# OBJET
# Les méthodes

# Nouvelles méthodes

## Motivation

1) Prendre en compte des applications de plus en plus complexes, tant dans le monde de la gestion (introduction du multimédia) que des applications techniques (CAO, simulation, cartographie ...) qui se caractérisent par :
   - ? le besoin de représentation d'objets complexes et volumineux
   - ? l'évolution fréquente de la structure et du comportement de ces objets
   - ? la gestion des exceptions.

2) Diminuer les coûts de développement et de maintenance (par la réutilisabilité).

## Les 3 étapes de l'Orienté Objet

**OOA**

**L'analyse orientée objet ("Analysis modelling")** donne une description du domaine du problème, découvre les entités majeures du domaine.

**OOD**

**La conception objet ("Design modeling")** complète et transforme les spécifications des classes d'analyse, spécifie une solution informatique.

**OOP**

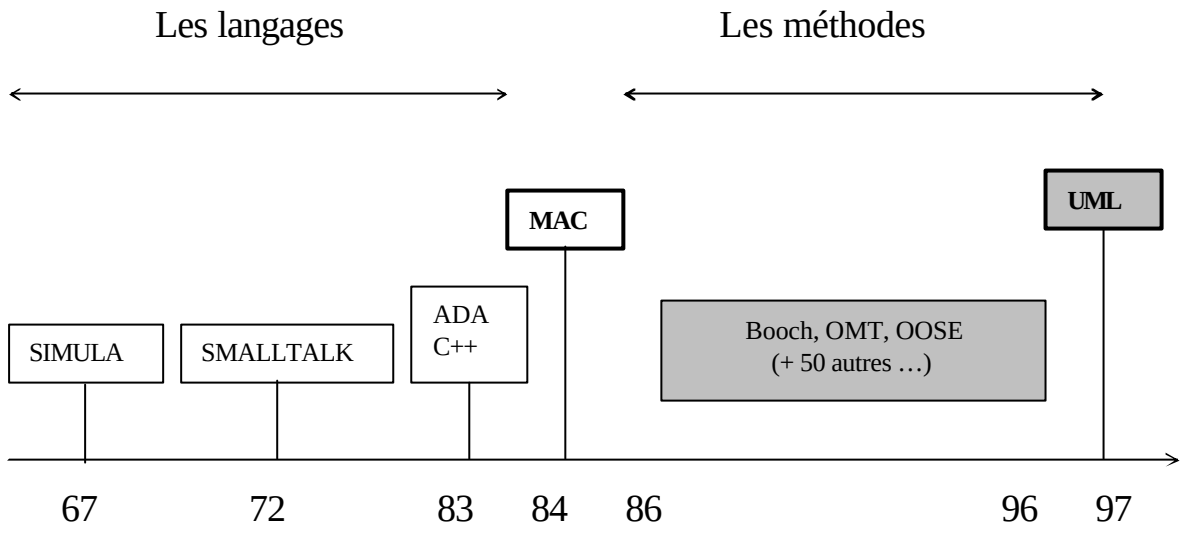
**La programmation objet ("Implémentation modelling")** conçoit les modules logiciels en fonction des logiciel et matériel choisis.

Ces définitions sont celles de l'OMG (*Object Management Group*)

Organisme à but non lucratif fondé en 1989
Plus de 700 entreprises adhérentes
Connu pour la norme CORBA (IDL, IIOP …)

# Historique

## Les grandes dates




Les premiers objets ont été développés pour gérer les interfaces graphiques.

Les suivants doivent être les objets métiers, c'est-à-dire ceux que l'entreprise manipule dans le cadre de son activité.

Depuis 1986, on peut considérer que c'est l'époque des pionniers en ce qui concerne les méthodes OO. On assiste à une profusion de publications.

1997, avec la naissance d'UML peut constituer le début de la période de transition vers la généralisation de cette méthode, qui doit permettre d'abaisser les coûts perpétuels de re-formation et ré-équipement d'outils, à chaque nouveau projet …

# Les pionniers

**Booch, du nom de son auteur : Grady Booch.**

Elle s'appelle aussi OOD pour Object Oriented Design, ce qui indique que la conception est son objectif principal.
Repose sur la dualité graphe de classes / graphe d'objets.
La méthode préconise dans sa démarche de travail de réaliser rapidement un prototype pour valider l'analyse et pour l'affiner.
La faiblesse de la méthode réside dans son graphisme qui donne aux classes et aux objets une forme de *nuage*. C'est poétique mais pas pratique du tout à utiliser.
La méthode Booch est intéressante pour la pureté de la mise en œuvre des concepts de l'Orienté Objet.

**OMT, de James Rumbaugh.**

Elle repose sur les trois visions d'un système:

la *vue statique* qui est celle des classes et de leurs structure,
la *vue dynamique* donne la représentation des états, des messages, des actions et des événements à travers un graphe états / transitions,
la *vue fonctionnelle* montre la circulation des données, ce qui est surtout utile dans le cas de système distribué.

La faiblesse principale de cette méthode est liée aux difficultés d'établissement des cohérences entre les trois vues.
La méthode est bien adaptée à l'analyse et à la conception de systèmes temps réels distribués.
Le livre de James Rumbaugh sur OMT est assez facile à comprendre.

**Objectory ou OOSE du suédois Ivar Jacobson.**

Elle a été mise au point chez Ericsson.
Cette méthode est plus fonctionnelle que réellement Orientée Objet.
Elle bénéficie d'un manuel gros, mal traduit, mais documenté et instructif.
Elle définit 3 types d'objets : entités, contrôles, interfaces.
Elle est la première à prendre en compte la notion de *cas d'utilisation*.
Elle a bien montré l'intérêt de systématiser les graphes d'interactions.

**OOA / OOD de Schlaer & Mellor.**

Elle conduit à la description la plus détaillée et la plus précise.
L'implémentation est quasi immédiate. A l'usage, elle a souffert de sa complexité et surtout d'avoir adopté des représentations différentes pour la spécification et la conception. Elle est un peu passée de mode.

**OOA de Coad & Yourdon**

Elle repose sur un graphe de classe très pratique qui est également utilisé pour représenter les envois de message correspondant à chaque cas d'utilisation de service complexe. Simple à comprendre, expliquée dans un livre assez facile à étudier, c'est une méthode bien adaptée aux petits projets.

**Objecteering ou Classe-Relation.**

Méthode d'une société française Softeam ou travaille P. Desfray.
Par rapport à des méthodes comme OMT ou Booch, elle ajoute la notion d'*hypergénéricité*. L'idée est intéressante car l'approche Orientée Objet montre que tous les systèmes informatiques semblent bien être réductibles à un certain nombre de types de base qui se conjuguent à l'infini.
Elle est intimement liée à un AGL.
Si elle débouche aux USA, elle deviendra une option crédible.

**OOM ou MERISE Orienté Objet.**

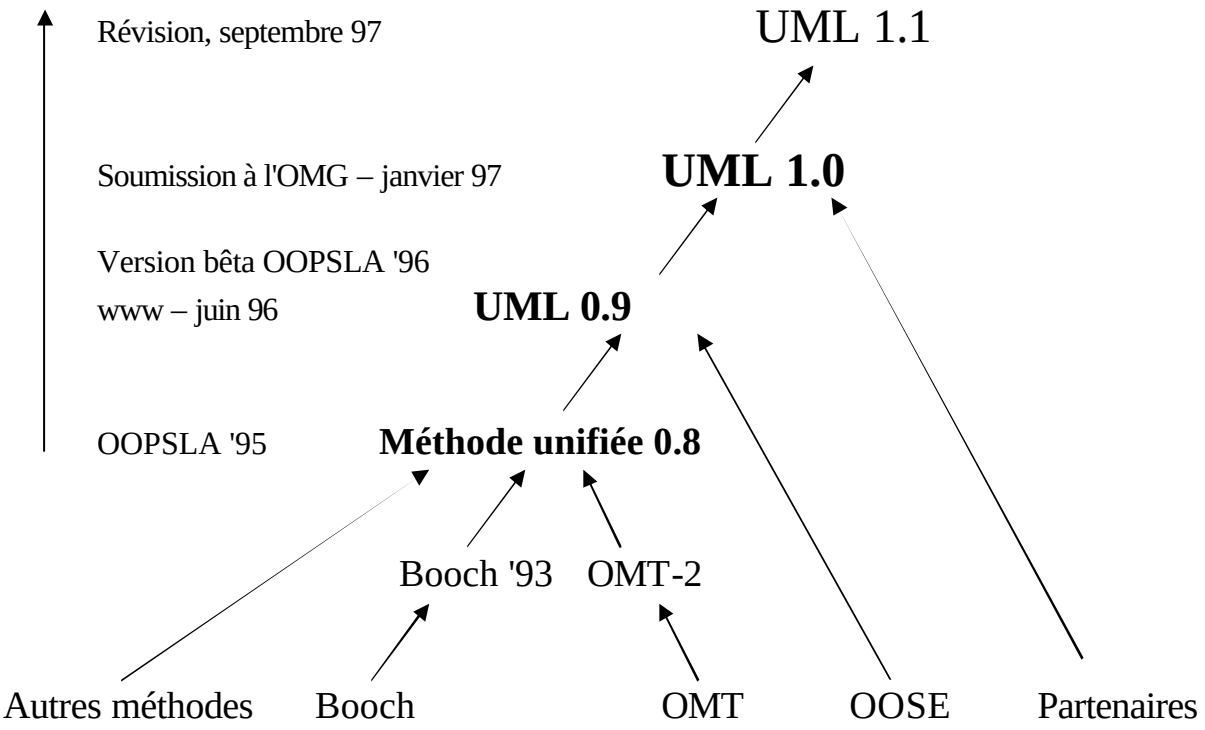
Rochfeld et Bouzeghoub ont développé un lifting de Merise qui intègre un noyau Orienté Objet. A vouloir rajouter les graphes propres à l'Orienté Objet aux MCD et différentes représentations de Merise, il est difficile d'imaginer que le résultat soit léger et souple.

**Objets Naturels**

Méthode développée chez Concis par Paul-André BRES.
Elle présente un apport intéressant sur le modèle des données de Merise.
Le modèle des Objets Naturels a pour fonction de formaliser les comportements d'objets qui sont construits à partir du modèle conceptuel des données de type Entité / Relation Etendu afin de déduire automatiquement les transactions interactives qui en découlent.
Elle offre un AGL qui génère les applications dans l'environnement de développement de GUPTA Technologies (SQL Windows, Team Windows).

# Histoire récente

Les trois locomotives en matière de méthodes Orientées Objets : Booch, Rumbaugh et Jacobson, ont travaillés à une convergence de leurs méthodes au sein de la société américaine RATIONAL.
Leurs travaux ont aboutis à la naissance d'UML (Unified Modeling Language).




Les créateurs d'UML sont arrivés à un **consensus en terme de modélisation** mais pas encore en temps que démarche.

La notation UML peut ainsi se substituer, sans perte d'information, aux notations des méthodes de Booch, OMT ou OOSE.

Par ailleurs de nombreuses entreprises ont annoncé leur intention de soutenir UML: Microsoft, IBM, ORACLE, Unisys, softeam, Hewlett-Packard, et bien d'autres...

# Les diagrammes

Modèles statiques

**Diagramme de classes et diagrammes d'objets.** Il décrivent la structure statique des objets et de leurs relations : agrégation, héritage ou association ; ainsi que les attributs et les opérations qui caractérisent chaque classe d'objets. Proposés par *OMT* et *Booch*.

Modèles d'usage

**Diagramme de cas d'utilisation (ou Use Cases).**
Ces cas d'utilisation constituent l'approche principale de la méthode *OOSE* de *Jacobson*. Un cas d'utilisation représente une partie du comportement du système par rapport à un acteur externe. Les cas d'utilisation donnent lieu à l'élaboration des scénarios.

Modèles dynamiques

**Diagramme d'états** (STD : State Transition Diagram de *Harel*). Décrit le comportement des objets les plus importants ou les plus représentatifs.

Deux diagrammes d'interaction:

**Diagramme de séquence** ou Event trace dans *OMT*, encore appelé diagramme de suivi d'événements. Met en évidence l'aspect temporel des interactions entre les classes.

**Diagramme de collaboration** ou Event flow dans *Booch*. Montre les interactions (envoi de messages) entre des objets.

**Diagramme d'activité**. Un peu à part. Il dérive à la fois des statecharts de *Harel*, mais aussi des Work flow diagrammes existants bien avant l'orienté objet.

Modèles architecturaux

**Diagramme de composants**. Montre les éléments physiques et leurs dépendances dans l'environnement de réalisation. (*Booch*).

**Diagramme de déploiement**. Décrit la configuration matérielle d'un système. (*Booch*).

**Conclusion:** UML prend le meilleur de chacune des 3 méthodes.

# Modèles statiques

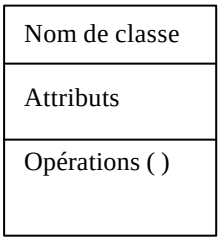
## Diagramme de classe

Il représente les éléments de modélisation statique, tels que des classes, leur contenu et leurs relations. Par contre il ne peut pas décrire les aspects dynamiques.

## Classes

Une *classe* décrit un groupe d'objets ayant des propriétés similaires (attributs), un comportement commun (opérations), des relations communes avec les autres objets ainsi qu'une même sémantique.
Le choix des classes est subjectif, il dépend fortement du contexte de l'application. *Personne, société, animal, processus* et *fenêtre* sont des classes d'objets.

Représentation graphique d'une classe :

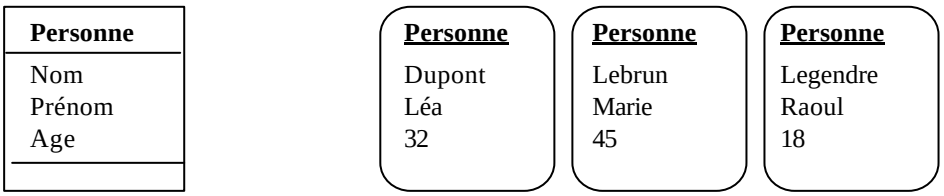



Le premier compartiment contient le nom de la classe, le second les attributs et le dernier les opérations.
Les compartiments peuvent être supprimés pour alléger les diagrammes.
Par défaut, les attributs sont cachés et les opérations sont visibles.

## Objets

Chaque objet a une identité et peut être distingué des autres. *Léa Dupont*, la *Société Simplex*, *Rintintin*, le *processus numéro 7648* et la *fenêtre du haut* sont des objets.

Généralement, les objets (instances de classe) ne sont représentés que pour donner des exemples.




***Classe*** ***Objets***

Les objets et les classes apparaissent souvent comme des noms communs dans les descriptions de problèmes.

**Attributs**

Un *attribut* est une valeur de donnée détenue par les objets d'une classe. Chaque nom d'attribut est unique à l'intérieur d'une classe (par opposition au fait d'être unique parmi toutes les classes). Ainsi la classe *Personne* et la classe *Société* peuvent avoir chacune un attribut *adresse*.

Syntaxe complète : | *visibilité* **nom** : type = valeur-initiale |

Seul le nom est obligatoire en analyse.

Visibilité : marqueur optionnel utilisé en conception

+ public
# protégé
- privé

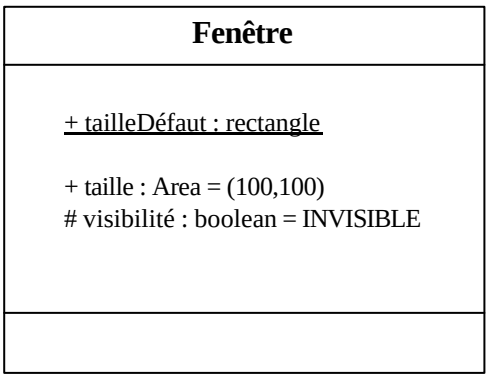
Type : type d'un langage de programmation en conception détaillée.

Valeur-initiale : servira à donner la valeur initiale d'un attribut à la création d'un nouvel objet.

**Attributs de classe**

C'est une sorte de variable globale pour la classe. C'est en fait une valeur unique partagée par tous les objets de la classe et accessible à tous, ce qui évite les recopies et les risques d'incohérence.

Pour le distinguer, il est précédé du symbole « $ » ou il est souligné.



## Opération

Une opération est un service que l'on peut demander à un objet pour réaliser un comportement. Tous les objets d'une classe partagent les mêmes opérations.

En analyse, on parle plutôt *d'opération*. En conception, on parle plutôt de *méthode.* Une méthode est l'algorithme ou la procédure qui produit le résultat de l'opération.

Syntaxe complète : *visibilité* **nom** (liste-paramètres) : type-retour

Seul le nom est obligatoire en analyse.

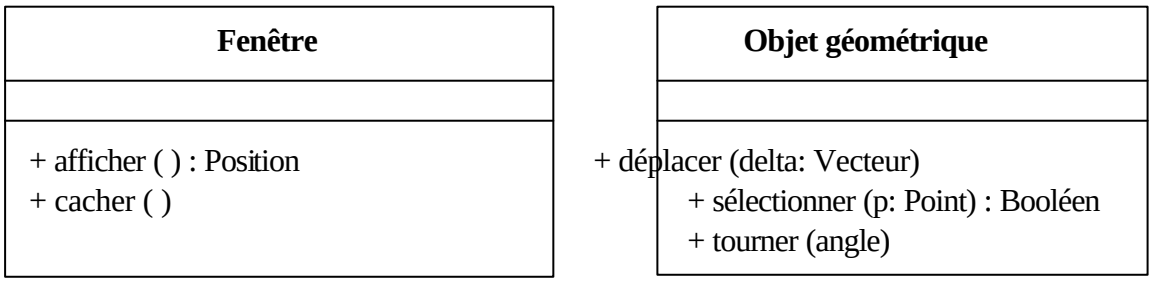
Visibilité : marqueur optionnel utilisé en conception

+ public
# protégé
- privé

Liste-paramètres : liste de paramètres formels, chacun étant spécifié comme suit :

Nom : type = valeur-défaut

Type-retour : optionnel si l'opération ne retourne pas de résultat (*void* du C++).




La *signature* définit le type des arguments d'entrée et de sortie (liste des paramètres nécessaires à l'exécution de l'opération).

## Opération de classe

Opération qui s'applique à la classe elle-même. C'est le cas de l'opération *créer* qui permet de générer une nouvelle instance d'objet.

**Liens et association**

Diagramme d'objets :

Un *lien* est une connexion sémantique entre des objets.

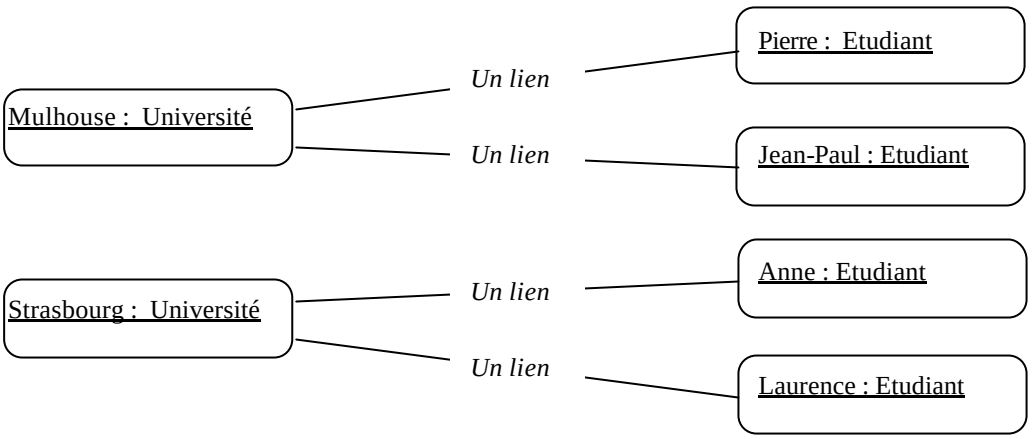
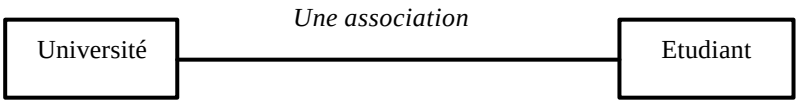
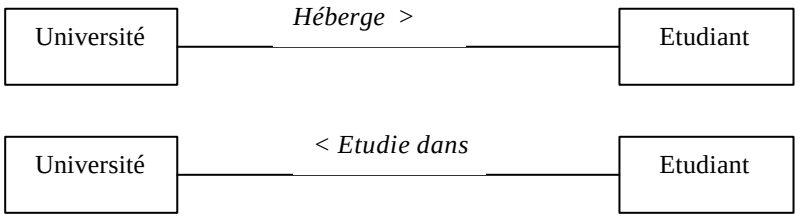



Diagramme de classes :

Un lien est une instance d'association. Une *association* décrit un groupe de liens ayant une structure et une sémantique commune.




Pour améliorer la lisibilité, l'association peut être décorée par une forme verbale active ou passive. Le sens de la lecture peut être précisé.




Remarque:

Les associations sont souvent implémentées dans les langages de programmation sous forme de *pointeurs* d'un objet sur un autre. Un pointeur est un attribut qui contient une référence explicite vers un autre objet ou vers un ensemble d'objets.

**Association ternaire**

Une association peut être binaire, ternaire, d'ordre quatre ou plus.
Dans la pratique, la grande majorité des association sont binaires. On rencontre peu de ternaires, et encore moins (voire pas du tout) d'ordre quatre ou plus.

Une association ternaire est nécessaire quand elle ne peut pas être subdivisée en associations binaires sans perte d'information.

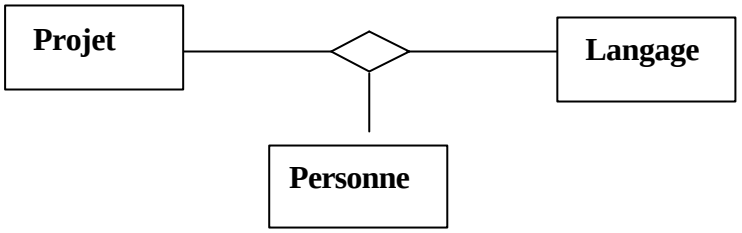



Diagramme de classes

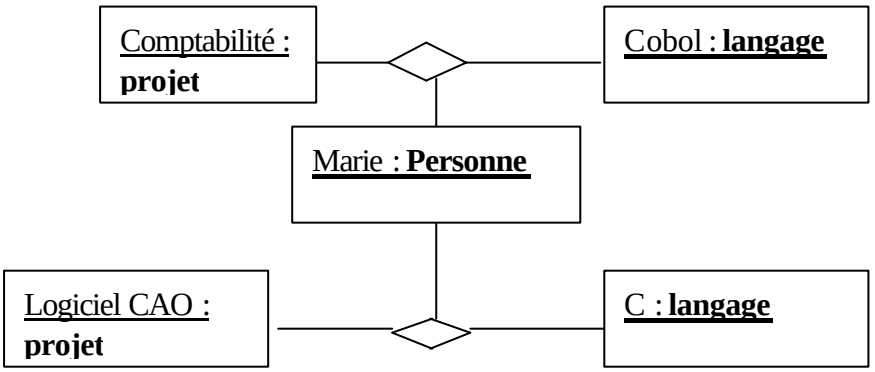
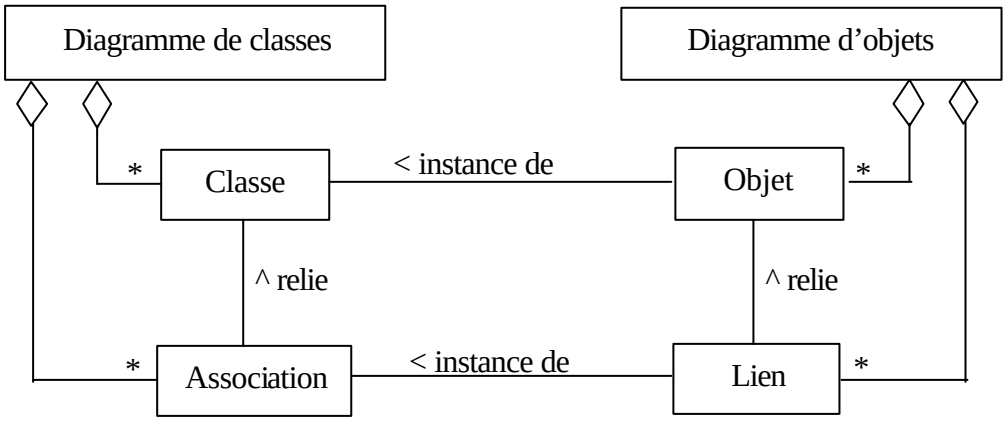



Diagramme d'objets

Extrait du métamodèle UML :

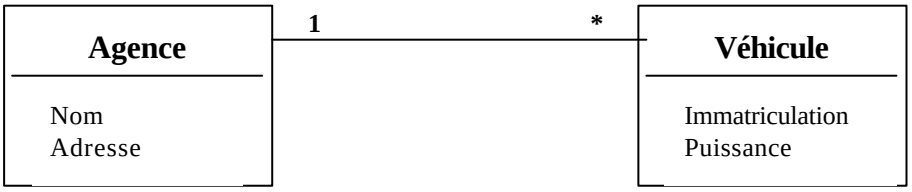


**Multiplicité**
La *multiplicité* précise combien d'instances d'une classe peuvent se rattacher à une seule instance d'une classe donnée.

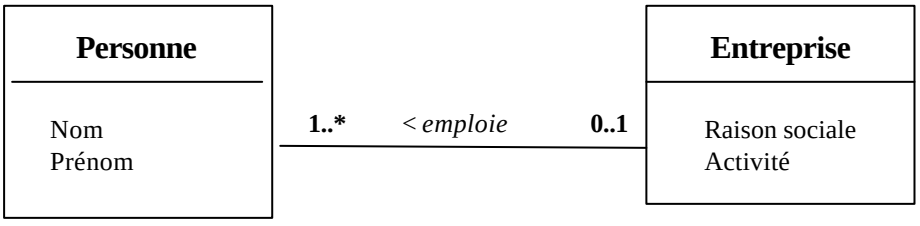
La notation générale adoptée est : **min . . max**

| | |
|---|---|
| **1** | obligatoire |
| **0..1** | optionnel |
| **0..*** ou ***** | quelconque (cas général) |
| **1..*** | au moins 1 |
| **1..5, 10** | entre 1 et 5, ou 10 |

Remarque:
La multiplicité est écrite au bout du trait symbolisant l'association, ce qui est l'inverse de MERISE.



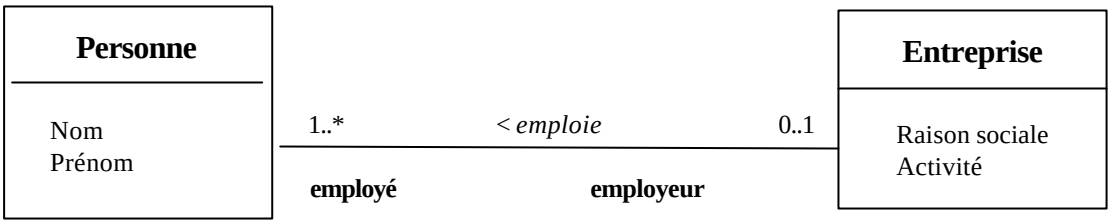

Une agence gère plusieurs véhicules (0 ou n); un véhicule est attaché à une et une seule agence.




Une personne a 0 ou 1 employeur. Une entreprise emploie de 1 à n salariés.

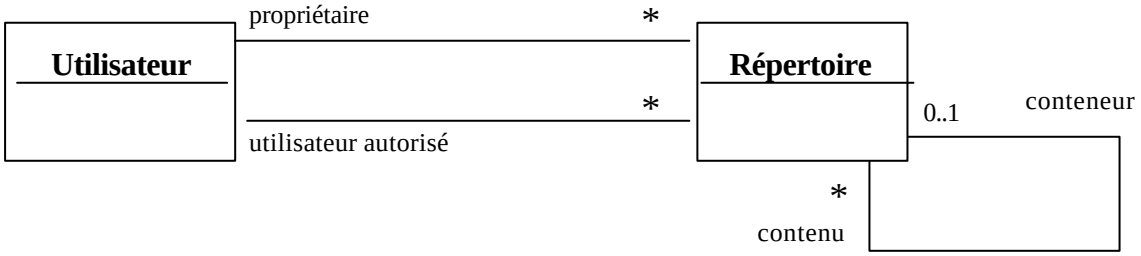
**Rôle**
Un rôle est une extrémité de l'association.

Le *nom de rôle* est un nom qui identifie de façon unique une extrémité de l'association.




L'utilisation des noms de rôle est facultative mais elle prête moins à confusion et est plus commode.
Ces noms de rôle peuvent se substituer au nom de l'association ou lui être ajoutés.

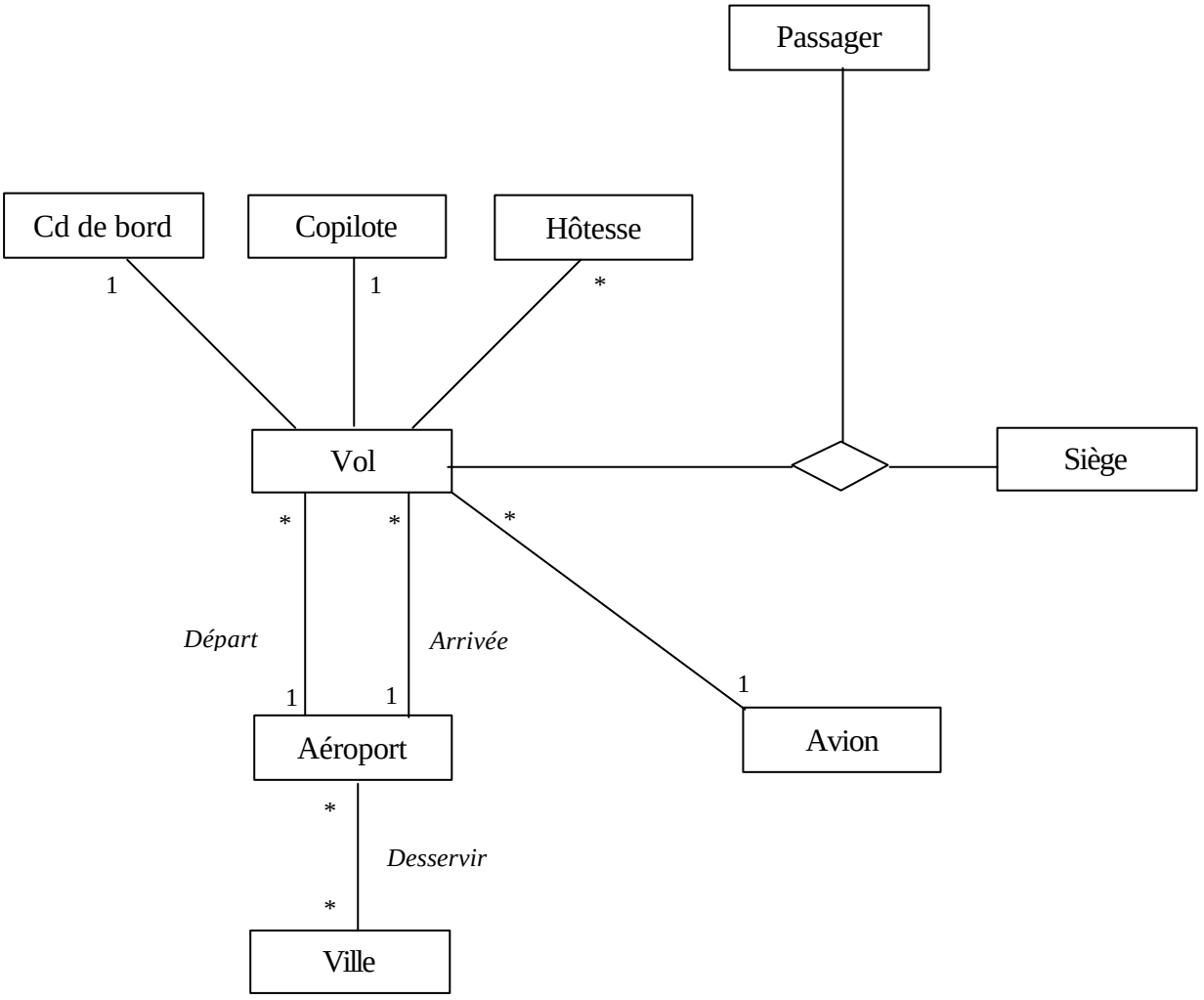
Les noms de rôles sont nécessaires pour des associations entre deux objets d'une même classe.
Les noms de rôles sont aussi utiles pour distinguer deux associations d'un même couple de classes.



## Exercice

Une compagnie aérienne propose des vols à des passagers.
Un vol a un aéroport de départ et un aéroport d'arrivée.
Les aéroports desservent des villes.
Un commandant de bord, un copilote, des hôtesses et un avion sont affectés à chaque vol.
Quand un passager réserve une place sur un vol, on lui attribue un numéro de siège.

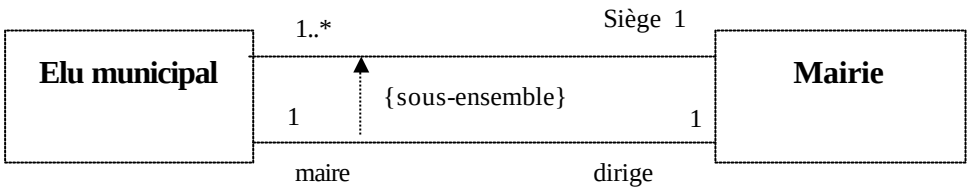
Créer les **classes** et les **associations** nécessaires à la modélisation de ce problème.



**Contrainte**
C'est une relation sémantique entre des éléments du modèle qui spécifie des conditions ou propositions devant rester vraies pour que le modèle soit valide.

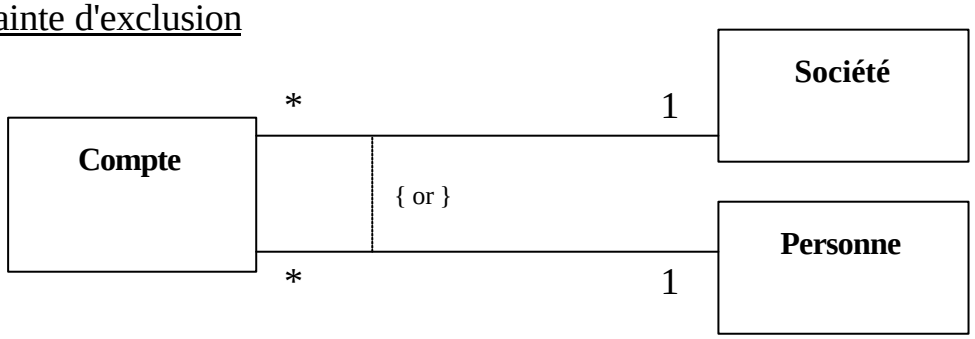
{ contrainte }

## Contrainte d'inclusion



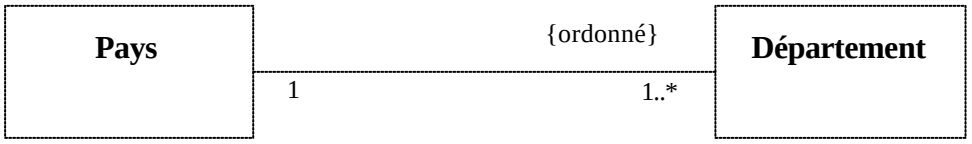

Le maire doit faire partie des élus qui siègent à la mairie.
L'association *dirige* est un sous-ensemble de l'association *siège*.

## Contrainte d'exclusion




La contrainte {or} ou {ou-exclusif} précise qu'une seule association est valide.

## Contrainte d'ordre



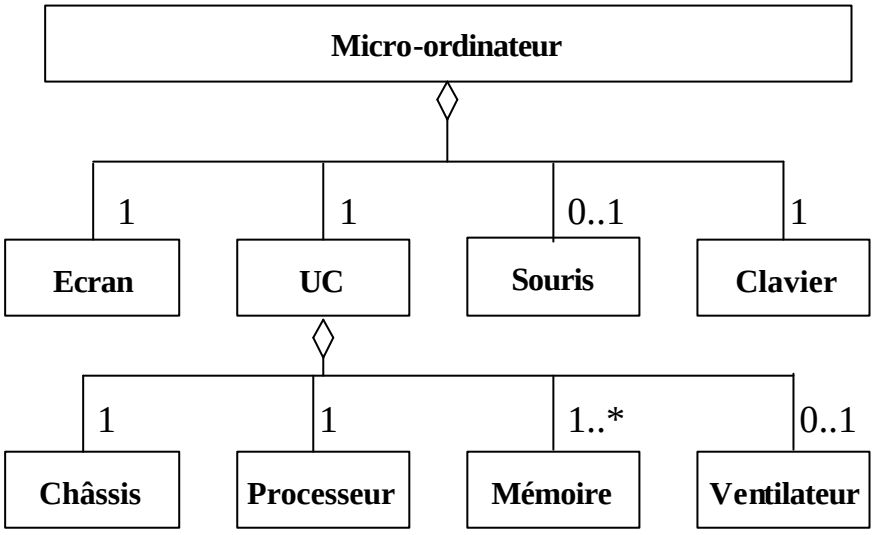

Les éléments du coté "plusieurs" ont un ordre explicite qui doit être préservé.

## Agrégation

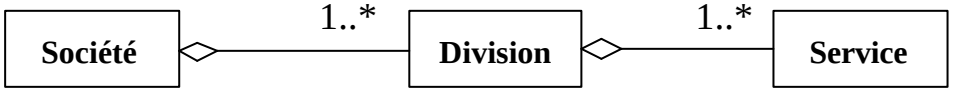
L'agrégation est une forme d'association forte dans laquelle un objet agrégat est *fait* de composants. Les composants font *partie de* l'agrégat.
L'agrégation met en relation des instances d'objets. Deux objets distincts sont englobés, l'un des deux est une partie de l'autre.

La décision d'utiliser l'agrégation est affaire de jugement et fréquemment arbitraire. Il n'est pas toujours évident de choisir entre association et agrégation.

Une agrégation peut être **fixe** si le nombre et les types des sous-parties sont prédéfinis.




Une agrégation **variable** a un nombre fini de niveaux mais le nombre de parties peut varier.

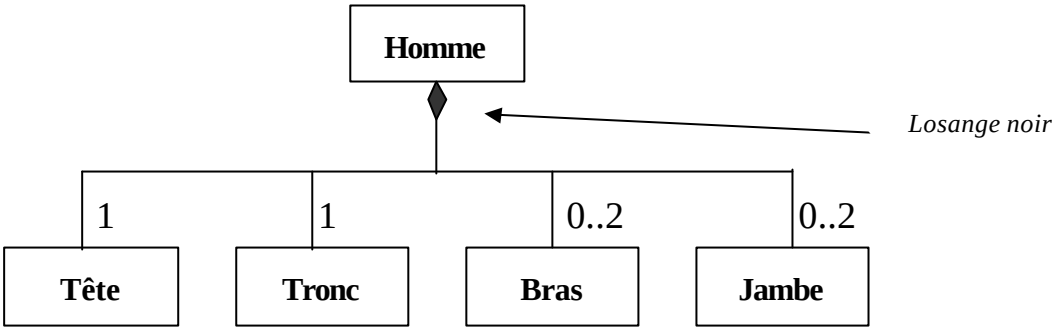



Une *Société* est une agrégation de *Divisions*, qui sont à leur tour des agrégations de *Services*. Une *Société* est donc indirectement une agrégation de *Services*.

## Composition

Une composition est une forme forte d'agrégation dans laquelle le cycle de vie des parties composantes est lié à celui du composé.

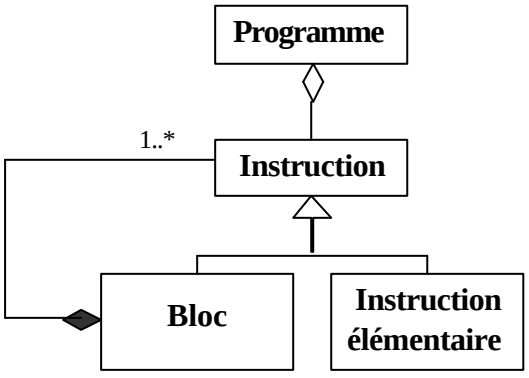
Les composants n'existent pas seuls. Ils peuvent être créées après le composé, mais ensuite elle vivent et meurent avec lui.
Si on détruit le composé, les composants n'existent plus.
Une fois établis, les liens ne peuvent pas être changés.




## Agrégat récursif

Un agrégat récursif contient directement ou indirectement une instance de la même sorte d'agrégat. Le nombre de niveaux potentiels est illimité.

Un programme informatique est un agrégat de blocs, incluant éventuellement des expressions composées récursives. La récursion s'achève sur des expressions simples. Les blocs peuvent être imbriqués sur un nombre arbitraire de niveaux.




Ce dessin illustre la forme habituelle d'un agrégat récursif:

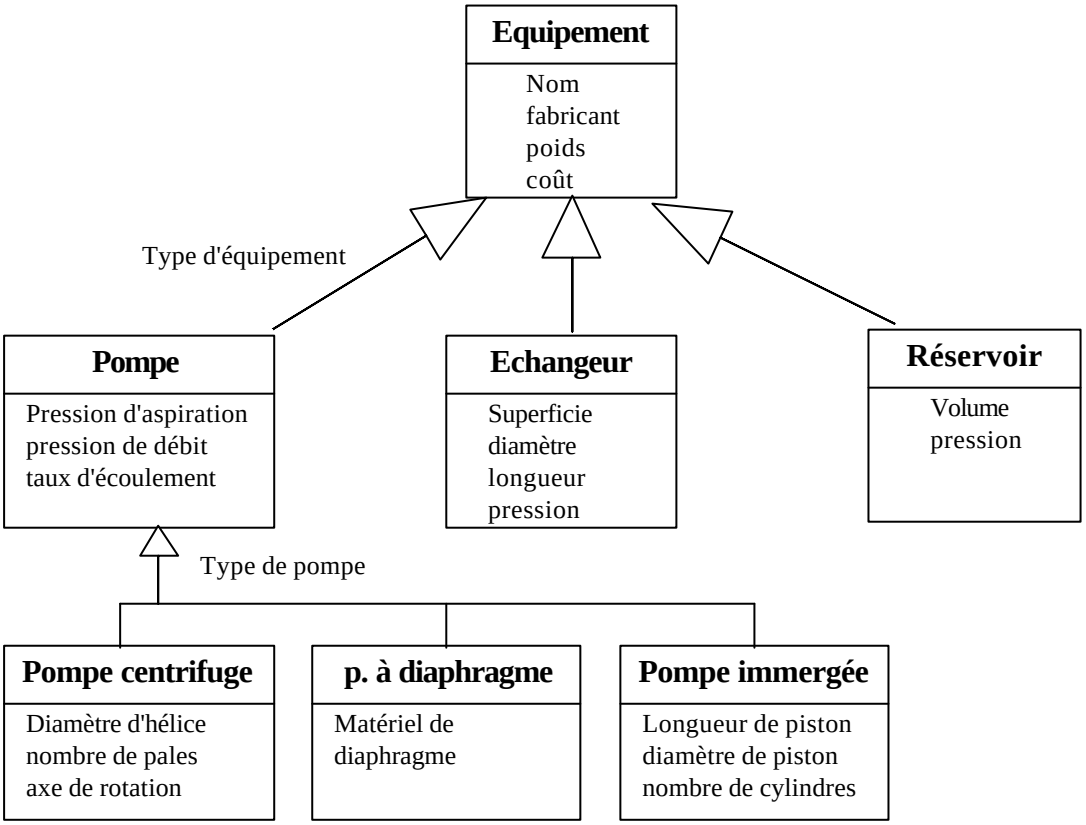
une super-classe et deux sous-classes, l'une étant une classe terminale de l'agrégat et l'autre un assemblage d'instances de la super-classe abstraite.

**Généralisation et héritage**
La généralisation met en relation les classes et constitue une façon de structurer la description d'un objet unique.

? La relation de classification correspond à des phrases du type "est une sorte de".

? Une instance d'une sous-classe est simultanément une instance de toutes ses classes ancêtres.

Principe de substitution de Liskov: une instance de l'élément plus spécifique peut être utilisé là où l'élément plus général est autorisé.

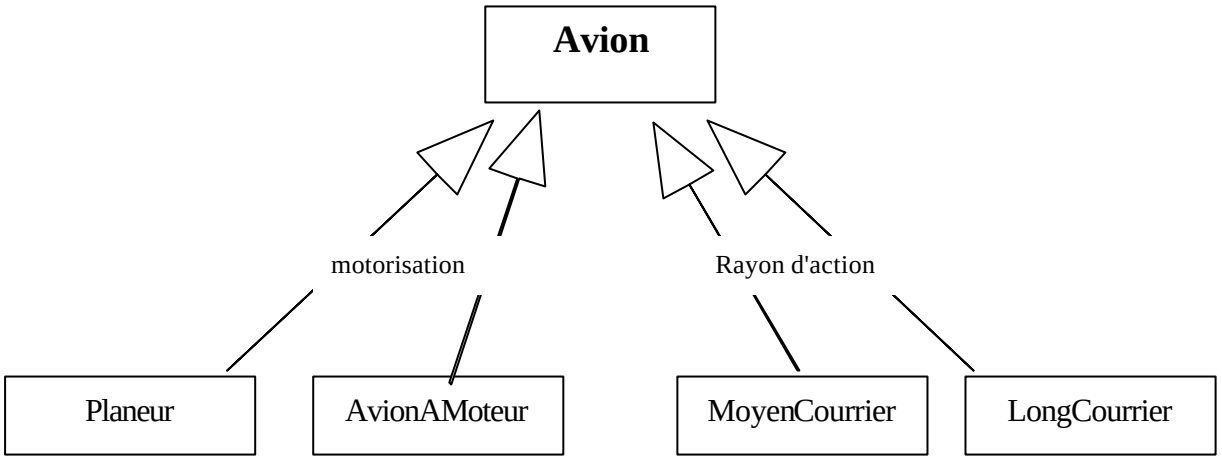



Les mots écrits près des triangles de généralisation sont des discriminants.
Un *discriminant* est un attribut de type énumération qui indique que la propriété d'un objet a été rendue abstraite par une relation de généralisation particulière.
Une seule propriété doit être discriminée à la fois.

L'**héritage** est le mécanisme par lequel des éléments plus spécifiques incorporent la structure et le comportement d'éléments plus généraux.

Extension de la notation

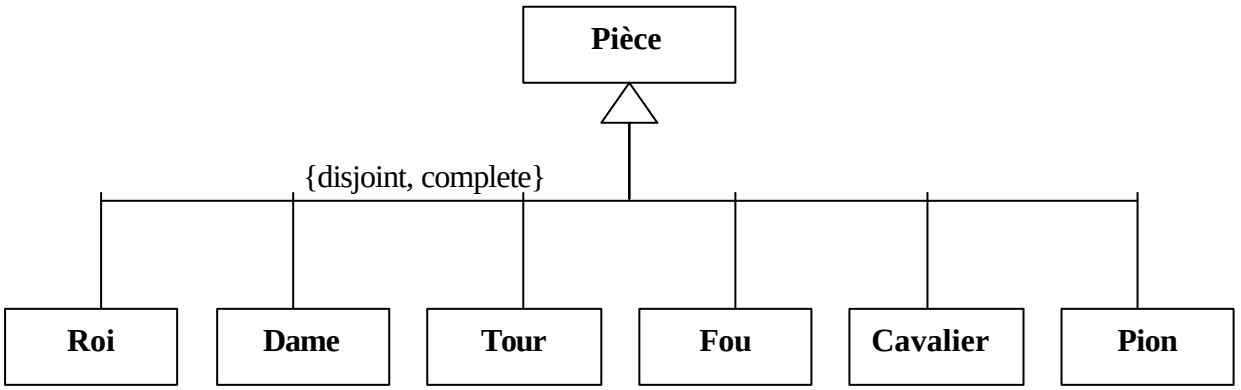
Les discriminants sont indispensables s'il existent plusieurs arbres de classification différents.




Contraintes sur la généralisation

Les deux seules contraintes prédéfinies en UML pour la généralisation sont :

- Disjoint
- Complete (ou incomplete)



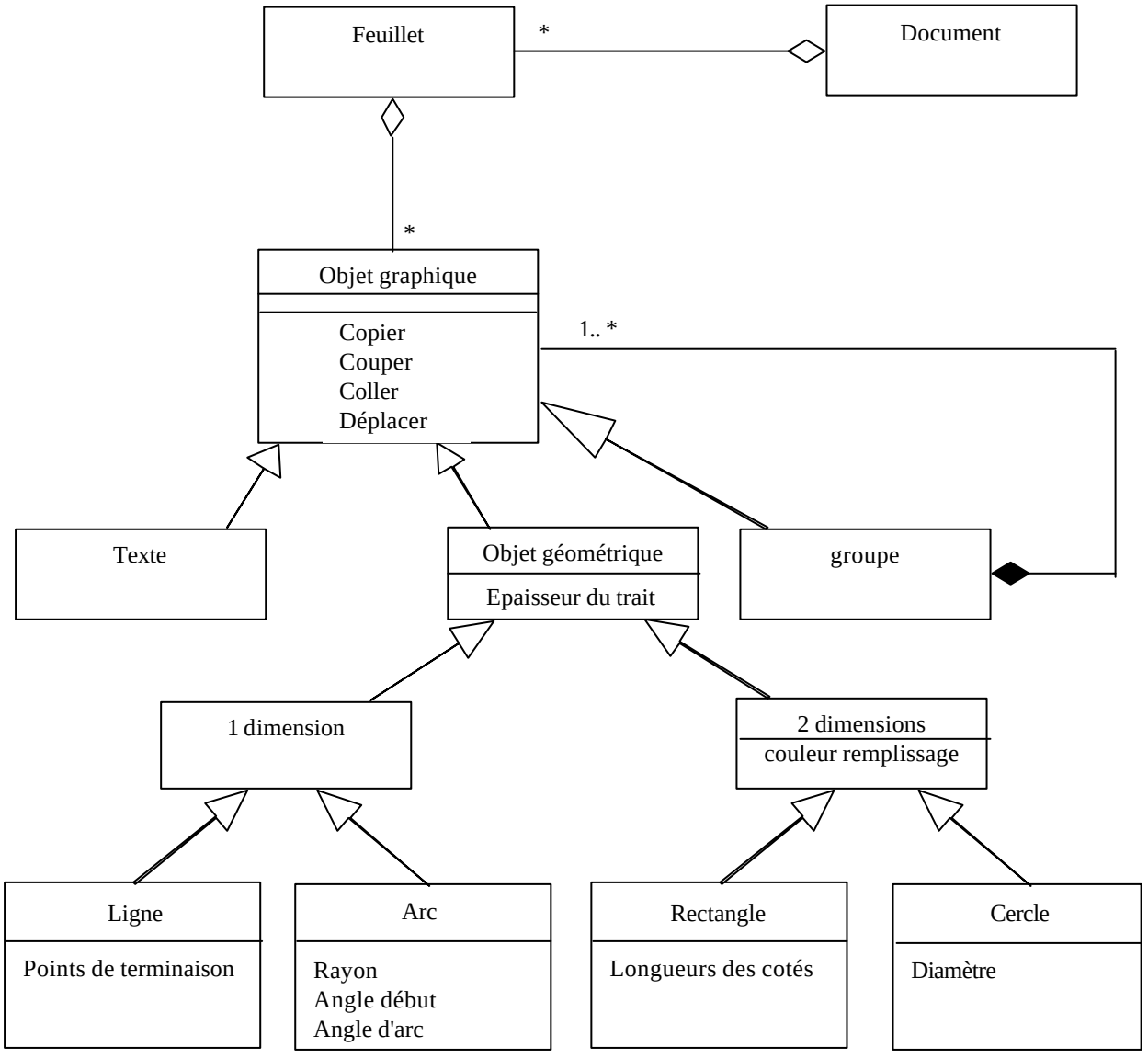
## Exercice

Représenter le modèle objet d'une gestion technique de documents.

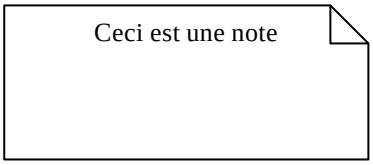
Chaque document est composé d'un ou plusieurs feuillets.
Un feuillet comporte du texte et des objets géométriques qui constituent deux types d'objets graphiques supportant des opérations de type : copier, couper, coller et déplacer.
Considérer les quatre objets géométriques suivant :

- cercle caractérisé par un diamètre,
- ligne caractérisée par ses points de terminaison,
- arc caractérisé par un rayon, un angle de début et un angle d'arc,
- rectangle caractérisé par les longueurs de ses cotés.

Indiquer les attributs: épaisseur du trait et couleur de remplissage.



**Note**
Une note est un commentaire placé sur un diagramme :

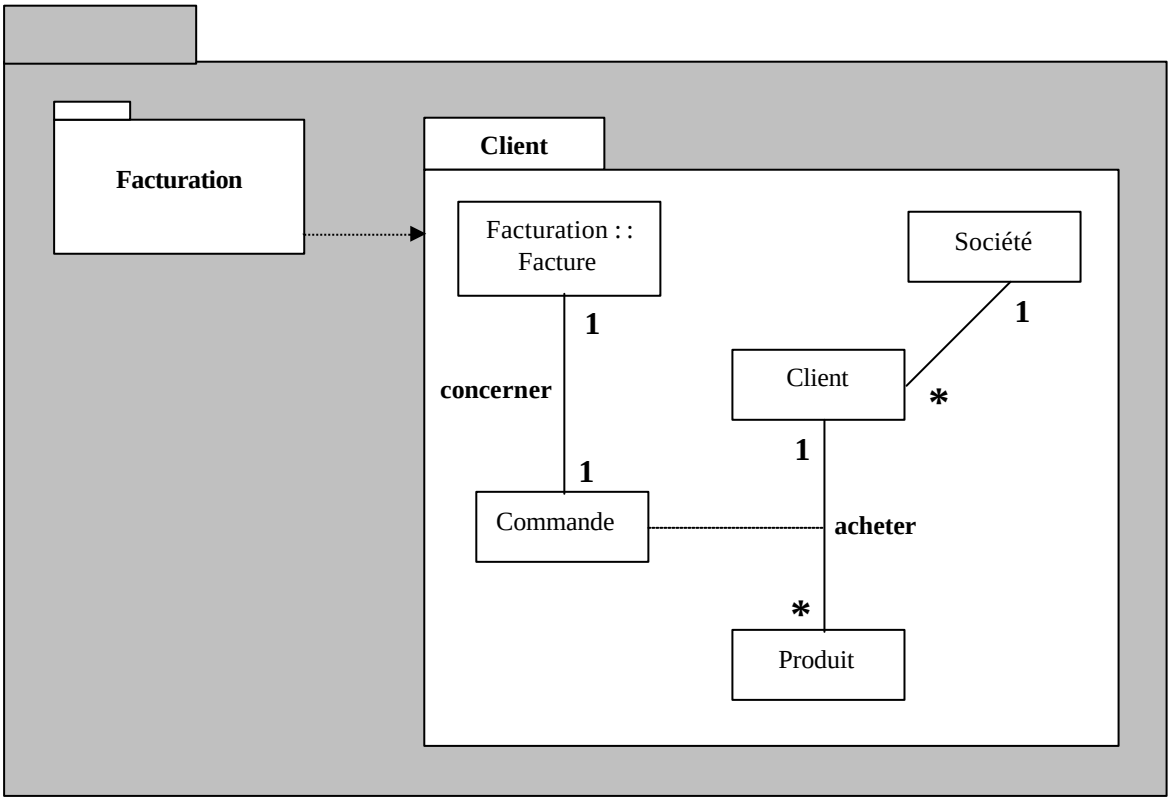



## Package

Un package est un regroupement d'éléments du modèle. Il peut comporter plusieurs types d'entités (classes, associations, package).
Un élément du modèle, et plus particulièrement une classe peut être représentée plusieurs fois dans un modèle. Il est alors bon d'indiquer le package d'appartenance d'une classe comme suit : *NomPackage : : NomClasse*

Le nom d'une classe est unique dans un package. Par contre il est possible d'avoir deux classes de même nom dans des packages différents.

Le système entier peut être vu comme un package unique de haut niveau contenant tout le reste. On peut aussi représenter les dépendances entre packages.



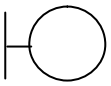
**Stéréotype**
Nouveau type d'élément de modélisation qui étend la sémantique du méta-modèle !

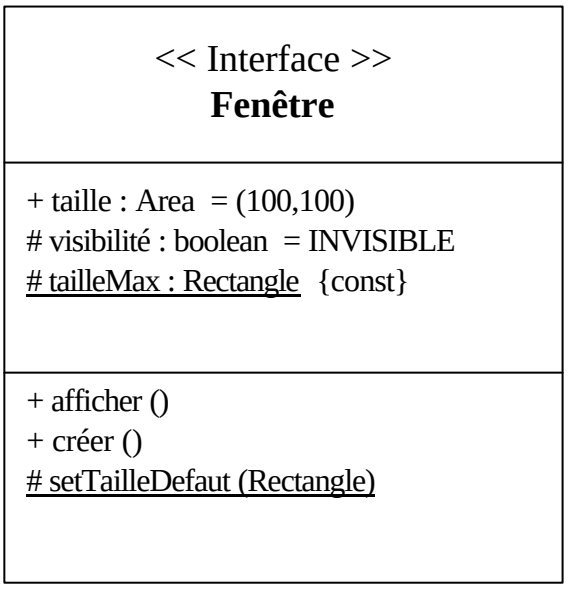
Un stéréotype permet de classer les éléments du modèle en grandes familles.
Il peut être associé à tout élément du modèle (classe, association, opération, attribut, package …)

On peut définir ses propre stéréotypes; par exemple les classes d'interface, les classes de contrôle, les classes entité.

On peut même y associer un(e) icône.

**<< Interface>>**






**Pattern**
Modèle générique résolvant un problème classique et récurrent.

- Solution qui a fait ses preuves, fondée sur l'expérience.
- Solution réutilisable dans des contextes différents.

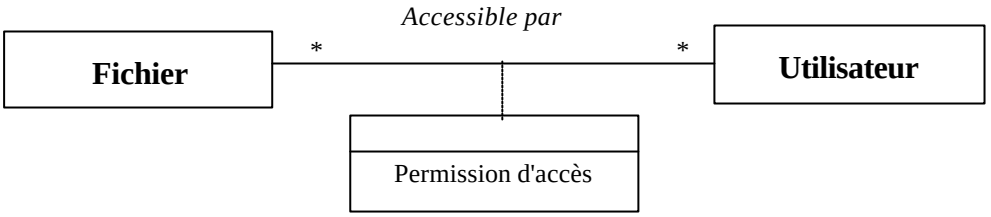
De plus en plus de Patterns sont disponibles au travers de livres et publications.

- Design Patterns
- Analysis Patterns

*L'agrégat récursif* qui modélise les structures arborescentes, est un exemple d'Analysis Pattern.

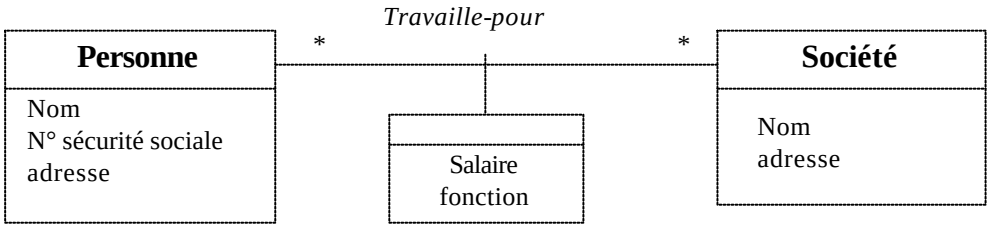
**Attribut d'association**
Un attribut d'association est une propriété des liens d'une association.

Les associations plusieurs à plusieurs sont la justification la plus contraignante de l'existence des attributs d'association. Un tel attribut est sans conteste une propriété du lien et ne peut être rattaché à l'un ou l'autre des objets.




| | | |
|---|---|---|
| /user/cdrgest | (lire) | Jean Martin |
| /user/cdrgest | (lire-écrire) | Marie Durand |
| /com/grmod/.login | (lire-écrire) | Jean Martin |

Autre exemple :



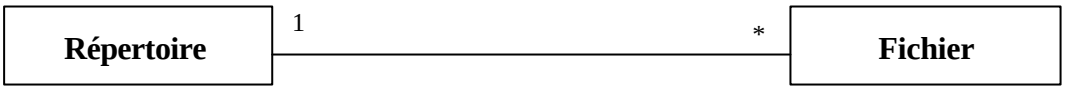

Chaque personne travaillant pour une société reçoit un salaire et occupe une fonction.

Il est possible de ranger les attributs des associations un à un et un à plusieurs dans la classe opposée au coté "un".
Cela n'est pas conseillé, car la souplesse d'évolution serait réduite si les multiplicités de l'association devaient changer.

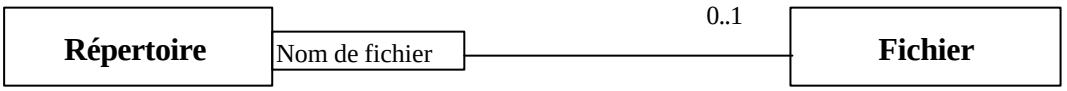
**Qualification**
Le qualificatif est un attribut spécial qui permet de filtrer les éléments de la cible. Cela réduit souvent la multiplicité effective d'une association.

Les associations un à plusieurs et plusieurs à plusieurs peuvent être qualifiées.

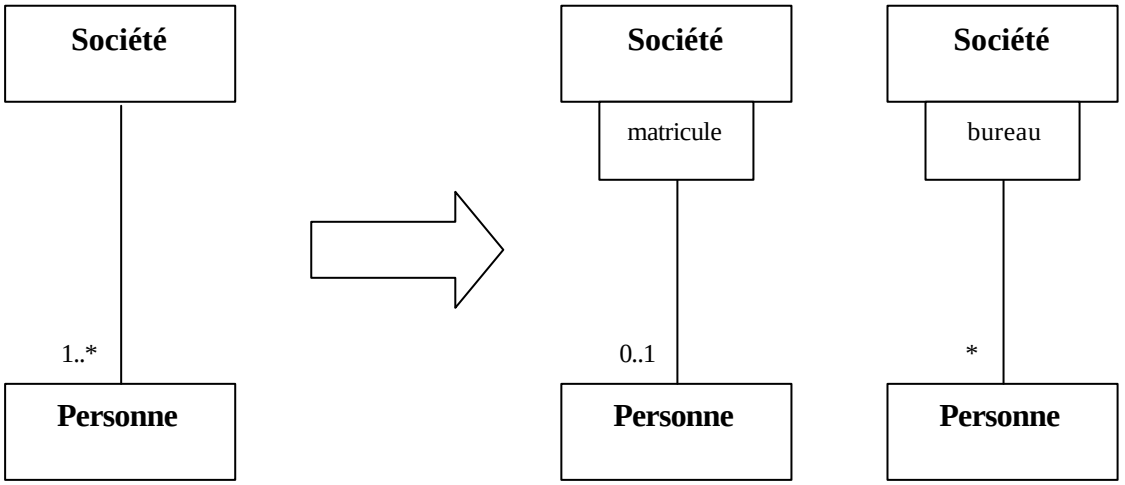
Exemple :




Un répertoire contient plusieurs fichiers.
Un fichier ne peut appartenir qu'à un seul répertoire.
Dans le contexte d'un répertoire, le nom de fichier (attribut de fichier) spécifie un fichier unique dans la classe fichier.




Autre exemple :



## Exercice

Modéliser la structure de l'arborescence des fichiers sous windows 95.

? Les fichiers, les raccourcis et les répertoires sont contenus dans des répertoires et possèdent un nom.

? Un raccourci peut concerner un fichier ou un répertoire.

? Au sein d'un répertoire donné, un nom ne peut identifier qu'un seul élément (fichier, sous-répertoire ou raccourci).