

UML



Le **diagramme de Classe** va permettre de représenter une vue statique du système d'information. **Pas de dynamisme** ici puisqu'on n'évoque pas les stimuli qui font réagir le SI, il s'agit plutôt des relations entre les Classes, des services rendus et utilisés par chacune d'elles et de l'articulation de l'ensemble.

Ce diagramme sera souvent utilisé pour vous **présenter des design pattern**, car il montre bien la structure (figée, statique) de la solution logicielle.

Une Classe

Elle est représentée de la façon suivante (attention, en fonction du contexte, on peut omettre ce que l'on veut).

NomClasse (*en italique si abstraite*)

(visibilité + = -) nomAttribut : typeAttribut
- nb_de_chevaux : int
- client : Person

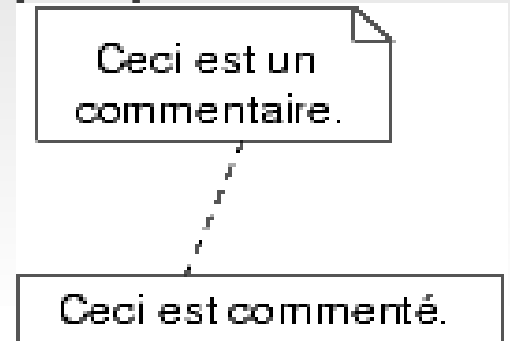
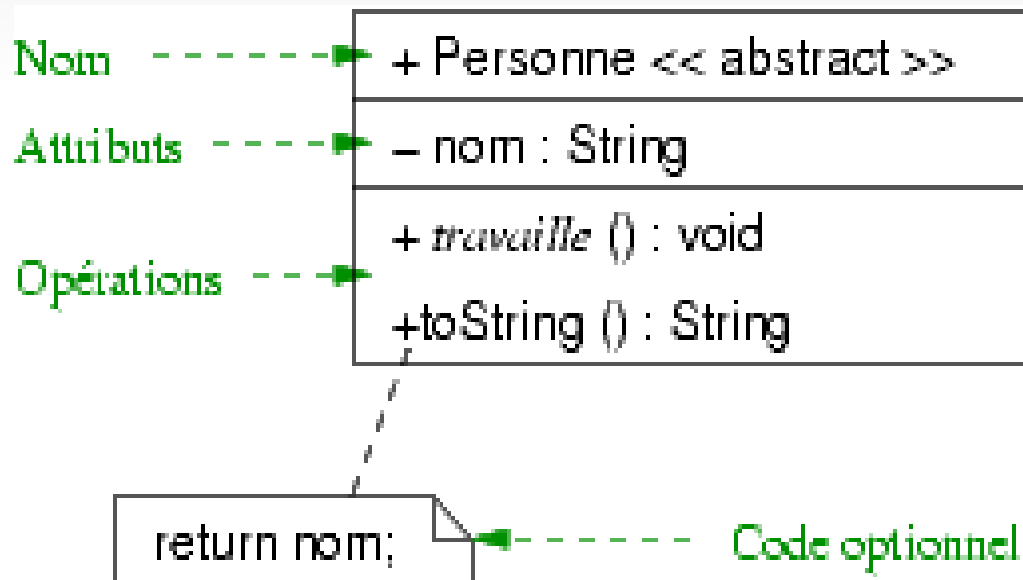
(visibilité + = -) nomMethode(args) : typeRetour
+ getNbCv() : int
+ setNbCv(int) : void
+ getClient() : Person

Cette représentation peut varier selon le moment où elle est utilisée. Si l'analyste en est à la conception, il restera plus générique. Lorsqu'il en arrivera à l'implémentation, le diagramme peut être bien plus complet, et différent (des classes supplémentaires apparaissent, des méthodes aussi...)

Détails sur les Classes

En UML, il est toujours possible de sortir du schéma grâce à **des commentaires** qui peuvent prendre la forme suivante.

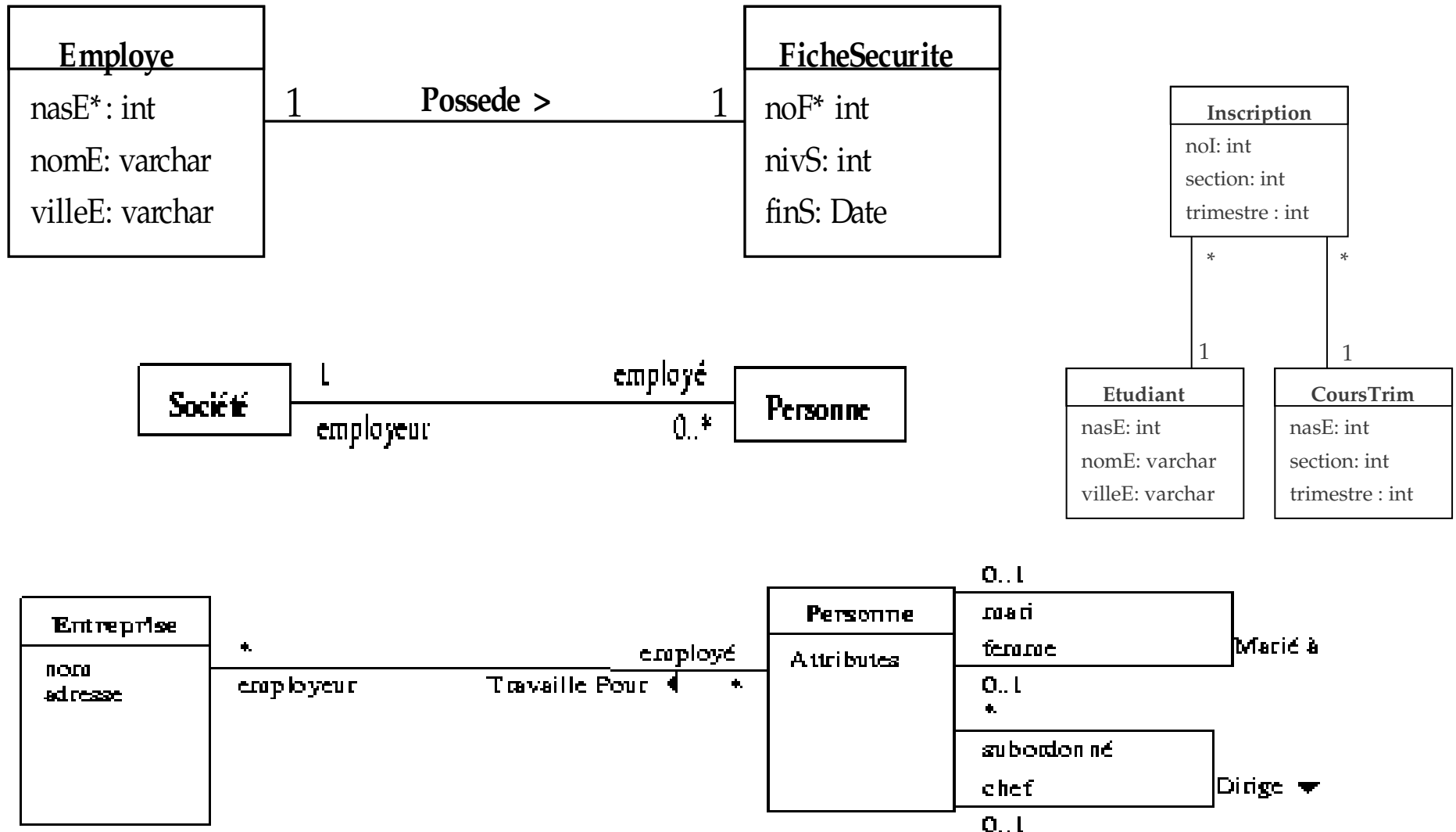
Ce qui nous donne le schéma suivant.



Les associations.

Les associations **sont des relations** entre Classes. Elles représentent *un lien durable ou ponctuel* entre deux objets, une appartenance, ou une collaboration. Elles sont représentées **par une ligne entre les classes**. Sur cette ligne, un verbe à l'infinitif permet d'expliquer la **sémantique** de l'association (non obligatoire). De même, on peut aussi donner un nom de chaque côté de l'association, afin de nommer le rôle de chacun. En théorie, l'association se lit de gauche à droite. Si le verbe se lit dans l'autre sens, il faut l'indiquer. Des cardinalités expriment le nombre d'instances en jeu dans la relation

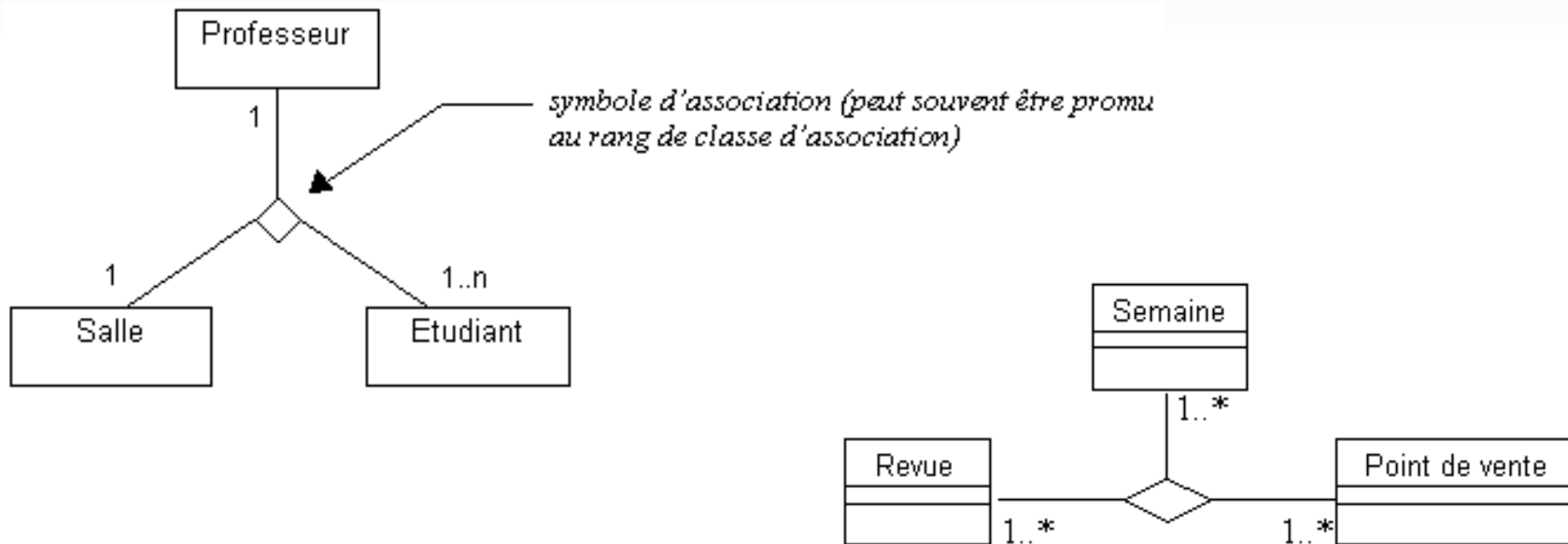
Les associations (exemples)



Par exemple, on peut lire tout en haut qu'**UN EMPLOYE POSSEDE UNE FICHE SECURITE**

Association n-aire

Il est possible que **plusieurs Classes** participent à l'association. Ce n'est alors plus une association binaire, mais **n-aire**. On l'indique par un losange. Méfiez-vous de la **complexité induite** (l'association implique l'ensemble des classes participantes).

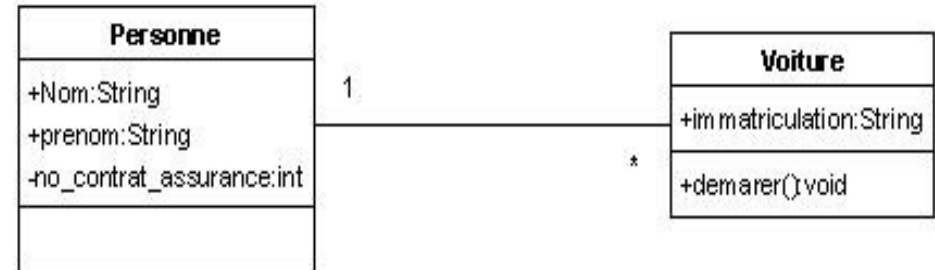


Multiplicité et Navigabilité...

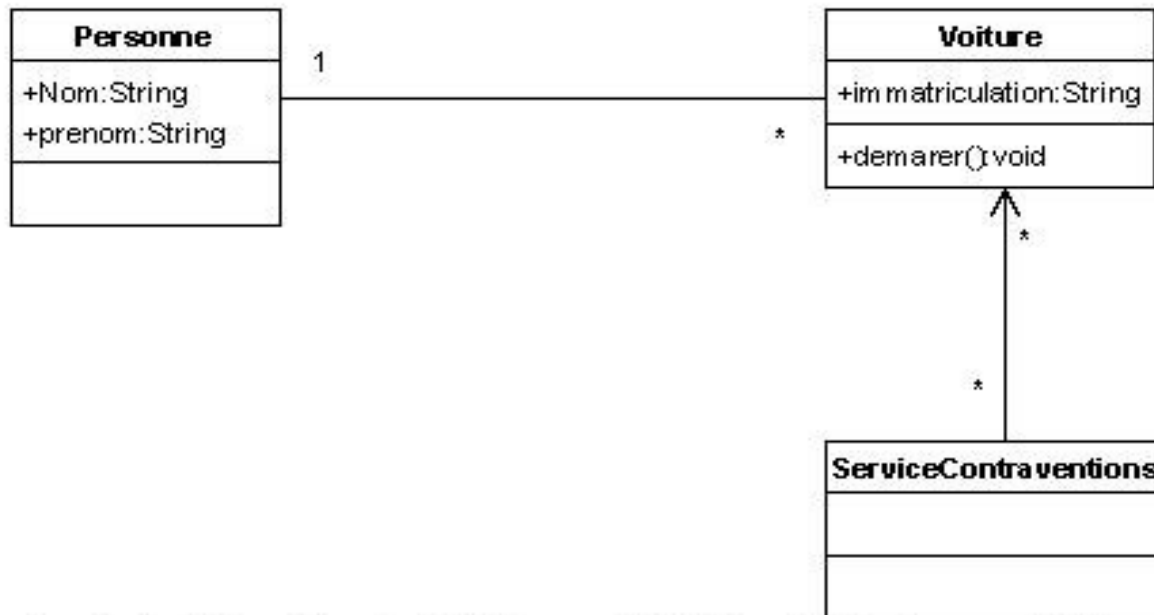
La **multiplicité** indique les cardinalités entre les classe et l'association. On l'exprime souvent par une **valeur finie** (3, 5 ou 17) ou un **intervalle** (2..3, 1..17, ou encore 2..*). Dans le cas de 0..*, on note **parfois** * tout simplement.

Par défaut, l'association peut être utilisée dans les deux sens. **Très souvent**, on s'aperçoit que l'association est **uni-directionnelle**. Elle est donc navigable dans un seul sens. On l'indique avec une **flèche** sur l'association (la flèche à côté du verbe indique le sens de la phrase...)

Exemple navigabilité



Created with Poseidon for UML Community Edition. Not for Commercial Use.

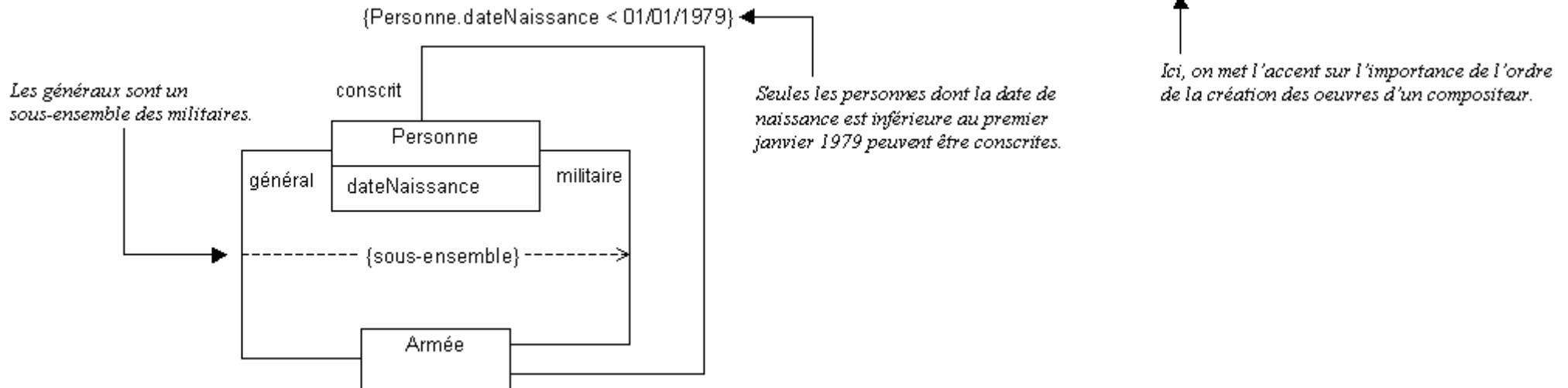
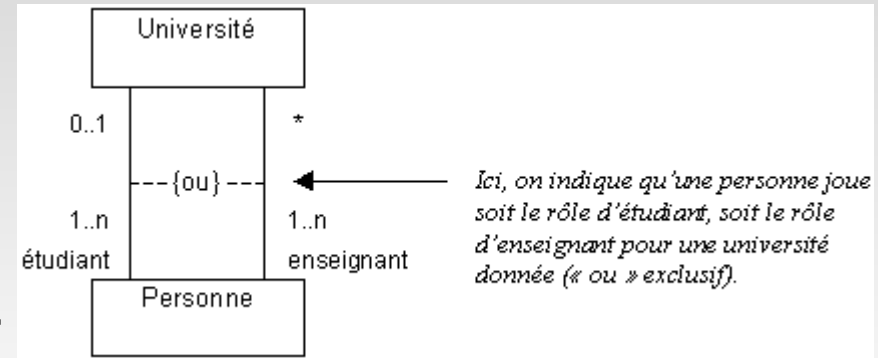


Created with Poseidon for UML Community Edition. Not for Commercial Use.

- Chaque instance de voiture a **un lien** vers le propriétaire
- Chaque instance de Personne a **un ensemble de lien** vers les voitures
- Le service de contravention est **associé** à une ou plusieurs voiture(s)
- La voiture **ne connaît pas** service de contravention

Des contraintes sur l'association

Il est possible d'exprimer des **contraintes** sur une association, afin de limiter les objets mis en jeu. Cela permet de mieux **cadrer** l'architecture de l'ensemble

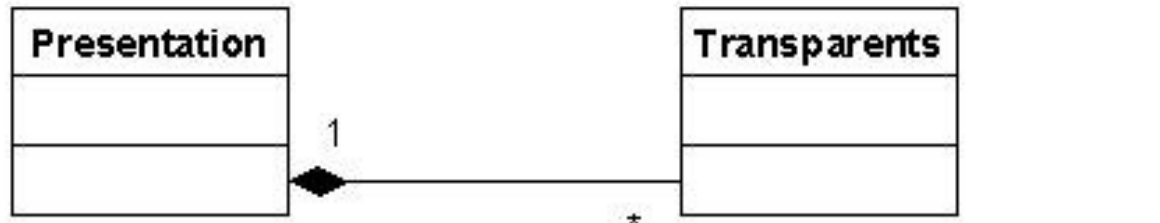


(bornes, ordonnancement, inclusion d'une association dans une autre).

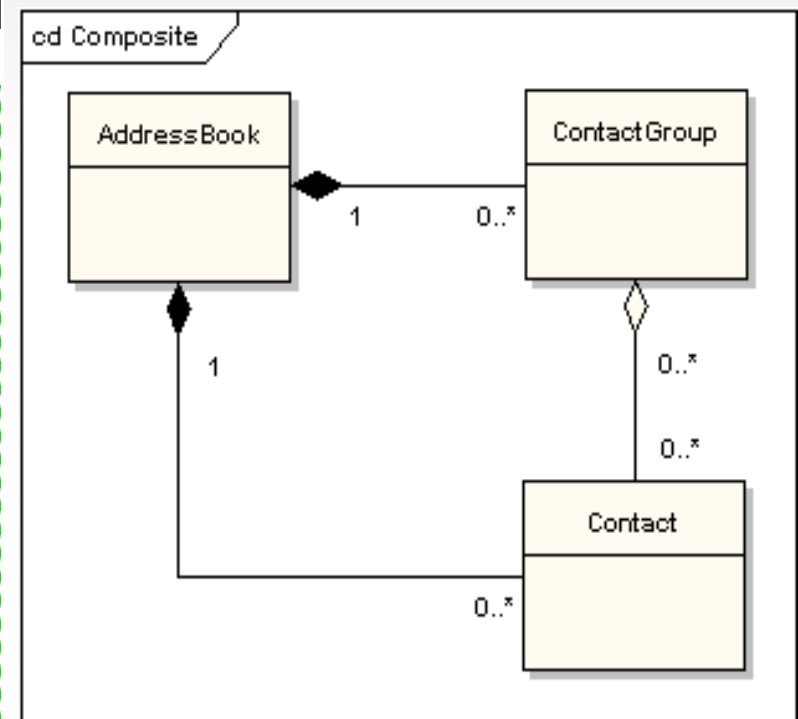
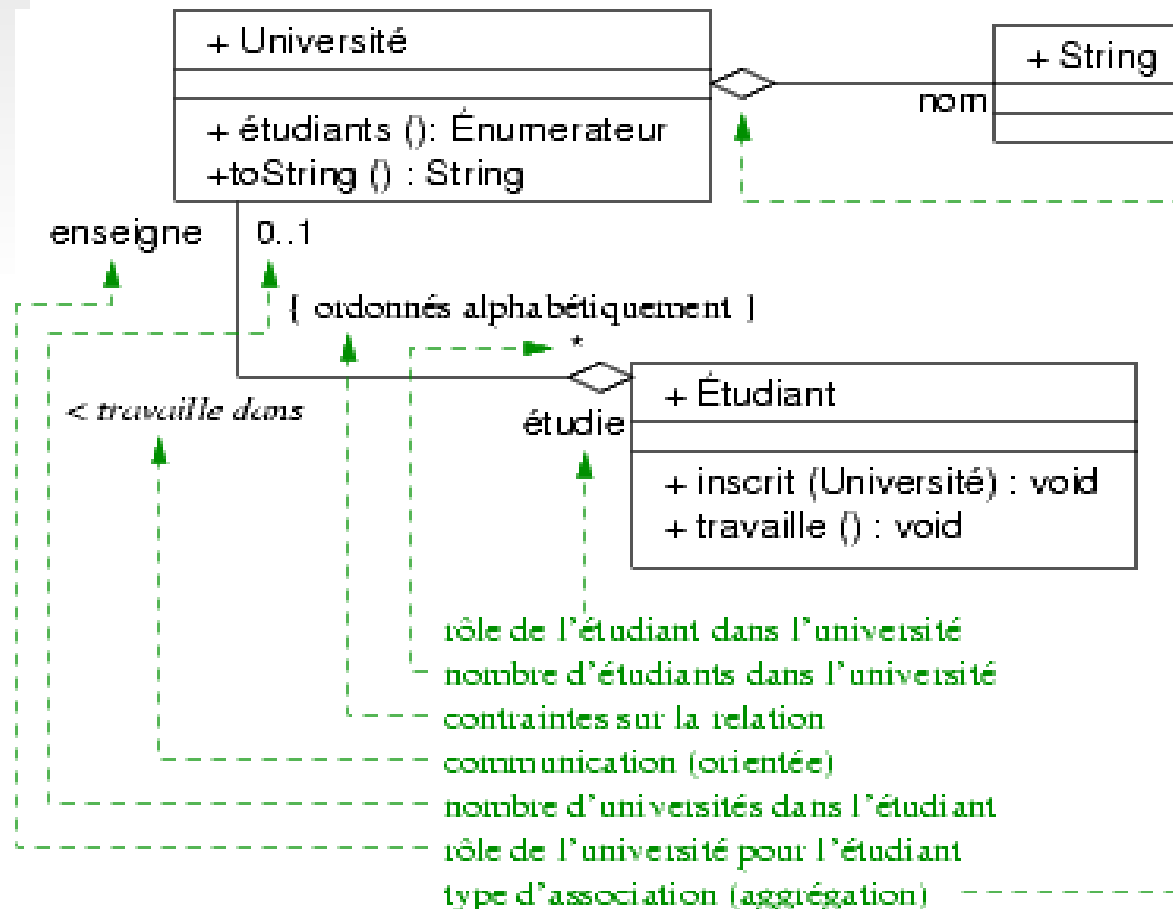
Agrégation et composition...

Ce sont deux types d'associations aux caractéristiques spécifiques. Les deux indiquent une **dépendance très forte** de l'élément par rapport à son contenant. La relation n'est **pas symétrique**. Il y a appartenance forte. La différence entre ces deux relations d'appartenance réside dans le niveau de dépendance du '**contenu**' ; Si le '**contenu**' est lié **exclusivement** au '**contenant**', et qu'il disparaît avec lui, alors on dit qu'il y a **COMPOSITION** (élément constitutif=*losange plein*). Sinon, il s'agit d'une **AGREGATION** (plutôt une coopération=*losange creux*).

Exemple Agrégats-composition

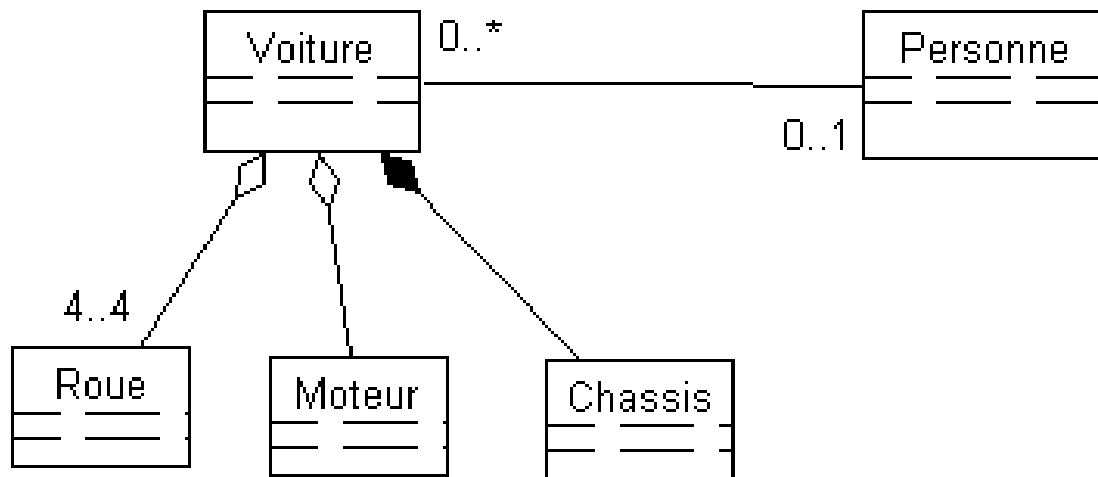


La **suppression** de la présentation entraîne la **disparition** des transparents qui la composent



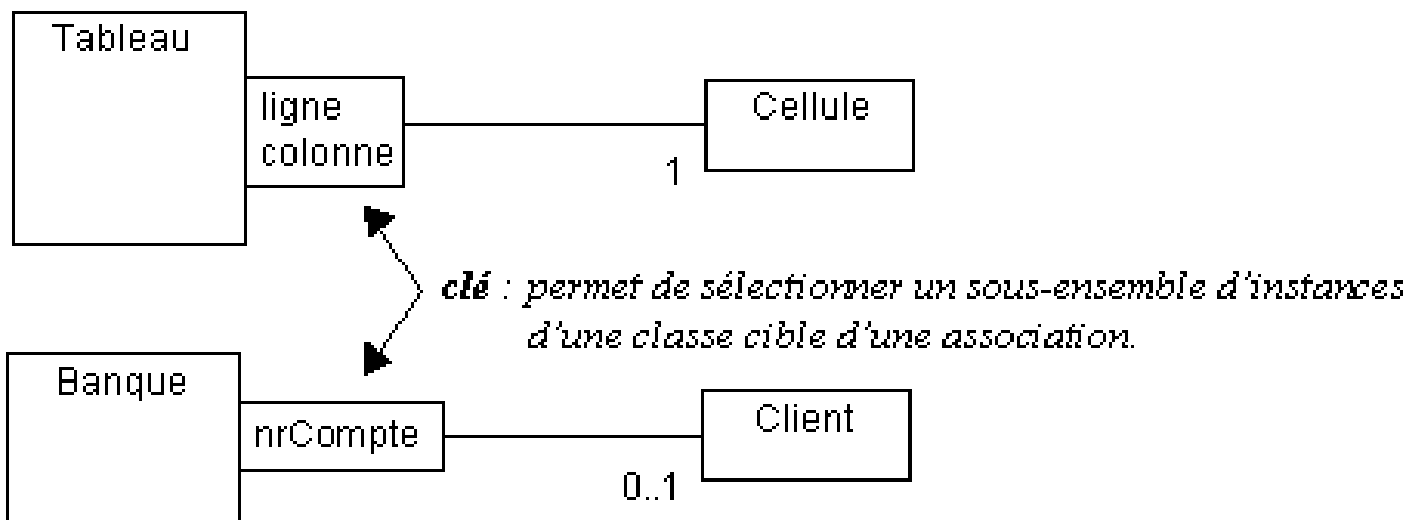
Mélange agrégats Compositions

Pourquoi y'a t-il parfois composition et parfois agrégation ?



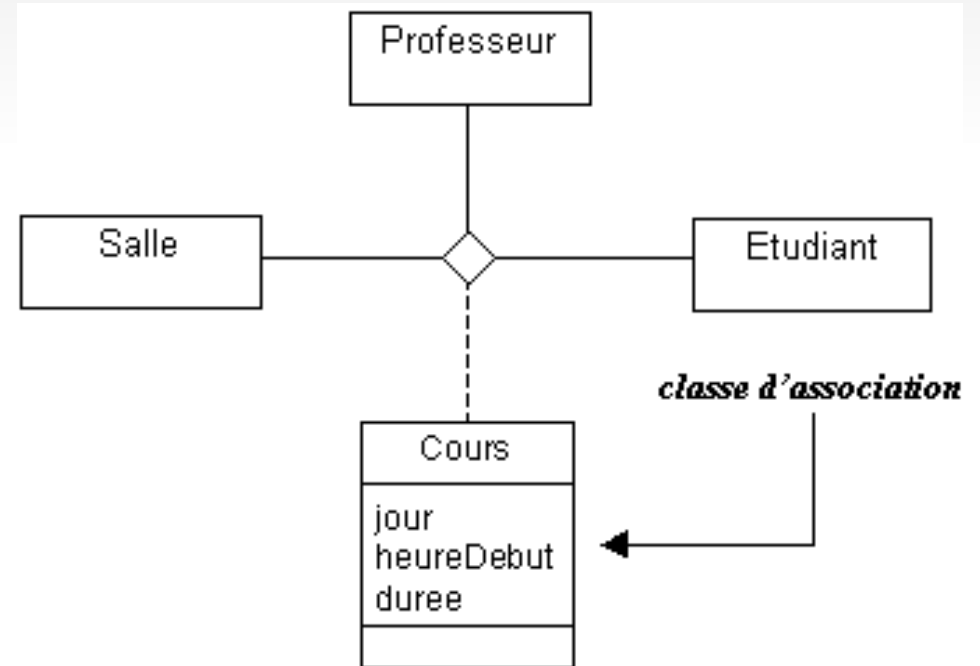
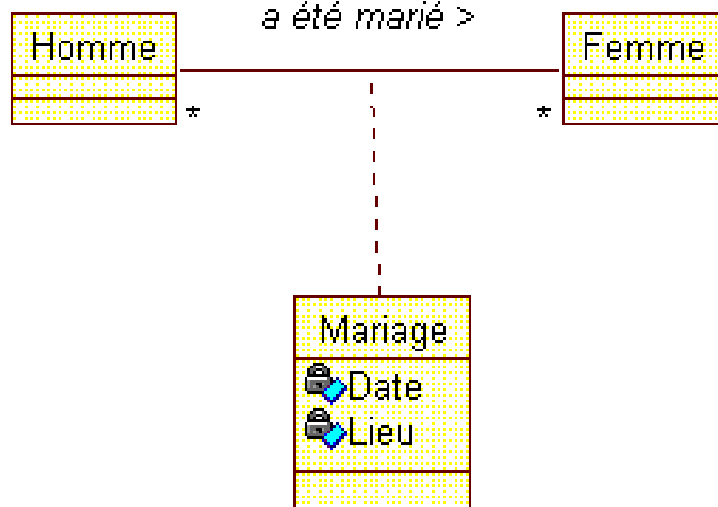
La qualification

Parfois, un **élément** (un attribut) permet la sélection d'un sous-ensemble d'objets (de 1 à n) participant à l'association. Souvent, cela permet de **réduire** la **cardinalité** de l'extrémité de l'association à **1...** Il pourra s'agir d'une clé dans une HashTable.



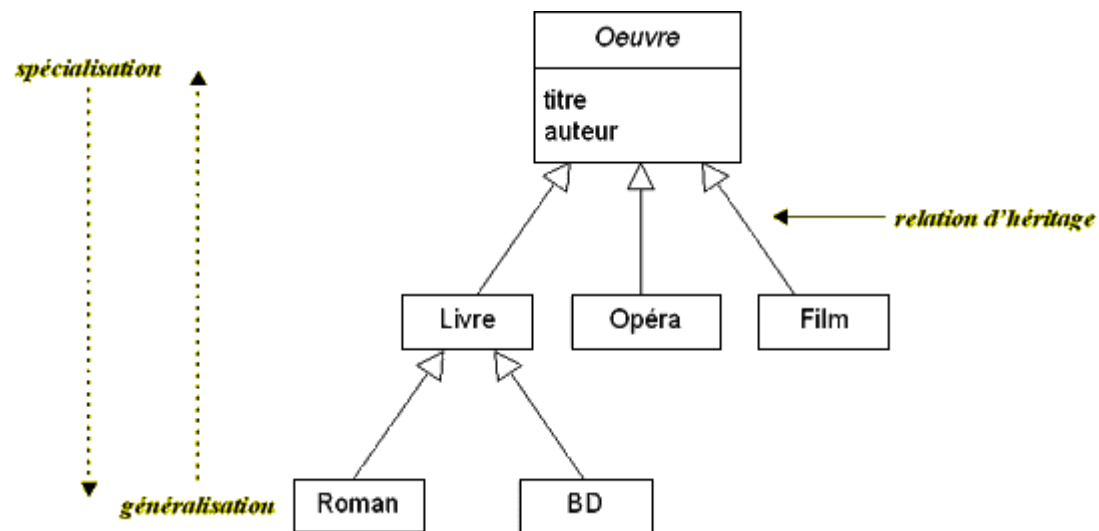
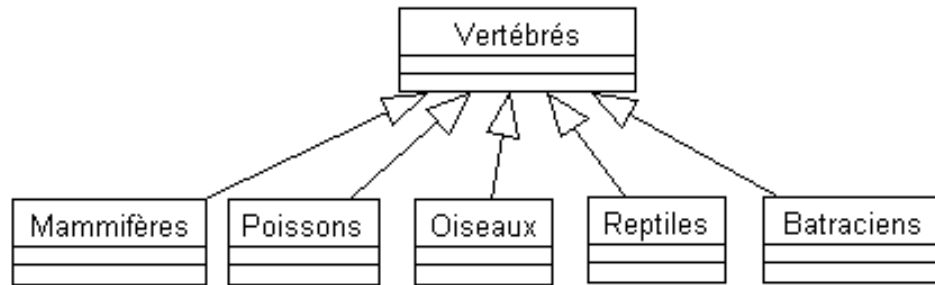
Les classes d'association.

Si vous détectez que votre association est **porteuse d'informations**, il est possible d'utiliser une classe d'association. Elle comportera **attributs, méthodes, etc...**



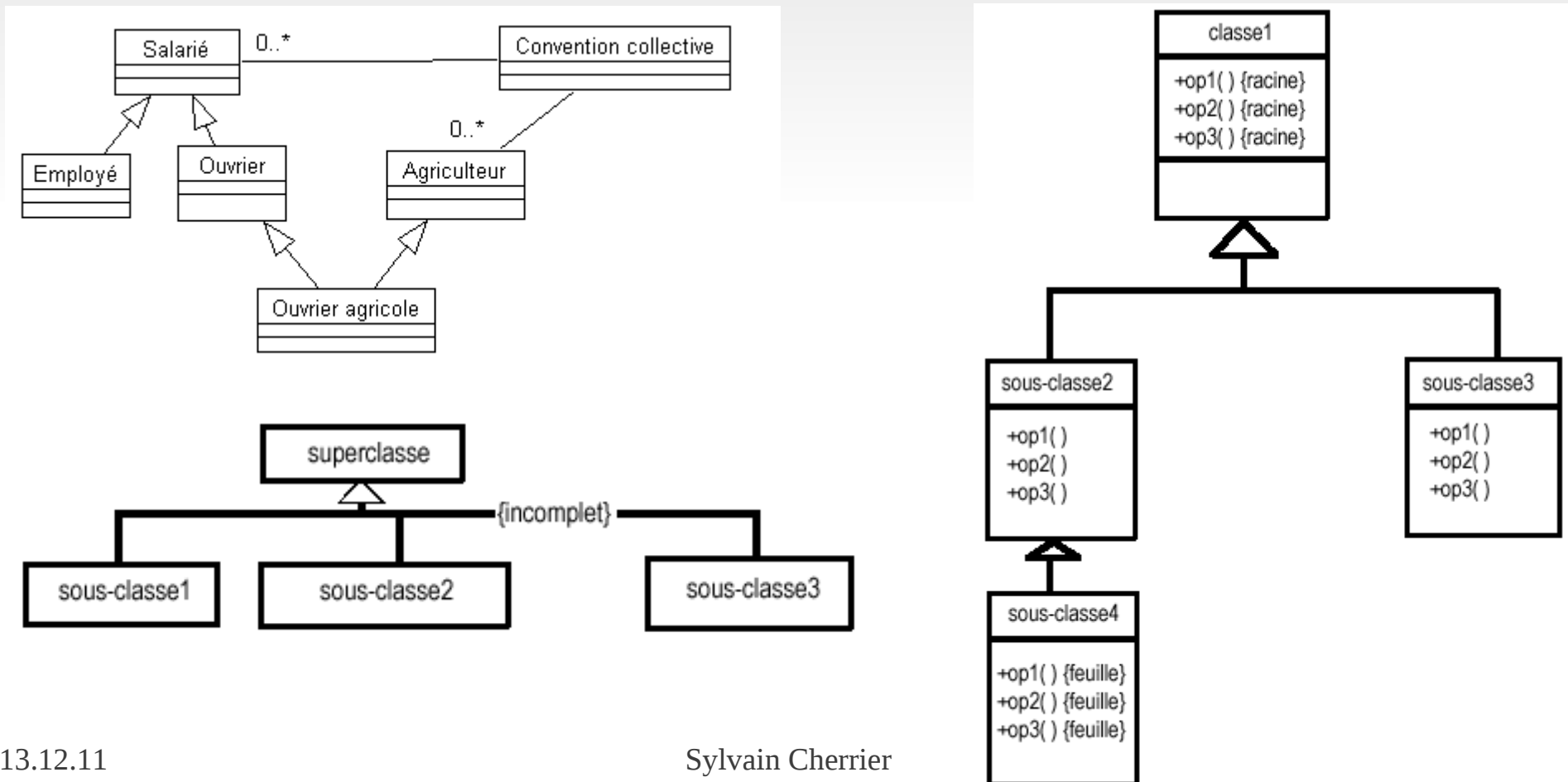
Héritage

cette notion '*principe de généralisation/spécialisation*' permet de définir les relations entre sous-Classe et Super-Classe.



Héritage suite

L'héritage peut être **contraint** par divers moyens, être porteur d'informations. Il peut être **multiple**. Il peut aussi définir des limites dans l'arbre...



Les interfaces

Il s'agit la encore de **factoriser** un ensemble d'attributs et d'opérations, décrivant de la sorte un service cohérent. Il s'agit en général d'un **comportement** générique qui est décrit, et qui n'est pas décrit par une arborescence de Classe.

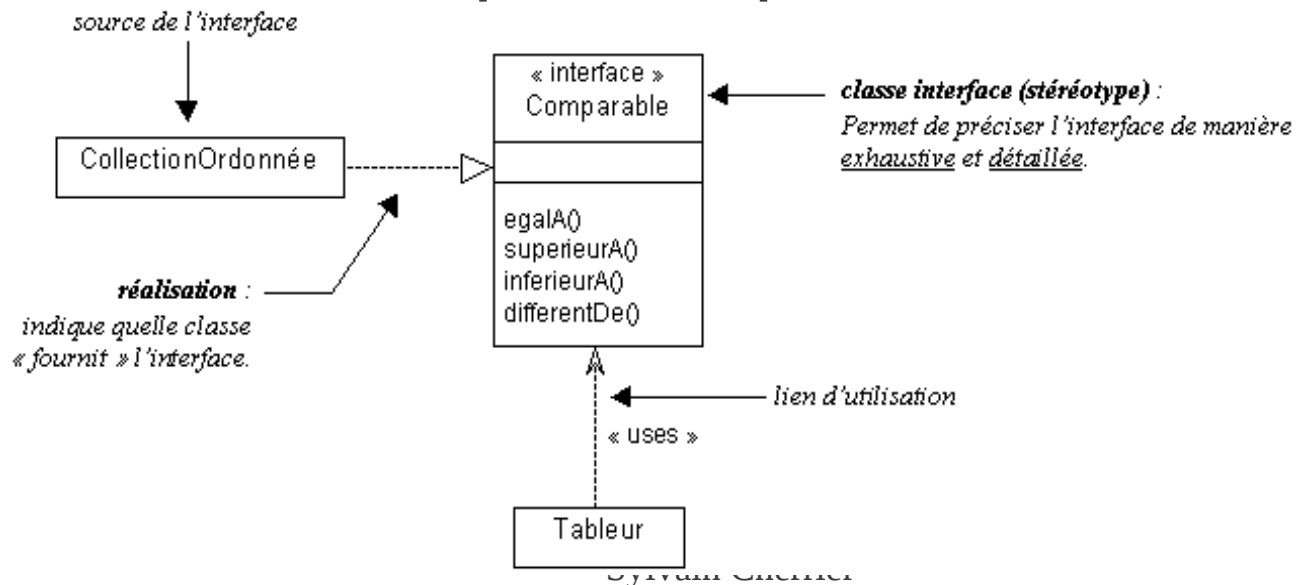
C'EST UN DESIGN PATTERN !!

Ainsi, la faculté d'être **Sauvegardable**, **Comparable**, **Jouable** peut être décrite via une **interface**, et ainsi implémentée dans de nombreuses Classes diverses. Certaines méthodes vont réclamer un objet qui implémente cette interface, peu importe le type d'Objet (Comparable pour TreeSet, en Java par ex).

Les interfaces suite.

Une **interface** est un **contrat**. Lorsqu'un objet client d'une **interface collabore** avec celle-ci, il ne connaît pas son type réel mais il sait comment le "**manipuler**" (*quels messages lui envoyer*).

Le but des interfaces est de **diminuer le couplage** entre deux classes. L'interface permet de ne présenter au client "*que ce qui l'intéresse*".



les interfaces

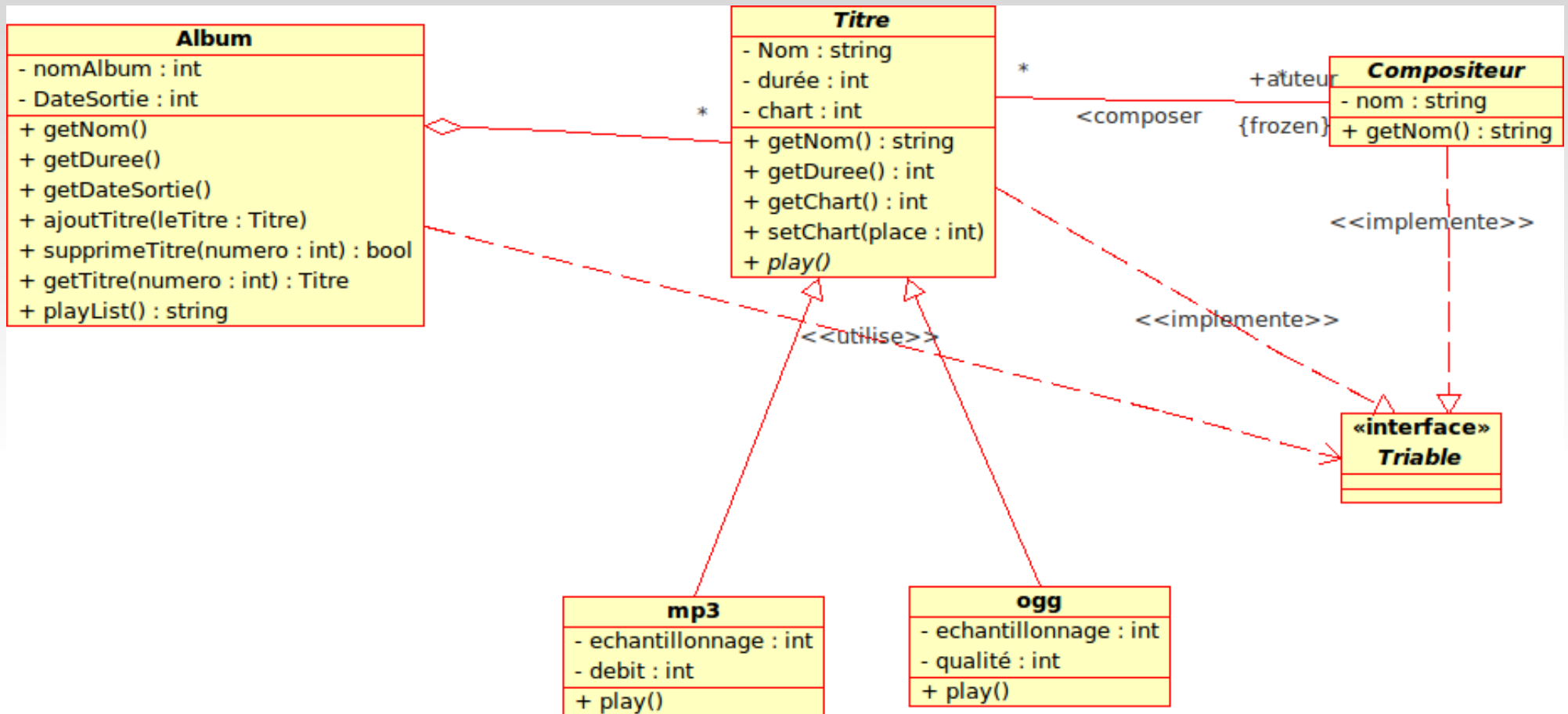
Une façon usuelle de présenter les interfaces insiste sur le fait que l'interface définit des **spécifications**. Elle impose la présence de certaines méthodes, quelque soit l'objet et sa position dans une arborescence de Classes. L'objet, lui, est chargé de **l'implémentation** de la **spécification**.

On pourra utiliser n'importe quel objet dont on veut être sûr qu'il implémentera une interface. (en C++ => classe entièrement virtuelle)

...

```
int c;  
Comparable objet1,objet2;
```

Autre exemple.



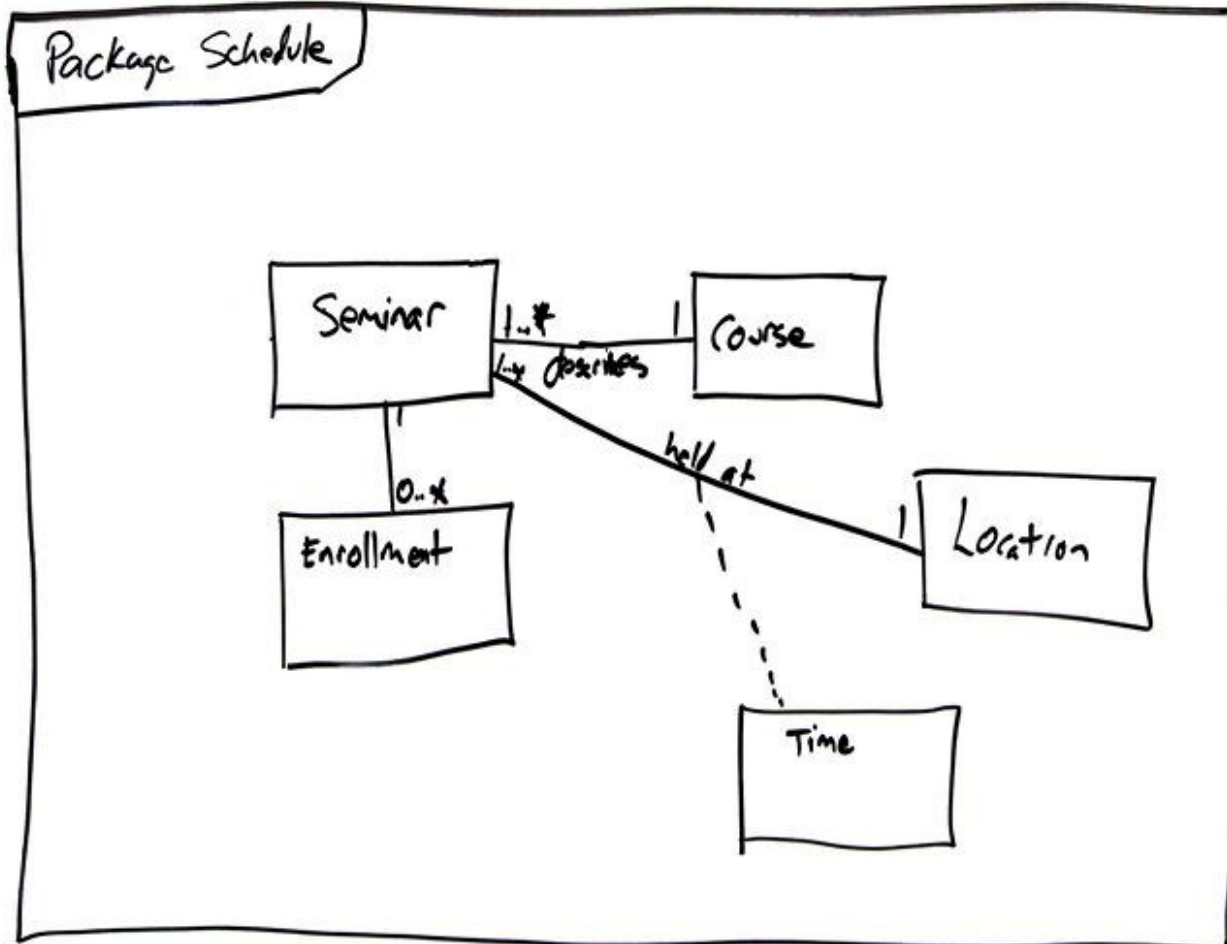
Ici, **Triable** permet d'indiquer qu'à la fois les *Titres* et les *Compositeurs* sont Triables. Cependant, l'héritage aurait été ici une *hérésie*.

Les Packages

Ce mécanisme permet de **regrouper** des éléments. Les éléments à l'intérieur du package auront des **accès privilégiés**. Le package étant un **espace de nommage**, tout nom de classe est **unique** en son sein.

L'idée du package est de **rapprocher** les éléments **sémantiquement proches**, offrant un ensemble de services homogènes et cohérents. Ils séparent le modèle en éléments logiques, et montrent leurs interactions à un plus haut niveau. Par contre, il faut *minimiser* les liens entre packages.

Exemple de package



Ici, un **package** cohérent de gestion de cours, avec des inscriptions dans des séminaires de cours, qui ont lieu dans des *endroits*.