

Conception Orientée Objet et notation UML

S. Ebersold

Conception orientée objets et UML

Plan du cours :

Introduction

Conception orientée objets et diagrammes UML

Analyse des besoins

Analyse / Conception des entités

Analyse / Conception des processus


Conclusion

Introduction : pourquoi UML

- Une dépendance du logiciel toujours plus forte
- Des économies mondiales dépendant de plus en plus du logiciel
- Des applications devenant plus grandes, plus importantes, plus complexes, plus distribuées
- Des métiers qui requièrent une plus grande productivité et une plus grande qualité avec un développement et un déploiement plus rapides
- Les achats d'entreprises : la refonte de leur système d'information, la mondialisation des filiales font qu'il est impératif d'avoir un langage de modélisation standardisé

Critères de succès des projets

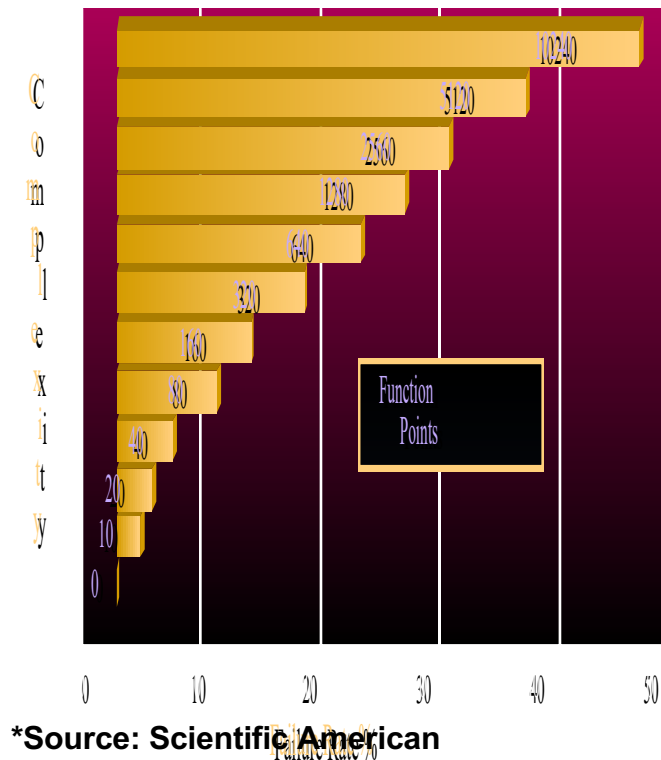
- Calendrier
- Complétude
- Performance
- Qualité
- Tolérance aux pannes
- Adaptabilité
- Extensibilité
- Portabilité
- Réutilisation (d'architecture)
- ...



Il est *impossible*
d'optimiser un système
pour répondre à *tous*
ces besoins en une fois!

Expérience avec Approches Traditionnelles

■ Dépassement de budget



- Temps de mise sur le marché inacceptable
- Faible qualité et fiabilité
- Résultat: quand les métiers en dépendent, le logiciel est vu comme une contrainte, et non comme un acquis stratégique

Raisons principales

- Gestion ad-hoc des besoins
- Communications imprécises et ambiguës
- Architectures fragiles
- Complexité écrasante
- Incohérences non détectées parmi les besoins, la conception et l'implémentation
- Tests insuffisants
- Évaluation subjective de l'état du projet
- Développement en cascade
- Propagation des changements non contrôlés
- Automatisation insuffisante

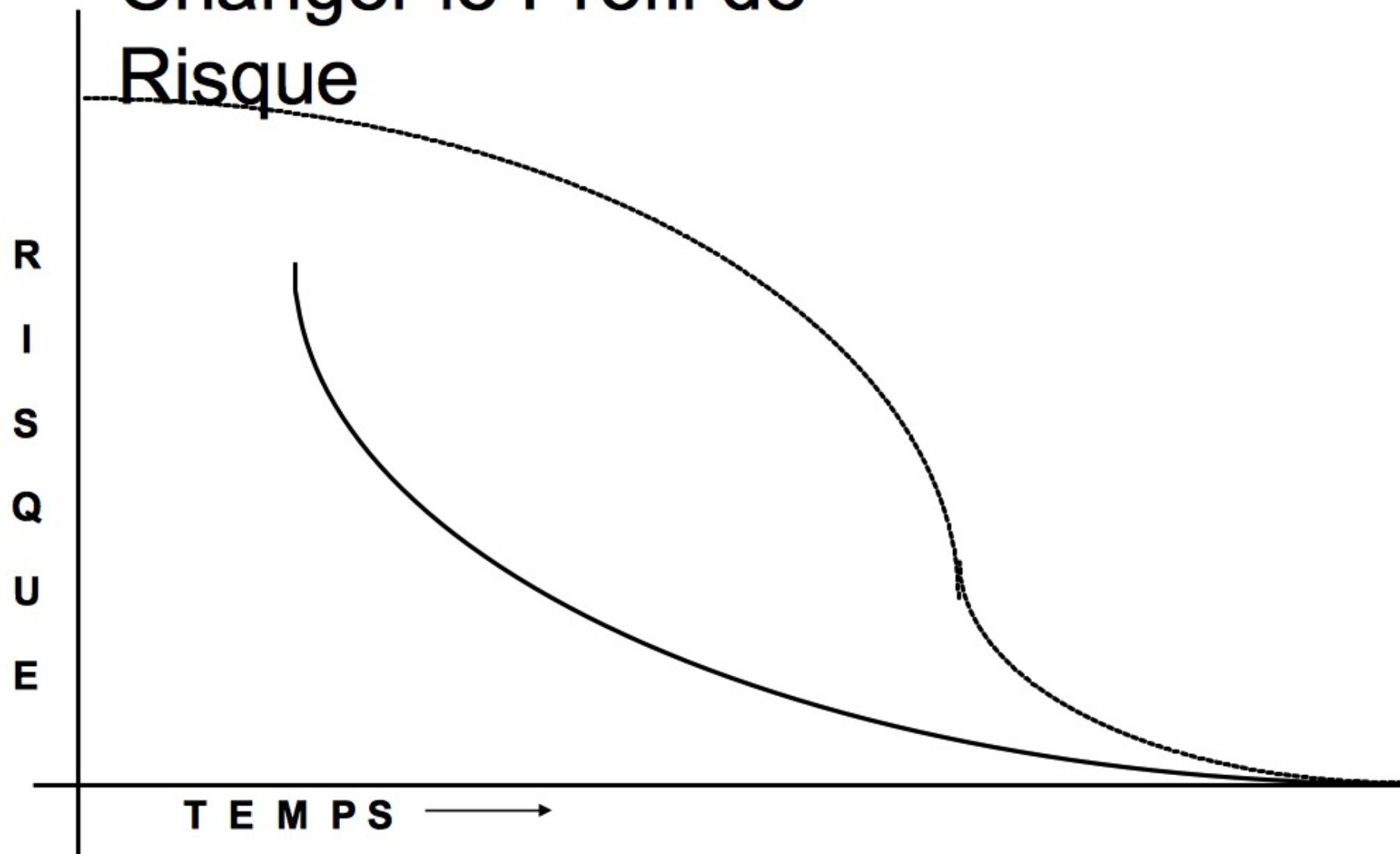
Sur quoi faut-il compter ?

- Matériel plus performant
- Langages de programmation
- Méthodes formelles
- Environnements de développement
- Bases de données
- Middleware
- Processus

NON !

-

Changer le Profil de Risque



Modéliser le logiciel de manière visuelle

La modélisation visuelle améliore la capacité à gérer la complexité logicielle

- Capturer la structure et le comportement des architectures et des composants
- Montrer comment les éléments d'un système s'articulent ensemble
- Cacher ou exposer les détails selon la tâche
- Maintenir la cohérence entre la conception et son implémentation

Promouvoir une communication non ambiguë

Réponse dans les Standards ?

- ISO 9000s, SEI/CMM, SPICE, ...
- Réponse partielle
 - ☐ Accent sur la documentation
 - ☐ Pas de processus clairement défini
 - ☐ Cadre d'évaluation
 - ☐ Orienté Qualité
 - ☐ Se placent à un autre niveau

Changer de Modèles !

- Le processus de développement du système informatique est un sous processus du processus de construction du métier
- On construit un Système d'information
 - pour faciliter des processus métiers
 - pour permettre des processus
- L'informatique est au service du métier et non l'inverse

Les avantages de l'approche UML

- La standardisation du langage de modélisation
- L'*utilisateur* au cœur de la démarche de conception du système
- Vers une gestion des connaissances de l'entreprise grâce à la démarche de conception de son système d'information

La modélisation

- *«Action d'élaboration et de construction intentionnelle, par composition de symboles, de modèles susceptibles de rendre intelligible un phénomène perçu complexe et d'amplifier le raisonnement de l'acteur projetant une intervention délibérée au sein du phénomène: raisonnement visant notamment à anticiper les conséquences de ces projets d'action possibles»*

J-L. Lemoigne, 1990

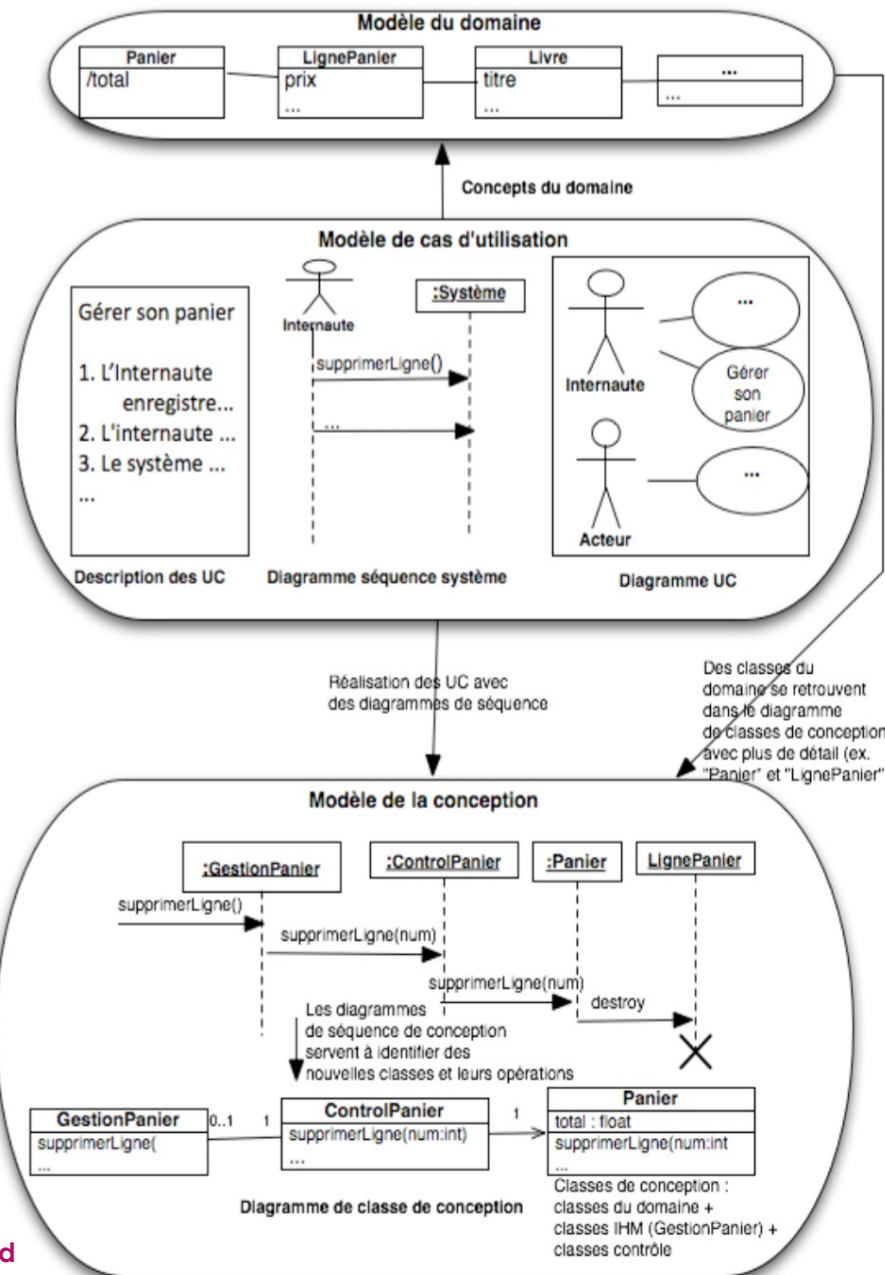
Le rôle des modèles

- Un modèle est une abstraction de quelque chose de réel qui permet de comprendre avant de construire, ou de retrouver les informations nécessaires pour effectuer des modifications et extensions
- Le modèle simplifie la gestion de la complexité en offrant des points de vue et niveaux d'abstractions + ou - détaillés selon les besoins

Le rôle des modèles

- Faciliter la compréhension humaine et la communication
- Supporter l'amélioration des processus de développement et de maintenance
- Supporter la conduite et la gestion des processus de développement et de maintenance
- Assurer un guidage automatique des acteurs
- Faciliter l'automatisation des processus de développement et de maintenance

Conception Orientée Objets et UML



Conception objet

+ organisation classes en packages UML
(composants) = Architecture logique

UML : Unified Modelling Language for Object oriented development

- UML et L'approche Objet
- La genèse d'UML
- La notation UML
- Les diagrammes d'UML

UML et L'approche objets

- Origine : Intérêts de l'approche objets
 - Stabilité de la modélisation
 - Construction itérative facilitée

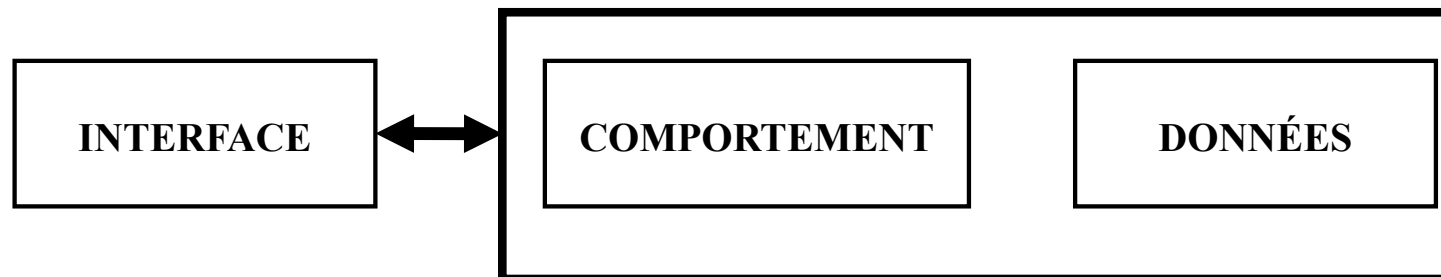
- L'approche objets :
 - Démarche de décomposition-réunion
 - Simplicité du modèle :
 - 4 concepts fondamentaux :
Classes, Objets, Messages, Héritage

UML et L'approche objets

- *Principe fondamental de l'approche par objets*
 - Programme → Organisme
 - Fonctions assurées → Objets manipulés
 - **Ce que fait le système → A qui il le fait**
- ou**
Qui fait Quoi

UML et L'approche objets

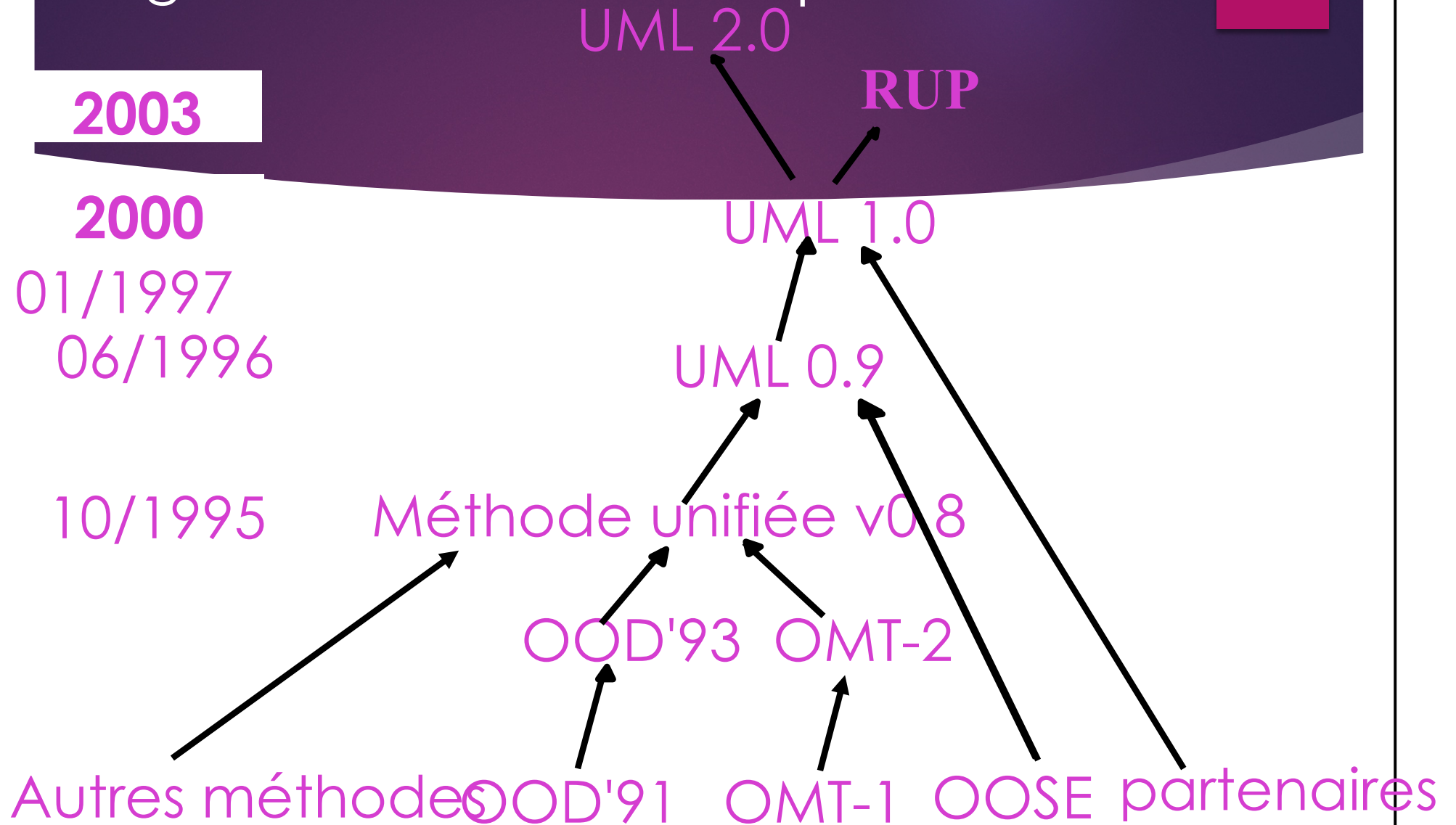
Encapsulation des données et des traitements concernant ces données
Communication entre objets - Relation Client / Serveur



La genèse d'UML

- Besoin de méthodes
- Besoin d'unification des méthodes existantes
 - Un grand nombre de méthodes orientées objet
 - Rapprochement de OOD et OMT

La genèse d'UML: Historique



La genèse d'UML : Éléments conservés

G. Booch
Embley
Fusion

Gamma et al.
Harel

I. Jacobson
B. Meyer
Odell

OMT
Shlaer-Mellor
Wirfs-Brock

Catégories et sous-systèmes
Classes singleton et objets composites
Description des opérations, numérotation
des messages
Frameworks, patterns et notes
Automates (state charts)
Cas d'utilisation (use cases)
Assertions sémantiques (pre-, postconditions)
Classification dynamique, éclairage
sur les événements
Associations
Cycle de vie des objets
Responsabilités (CRC)

La notation UML

- Concepts de base de la notation UML
 - Les éléments communs
 - Éléments de modélisation
 - Éléments de visualisation
 - Les types primitifs
 - Booléens, Expression, Liste, Point, Chaîne, Temps, Non-interprété, Multiplicité , Nom

La notation UML

UML : c'est aussi

- Le langage d'expression des contraintes OCL
- La construction de profils afin d'étendre la langage même
- Un formalisme unique pour représenter le méta-modèle et les modèles

Les diagrammes UML

- Diagrammes de classes (classes, fonctionnalités, relations)
- Diagrammes de cas d'utilisation (interactions des utilisateurs avec le système)
- Diagrammes d'objets (exemples de configurations d'instances)
- Diagrammes de collaboration (interactions entre objets : accent mis sur les liens)
- Diagrammes de séquences (interactions entre objets : accent mis sur les séquences)

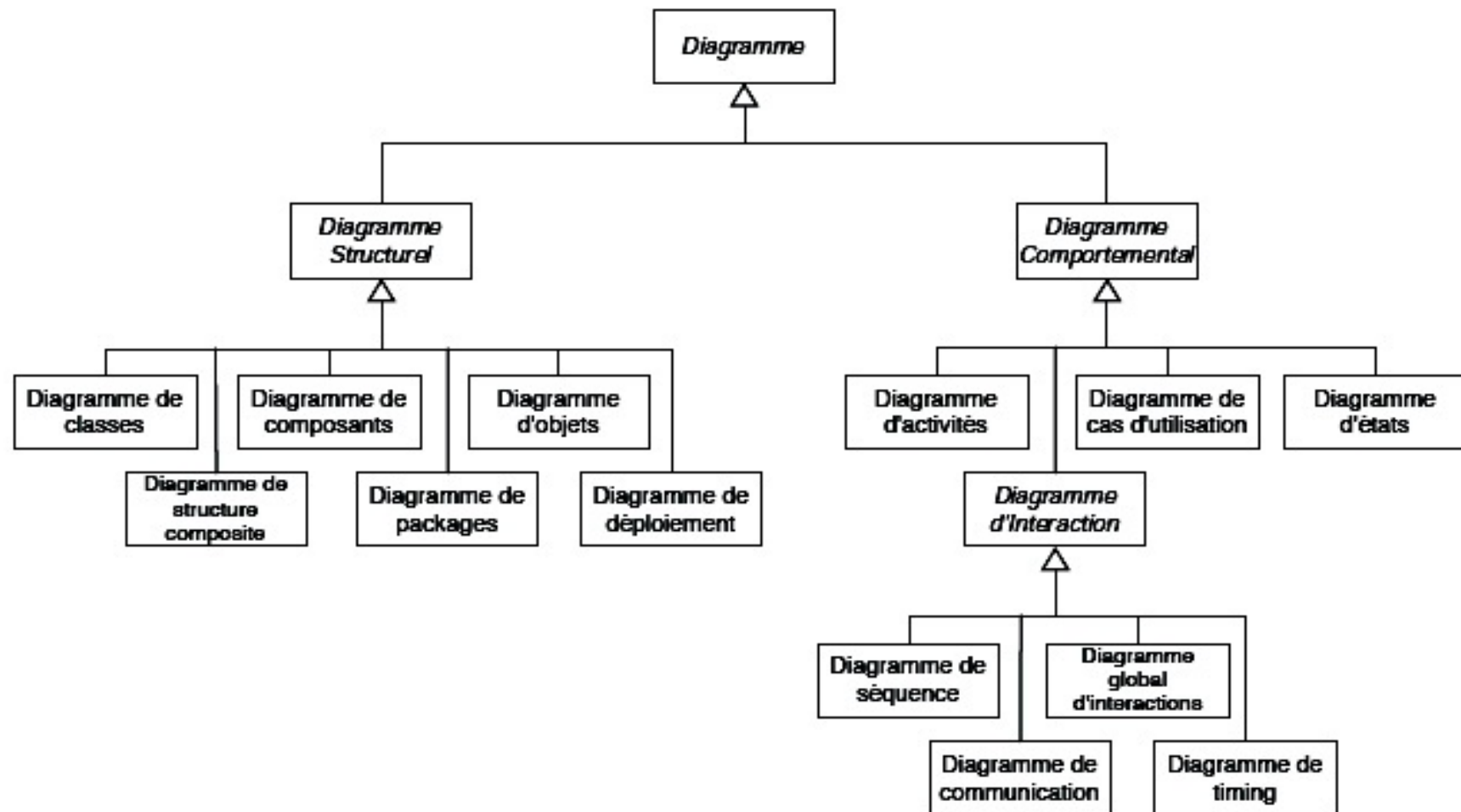
Les diagrammes UML

- Diagrammes d'états-transitions ou Machine d'états [2.0] (changements d'états d'un objet au cours de sa vie)
- Diagrammes d'activités (comportement procédural et parallèle)
- Diagrammes de composants (structure et connexion des composants)
- Diagrammes de déploiement (déploiement des artefacts en nœuds)

Les diagrammes UML 2.0

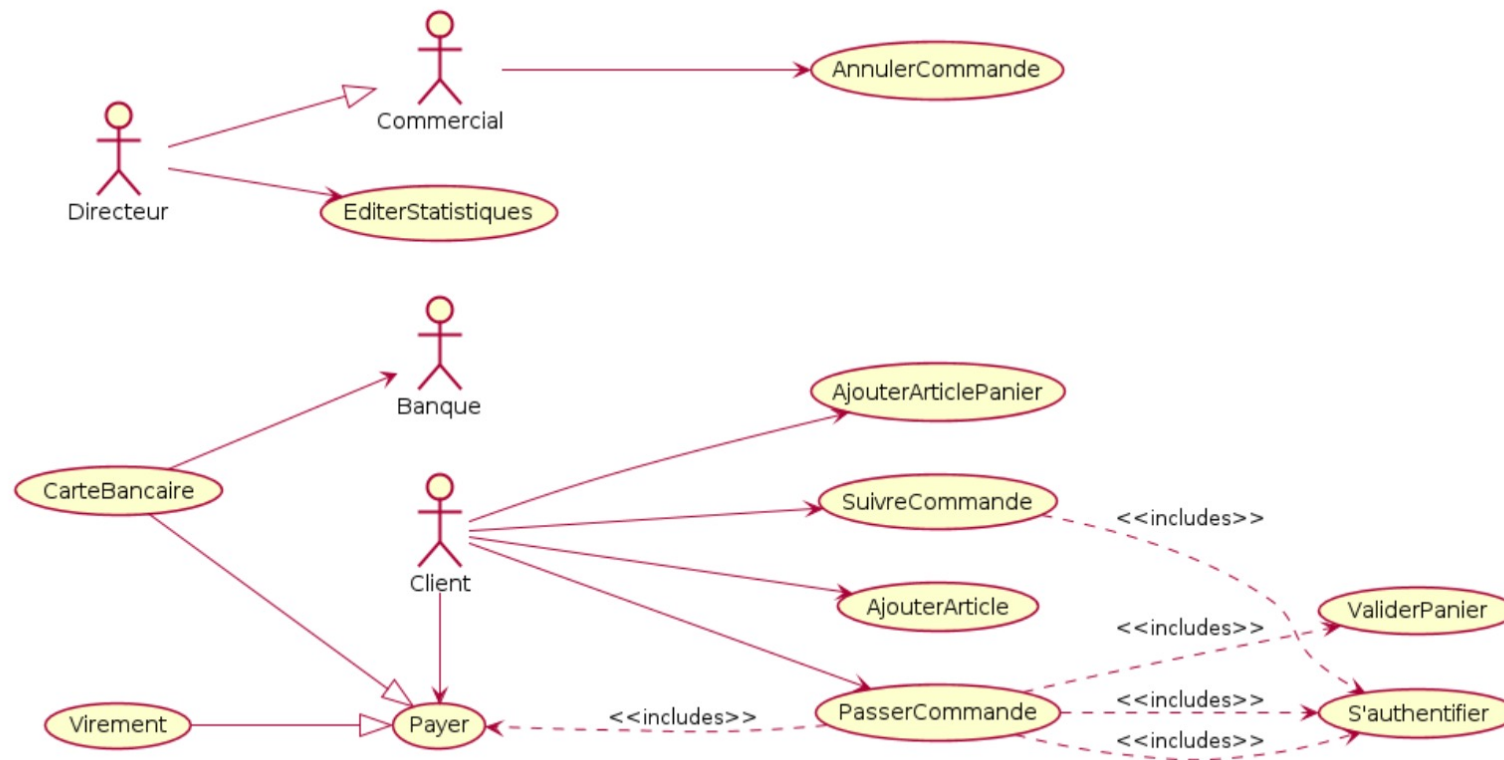
- Diagrammes de communication (diagrammes de collaboration)
- Diagrammes de vue d'ensemble des interactions (synthèse de diagrammes de séquences et d'activités)
- Diagrammes de structures composites (décomposition d'une classe au moment de l'exécution)
- Diagrammes de timing (interactions entre objets : accent mis sur le « timing »)

Les diagrammes UML 2.0



Analyse / conception des besoins

Les diagrammes des Cas d'utilisation



Les diagrammes UML : Cas d'utilisation

- Il s'agit de la solution UML pour représenter le modèle conceptuel du système
- I. Jacobson identifie les caractéristiques suivantes pour les modèles :
 - Un modèle est une simplification de la réalité.
 - Il permet de mieux comprendre le système qu'on doit développer.
 - Plus un modèle est proche de la réalité, meilleur il est.

Les diagrammes UML : Cas d'utilisation

- Les use cases diagrams, permettent de modéliser les besoins des clients d'un système
 - Ils centrent l'expression des exigences d'un système sur ses utilisateurs
 - Ils se limitent aux préoccupations "réelles" de ces utilisateurs
 - ils ne présentent pas de solutions d'implémentation

Les diagrammes UML : Cas d'utilisation

- ils ne forment pas un inventaire fonctionnel du système
- Ils permettent de structurer les besoins de ses utilisateurs et les objectifs correspondants du système
- Ils ne doivent pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins !

Les diagrammes UML : Cas d'utilisation

- Une fois identifiés et structurés, ces besoins modélisés par un diagramme des cas d'utilisation :
 - définissent le contour du système à modéliser (ils précisent le but à atteindre),
 - permettent d'identifier les fonctionnalités principales (critiques) du système.

Les diagrammes UML : Cas d'utilisation

- Ils identifient les utilisateurs du système (acteurs) et leurs interactions avec le système.
- Ils permettent de classer les acteurs et de structurer les objectifs du système.
- Ils servent de base à la traçabilité des exigences d'un système dans un processus de développement intégrant UML.

Les diagrammes UML : Cas d'utilisation

- Éléments de base des cas d'utilisation
 - Acteur : entité externe qui agit sur le système en échangeant de l'information
 - L'acteur peut consulter ou modifier l'état du système.
 - En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin.
 - L'acteur identifié est brièvement décrit
 - Les acteurs peuvent être classés, afin de faciliter la navigation dans le modèle des cas d'utilisation

Éléments de base : acteurs

- Quatre catégories d'acteurs :
 - Acteurs principaux : Utilisent les fonctions principales du système
 - Acteurs secondaires : Effectuent des tâches administratives ou de maintenance
 - Matériel externe : Dispositifs matériels incontournables faisant partie du domaine de l'application et qui doivent être utilisés
 - Autres systèmes : Systèmes avec lesquels le système considéré doit interagir

UML : Cas d'utilisation

Éléments de base

- Use case : ensemble d'actions réalisées par le système, en réponse à une action d'un acteur.
 - Un use case regroupe une famille de scénarios d'utilisation (séquences d'interactions), selon un critère fonctionnel
 - Un use case se décrit en terme d'informations échangées et d'étapes dans la manière d'utiliser le système
 - Les uses cases peuvent être structurés
 - Les uses cases peuvent être organisés en paquetages (packages).

C'est l'ensemble des use cases qui décrit les objectifs (le but) du système.

Les diagrammes UML : Cas d'utilisation Scénario

- Un scénario est une **instance** d'un use case
 - C'est une série d'événements qui se produisent durant l'exécution du système
- Un scénario peut être représenté de plusieurs manières :
 - langage naturel : une série de phrases décrivant dans l'ordre les choses qui se produisent.
 - diagrammes d'interactions d'objets
 - langage formel

Les diagrammes UML : Cas d'utilisation Scénario

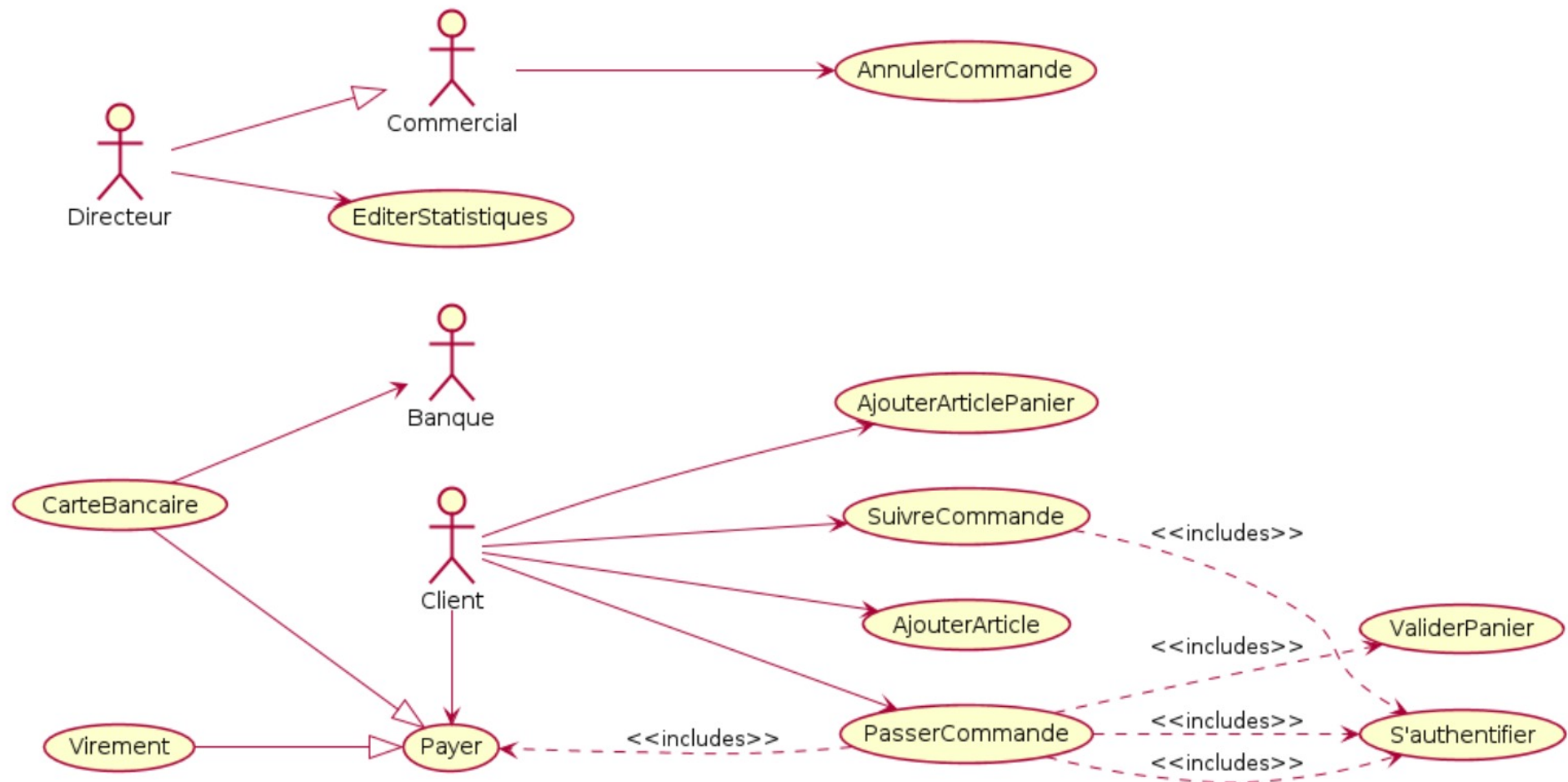
- Plusieurs scénarios à plusieurs niveaux de détails sont nécessaires pour décrire le comportement d'un système
- Chaque use case aura plusieurs scénarios
 - scénarios primaires (**sous cas d'utilisation**)
 - scénarios secondaires (**instances des précédents**)

UML : Cas d'utilisation

Relations entre cas d'utilisation

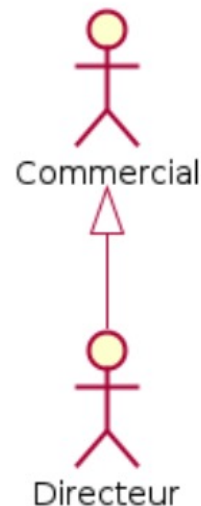
- Les diagrammes des cas d'utilisation représentent les relations entre les cas d'utilisation et les acteurs du système
- Quatre types de relations entre acteurs et cas d'utilisation
 - Communication (Acteur – UC)
 - Généralisation (Acteur-Acteur ou UC – UC)
 - Utilisation (UC – UC)
 - Extension (UC – UC)

UML : Cas d'utilisation



UML : Cas d'utilisation

- Relations entre acteurs : Généralisation
 - Un directeur est une sorte de commercial : il peut faire avec le système tout ce que peut faire un commercial, **et** d'autres choses.

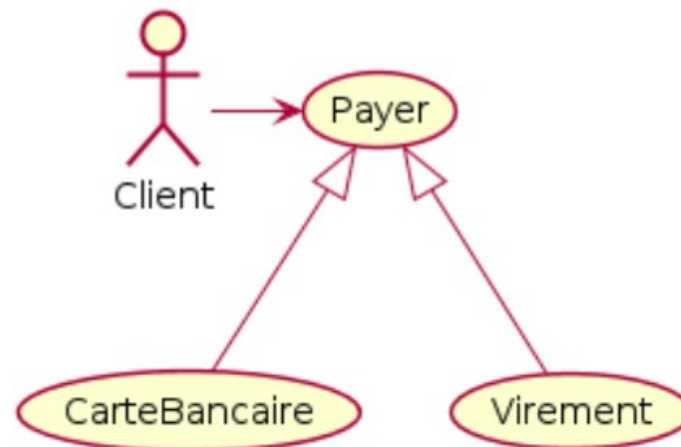


UML : Cas d'utilisation

■ Relations entre cas d'utilisation : généralisation



- le cas A est une generalisation du cas du cas B : on lit B est une sorte de A

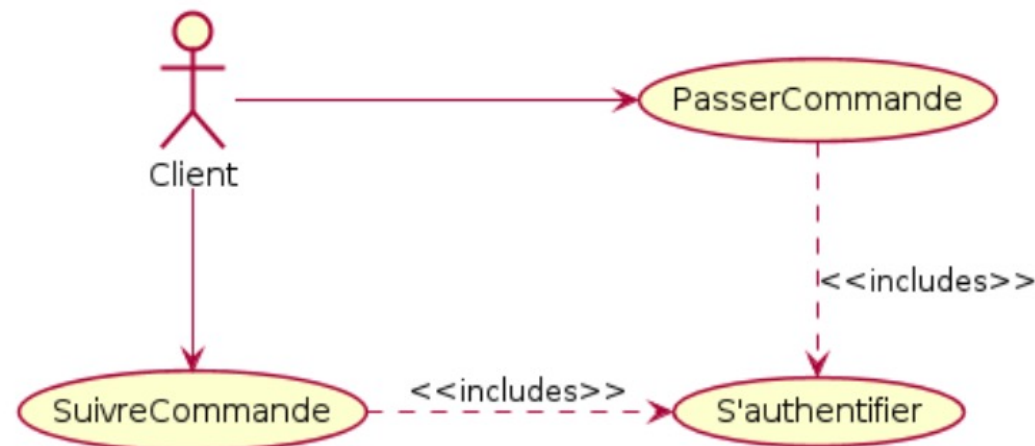


UML : Cas d'utilisation

■ Relations entre cas d'utilisation : Réutilisation

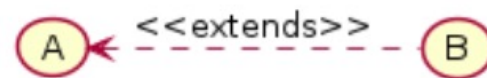


- B est une partie obligatoire de A : on lit A inclut B

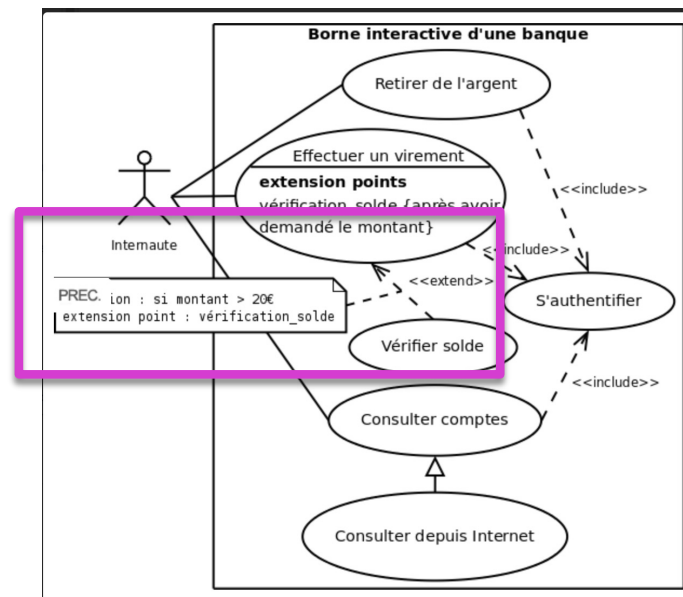


UML : Cas d'utilisation

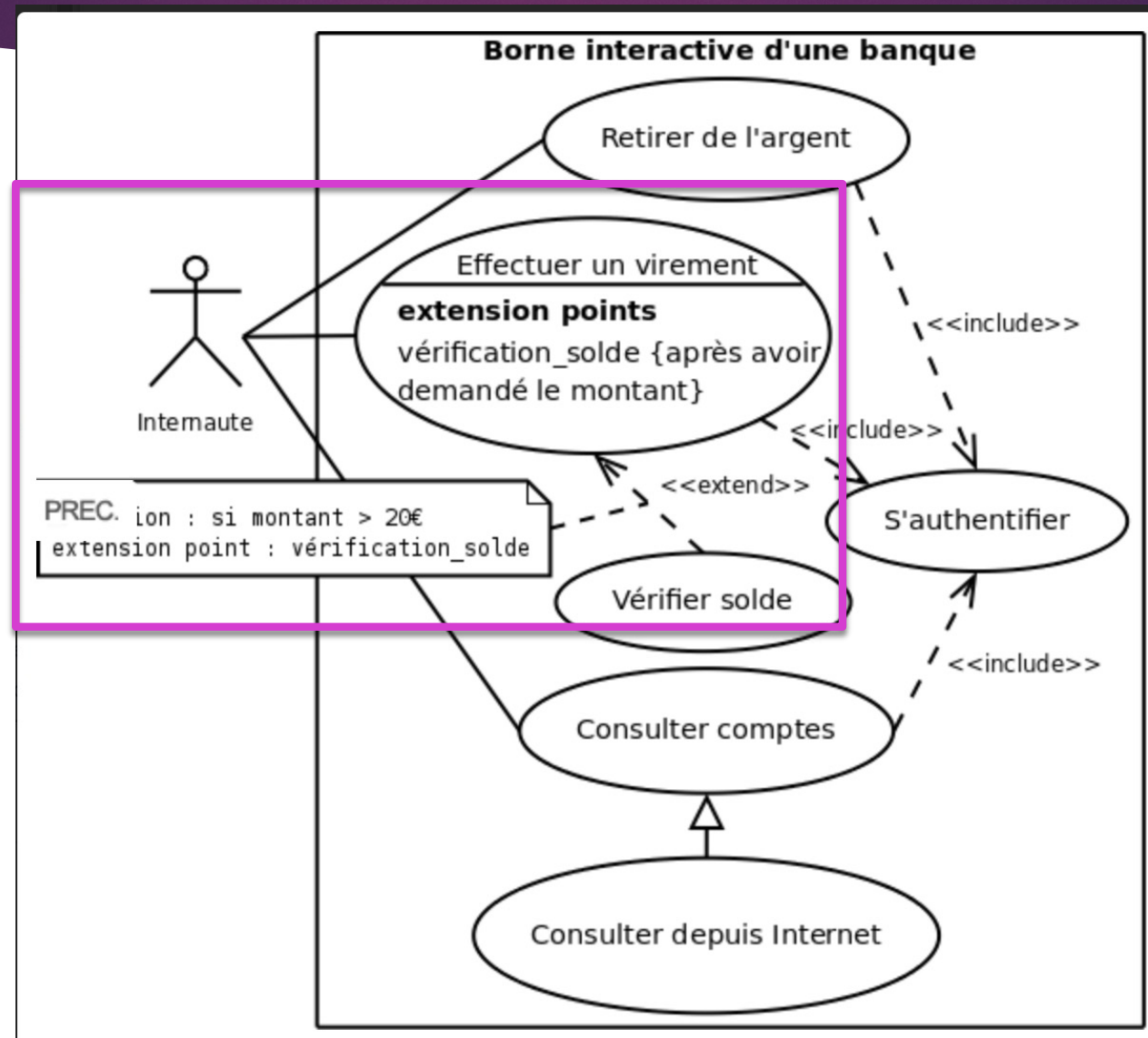
■ Relations entre cas d'utilisation : extension



- B est une partie optionnelle de A : on lit B étend A



UML : Cas d'utilisation



UML : Cas d'utilisation : Construction

- Un cas d'utilisation doit être :
 - Simple, intelligible, clair, concis
 - En général, il n'y a qu'un seul acteur par cas
- Interrogations :
 - Quelles sont les tâches de l'acteur ?
 - Quelles informations l'acteur doit-il créer, sauvegarder, modifier, détruire ou simplement lire ?
 - L'acteur devra-t-il informer le système des changements externes ?
 - Le système devra-t-il informer l'acteur des changements internes ?

UML : Cas d'utilisation

Règles de mise en oeuvre

- Début du cas d'utilisation :
 - événement qui déclenche le cas d'utilisation

- Fin du cas d'utilisation :
 - événement qui cause l'arrêt du cas d'utilisation

- Interaction entre le cas d'utilisation et les acteurs
 - décrit clairement ce qui est dans et ce qui est hors du système

UML : Cas d'utilisation

Un exemple : *Jeu du monopoly* :

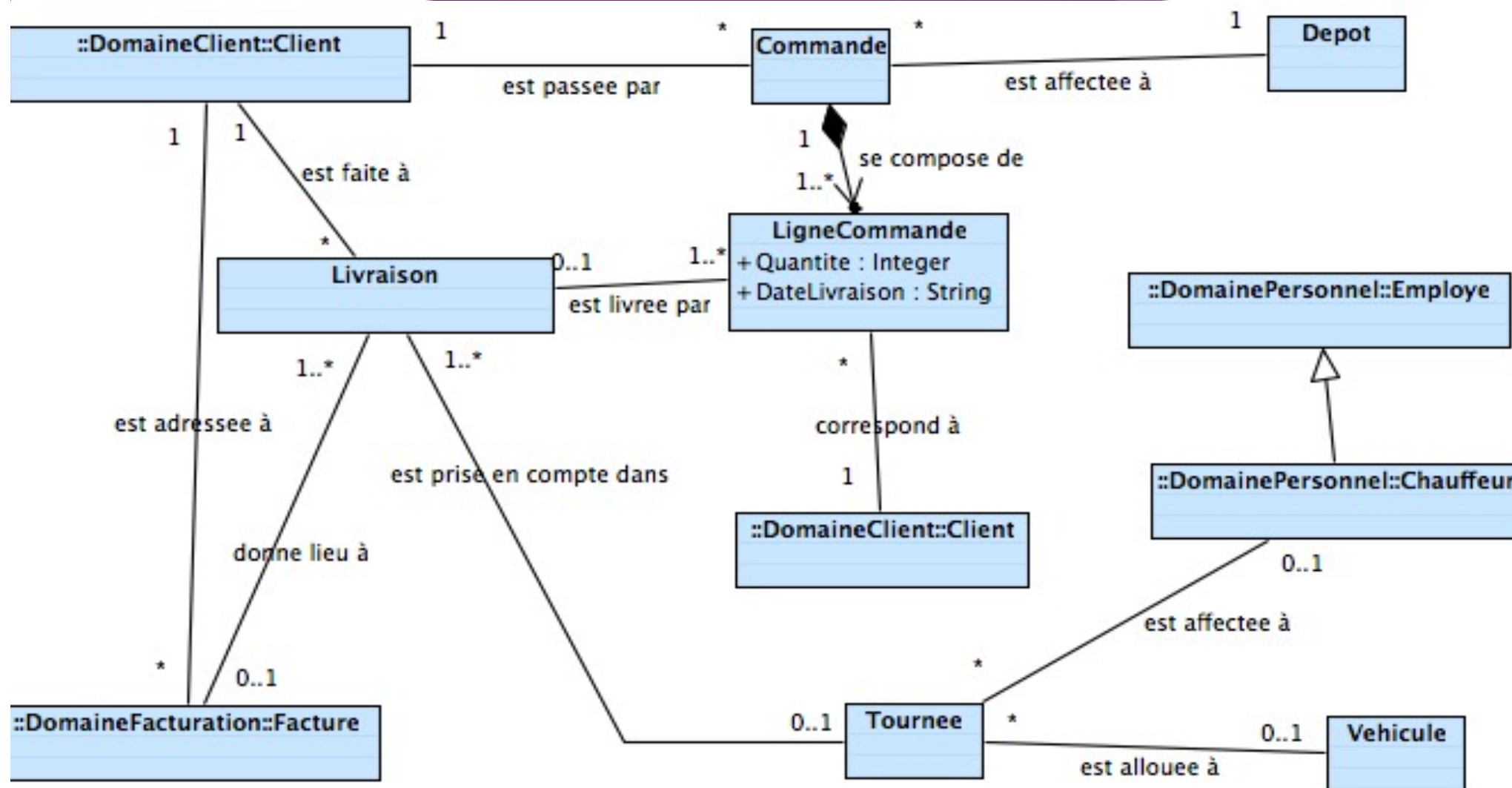
- Quels acteurs ?
- Quels cas d'utilisation ?
- Quels Scénarios ?



Analyse / Conception des entités

- *Les diagrammes de classes (représentation statique)*
- *Les diagrammes d'objets (représentation dynamique)*
- *Les machines à états finis (représentation dynamique)*

UML : Diagramme de classes



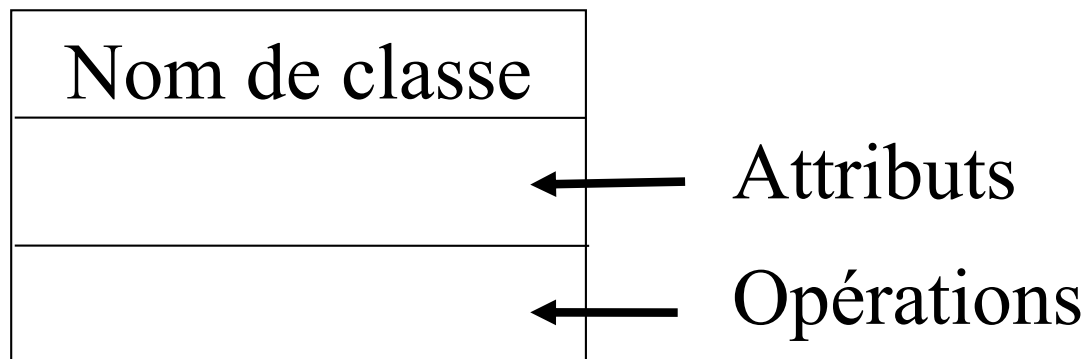
Les diagrammes UML : Diagrammes de classes

- Le diagramme de classe
 - définit le modèle conceptuel des entités du système
 - permet d'identifier les entités et leurs relations
 - donne l'architecture statique du système

Les diagrammes UML : Diagrammes de classes

■ *Les classes*

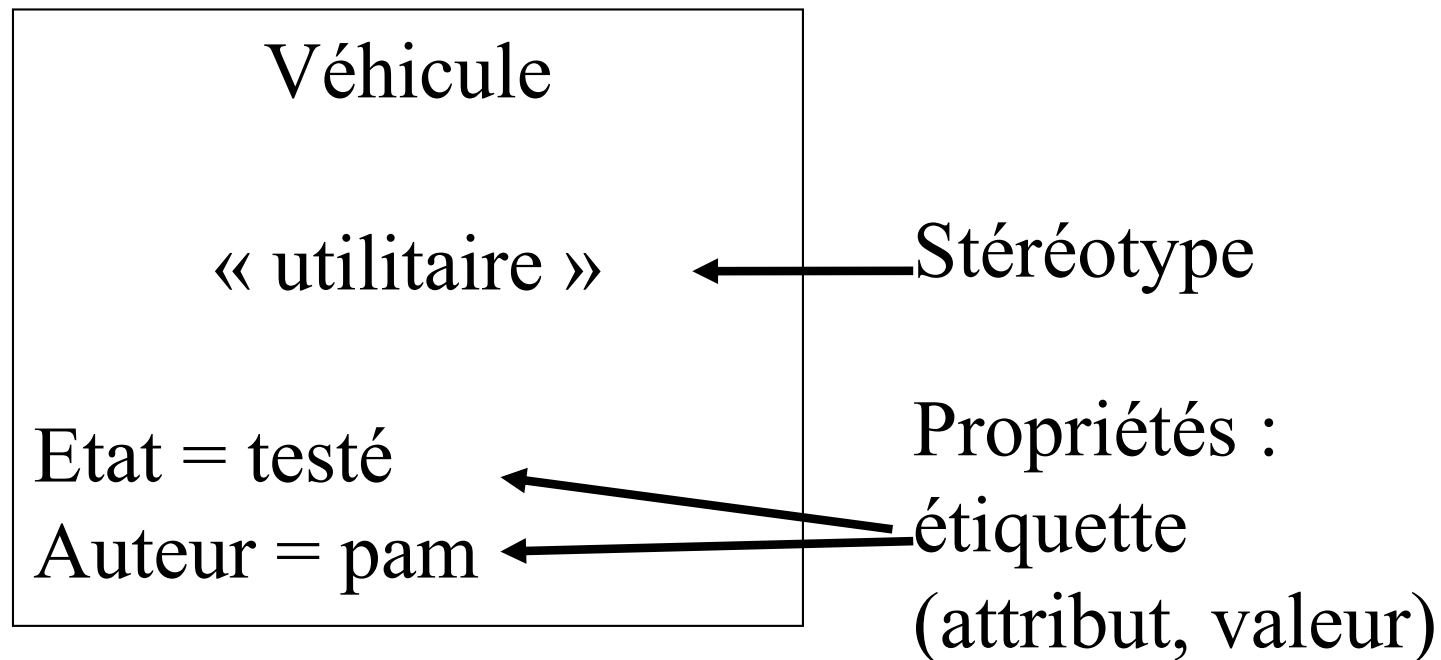
□ *Représentation graphique*



UML : Diagrammes de classes

Les classes

□ Stéréotypes et propriétés



UML : Diagrammes de classes

Les classes

□ Attributs et opérations

Nom de classe

Nom-Attribut : Type-Attribut = Valeur-initiale

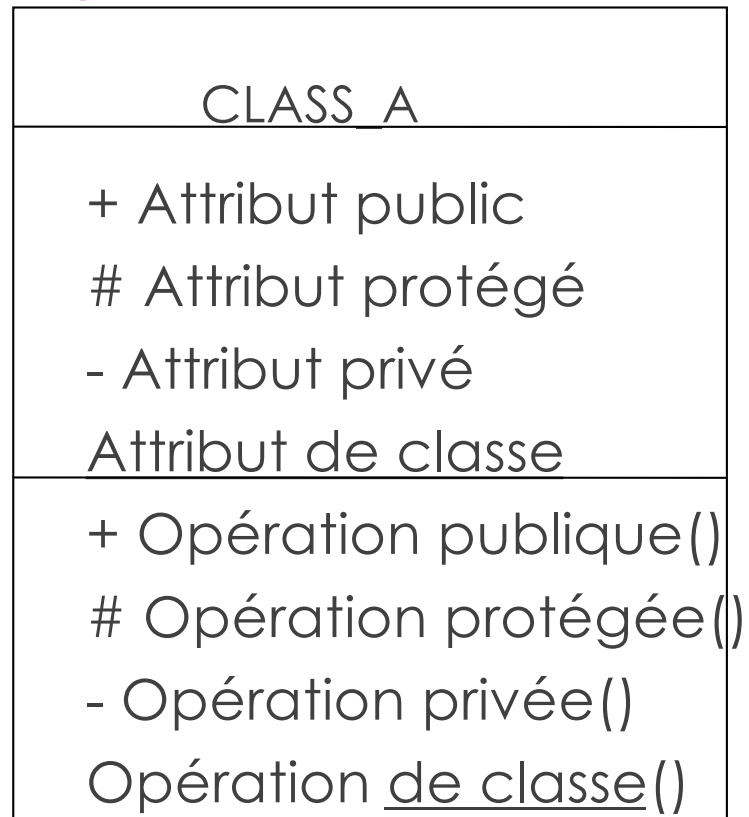
/Attribut-dérivé

**Nom_Opération (Nom-argument : type-argument =
valeur-par-défaut,...) : Type-retourné**

UML : Diagrammes de classes

Les classes

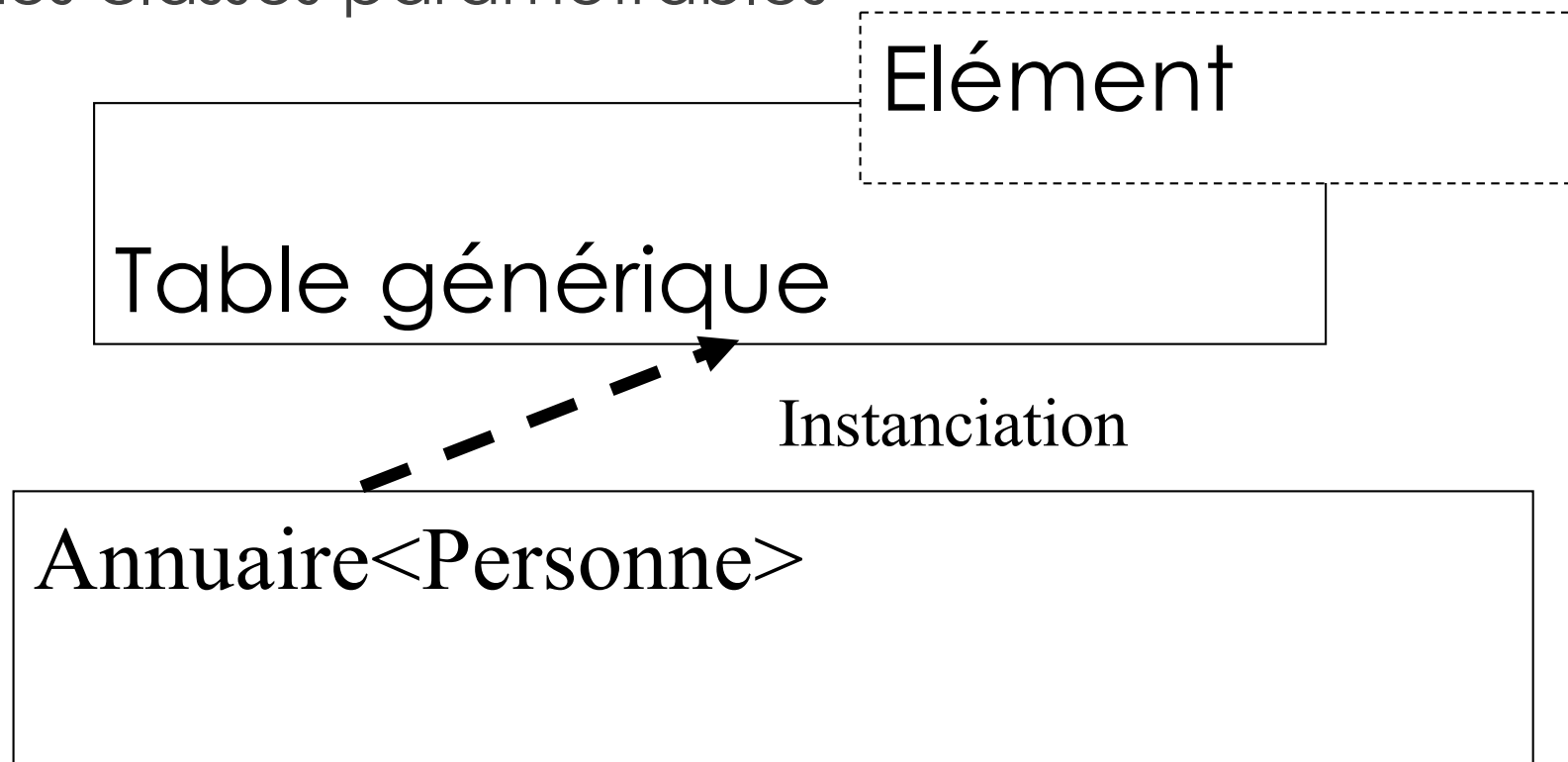
□ Visibilité



UML : Diagrammes de classes

Les classes

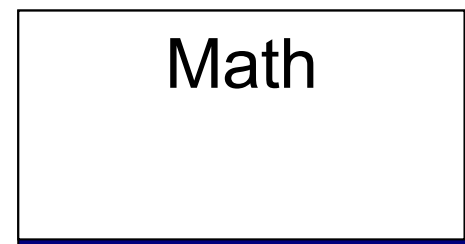
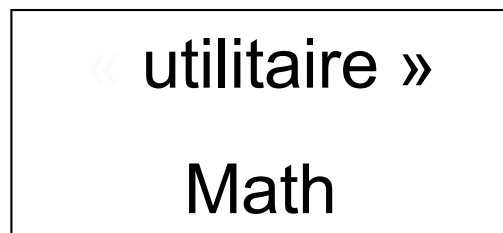
- Les classes paramétrables



UML : Diagrammes de classes

Les classes

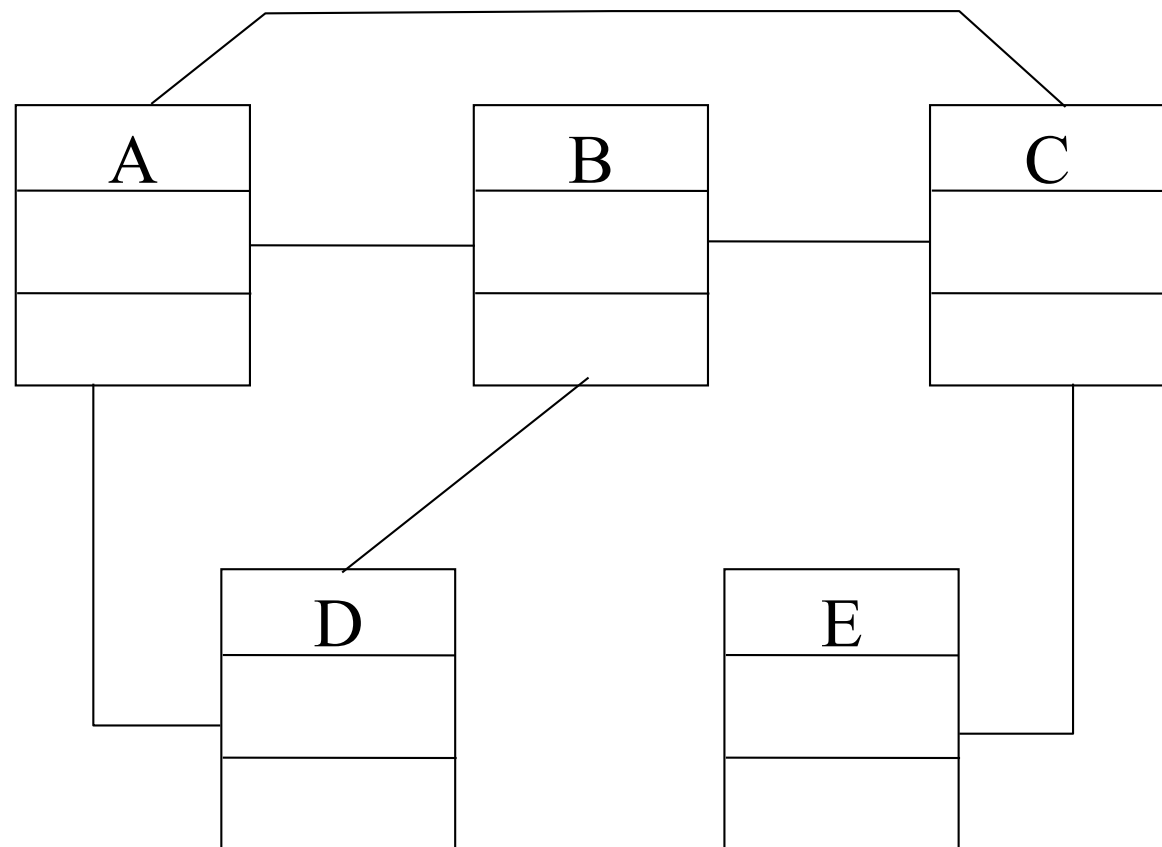
- *Les classes utilitaires*
 - *non instanciables*
 - *non abstraites*
 - *non types de données*
 - *spécifications pures*
 - *stéréotype : « utilitaire »*



UML : Diagrammes de classes

Les associations

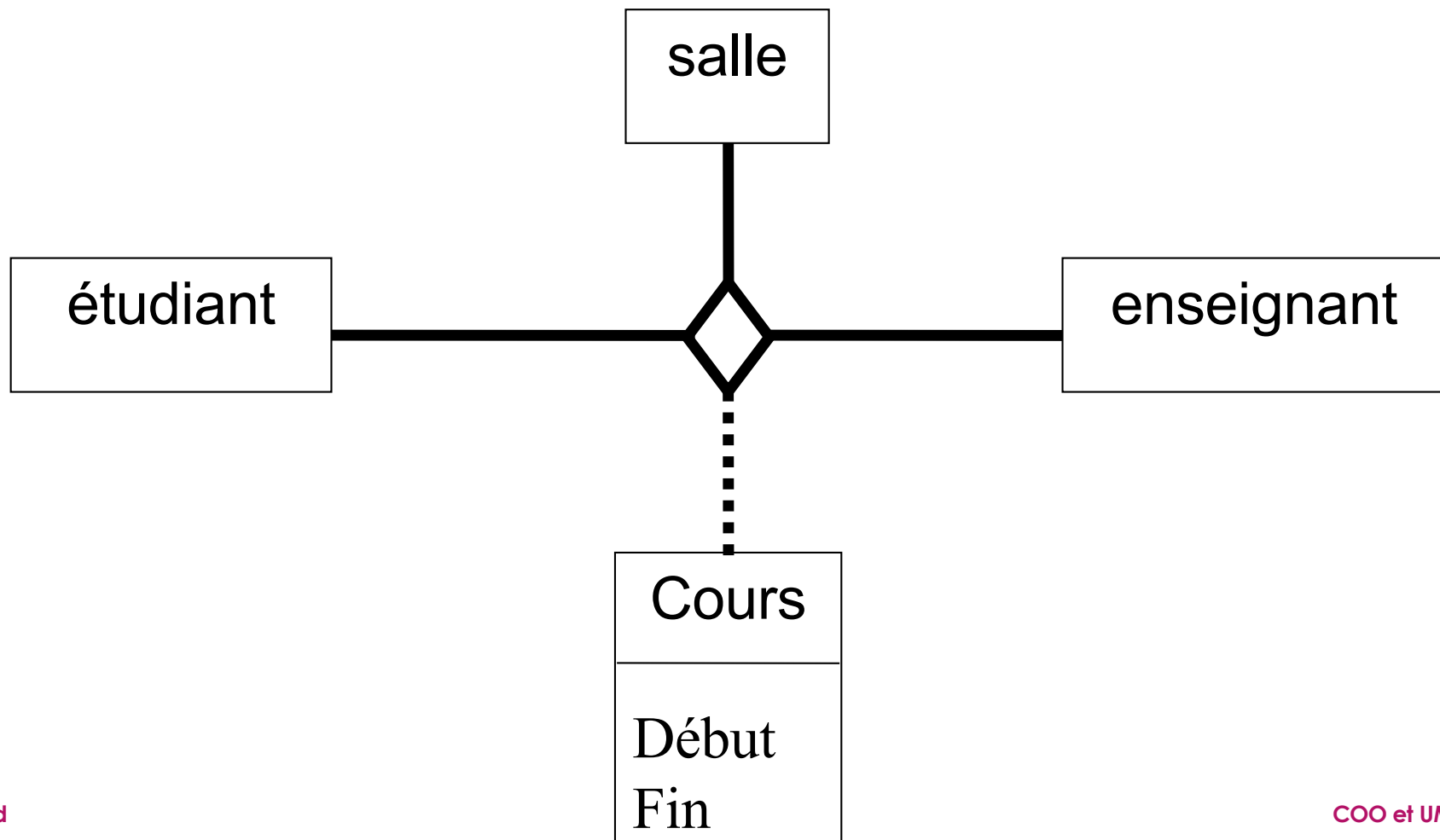
■ Représentation



UML : Diagrammes de classes

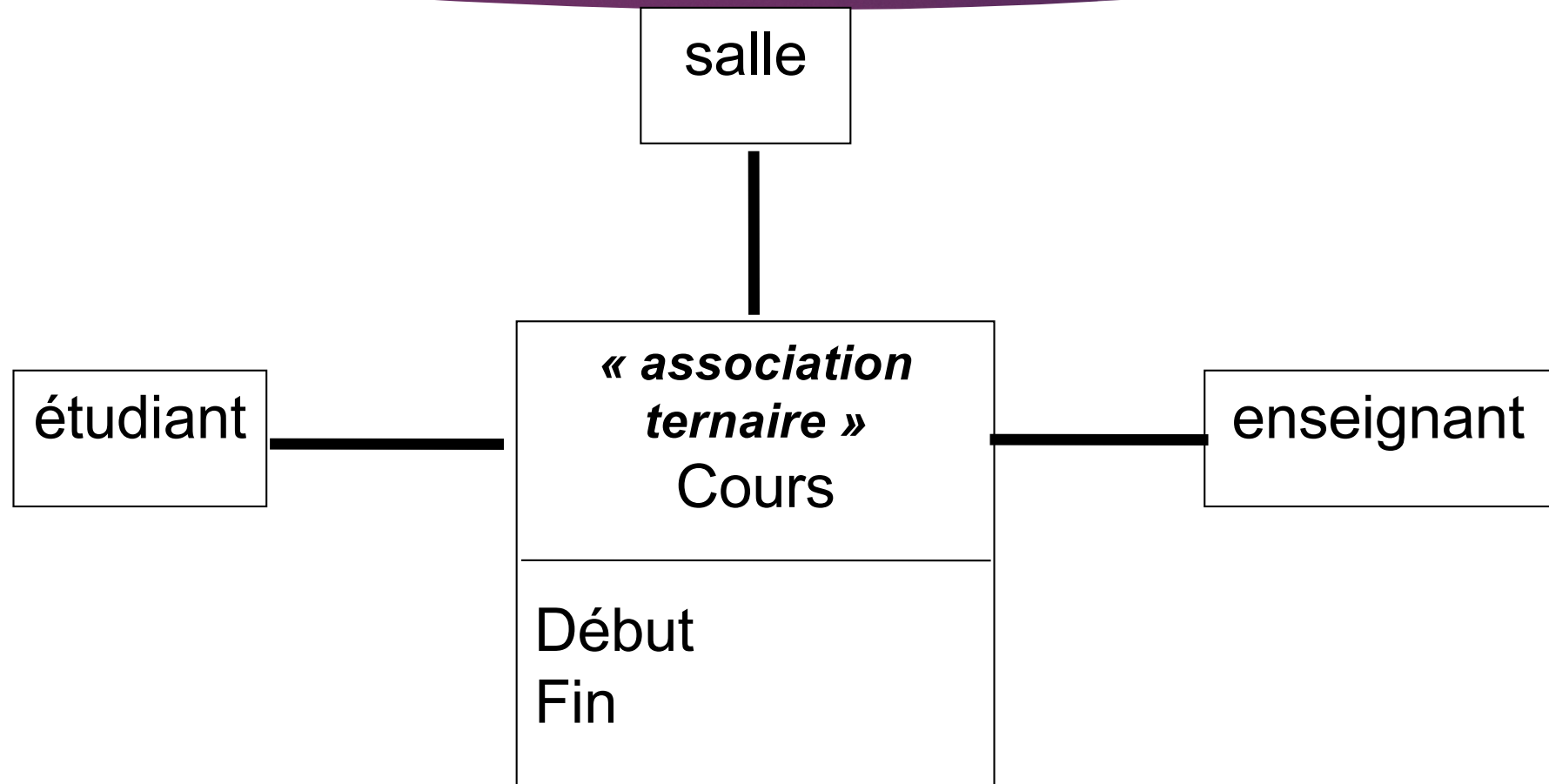
Les associations

- En général binaires, parfois n-aires :



UML : Diagrammes de classes

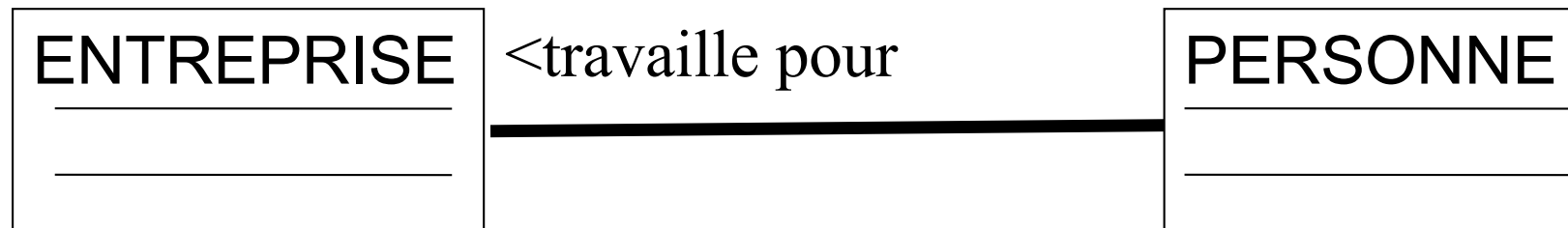
Les associations n-aires



UML : Diagrammes de classes

Les associations

- Nommage des associations
 - En général par une forme verbale
 - Dirigé



UML : Diagrammes de classes

Les associations

- Nommage des rôles des associations
 - Rôle = Extrémité d'une association
 - Utilisation d'une forme nominale
 - En général exclusif du nommage de l'association (à privilégier)

UML : Diagrammes de classes :

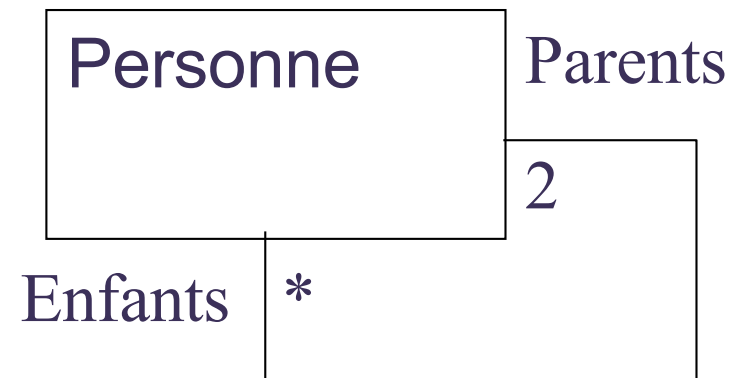
Nommage des rôles des associations

Justification :

- Plusieurs associations



- Associations réflexives



UML : Diagrammes de classes

Les associations

- Multiplicité des associations
 - Information portée par le rôle
 - Indique le nombre d'objets en relation de part et d'autre de l'association

1	Un et un seul
0..1	Zéro ou un
M..N	de M à N (entiers naturels)
*	de Zéro à plusieurs
0..*	de Zéro à plusieurs
1..*	de 1 à plusieurs

UML : Diagrammes de classes

Les associations

- Multiplicité des associations
 - conservées sauf : création, destruction
 - valeur de multiplicité $> 1 \Rightarrow$ Collection
 - valeur 0 \Rightarrow tests de la présence de liens

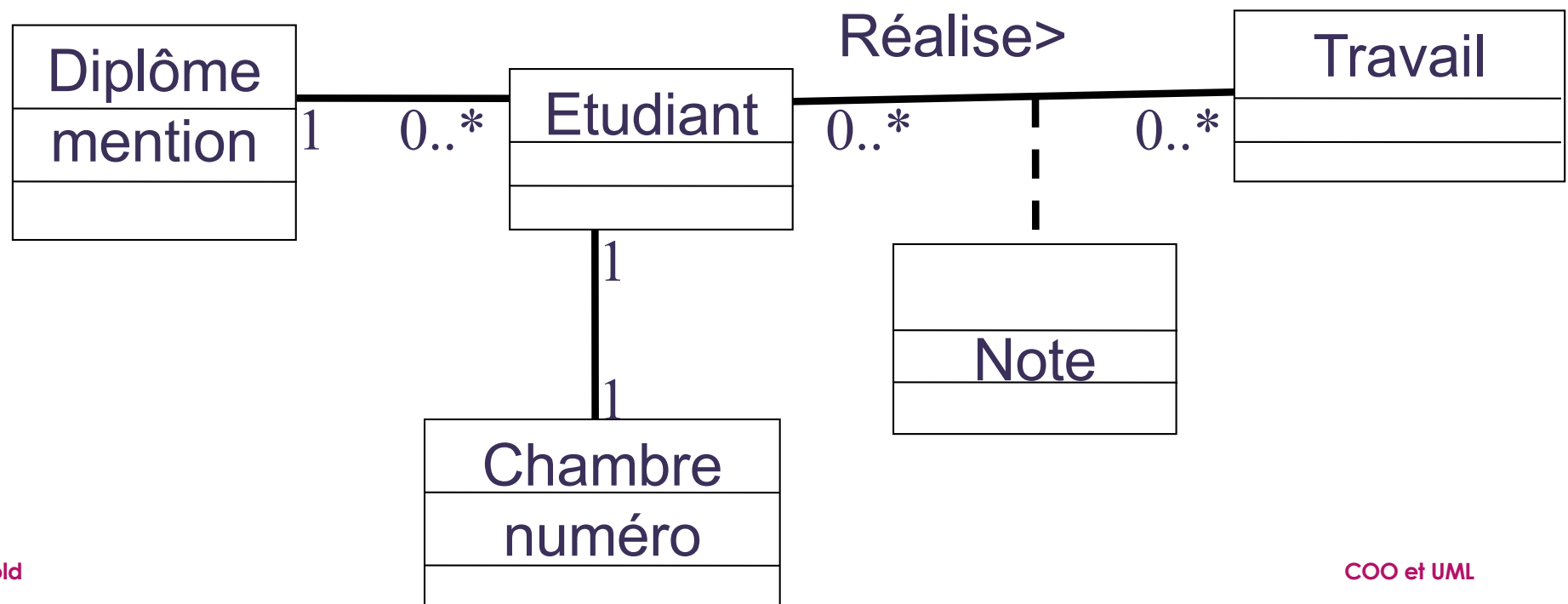


UML : Diagrammes de classes

Les associations

► Attributs des associations

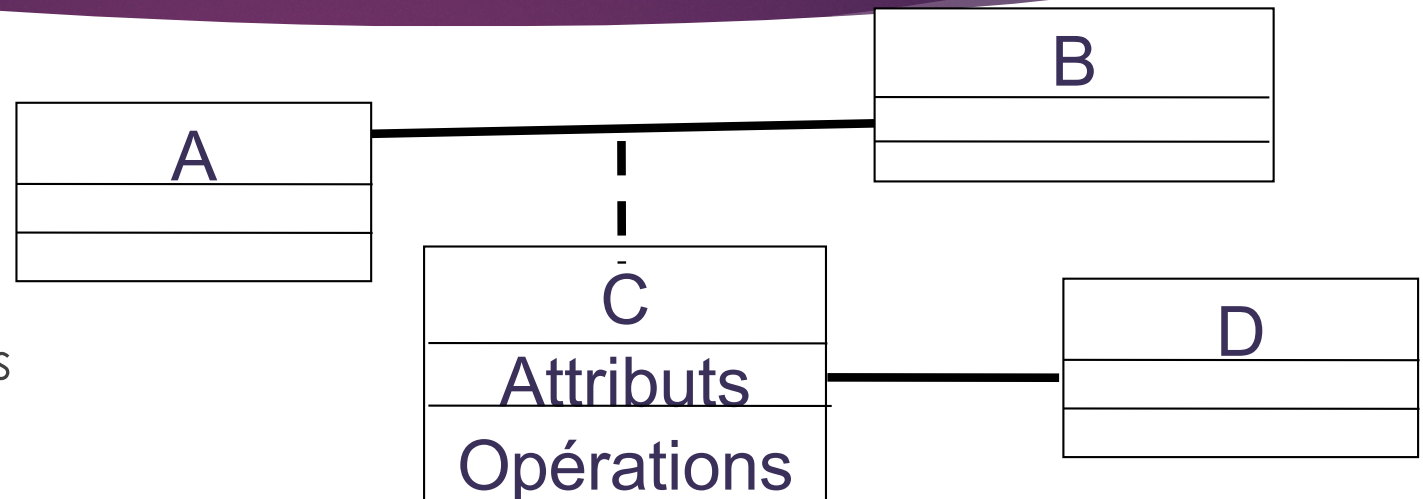
- Difficile de les attribuer à une classe dans les associations N-N



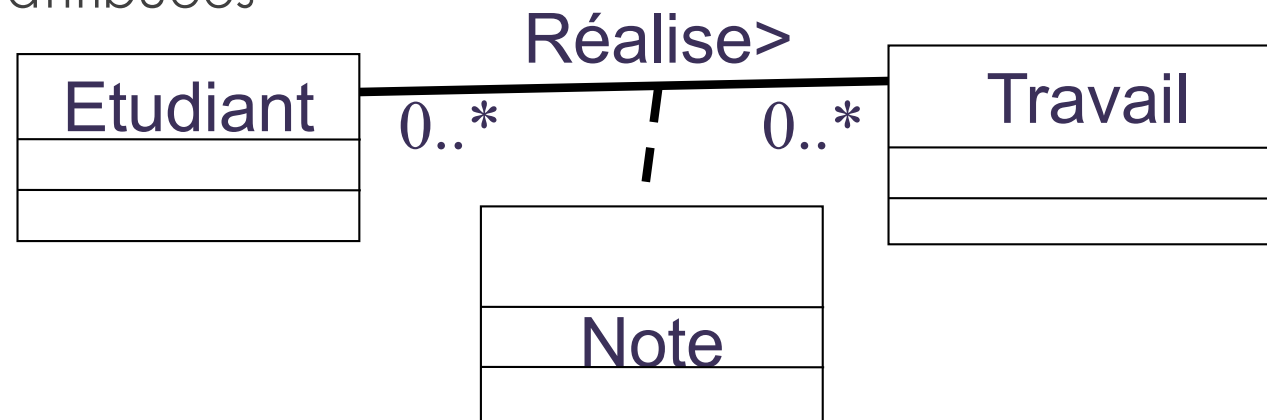
UML : Diagrammes de classes

Les associations

- Classes-associations



- Associations attribuées



UML : Diagrammes de classes

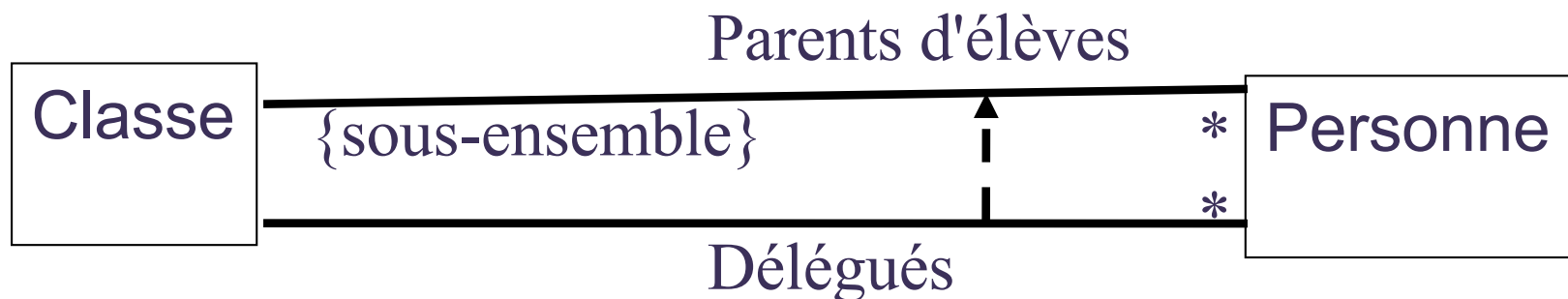
Les associations

■ Contraintes sur les associations

□ Sur un rôle



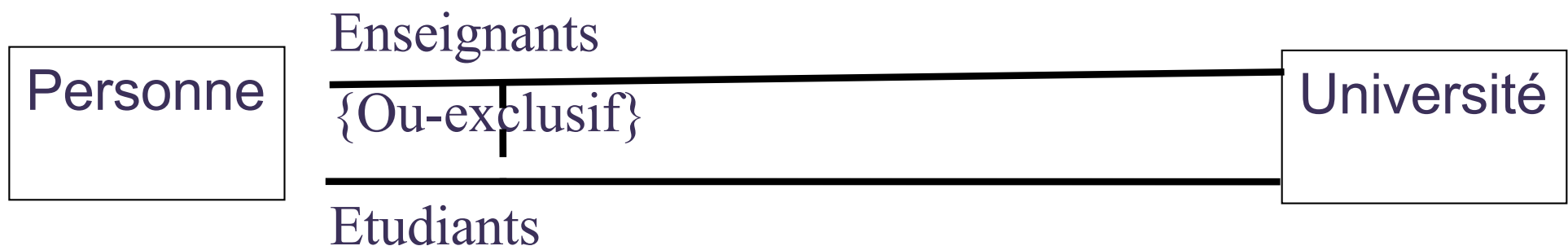
□ Sur les objets participant à une relation



UML : Diagrammes de classes

Les associations

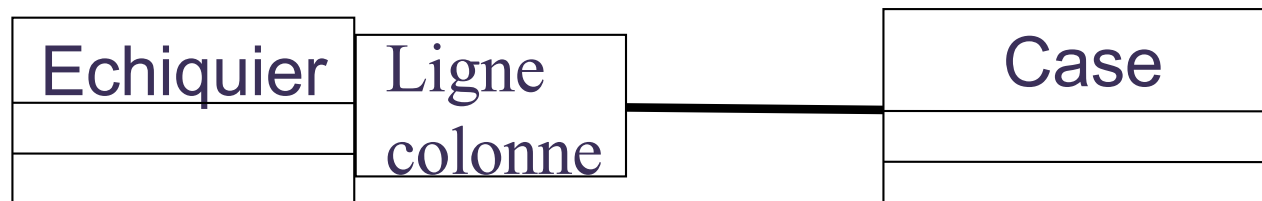
- Contraintes sur les associations
 - Sur un ensemble de relations



UML : Diagrammes de classes

Les associations

- Association avec restriction
 - Utilisation d'une clé qui réduit la multiplicité
 - La paire (instance d'une classe de l'association, valeur de clé) identifie un sous ensemble des instances de la classe associée.



UML : Diagrammes de classes

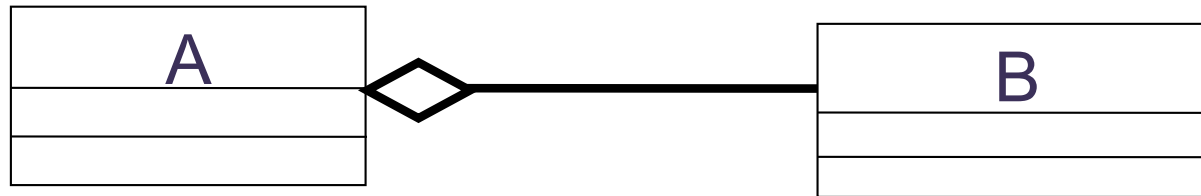
Les agrégations

- Association non symétrique
 - une extrémité joue un rôle prépondérant
 - concerne un seul rôle, quelle que soit l'arité
 - critères :
 - Action sur une classe implique action sur une autre classe.
 - Objets d'une classe subordonnés aux objets d'une autre classe
 - Propagation des valeurs d'attributs
 - Classe faisant partie d'une autre classe

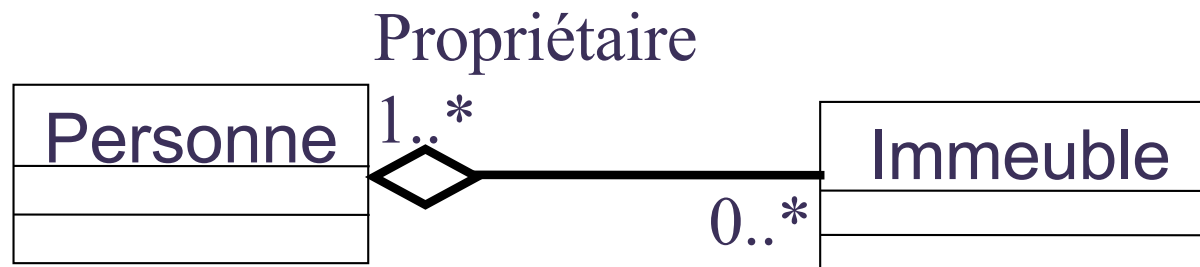
UML : Diagrammes de classes

Les agrégations

■ Représentation



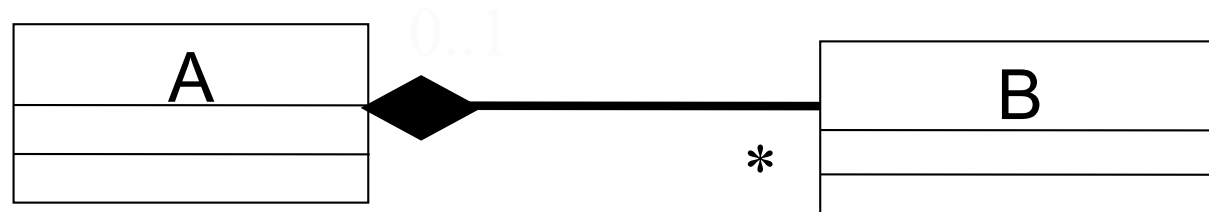
■ Agrégation multiple



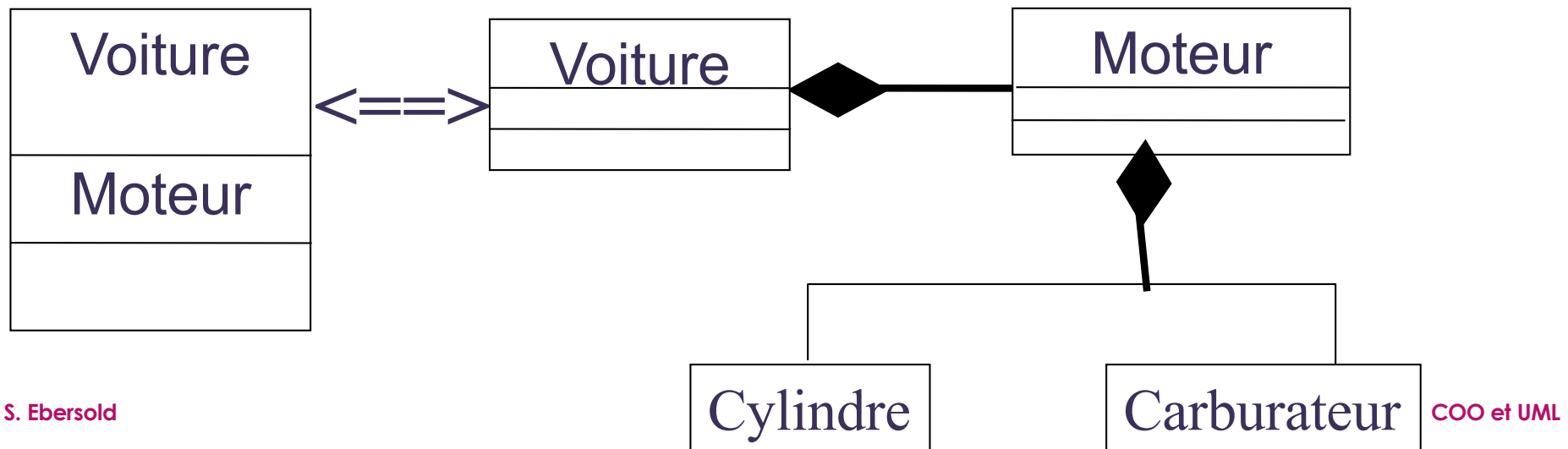
UML : Diagrammes de classes

La composition

■ Agrégation particulière



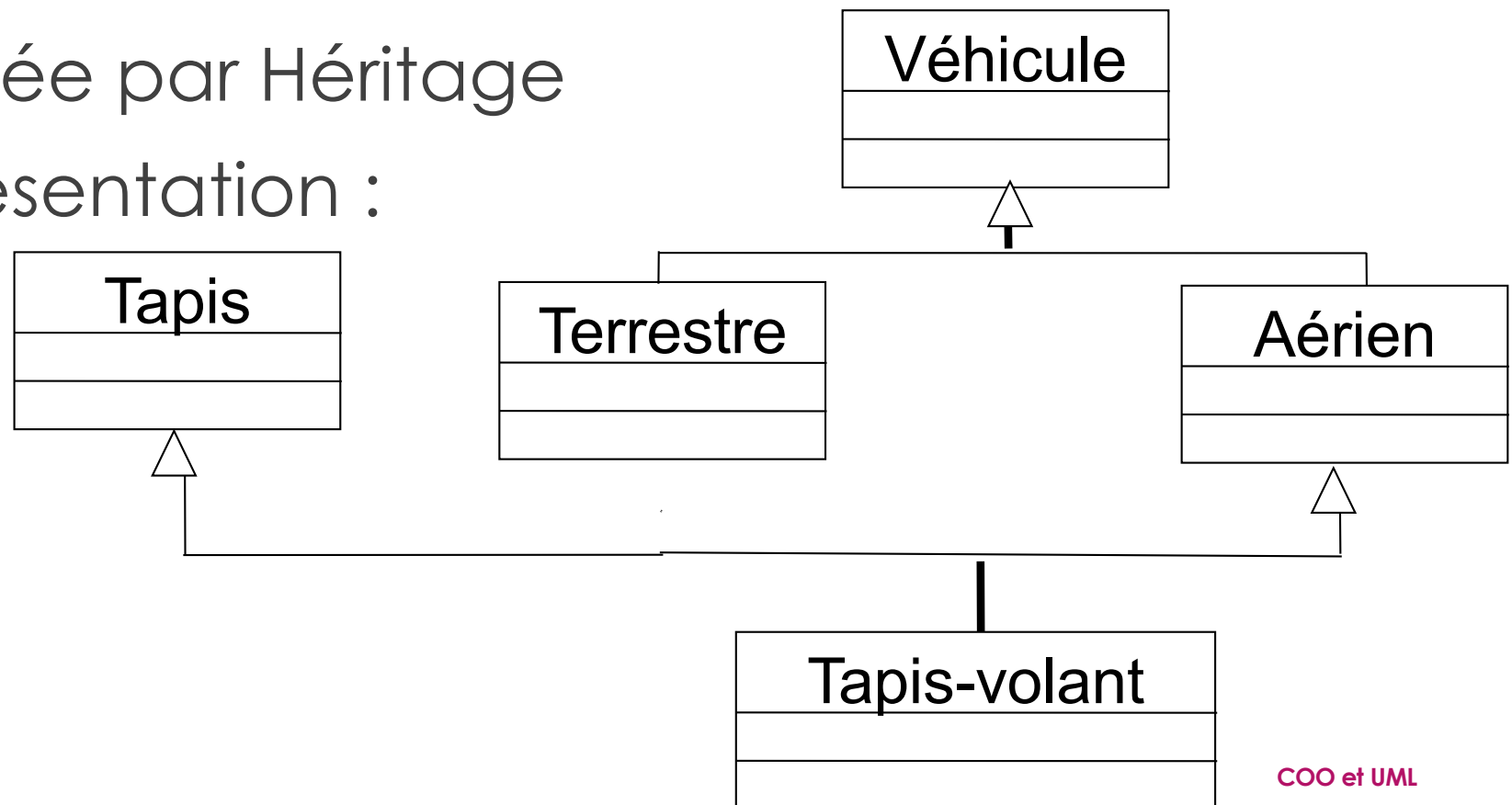
■ Sémantiquement équivalente à un attribut



UML : Diagrammes de classes

La généralisation (classification)

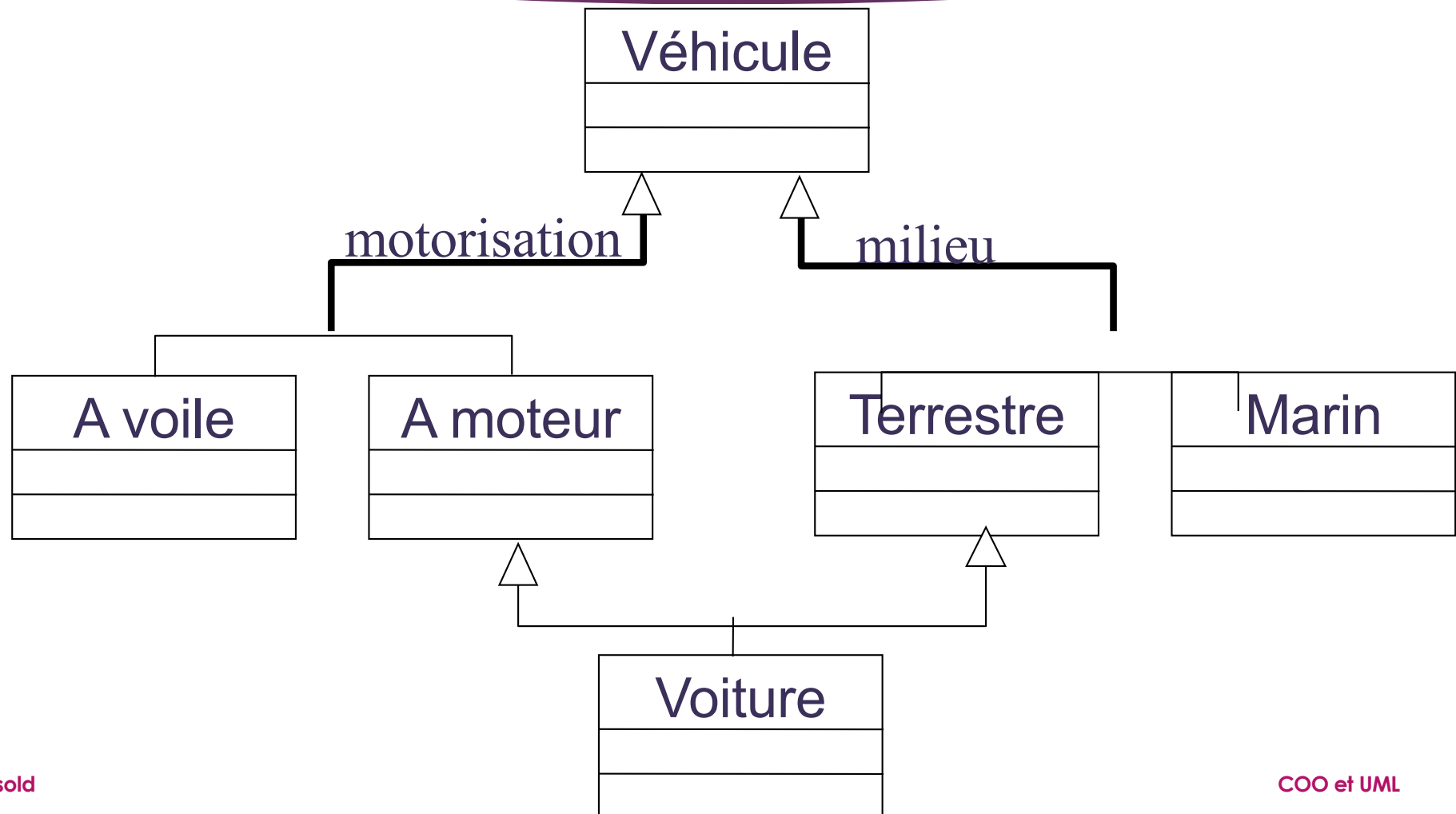
- Simple ou multiple
- Réalisée par Héritage
- Représentation :



UML : Diagrammes de classes

La généralisation

■ Critères

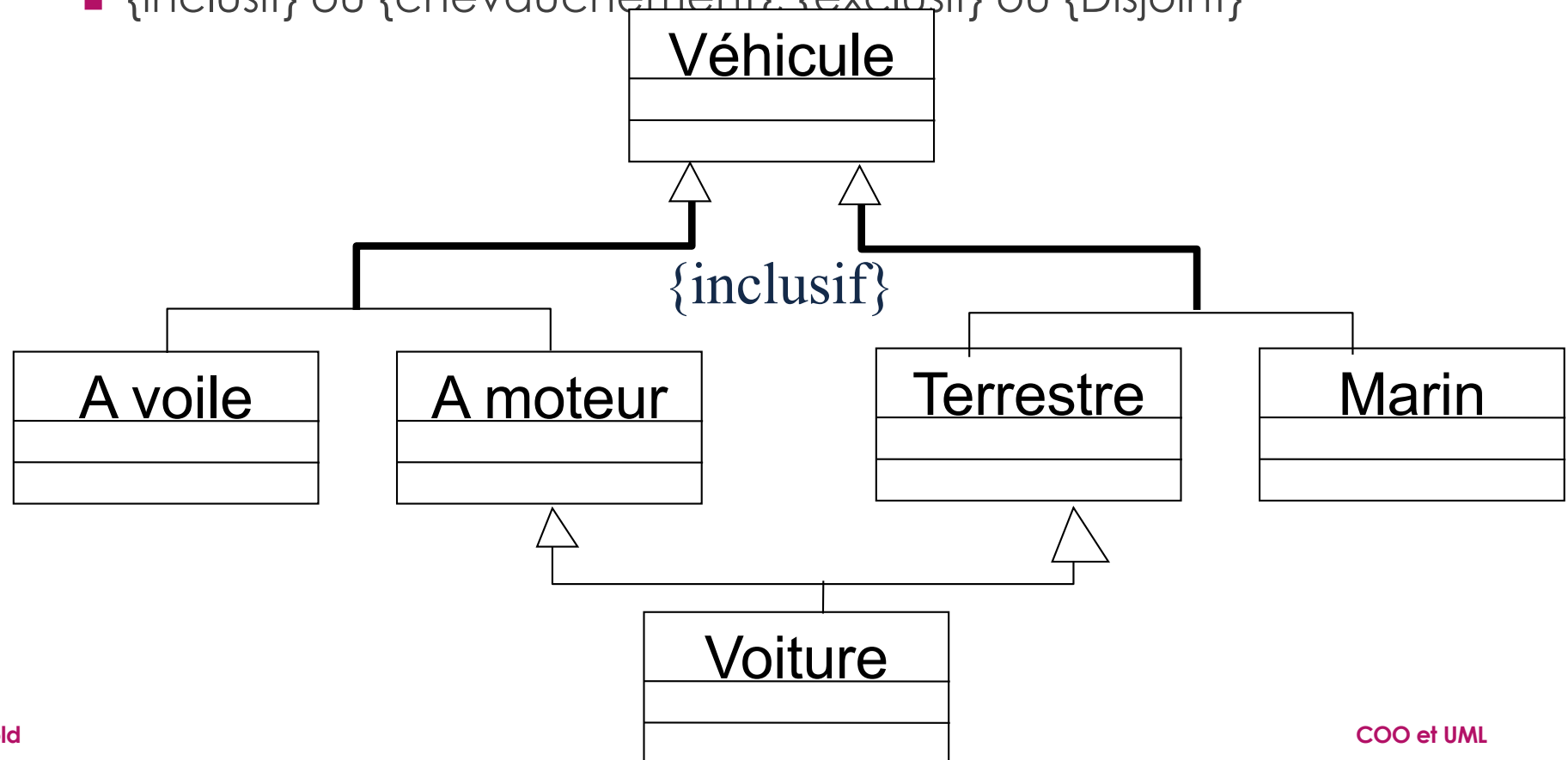


UML : Diagrammes de classes

La généralisation

- Contraintes

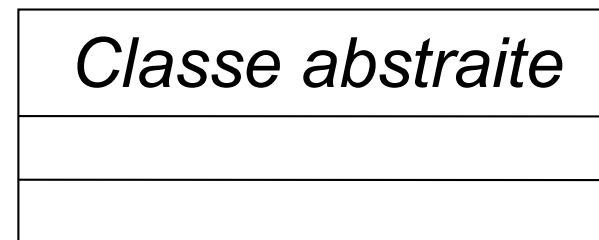
- {inclusif} ou {chevauchement}, {exclusif} ou {Disjoint}



UML : Diagrammes de classes

Classes abstraites

- Classes non instanciables
- Spécifications générale concrétisées dans les sous-classes
- Désignée au moyen d'une propriété booléenne *Abstraite* définie pour tous les éléments généralisables (types, paquetages, stéréotypes) et les opérations
- Représentation



UML : Diagramme de classes : Monopoly

