

Génie Logiciel : Conception
Rappels sur UML

Sébastien Mosser
INF-5153, Hiver 2019, Cours #2.1

UQÀM

Avertissement

INF-5153 N'EST PAS UN COURS D'UML
JE NE SUIS PAS "PROF D'UML"
(même si ça peut en avoir l'air)

Crédits

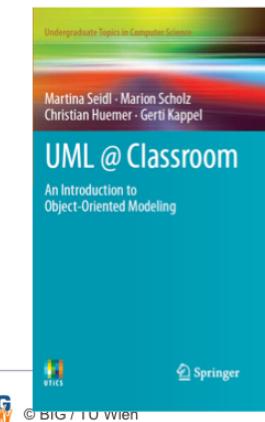
UNIVERSITÉ CÔTE D'AZUR

Philippe Collet
Université Côte d'Azur
Laboratoire I3S, SPARKS
"Prof d'UML" de père en fils depuis 1999

https://www.i3s.unice.fr/Philippe_Collet

Crédits (la suite) & Bibliographie

- The lecture is based on the following book:



**UML @ Classroom:
An Introduction to Object-Oriented
Modeling**
Martina Seidl, Marion Scholz, Christian
Huemer and Gerti Kappel
Springer Publishing, 2015
ISBN 3319127411

- Use Case Diagram
- Structure Modeling
- State Machine Diagram
- Sequence Diagram
- Activity Diagram

<http://www.uml.ac.at/en/>





Avertissement

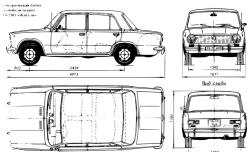
Le vocabulaire “usuel” est laissé en anglais pour vous permettre de trouver de la documentation facilement.

- 1 Pourquoi concevoir UML ?
- 2 Cas d'utilisations
- 3 Des Objets aux Classes
- 4 Modèles comportementaux

- Pourquoi concevoir UML ?
- 1

Pourquoi Modéliser ?

Spécifier la structure et le comportement d'un système



Aider à la construction d'un système



Visualiser un système



Documenter les décisions



P. Collet

5



D'où vient UML ?

- Dans les années 90 :
 - Pléthora de notations et de méthodes (OMT, Booch, ...)
- Personne n'y comprend plus rien !
 - Besoin d'un langage standard
- Création de l'entreprise Rationale
 - aka "La communauté de l'objet"
- Alliance des auteurs des méthodes existantes



UML, avec un U comme "Unifié"

depuis 1997

Mélange de plusieurs notations



Standardisation (OMG)

Dernière version : UML 2.5.1
(12.17, 800 pages)

*Au pays des objets où sétiendent les ombres
Un Langage pour les gouverner tous
Un Langage pour les trouver
Un Langage pour les amener tous,
Et dans les ténèbres les lier
Au pays des objets où sétiendent les ombres.*

Qu'est-ce qu'UML ?

Descriptions graphiques et textuelles

Syntaxe & Sémantique (ambiguïté ?)

Structure & Comportement

Ingénierie : avant ou rétro

Les points forts

- Un langage normalisé, donc ...

- précis (rappel : 800 pages de spécifications),
- stable (11 versions en 22 ans),
- et outillé.

• https://fr.wikipedia.org/wiki/Comparaison_des_logiciels_d%27UML

- Support de communication

- Cadre l'analyse et les abstractions
- Universel

VERSION	ADOPTION DATE
2.5.1	December 2017
2.4.1	July 2011
2.3	May 2010
2.2	January 2009
2.1.2	October 2007
2.0	July 2005
1.5	March 2003
1.4	September 2001
1.3	February 2000
1.2	July 1999
1.1	December 1997

Organisation d'UML

- 13 diagrammes, réalisés à partir des besoins utilisateurs
- Différentes aspects liés aux différentes facettes d'un système objet
 - Fonctionnel : Interactions entre Utilisateurs & Système
 - Statique : Conception structurelle (abstractions, relations)
 - Dynamique : Évolution des objets dans le temps
- On utilise uniquement ce dont on a besoin !
 - Mantra : "Small is beautiful", "Keep it Stupid, Simple", ...

Les points faibles

- UML est un langage

- Complexe et subtil

- Lourd

- Ambigu par endroits

- UML n'est pas une méthode !

- Rien pour trouver la BONNE conception

- Au mieux permet de discuter

- (ça tombe bien, il a été créé pour ça)

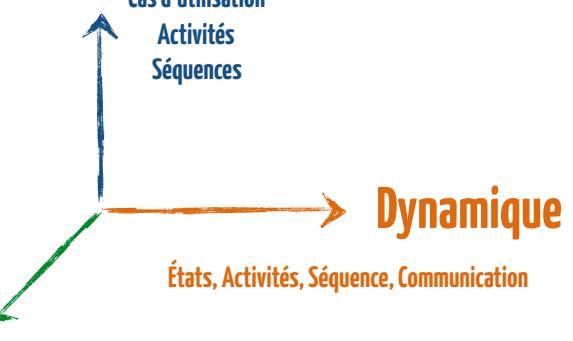


Principaux diagrammes UML

Fonctionnel

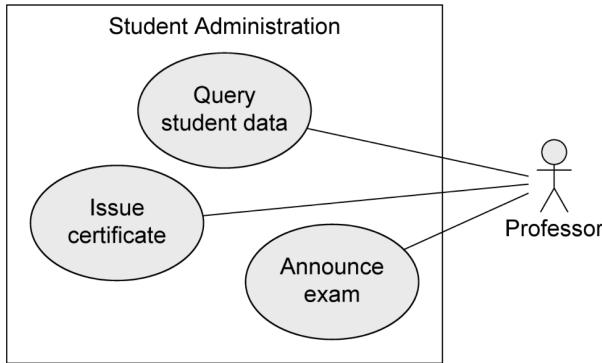
Cas d'utilisation
Activités
Séquences

Objets
Classes
Packages
Composants
Déploiement

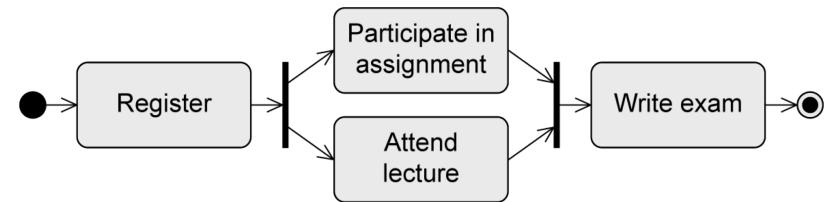


Statique

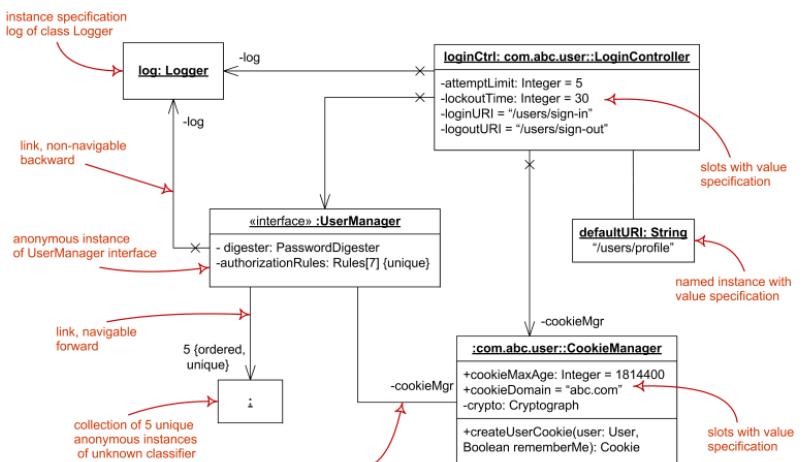
Cas d'utilisations



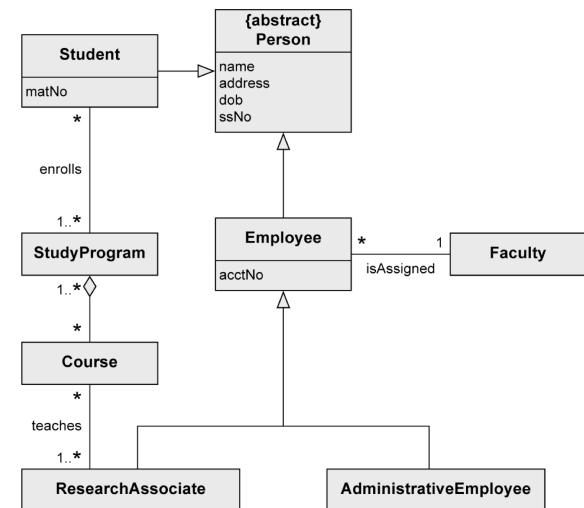
Activités



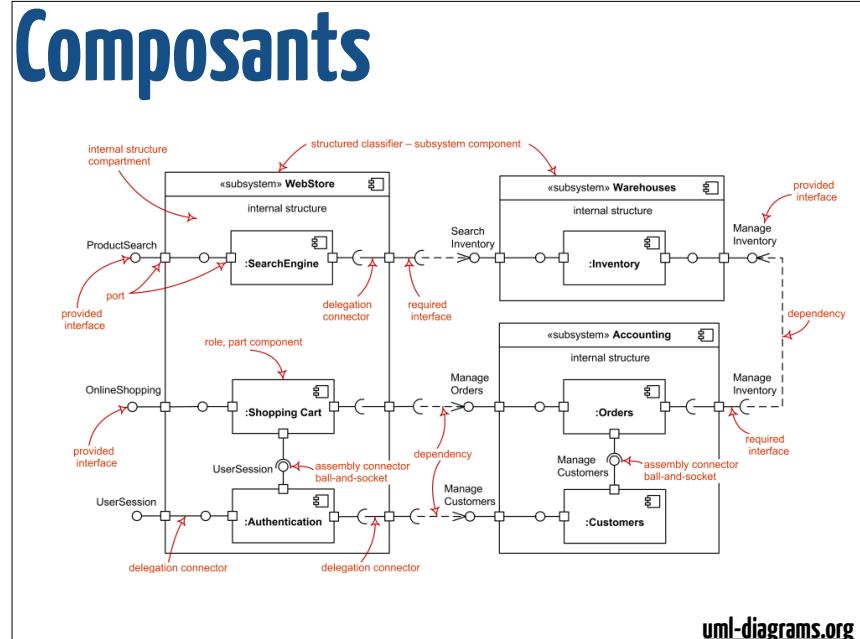
Objets



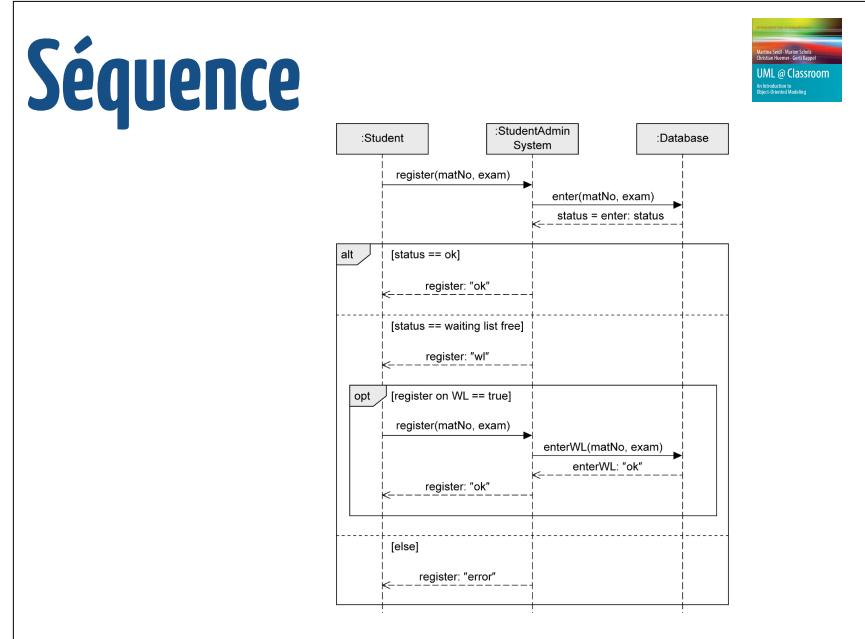
Classes



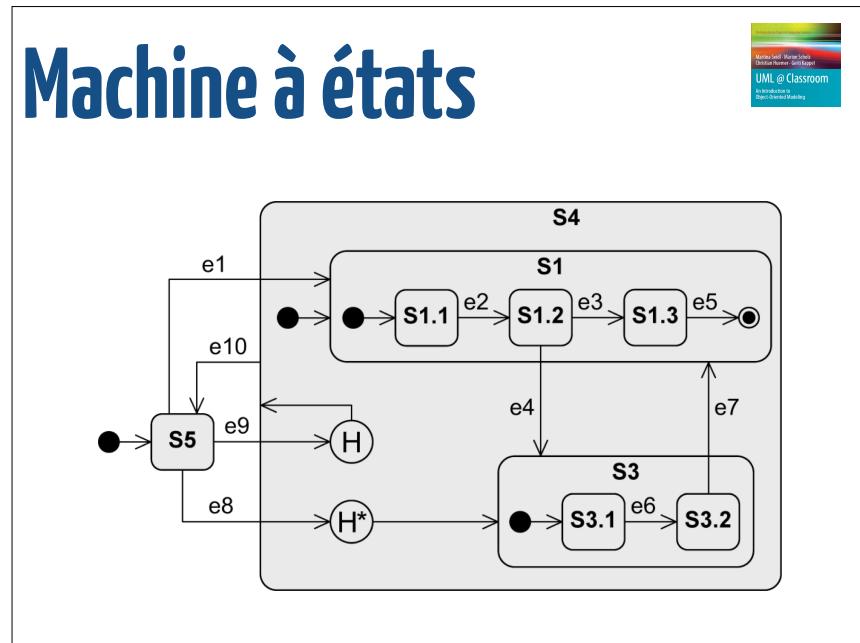
Composants



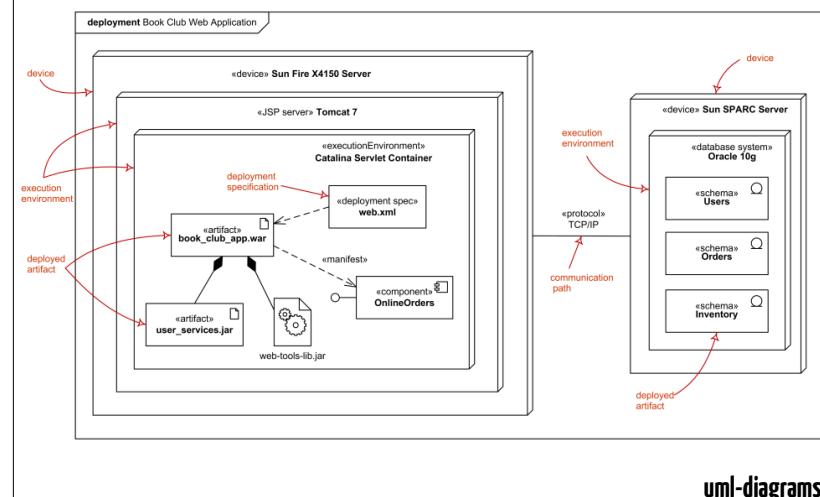
Séquence



Machine à états



Déploiement



Cas d'utilisations 2



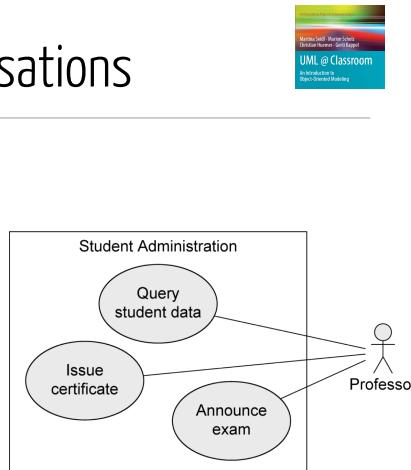
Pourquoi des cas d'utilisation ?

- Concept fondamental de plusieurs approches de modélisation objet
 - Attentes des clients et/ou des parties prenantes
 - Définis durant l'analyse, ils étayent la conception
- Un diagramme de cas d'utilisation répond aux questions suivantes :
 - Qu'est-ce qui est décrit dans le modèle de conception (Système)
 - Qui interagit avec le système (Acteurs)
 - Que peuvent faire les acteurs (Cas d'utilisations)



Exemple de cas d'utilisations

- Système :
 - Administration des étudiants
- Acteur :
 - Professeur
- Cas d'utilisation :
 - Requérir les informations étudiantes
 - Émettre un certificat
 - Annoncer la date d'un examen



Acteurs

- Les acteurs interagissent avec le système uniquement :
 - En initiant un cas d'utilisation (**Acteur primaire, souvent à gauche**)
 - En étant utilisé par un cas d'utilisation (**Acteur secondaire, souvent à droite**)
- Un acteur est le “rôle” endossé par un utilisateur
 - On peut être plusieurs acteurs en restant la même personne
- Les acteurs ne font pas parti du système !!
 - Les cas d'utilisation représente l'interface “fonctionnelle”
 - Un acteur n'est pas forcément un humain (p. ex. “Serveur de courriel”)

Identifier les acteurs

- Qui utilise les cas d'utilisation principaux ?
- Qui a besoin de support dans son travail quotidien ?
- Qui est responsable de l'administration du système ?
- Quels sont les systèmes externes avec lesquels le système va communiquer ?
- Qui est intéressé par les résultats des calculs fait par le système ?

(bonnes pratiques, à adapter)



Cas d'utilisations

- Décrivent les fonctionnalités attendues du système conçu
- Doit proposer un bénéfice tangible pour un ou plusieurs acteurs communiquant avec ce cas d'utilisation
- Dérivent de la collecte du besoin
- L'union des cas d'utilisation représente la totalité des fonctionnalités du système



Identifier les cas d'utilisation

- Quelles sont les tâches principales que les acteurs doivent effectuer ?
- Un acteur doit-il interroger ou modifier des données contenues dans le système ?
- Le système doit-il être informé de changements faits dans d'autres systèmes ?
- Doit-on informer un acteur d'événements imprévus ?

(bonnes pratiques, à adapter)

Description d'un cas d'utilisation

Structured approach

- Name
- Short description
- Precondition: prerequisite for successful execution
- Postcondition: system state after successful execution
- Error situations: errors relevant to the problem domain
- System state on the occurrence of an error
- Actors that communicate with the use case
- Trigger: events which initiate/start the use case
- Standard process: individual steps to be taken
- Alternative processes: deviations from the standard process



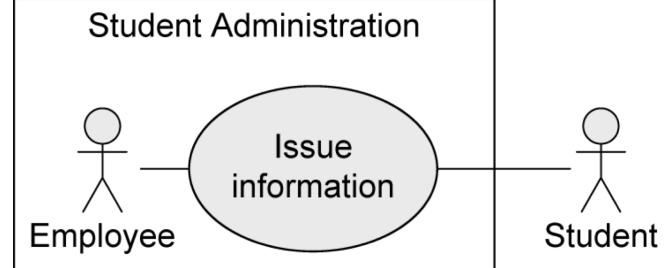
[A. Cockburn: Writing Effective Use Cases, Addison Wesley, 2000]



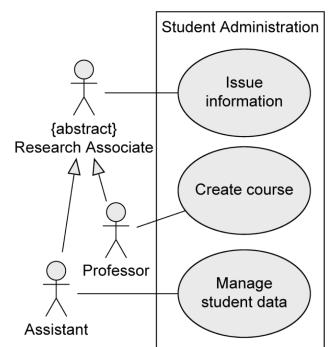
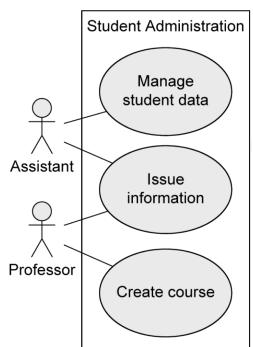
Mauvaises pratiques



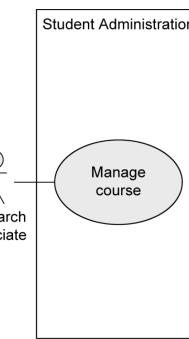
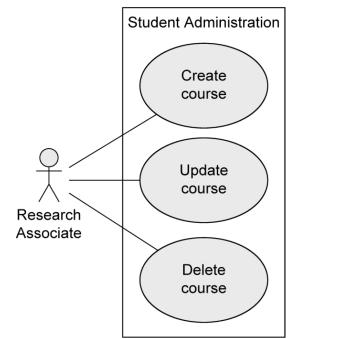
Mauvaises pratiques



Mauvaises pratiques



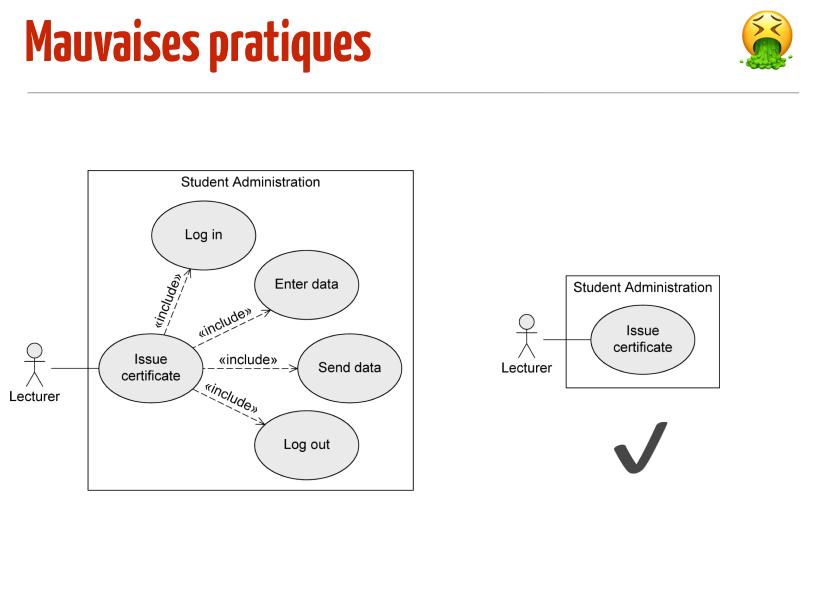
Mauvaises pratiques



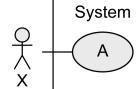
(ne pas abuser des "managers")



Mauvaises pratiques



Notation Elements (1/2)

Name	Notation	Description
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system



40

Notation Elements (2/2)

Name	Notation	Description
Association		Relationship between use cases and actors
Generalization		Inheritance relationship between actors or use cases
Extend relationship		B extends A: optional use of use case B by use case A
Include relationship		A includes B: required use of use case B by use case A

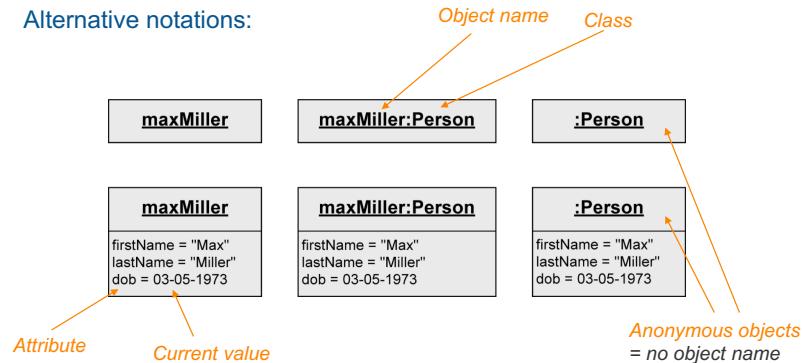


41

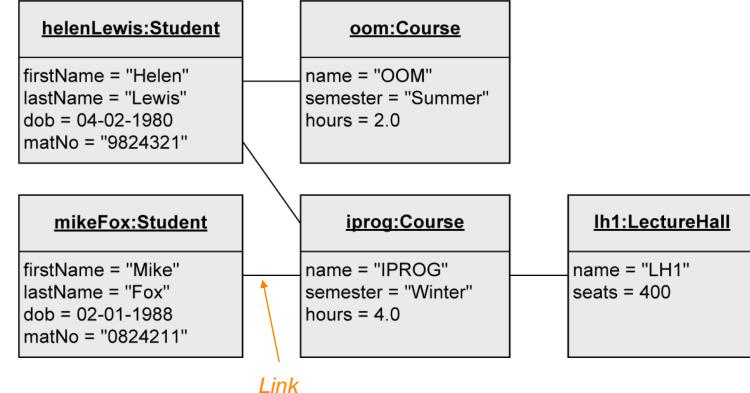
Des Objets aux Classes

Modélisation **Objet**, pas “Classe”

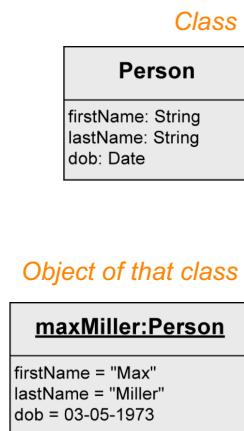
Alternative notations:



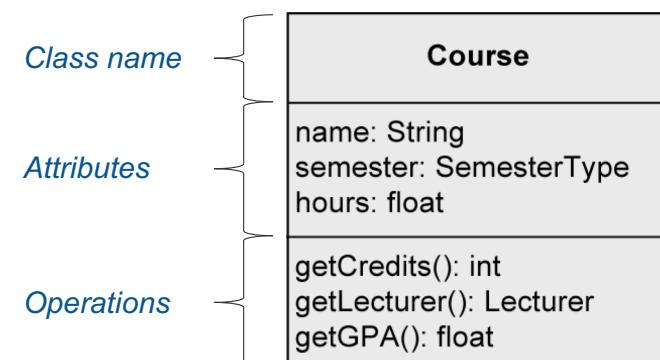
Diagrammes d’objets



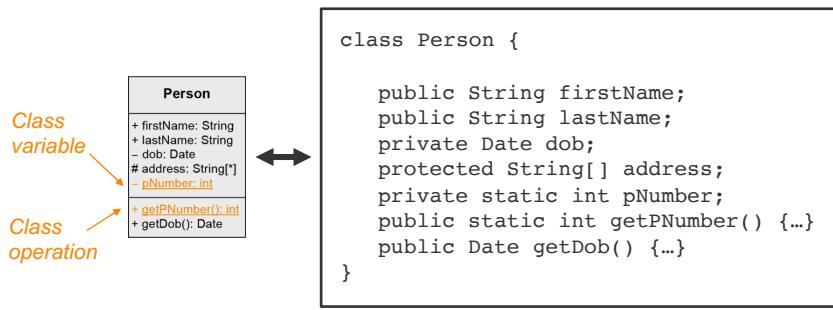
Classe : Fabrique d’objets



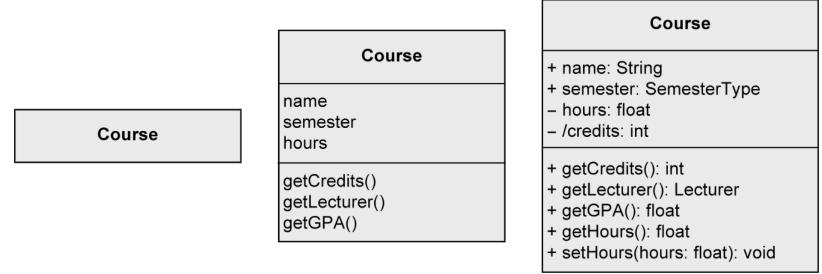
Éléments de Syntaxe



Liens entre modèle et code

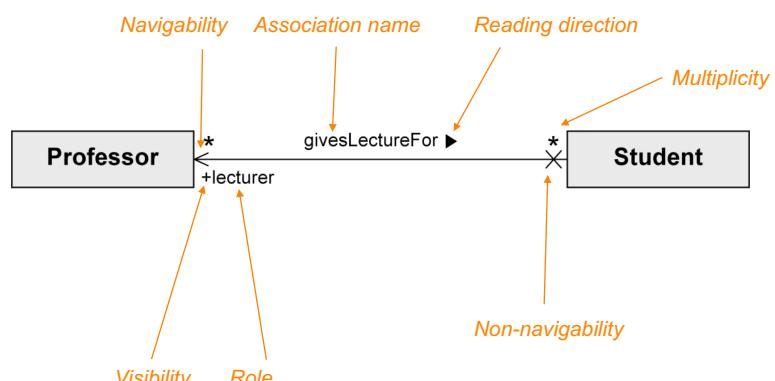


Différents niveaux de granularité

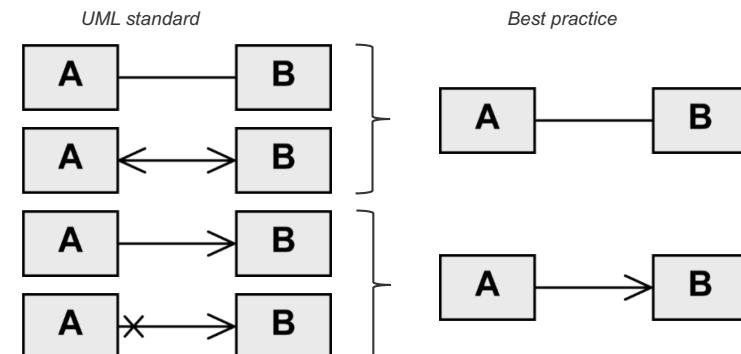


A adapter en fonction des besoins et de la complexité du modèle

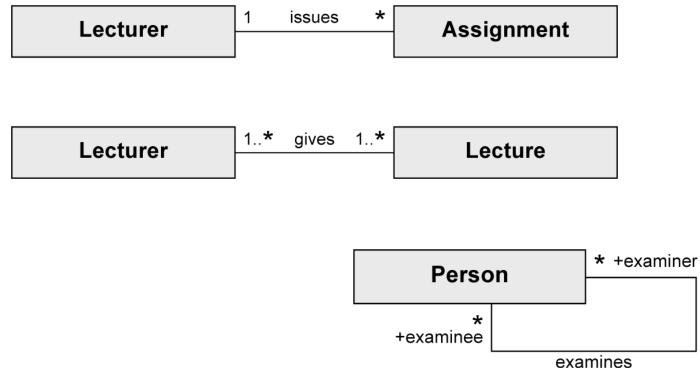
Association entre classes



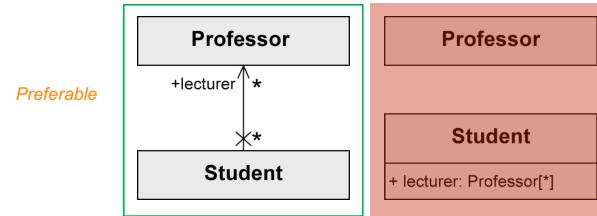
Navigabilité



Multiplicité & Rôles



Références = Associations



- Java-like notation:

```

class Professor {...}

class Student{
    public Professor[] lecturer;
    ...
}
  
```

Agrégation

- Exprime une relation d'appartenance “faible”
- “La partie peu exister sans son tout”.
- Amène à un graphe orienté acyclique d'appartenance



- Exemple :



- Un étudiant fait partie (ou non) d'une seule classe de laboratoire
- Un cours fait partie d'un ou plusieurs programmes d'étude

Composition

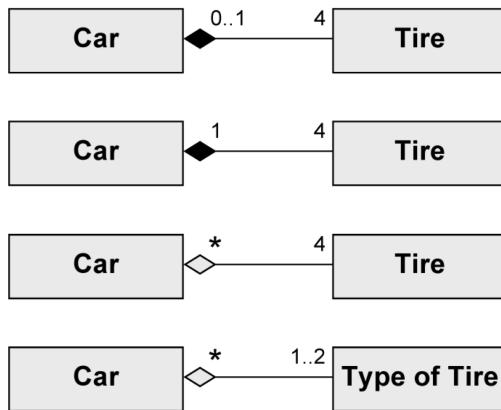
- Relation de dépendance explicite entre la partie et son tout
- Une partie peut être contenue par seulement un seul tout
- (donc multiplicité = 1 du côté de la composition, sinon c'est faux)
- Si le tout est détruit, alors la partie l'est aussi.
- Si je détruit INF-5153, allez-vous survivre à cette action? 😷



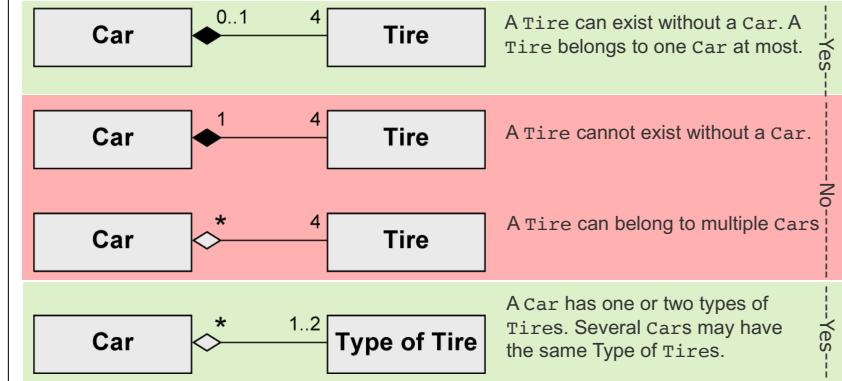
If the **Building** is deleted,
the **LectureHall** is also deleted

The **Beamer** can exist without the
LectureHall, but if it is contained in the
LectureHall while it is deleted, the **Beamer**
is also deleted

Agrégation vs Composition

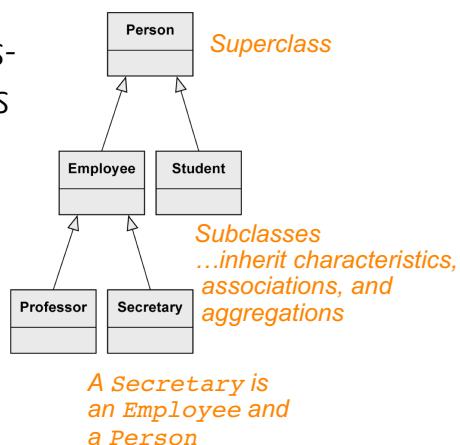


Agrégation vs Composition



Généralisation

Chaque instance d'une sous-classe est une instance des classes supérieures



Définit un ordre partiel sur les concepts

(vous faites de l'algèbre sans le savoir)

“Les mots ont du sens”

Même si UML est caricaturé comme un langage de “boîtes et de flèches”, il vient avec une sémantique forte qui permet de caractériser précisément un système

Corolaire :

souvent ce que vous écrivez ne veut pas dire ce que vous pensez !

Exemple : Généralisation (un peu d'algèbre)

Definition (Partial Order)

A **partial order** (A, \leq) is given by a set A and a binary relation \leq on A , such that

- for all $x \in A$: $x \leq x$ (reflexivity)
- for all $x, y, z \in A$: $x \leq y$ and $y \leq z$ imply $x \leq z$ (transitivity)
- for all $x, y \in A$: $x \leq y$ and $y \leq x$ imply $x = y$ (antisymmetry)

Definition (Total Order)

A partial order (A, \leq) is called a **total order**, if additionally

- for all $x, y \in A$: $x \leq y$ or $y \leq x$ or $x = y$ (trichotomy)

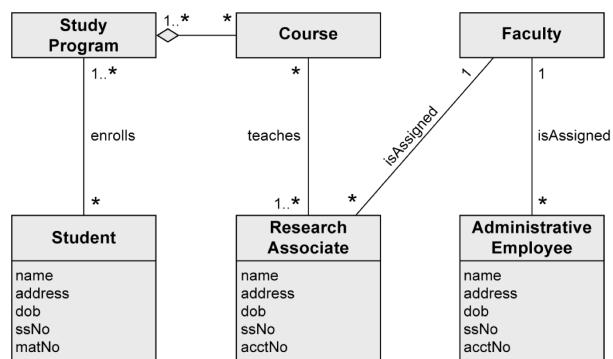
<http://theo.cs.ovgu.de/lehre/lehre16s/modelling/slides6.pdf>

Définition d'une sémantique formelle

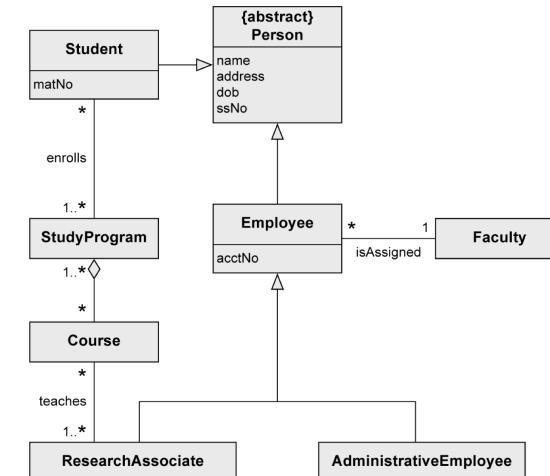
- A class hierarchy is given by a partial order (C, \leq)
 - antisymmetry means that cyclic subclasses are forbidden
- Each class $c \in C$ is interpreted as a finite set $\mathcal{S}(c)$
 - $\mathcal{S}(c)$ is the set of objects that are instances of class c
- If $c \leq d$, then $\mathcal{S}(c) \subseteq \mathcal{S}(d)$ must hold
 - hence, “each c is a d ”

<http://theo.cs.ovgu.de/lehre/lehre16s/modelling/slides6.pdf>

On peut vivre sans généralisation



On peut vivre sans généralisation



Rappels de Syntaxe



Name	Notation	Description
Class	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <pre>- a1: T1 - a2: T2 + o1(): void + o2(): void</pre> </div>	Description of the structure and behavior of a set of objects
Abstract class	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="margin-right: 10px;">oder</div> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">{abstract}</div> <div style="border: 1px solid black; padding: 2px; border-top-right-radius: 10px; border-bottom-right-radius: 10px;">A</div> </div>	Class that cannot be instantiated
Association	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">B</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">B</div> </div> <div style="display: flex; align-items: center; margin-top: 5px;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">B</div> </div>	Relationship between classes: navigability unspecified, navigable in both directions, not navigable in one direction

Rappels de Syntaxe



Name	Notation	Description
Shared aggregation	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">◆</div> <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">B</div> </div>	Parts-whole relationship (A is part of B)
Strong aggregation = composition	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">◆◆</div> <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">B</div> </div>	Existence-dependent parts-whole relationship (A is part of B)
Generalization	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; border-top-right-radius: 10px; border-bottom-right-radius: 10px;">→</div> <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">B</div> </div>	Inheritance relationship (A inherits from B)
Object	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">o</div> <div style="border: 1px solid black; padding: 2px; border-top-right-radius: 10px; border-bottom-right-radius: 10px;">C</div> </div>	Instance of a class
Link	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">o1</div> <div style="border: 1px solid black; padding: 2px; margin: 0 10px;">—</div> <div style="border: 1px solid black; padding: 2px; border-top-right-radius: 10px; border-bottom-right-radius: 10px;">o2</div> </div>	Relationship between objects

Rappels de Syntaxe



Name	Notation	Description
n-ary association	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">◆</div> <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">B</div> </div> <div style="text-align: center; margin-top: 10px;">C</div>	Relationship between n (here 3) classes
Association class	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">A</div> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">—</div> <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">B</div> </div> <div style="text-align: center; margin-top: 10px;">C</div>	More detailed description of an association
xor relationship	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">B</div> <div style="border: 1px solid black; padding: 2px; border-top-left-radius: 10px; border-bottom-left-radius: 10px;">{xor}</div> <div style="border: 1px solid black; padding: 2px; margin-left: 10px;">C</div> </div> <div style="text-align: center; margin-top: 10px;">A</div>	An object of C is in a relationship with an object of A or with an object of B but not with both

Modèles
comportementaux
(séquence uniquement)

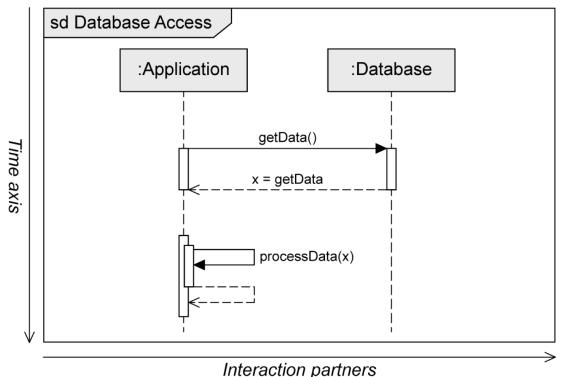


Diagrammes de séquence

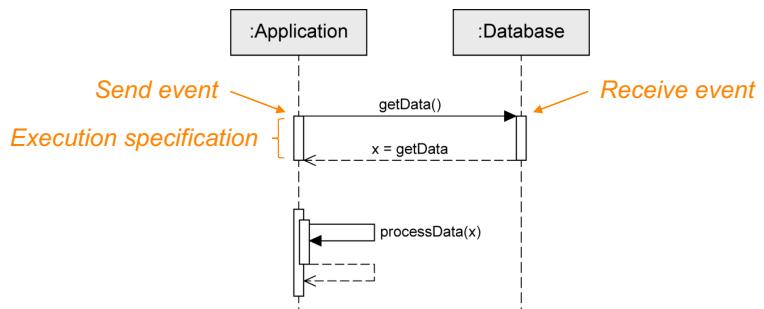
- Diagramme à deux dimensions :

- En X les interactions entre les partenaires

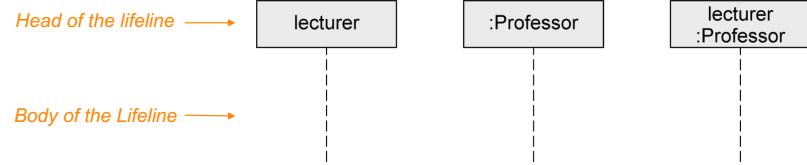
- En Y le temps



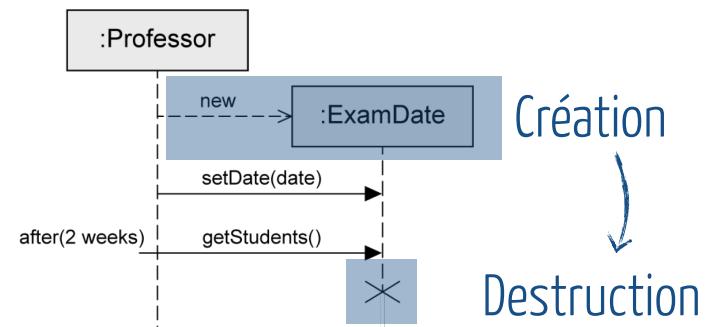
Échange de messages entre objets



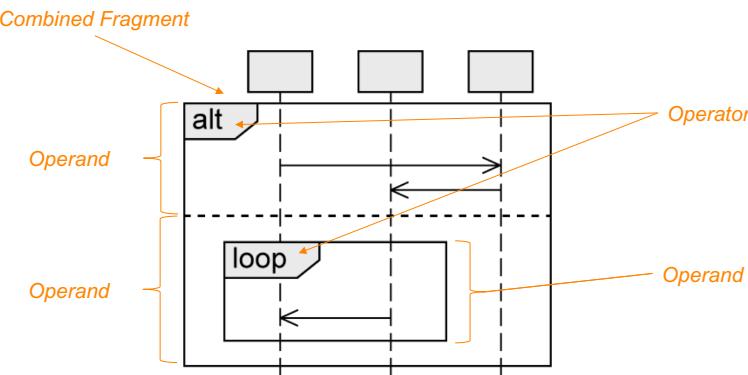
Lignes de vies



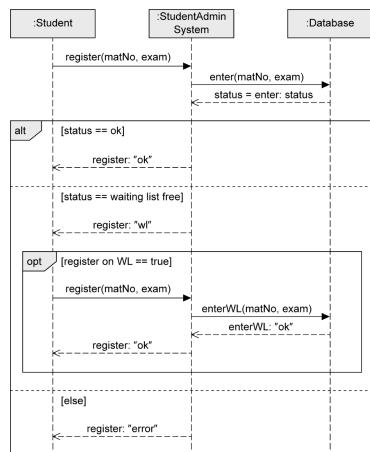
Création d'objets



Fragments combinés



Alternatives & Optionalité



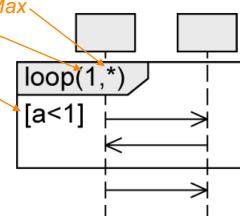
15

Attention, en conception,
on utilise UML pour
modéliser, pas pour
programmer.



Boucles

loop is
executed at
least once, then
as long as **a<1**
is true



Notation alternatives:

$loop(3,8) = loop(3..8)$
 $loop(8,8) = loop(8)$
 $loop = loop(*) = loop(0,*)$

17

UML pour modéliser, pas pour programmer

Elements de syntaxe



Name	Notation	Description
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs

Elements de syntaxe



Name	Notation	Description
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous communication		Sender continues its own work after sending the asynchronous message
Lost message		Message to an unknown receiver
Found message		Message from an unknown sender