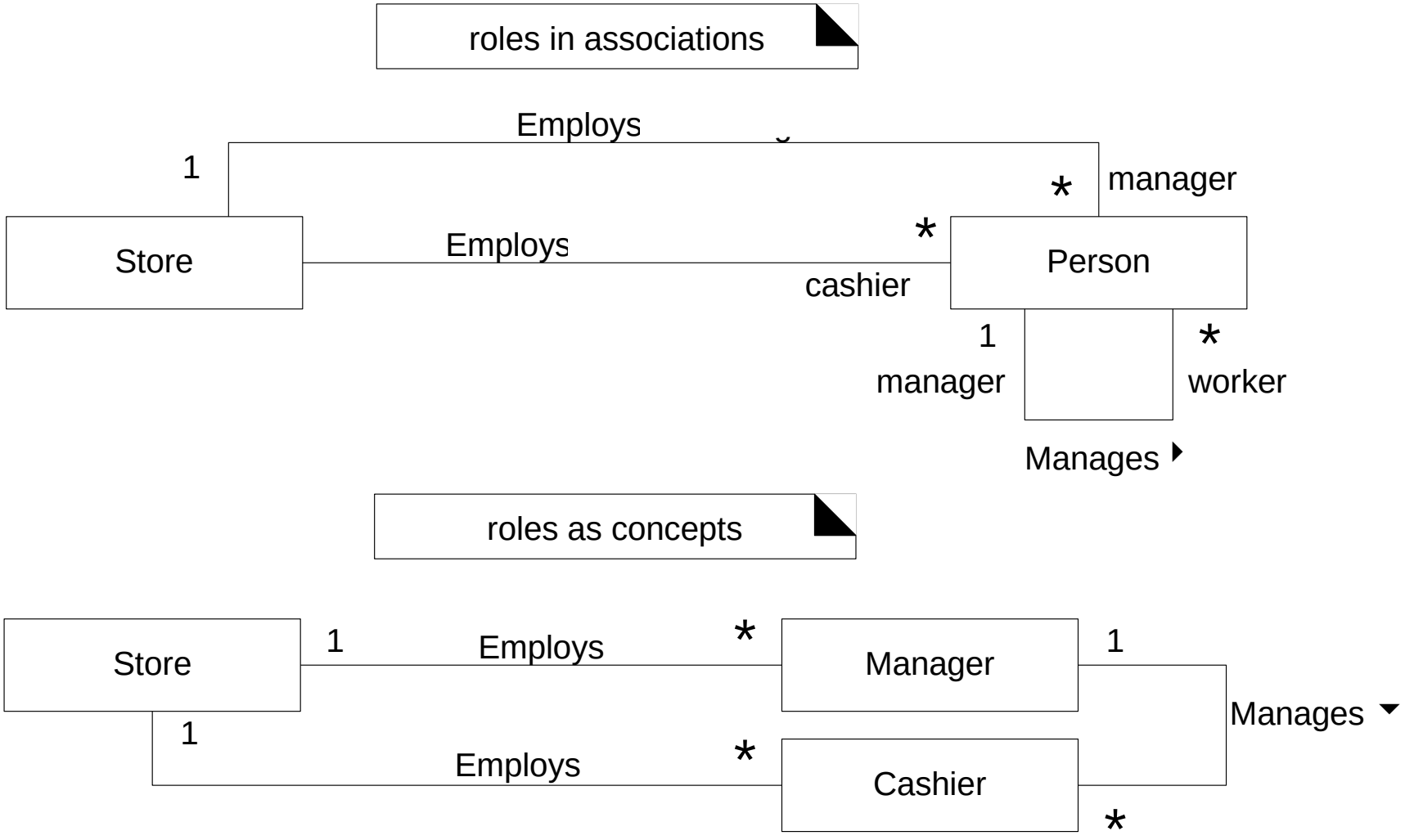
Création du modèle



Solution – Modèle du domaine SysGDP



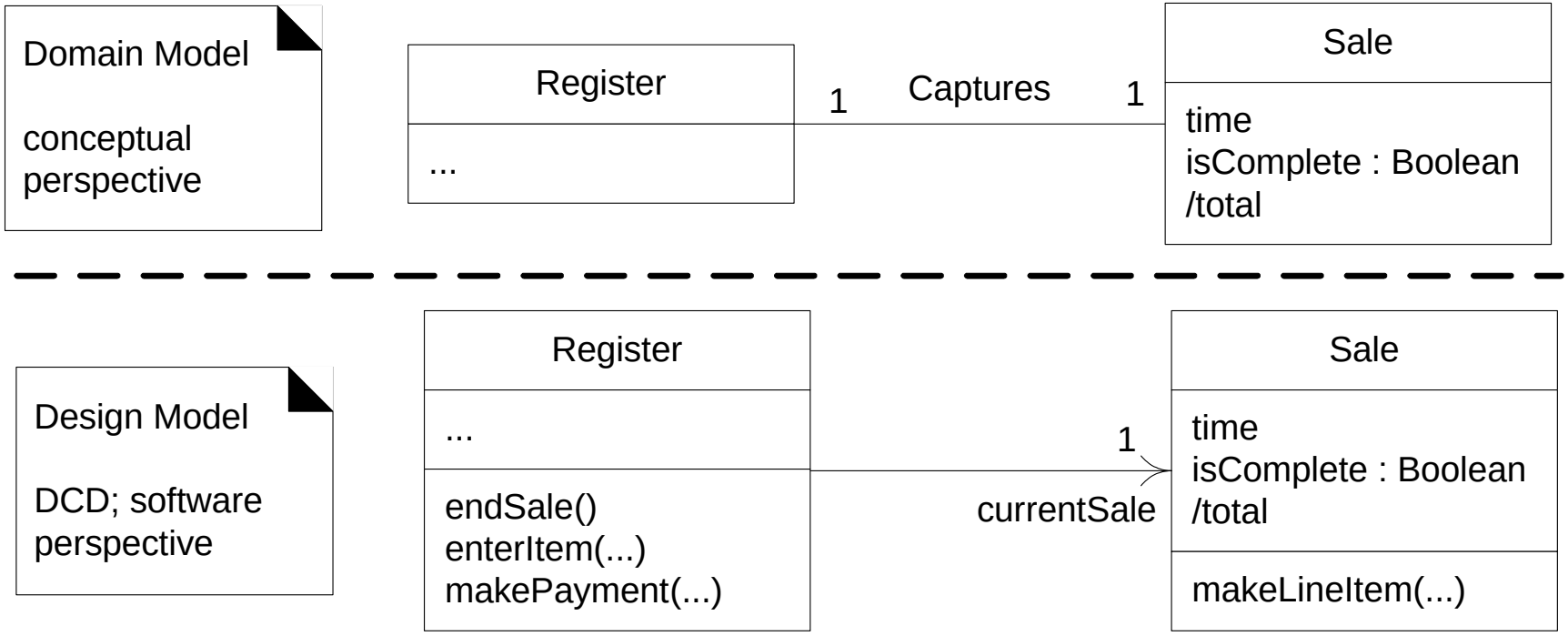
Rôles



Module 7

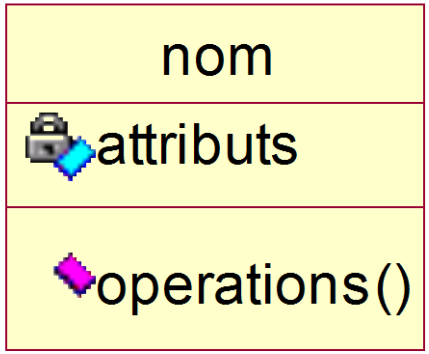
- Diagramme de classe de conception

NexGen POS – Classes conservées (mais transformées)

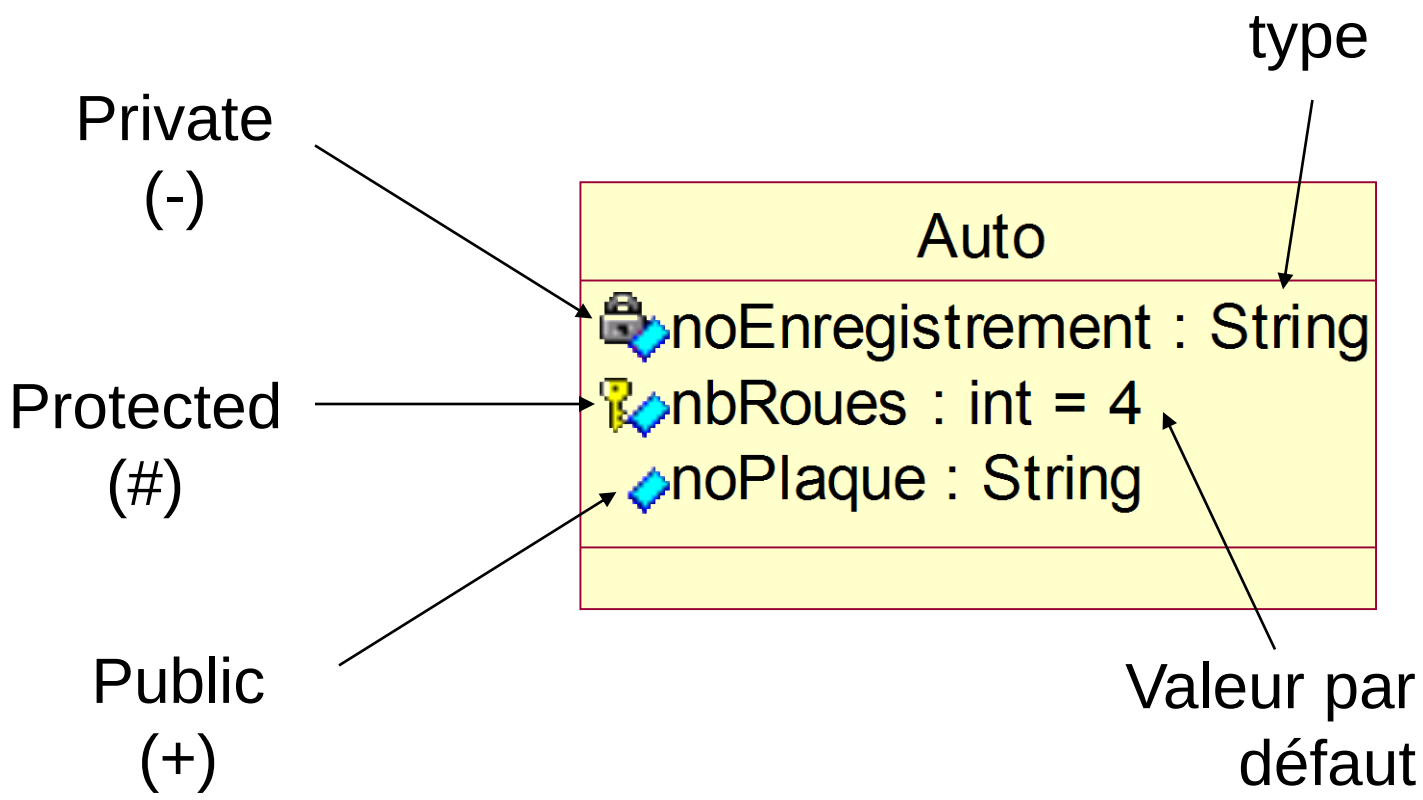


Représentation des classes en UML

- une ***classe*** est représentée par un ***rectangle*** divisé en ***trois compartiments***:
  - le compartiment du ***nom***
  - le compartiment des ***attributs***
  - le compartiment des ***opérations***
- la syntaxe dans les différents compartiments est ***indépendante*** des langages de programmation

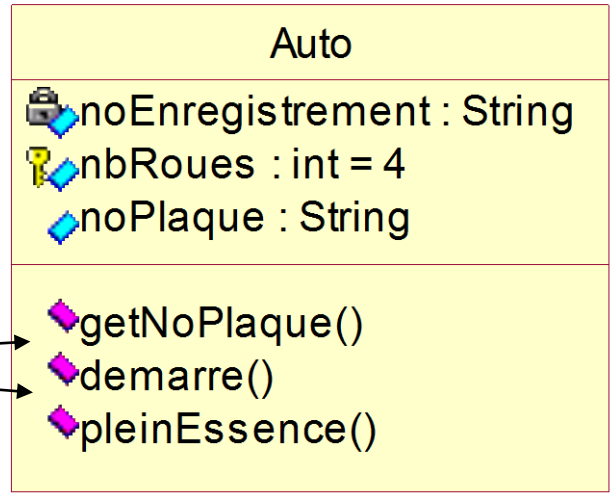


Exemples de classe avec attributs



Exemple de classe avec opérations

Méthodes publiques

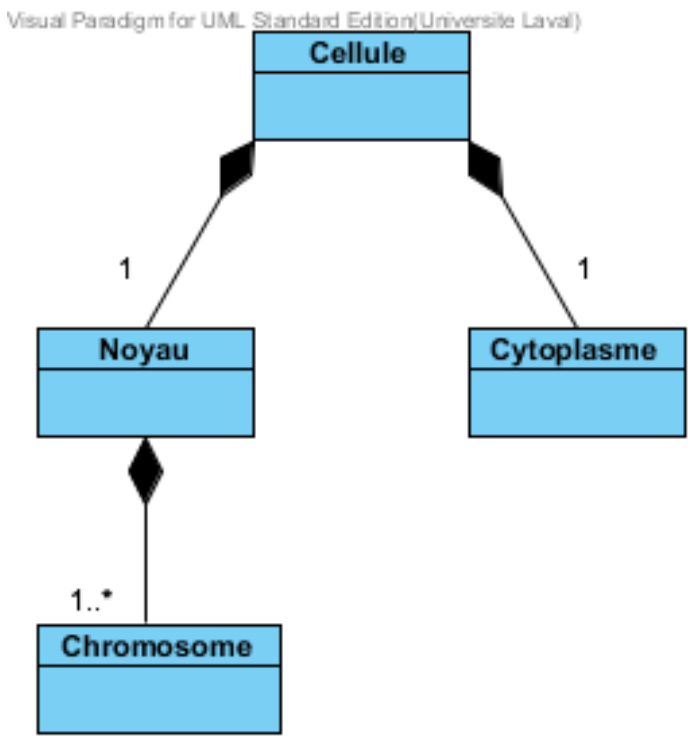


Compartiment des opérations/méthodes

- Les ***opérations*** permettent de ***manipuler*** les attributs et d'effectuer d'autres actions. Elles sont appelées pour des instances (objets) de la classe.
- La ***signature*** des opérations comprend:
  - un ***type*** de retour
  - un ***nom***
  - 0 ou plus ***paramètres*** (ayant chacun un type, un nom et possiblement une valeur par défaut)
  - une ***visibilité*** (private(-), protected(#), public(+))
- Les opérations constituent l'***interface*** de la classe avec le monde extérieur

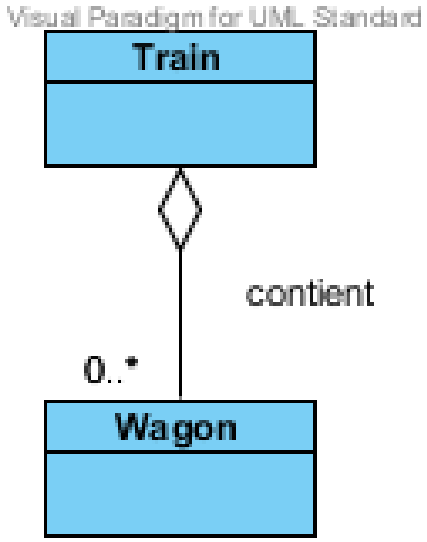
Composition («est constitué de »)

- Indique une relation ***tout-partie avec inclusion « physique »***
- se représente par une ligne avec un ***losange plein***
- Elle possède des ***rôles***, ***multiplicité***, etc, comme une association normale

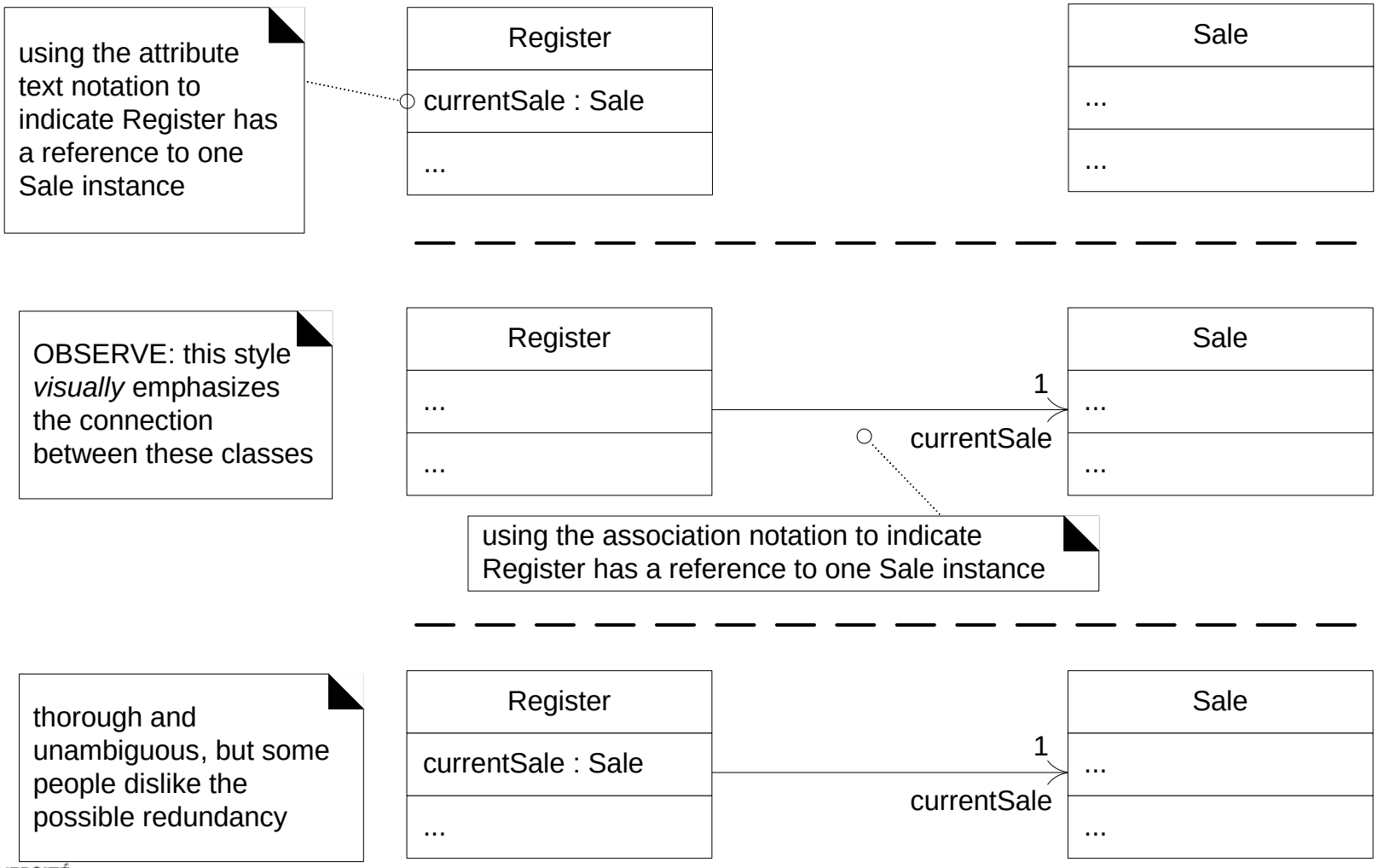


Agrégation («a un»)

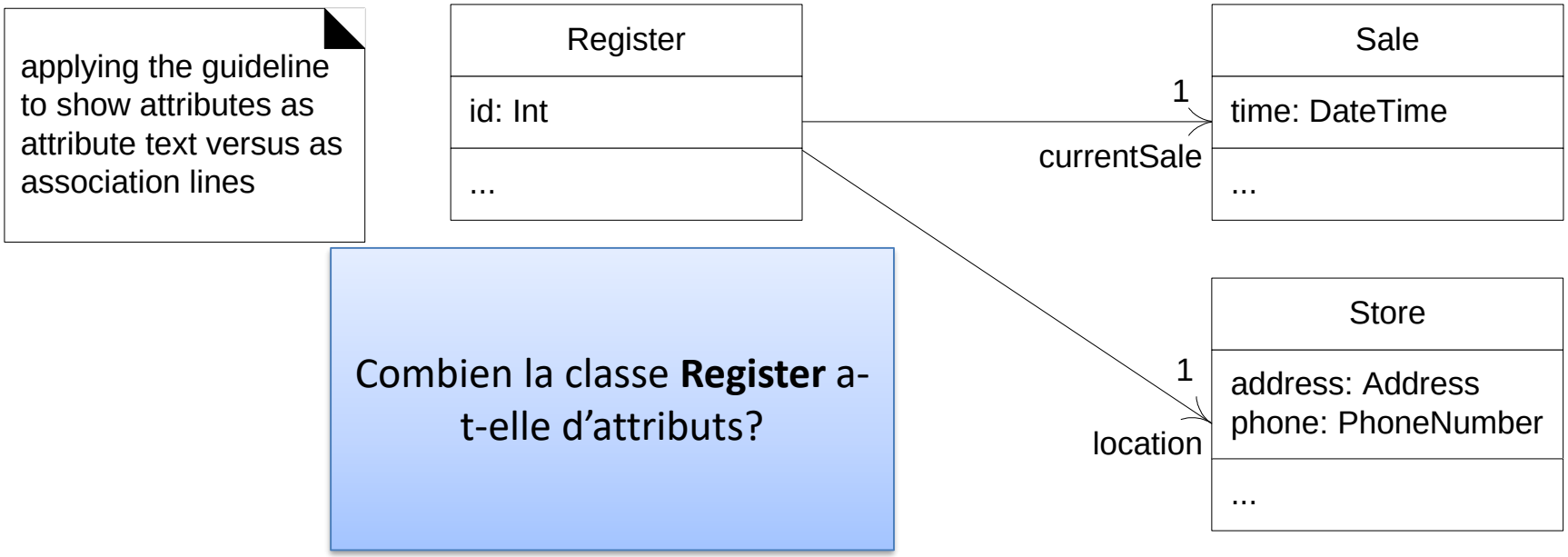
- Indique une relation ***tout-partie*** (moins forte que la composition)
- Elle possède des ***rôles***, ***multiplicité***, etc, comme une association normale
- Se représente par une ligne avec un losange vide
- Techniquement, n'indique rien de plus qu'une relation ordinaire sous UML.
- En pratique, très utile pour la communication.



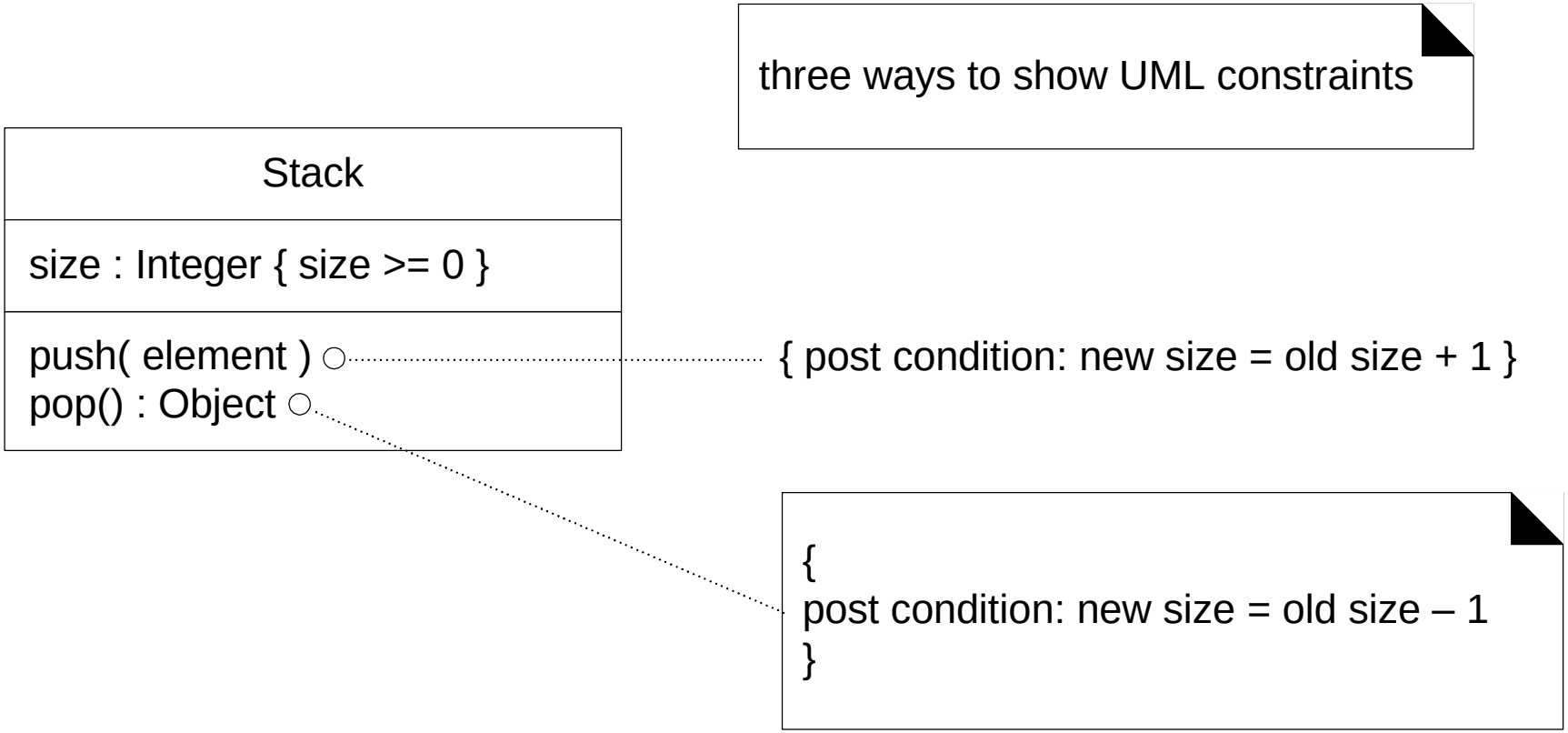
Représentation des attributs



Représentation des attributs

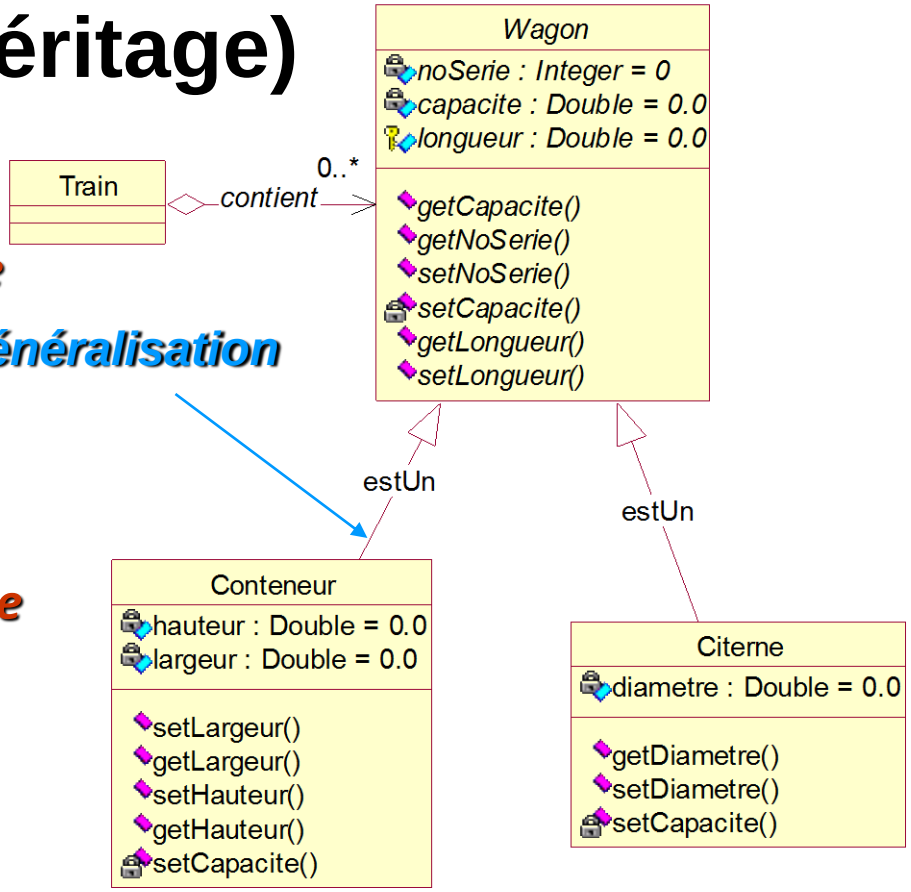


Contraintes



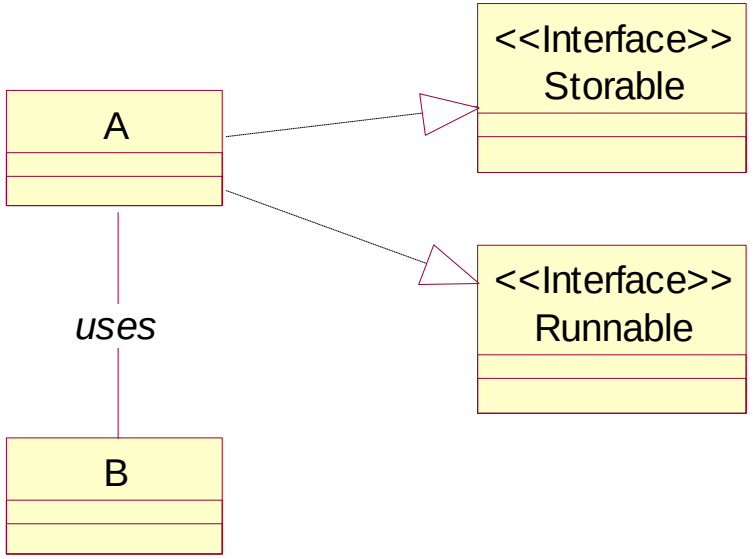
Représentation des généralisations (héritage)

- On représente les ***généralisations*** par une ***flèche*** allant de la classe spécifique à la classe générale
- Lorsqu'une classe spécifique hérite de plusieurs super-classes, on dit qu'il y a ***héritage multiple***



Interfaces

- Les ***interfaces*** sont des classes contenant seulement des ***opérations*** sans ***implémentation***.
- Une classe peut ***implémenter*** une interface
- Le symbole d'implantation est une ***flèche pointillée*** allant de la classe ***vers*** l'interface
- Ici, la classe A implante les interfaces Storable et Runnable et B utilise ces implantations

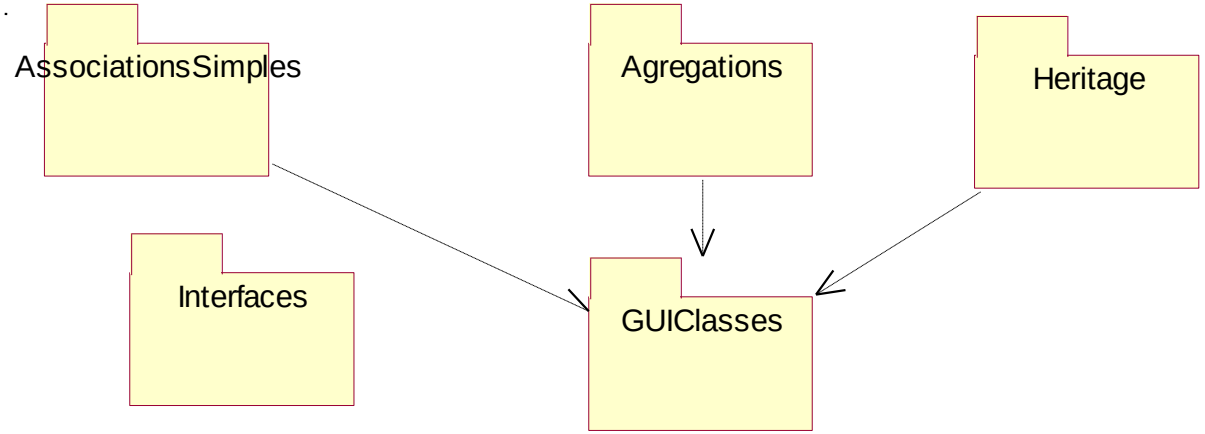


Module 8

- Architecture logique
- Diagramme de packages

Les packages

- Un package est représenté par un ***dossier*** (folder)
- Les Packages ont eux aussi un ***niveau de visibilité*** est des ***relations*** qui sont ici: la ***dépendance***, le ***raffinement***, et la ***généralisation***



Les Packages – En Java

- Pour les systèmes comprenant plusieurs classes, il convient de regrouper celles-ci en entités logiques, les ***packages***
- Un package est un ***ensemble de packages*** et de ***classes*** reliés sémantiquement entre eux
- Chaque package est muni de ***classes publiques*** qui lui permet de communiquer sa fonctionnalité aux autres packages qui peuvent l'importer

Architecture logique

- L'**architecture logique**, c'est l'organisation à grande échelle des classes logicielles en packages, sous-systèmes et couches.
- On la nomme architecture logique, car elle n'implique aucune décision quant à la façon dont ces éléments seront déployés physiquement.

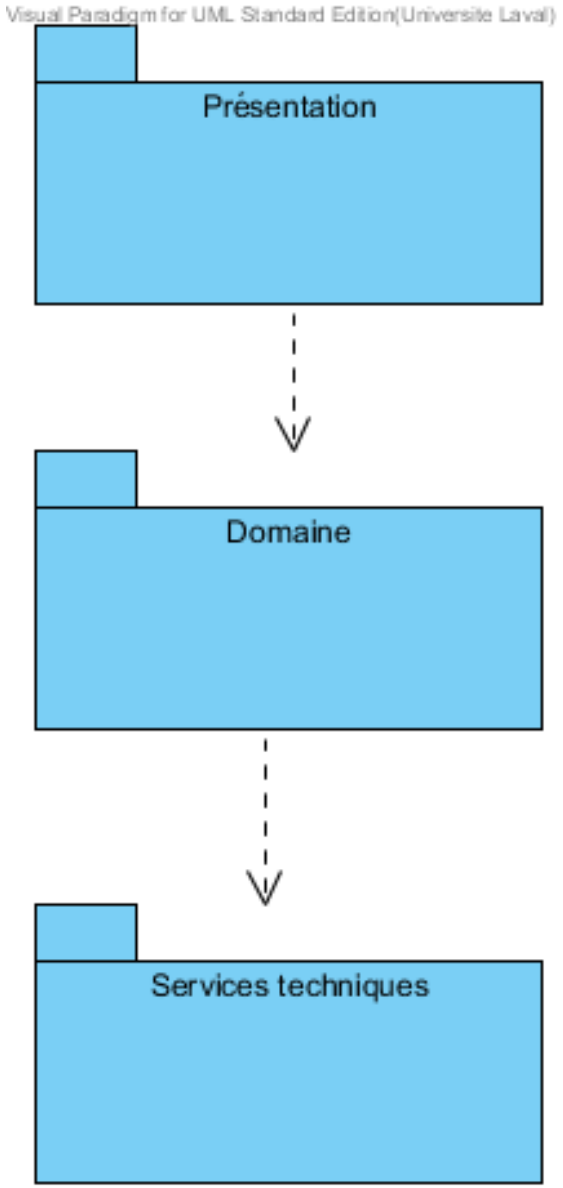
Architecture en couche

- Une **couche** est un regroupement à très forte granularité de classes, packages et sous-systèmes qui a une responsabilité de cohésion pour un aspect majeur du système.
- **Architecture en couche**: organisée de manière à ce que les couches accèdent aux couches inférieures, mais pas l'inverse.

Architecture en couche - Avantages

- Réduction du couplage et des dépendances
- Maintenance facilitée
- Plus facile de remplacer certaines couches par de nouvelles implémentations
- Les couches les plus basses contiennent des fonctionnalités réutilisables
- La segmentation facilite le développement en équipes

Architecture en couche « classique »



Interface utilisateur

Logique applicative et objets du domaine

Objets à usage général (ex: classes accès à une BD) – généralement indépendant de l'application

Quel est le contraire d'une architecture basée sur le principe Modèle-Vue ?

- Une interface-moteur!
  - La logique applicative et l'interface utilisateur sont mélangées
    - Button1.onClick() fait des traitements intelligents!
  - Rien de réutilisable!

Module 9

- Diagramme d'intéraction
  - Diagramme de séquence
  - Diagramme de communication

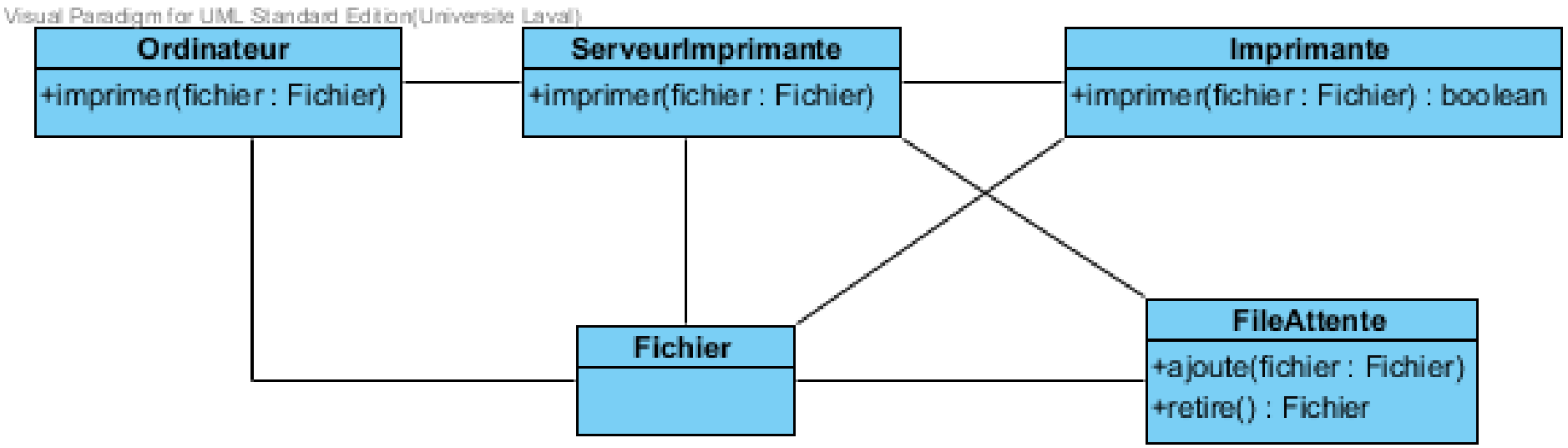
Introduction

- Le diagramme de classes montre la structure **statique** du système en terme de classes/objets et de relations entre elles.
- Cependant, cette description ne montre pas comment ces classes/objets interagissent ensemble pour réaliser des tâches et fournir les fonctionnalités du système
- Une étude **dynamique** montre comment les objets interagissent dynamiquement à différents moments durant l'exécution du système

Les deux types de diagrammes d'interaction

- Le **diagramme de séquences** met l'accent sur l'aspect temporel
- Le **diagramme de communication** met l'accent sur les relations entre les objets
- Ils présentent sensiblement la même information

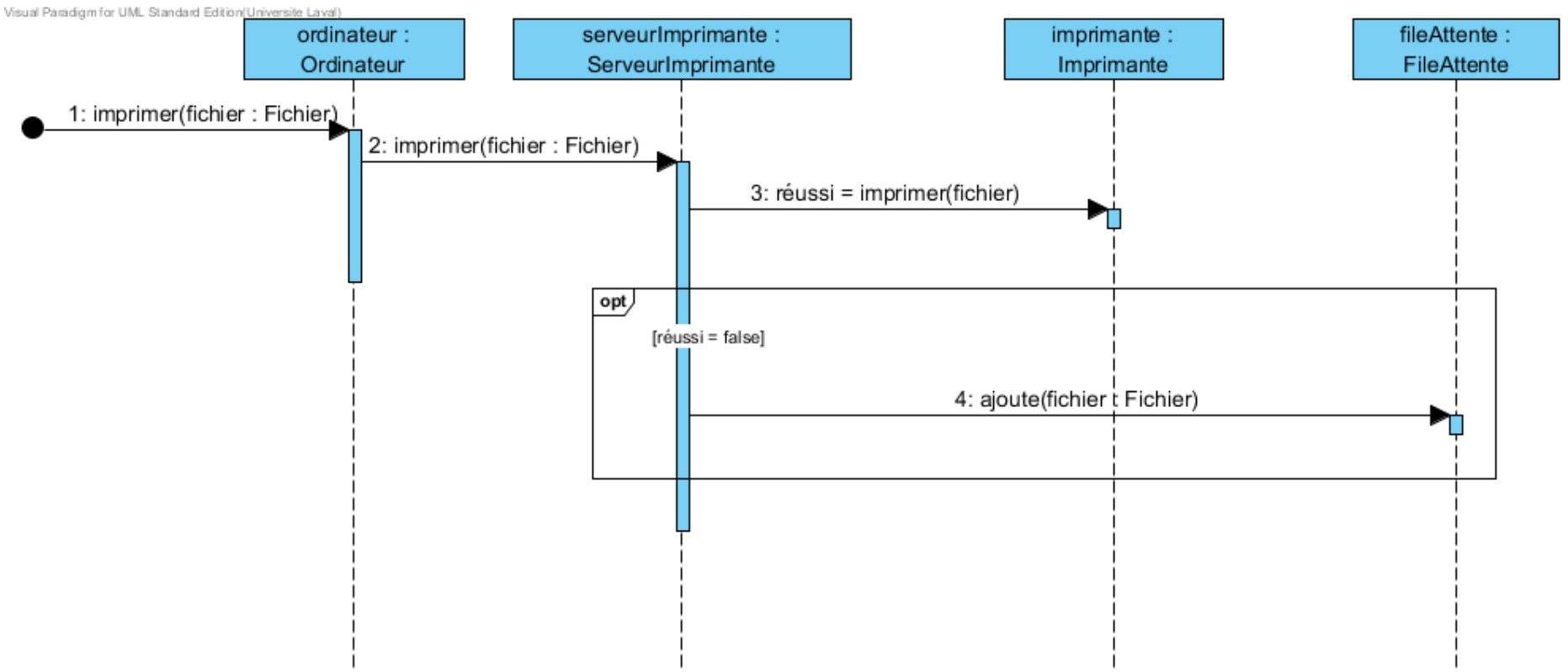
Diagramme de classes



Des questions subsistent. Comment fonctionnent ces classes? Comment accomplissent-elles quelque chose?

Diagramme de séquence

- La même information, mais représentée sous une forme différente à l'aide d'un diagramme de séquence:



Les diagrammes de *séquence**

- Ils se concentrent sur la ***séquence des messages envoyés entre les objets*** (c'est-à-dire ***comment*** et ***quand*** les messages sont envoyés et reçus entre les objets)
- Ils possèdent ***deux axes***: l'***axe vertical*** montre le ***temps*** tandis que l'***axe horizontal*** montre l 'ensemble des ***objets en interaction*** dans un ***scénario*** ou une ***scène*** spécifique d'un scénario.

Tel que proposé par Larman, les diagrammes de séquence sont utilisés de deux manières différentes

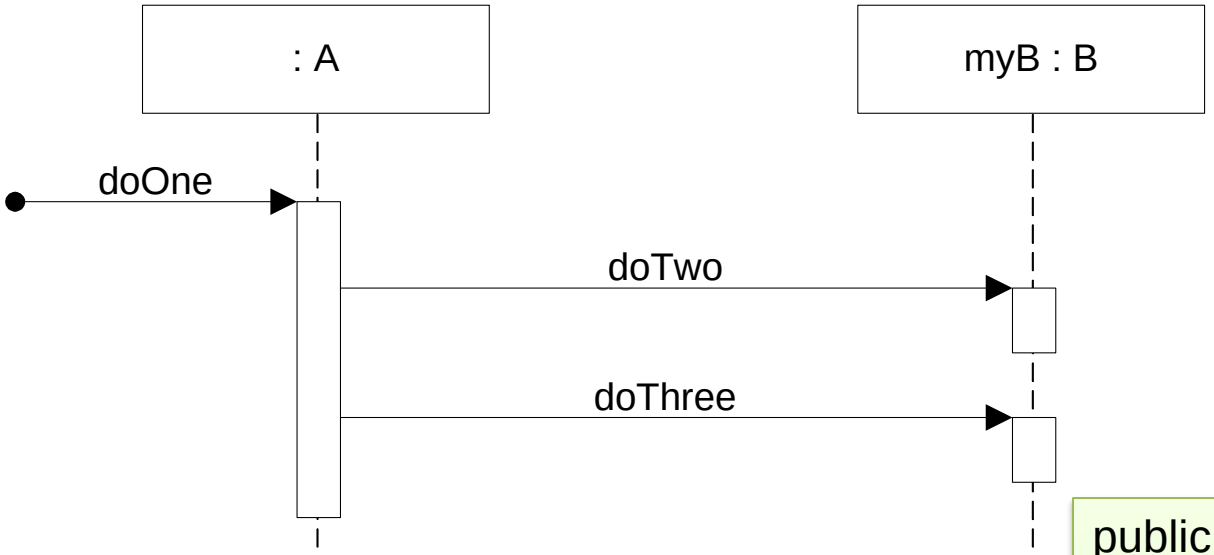
1. **Diagramme de séquence système (DSS)**
   - Documentation d'un scénario d'un use-case
   - Se concentre sur la description de l'interaction, souvent dans des termes proches de l'utilisateur et sans entrer dans les détails de la synchronisation
   - L'inscription portée sur une flèche correspond à un événement (ou une action) qui survient dans le domaine de l'application
   - Les flèches ne traduisent pas à ce niveau des envois de messages au sens des langages de programmation.

Tel que proposé par Larman, les diagrammes de séquence sont utilisés de deux manières différentes

2. Diagramme de séquence (de conception)

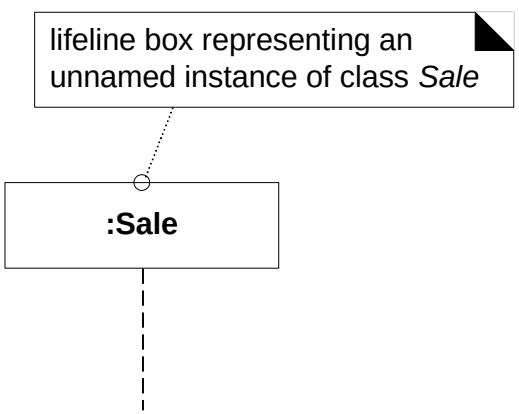
- Correspond à un usage plus « informatique »
- Permet la représentation précise des interactions entre objets, autrement dit le séquencement des flots de contrôle
- Exemple : Représentation des structures de contrôle (IF THEN ELSE) et de la structure de la boucle WHILE et FOR

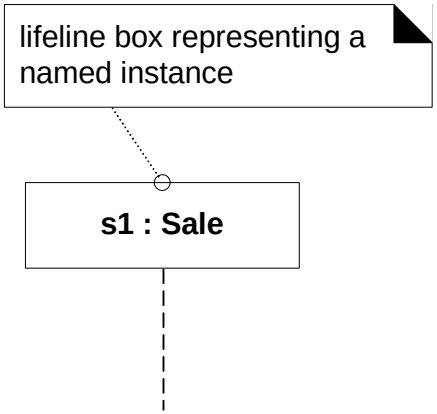
Correspondance diagramme-code



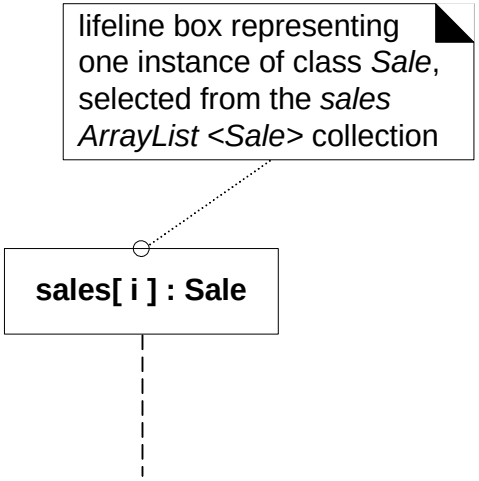

```
public class A
{
        Public void doOne()
        {
                myB.doTwo();
                myB.doThree();
        }
}
```

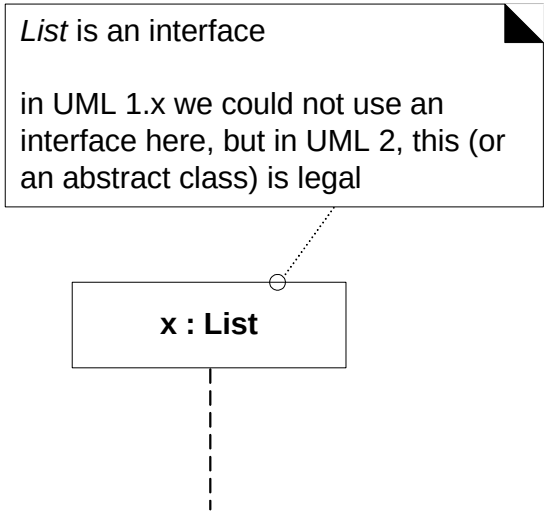
Représentation des objets





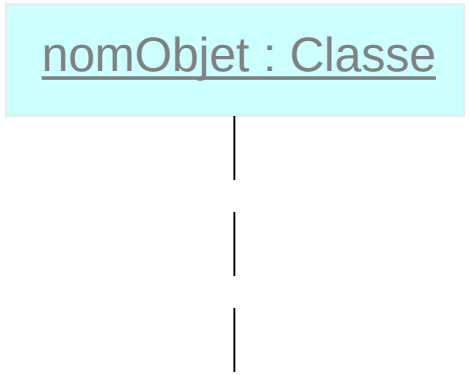
Sachez distinguer le nom de l'objet et le nom de la classe





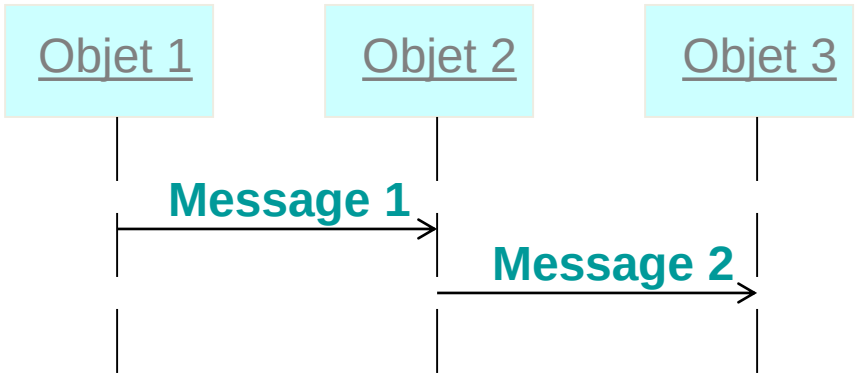
Représentation des objets*

- Un objet est matérialisé par un rectangle et une barre verticale, appelée ligne de vie de l'objet



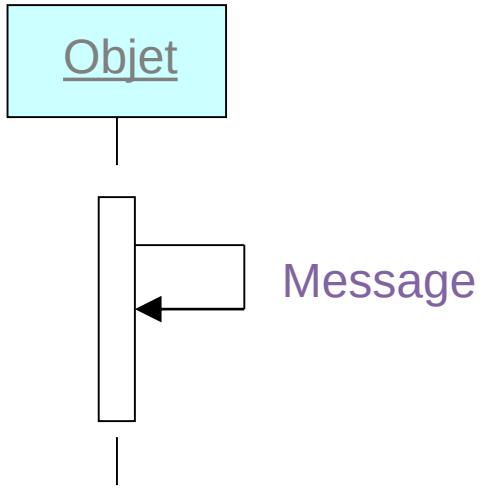
  - Lorsqu'on n'a plus besoin de l'objet dans la séquence, on peut interrompre sa ligne de vie à partir de l'instant où cet objet n'intervient plus dans la séquence

Représentation des interactions*



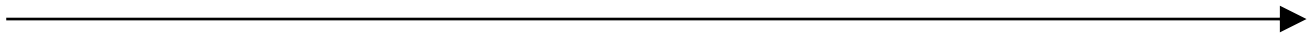
Message 1 précède dans le temps Message 2

Un objet peut appeler ses propres méthodes



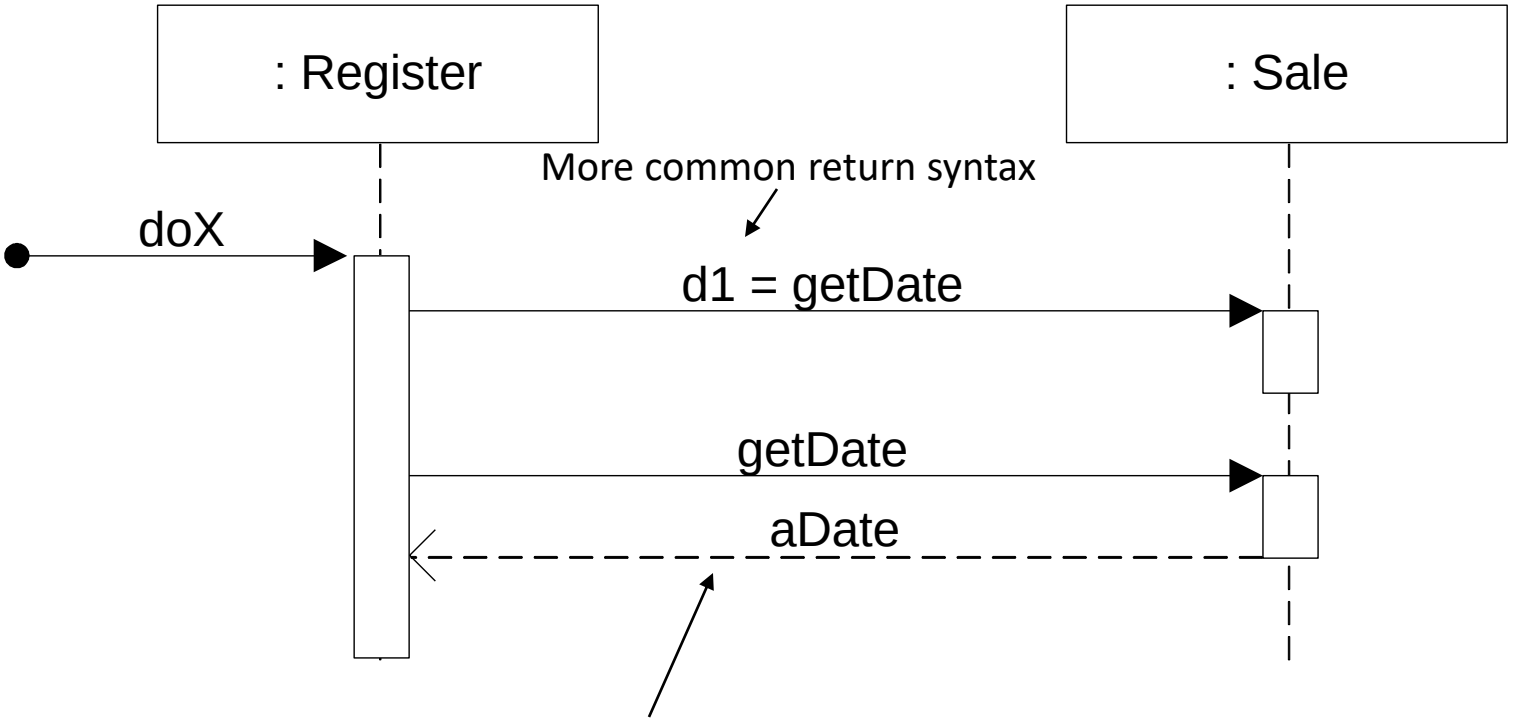
Spécification d'un appel/message

return = msgName(param:paramType1, …) : returnType



Heureusement, nous ne sommes pas obligés de toujours utiliser la syntaxe complète (!)

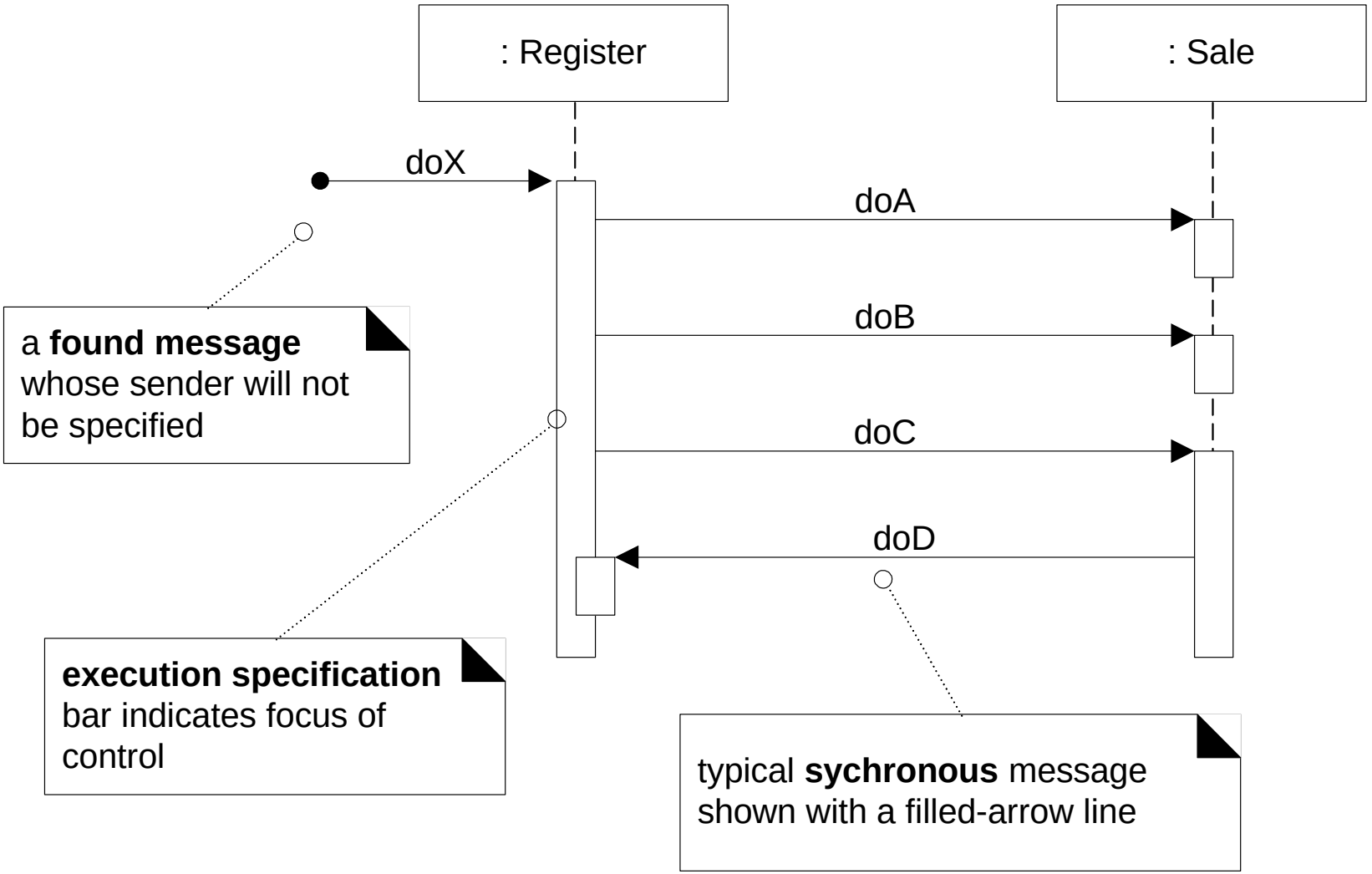
Notation simplifiée



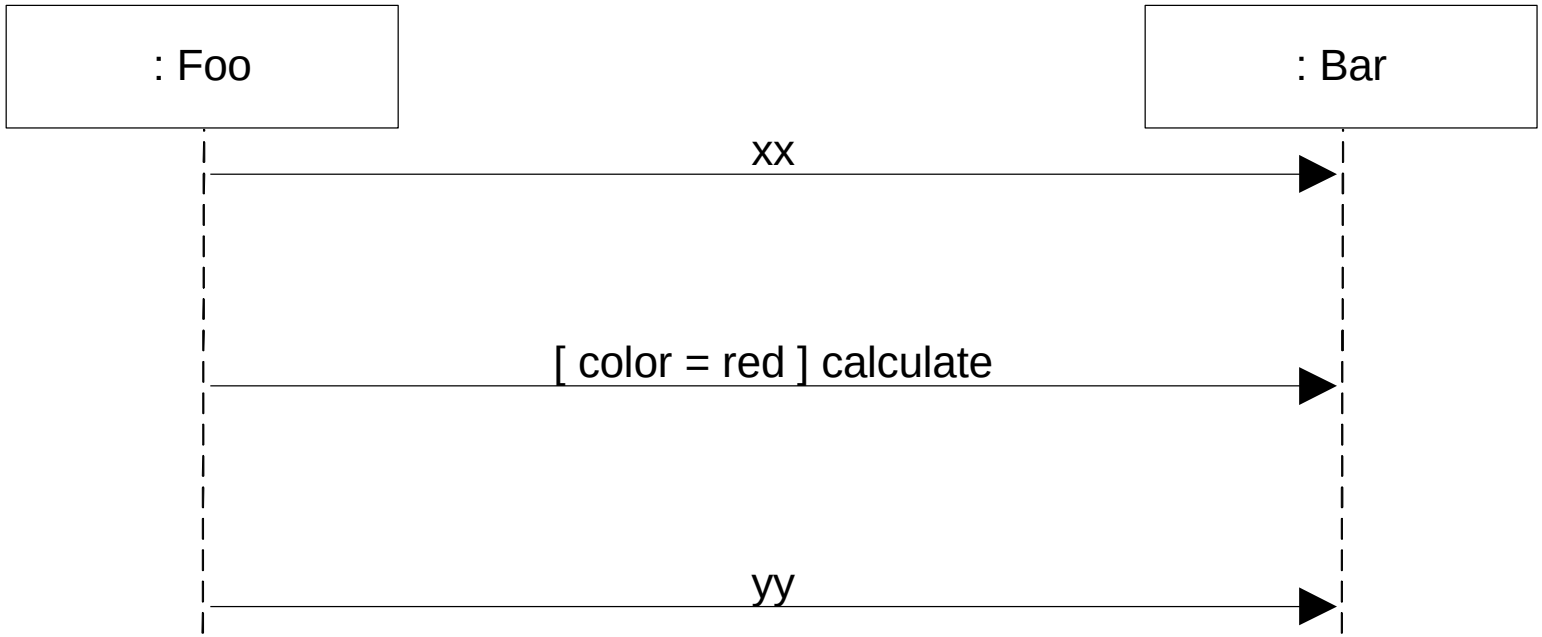
Notation simplifié

- Quel est le type de d1?
  - String
  - Date
  - Json
  - Moment
  - DateTime
- Le type dans la majorité des cas est nécessaire pour comprendre le fonctionnement

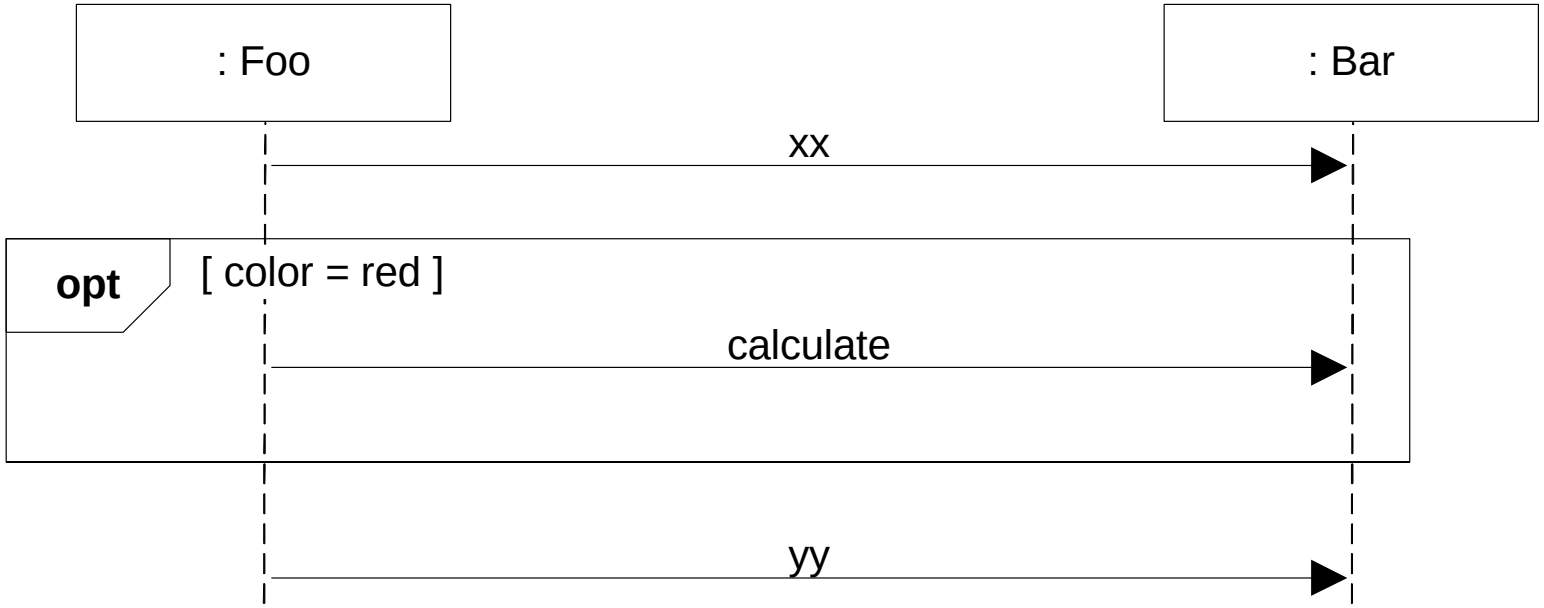
Barre de spécification d'exécution / activation



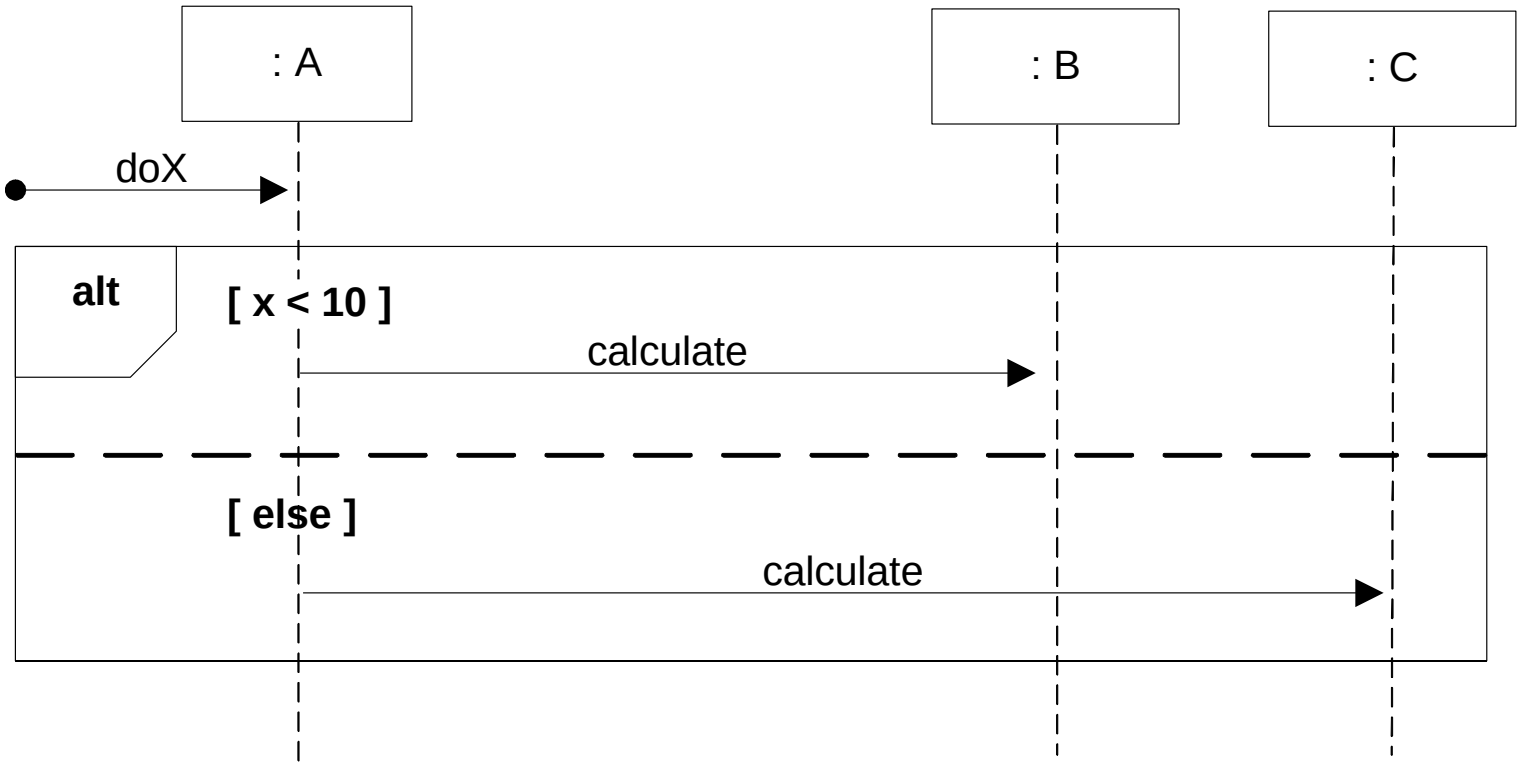
Énoncé conditionnel (if)



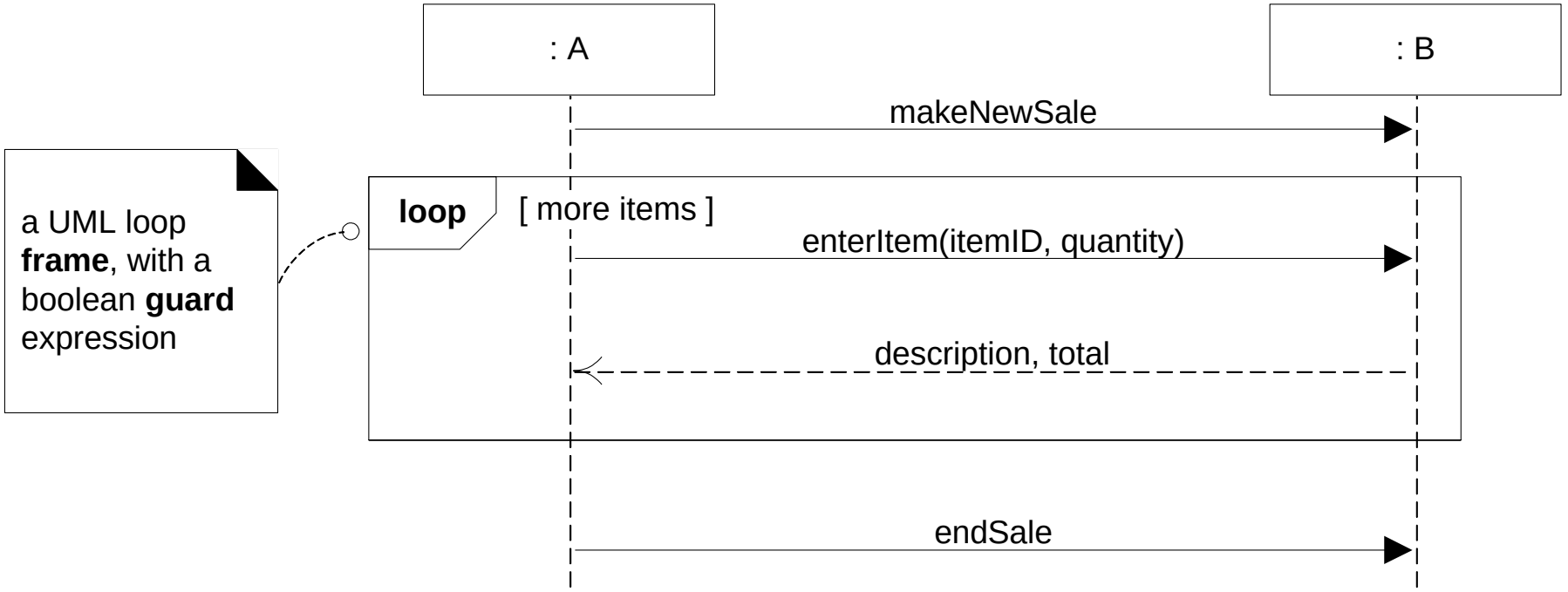
Groupe d'énoncés conditionnels (if)



Alternative (if then else)



Boucle


: A
: B
makeNewSale
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale
a UML loop **frame**, with a boolean **guard** expression

Boucle imbriquée dans un énoncé optionnel

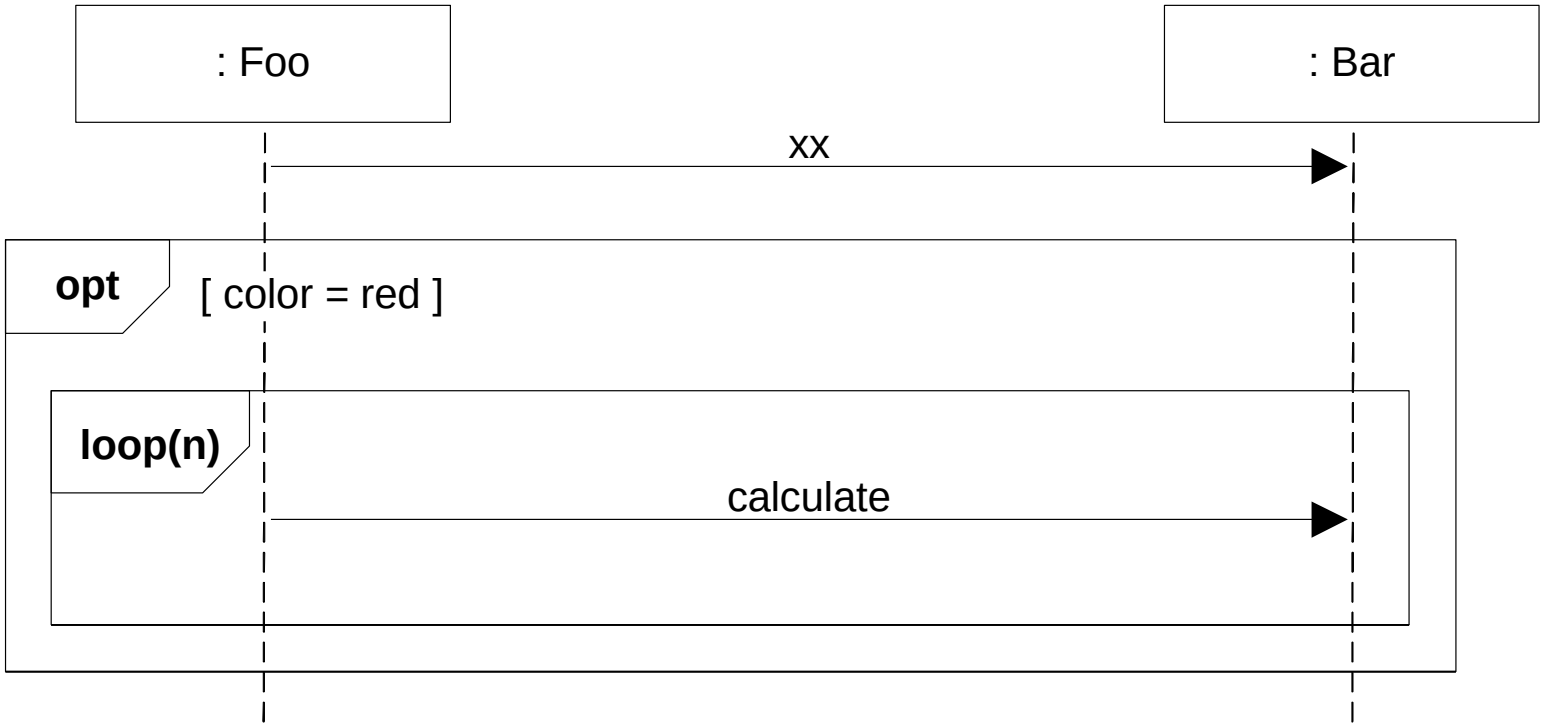


Diagramme de communication

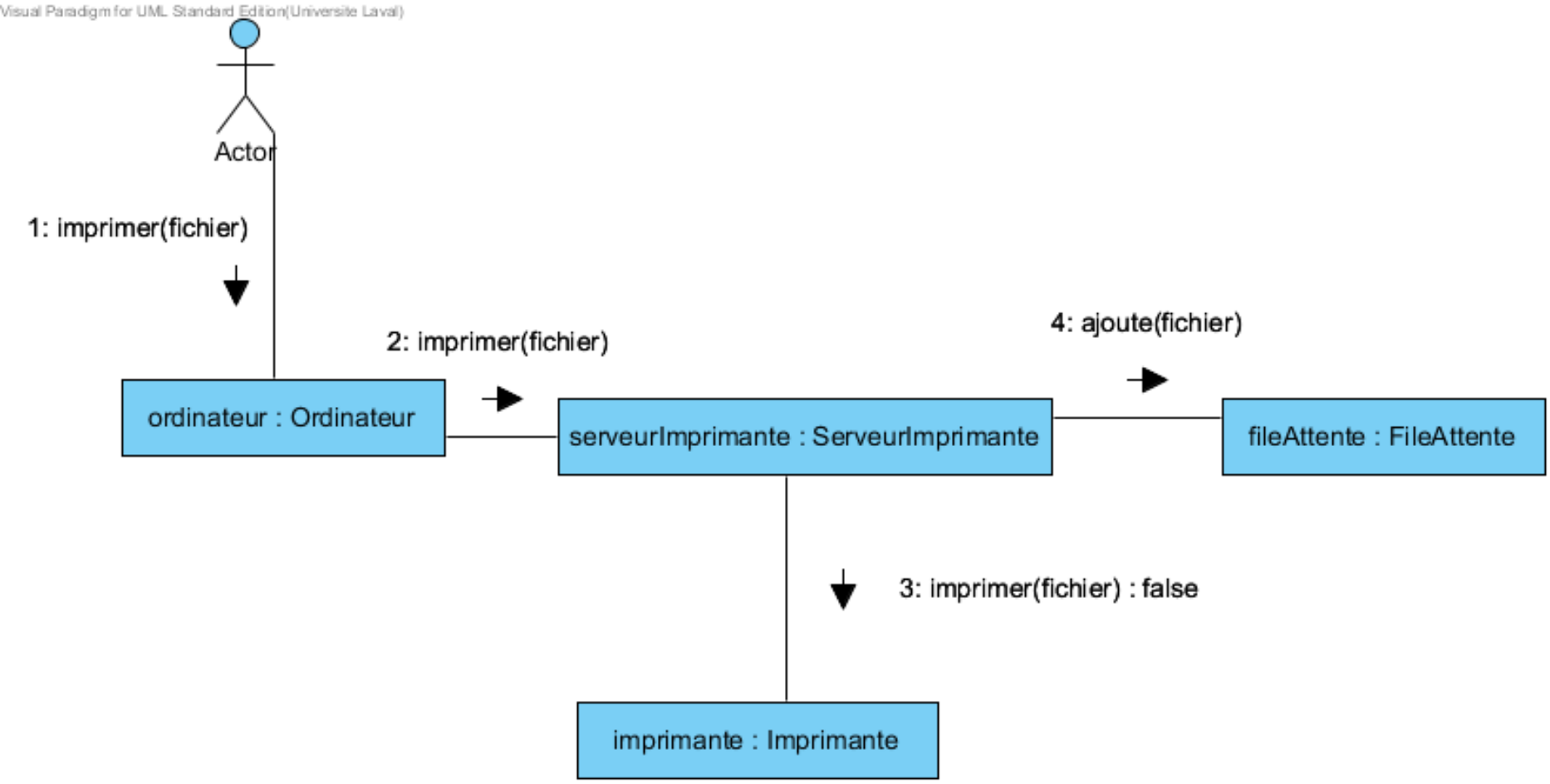
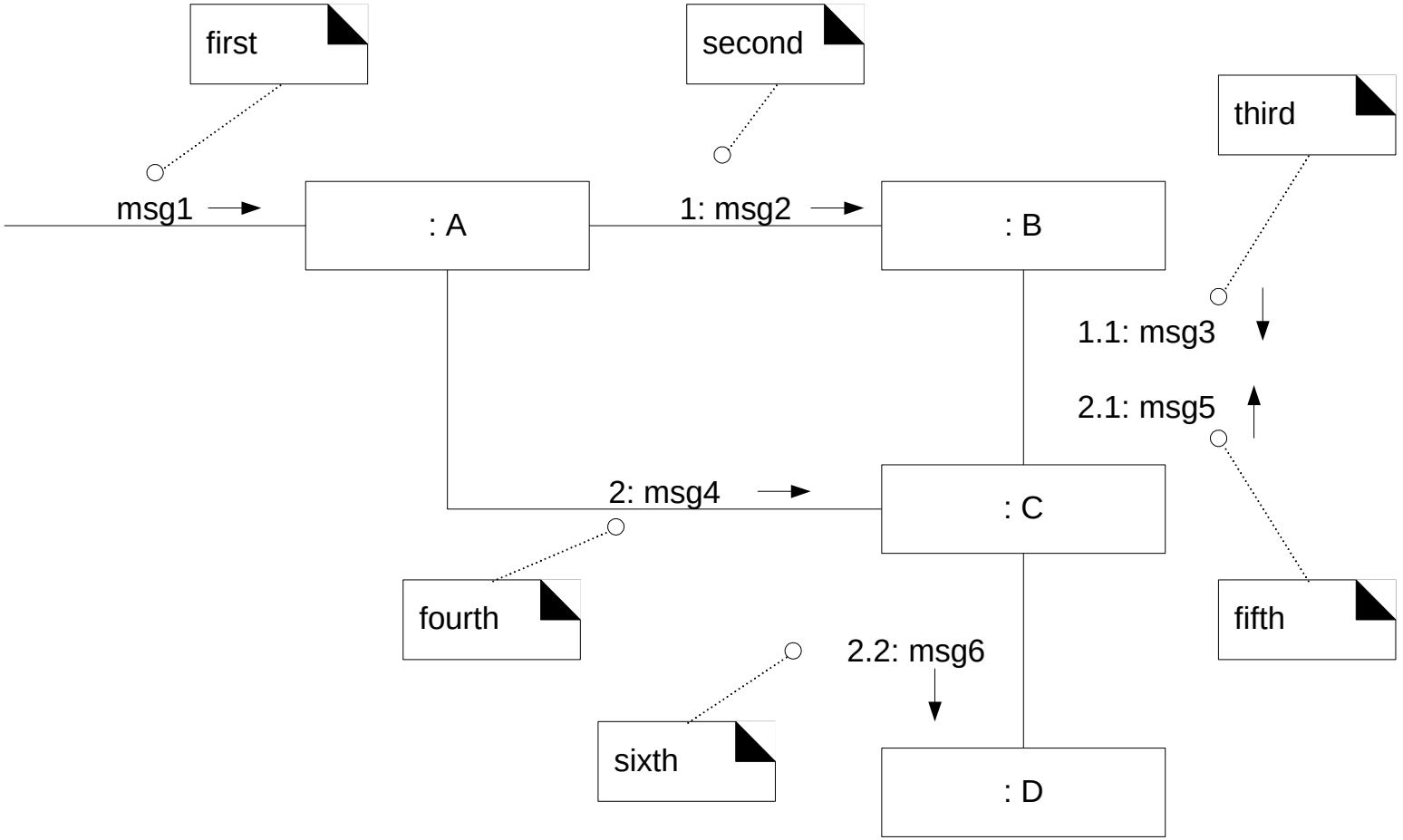
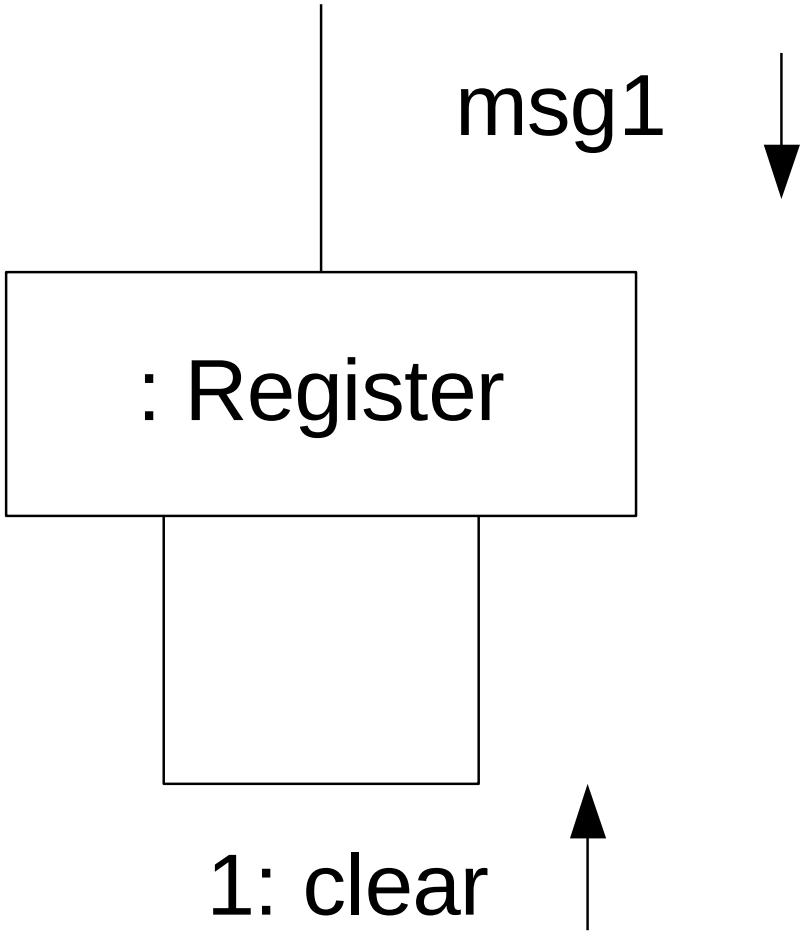


Diagramme de communication



L’objet appelle lui-même une de ses méthodes



Condition

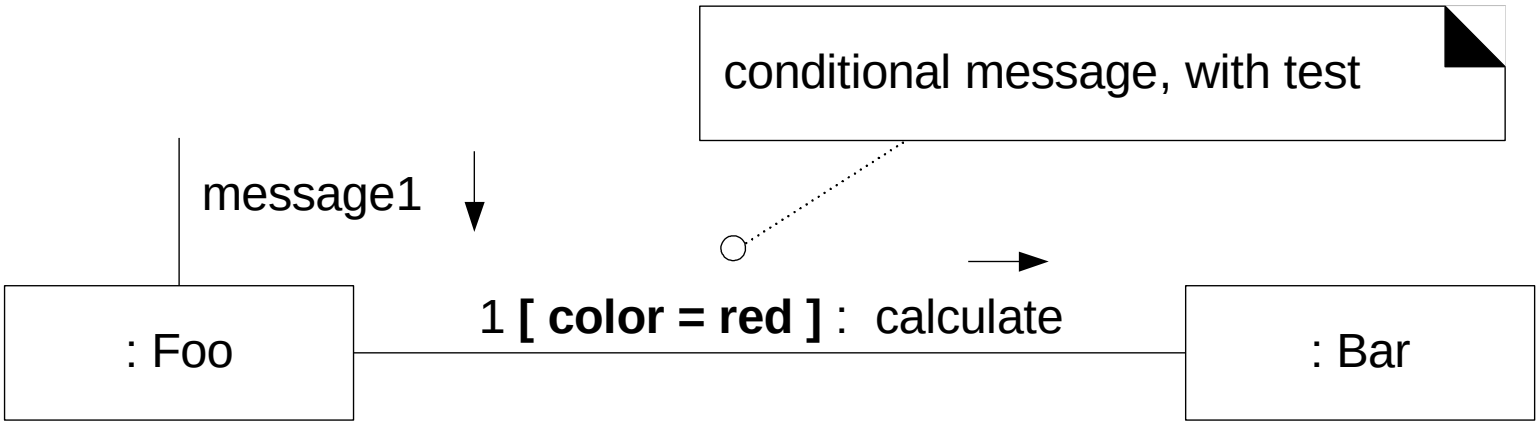


Diagramme de séquence ou de communication?

- Diagramme de séquence
  - Plus facile de voir la séquence des appels dans le temps
  - Notation UML plus expressive
  - Bien supporté dans la plupart des outils UML
- Diagramme de communication
  - Permet édition dans un espace limité (ex: tableau blanc)
  - Plus facile à modifier si on travaille à la main
  - Focus sur l'aspect spatial des relations

Quand créer un diagramme d'interaction?

- Pour spécifier ce qui se passe à l'interne du système lors de la réalisation d'un scénario

  (ou d'une partie d'un scénario)
- En pratique, on crée ces diagrammes pour réfléchir aux parties les plus complexes du système
- Facilite la collaboration/communication au sein de l'équipe (rappelez-vous toujours: l'équipe cherche à s'entendre sur un design)

Exercice - Tracez le diagramme de classes et le diagramme de séquence à partir du code (page suivante)

- En exercice
  - Tracez le diagramme de classes de conception correspondant à l'ensemble du code.
  - Tracez le diagramme de séquence pour un appel à l'opération *Manager.embauche.*

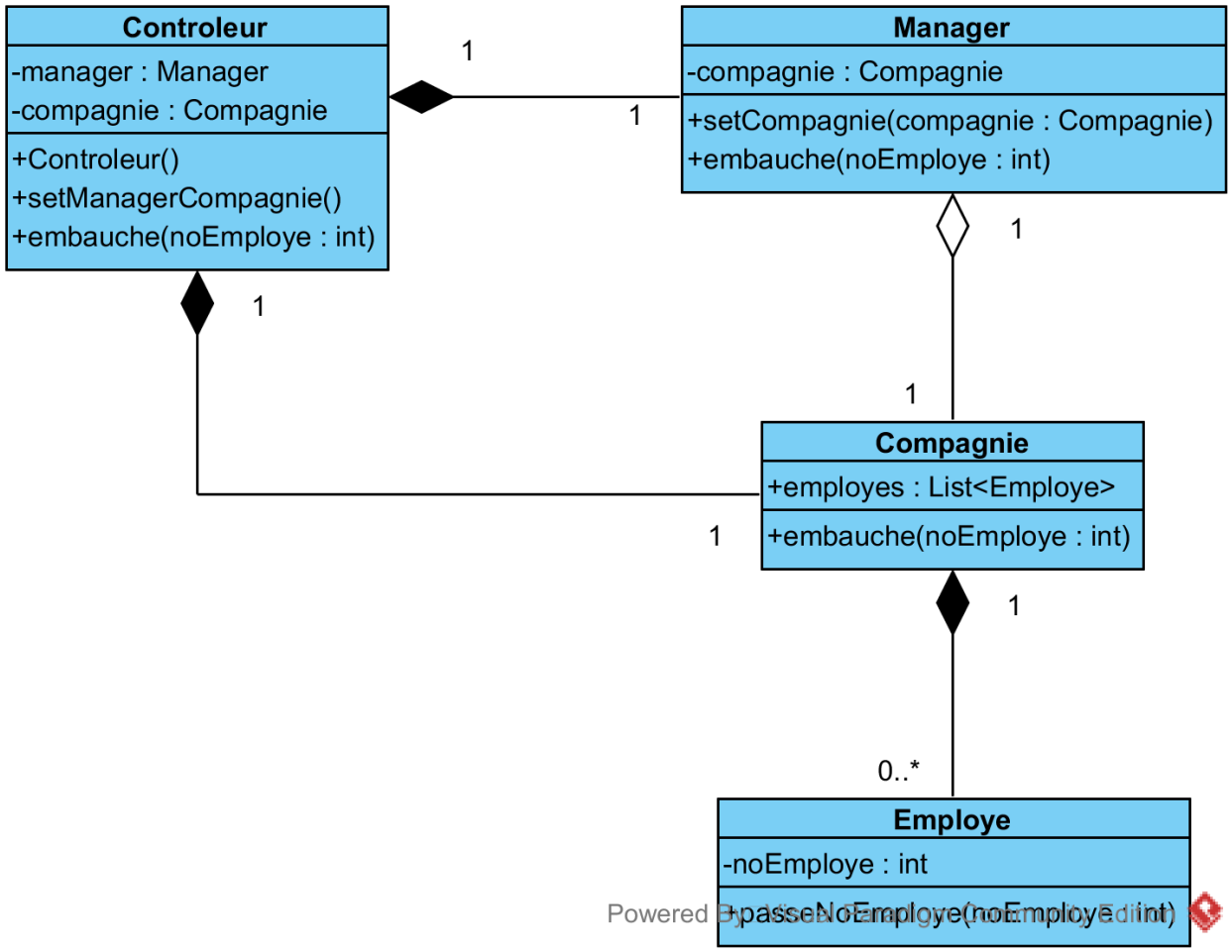
Tracez le diag. de classes et le diag. de séquence

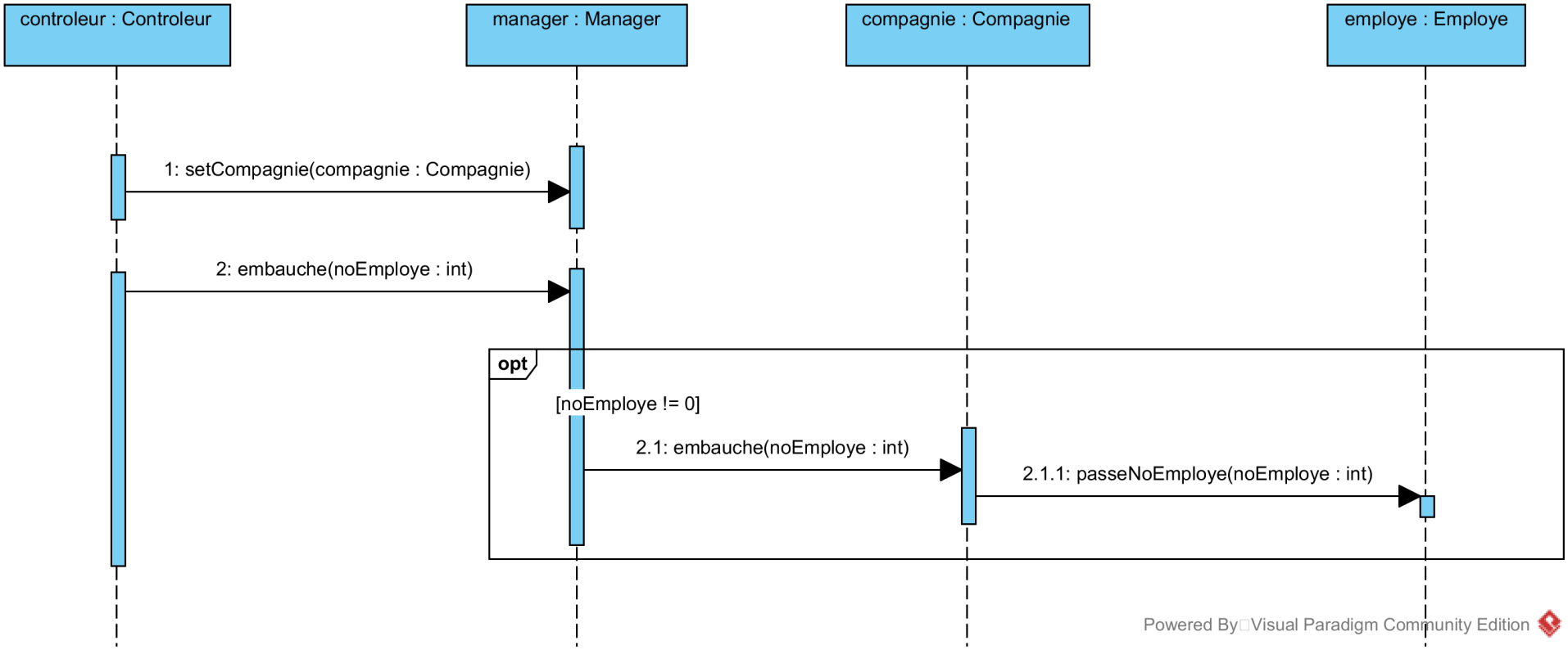
```
public class Controleur{
    private Manager manager;
    private Compagnie compagnie;

    public Controleur(){
        this.manager = new Manager();
        this.compagnie = new Compagnie();
    }

    public void setManagerCompagnie(){
        manager.setCompagnie(this.compagnie);
    }

    public void embauche(int noEmploye){
        manager.embauche(noEmploye);
    }
}
```

```
public class Manager{
    private Compagnie compagnie;

    public void setCompagnie(Compagnie compagnie){
        this.compagnie = compagnie;
    }

    public void embauche(int noEmploye){
        if(noEmploye != 0){
            compagnie.embauche(noEmploye);
        }
    }
}
```

```
public class Compagnie{
    public List<Employe> employes = new ArrayList();

    public void embauche(int noEmploye){
        Employe nouveau = new Employe();
        nouveau.passeNoEmploye(noEmploye);
        employes.add(nouveau);
    }
}
```

```
public class Employe{
    private int noEmploye;

    public void passeNoEmploye(int noEmploye){
        this.noEmploye = noEmploye;
    }
}
```


Controleur
-manager : Manager
-compagnie : Compagnie
+Controleur()
+setManagerCompagnie()
+embauche(noEmploye : int)
Manager
-compagnie : Compagnie
+setCompagnie(compagnie : Compagnie)
+embauche(noEmploye : int)
Compagnie
+employes : List<Employe>
+embauche(noEmploye : int)
Employe
-noEmploye : int
+passeNoEmploye(noEmploye : int)
1
1
1
1
1
1
1
1
0..*


controleur : Controleur
manager : Manager
compagnie : Compagnie
employe : Employe
1: setCompagnie(compagnie : Compagnie)
2: embauche(noEmploye : int)
opt
[noEmploye != 0]
2.1: embauche(noEmploye : int)
2.1.1: passeNoEmploye(noEmploye : int)
Powered By Visual Paradigm Community Edition

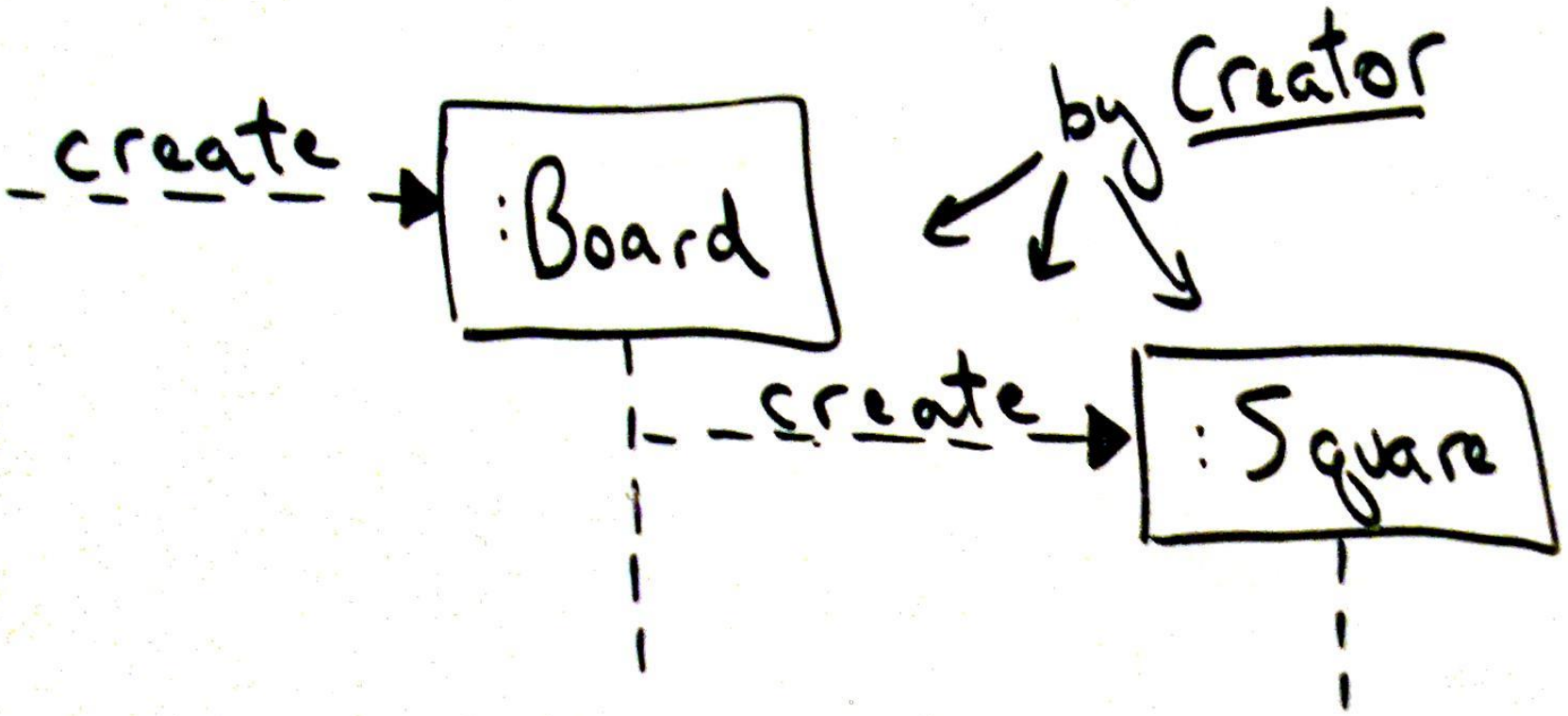
Module 11

- Grands principes en conception orienté objet
  - Créateur
  - Expert en information
  - Contrôleur
  - Faible couplage
  - Forte cohésion

Principe 1 : Créateur

- **Problème**:
  - Qui devrait créer les instances de la classe A?
- **Solution**:
  - La classe qui…
    - Contient ou agrège les A
    - Enregistre les A
    - Utilise étroitement les A
    - Possède les données pour initialiser des objets A

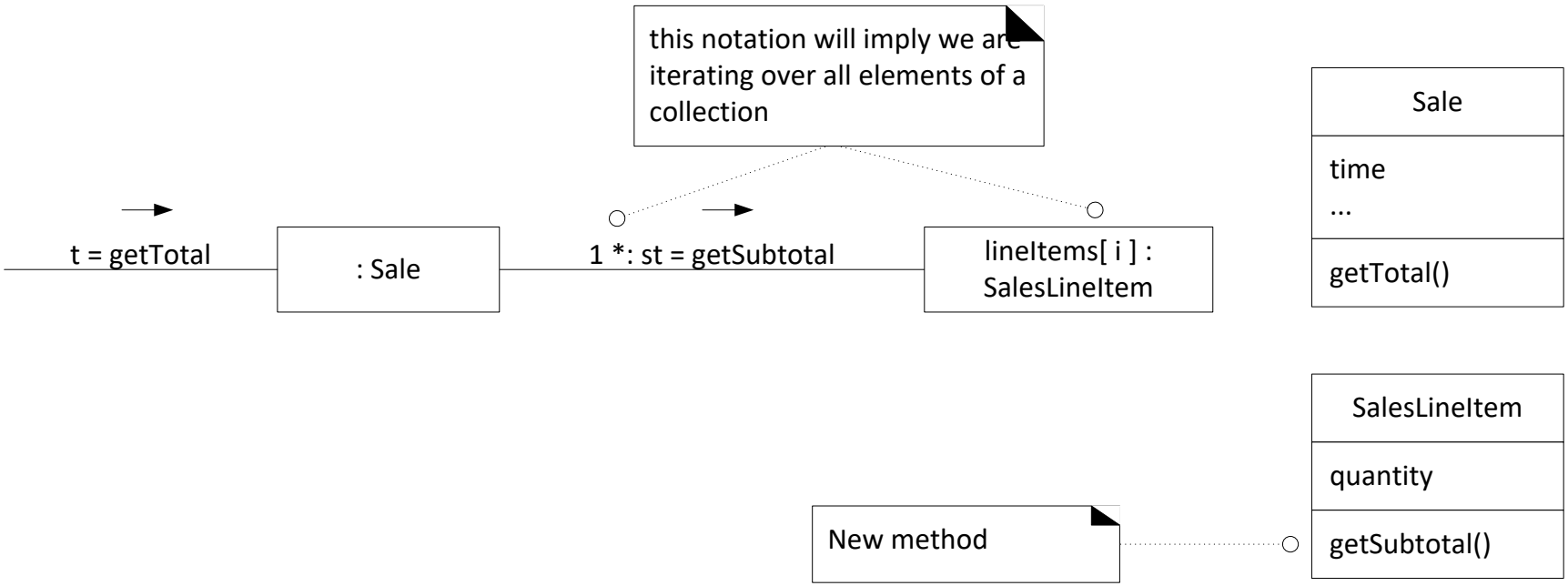
Le tableau de jeu !



Principe 2: Expert en information

- **Problème**:
  - À quel classe affecter une certaine responsabilité (méthode) ?
- **Solution**:
  - À la classe qui possède les informations nécessaires pour s'en acquitter

Qui calcule le montant total à exiger au client?



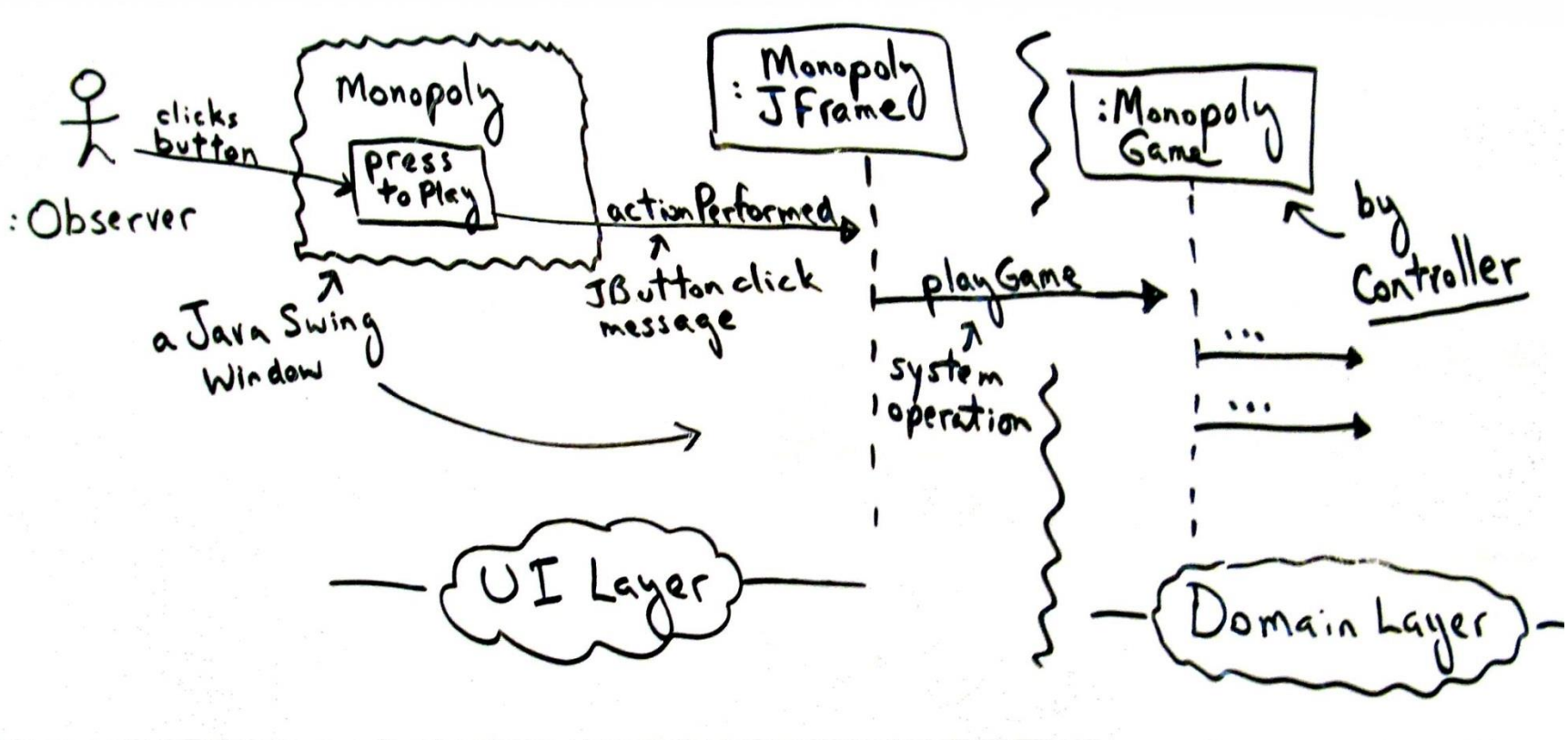
Analogie avec la vraie vie vraie

- En entreprise, à qui demande-t-on tel ou tel rapport?
- À celui qui a l'information!

Principe 3: Contrôleur

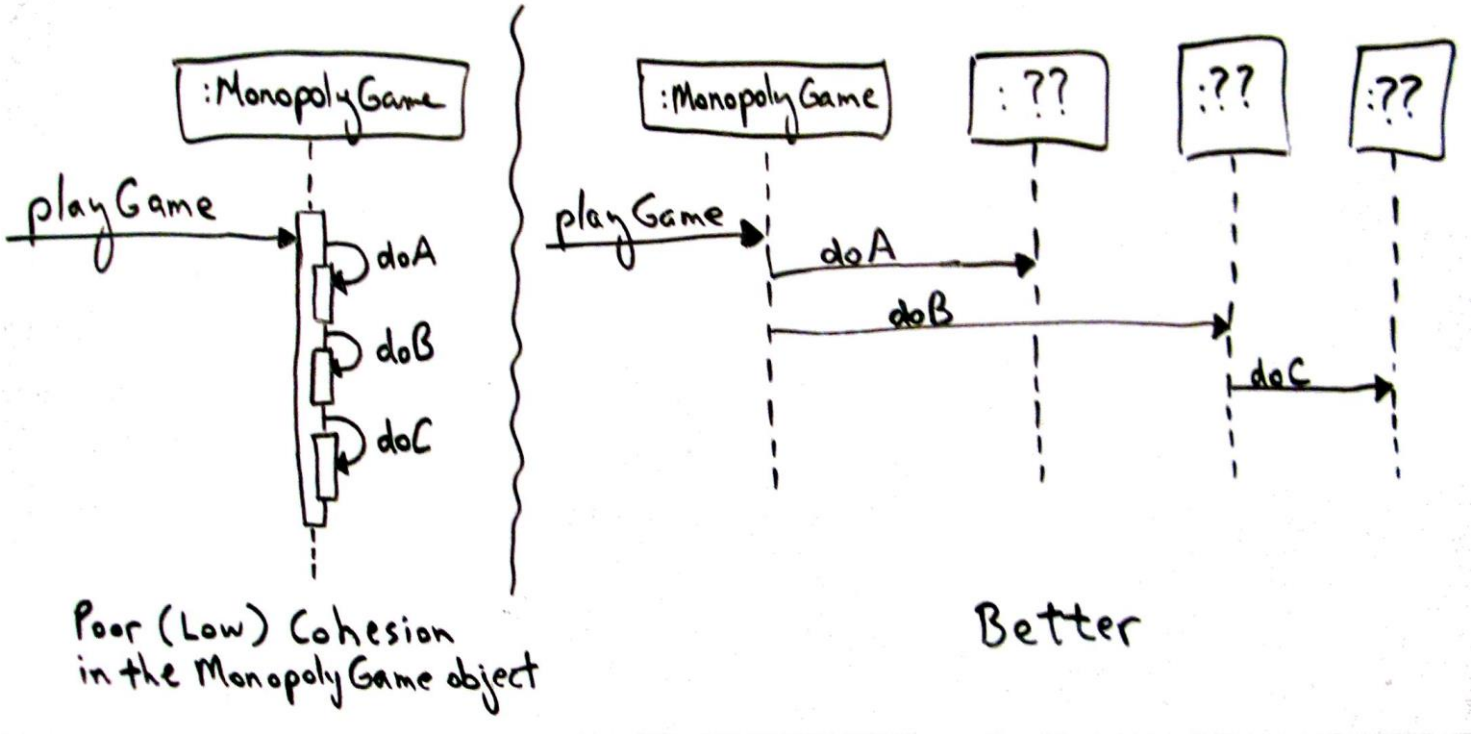
- **Problème**:
  - Quel est le premier objet au-delà de la couche présentation qui reçoit les « messages » de l'utilisateur et contrôle l'accès aux objets de la couche du domaine?
- **Solution**:
  - Généralement: un objet qui représente le « système global » ou un « objet racine »
  - Plus rarement: un objet qui représente un cas d'utilisation

Contrôleur pour Monopoly



Le rôle du contrôleur

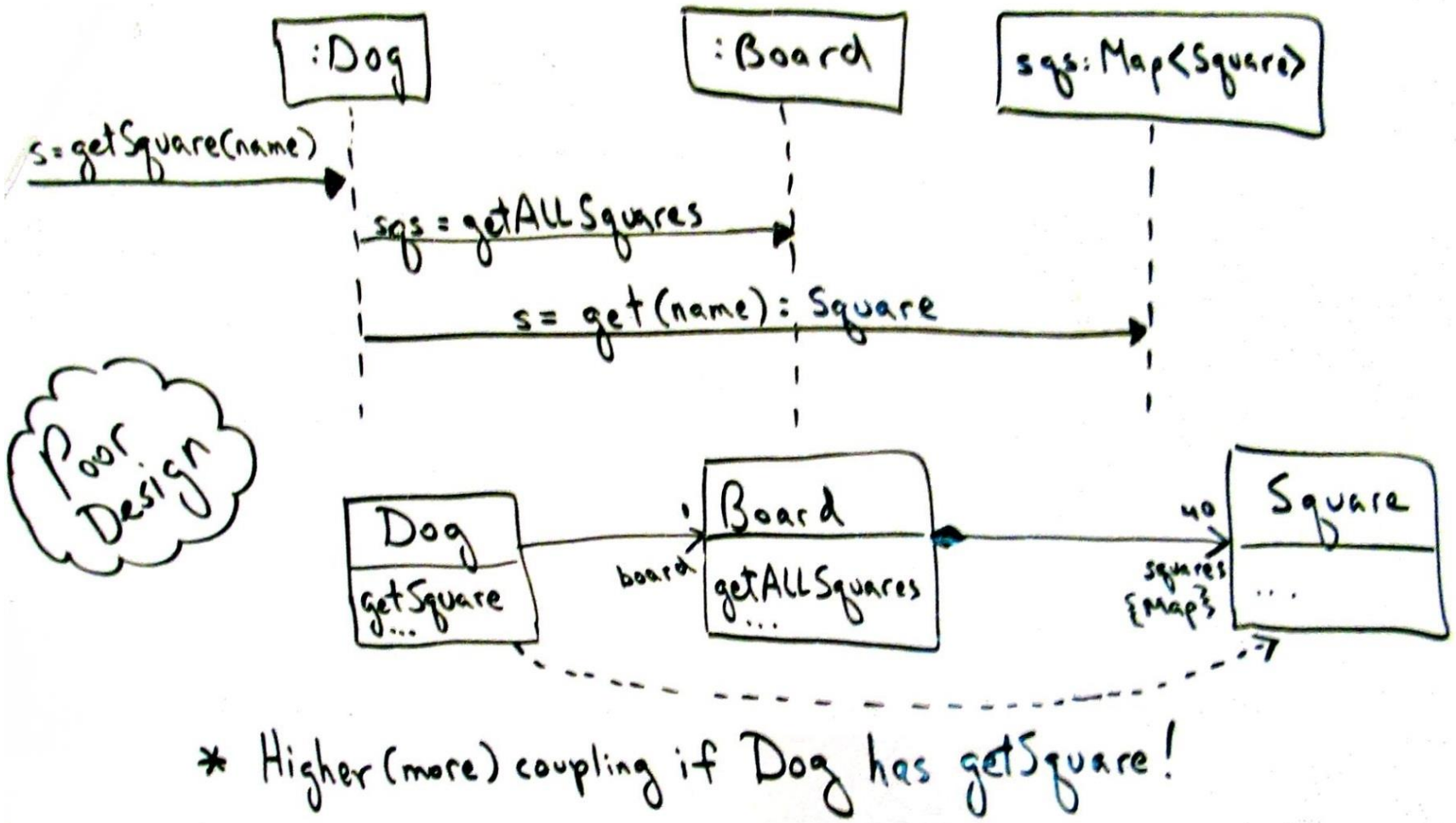
- Le contrôleur délègue les tâches aux autres objets. Il ne fait normalement pas faire grand-chose par lui-même



Principe 4: Faible couplage

- **Problème**:
  - Comment réduire l'impact des modifications futures?
- **Solution**:
  - Affecter les responsabilités aux classes de manière à éviter tout couplage inutile entre les classes

Où placer une méthode permettant de retrouver une case à partir de son nom?

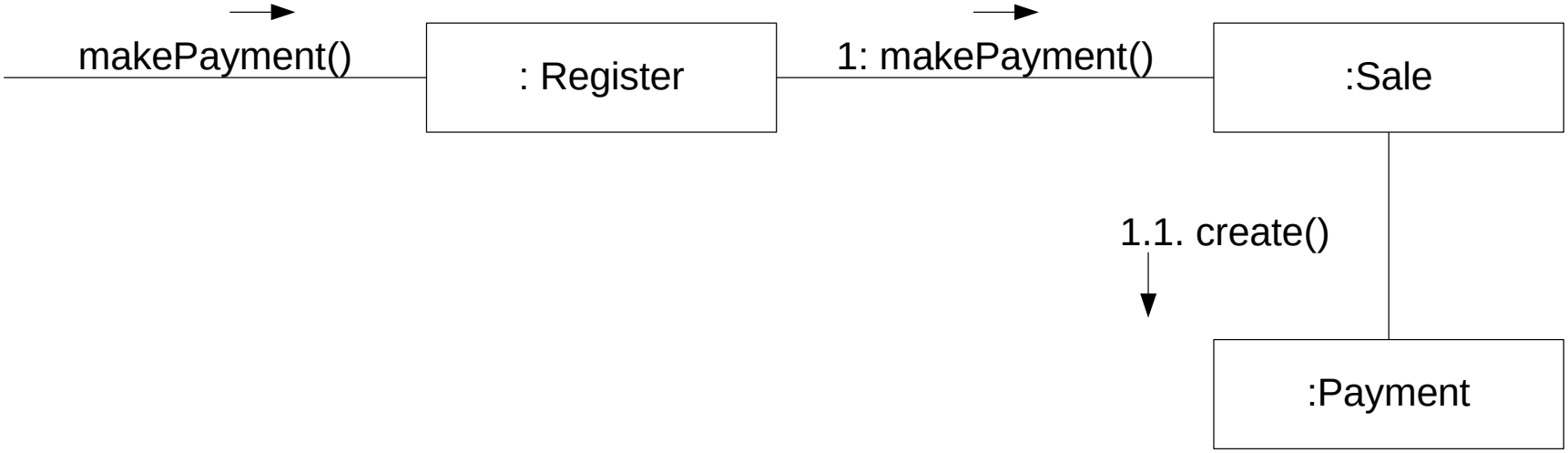


Principe 5: Forte cohésion

- **Problème**:
  - Comment s'assurer que les objets restent compréhensibles et faciles à gérer, et qu'ils contribuent au faible couplage?
- **Solution**:
  - Affecter les responsabilités de manière à ce que la cohésion demeure élevée
- **Attention**:
  - Va parfois à l'encontre d'autres principes

Qui devrait créer l'objet paiement?

- Selon le principe **Forte cohésion**:



UNIVERSITÉ LAVAL