

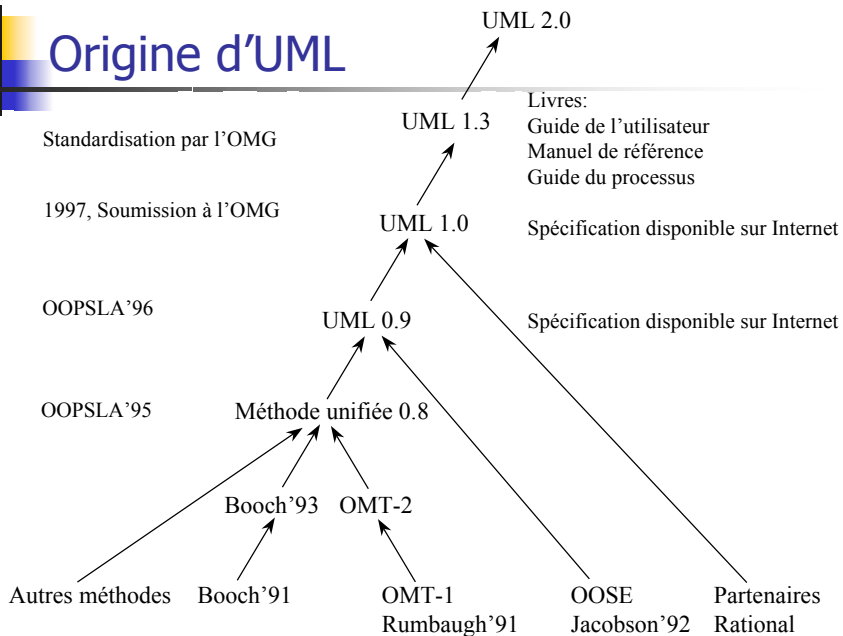
# Unified Modeling Language

*Langage unifié pour la modélisation objet*

J. Ralyté



## Origine d'UML







## Objectifs d'UML

- Proposer un langage visuel de modélisation
  - Utilisable par toutes les méthodes
  - Adapté à toutes les phases du développement
  - Compatible avec toutes les techniques de réalisation
- Proposer des mécanismes d'extension et de spécialisation pour pouvoir étendre les concepts de base
- Être indépendant des langages particuliers de programmation



## Objectifs d'UML

- Proposer une base formelle pour comprendre le langage de modélisation
- Encourager l'application des outils OO
- Permettre l'utilisation des concepts de développement de haut-niveau comme des cadres de référence (frameworks), des patrons (patterns) et des composants
- Intégrer les résultats de la pratique (l'expérience)



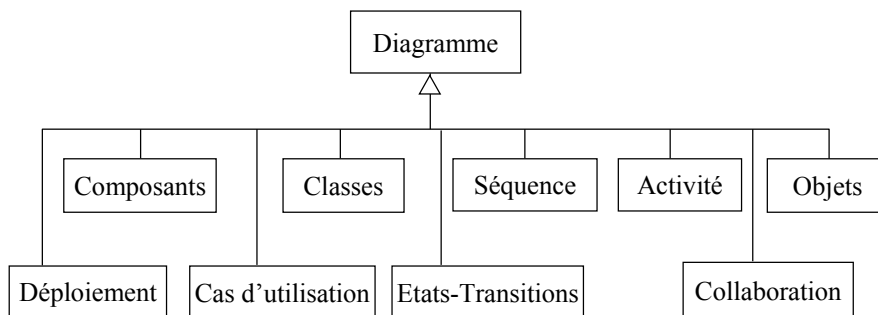


## Ce que UML offre

- La notation simplifiée pour les modèles orientés-objet
- Nouvelles sémantiques
- Les diagrammes standards



## Diagrammes d'UML







## Diagrammes d'UML

- Découverte des besoins :
  - **Diagramme de cas d'utilisation** : fonctions du système du point de vue de l'utilisateur
- Analyse :
  - **Diagramme de classes** : structure statique en termes de classes et de relations
  - **Diagramme d'objets** : objets et leurs relations. Diagrammes de collaboration simplifiés, sans représentation des envois de message
  - **Diagramme de séquence** : représentation temporelle des objets et de leurs interactions



## Diagrammes d'UML

- Analyse (suite) :
  - **Diagramme de collaboration** : représentation spatiale des objets, des liens et des interactions
  - **Diagramme d'états-transitions** : comportement d'une classe en terme d'états (Statecharts)
  - **Diagramme d'activités** : comportement d'une opération en termes d'actions
- Conception :
  - **Diagramme de déploiement** : déploiement des composants sur les dispositifs matériels
  - **Diagrammes de composants** : composants physiques d'une application





## UML – Diagramme de classes

---

Un diagramme de classes exprime la structure statique d'un système, en termes de *classes* et de *relations* entre ces classes



## Concepts de base

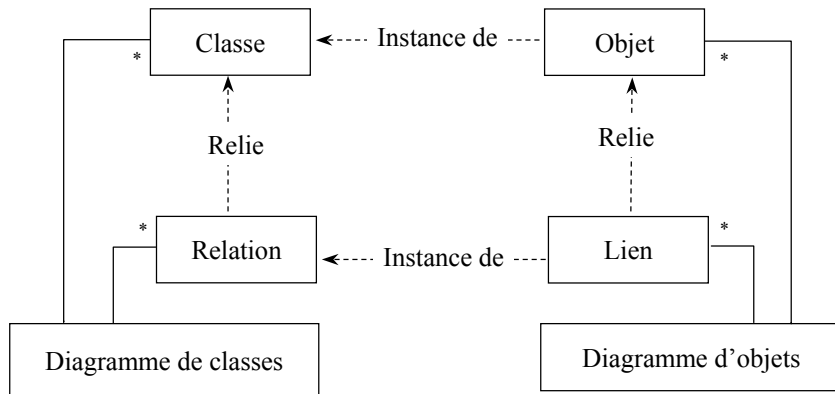
---

- **Objet** : Un objet est une entité du monde réel ou virtuel qui contient de l'*information* et qui a un certain *comportement*
  - 3 caractéristiques fondamentales d'un objet
    - Un état
    - Un comportement
    - Une identité
- **Classe d'objets** : Regroupement des objets de même type
  - Identification et description des caractéristiques communes à un ensemble d'objets
  - Domaine de définition d'un ensemble d'objets. Chaque objet appartient à une classe





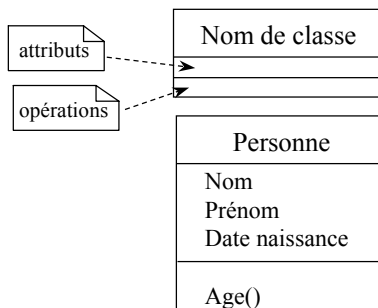
## Diagramme de classes



## Classe

Une **classe** est une description abstraite d'un ensemble d'objets ayant :

- des propriétés similaires,
- un comportement commun,
- des relations communes avec d'autres objets et
- des sémantiques communes



Nom de classe

Personne

Poste de travail

Département





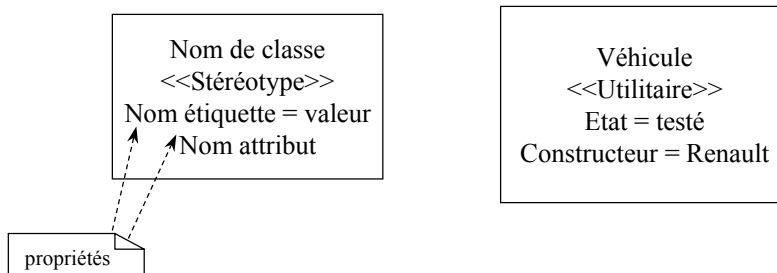
## Stéréotypes et propriétés de classe

- Les stéréotypes :
  - <<signal>>, une occurrence remarquable qui déclenche une transaction dans un automate
  - <<interface>>, une description des opérations visibles
  - <<méta classe>>, la classe d'une classe
  - <<utilitaire>>, une classe réduite au concept de module et qui ne peut être instanciée
- Les propriétés désignent toutes les valeurs attachées à un élément de modélisation, comme les attributs, les associations et les étiquettes. Une étiquette est une paire (attribut, valeur) définie par l'utilisateur



## Stéréotypes et propriétés de classe

Exemple :

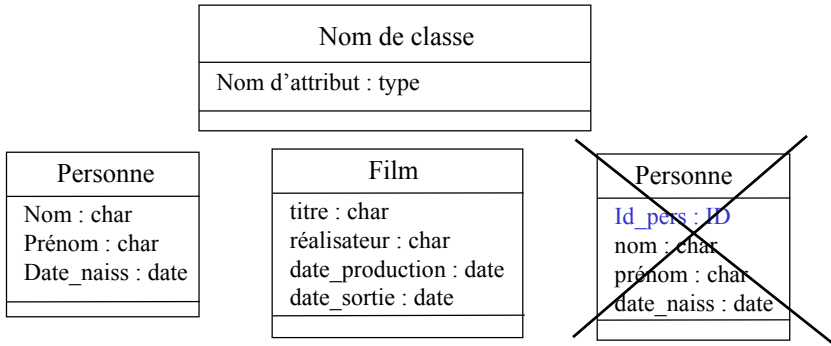






## Attributs de classe

Un **attribut** définit une propriété des objets d'une classe

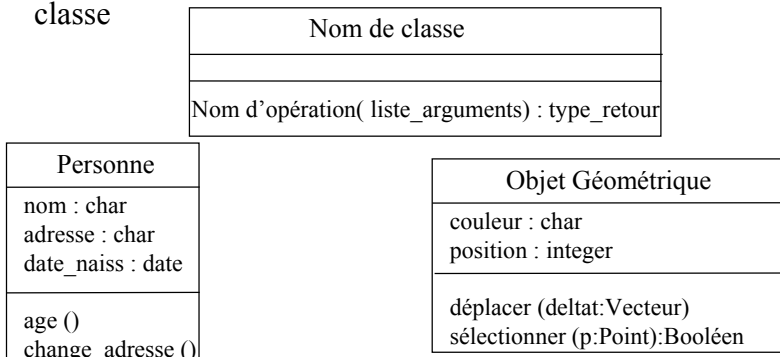


- Chaque nom attribut est unique dans une classe
- Identification est fournie par le système, elle n'est pas à considérer dans le modèle objet



## Opérations de classe

Une **opération** définit une fonction appliquée à des objets d'une classe



La syntaxe pour la description des opérations :

**Nom\_opération** (Nom\_Argument : Type\_Argument =  
Valeur\_Par\_Défaut, ...) : Type\_Retourné

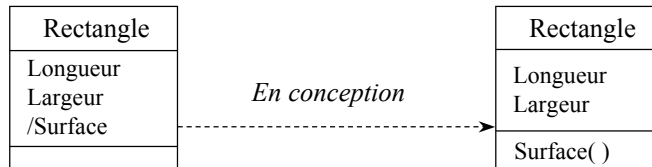




## Attributs dérivés

Les **attributs dérivés** offrent une solution pour allouer des propriétés à des classes, tout en indiquant clairement que ces propriétés sont dérivées d'autres propriétés déjà allouées

Exemple:



## Visibilité des attributs et des opérations

- 3 niveaux de visibilité pour les attributs et les opérations:
  - public (+) qui rend l'élément visible à tous les clients de la classe
  - protégé (#) qui rend l'élément visible aux sous-classes de la classe
  - privé (-) qui rend l'élément visible à la classe seule
- Certains attributs et certaines opérations peuvent être visibles globalement, dans toute la portée lexicale de la classe
- Variables et opérations de classe





## Visibilité des attributs et des opérations

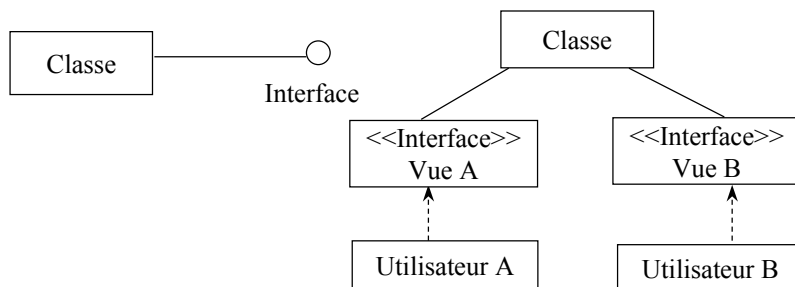
Exemple :

A
+ Attribut public # Attribut protégé - Attribut privé <u>Attribut de classe</u>
+ Opération publique( ) # Opération protégée( ) - Opération privée( ) <u>Opération de classe( )</u>



## Interface de classe

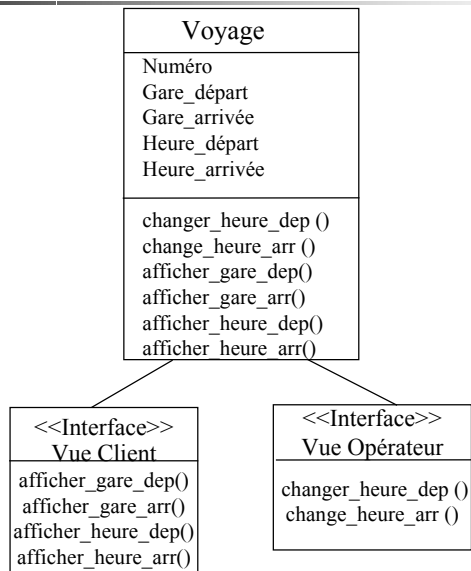
- Une **interface** utilise un type pour décrire le comportement visible d'une classe, d'un composant, ou d'un paquetage
- C'est un stéréotype d'un type
- Peut aussi se représenter au moyen de classes stéréotypées
- Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par un ou plusieurs éléments





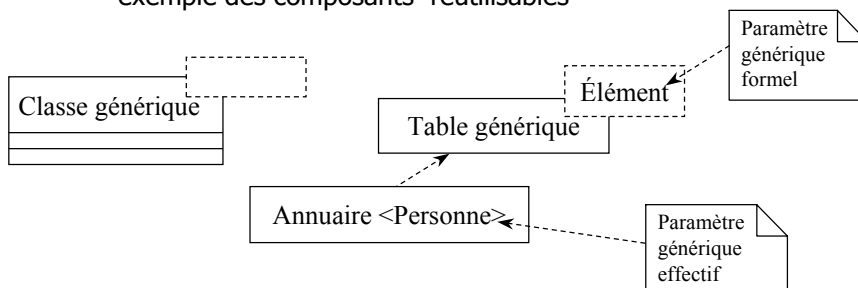


## Interface de classe



## Classe paramétrable

- Une **classe paramétrable** est un modèle de classes
- Une classe réelle est une instance d'une classe paramétrable
- Les classes paramétrables permettent de construire des collections universelles, typées par les paramètres effectifs
- Ce type de classes n'apparaît généralement pas en analyse
- Surtout utilisées en conception détaillée, pour incorporer par exemple des composants réutilisables

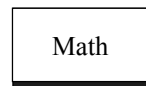
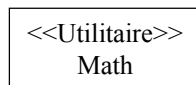






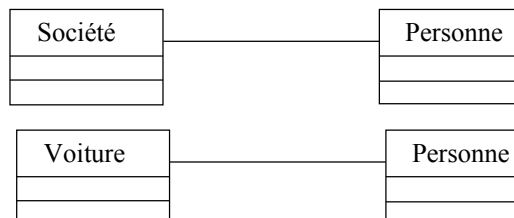
## Classe utilitaire

- Une **classe utilitaire** permet de regrouper des éléments au sein d'un module, sans pour autant vouloir construire une classe complète
- Les classes utilitaires ne peuvent pas être instanciées car elles ne sont pas des types de données
- Le stéréotype <<Utilitaire>> spécialise les classes en classes utilitaires
- Ce type de classes n'apparaît généralement pas en analyse



## Association

- Une **association** représente une relation structurelle entre classes d'objets
- Une association symbolise une information dont la durée de vie n'est pas négligeable par rapport à la dynamique générale des objets instances des classes associées

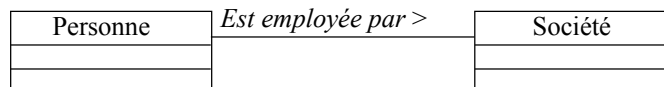
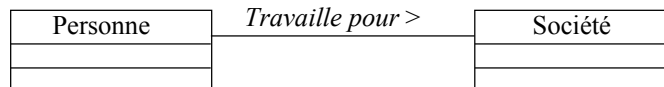
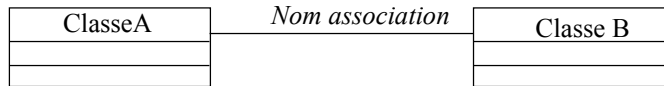






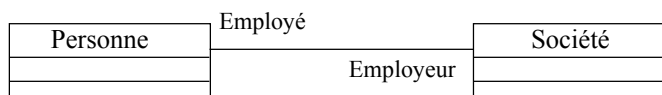
## Noms des associations

- Le nom d'une association apparaît en Italique, au milieu de la ligne qui symbolise l'association
- L'usage recommande de nommer les associations par une forme verbale, soit active, soit passive



## Noms des rôles

- Chaque association binaire possède 2 rôles
- Le rôle décrit comment une classe voit une autre classe au travers d'une association
- Les noms des associations et les noms des rôles ne sont pas exclusifs l'un de l'autre

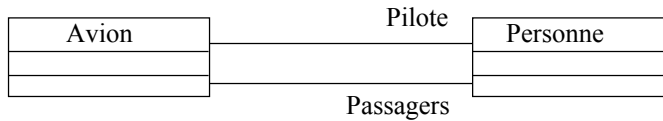




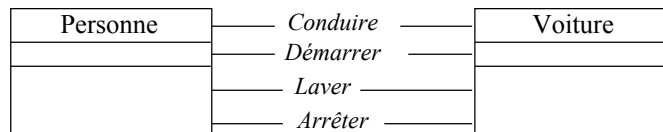


## Noms des rôles

- Les noms des rôles prennent tout son intérêt lorsque plusieurs associations relient 2 classes. Chaque association exprime un concept distinct



- La présence d'un grand nombre d'associations entre 2 classes peut être suspecte. Signe d'une mauvaise décomposition



## Multiplicité des associations

- La **multiplicité** est une information portée par le rôle, qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe

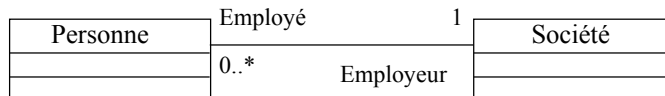
1	Un et un seul
0 .. 1	Zéro ou un
M .. N	De M à N (entiers naturels)
*	De zéro à plusieurs
0 .. *	De zéro à plusieurs
1 .. *	De un à plusieurs





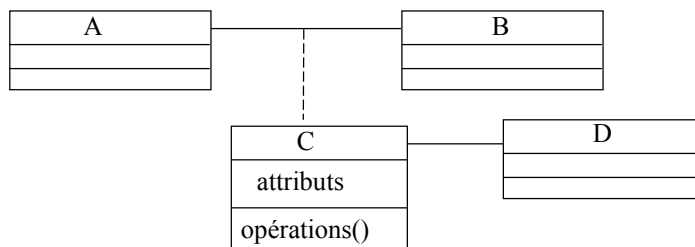
## Multiplicité des associations

- Une valeur de multiplicité supérieure à 1 implique une collection d'objets
- Les valeurs de multiplicité expriment des contraintes liées au domaine d'application, valables durant l'existence des objets
- La détermination de valeurs de multiplicité optimales est très importante



## Classe-association

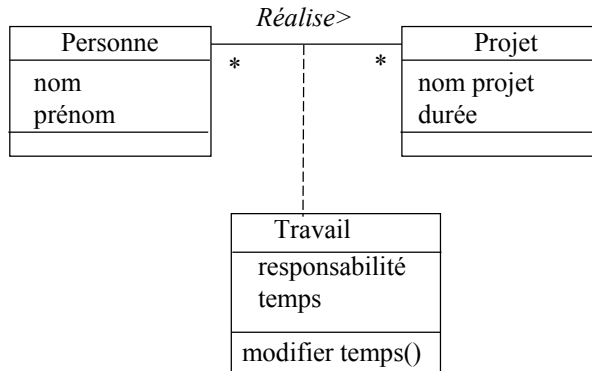
- Une association peut être représentée par une classe pour ajouter, par exemple, des attributs et des opérations dans l'association
- Nommée, **classe associative** ou **classe-association**, c'est une classe comme les autres. La notation utilise une ligne pointillée pour attacher une classe à une association







## Classe association



## Placement des attributs selon les valeurs de multiplicité

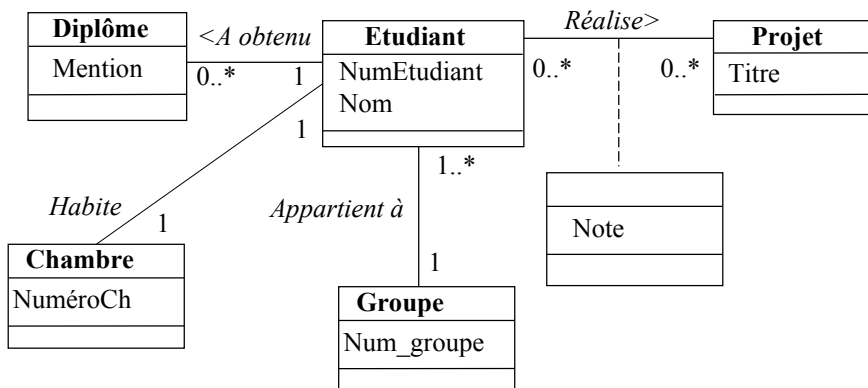
- La réification des associations prend tout son intérêt pour les associations **N vers N**
- Pour les associations **1 vers 1**, les attributs de l'association peuvent toujours être déplacés dans une des classes qui participent à l'association
- Pour les associations **1 vers N**, le déplacement est généralement possible vers la classe côté N; il est fréquent de promouvoir l'association au rang de classe pour augmenter la lisibilité ou en raison de la présence d'associations vers d'autres classes



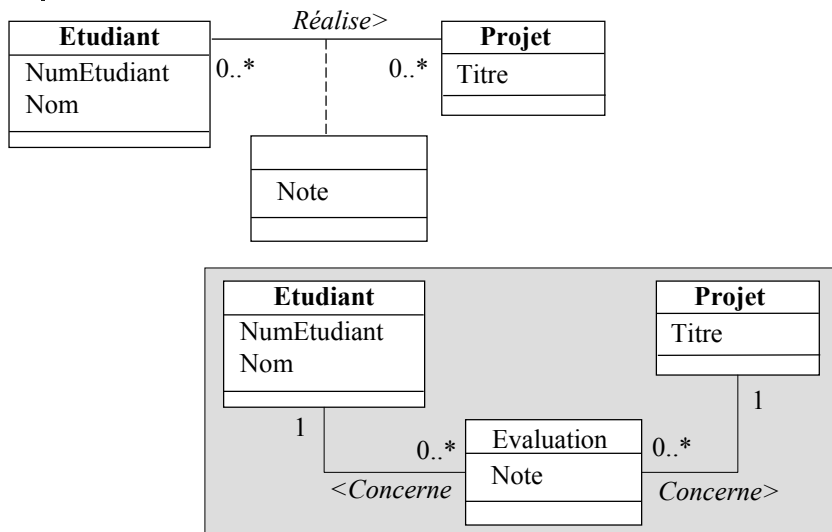


## Placement des attributs selon les valeurs de multiplicité

### Exemple



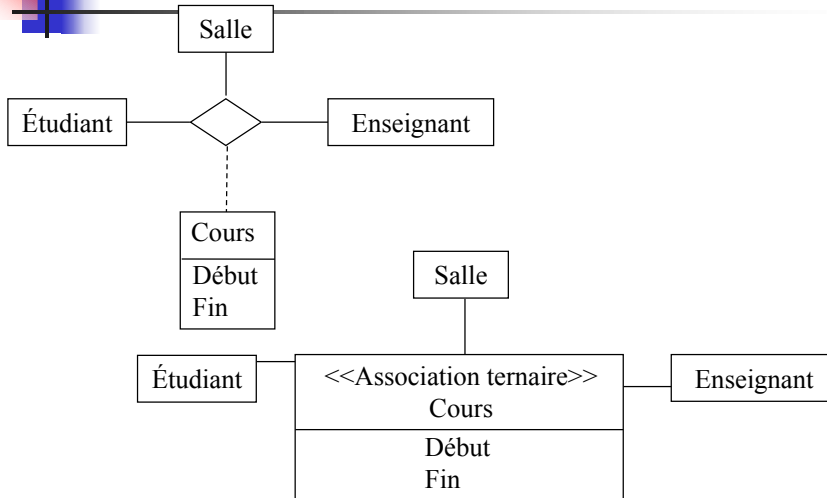
## Transformation d'une association *N vers N*







## Association n-aire

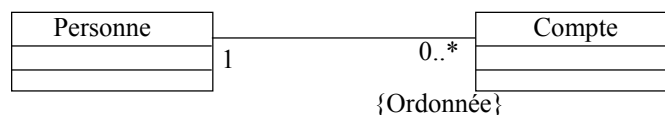


Représentation d'une association ternaire au moyen d'une classe avec stéréotype



## Contraintes sur les associations

- Les contraintes se représentent dans les diagrammes par des expressions placées entre accolades
- La contrainte **{ordonnée}** peut être placée sur le rôle pour spécifier qu'une relation d'ordre décrit les objets placés dans la collection

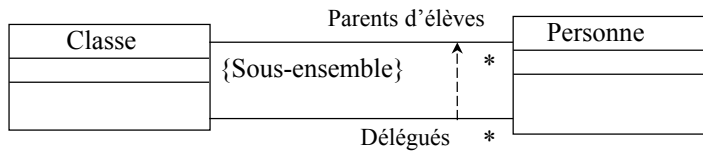




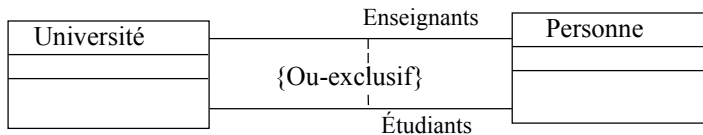


## Contraintes sur les associations

- La contrainte **{Sous-ensemble}** indique qu'une collection est incluse dans une autre collection



- La contrainte **{Ou-exclusif}** précise que, pour un objet donné, une seule association parmi un groupe d'associations est valide

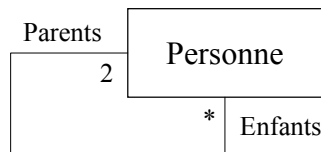


\*



## Association réflexive

- Association réflexive** – une association entre les objets de la même classe
- Les noms de rôles sont essentiels à la clarté du diagramme

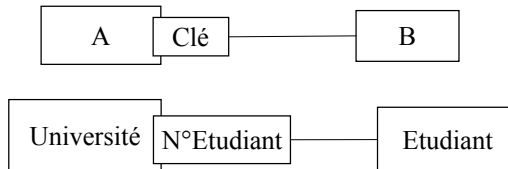






## Restriction des associations

- Une **restriction** (**qualification** pour UML) d'une association consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association
- Une restriction est réalisée au moyen d'une **clé**, ensemble d'attributs particuliers, et utilisée avec un objet de la classe de départ
- Chaque instance de la classe A accompagnée de la valeur de la clé, identifie un sous-ensemble des instances de B qui participent à l'association



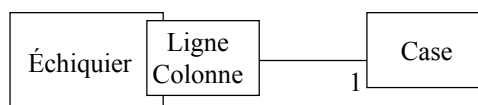
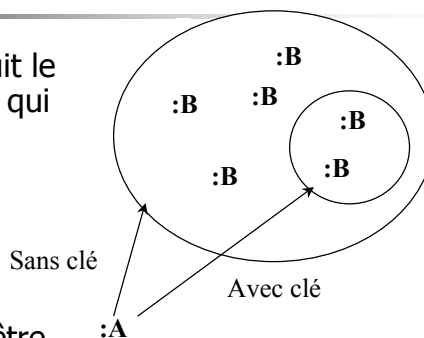
© J.Ralyté, Université de Genève

39



## Restriction des associations

- Une restriction réduit le nombre d'instances qui participent à une association
- La restriction peut être opérée en combinant les valeurs des différents attributs qui forment la clé



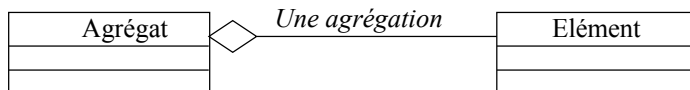
© J.Ralyté, Université de Genève

40



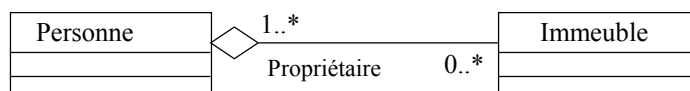
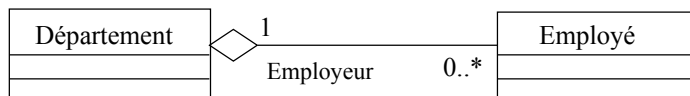
## Agrégation

- Une **agrégation** représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité
- Les critères suivants impliquent une agrégation :
  - une classe fait partie d'une autre classe
  - les valeurs d'attributs d'une classe se propagent dans les valeurs d'attributs d'une autre classe
  - une action sur une classe implique une action sur une autre classe
  - les objets d'une classe sont subordonnés aux objets d'une autre classe



## Agrégation

- L'agrégation peut être simple ou multiple







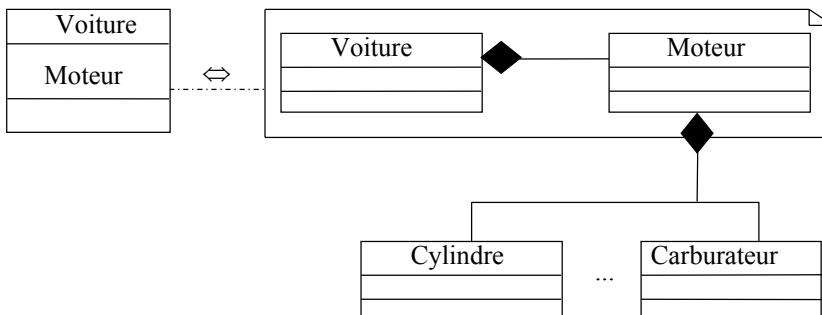
## Composition

- Une **composition** représente une association non symétrique dans laquelle une des extrémités joue un rôle prédominant par rapport à l'autre extrémité
- **Le composant est physiquement contenu dans le composé**
- La composition implique une contrainte sur la valeur de la multiplicité du côté de l'agrégat : elle ne peut prendre que les valeurs 0 ou 1
- La valeur 0 du côté du composant correspond à un attribut non renseigné



## Composition

- La composition et les attributs sont sémantiquement équivalents
- La notation par composition s'emploie dans un diagramme de classes lorsqu'un attribut participe à d'autres relations dans le modèle





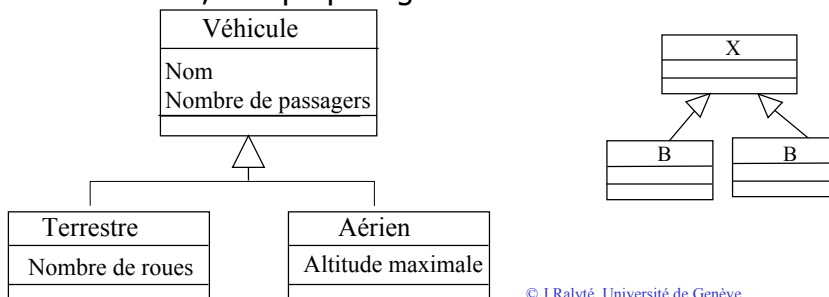
## Navigation

- Une **navigation** est une association unidirectionnelle
  - Les associations décrivent le réseau de relations structurelles entre les classes
  - Par défaut, les associations sont navigables dans les 2 directions
  - Si seule une direction est utile, ceci se représente par une flèche portée par le rôle vers lequel la navigation est possible



## Généralisation

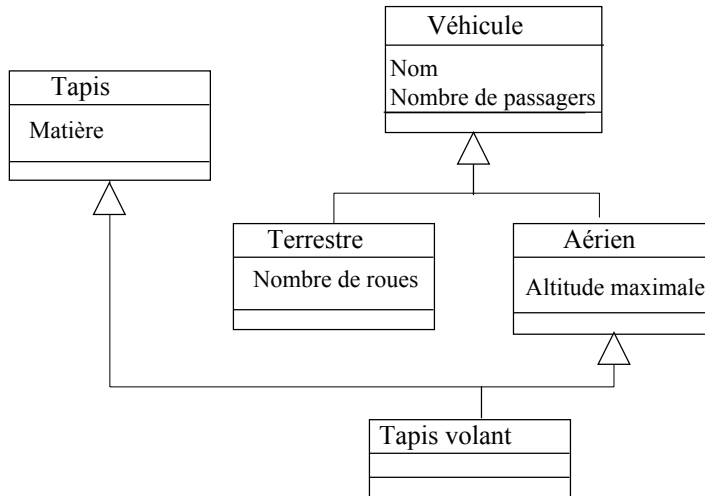
- Une relation de **généralisation** permet d'extraire les propriétés et les opérations communes à plusieurs classes et de les définir dans une classe générique
- La relation de généralisation signifie *est un* ou *est une sorte de*
- La généralisation s'applique principalement aux classes, aux paquetages et aux cas d'utilisation





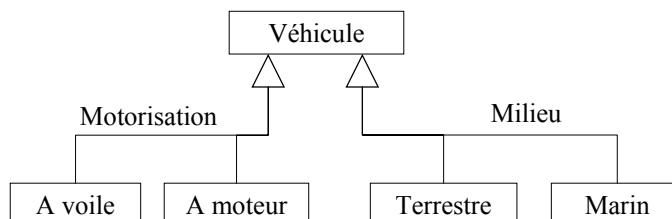


## Généralisation multiple



## Généralisation

- Une classe peut être spécialisée selon plusieurs critères simultanément
- Différentes contraintes peuvent être appliquées aux relations de généralisation
- Par défaut, la généralisation symbolise une décomposition exclusive

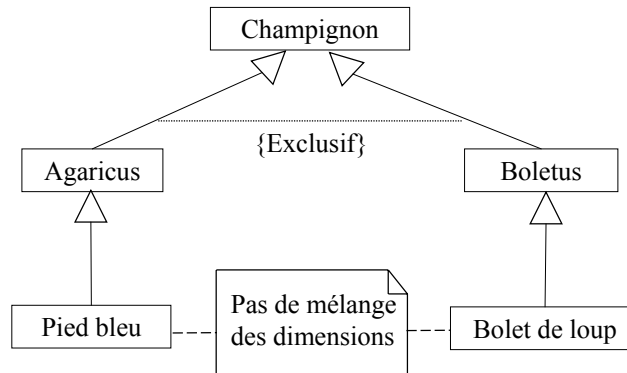






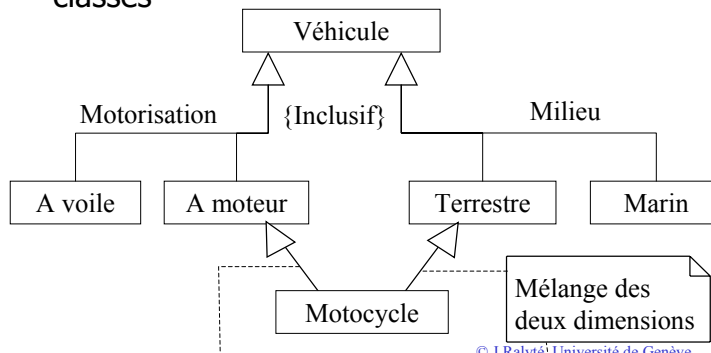
## Contraintes de généralisation

- La contrainte **{Disjoint}** ou **{Exclusif}** indique qu'une classe descendante d'une classe A peut être descendante d'une seule sous-classe de A



## Contraintes de généralisation

- La contrainte **{Chevauchement}** ou **{Inclusif}** indique qu'une classe descendante d'une classe A appartient au produit cartésien des sous-classes de la classe A. Un objet concret est alors construit à partir d'une classe obtenue par mélange de plusieurs super-classes

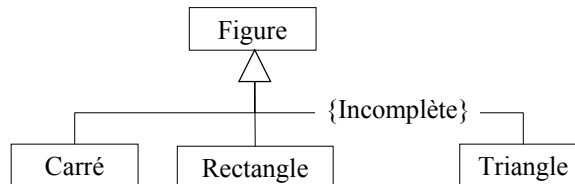






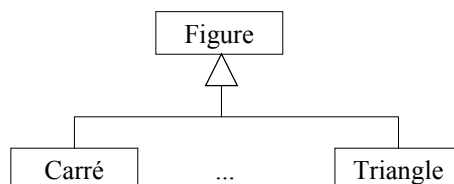
## Contraintes de généralisation

- La contrainte **{Complète}** indique que la généralisation est terminée et qu'il n'est pas possible de rajouter des sous-classes.
- Inversement, la contrainte **{Incomplète}** désigne une généralisation extensible



## Contraintes de généralisation

- Ne pas confondre, la contrainte **{Incomplète}** avec une vue incomplète
  - Une contrainte véhicule un contenu sémantique qui affecte le modèle
  - Une vue incomplète ne signifie pas que le modèle est incomplet. Une vue partielle se symbolise par la présence de points d'élision qui se substituent aux éléments de modélisation

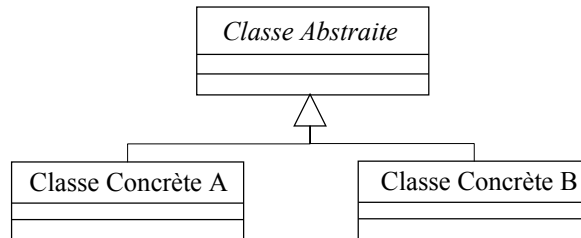






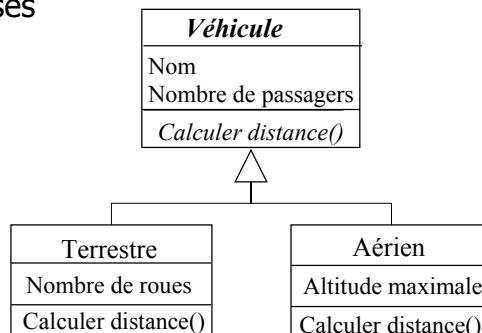
## Classe abstraite

- Une **class abstraite** définit les propriétés et le comportement commun à plusieurs sous-classes **concrètes**, mais elle ne peut pas exister sans ses sous-classes concrètes
- Une classe abstraite n'a pas d'instances directes, c'est une spécification plus abstraite pour des objets instances de ses sous-classes



## Classe abstraite

- Les classes abstraites forment une base pour les logiciels extensibles, elles sont utilisées en conception/programmation orienté-objet
- Une classe abstraite peut avoir des **opérations abstraites**: le corps d'une opération abstraite n'est pas défini, il doit être défini explicitement dans les sous-classes





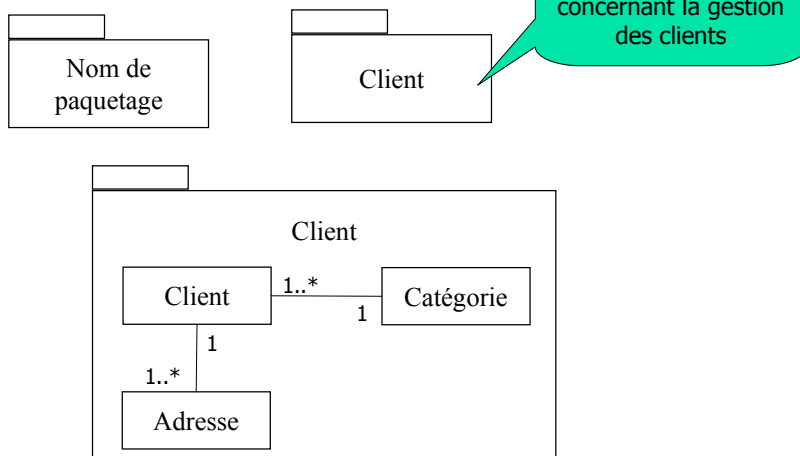


## Paquetage

- Un **paquetage** – un mécanisme général pour la partition des modèles en sous-modèles et le regroupement des éléments de modélisation
- Un paquetage correspond à un sous-ensemble du modèle et il contient, selon le modèle, des classes, des objets, des relations, des composants ou des nœuds, ainsi que les diagrammes associés
- Un regroupement d'éléments selon un critère purement logique



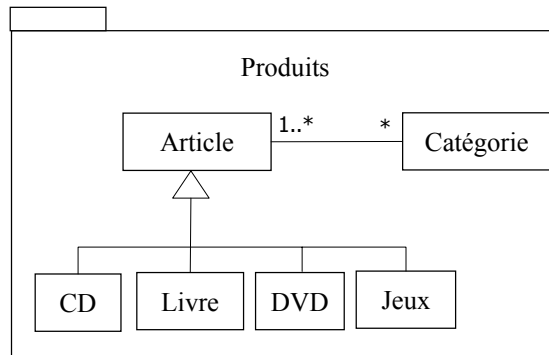
## Représentation de paquetages





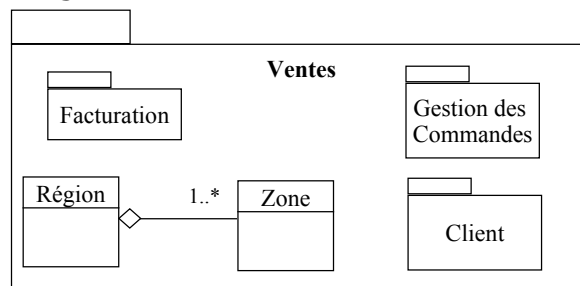


## Représentation de paquetages



## Propriétés de paquetages

- Un paquetage peut contenir d'autres paquetages
- Chaque élément du modèle appartient à un paquetage. Le paquetage de plus haut niveau est le paquetage racine représentant le modèle entier
- Le système complet est une hiérarchie de paquetages

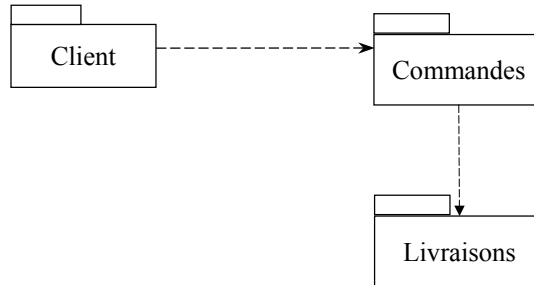






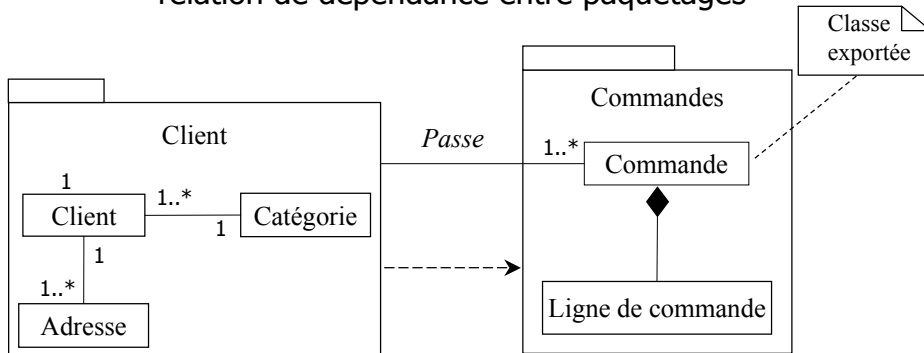
## Relations entre paquetages

- La forme générale du système est exprimée par la hiérarchie de paquetages et par le réseau de relations de dépendance entre paquetages



## Relations entre paquetages

- Une classe contenue par un paquetage peut également apparaître dans un autre paquetage sous la forme d'un élément importé, au travers d'une relation de dépendance entre paquetages

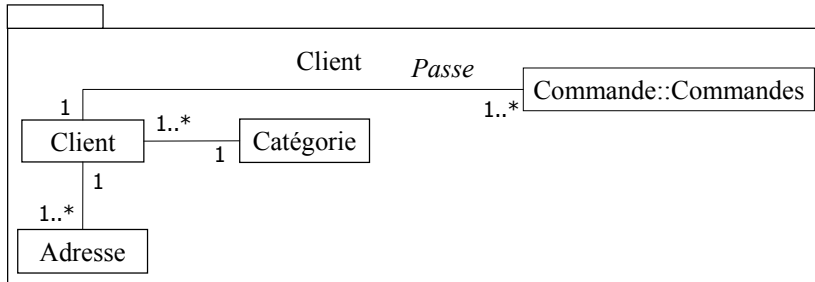






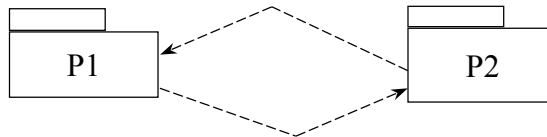
## Relations entre paquetages

- Une classe contenue par un paquetage peut également apparaître dans un autre paquetage sous la forme d'un élément importé, au travers d'une relation de dépendance entre paquetages



## Relations entre paquetages

- Eviter les dépendances cycliques entre paquetages



- Eviter les dépendances circulaires transitives entre paquetages

