

## Modélisation Objet

### Le langage UML



## Organisation et plan du cours

CM-01

I - Introduction : l'histoire, la modélisation  
II - Diagrammes de classes et d'objets – concepts de base

CM-02

III - Diagrammes de classes – concepts avancés

CM-03

IV - Diagramme de cas d'utilisation

CM-04

V - Diagramme de séquences

CM-05

VI - Diagramme d'états-transitions  
VII - Diagramme d'activités

CM-06

VIII - Diagramme de composants  
IX - Diagramme de déploiement  
X - Diagramme de paquetages

## I

### Introduction

## UML C'est quoi exactement ?

- ◆ Comme disait ma grand-mère ... un bon dessin vaut mieux qu'un long discours !
- ◆ Quel sont les premiers outils que l'on utilise pour construire une maison, un immeuble, une ville :
  - La pelle et le seau ? Le crayon et le papier ?
- ◆ Quand un système est complexe, il est nécessaire de pouvoir en faire des représentations simples, axées sur un point de vue particulier.
- ◆ Exemples : le plan électrique d'une habitation, une carte routière, ...

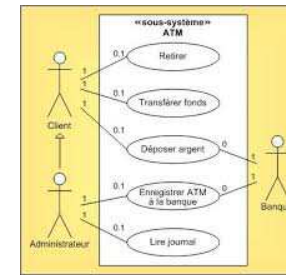
## UML C'est quoi exactement ?

- ◆ En informatique aussi, les systèmes peuvent être complexes !
- ◆ Il devient alors nécessaire, quelque soit la démarche choisie, de faire des représentations – pour un point de vue particulier – du système que l'on construit.
- ◆ On parle alors de modèles et de modélisation.

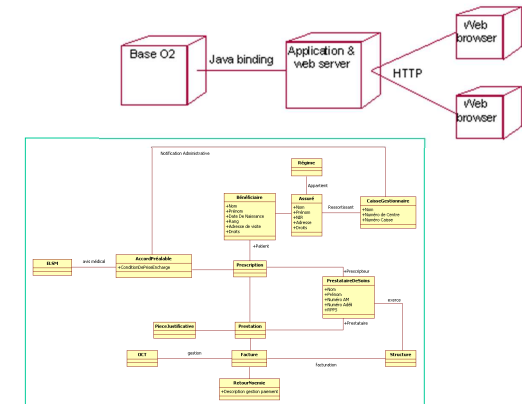
## UML C'est quoi exactement ?

- ◆ Exemples de modélisations :

*Point de vue du client : quelles sont les différentes fonctions de mon système ? Pour qui est-il fait ?*



*Point de vue de l'informaticien : quels sont les différents composants mis en œuvre ? Comment communiquent ils ? Quelles sont les informations manipulées ?*



## UML C'est quoi exactement ?

- ◆ Pour partager un modèle avec d'autres personnes, il faut s'assurer au préalable :
  - que les personnes comprennent les conventions de représentation et les notations utilisées,
  - qu'il ne pourra pas y avoir d'ambiguïté sur l'interprétation du modèle.
- ◆ D'où l'émergence de langages de modélisation de systèmes informatiques.
- ◆ Un langage de modélisation doit définir sans ambiguïté les concepts utilisables, leur représentation, les règles et contraintes associées.

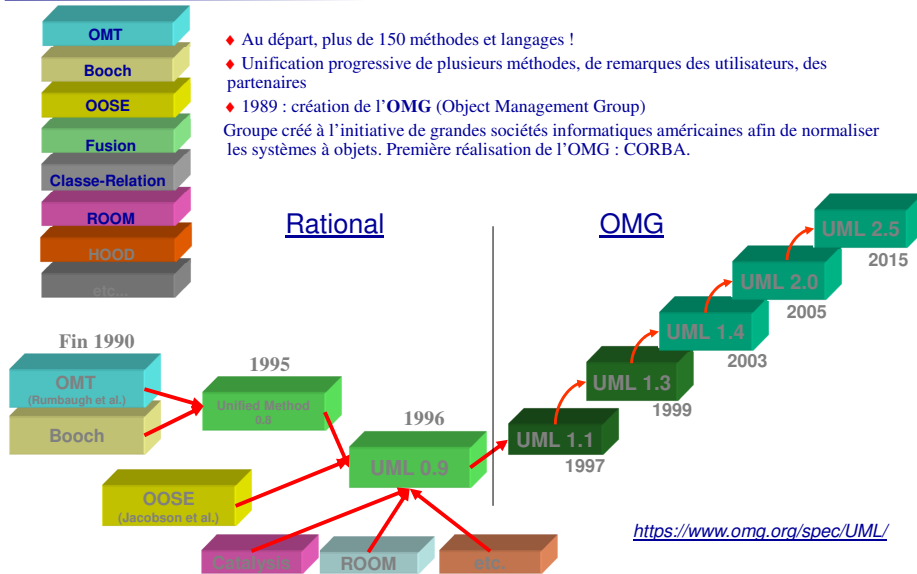
## UML C'est quoi exactement ?

### UML = Unified Modeling Language Langage unifié pour la modélisation

- ◆ Langage de modélisation objet, indépendant de la méthode utilisée.
- ◆ UML est un langage pour :
  - Comprendre et décrire un problème,
  - Spécifier un système,
  - Concevoir et construire des solutions,
  - Documenter un système,
  - Communiquer.



## UML C'est quoi exactement ? – La genèse

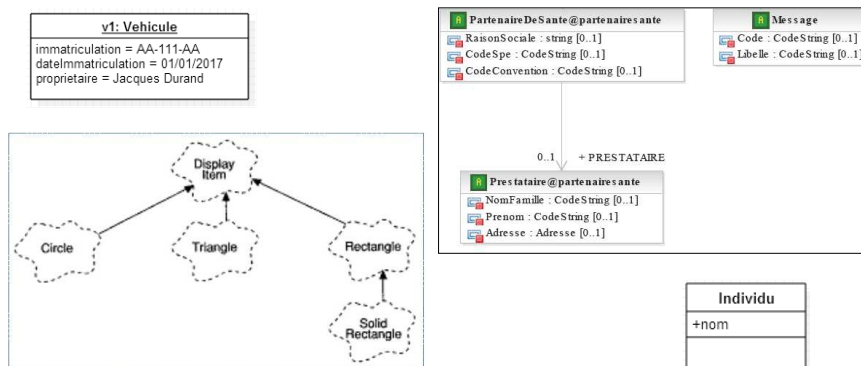


## UML c'est quoi exactement ? – La genèse

- ♦ UML a été « boosté » par l'essor de la programmation orientée objet.
- ♦ Historique de l'approche orientée objet :
  - Simula (1967),
  - Smalltalk (1976),
  - C++ (1985),
  - Java (1995),
  - .net ...
- ♦ UML permet de modéliser une application selon une vision objet, indépendamment du langage de programmation.

## UML c'est quoi exactement ? – La genèse

- ♦ Comment représenter une classe ? Un objet ?



- ♦ Tout est défini dans la spécification UML !
  - <http://www.omg.org/spec/UML/>

## UML c'est quoi exactement ? – La portée

- ♦ UML reste au niveau d'un langage et ne propose pas de processus de développement
  - ni ordonnancement des tâches,
  - ni répartition des responsabilités,
  - ni règles de mise en œuvre.
- ♦ Il existe de (très) nombreux outils pour faire de la modélisation UML.
- ♦ Certains ouvrages et AGL basés sur UML proposent un processus en plus de UML.
  - Exemple : le processus unifié UP.

## UML c'est quoi exactement ? – Le méta-modèle

- ◆ UML est bien plus qu'un outil pour dessiner des représentations mentales !
- ◆ La notation graphique n'est que le support du langage.
- ◆ UML repose sur un **méta-modèle**, qui normalise la sémantique de l'ensemble des concepts.

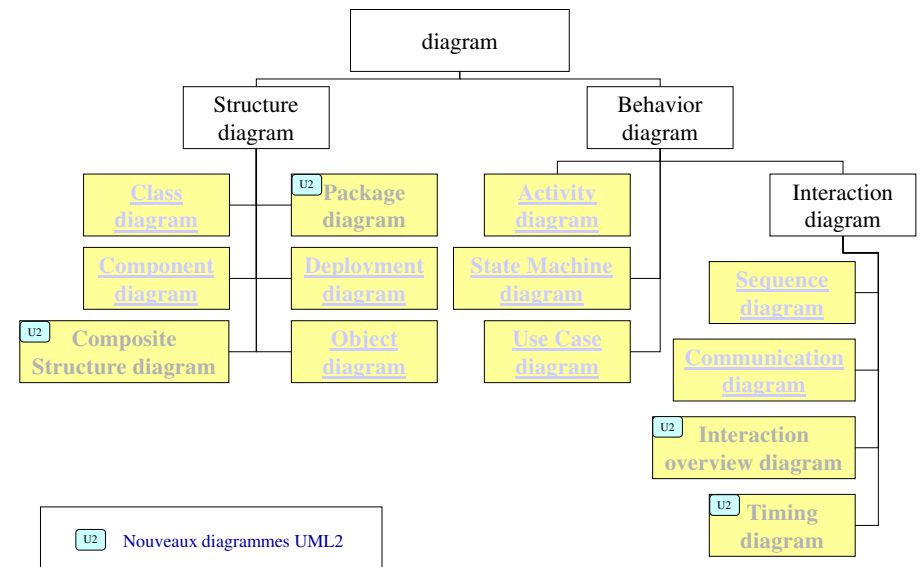
## UML c'est quoi exactement ? – En résumé

- ◆ UML est un langage de modélisation objet
- ◆ UML n'est pas une méthode
- ◆ UML convient pour tous les types de systèmes, tous les domaines métiers, et tous les processus de développement
- ◆ UML est dans le domaine public
- ◆ Les concepts véhiculés dans UML sont définis et non équivoques

## Tour d'horizon des diagrammes – 13 diagrammes !

- ◆ Historiquement UML proposait 9 types de diagrammes (UML 1.x).
- ◆ UML 2 a enrichi les concepts des diagrammes existants et a ajouté 4 nouveaux types de diagrammes.
- ◆ Les diagrammes manipulent des concepts parfois communs (ex. classe, objet, ...), parfois spécifiques (ex. cas d'utilisation).
- ◆ Quelle différence entre un modèle et un diagramme ?

## Tour d'horizon des diagrammes – 13 diagrammes !



## Tour d'horizon des diagrammes – 13 diagrammes !

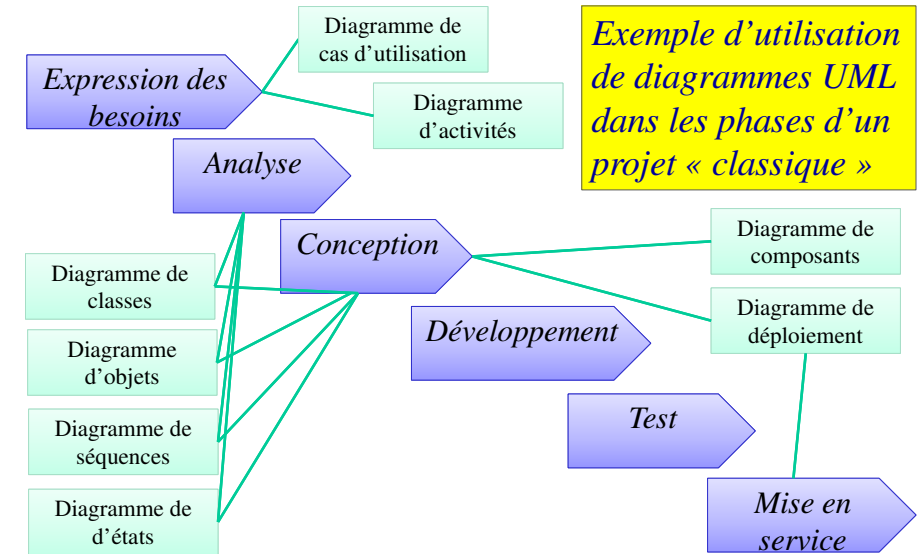
◆ UML est une grande boîte à outils, et comme toute boîte à outils ....

- On utilise pas tout ...
- Certains outils servent plus souvent que d'autres ...
- Dans chaque situation il faut choisir le bon outil !



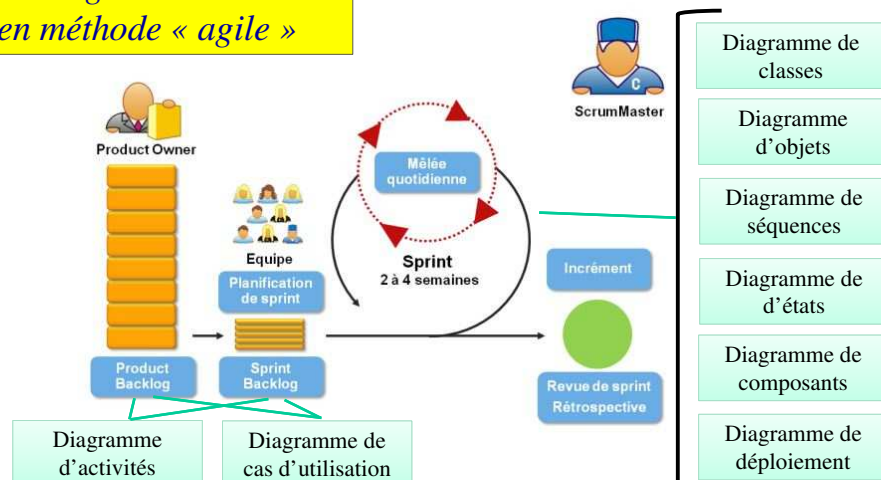
◆ En tant que langage, il faut aussi s'assurer que l'on va être compris !

## Tour d'horizon des diagrammes – 13 diagrammes !



## Tour d'horizon des diagrammes – 13 diagrammes !

*Exemple d'utilisation de diagrammes UML en méthode « agile »*



## Tour d'horizon des diagrammes – Restrictions & extensions

- ◆ **Restrictions** : au sein d'une organisation ou d'une équipe, on utilise un « sous-ensemble » de UML :
  - Sous-ensemble de diagrammes,
  - Sous-ensemble des possibilités offertes par chaque diagramme.
- ◆ **Extensions** : UML possède des mécanismes d'extension, qui permettent d'adapter le langage à une organisation, une équipe ou un domaine particulier.
- ◆ Un **profil** UML est un ensemble cohérent de concepts UML, d'extensions/restrictions, de contraintes, de règles et notations.
  - Exemples : profil EJB, profil SIG, profil RT, ...



## II Le diagramme de classes et le diagramme d'objets

### Concepts de base



## Classes et objets

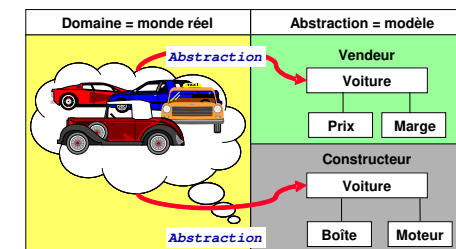
- ◆ Le diagramme de classes permet de représenter des classes, leurs propriétés et leurs relations avec d'autres classes.
- ◆ Le diagramme d'objets permet de représenter des objets, les valeurs de leurs propriétés et leurs relations avec d'autres objets.
- ◆ Mais au fait ....
  - Qu'est ce qu'un objet ?
  - Qu'est ce qu'une classe ?
  - Une classe peut-elle être un objet ? Et inversement ?

## Les objets

- ◆ Un objet est une entité identifiable du monde réel.
- ◆ Les objets informatiques définissent une représentation simplifiée des entités du monde réel.
- ◆ Un objet informatique est une structure de données valorisées qui répond à un ensemble de messages.
- ◆ Un objet peut représenter une entité concrète (personne, guitare, véhicule, ...) ou abstraite (gestionnaire de flux, ...).

## Les objets

- ◆ Une **abstraction** est un résumé, un condensé, une mise à l'écart des détails non pertinents dans un contexte donné.
- ◆ Mise en avant des caractéristiques essentielles et utiles.
- ◆ Dissimulation des détails (complexité).



## Les objets

---

- ◆ L'état d'un objet :
  - regroupe les valeurs instantanées de tous les attributs d'un objet
  - évolue au cours du temps
  - est la conséquence des comportements passés
- ◆ Exemples :
  - un signal électrique : l'amplitude, la pulsation, la phase, ...
  - une voiture : la marque, la puissance, la couleur, le nombre de places assises, ...
  - un étudiant : le nom, le prénom, la date de naissance, l'adresse, ...

## Les objets

---

- ◆ Le comportement
  - décrit les actions et les réactions d'un objet
  - regroupe toutes les compétences d'un objet
  - se représente sous la forme d'opérations (méthodes)
- ◆ Un objet peut faire appel aux compétences d'un autre objet
- ◆ L'état et le comportement sont liés
  - Le comportement dépend souvent de l'état
  - L'état est souvent modifié par le comportement

## Les objets

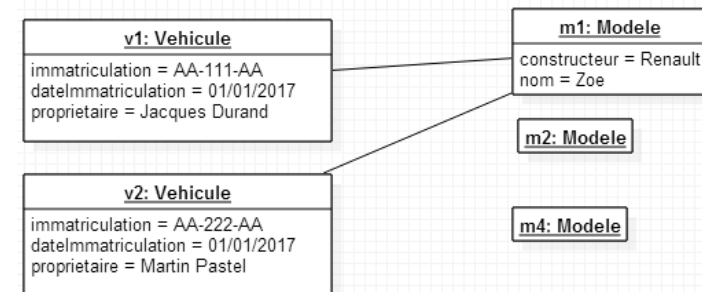
---

- ◆ Tout objet possède une identité qui lui est propre et qui le caractérise.
- ◆ L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état.

## Les objets

---

- ◆ Le diagramme d'objets permet la représentation d'objets du système que l'on modélise.
- ◆ Exemple :



## Communication entre objets

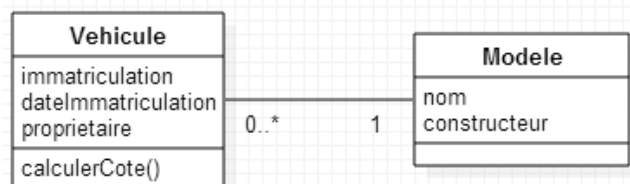
- ◆ Application = société d'objets collaborant
- ◆ Les objets travaillent en synergie afin de réaliser les fonctions de l'application.
- ◆ Le comportement global d'une application repose sur la communication entre les objets qui la composent.
- ◆ Les objets
  - ne vivent pas en ermites,
  - interagissent les uns avec les autres,
  - communiquent en échangeant des messages.

## Les classes

- ◆ La classe
  - est une description abstraite d'un ensemble d'objets
  - peut être vue comme la factorisation des éléments communs à un ensemble d'objets
  - c'est un modèle d'objets ayant les mêmes types de propriétés et de comportements. Chaque instance d'une classe possède ses propres valeurs pour chaque attribut.

## Les classes

- ◆ Le diagramme de classes permet la représentation des descripteurs d'objets du système que l'on modélise.
- ◆ Exemple :



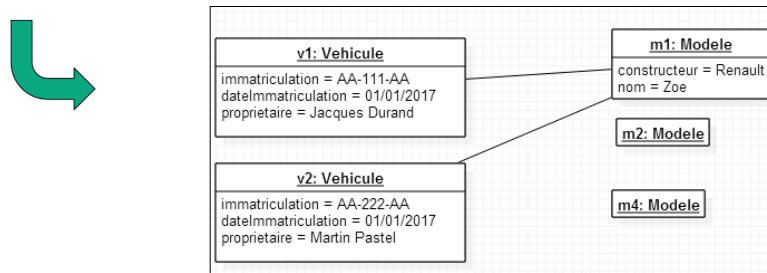
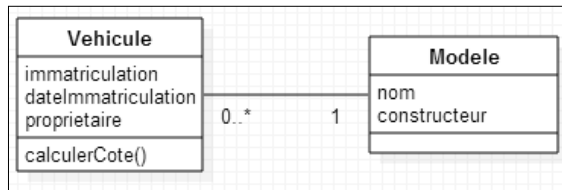
## Les classes

- ◆ Le diagramme de classes est le diagramme le plus connu et le plus utilisé.
- ◆ Structure interne statique d'un système.
- ◆ Diagrammes de classes et d'objets sont liés :
  - Le diagramme de classes permet de représenter :
    - des classes (ensembles d'objets),
    - des relations entre classes (ensemble de liens entre objets).
  - Le diagramme d'objets permet de représenter :
    - des instances particulières des classes, des objets,
    - les liens entre ces objets.



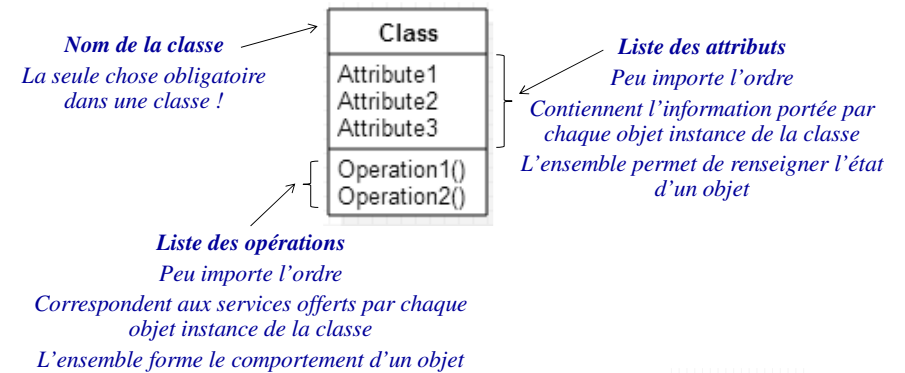
## Les classes

- ◆ Diagrammes de classes et d'objets sont liés !

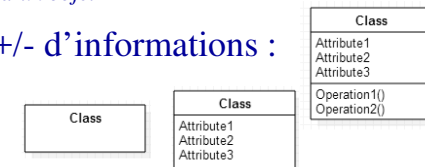


## Diagrammes de classes – concepts de base

- ◆ Représentation d'une classe :



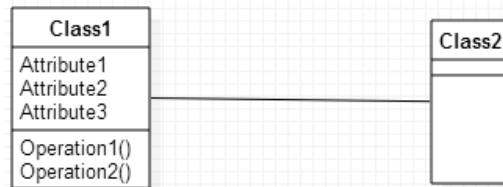
- ◆ Possibilité d'afficher +/- d'informations :



## Diagrammes de classes – concepts de base

- ◆ L'association exprime une connexion structurelle entre classes.

- ◆ Notation :



- ◆ Exemple : « une personne est employée par une entreprise » pourra se traduire par :



## Diagrammes de classes – concepts de base

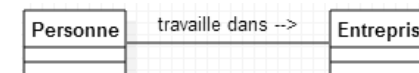
- ◆ Oui mais ... l'association entre Personne et Entreprise



peut être interprétée de plusieurs façons :

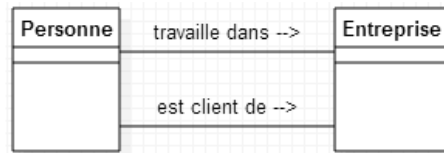
- « une personne travaille dans une entreprise »
- « une personne dirige une entreprise »
- « une personne est cliente d'une entreprise »
- ....

- ◆ Il est alors possible d'indiquer la signification de l'association :

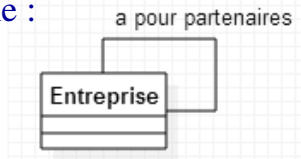


## Diagrammes de classes – concepts de base

- ◆ Sur un diagramme, il est possible de créer plusieurs associations entre les mêmes classes :



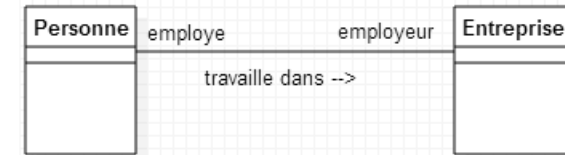
- ◆ Il est aussi possible de créer une association d'une classe avec elle-même :



- ◆ Comment cela se traduit il au niveau des objets ?

## Diagrammes de classes – concepts de base

- ◆ Chaque extrémité de l'association peut aussi être nommée :

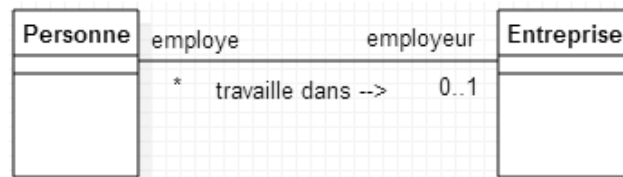


- ◆ Le rôle décrit comment les objets de la classe correspondante sont perçus par les objets de la classe à l'autre extrémité de l'association.

- ◆ Un rôle a la même nature qu'un attribut...

## Diagrammes de classes – concepts de base

- ◆ Chaque rôle porte une indication de multiplicité : il s'agit du nombre d'objets de la classe considérée pouvant être liés à un objet de l'autre classe.



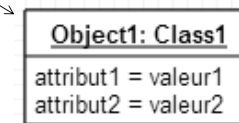
- ◆ La multiplicité s'exprime sous la forme d'un intervalle : nombre minimal et nombre maximal d'objets en relation.

Valeur	Signification
1	un et un seul
0..1	zéro ou un
M .. N	de M à N (entiers naturels)
* ou 0..*	de zéro à plusieurs
1 .. *	un à plusieurs

## Diagrammes d'objets – concepts de base

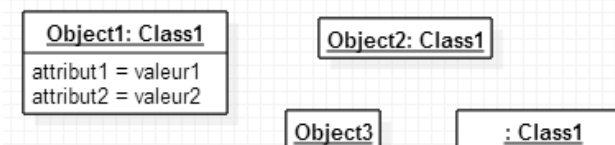
- ◆ Représentation d'un objet :

Nom de l'objet : nom de la classe  
Souligné



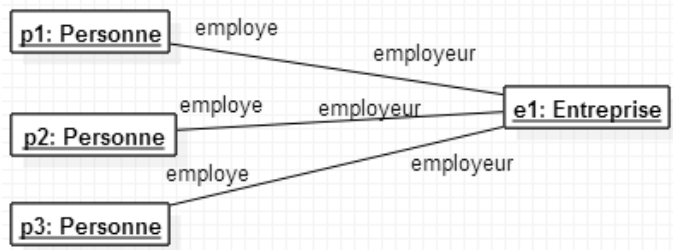
Valeurs des attributs  
Peu importe l'ordre  
L'ensemble correspond à l'état d'un objet

- ◆ Possibilité d'afficher +/- d'informations :



## Diagrammes d'objets – concepts de base

- ◆ Les associations entre classes permettent de créer des liens entre objets :



## Exercice 1



- ◆ Réaliser un diagramme de classes pour représenter les concepts suivants issus d'un système de réservation de véhicules :
  - Un client peut effectuer des réservations de véhicules.
  - Chaque client est décrit par son nom, son prénom et son numéro d'inscription.
  - Un véhicule possède un numéro d'immatriculation, une marque, une date de mise en service, une puissance fiscale, une vitesse maximale.
  - Pour un client donné, on souhaite pouvoir calculer la puissance moyenne des véhicules qu'il réserve.
- ◆ Illustrer votre diagramme de classes par un diagramme d'objets.



université  
PARIS-SACLAY

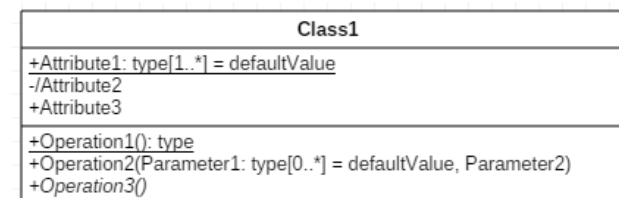
### III Le diagramme de classes et le diagramme d'objets

### Concepts avancés



## Propriétés complètes des attributs

- ◆ La représentation complète d'une classe fait apparaître les attributs avec différentes caractéristiques en plus du nom (type, valeur par défaut, degré de visibilité, ...), et les opérations avec leur signature complète :



## Propriétés complètes des attributs

- ◆ La forme complète de représentation d'un attribut est la suivante :  
`<visibilité> <nomAttribut> : <type> [borneInf..borneSup] = <valeur par défaut> {propriétés}`
- ◆ Seul le nom est obligatoire !

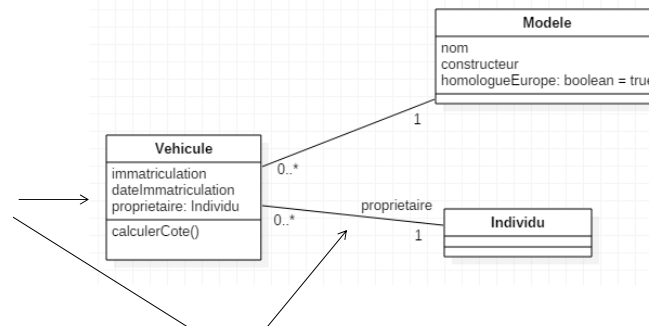
## Propriétés complètes des attributs

- ◆ Le symbole de visibilité correspond au concept objet d'encapsulation. Il représente le degré de protection de l'attribut :
  - + : publique (accessible à toutes les autres classes)
  - # : protégé (accessibles uniquement aux sous-classes)
  - ~ : paquetage (accessible uniquement aux classes du paquetage)
  - - : privé (inaccessible à tout objet hors de la classe)
- ◆ La visibilité peut être précisée sur chaque attribut et sur chaque opération.

## Propriétés complètes des attributs

- ◆ Le type permet de fixer l'ensemble des valeurs possibles que peut prendre un attribut.
- ◆ Il peut s'agir :
  - d'un type standard : integer, string, boolean, real
  - d'une classe : on préférera très souvent utiliser une association

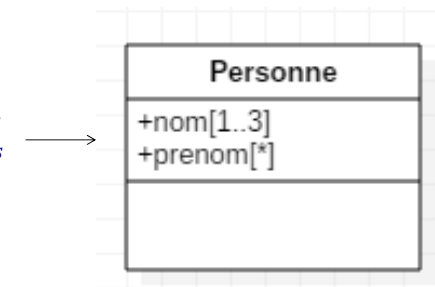
*Préférez l'association et supprimez l'attribut « propriétaire » !*



## Propriétés complètes des attributs

- ◆ Un attribut peut prendre plusieurs valeurs.
- ◆ Il est alors possible de préciser le nombre minimal et le nombre maximal de valeurs que peut prendre l'attribut.

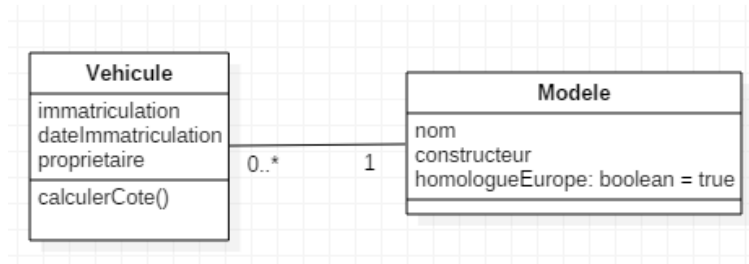
*On spécifie ici qu'une personne possède entre 1 et 3 noms et 0 à plusieurs prénoms*



- ◆ En l'absence d'indication, la valeur est unique ([1..1]).

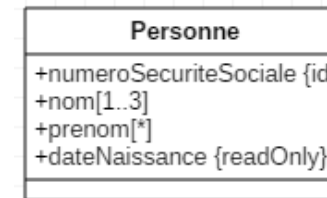
## Propriétés complètes des attributs

- ◆ Une valeur par défaut peut être précisée sur un attribut. Cette valeur est affectée à l'attribut à la création des instances de la classe.



## Propriétés complètes des attributs

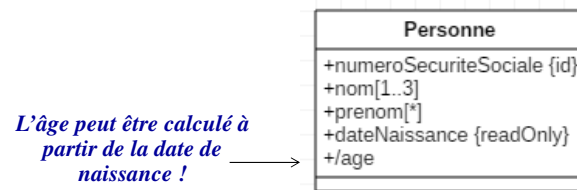
- ◆ Il est possible d'attribuer des propriétés aux attributs, en les indiquant entre accolades.



- ◆ Parmi les propriétés les plus utilisées :
  - {readOnly} indique que la valeur de l'attribut ne peut pas être modifiée,
  - {id} indique que l'attribut fait partie de l'identifiant des instances de la classes.

## Propriétés complètes des attributs

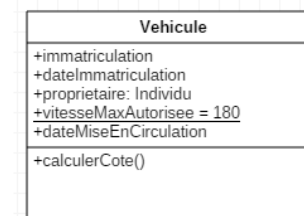
- ◆ Lorsque la valeur d'un attribut peut être calculée par une fonction basée sur la valeur d'autres attributs, on parle d'attribut calculé ou attribut dérivé.
- ◆ Un attribut calculé est précédé du signe « / »



## Propriétés complètes des attributs

- ◆ Chaque instance d'une classe contient une valeur spécifique pour chacun de ses attributs. Dans certains cas cependant, il est utile de pouvoir définir des attributs dont la valeur est commune à l'ensemble des instances de la classe.
- ◆ On parle alors d'attribut statique ou attribut de classe, et on souligne l'attribut.

*La vitesse maximale autorisée est la même pour toutes les instances de la classe !*



## Propriétés complètes des opérations

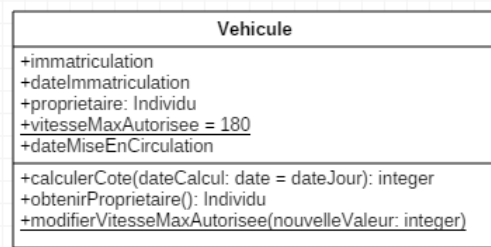
- ◆ La forme complète de représentation d'une opération est la suivante :  
`<visibilité> <nomOpération> (listeParamètres) : <typeRetour> [borneInf..borneSup] {propriétés}`
- ◆ Comme pour les attributs, seul le nom, suivi des parenthèses ( ), est obligatoire !
- ◆ Chaque paramètre est décrit sous la forme :  
`<direction> <nom> : <type> [borneInf...borneSup] = <valeur par défaut> {propriétés}`

## Propriétés complètes des opérations

- ◆ Certains concepts présentés pour les attributs s'appliquent aux opérations :
  - La visibilité de l'opération,
  - L'intervalle pour indiquer le nombre de valeurs du retour,
  - Sur les paramètres : le type, l'intervalle pour préciser le nombre de valeurs autorisées et la valeur par défaut,
  - Les propriétés.
- ◆ Il existe cependant des différences :
  - Lorsque l'opération renvoie un objet, il n'est pas possible de remplacer le typeRetour par une association !
  - Un paramètre peut être préfixé en indiquant sa direction : in, out ou inout.

## Propriétés complètes des opérations

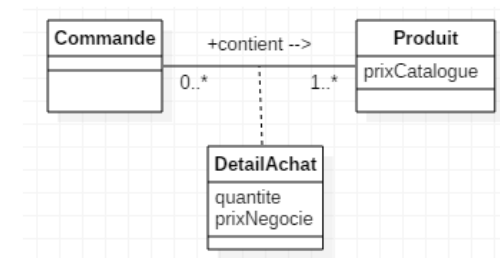
- ◆ Il est également possible qu'une opération soit définie comme « statique » : cette opération est appelée sur la classe directement.



- ◆ Ces opérations ne peuvent pas manipuler d'attributs qui ne soient pas statiques !

## Classe d'association

- ◆ La classe d'association est ... une classe ! Cette classe permet de faire porter des informations aux liens entre instances de classes.



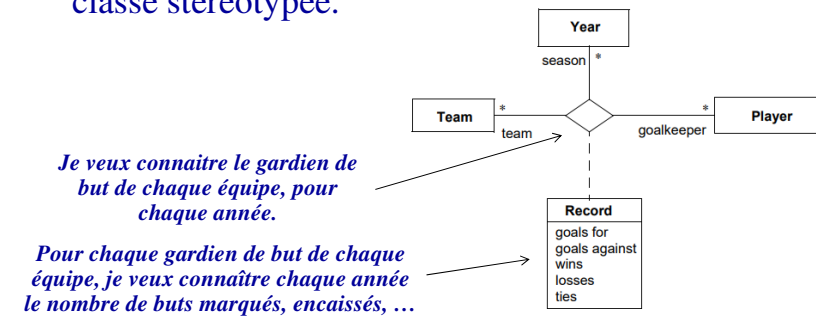
- ◆ Une telle classe peut être dotée d'attributs, d'opérations, et de relations avec d'autres classes.

## Classe d'association

- ◆ Exemples d'utilisation de la classe d'association :
  - Des personnes empruntent des livres à la bibliothèque. Il est nécessaire de pouvoir retrouver la date de chaque emprunt.
  - Des personnes signent des accords entre eux. Il est nécessaire de conserver la date de ces accords et la ville où a eu lieu la signature.
  - Des personnes travaillent dans des entreprises. Pour chacun de leurs emplois, il est nécessaire de connaître le temps de travail et le type de contrat signé avec l'entreprise.

## Associations n-aire

- ◆ L'association permet de relier plus de deux classes. On parle d'association ternaire pour trois classes, ou plus généralement d'association n-aire.
- ◆ Représentation : un losange blanc ou au moyen d'une classe stéréotypée.



## Associations n-aire

- ◆ Attention, il est souvent possible de modéliser un problème avec plusieurs associations binaires plutôt qu'avec une association n-aire.
- ◆ Exemple :
  - Un employé emprunte un véhicule de fonction pour se rendre sur un site
  - Un employé possède un véhicule de fonction. L'employé se rend sur des sites avec son véhicule de fonction

## Navigabilité d'une association

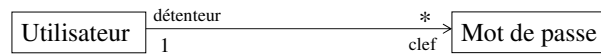
- ◆ Les associations permettent par défaut une navigation bidirectionnelle : il est possible de déterminer les liens de l'association depuis les instances de chaque classe.
- ◆ Il est possible de limiter la navigabilité en spécifiant un seul sens de navigation :



## Navigabilité d'une association

### ◆ Exemple :

*Une instance d'utilisateur peut accéder à des instances de Mot de passe, mais pas l'inverse.*



### ◆ A ne pas confondre avec le sens de lecture de l'association !

## Les différentes relations entre classes

### ◆ Concernant les relations entre classes, il existe d'autres relations en plus de l'association :

- L'agrégation,
- La composition,
- La généralisation/spécialisation,
- La dépendance.

## Agrégation et composition

### ◆ L'agrégation est une forme particulière d'association binaire, mais dissymétrique, où l'une des extrémités est prédominante par rapport à l'autre.

### ◆ Représentation des relations de type :

- tout et parties,
- composé et composants,
- maître et esclaves.

### ◆ Exemple : un élevage est composé d'un certain nombre de chevaux :



## Agrégation et composition

### ◆ Deux types d'agrégation :

- **Agrégation** partagée – notion de co-propriété
  - La création (resp. la destruction) des composants est indépendante de la création (resp. la destruction) du composite
  - Un objet peut faire partie de plusieurs composites à la fois
  - Notation :



- **Composition**

- Cas particulier de l'agrégation : attributs contenus physiquement par l'agrégat
- La création (resp. la destruction) du composite entraîne la création (resp. la destruction) des composants.
- Un objet ne fait partie que d'un seul composite à la fois.
- Notation :



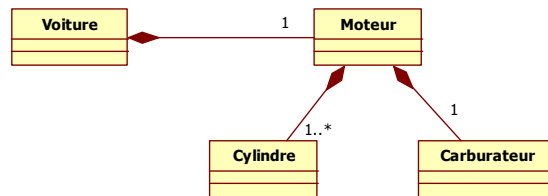


## Agrégation et composition

### ◆ Agrégation

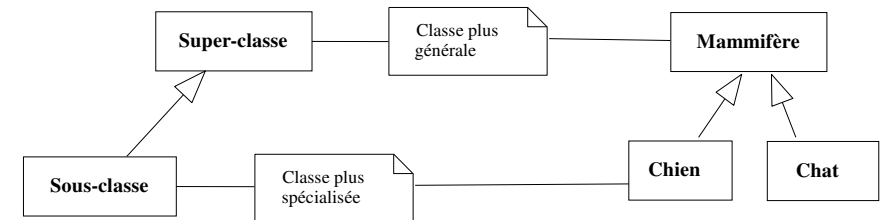


### ◆ Composition



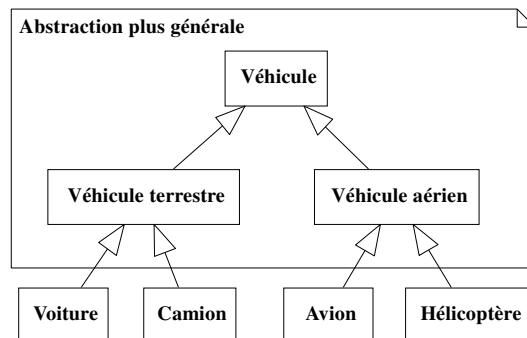
## Généralisation/Spécialisation

- ◆ La relation de généralisation/spécialisation permet de gérer les relations « est du type » ou « est une sorte de »
- ◆ Cette relation correspond au concept d'héritage en objet.



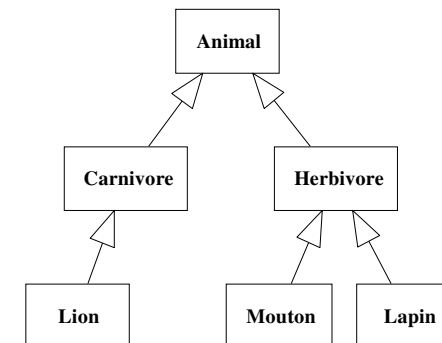
## Généralisation/Spécialisation

- ◆ Les instances d'une classe sont aussi instances des super-classes. Elle « profitent » des attributs, opérations et relations définies dans les super-classes.



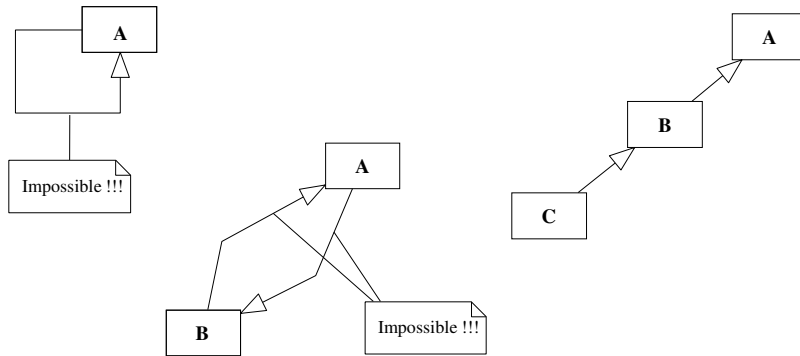
## Généralisation/Spécialisation

- ◆ La généralisation peut se traduire par « *est un* » ou « *est une sorte de* »



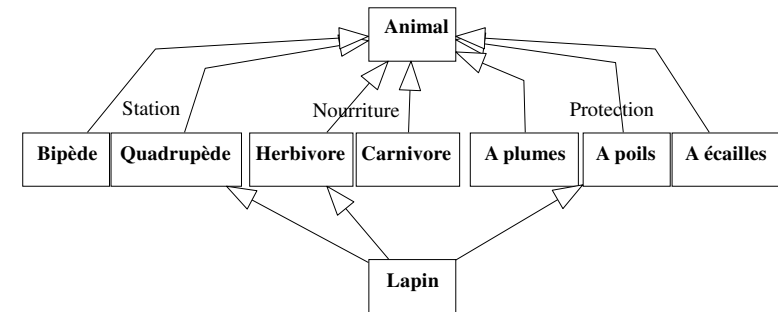
## Généralisation/Spécialisation

- ◆ Propriétés de la généralisation : non-réflexive, non-symétrique, transitive



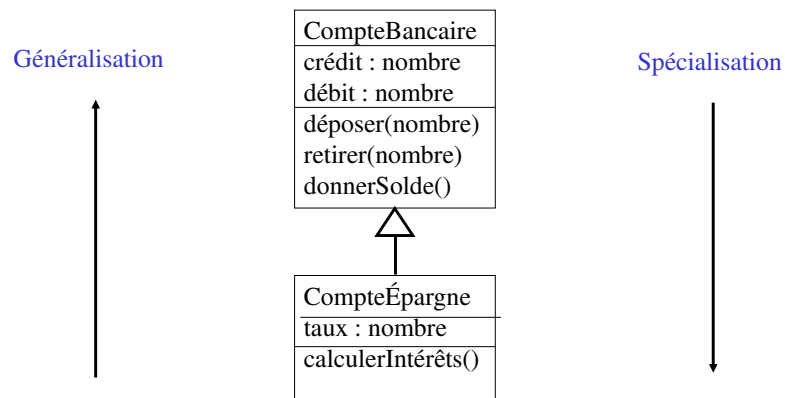
## Généralisation/Spécialisation

- ◆ La généralisation multiple est modélisable en UML !



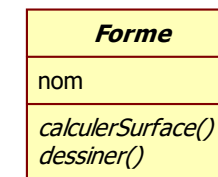
## Généralisation/Spécialisation

- ◆ Exemple :



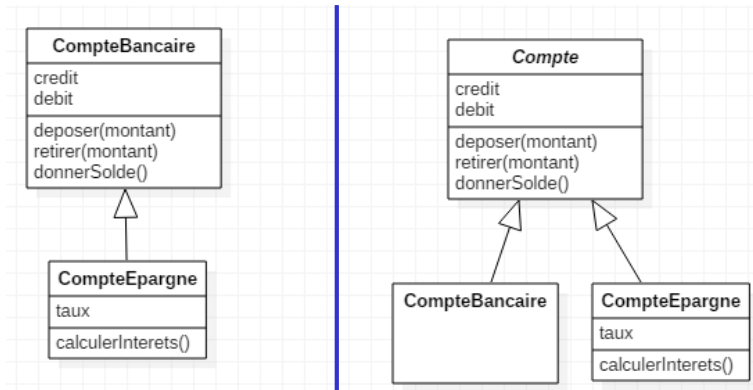
## Généralisation/Spécialisation

- ◆ Classe abstraite : classe qui ne peut avoir aucune instance directe ; on écrit son nom en *italique*
- ◆ Opération abstraite : opération incomplète qui a besoin de la classe fille pour fournir une implémentation ; on écrit son nom en italique.



## Généralisation/Spécialisation

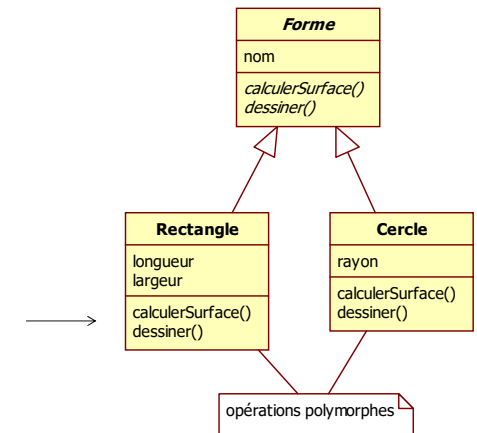
- ◆ Ces deux modèles sont ils équivalents ?



## Généralisation/Spécialisation - polymorphisme

- ◆ Le polymorphisme est un concept objet selon lequel un même message peut être interprété de différentes façons, selon le récepteur.

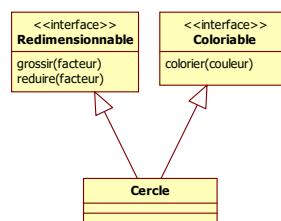
*Une méthode peut être définie par le même nom dans différentes classes.*



## Généralisation/Spécialisation - interfaces

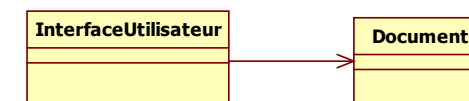
- ◆ L'interface est une classe totalement abstraite, sans attribut et dont toutes les opérations sont abstraites.
- ◆ La concept d'interface permet la définition d'un contrat pour toutes les classes qui l'implémentent. La relation entre une interface et une classe qui implémente l'interface est appelée relation de réalisation.

- ◆ Notation :



## Dépendance

- ◆ La relation de dépendance est une relation sémantique entre deux éléments selon laquelle un changement apporté à l'un peut affecter l'autre.
- ◆ Implique uniquement que des objets d'une classe peuvent fonctionner ensemble.
- ◆ Notation :



## Exercice 2



◆ Réaliser un diagramme de classe pour chaque énoncé :

- Une voiture :
  - Quatre roues, des pneus
  - Un moteur,
  - Un coffre,
  - Des passagers....
- Un chien : c'est un mammifère qui appartient à un propriétaire, possède quatre membres, donne naissance éventuellement à des chiots, ....

## Les diagrammes un par un



## IV Le diagramme de cas d'utilisation

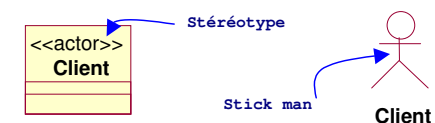


## Le diagramme de cas d'utilisation

- ◆ Ce diagramme permet une description, en prenant le **point de vue de l'utilisateur**, du système à construire.
- ◆ Pas d'aspects techniques.
- ◆ Les deux concepts de base du diagramme :
  - l'acteur,
  - Le cas d'utilisation.

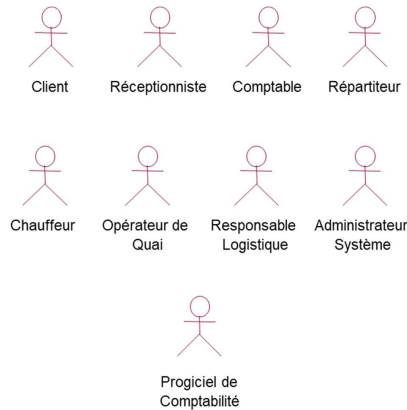
## Acteur - définition

- ◆ Un acteur est une **entité externe** au système :
  - qui attend un ou plusieurs services du système,
  - à qui le système fournit une interface d'accès,
  - qui interagit avec le système par échange de messages.
- ◆ C'est une personne ou un autre système
- ◆ Les acteurs sont décrits par une abstraction ne retenant que le rôle qu'ils jouent vis à vis du système
- ◆ Notation :



## Acteur - définition

### ◆ Exemples d'acteurs :



### ◆ Tous, humains ou système, interagissent avec le système considéré !

## Acteurs vs Utilisateurs

- ◆ On ne doit pas raisonner en terme d'entité physique, mais en terme de **rôle** que l'entité physique joue
- ◆ Un acteur représente un rôle joué par un utilisateur qui interagit avec le système
- ◆ La même personne physique peut jouer le rôle de plusieurs acteurs (joueur, banquier)
- ◆ Plusieurs personnes peuvent également jouer le même rôle, et donc agir comme le même acteur (tous les joueurs)

## Cas d'utilisation - définition

- ◆ Un cas d'utilisation modélise un **service rendu** par le système
- ◆ Il exprime les interactions entre les acteurs et le système
- ◆ Il apporte une valeur ajoutée "notable" aux acteurs concernés.
- ◆ Règle de nommage : verbe (+ complément)
- ◆ Notation :

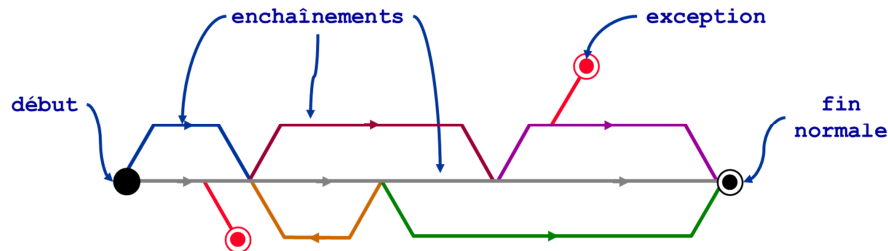


## Cas d'utilisation, enchaînements et scénarios

- ◆ **Cas d'utilisation** : représente un cas en général, une représentation générale et synthétique d'un ensemble de scénarios similaires décrits sous la forme d'enchaînements  
*Exemple* : un joueur joue un coup en lançant 2 dés
- ◆ **Enchaînement** : succession d'étapes qui se réalisent lorsqu'un acteur déclenche un cas d'utilisation.  
*Exemple* : un joueur joue un coup, les 2 dés ont une valeur identique, le joueur peut ensuite rejouer un autre coup
- ◆ **Scénario** : exécution d'un ou plusieurs enchaînements, joignant le début du cas d'utilisation à une fin normale ou non.  
*Exemple* : le joueur Pierre joue : il obtient 7 avec les dés, se déplace rue de Belleville et achète la propriété

## Cas d'utilisation vs scénario

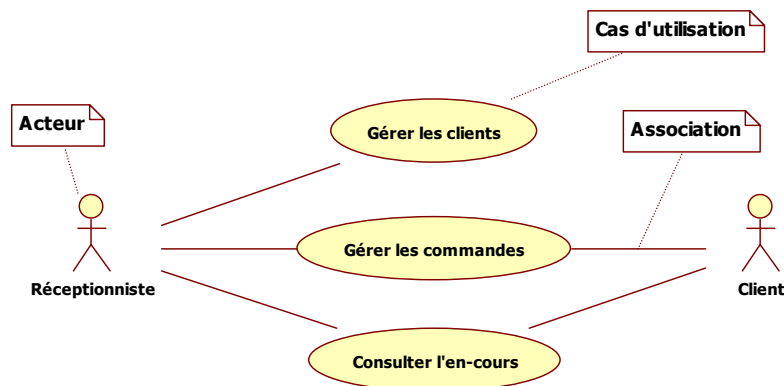
- ◆ Un cas d'utilisation contient en général plusieurs enchaînements.
- ◆ Pour décrire lisiblement un cas d'utilisation, on distinguera les trois types d'enchaînements :
  - Nominal : déroulement normal du cas
  - Alternatif : embranchements dans la séquence nominale
  - Exception : interviennent quand une erreur se produit



## Intérêt des cas d'utilisation

- ◆ Un moyen de déterminer le **but** et le **périmètre** d'un système
- ◆ Utilisé par les utilisateurs finaux pour exprimer leur attentes et leur besoins → permet d'impliquer les utilisateurs dès les premiers stades du développement
- ◆ Support de communication entre les équipes et les clients
- ◆ Découpage du système global en grandes tâches qui pourront être réparties entre les équipes de développement
- ◆ Permet de concevoir les Interfaces Homme-Machine
- ◆ Constitue une base pour les tests fonctionnels

## Exemple de diagramme de cas d'utilisation

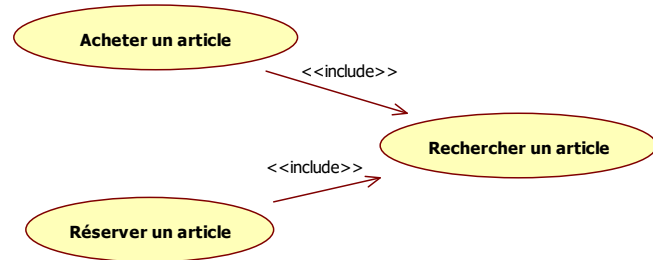


## Relations entre cas d'utilisation

- ◆ UML définit trois types de relations standardisées entre cas d'utilisation :
  - une relation d'inclusion,
  - une relation d'extension,
  - une relation de généralisation/spécialisation.
- ◆ On peut également généraliser les acteurs

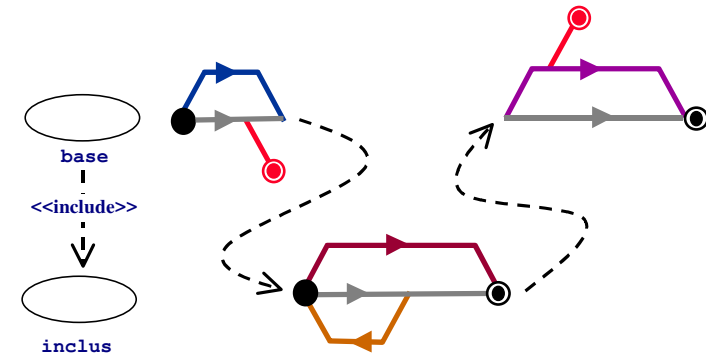
## Relations entre cas d'utilisation

- ◆ `<<include>>` permet d'incorporer le comportement d'un autre cas d'utilisation.
- ◆ Le cas de base en incorpore explicitement un autre, à un endroit spécifié dans sa description.



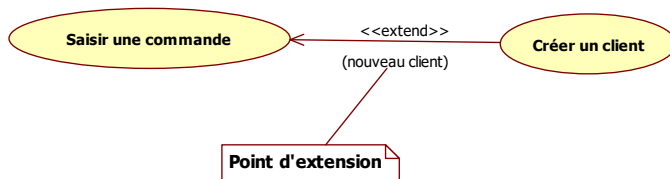
## Relations entre cas d'utilisation

- ◆ La relation `<<include>>` se traduit de la façon suivante en terme d'enchaînements :



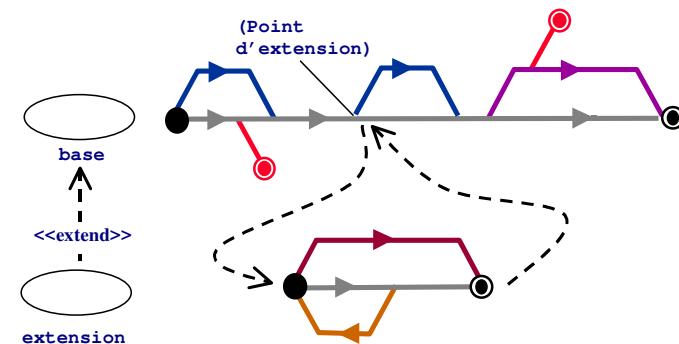
## Relations entre cas d'utilisation

- ◆ `<<extend>>` permet de modéliser la partie d'un cas d'utilisation considérée comme facultative dans le comportement du système.
- ◆ Le cas de base peut fonctionner seul, mais il peut aussi être complété par un autre, sous certaines conditions, et uniquement à certains points particuliers de son flot d'événements appelés points d'extension.



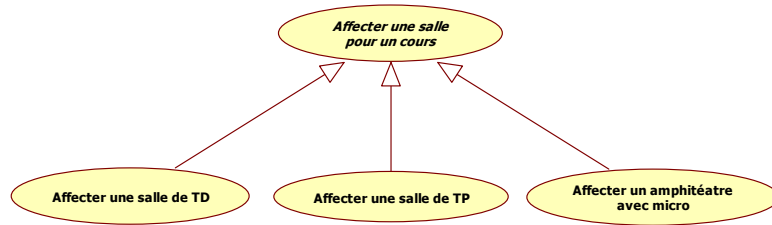
## Relations entre cas d'utilisation

- ◆ La relation `<<extend>>` se traduit de la façon suivante en terme d'enchaînements :

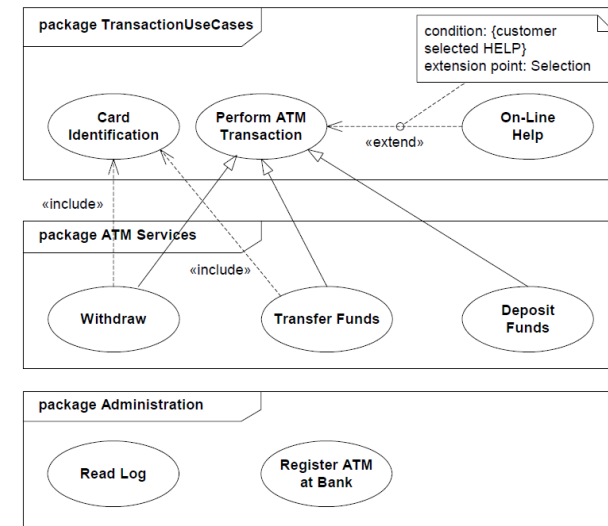


## Relations entre cas d'utilisation

- ◆ Les cas d'utilisation peuvent être hiérarchisés par généralisation.
- ◆ Les cas d'utilisation descendants héritent de la sémantique de leur parent. Ils peuvent comprendre des interactions spécifiques supplémentaires, ou modifier les interactions héritées.



## Relations entre cas d'utilisation



## Spécification

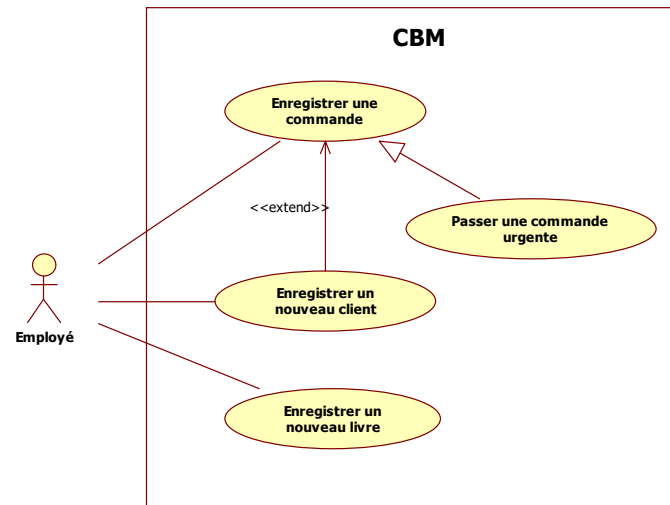
- ◆ La description textuelle n'est pas normalisée par UML !
- ◆ Convergence vers un modèle standard :
  - Sommaire d'identification  
inclut titre, but, résumé, dates, version, responsable, acteurs...
  - Description des enchaînements  
décrit les enchaînements nominaux, les enchaînements alternatifs, les exceptions, mais aussi les préconditions, et les postconditions.
  - Exigences fonctionnelles
  - Besoins d'IHM  
ajoute éventuellement les contraintes d'interface homme-machine
  - Contraintes non-fonctionnelles  
ajoute éventuellement les informations suivantes : fréquence, volumétrie, disponibilité, fiabilité, intégrité, confidentialité, performances, concurrence, etc.
- ◆ On peut aussi réaliser des diagrammes

## Exemple : La CBM

- ◆ La CBM (Computer Books by Mail) est une société de distribution d'ouvrages d'informatique qui agit comme intermédiaire entre les librairies et les éditeurs.
- ◆ Elle prend des commandes en provenance des libraires, s'approvisionne (à prix réduit) auprès des éditeurs concernés et livre ses clients à réception des ouvrages
- ◆ Il n'y a donc pas de stockage de livres.
- ◆ Seules les commandes des clients solvables sont prises en compte.
- ◆ Les commandes « urgentes » font l'objet d'un traitement particulier.
- ◆ La CBM désire mettre en place un Système Informatique lui permettant de gérer les libraires et les livres, et d'enregistrer les commandes.



## Diagramme de cas d'utilisation



## Spécification textuelle du cas « Enregistrer une commande »

**Acteur** : l'employé de la coopérative

**Objectif** : enregistrer une commande de livres

**Précondition** : le libraire existe

**Enchaînement nominal** :

- 1 - l'employé sélectionne le libraire et vérifie sa solvabilité
- 2 - l'employé vérifie l'existence des livres
- 3 - l'employé précise la quantité pour chaque livre
- 4 - L'employé confirme la commande

**Postcondition** : une nouvelle commande est créée.

**Enchaînement d'exception 1** :

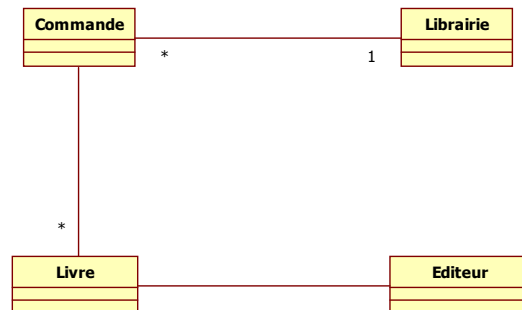
- 1a - le libraire n'est pas solvable
- 1b - le système alerte l'employé et lui propose d'arrêter l'enregistrement

**Enchaînement d'exception 2** :

- 2a - un des livres n'existe pas
- 2b - Le système édite une lettre qui pourra être envoyée au libraire. La commande est placée en attente.

## Diagrammes complémentaires

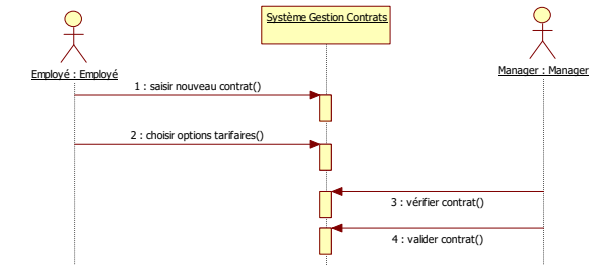
- ◆ On peut ajouter à chaque cas d'utilisation un diagramme de classes simplifié, appelé Diagramme de Classes Participantes (DCP)



## Diagrammes complémentaires

- ◆ Autres diagrammes possibles pour compléter la description d'un cas d'utilisation :

- Diagramme d'activités
- Diagramme de séquence (vision boîte noire)



## Pour finir : quelques pièges à éviter

- ◆ Des cas d'utilisation trop petits et trop nombreux
  - Jacobson : pas plus de 20 UC !
  - Larman : test du patron
  - Test de la taille
- ◆ Trop d'importance au diagramme
  - Pas trop de relations entre UC !
- ◆ Décomposition fonctionnelle
  - Garder le point de vue de l'utilisateur et pas le point de vue interne !
- ◆ Confusion entre processus métier et UC
  - Pas d'interactions entre acteurs directement !
  - Se concentrer sur les actions à automatiser !



## Exercice 3



Représenter par un diagramme de cas d'utilisation les éléments de l'énoncé suivant :

- ◆ Le système de déclaration des impôts en ligne permet aux contribuables :
  - De saisir toutes les informations relatives à leurs revenus de l'année précédente,
  - De mettre à jour leurs informations administratives (adresse postale, ...).
- ◆ La déclaration en ligne n'est accessible qu'aux contribuables authentifiés, au moyen de leur n° fiscal et de leur mot de passe.
- ◆ Lors de la saisie, le contribuable choisit de réaliser une déclaration simplifiée ou une déclaration complète.
- ◆ La déclaration d'impôts d'une année donnée n'est possible qu'après ouverture du service par les services fiscaux.

## Les diagrammes un par un



V

**Le diagramme de séquence**  
**Le diagramme de communication**



## Objectifs

- ◆ Décrire des scénarios particuliers
- ◆ Capturer l'ordre des interactions entre les parties du système (objets, composants, acteurs, ...)
- ◆ Décrire les interactions déclenchées lorsqu'un scénario d'un cas d'utilisation est exécuté

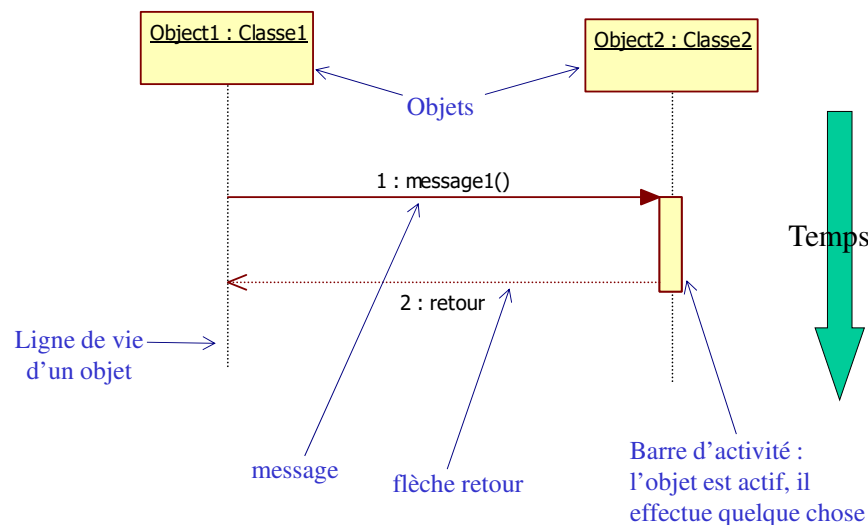
## Diagramme de séquence / diagramme de communication

- ◆ Ces diagrammes comportent :
  - des objets dans une situation donnée (instances)
  - les messages échangés entre les objets
- ◆ Les objets sont les instances des classes identifiées et décrites dans le diagramme de classes.

## Diagramme de séquence / diagramme de communication

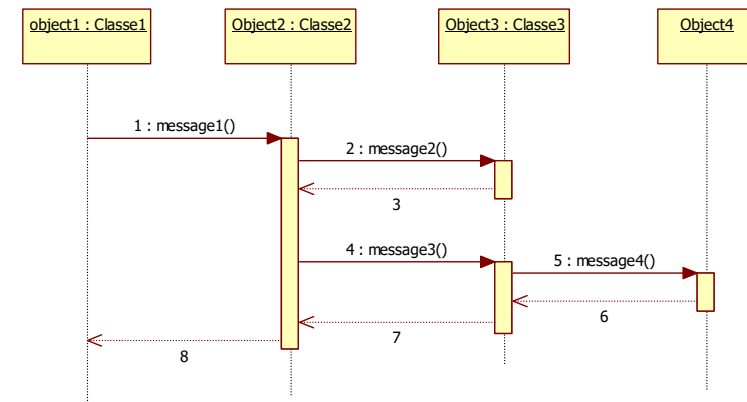
- ◆ UML propose deux types de diagrammes pour modéliser la collaboration entre les objets du système :
  - Le diagramme de séquences,
  - Le diagramme de communication
- ◆ Ces deux diagrammes sont regroupés sous le terme de diagrammes d'interactions.
- ◆ Nous nous focaliserons dans un premier temps sur le diagramme de séquences.

## Diagramme de séquence en détail



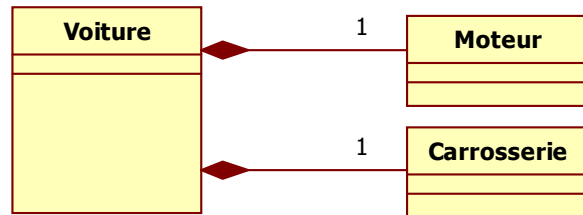
## Messages imbriqués

- ◆ Un message peut conduire à l'envoi d'un ou plusieurs messages par le récepteur



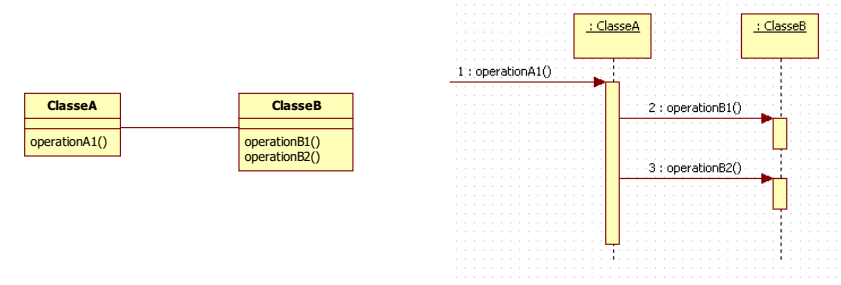
## Exemple

- ◆ Comment calculer le poids d'une voiture, égal au poids du moteur plus le poids de la carrosserie ?



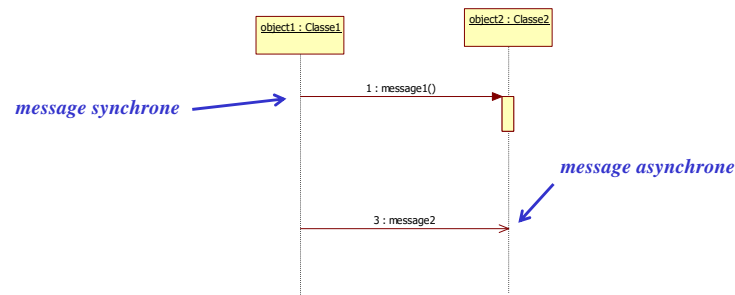
## Diagrammes de séquences et diagramme de classes

- ◆ Le diagramme de classes présente une vue statique du système.
- ◆ Le diagramme de séquences présente une vue dynamique du système.



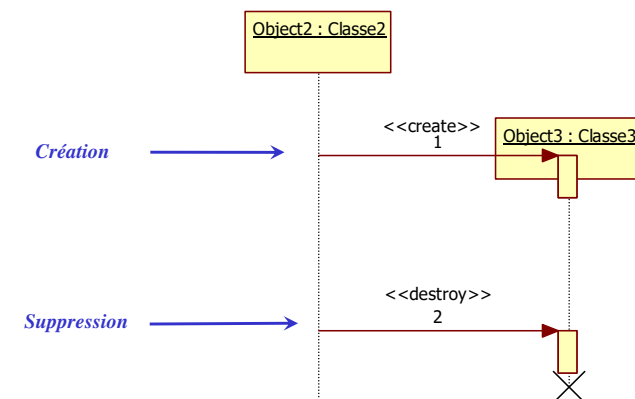
## Messages synchrones/asynchrones

- ◆ Message synchrone : l'émetteur attend le retour du récepteur du message
- ◆ Message asynchrone : l'émetteur n'est pas bloqué, il continue ses traitements



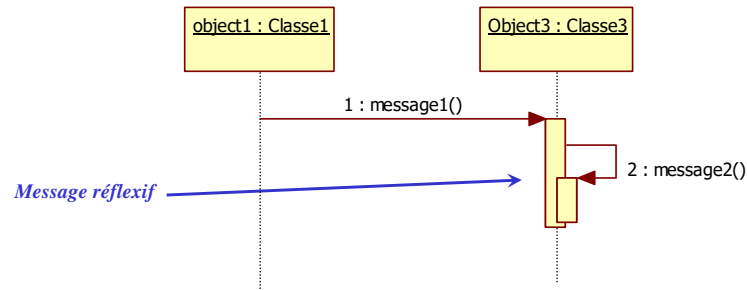
## Messages de création/destruction

- ◆ Rappel : les objets naissent, vivent et meurent



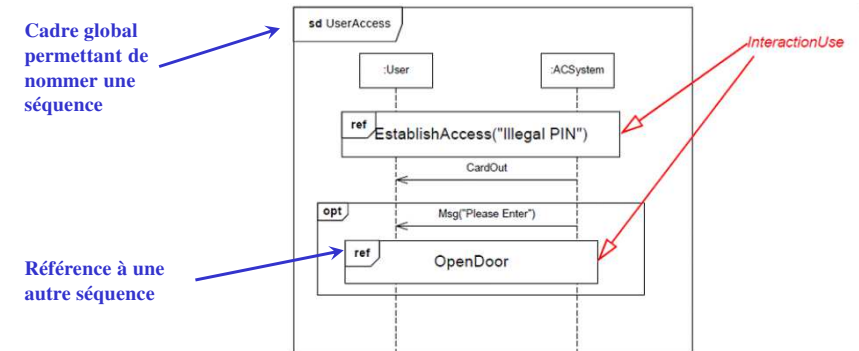
## Messages réflexifs

- ◆ Envoi d'un message d'un objet à lui-même



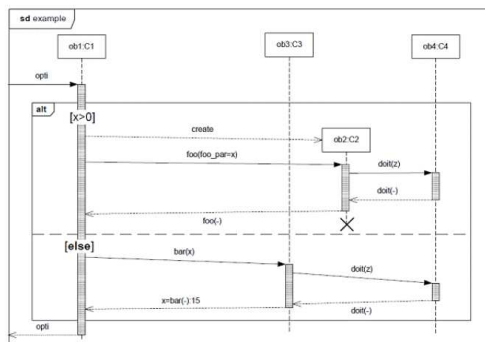
## Fragments (UML2)

- ◆ Cadres d'interaction et fragments combinés permettent, depuis UML 2, d'enrichir les diagrammes de séquence pour structurer les interactions complexes



## Fragments (UML2)

- ◆ De la même façon, les fragments combinés permettent d'enrichir un diagramme de séquence.
- ◆ Exemple : fragment de type **alt** (alternative) :



- ◆ Autres fragments utiles : **opt** (partie optionnelle), **loop** (boucle).

## Utilisation du diagramme de séquence

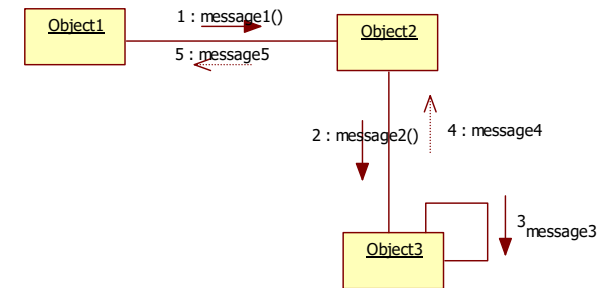
- ◆ Dans la vue fonctionnelle du système
  - Diagramme complémentaire de description d'un cas d'utilisation
  - Décrire un scénario d'un cas d'utilisation : décrire les interactions entre le système et son environnement (vision boîte noire)
  - Appelé diagramme de séquence « système »
- ◆ Dans l'analyse détaillée et la conception OBJET du système
  - Décrire les interactions internes du système : les interactions entre les objets

## Diagramme de communication

- ◆ Diagramme de séquences : l'accent est mis sur l'ordre temporel des interactions
- ◆ Diagramme de communication : l'accent est mis sur l'examen des interactions vis-à-vis des liens entre objets.
- ◆ Équivalents en UML1, quelques nouveautés sur le diagramme de séquences en UML2, non disponibles sur le diagramme de communication

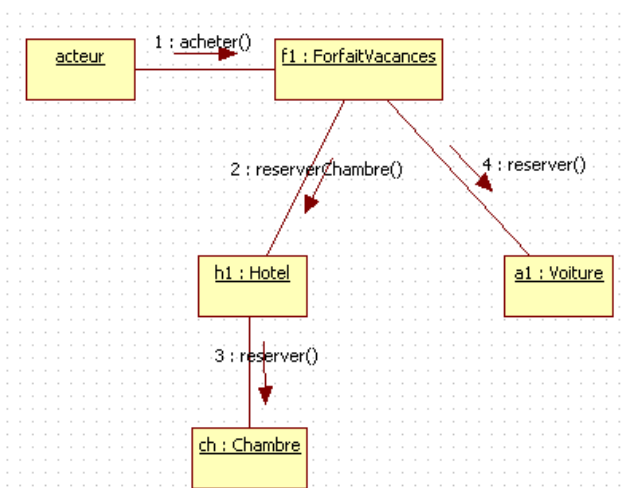
## Diagramme de communication

- ◆ Focalisation sur les liens entre objets d'une interaction



## Diagramme de communication

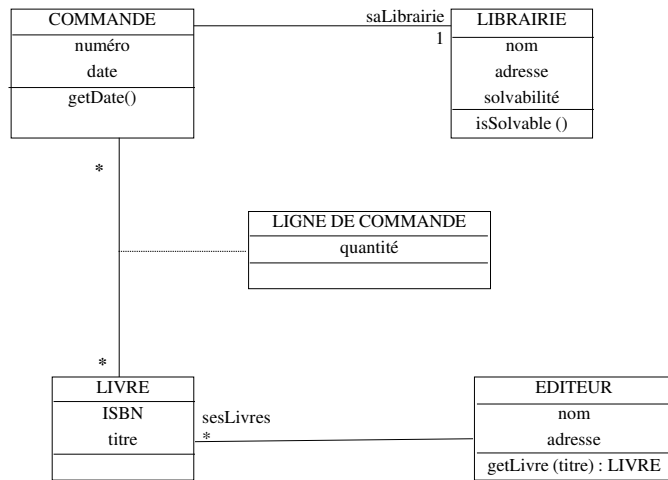
- ◆ Exemple :



## Exemple : La CBM

- ◆ Description du système :
  - Reprendre l'énoncé de l'exercice sur les cas d'utilisation.
- ◆ Objectif :
  - Réaliser un diagramme de classes pour le système CBM,
  - Réaliser un diagramme de séquence **objet** lorsque qu'un utilisateur demande les détails de toutes les commandes d'une librairie donnée.

## CBM : diagramme de classes



## CBM : un diagramme de séquence

- ◆ Diagramme de séquence : « Communiquer les détails de toutes les commandes d'une librairie donnée »
- ◆ Ce diagramme correspond à la méthode *getDetailCommandes()* de la classe LIBRAIRIE