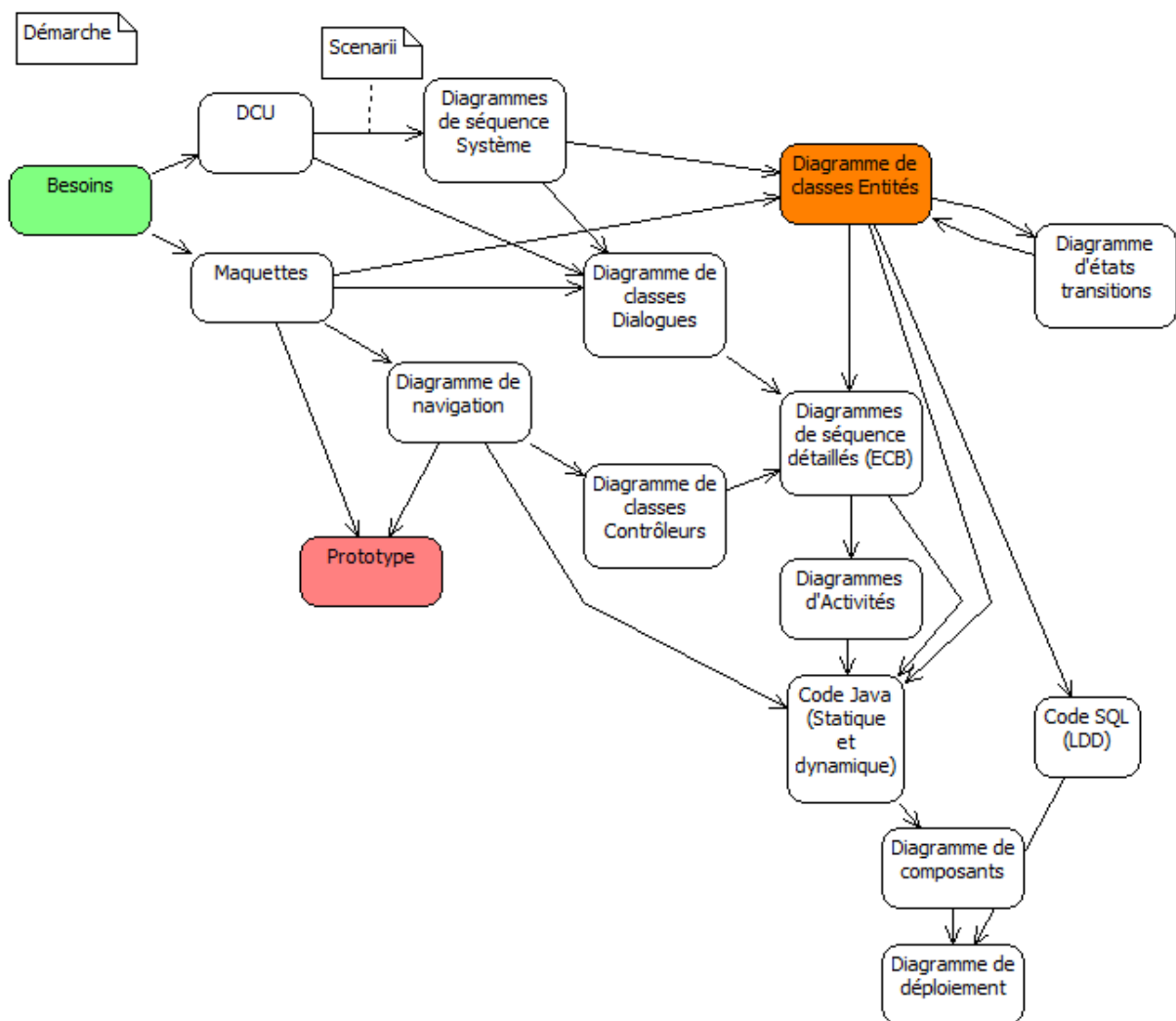


# UML

## Diagramme de Classes (DCL)



## Table des matières

Chapitre 1 - LE DIAGRAMME DE CLASSES (DCL).....	<a href="#">6</a>
1.1 - Présentation.....	<a href="#">7</a>
1.2 - Le concept de Classe.....	<a href="#">8</a>
1.2.1 - Définition.....	<a href="#">8</a>
1.2.2 - Représentation graphique.....	<a href="#">9</a>
1.2.3 - Visibilité.....	<a href="#">10</a>
1.2.4 - Déclarations des attributs et des opérations.....	<a href="#">11</a>
1.2.5 - Les attributs dérivés.....	<a href="#">12</a>
1.2.6 - Classe utilitaire.....	<a href="#">13</a>
1.2.7 - Classes et stéréotypes.....	<a href="#">14</a>
1.3 - Le concept d'association.....	<a href="#">15</a>
1.3.1 - Définition.....	<a href="#">15</a>
1.3.2 - Multiplicité.....	<a href="#">16</a>
1.3.3 - Nommage d'une association.....	<a href="#">19</a>
1.3.4 - Nommage des Rôles.....	<a href="#">20</a>
1.3.5 - Les associations réflexives.....	<a href="#">21</a>
1.3.5.1 - Définition.....	<a href="#">21</a>
1.3.5.2 - Représentation.....	<a href="#">21</a>
1.3.5.3 - Exemples.....	<a href="#">21</a>
1.3.5.4 - Exercices : hiérarchie des salariés et parrainage.....	<a href="#">22</a>
1.3.6 - Autres attributs d'une association.....	<a href="#">23</a>
1.3.6.1 - Tri.....	<a href="#">23</a>
1.3.6.2 - Navigabilité.....	<a href="#">24</a>
1.3.6.3 - La dépendance.....	<a href="#">25</a>
1.3.6.4 - La contrainte frozen.....	<a href="#">26</a>
1.3.6.5 - Qualification d'une association.....	<a href="#">27</a>
1.3.7 - Association à N dimensions.....	<a href="#">28</a>
1.3.8 - Classe-association.....	<a href="#">29</a>
1.3.9 - Contraintes sur associations.....	<a href="#">31</a>
1.3.9.1 - Association ordonnée (ordered).....	<a href="#">31</a>
1.3.9.2 - Sous-ensemble ou inclusion.....	<a href="#">32</a>
1.3.9.3 - Ou exclusif (XOR).....	<a href="#">33</a>
1.3.9.4 - Exercices.....	<a href="#">34</a>
1.4 - Exercices intermédiaires.....	<a href="#">35</a>
1.5 - Généralisation-Spécialisation : l'héritage.....	<a href="#">36</a>
1.5.1 - Définition.....	<a href="#">36</a>
1.5.2 - Représentation.....	<a href="#">37</a>

1.5.3 - Contraintes de spécialisation.....	38
1.5.4 - Diagrammes avec contraintes.....	40
1.5.4.1 - Totalité (Complétude et Non Distinction).....	40
1.5.4.2 - Partition (Complétude et Distinction).....	41
1.5.4.3 - Exclusion (Non Complétude et Distinction).....	42
1.5.4.4 - Standard (Non Complétude et Non Distinction).....	43
1.5.5 - Exemple de partition sur valeurs nulles.....	44
1.5.6 - Exemple de partition sur multiplicité.....	45
1.5.7 - Notions supplémentaires.....	46
1.6 - Exercices intermédiaires.....	47
1.7 - Agrégation et composition.....	48
1.7.1 - Agrégation.....	48
1.7.1.1 - Définition.....	48
1.7.1.2 - Représentation.....	48
1.7.2 - Composition.....	50
1.7.2.1 - Définition.....	50
1.7.2.2 - Représentation.....	50
1.7.2.3 - Exemple.....	51
1.7.2.4 - Exercice : classe Form.....	51
1.7.3 - Le Design pattern Composite du GoF.....	52
1.8 - Le concept d'interface.....	53
1.8.1 - Définition.....	53
1.8.2 - Représentation.....	54
1.8.3 - Interface et héritage : l'interface Tree de Java.....	55
1.8.4 - Le Design Pattern Factory.....	56
1.8.5 - Exercice : DAOs.....	58
1.9 - Dépendance.....	59
1.9.1 - Présentation.....	59
1.9.2 - Représentation.....	59
1.10 - Les énumérations.....	60
1.11 - Paquetage.....	61
1.12 - Le sens des flèches en UML.....	62
1.13 - UML et ECB (Entity, Control, Boundary).....	63
1.13.1 - Le modèle ECB détaillé.....	63
1.13.2 - Diagramme de classes participantes.....	64
Chapitre 2 - LA CONSTRUCTION DU DCL.....	66
2.1 - Les démarches.....	67
2.1.1 - Les 3 principales démarches.....	67
2.1.2 - Le cahier des charges.....	68
2.1.3 - Remarques.....	68
2.2 - La démarche par l'analyse du discours.....	69
2.3 - La démarche ascendante en détail.....	73
2.3.1 - Exemple : Bon de commande.....	74

2.3.2 - Le Dictionnaire des Données.....	75
2.3.3 - Les Dépendances Fonctionnelles (DF).....	76
2.3.4 - Matrice des DF.....	77
2.3.5 - Le Graphe des Dépendances Fonctionnelles (GDF).....	78
2.3.6 - Le DCL.....	81
2.4 - Les formes normales.....	85
2.4.1 - Première Forme Normale (1ère FN).....	86
2.4.2 - Deuxième Forme Normale (2ème FN).....	88
2.4.3 - Troisième Forme Normale (3ème FN).....	90
2.4.4 - Forme Normale de Boyce-Codd (BCNF).....	92
Chapitre 3 - LE DIAGRAMME DE CLASSES ET LA POO.....	94
3.1 - Tableau synthétique.....	95
3.2 - Quelques exemples avec Java et PHP.....	96
3.2.1 - Classe.....	96
3.2.2 - Association 1,*.....	97
3.2.3 - Association *,*.....	100
3.2.4 - Association *,* avec une classe association.....	102
3.2.5 - Association orientée.....	103
3.2.6 - Héritage.....	105
3.2.7 - Agrégation.....	107
3.2.8 - Composition.....	109
3.2.9 - Réflexive.....	110
3.2.10 - Interface.....	111
3.2.11 - Dépendance.....	113
Chapitre 4 - LE DIAGRAMME DE CLASSES ET SQL.....	114
4.1 - Quelques règles de passage du modèle Objet au modèle relationnel....	115
4.2 - Une classe.....	116
4.3 - Association binaire 1-*.....	117
4.4 - Association binaire 1-1.....	118
4.5 - Association binaire *-*.....	119
4.6 - Une classe-association.....	120
4.7 - Héritage.....	122
4.7.1 - UML.....	122
4.7.2 - SQL.....	123
4.7.2.1 - Distinction.....	123
4.7.2.2 - Push-Down.....	124
4.7.2.3 - Push-Up.....	125
4.8 - Agrégation et Composition.....	126
4.9 - Quelques contraintes.....	127
4.9.1 - Partition stagiaire-formateur.....	127
4.9.2 - DET état-civil.....	130
4.10 - Rétro-ingénierie.....	134
Chapitre 5 - ANNEXES.....	135

5.1 - Le modèle Cours.....	<a href="#">136</a>
5.1.1 - DCL (Diagramme de classes).....	<a href="#">136</a>
5.1.2 - Codes java.....	<a href="#">137</a>
5.1.2.1 - Utiliser StarUML pour générer du code Java.....	<a href="#">137</a>
5.1.2.2 - Code généré par StarUML !!!.....	<a href="#">139</a>
5.2 - Autres exercices.....	<a href="#">140</a>
5.2.1 - Chevaux.....	<a href="#">140</a>
5.2.2 - Bateaux.....	<a href="#">141</a>
5.2.3 - MusikScope.....	<a href="#">142</a>
5.2.4 - Elections.....	<a href="#">143</a>
5.3 - StarUML et les Design Patterns.....	<a href="#">144</a>

# **CHAPITRE 1 - LE DIAGRAMME DE CLASSES (DCL)**

## 1.1 - PRÉSENTATION

Le Diagramme De Classes est un diagramme statique.

Il existe deux concepts clés de base : la **classe** et l'**association**.

A cela il faut ajouter les concepts d'**héritage** et les **paquetages**.

Ainsi que le concept d'**interface**.

Plus les notions associées : encapsulation, polymorphisme, surcharge, ...

## 1.2 - LE CONCEPT DE CLASSE

### 1.2.1 - Définition

Une classe est une représentation abstraite d'un ensemble d'objets réels ou virtuels qui possèdent une structure, un comportement et des associations similaires (cf l'exemple des clients particuliers et des clients entreprises au paragraphe ).

Une classe est définie par un **nom**, des **attributs** (membres statiques appelés « propriétés » au niveau Objet et/ou programmation) et des **opérations** (membres dynamiques appelés « méthodes » au niveau Objet et/ou programmation).

On peut aussi dire qu'un objet a une **identité**, un **état** et un **comportement** (ou des responsabilités).

Les **attributs** – plus exactement les valeurs des attributs - représentent l'état d'un objet à un instant t. Fondamentalement les attributs sont privés ou protégés. L'accès aux valeurs des attributs ne s'effectue que via des méthodes.

C'est des principes de l'**encapsulation**.

Les **opérations** (abstraites) implémentées par des **méthodes concrètes** (implémentées par des procédures – void – ou des fonctions typées) modifient l'état d'un objet (principalement des constructeurs et les setters - modificateurs) ou rendent des services à d'autres classes, les getters – accesseurs – et autres méthodes de calcul.

Les règles de nommages sont les suivantes (issues du monde Java) :

- ✓ Tous les noms sont camélisés (sauf les noms des paquetages (packages) qui sont tout en minuscules et formés comme une uri ; par exemple fr.pb.utilitaires).
- ✓ Les noms des classes et des interfaces commencent par une majuscule.
- ✓ Les noms de tous les autres éléments (attributs, opérations, ...) commencent par une minuscule.



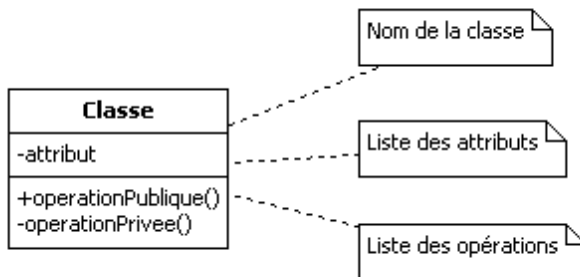
## 1.2.2 - Représentation graphique

Une classe est représentée graphiquement par un rectangle à trois compartiments.

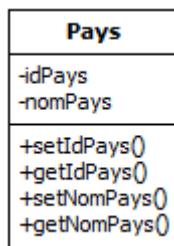
Le premier compartiment pour le **nom de la classe**,

Le deuxième pour les **attributs de la classe**,

Le troisième pour les **opérations de la classe**.



Exemple :



Notes :

Conception objet : attributs + opérations (attributes and operations)

Programmation : propriétés + méthodes

Par convention les attributs sont des noms et les méthodes des verbes.

Les noms des attributs devraient être uniques dans tout le modèle. Certains AGL rendent uniques dans le dictionnaire des données le nom des attributs du modèle.

### 1.2.3 - Visibilité

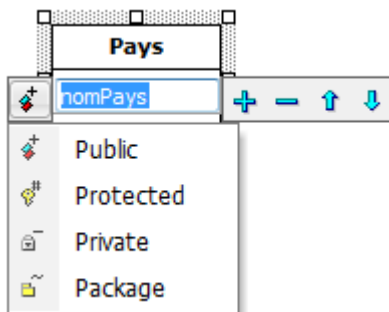
Les éléments d'une classe peuvent être **publics**, **protégés**, de **niveau paquetage** ou **privés**.

Un élément **public** (public) est visible dans tout le modèle. L'élément est précédé du symbole (+).

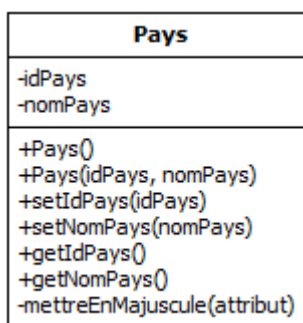
Un élément **protégé** (protected) est visible à l'intérieur de la classe et par ses descendants. L'élément est précédé du symbole (#).

Un élément de niveau **paquetage** (package) est visible par toutes les classes de son paquetage. L'élément est précédé du symbole (~).

Un élément **privé** (private) est seulement visible par la classe elle-même. L'élément est précédé du symbole (-).



- Exemple



Dans cet exemple tous les attributs sont privés. L'accès aux attributs est réalisé via des opérations publiques et/ou privées de visualisation et de mises à jour (les getters – accesseurs - et les setters – modificateurs).

Des opérations privées peuvent être présentes.

## 1.2.4 - Déclarations des attributs et des opérations

La syntaxe complète pour la déclaration d'un attribut est la suivante :

```
[visibilité] nom[cardinalité] [:type] [= valeur-initiale] [{contraintes}]
```

Exemples :

age	
-age : Integer = 0	Typé et initialisé
#age [0..1] : Integer	Multiplicité: le mini de 0 signifie facultatif
#dateDeNaissance : Date {frozen}	La valeur de l'attribut n'est pas modifiable
#motsCles [*] : String {addOnly}	Multiplicité: plusieurs valeurs ; la contrainte exprime le fait que la valeur ne peut être qu'ajouter, elle ne peut être ni modifiée, ni supprimée.

{addOnly} à utiliser lors de l'ajout d'une opération bancaire par exemple. L'annulation n'est pas la suppression. Il y aura un débit ou un crédit inverse.

(\*) plus de détails sur la contrainte {frozen} plus bas.

La syntaxe complète pour la déclaration d'une opération est la suivante :

```
[visibilité] nom[(paramètres)] [:type] [{contrainte}]
```

```
paramètres := [in | out | inout] nom [: type] [=default] [{contraintes}]
```

Exemples :

```
getAge()
+getAge() : Integer
-setDateDeNaissance(in dateDeNaissance : Date)
+getAge() : Integer {isQuery}
```

Note : {isQuery} est le résultat d'une requête.

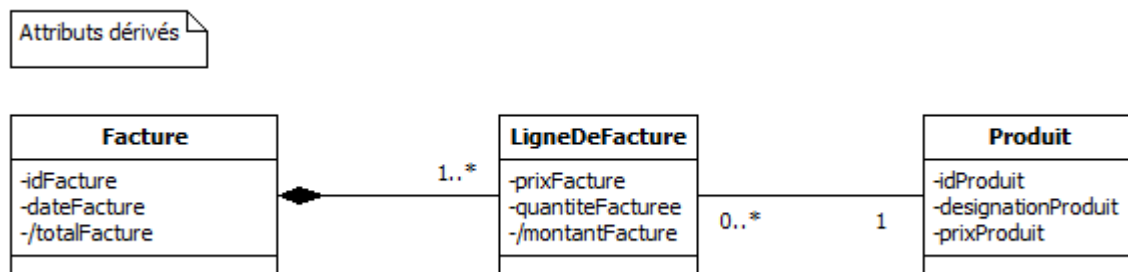
### 1.2.5 - Les attributs dérivés

Un attribut dérivé est un attribut dont la valeur s'obtient par calcul à partir de valeurs d'autres attributs. Le nom de l'attribut est précédé d'une barre oblique (/). Avec StarUML 5 il faut le faire « manuellement » alors que cela est possible dans StarUML 2.

Un attribut dérivé ne sera pas nécessairement mise en persistance. Cela dépendra de plusieurs facteurs, entre autres la complexité du calcul (si le calcul est complexe, donc coûteux, on aura tendance à rendre persistant le résultat du calcul).

Dans cet exemple le montant facturé est le résultat du calcul du prix facturé du produit par la quantité facturée et le total de la facture est le résultat du calcul de la somme des montants facturés.

Un attribut dérivé sera {readOnly}.

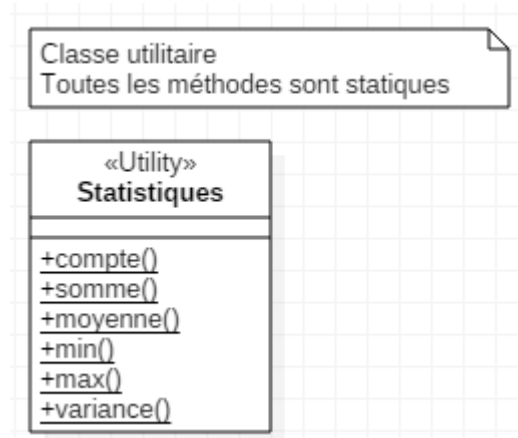


cf aussi le Design Pattern Observer du GoF et son implémentation avec des triggers au niveau de la BD.

Note : avec StarUML pour afficher les contraintes il faut sélectionner la classe, cliquer droit et sélectionner Format puis Show Properties.

### 1.2.6 - Classe utilitaire

Tous les membres ont une portée de classe (et non pas d'instance).  
Les attributs et les opérations sont des variables et des méthodes globales.  
La classe ne contient que des membres statiques.  
La classe n'a pas à être instanciée.



### 1.2.7 - Classes et stéréotypes

Ivar Jacobson propose 3 stéréotypes : Entity et Control et Boundary.

Boundary : classe de frontière. Les objets de ces classes sont visibles et ils fournissent un moyen de communication entre le système et l'extérieur.

Entity : classe entité. Les objets de ces classes représentent les données du domaine de l'étude. Elles sont généralement persistantes.

Control : classe contrôle. Les objets de ces classes sont internes au système. Elles contrôlent le comportement des cas d'utilisation. Les classes représentent une activité de contrôle, une règle de gestion, un « chef d'orchestre », ...

Vous pouvez par ailleurs créer vos propres stéréotypes.

## 1.3 - LE CONCEPT D'ASSOCIATION

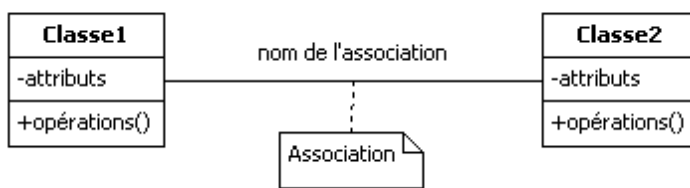
### 1.3.1 - Définition

Une association est un lien entre classes. Elle exprime une relation entre deux classes.

Une association est éventuellement identifiée par un nom.

Par défaut une association est bidirectionnelle (cf le paragraphe sur la navigabilité).

Une association est représentée par un trait plein.



Note : une association est un des quatre types de relation d'UML. Les autres étant la généralisation, la dépendance, la réalisation.

### 1.3.2 - Multiplicité

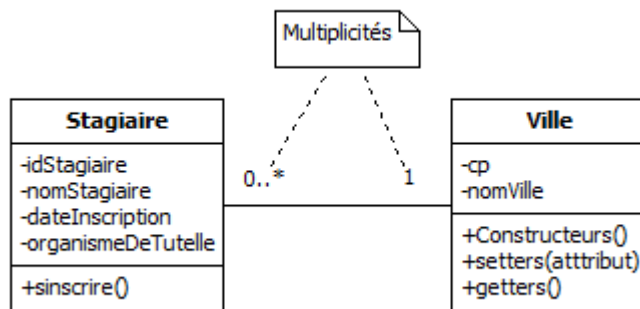
La multiplicité exprime la participation d'une classe à l'association, le nombre de fois où un objet d'une classe est relié à un autre objet d'une autre classe.

Les valeurs possibles sont les suivantes :

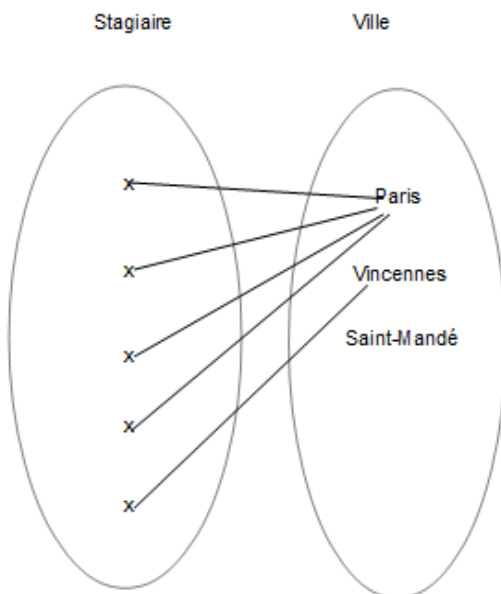
Multiplicité	Description
0..* (ou *)	De 0 à plusieurs
1..*	De 1 à plusieurs
0..1	De 0 à 1
1 (ou 1..1)	Un et un seul
n,m	Liste de valeurs possibles. De n à m.
n	Une seule valeur

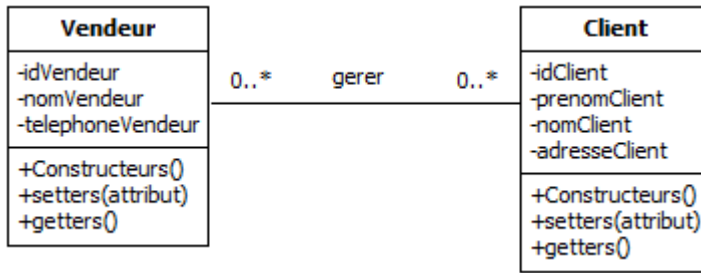


- Exemples



Le modèle Stagiaire-Ville signifie ceci : un **stagiaire habite** une et une seule **ville** et dans une **ville** il peut y avoir de zéro à N **stagiaires**.





Le modèle Client–Vendeur signifie ceci : un **client est suivi** par aucun ou plusieurs vendeurs et un vendeur **a dans son portefeuille** aucun ou plusieurs clients.

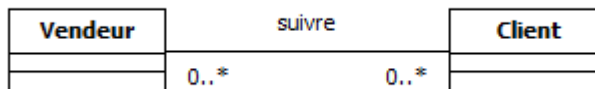
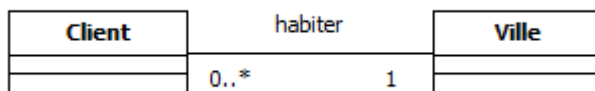
### 1.3.3 - Nommage d'une association

Le nommage – facultatif - améliore la lisibilité du modèle. Il devient obligatoire s'il existe plusieurs associations entre 2 mêmes classes (cf un exemple au paragraphe suivant).

Il est fait sous forme verbale. Il apparaît sur le lien de l'association.

Il est possible de préciser le sens de la lecture de l'association avec une flèche à la fin du verbe (à ne pas confondre avec la navigabilité cf plus loin).

Un client habite une ville (la lecture de gauche à droite est souvent suffisante et ne nécessite donc pas la précision du sens de la lecture ; la lecture en sens inverse emploie le plus souvent la forme passive).



### 1.3.4 - Nommage des Rôles

L'extrémité d'une association s'appelle un rôle.

Une association binaire possède deux rôles.

Un rôle est exprimé sous forme nominale.

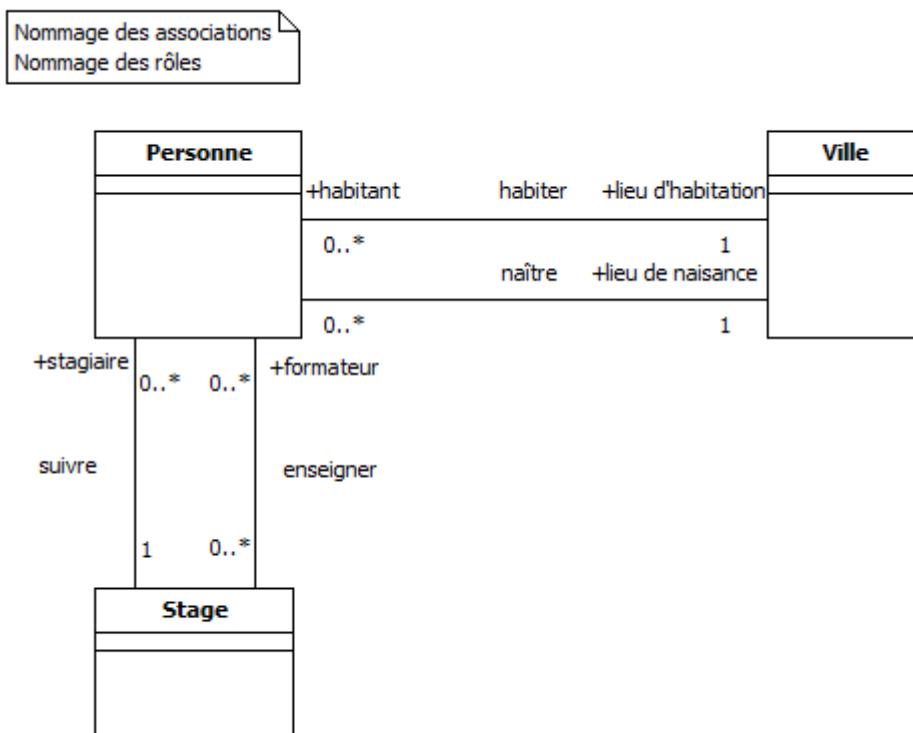
Le nommage des rôles montre comment une classe voit une autre classe.

Le nommage des rôles est souvent exclusif du nommage de l'association.

Le nommage des rôles n'est pas toujours nécessaire.

Il devient nécessaire lorsque l'association est réflexive.

Le nommage des rôles permet de distinguer la fonction de chaque participant.



Dans cet exemple le nommage permet de distinguer le lieu d'habitation du lieu de naissance.

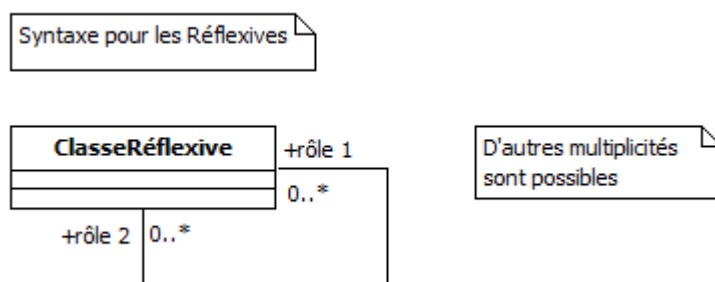
Et aussi le fait qu'une personne soit stagiaire ou formateur mais le nommage de l'association pourrait suffire.

### 1.3.5 - Les associations réflexives

#### 1.3.5.1 - Définition

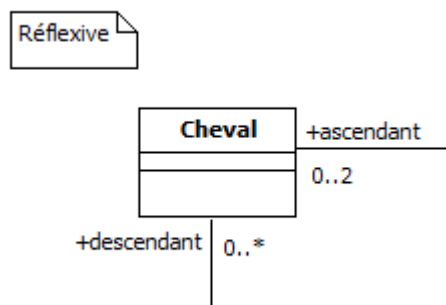
Une association réflexive est une association sur la même classe.  
Sur une association réflexive le nommage des rôles est obligatoire autrement le diagramme est illisible.

#### 1.3.5.2 - Représentation



#### 1.3.5.3 - Exemples

Un cheval est descendant de 2 chevaux (mais ...) .  
Un cheval est ascendant de plusieurs chevaux ou d'aucun cheval.

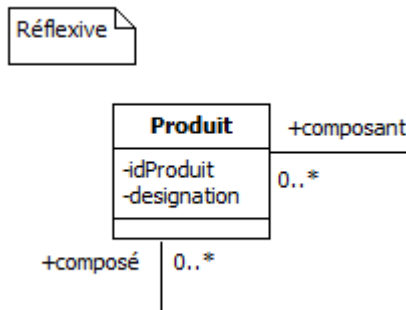


Note : attention à la position du rôle (à côté de la classe « source »).  
Un cheval, en tant qu'ascendant, a aucun ou plusieurs descendants.  
Un cheval, en tant que descendant, a aucun ou 2 parents.

**Autre exemple :**

Un produit est composé de 0 à N composants.

Un produit est composant de 0 à N composés ou composites.



mais il est préférable de modéliser ce dernier en utilisant le Design Pattern Composite du GoF.

#### 1.3.5.4 - Exercices : hiérarchie des salariés et parrainage

1 - Représentez les rapports hiérarchiques directs entre salariés dans une organisation de type « fordiste ».

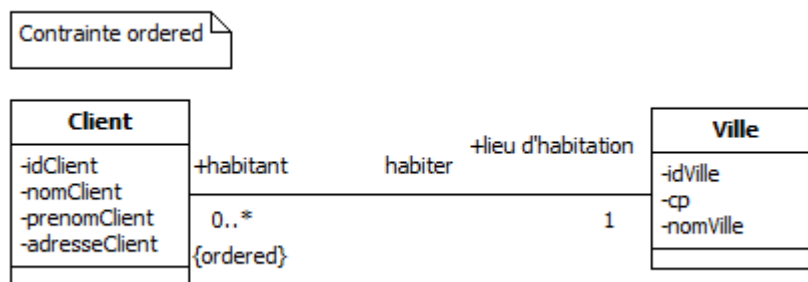
2 - Représentez le système de parrainage entre clients (site 3 Suisses).

### 1.3.6 - Autres attributs d'une association

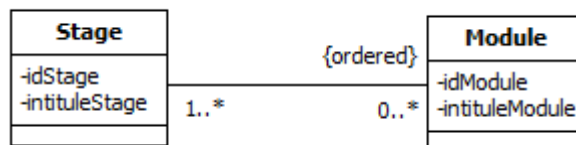
#### 1.3.6.1 - Tri

Une association peut être triée ou ne pas être triée.

Dans cet exemple les clients sont triés en fonction de leur ville.



Le déroulé d'un stage suit un certain ordre.



Note : avec StarUML pour afficher les contraintes il faut sélectionner l'association, cliquer droit et sélectionner Format puis Show Properties.

### 1.3.6.2 - Navigabilité

- **Définition**

La navigabilité c'est le fait que l'on puisse accéder à un objet d'une classe à partir d'un autre objet. Par défaut une association est navigable dans les 2 sens.

- **Exemples**

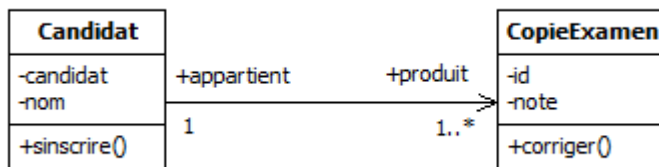
#### Bidirectionnelle :

Dans le diagramme précédent (Client-Ville) l'association est navigable dans les deux sens : il est possible d'accéder à une ville via le client et à un client via la ville.

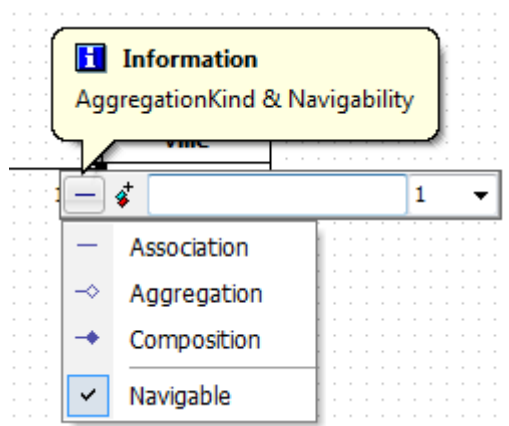
Ce qui signifie en POO que la classe Ville aura un attribut de type List<Ville> et que la classe Client aura un attribut de type Ville.

#### Unidirectionnelle :

Dans cet exemple une copie d'examen appartient à une personne mais la personne ne peut être identifiée à partir de la copie. L'association est représentée avec un trait plein et une flèche ouverte.



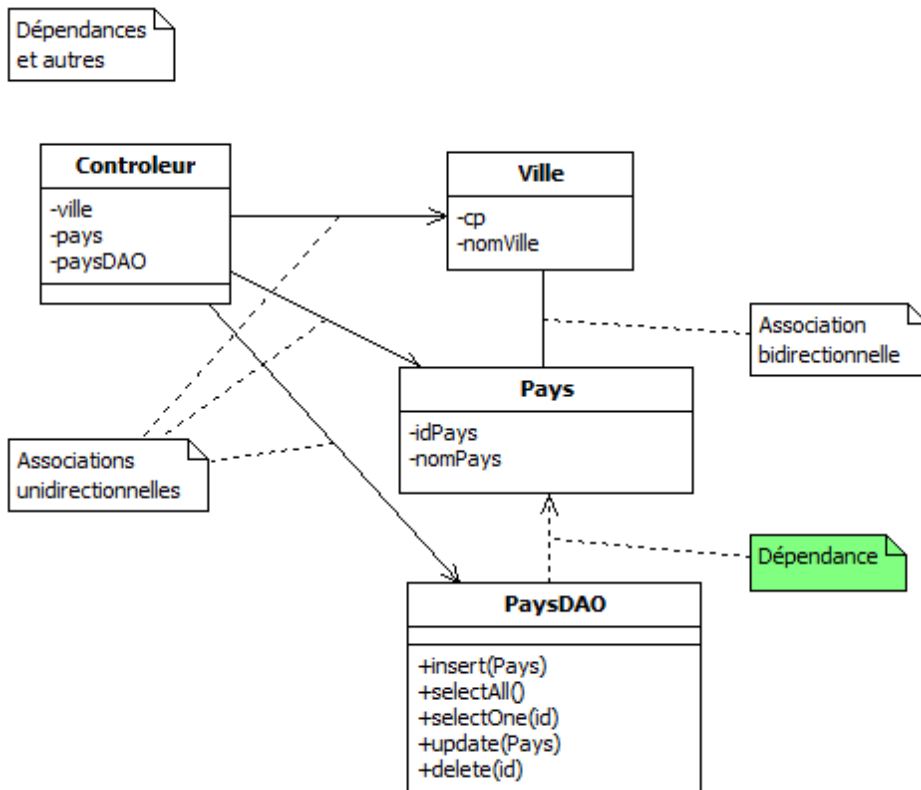
Note : pour éliminer certaines dépendances - surtout pour des classes appartenant à des packages différents - il faut rendre les associations navigables dans un seul sens. L'association de base avec sa navigabilité bi-directionnelle implique des dépendances réciproques.





### 1.3.6.3 - La dépendance

Il y a dépendance entre 2 classes lorsque qu'une classe utilise durant son exécution une autre classe. Le fait qu'une opération d'une classe ait un paramètre du type d'une autre classe crée un lien de dépendance et aussi le fait qu'un attribut soit du type d'une autre classe.



Note : si l'on compare à l'association simple ou à la composition (cf plus loin) la dépendance est une relation moins forte entre deux classes et dont la durée est plus courte puisqu'elle n'existe que durant l'exécution du code.

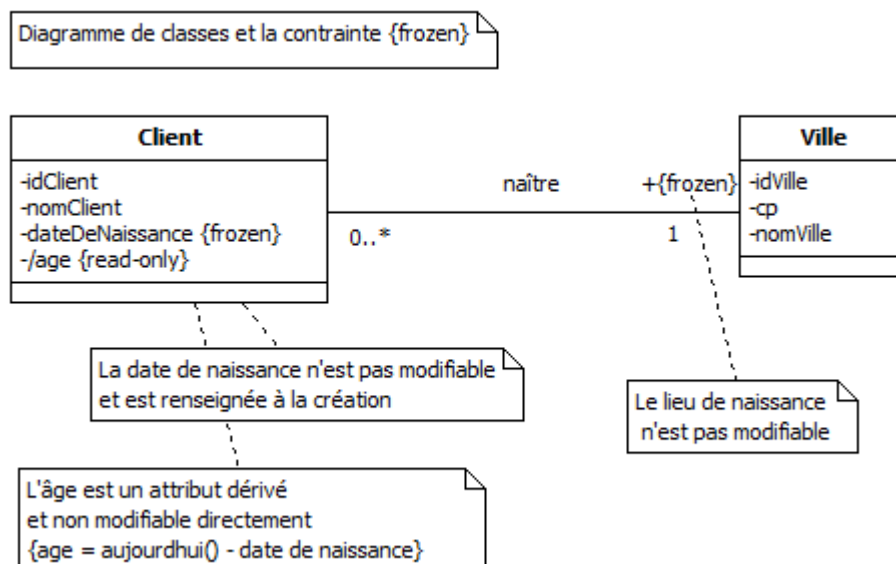
### 1.3.6.4 - La contrainte frozen

Cette contrainte précise qu'une **association** est unique et irréversible (non modifiable).

Pour un **attribut** elle indique que la valeur est non modifiable. L'attribut est donc NOT NULL et la valeur est attribuée à l'instanciation de l'objet ; il existe donc un paramètre pour cet attribut dans le constructeur et il n'existe pas de setter.

A noter la différence entre {frozen} et {read-only} ; une contrainte de type {read-only} précise que la valeur d'un attribut ne peut être modifiée directement mais peut l'être indirectement. C'est le cas d'un attribut dérivé comme l'âge, le total d'une facture, etc.

Exemple : date de naissance et lieu de naissance.



La lecture :

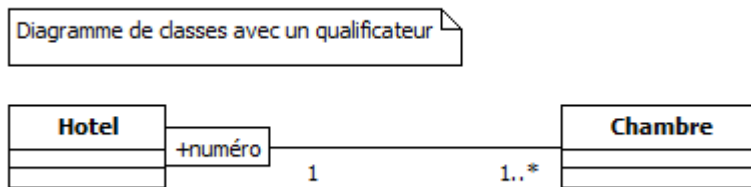
la date de naissance n'est pas modifiable,

l'âge est obtenu par calcul et n'est pas modifiable directement, c'est un trigger qui modifie la valeur, un client est né dans une ville et cela n'est pas modifiable.

Note : un élément {frozen} peut être valorisé dans un Constructeur ou bien via un « setter-singleton ».

### 1.3.6.5 - Qualification d'une association

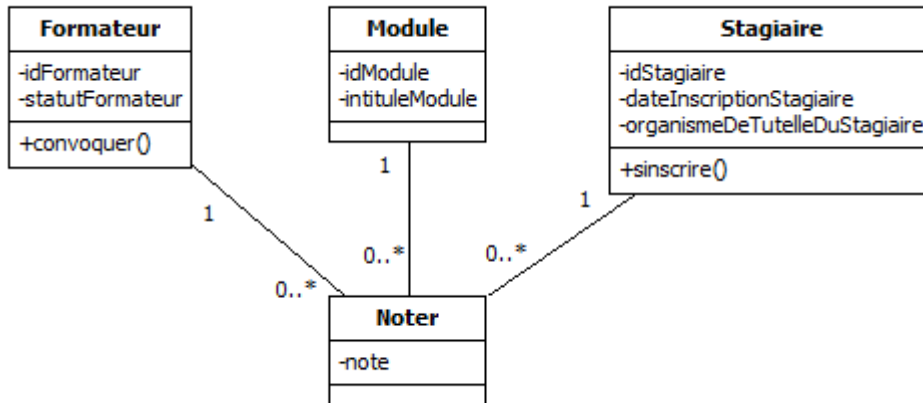
L'ajout d'un qualificateur permet d'identifier de façon unique un objet en fonction d'un autre. C'est une identification relative.



Note : dans une BD la clé primaire de chambre est composite et donc à moitié formée par la FK correspondant à la PK de hôtel.

### 1.3.7 - Association à N dimensions

Certains auteurs ont proposé des associations avec une arité supérieur à 2.

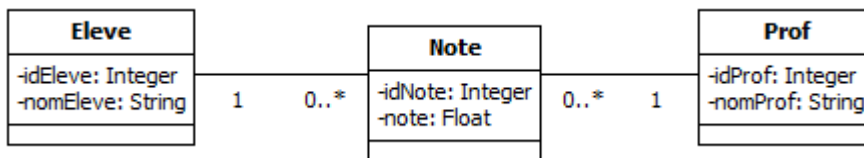


Mais il est préférable de créer une classe-association et de créer éventuellement une contrainte de simultanéité lorsque l'arité est de 3 ; au-delà la classe-association est impossible.

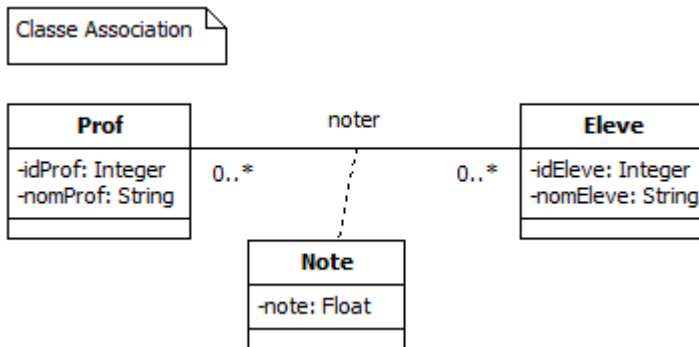
### 1.3.8 - Classe-association

Pour les associations \*,\* qui portent des attributs il est préférable de créer une classe association car l'attribut doit être déplacé parce qu'il ne peut l'être ni sur une classe ni sur l'autre.

Les élèves sont notés par des profs.

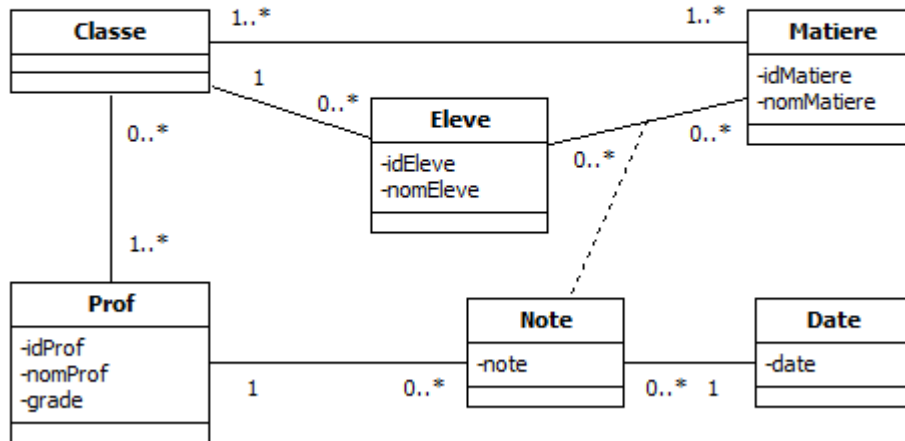


qui devient



ou encore ...

Quatenaire : un élève est noté par un prof dans une matière à une date précise  
Représentation avec des binaires

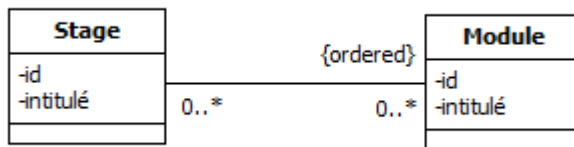


### 1.3.9 - Contraintes sur associations

#### 1.3.9.1 - Association ordonnée (ordered)

Cette contrainte exprime le fait que les instances doivent être en association ordonnée.

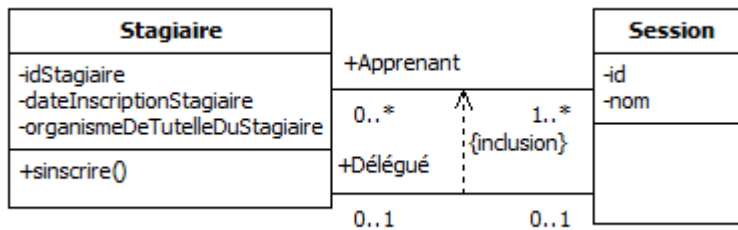
Exemple : Les modules d'un stage doivent avoir un ordre certain.



### 1.3.9.2 - Sous-ensemble ou inclusion

Il y a inclusion {subset} lorsqu'une instance d'une association existe entre 2 classes et qu'elle ne peut l'être que si une autre instance existe dans une autre association.

Dans une session pour être délégué il faut être inscrit dans cette session.



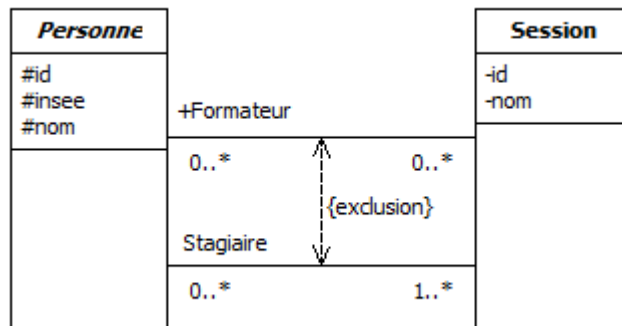
Note : utilisation du Name pour la contrainte avec StarUML.



### 1.3.9.3 - Ou exclusif (XOR)

Il y a exclusion ou partition lorsqu'une instance d'une association existe entre 2 classes et qu'elle ne peut exister dans une autre association.

Dans une session de formation une personne est soit Formateur soit Stagiaire.



Note :

La table de vérité du XOR

A	B	A XOR B
VRAI	VRAI	FAUX
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX

Le résultat est VRAI si un et un seul des opérandes A et B est VRAI (soit l'un, soit l'autre).

Le résultat est VRAI si les deux opérandes A et B ont des valeurs distinctes.

Le résultat est VRAI si un nombre impair d'entrées est vrai.

#### 1.3.9.4 - Exercices

1 - Pour un produit donné une usine achète soit à un fournisseur externe soit à un fournisseur interne.

2 - Un établissement se fournit auprès d'un fournisseur externe s'il s'agit d'un produit fini ou auprès d'un établissement de l'entreprise s'il s'agit d'un produit intermédiaire.

## **1.4 - EXERCICES INTERMÉDIAIRES**

Au choix ...

- 1) Reprenez le DCU du site marchand et les maquettes et faites le DCL.
  
- 2) Reprenez le DCU du SDP et les maquettes et faites le DCL.
  
- 3) Si la démarche ascendante a été vue : CinéScope.

## 1.5 - GÉNÉRALISATION-SPÉCIALISATION : L'HÉRITAGE

### 1.5.1 - Définition

La **généralisation** est un processus qui permet de créer une classe générique à partir de plusieurs classes. On prend des classes existantes et on crée de nouvelles classes qui regroupent les parties communes.

Son corollaire, la **spécialisation**, est un processus qui permet de créer des sous-ensembles de classes à partir d'une seule classe. On part de classes existantes et par dérivation on crée de nouvelles classes plus spécialisées en ne spécifiant que les différences.

Elles permettent de créer des associations de type classification.

La généralisation-spécialisation implique le concept d'**héritage** : les sous-classes (classes enfants) héritent des *attributs* et des *opérations* de la sur-classe (classe parent).

La **généralisation** est une abstraction.

La **spécialisation** se résume au fait que l'on peut remplacer le mécanisme d'héritage par "*est un*" ou "*est une sorte de*". La spécialisation est une spécification.

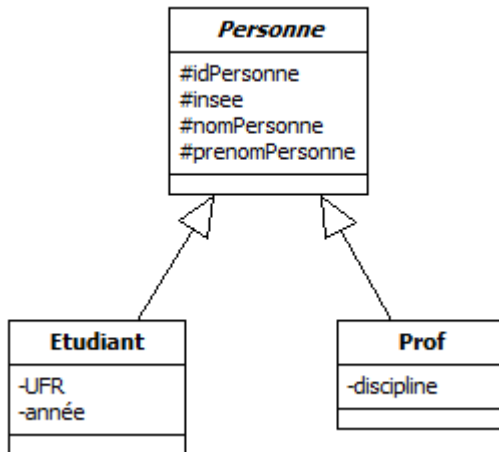
La généralisation est transitive.

Les indices ou critères qui permettent de généraliser ou de spécialiser sont les suivants :

- ✓ factorisation (généralisation),
- ✓ classification (spécialisation),
- ✓ attributs nuls structurels (spécialisation),
- ✓ multiplicité structurelle mini de 0 sur une association (spécialisation).

## 1.5.2 - Représentation

L'héritage est représentée par un trait plein et une flèche vide fermée orientée de la classe spécifique vers la classe générique.



La classe *Personne* est une classe abstraite.

Un étudiant **est une** *Personne*, un prof **est une** *Personne*.

Par défaut la généralisation permet le chevauchement (un objet réel peut être instancié dans 2 sous-classes : un doctorant est un étudiant mais il peut aussi enseigner) {Chevauchement} ou {Inclusion}.

Elle n'implique pas la contrainte d'exclusivité : {Exclusion} ou {Disjonction}.

Dans l'exemple ci-dessus un Prof peut être un Etudiant et vice-versa.

Par ailleurs il est possible de poser la contrainte {Complète} ou {Incomplète} pour préciser la possibilité de rajouter des classes spécifiques ou pas.

Donc deux types de contraintes sont liées à l'héritage :

- ✓ la distinction (chevauchement, disjonction),
- ✓ la complétude (couverture).

### 1.5.3 - Contraintes de spécialisation

Il est possible de prendre en compte les contraintes de **complétude** (est-ce que tous les objets de la classe générique sont aussi des objets spécifiques ?) et de **distinction** (est-ce qu'un objet d'une classe spécifique peut aussi être présent dans une autre classe spécifique ? Ou en d'autres mots est-ce que l'intersection est vide ?) :

Cette double distinction implique quatre contraintes :

Distinction	OUI	NON
Complétude		
OUI	XT (Partition)	T (Totalité)
NON	X (Exclusion)	

Complétude + Distinction = **Partition** (XT) : un objet parent appartient nécessairement à un sous-ensemble et n'appartient qu'à un seul sous-ensemble (un Salarié est soit Cadre soit Employé). La somme des objets des sous-ensembles est **égale** au nombre des objets de l'ensemble.

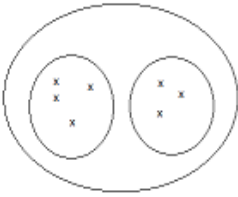
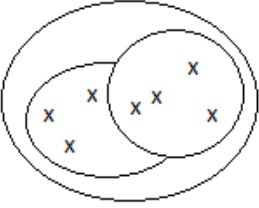
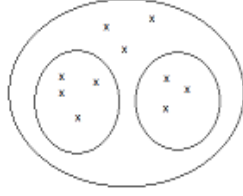
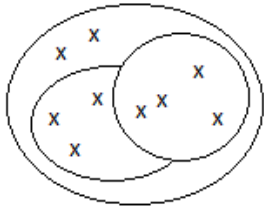
Complétude + Non Distinction = **Totalité** (T) : un objet parent appartient nécessairement à un sous-ensemble et peut appartenir à plusieurs sous-ensembles (dans le cadre d'une publication cinématographique il y a des acteurs et des réalisateurs et certaines personnes sont les deux à la fois). La somme des objets des sous-ensembles est **supérieure** au nombre des objets de l'ensemble.

Non Complétude + Distinction = **Exclusion** (X) : un objet parent n'appartient pas nécessairement à un sous-ensemble mais n'appartient qu'à un seul sous-ensemble (une Personne est soit Formateur soit Stagiaire soit autre ; par exemple le personnel administratif, mais il ne s'agit pas d'un sous-ensemble). La somme des objets des sous-ensembles est **inférieure** au nombre des objets de l'ensemble.

Non Complétude + Non Distinction.

La somme des objets des sous-ensembles est éventuellement **supérieure** au nombre d'objets de l'ensemble (une Personne est Enseignant ou Etudiant ou Enseignant/Chercheur ou Autre).

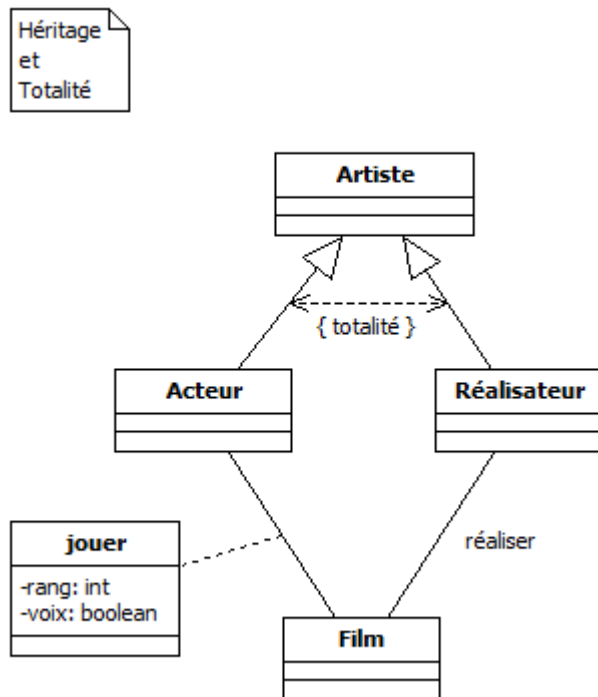
Note : Merise utilise les termes de Couverture et de Disjonction !

Partition (XT) Complétude + Distinction		Formateurs et Stagiaires dans le sous-domaine pédagogique d'un centre de formation. Une personne est soit Formateur, soit Stagiaire. La classe Personne est abstraite.
Totalité (T) Complétude + Non Distinction		Profs et Etudiants dans le sous-domaine pédagogique de la Fac. La classe Personne est abstraite.
Exclusion (X) Non Complétude + Distinction		Formateurs, Stagiaires et secrétaires, commerciaux, ... d'un centre de formation. Une personne est soit Formateur, soit Stagiaire, soit autre sans qu'il soit nécessaire de créer une autre classe spécifique. La classe Personne est concrète.
Non Complétude et Non Distinction		Profs, étudiants, appariteurs, secrétaires, ... à la Fac. Une Personne est soit Prof, soit Etudiant, soit autre sans qu'il soit nécessaire de créer une autre classe spécifique. La classe Personne est concrète.

## 1.5.4 - Diagrammes avec contraintes

### 1.5.4.1 - Totalité (Complétude et Non Distinction)

Un artiste, dans le monde du cinéma, peut être acteur ou réalisateur ou les deux.  
La classe Artiste est Abstraite.

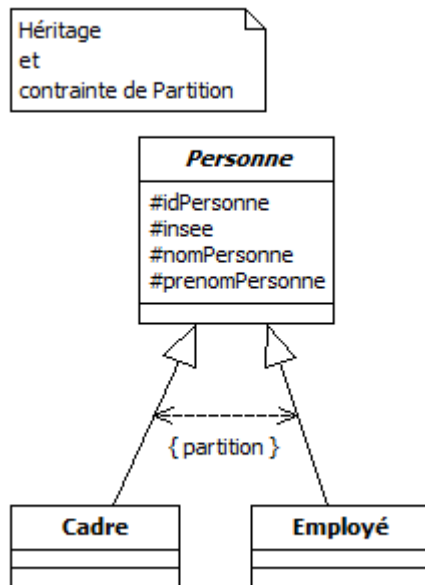


Note : dans le cadre de CinéScope un Artiste est soit Acteur soit Realisateur mais parce que dans Pariscope il y a aussi des Compositeurs et des Interprètes et c'est pour cela que la classe Artiste est Concrète.



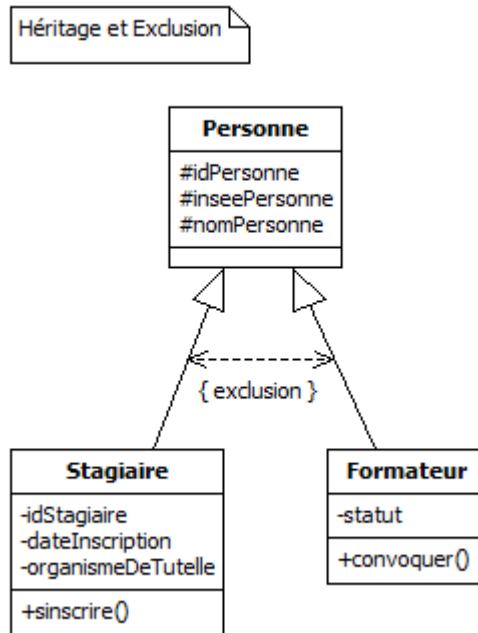
## 1.5.4.2 - Partition (Complétude et Distinction).

Dans cette entreprise de services il n'y a que des cadres et des employés.  
La classe Personne est Abstraite.



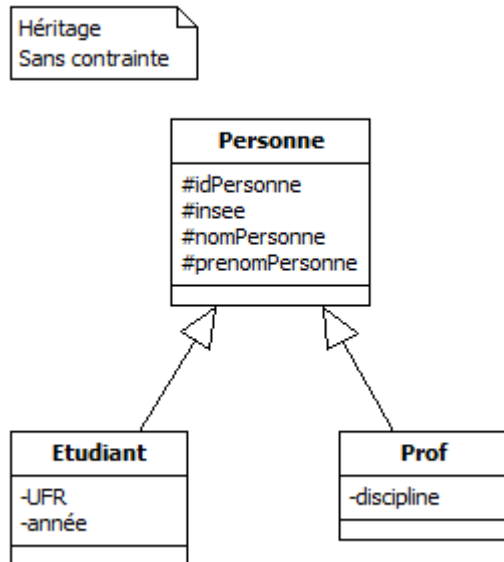
## 1.5.4.3 - Exclusion (Non Complétude et Distinction)

Dans un centre de formation il y a des stagiaires, des formateurs et le personnel administratif et commercial. Chacun a une fonction et une seule.  
La classe Personne est concrète.



## 1.5.4.4 - Standard (Non Complétude et Non Distinction)

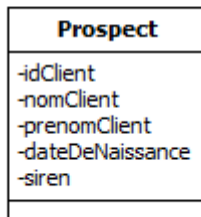
A la Fac il y a des étudiants, des profs et les autres.  
Certains étudiants font de l'enseignement (les doctorants).  
La classe Personne est concrète.



### 1.5.5 - Exemple de partition sur valeurs nulles

#### Avant

Heritage et partition (version a)



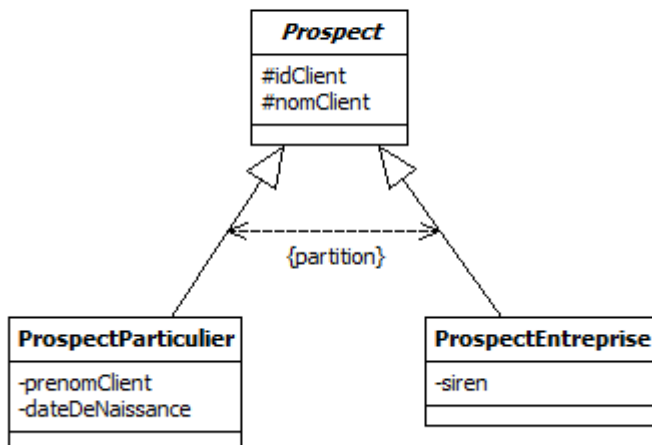
Pour certains prospects le prénom et la date de naissance n'ont pas de sens.

Pour d'autres c'est de code SIREN qui n'a pas de sens.

En fait il y a deux catégories bien distinctes de prospects : les particuliers et les entreprises.

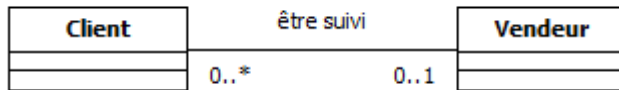
#### Après

Heritage et partition (version b)



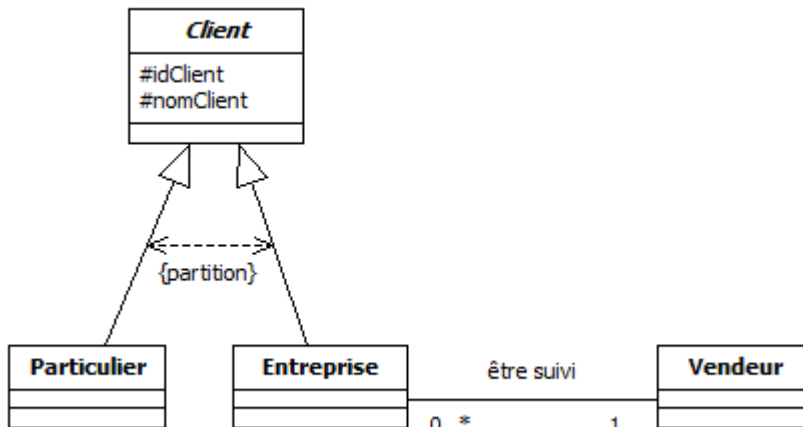
### 1.5.6 - Exemple de partition sur multiplicité

Avant



La multiplicité mini de 0 côté vendeur pose question.  
En fait certains clients n'ont pas de vendeur attribué.

Après



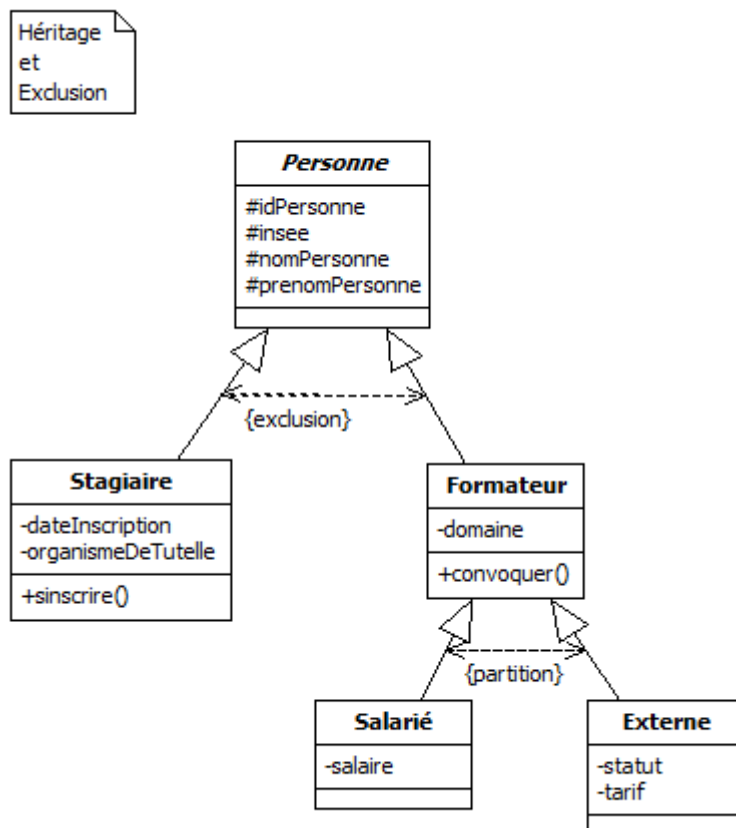
## 1.5.7 - Notions supplémentaires

- **Classe abstraite**

Une classe abstraite est une classe non instanciable (comme la classe *Personne*).  
Son nom est en italique.

- **Classe finale**

Une classe finale est une classe non-héritable.



La classe *Personne* est abstraite.

Les classes **Stagiaire**, **Salarié** et **Externe** sont **finales**.

Exclusion : non complétude et disjonction (un formateur n'est pas un stagiaire et vice-versa et il existe des salariés qui ne sont pas formateur).

Partition : un formateur est soit salarié soit externe (en freelance).

## **1.6 - EXERCICES INTERMÉDIAIRES**

Reprenez CinéScope et appliquez la généralisation-spécialisation ... si nécessaire.

## 1.7 - AGRÉGATION ET COMPOSITION

### 1.7.1 - Agrégation

#### 1.7.1.1 - Définition

Formulation simplifiée : un agrégat est composé d'autres éléments qui sont des composants.

L'agrégation se résume au fait que l'on peut remplacer le mécanisme par *"est une partie de"* en ce qui concerne le composant et par *"a un"* ou *"contient"* en ce qui concerne le composé.

Par exemple :

- un disque dur **est une partie d'**un ordinateur,
- un ordinateur **a un** disque dur.

C'est une association asymétrique entre deux ou plusieurs classes. La classe agrégative joue un rôle dominant dans l'agrégation.

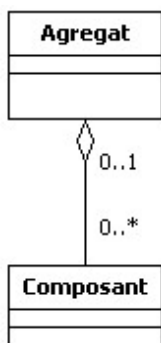
L'agrégation est toujours remplaçable par une association simple.

C'est une **relation faible**.

Elle est impliquée par les facteurs suivants :

- ✓ une classe fait partie d'une autre classe,
- ✓ les valeurs des attributs se propagent,
- ✓ les actions se propagent,
- ✓ les classes sont subordonnées aux autres classes.

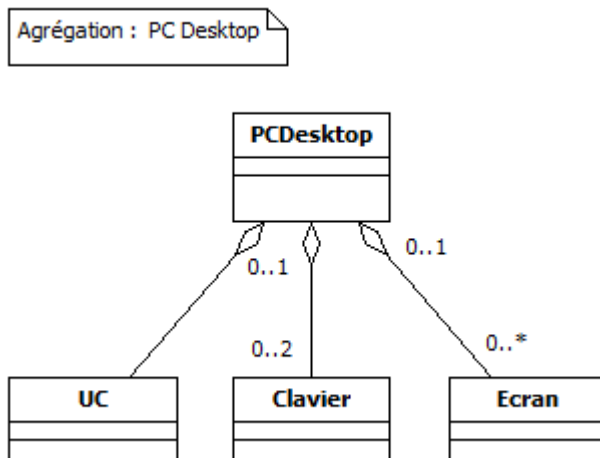
#### 1.7.1.2 - Représentation



Note : cf le pattern Composite et la relation de Composition pour d'autres modélisations et exemples (pensez à la relation Dossier et Fichier).

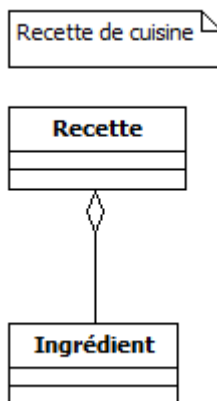


- Exemple



Note : il s'agit bien d'un ordinateur, et non pas d'une unité centrale. Un ordinateur (serveur) peut justement n'être composé que d'une unité centrale. Les multiplicités côté composants pourraient (devaient surtout dans le cas de serveurs) être de 0..\*.

Autre exemple : recette de cuisine.



## 1.7.2 - Composition

### 1.7.2.1 - Définition

Formulation simplifiée : un élément composite est composé d'éléments composants. Les éléments composants dépendent de l'élément composite.

Par exemple les éléments d'une fenêtre (Frame de type Swing, éléments d'un formulaire HTML).

**Il y a une relation de contenance.**

La composition est un **agrégation forte**.

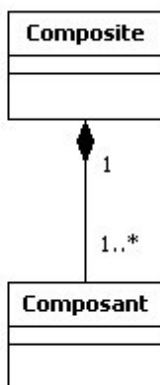
La multiplicité du côté du composite est de 1. Le composant appartient au composite.

Le losange de l'association est rempli, il est noir.

La destruction du composite entraîne la destruction des composants.

La notation par composition est identique à la représentation par attribut. L'unique différence est que les classes "composantes" peuvent être en relation avec d'autres classes.

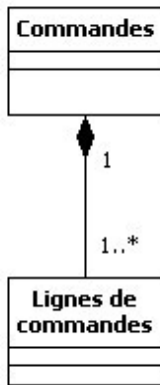
### 1.7.2.2 - Représentation



Note : la multiplicité côté Composite est facultative puisque toujours égale à 1.

### 1.7.2.3 - Exemple

Une commande est composée d'au moins une ligne de commande. Une ligne de commande appartient nécessairement à une et une seule commande.



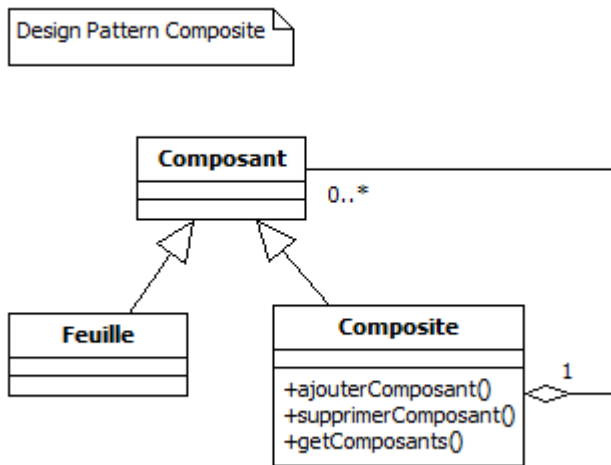
Autres exemples : un PC-Portable, les éléments d'une fenêtre, les éléments d'un formulaire, ...

### 1.7.2.4 - Exercice : classe Form

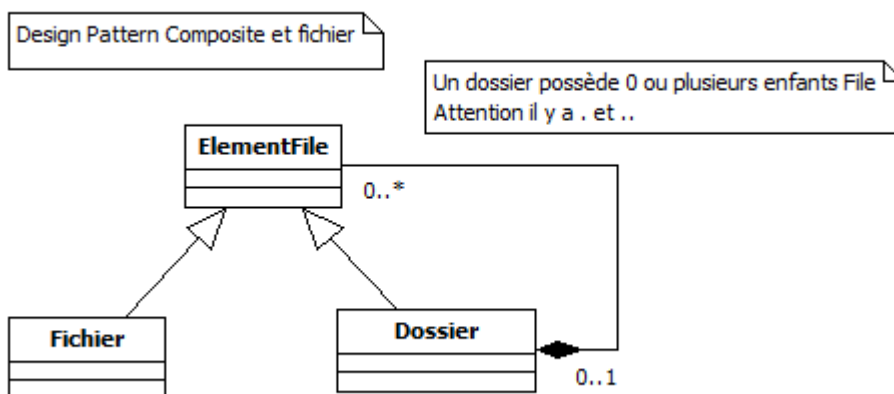
1 - Représentez la classe Form HTML.

2 – La machine à café.

### 1.7.3 - Le Design pattern Composite du GoF



Exemple :



Pensez aussi aux recettes de cuisine ...

## 1.8 - LE CONCEPT D'INTERFACE

### 1.8.1 - Définition

Une interface définit une abstraction et un contrat.

Une interface peut hériter d'une autre interface.

Une classe qui implémente une interface dépend de l'interface.

Une classe peut posséder et dépendre de plusieurs interfaces.

Les interfaces ne possèdent ni attributs, ni états, ni associations.

- **Une abstraction**

Une interface spécifie une collection d'opérations qui peuvent être utilisées pour définir un service.

Elle spécifie les opérations visibles d'une classe ou d'un composant. Elle peut ne spécifier qu'une partie du comportement d'une classe.

Dans ce cadre c'est une « fenêtre », une « view », sur une classe. L'interface est en relation de « fourniture » avec la classe.

- **Un contrat**

L'interface définit un contrat.

La classe qui implémente une interface doit implémenter le code spécifié dans l'interface.

C'est-à-dire qu'une classe « se conforme » à l'interface en question.

- **La dépendance**

Ceci découle du contrat. Une classe implémentant une ou plusieurs interfaces dépend de la ou des Interfaces.

- **Remarque**

Toute classe devrait proposer au moins une interface ; elle peut en proposer plusieurs.

C'est le cas de la classe ArrayList avec l'interface List en Java, ou encore de la classe TreeSet avec l'interface Set.

Cf SQL\_JAVA.odt avec l'interface IDAOPays et la classe PaysImpl;

Une classe peut implémenter plusieurs interfaces.

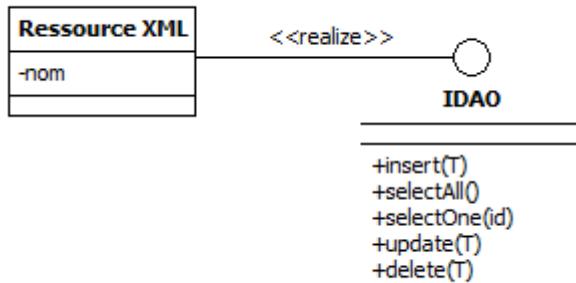
Cf, par exemple, la classe TreeSet de Java qui implémente les interfaces Serializable, Cloneable, Iterable<E>, Collection<E>, NavigableSet<E>, Set<E>, SortedSet<E>.

<http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>

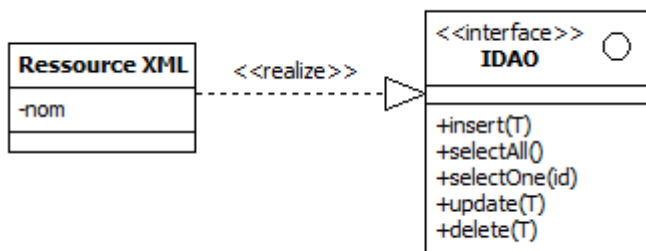
<http://docs.oracle.com/javase/7/docs/api/java/util/Set.html>

## 1.8.2 - Représentation

UML représente les interfaces par un petit cercle relié par un trait à la classe en mode iconique.

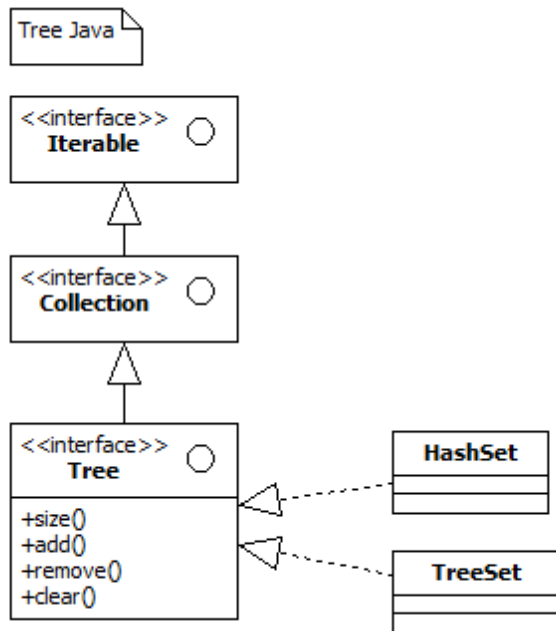


Une interface peut être représentée au moyen d'une classe d'interface stéréotypée, et l'implémentation au moyen d'un trait discontinu terminé par une flèche fermée vide allant de la classe vers l'interface.  
 Note : Le sens de la flèche est le même que celui de l'héritage.



### 1.8.3 - Interface et héritage : l'interface Tree de Java

Une Interface peut étendre une autre Interface (extends).



L'interface `Tree` de Java, que les classes `HashSet` et `TreeSet` implémentent, hérite de l'interface `Collection` qui elle-même hérite de l'interface `Iterable`.

### **1.8.4 - Le Design Pattern Factory**

Consultez le Design Pattern DAO de SUN ainsi que le Design Pattern Factory.

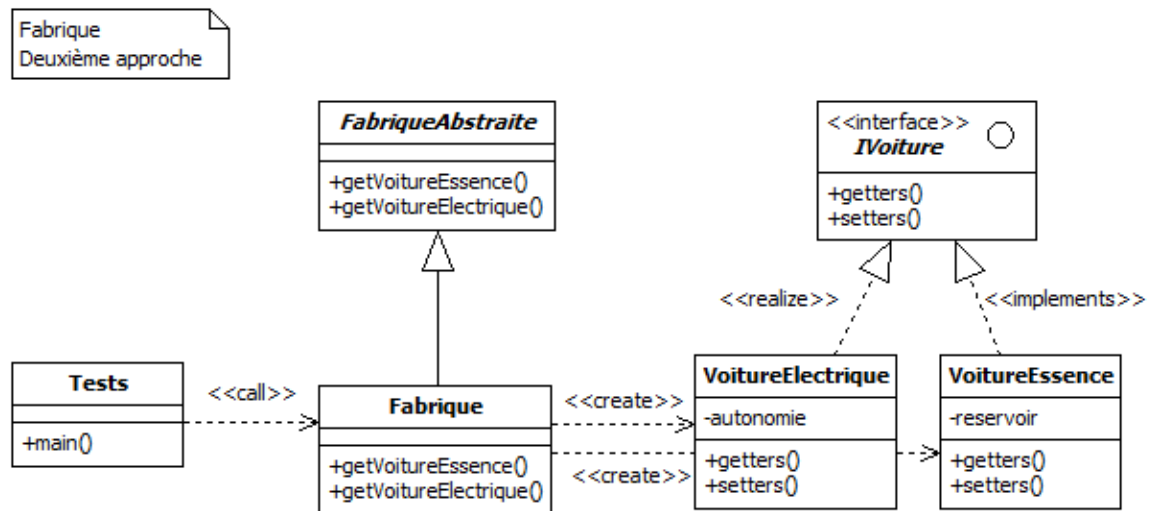
<http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

cf aussi SQL\_JAVA.odt et l'exemple de Fabrique.

[http://fr.wikipedia.org/wiki/Fabrique\\_\(patron\\_de\\_conception\)](http://fr.wikipedia.org/wiki/Fabrique_(patron_de_conception))



Le design pattern factory en pratique.



La « FabriqueAbstraite » spécifie les caractéristiques d'une fabrique. Pensez à ce que pourrait être une Usine Renault.

La « Fabrique » concrète est conforme au modèle abstrait. Pensez à une usine de Twingo et à une usine de Mégane.

Les usines – concrètes – fabriquent des voitures – selon un modèle (IVoiture ... 4 roues, un moteur écolo, ...).

### **1.8.5 - Exercice : DAOs**

Réalisez le diagramme de classes du DAO standard.

## 1.9 - DÉPENDANCE

### 1.9.1 - Présentation

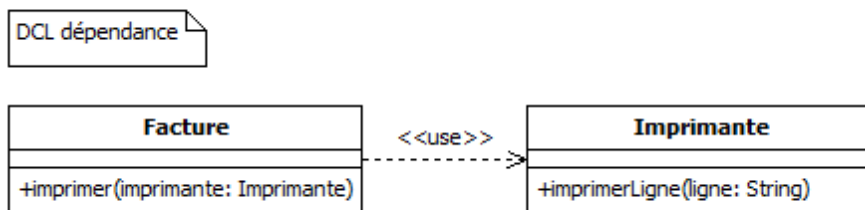
Un élément A (source) dépend d'un élément cible, lorsque A utilise des services de B.

La modification de la cible peut entraîner une modification de la source.

On utilise souvent une dépendance quand une classe en utilise une autre comme argument dans la signature d'une opération.

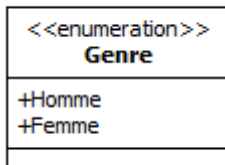
### 1.9.2 - Représentation

La dépendance est représentée par un trait discontinu terminé par une flèche ouverte.



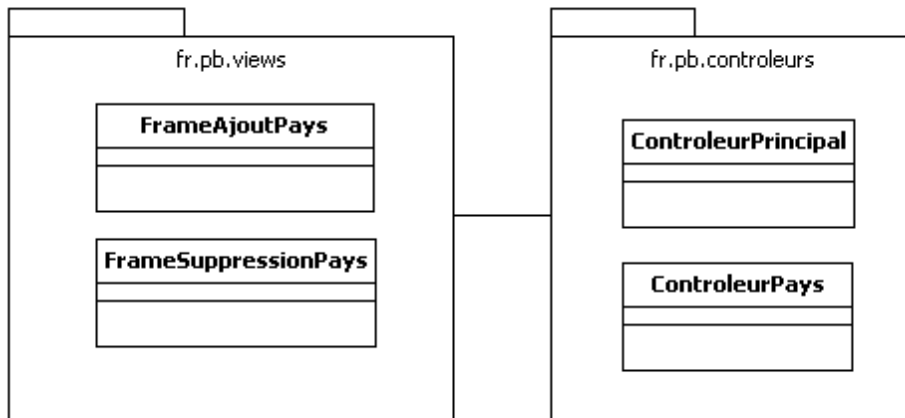
## 1.10 - LES ÉNUMÉRATIONS

Une énumération est un type UML, possédant un nom, et utilisé pour énumérer, lister un ensemble de littéraux correspondant à toutes les valeurs possibles que peut prendre une expression de ce type.



## 1.11 - PAQUETAGE

Un paquetage est un regroupement logique ou physique d'éléments du modèle.

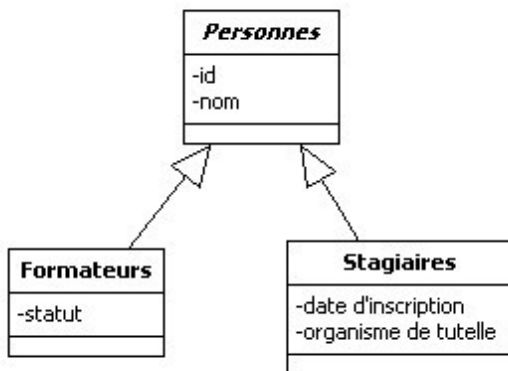


cf dpkg.odt pour plus de détails.

## 1.12 - LE SENS DES FLÈCHES EN UML

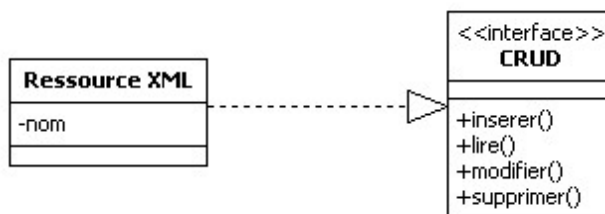
La flèche part de l'élément dépendant et est dirigée vers l'élément autonome.

### Héritage



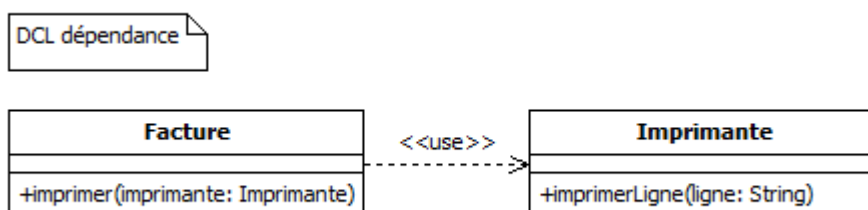
### Interface et implémentation

Même flèche et même sens que l'héritage mais avec un trait pointillé.  
La classe implémente l'interface.



### Dépendance

Un élément A dépend d'un élément B, lorsque A utilise des services de B.  
Contrat dépend de l'interface Imprimante.  
C'est le cas d'une opération qui possède un paramètre de la classe en question.

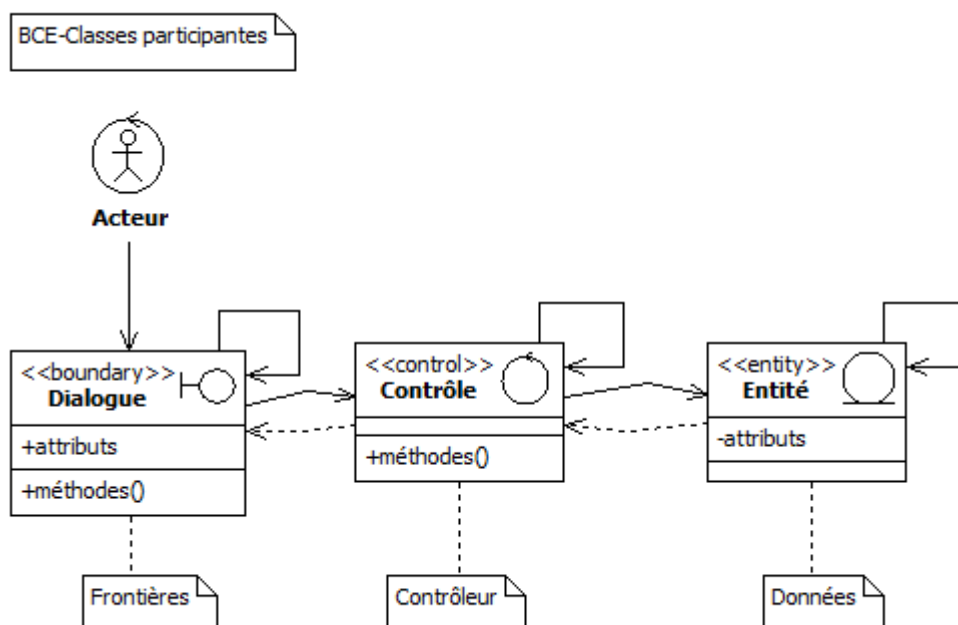


## 1.13 - UML ET ECB (ENTITY, CONTROL, BOUNDARY)

### 1.13.1 - Le modèle ECB détaillé

Le schéma suivant représente la classification des classes selon le pattern ECB de Jacobson. Il met l'accent sur la présence ou l'absence de membres dans une classe en fonction de sa catégorie et la communication inter-classes.

Attention dans ce schéma le sens (la sémantique) des flèches n'est pas celui du DCL mais plutôt d'un Diagramme de Séquence.

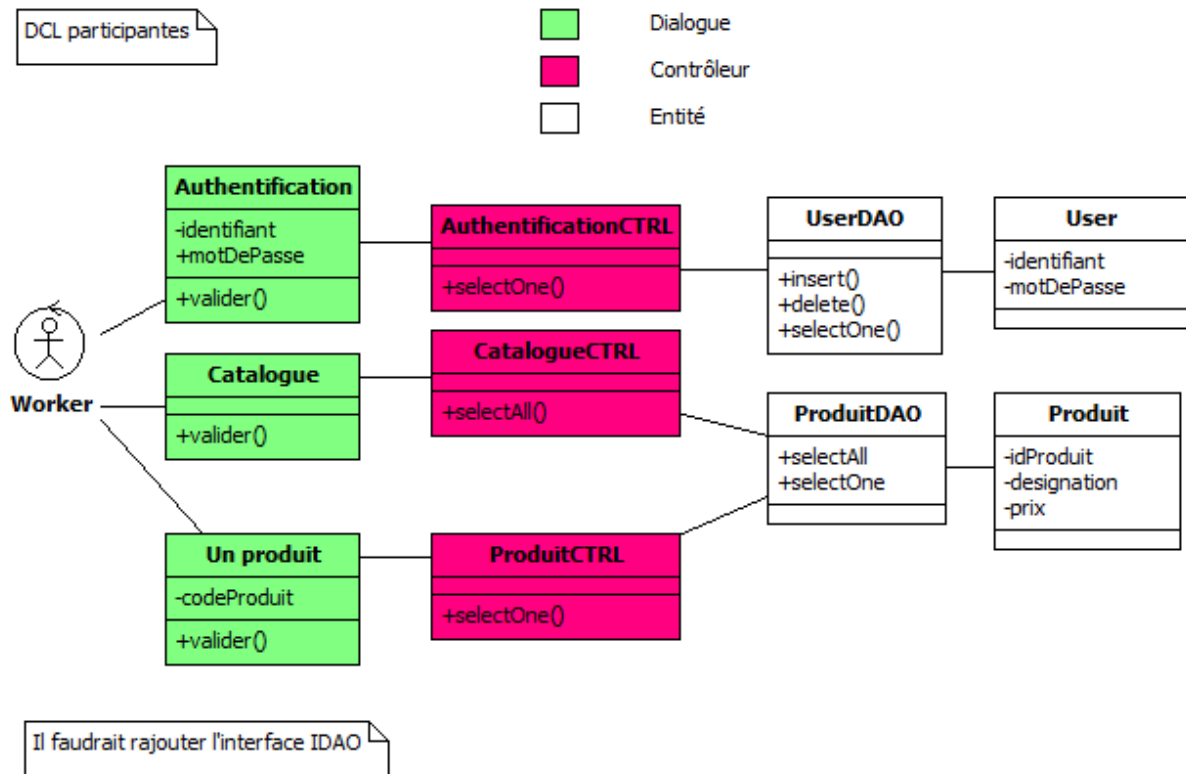


Note : le DAO est à la frontière entre la couche entité et la couche contrôle.

### 1.13.2 - Diagramme de classes participantes

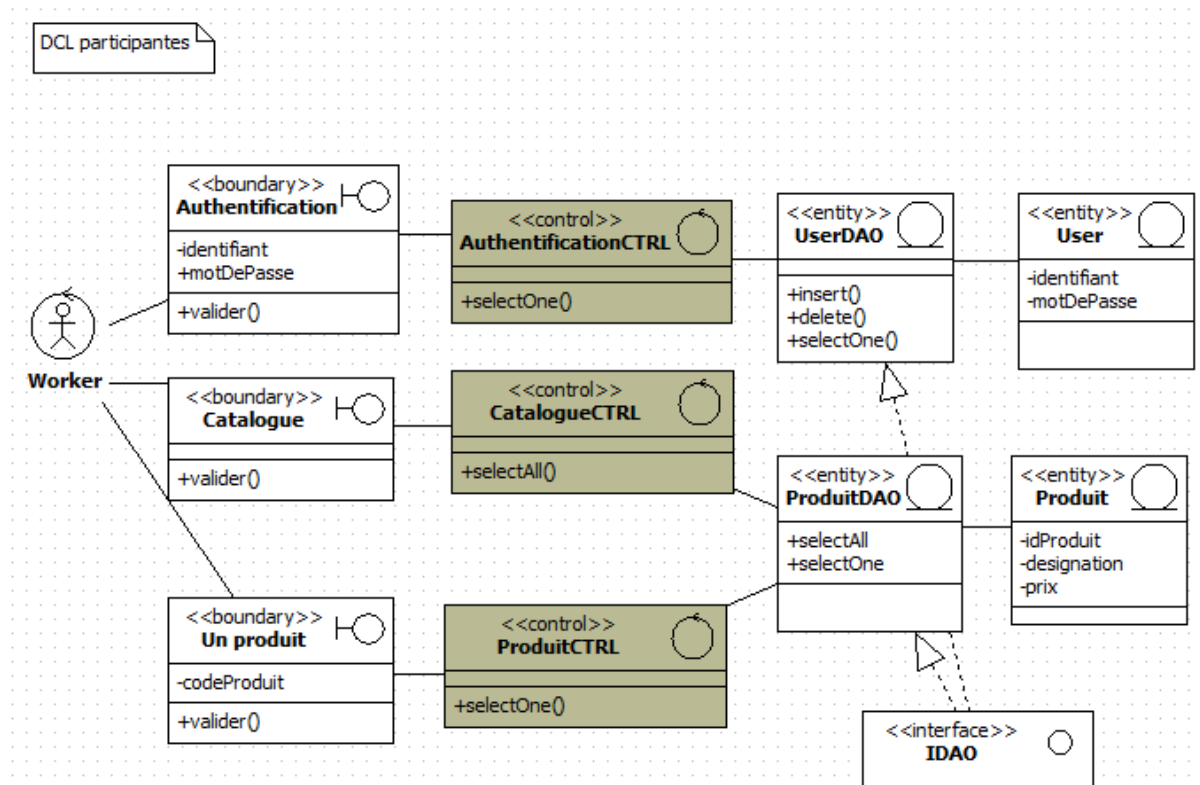
Le DCL participants est un diagramme de classes qui représente toutes les classes (boundary, control, entity) de l'application.

La couleur de fond est différente en fonction de la catégorie.



.../...





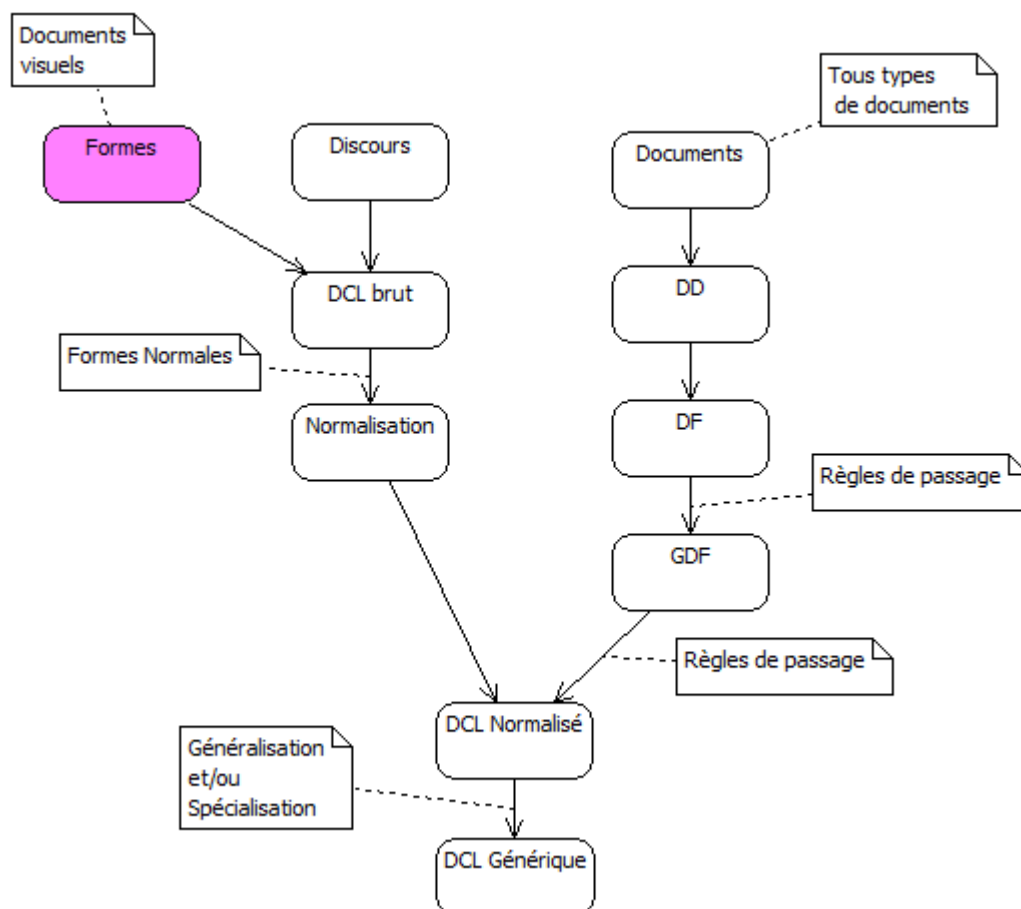
## **CHAPITRE 2 - LA CONSTRUCTION DU DCL**

## 2.1 - LES DÉMARCHES

### 2.1.1 - Les 3 principales démarches

Les démarches présentées ici sont des processus visant à créer le DCL du domaine et donc orientées vers la création de la Base de Données. Les démarches présentées ne font pas partie ni d'UML ni de UP et dérivées (RUP, TUP, 2TUP, ...).

- ✓ Démarche par l'analyse du discours (ou règles de gestion),
- ✓ Démarche ascendante,
- ✓ Démarche par les formes.



### **2.1.2 - Le cahier des charges**

Le cahier des charges fonctionnel est le document par lequel le demandeur exprime ses besoins (ou ceux qu'il est chargé d'exprimer) en termes de fonctions de service et de contraintes. Pour chacune d'elles sont définis des critères d'appréciation ainsi que leurs niveaux, chacun de ces niveaux est assorti d'un certain degré de flexibilité.

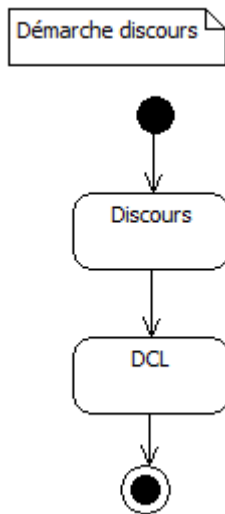
Le cahier des charges fonctionnel doit être rédigé indépendamment des solutions envisageables et doit permettre l'expression du besoin dans des termes compréhensibles par les utilisateurs.

Cf le DCU et pour SCRUM le backlog.

### **2.1.3 - Remarques**

## 2.2 - LA DÉMARCHE PAR L'ANALYSE DU DISCOURS

A partir du discours (des règles de gestion) on déduit le DCL.



Les classes et les associations sont repérées directement à partir du discours; un nom devient une classe et un verbe une association.

Inspiré de :

Peter Pin-Shan Chen, alors au M.I.T. est le créateur du modèle E-R (Entity-Relationship).

[Chen76] Chen P.P., « The Entity-Relationship Model - Towards a Unified View of Data », ACM Transactions on Database Systems, Vol. 1, N° 1, mars 1976.

Démarche plus complète :

- ✓ repérer les classes,
- ✓ repérer les associations (éventuellement les nommer),
- ✓ poser les multiplicités,
- ✓ lister les attributs et les placer,
- ✓ lister les méthodes et les placer.

**Notes** : concernant les attributs, les noms des attributs doivent être qualifiés (nomStagiaire et non pas nom) et un attribut doit être UNIQUE dans tout le diagramme de classes.

### Exemple

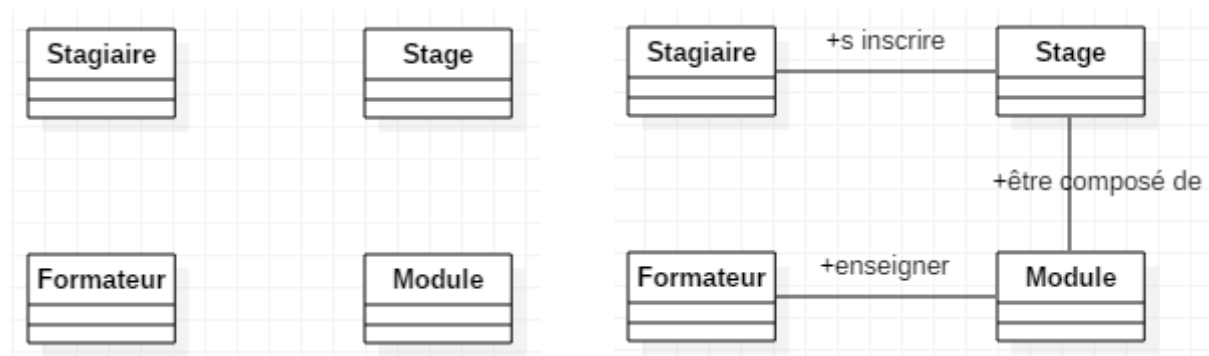
Règle 1 : un **stagiaire** **s'inscrit** à un **stage**.

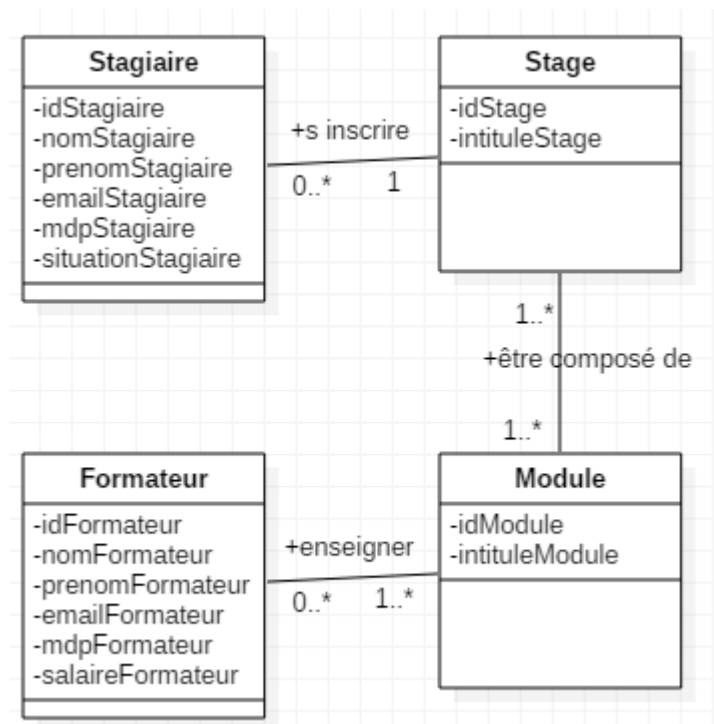
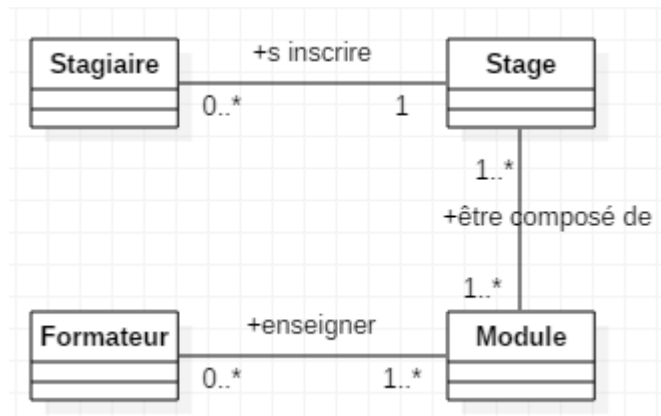
Règle 2 : les **stages** **sont composés** de plusieurs **modules**.

Règle 3 : un **formateur** **enseigne** plusieurs **modules**.

Classes : stagiaire, stage, module, formateur.

Associations : s'inscrire, composer, enseigner.





mais ...

Un stagiaire a un nom, un prénom, un e-mail, etc.

Le verbe « **avoir** » signifie « est caractérisé par ».

Le nom, le prénom, etc, **ne sont pas des objets ayant une existence propre**. C'est une caractéristique intrinsèque du stagiaire.

Ces relations seront traduites non pas en associations mais en attributs de classe. nom, prénom, etc ne sont pas des objets.

Un paysan a un tracteur.

Le verbe avoir signifie « possède ». Le tracteur n'est pas une caractéristique intrinsèque du paysan, c'est un **objet distinct ayant une existence propre**.

Cette relation va se traduire en deux classes et une association; un paysan est un objet et un tracteur aussi.

Le verbe **avoir**, dans le premier cas, peut être « traduit » par « a comme caractéristique » – et c'est un attribut –, et dans le deuxième cas par « possède » et c'est une autre classe reliée par une association.

Le verbe « **être** » est le plus souvent traduit par un objet donc une classe.

Pascal est un formateur.

UML est un enseignement.

La salle 14 est une salle de cours.

La démarche ascendante et l'analyse des dépendances fonctionnelles permettent – le plus souvent - de lever ces ambiguïtés. Les règles de normalisation en 1<sup>ère</sup>, 2<sup>ème</sup>, ... forme normale de Codd et Ali permettent, elles aussi, d'éliminer les anomalies de redondance et de mises à jour dans les modèles.

### Exercices :

1) Modifiez le DCL « Formation » avec les nouvelles règles de gestions suivantes :

RG 4 : un stagiaire peut être inscrit à plusieurs sessions de stages.

RG 5 : un formateur intervient lors d'une session de module à partir d'une certaine date pour un certain nombre de jours.

2) Les minis projets réalisés en PHP.

3) La modélisation d'un bureau de vote.



## 2.3 - LA DÉMARCHE ASCENDANTE EN DÉTAIL

Ascendante : du concret vers l'abstrait.

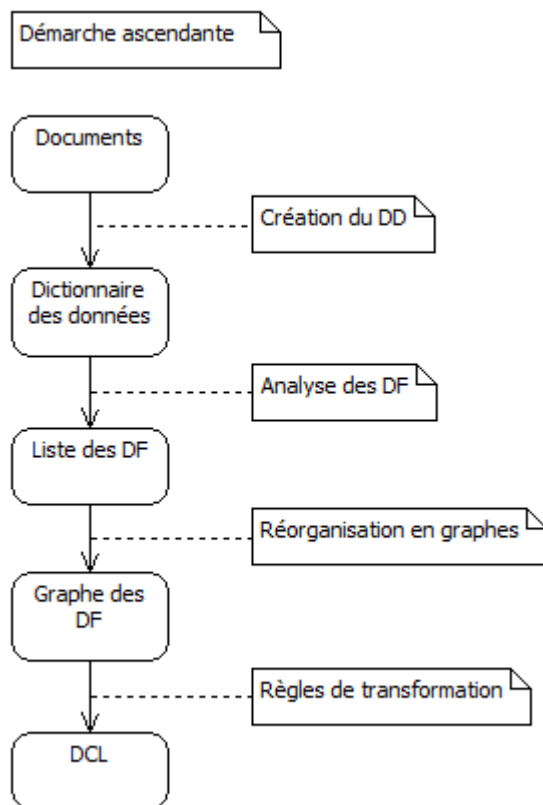


Tableau des sigles :

DD	Dictionnaire des données
DF	Dépendance fonctionnelle
GDF	Graphe des dépendances fonctionnelles
DCL	Diagramme de classes

### 2.3.1 - Exemple : Bon de commande

Construction du DCL à partir d'un bon de commande.

N°Bon	_____	Date	_____
Code client	_____		
Nom	_____		
Adresse	_____		
Vendeur	_____		

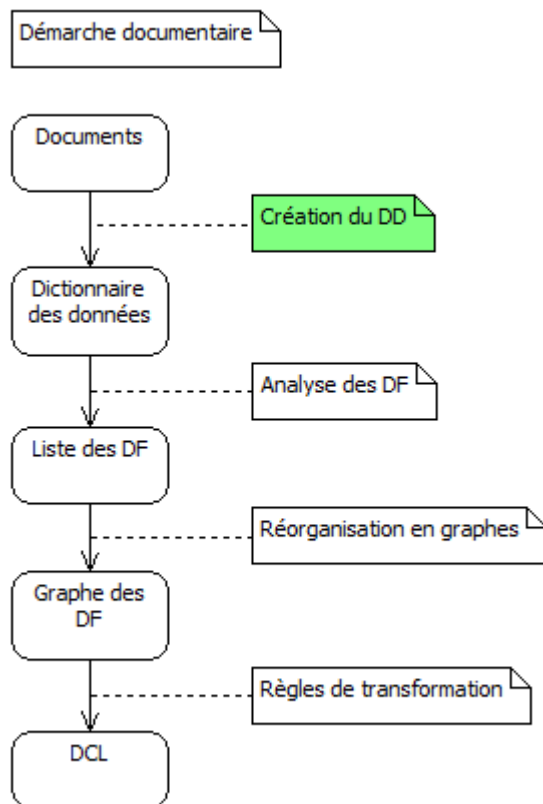
  

Réf	Libellé	Prix	Quantité	Montant
_____	_____	_____	_____	_____
_____	_____	_____	_____	_____
_____	_____	_____	_____	_____

Total \_\_\_\_\_

Il y a 12 informations dans ce document.

### 2.3.2 - Le Dictionnaire des Données



A partir des **documents** disponibles (Écrans, états, structures des fichiers et des bases de données existant, ...) on établit le **dictionnaire des données (DD** : document d'origine, libellé, code, signification, type, longueur, nature - élémentaire, calculée, concaténée - règle de calcul ou contrainte d'intégrité, règle de forme, ...). Les noms des données sont uniques ; la règle est d'utiliser des compléments de nom (nomDuStagiaire, nomDuFormateur, dateDeNaissance, dateDEmbauche, ...). Ensuite on **épure le dictionnaire** (synonymes - noms différents recouvrant le même attribut : salarié et employé -, polysèmes - même nom pour deux informations différentes : date pour date de facture et date de commande, ...).

Libellé	Code	Type	Longueur	Nature	Calcul	Autre règle
N° Bon	idCommande	Numérique	5	Calculée	+1	
Date	dateCommande	Date	10	Élémentaire		JJ/MM/AAAA
Code client	idClient	Numérique	5	Calculée	+1	
Nom	nomClient	Texte	50	Élémentaire		
Adresse	adresseClient	Texte	100	Élémentaire		
Vendeur	nomVendeur	Texte	50	Élémentaire		
Réf	idProduit	Numérique	5	Calculée	+1	
Libellé	designationProduit	Texte	50	Élémentaire		
Prix	prixProduit	Numérique	8	Élémentaire		
Quantité	quantiteCommandee	Numérique	5	Élémentaire		>0
Montant	montantLigCommande	Numérique	8	Calculée	prix * quantité	
Total	totalCommande	Numérique	8	Calculée	somme(prix * quantité)	

### 2.3.3 - Les Dépendances Fonctionnelles (DF)

Cette définition est fondamentale. C'est la seule qui réclame notre intelligence !!!

**Dépendance fonctionnelle** : il y a dépendance fonctionnelle entre deux attributs lorsque la connaissance d'une valeur d'un attribut permet de déterminer une et une seule valeur d'un autre attribut.

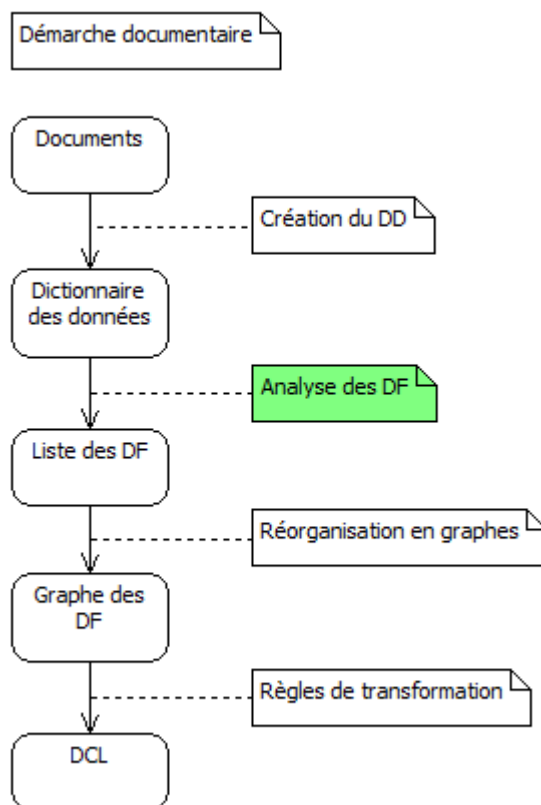
La dépendance fonctionnelle est notée  $A1 \rightarrow A2$ .

Elle signifie que A1 détermine A2 ou que A2 dépend de A1.

Exemple :  $idClient \rightarrow nomClient$ .

En revanche  $nomClient$  ne détermine pas  $idClient$  (A partir du nom d'un client on ne peut pas retrouver un seul code client).

On établit la liste des DF.



Quelques caractéristiques des dépendances fonctionnelles :

la réflexivité :  $a1 \rightarrow a1$ ,

la transitivité : si  $a1 \rightarrow a2$  et  $a2 \rightarrow a3$  alors  $a1 \rightarrow a3$ ,

l'augmentativité : si  $a1 \rightarrow a2$  alors  $a1, a_n \rightarrow a2$ .

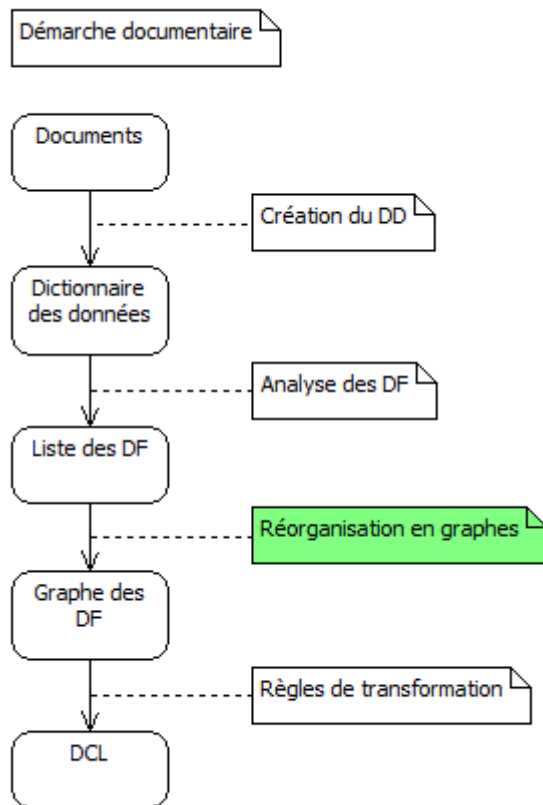
cf [http://www.dil.univ-mrs.fr/~novelli/BD/Initiation\\_DF\\_NF.pdf](http://www.dil.univ-mrs.fr/~novelli/BD/Initiation_DF_NF.pdf)

### 2.3.4 - Matrice des DF

Attributs		1	2	3	4	5	6	7	8	9	10	11	12
idCommande	1	x	x	x	x	x	x						x
dateCommande	2		x										
idClient	3			x	x	x							
nomClient	4				x								
adresseClient	5					x							
nomVendeur	6						x						
idProduit	7							x	x	x			
designationProduit	8							x	x	x			
prixProduit	9									x			
quantiteCommandee	10										x		
montantLigneDeCommande	11											x	
totalCommande	12												x

### 2.3.5 - Le Graphe des Dépendances Fonctionnelles (GDF)

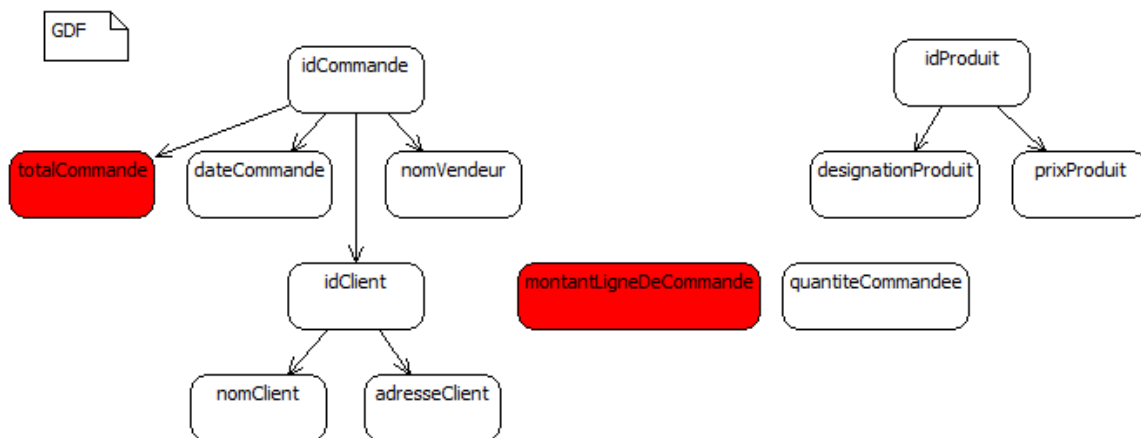
Il s'agit d'ordonner, pour avoir une vision synthétique, les résultats de l'analyse des DF faite précédemment.



Quelques règles de transformations :

- ✓ l'attribut source du plus grand nombre de DF est situé en haut du graphe,
- ✓ une dépendance fonctionnelle est représentée par une flèche,
- ✓ les dépendances qui s'obtiennent par transitivité sont éliminées (couverture minimale),
- ✓ pour les attributs orphelins il faut rechercher les sources multiples de dépendances fonctionnelles.

### Phase 1 : transformation directe de la matrice en GDF



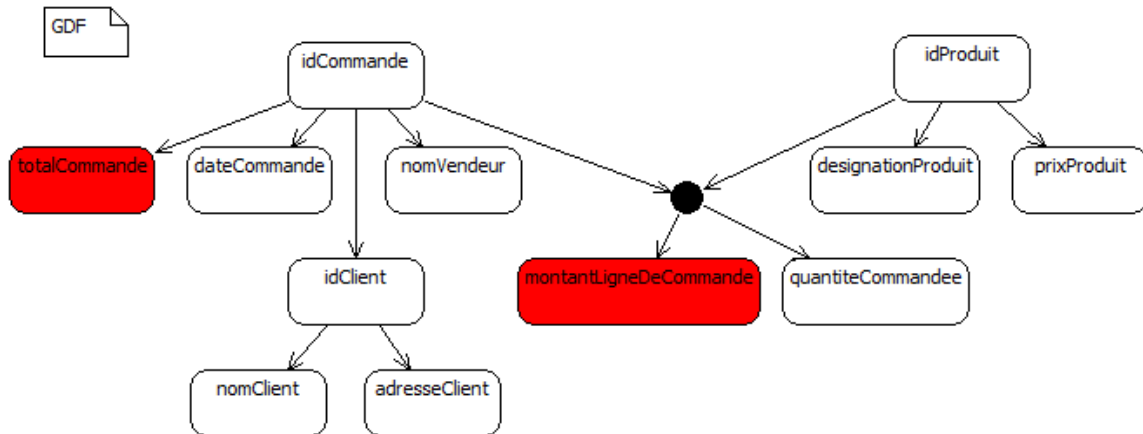
Certains attributs sont orphelins, il faut donc rechercher des dépendances fonctionnelles multi-sources.

**Phase 2 : prise en considération des DF ayant des sources multiples**

Il existe deux dépendances fonctionnelles multi-sources :

$\text{idCommande} + \text{idProduit} \rightarrow \text{montantLigneDeCommande}$

$\text{idCommande} + \text{idProduit} \rightarrow \text{quantiteCommandee}$



Notes :

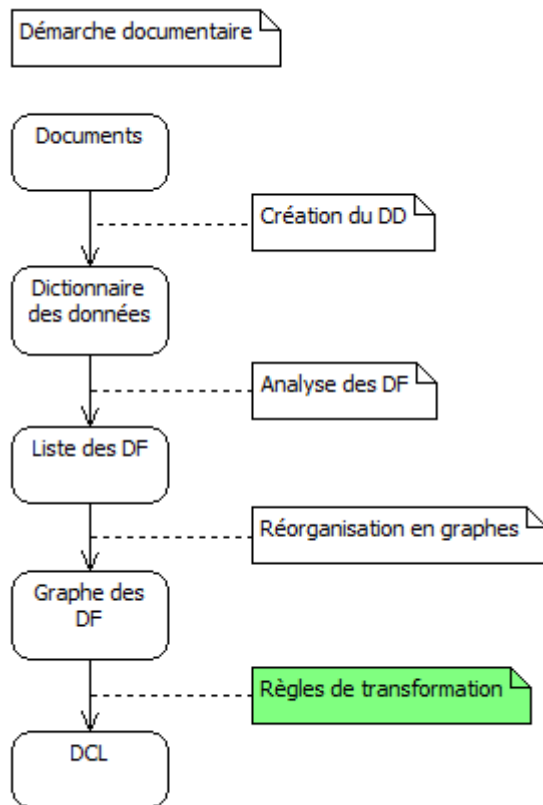
Le total et le montant sont calculés et seront éventuellement éliminés du DCL. La règle n'est pas absolue. Si l'on devait recalculer le montant et le total quelque temps après l'insertion de la commande, et qu'une variation de prix soit intervenue entre temps, l'élimination du montant ne permettrait pas de recalculer le total de la facture tel qu'il était au moment de la commande.

Si vous ajoutiez un idVendeur vous auriez une autre classe, la classe vendeur et une association entre vendeur et commande.



### 2.3.6 - Le DCL

A partir du GDF on établit le DCL.



Les règles de transformation sont :

GDF	DCL
Arbre	Classe
Sommet d'un arbre	Attribut identifiant d'une classe
Feuille terminale d'un arbre	Attribut non-identifiant d'une classe
Concaténation	Association de type non Père-Fils (multiplicités de type * - *)
Feuille d'une concaténation	<b>Classe association</b> Attribut d'une <b>classe-association</b>
DF inter-sommets	Association inter-classes avec au moins une multiplicité maxi de 1
Un attribut ayant plusieurs sommets	Une classe et 2 associations (*)

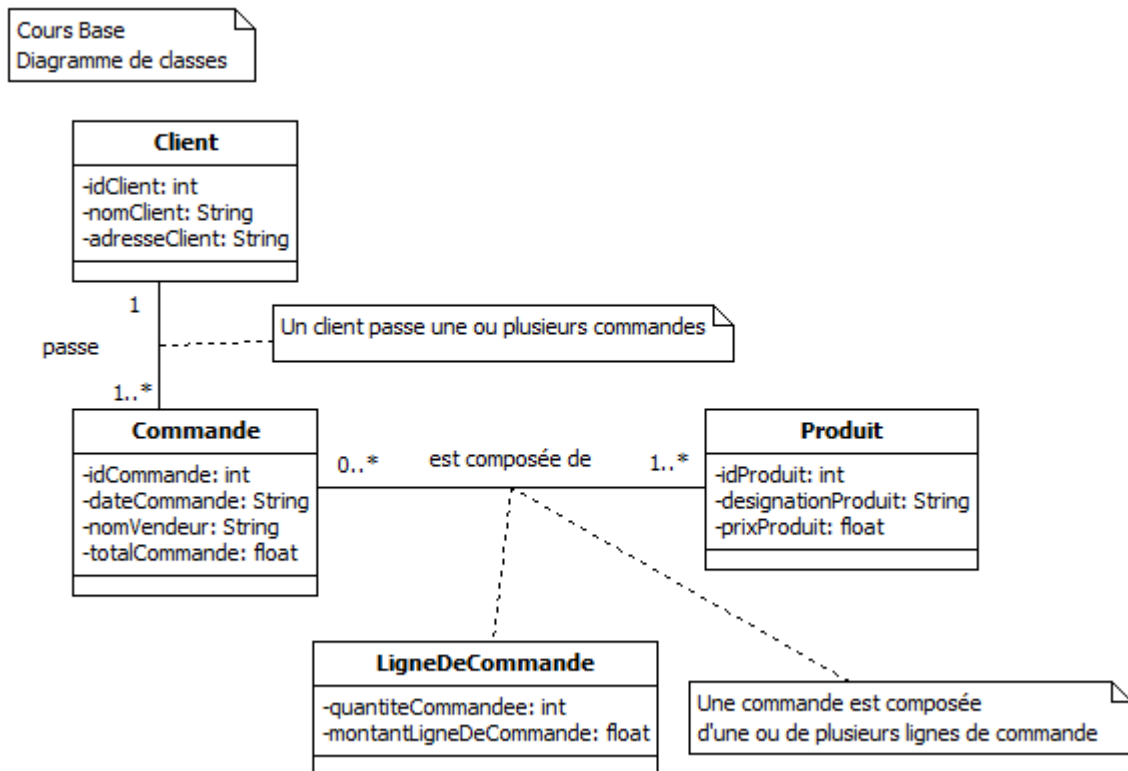
Note : en théorie des graphes, un arbre est un graphe non orienté, acyclique.

(\*)

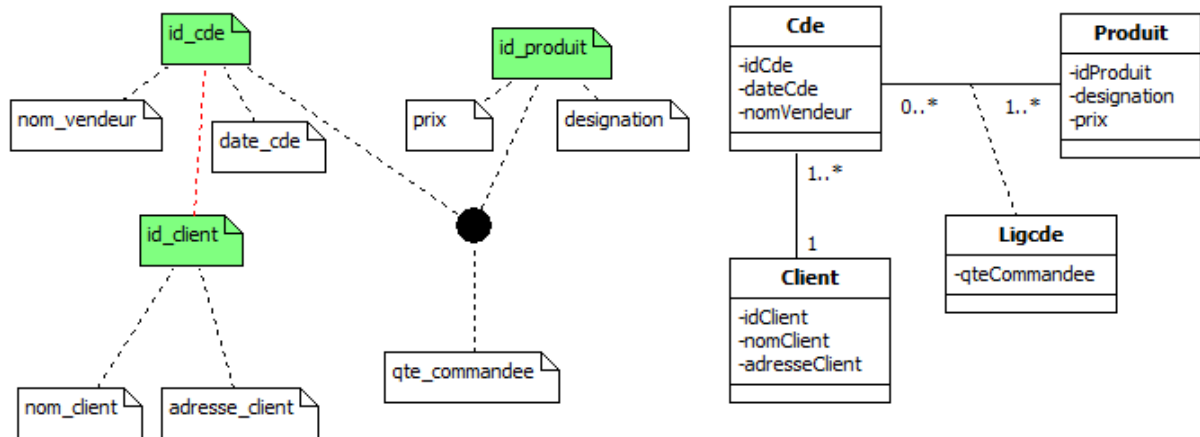
client → vendeur (un client a un vendeur attribué)

ville → vendeur (pour une ville il y a un seul vendeur)

## Diagramme de classes



# Résumé : Graphe des Dépendances Fonctionnelles et Diagramme de classes



Arbre -> Classe  
 Sommet d'arbre (en vert) -> identifiant  
 Feuille (en blanc) -> attribut  
 Concaténation (cercle noir) -> Classe-Association  
 DF inter-sommets (discontinu rouge) -> Association avec une multiplicité max de 1

## 2.4 - LES FORMES NORMALES

La normalisation a pour objectif d'éliminer les redondances dans la base de données ainsi que les anomalies de mise à jour.

Les classes doivent vérifier les règles suivantes :

- ✓ Première forme normale,
- ✓ Deuxième forme normale,
- ✓ Troisième forme normale,
- ✓ BCNF (Boyce Codd Normal Form ou Forme Normale de Boyce Codd),
- ✓ Quatrième forme normale,
- ✓ Cinquième forme normale,
- ✓ DKNF (Domain Key Normal Form ou Forme Normale Domaine Clef),
- ✓ Sixième forme normale.

Les formes normales ont été initiées par Codd dans le cadre du modèle relationnel ("A Relational Model of Data for Large Shared Data Banks"). En fait les 3 premières ainsi que la BCNF.

Les 4 et 5<sup>ème</sup> et la DKNF par Ronald Fagin.

La 6<sup>ème</sup> par C.J. Date et ali.

<http://fsmrel.developpez.com/basesrelationnelles/normalisation/>

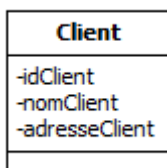
### 2.4.1 - Première Forme Normale (1<sup>ère</sup> FN)

Tous les attributs sont élémentaires et il existe un attribut identifiant.

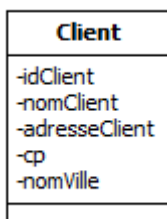
Sinon on décompose un attribut en plusieurs attributs et/ou on crée un attribut identifiant.

#### Exemple

Cette classe n'est pas en 1<sup>ère</sup> FN



Cette classe est en 1<sup>ère</sup> FN



## Démonstration

Avant

#idClient	nomClient	adresseClient
1	Tintin	33, rue de la Paix – 75002 – Paris
2	Milou	33, rue de la Paix – 75002 – Paris
3	Casta	5, rue de la Pompe – 75016 – Paris
4	Tournesol	3, rue Roubo – 75011 – Paris
5	Haddock	7, rue Oberkampf – 75011 – Paris

Après

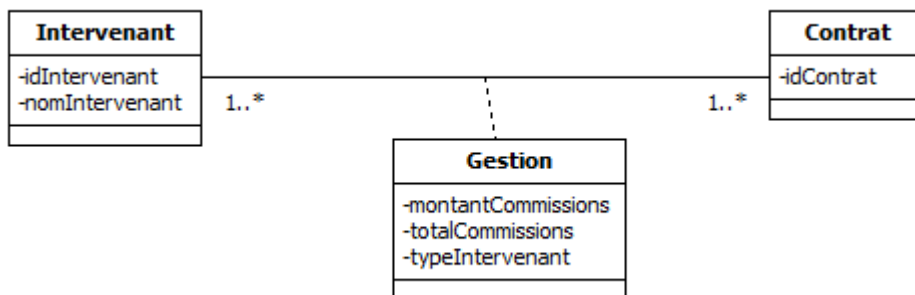
#idClient	nomClient	adresseClient	cp	ville
1	Tintin	33, rue de la Paix	75002	Paris
2	Milou	33, rue de la Paix	75002	Paris
3	Casta	5, rue de la Pompe	75016	Paris
4	Tournesol	3, rue Roubo	75011	Paris
5	Haddock	7, rue Oberkampf	75011	Paris

## 2.4.2 - Deuxième Forme Normale (2<sup>ème</sup> FN)

Tout attribut dépend de l'identifiant par une dépendance fonctionnelle (DF) élémentaire.  
Donc chaque attribut dépend de tout l'identifiant et non pas d'une partie.  
Sinon on décompose en plusieurs classes.

### Exemple

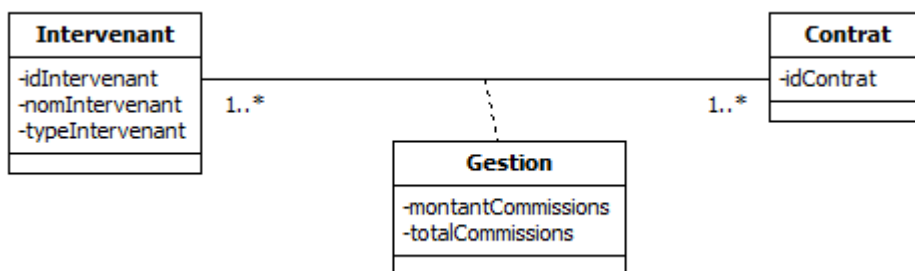
Modélisation qui n'est pas en 2<sup>ème</sup> FN



Or il existe une DF entre Intervenant et Type intervenant (la règle de gestion étant qu'un intervenant est d'un seul type),

Et une autre DF entre Intervenant + Année et Total commissions (la règle de gestion est que le total des commissions est le montant annuel des commissions).

Modélisation en 2<sup>ème</sup> FN





### Démonstration

Avant

Intervenants : I1, I2

Contrats : C1, C2, C3

### Gestion

Intervenant, Contrat	Montant	Total	TypeIntervenant
I1, C1	100	10000	Senior
I1, C2	200	10000	Junior
I1, C3	200	11000	Senior

Il y a une incohérence : l'intervenant pourrait être Senior et Junior.

Après

### Intervenant

Intervenant	TypeIntervenant
I1	Senior
I1	Junior

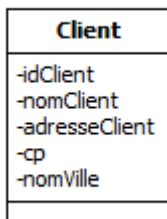
**Exercice** : reprenez l'exercice « Chevaux » et vérifiez la normalisation en 2<sup>e</sup> FN.

### 2.4.3 - Troisième Forme Normale (3<sup>ème</sup> FN)

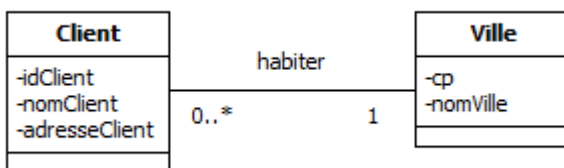
Tout attribut doit dépendre de l'attribut identifiant par une DF directe.  
Donc tous les attributs non identifiants sont indépendants entre eux.  
Sinon on décompose en deux classes.

#### Exemple

Modélisation qui n'est pas en 3<sup>ème</sup> FN



Modélisation en 3<sup>ème</sup> FN



**Démonstration**

Avant

**Client**

#IdClient	NomClient	AdresseRue	Cp	NomVille
1	Tintin	33, rue de la Paix	75011	Paris XI
2	Milou	33, rue de la Paix	75012	Paris XII
3	Casta	5, rue de la Pompe	75011	Lyon
4	Tournesol	3, rue Roubo	75012	Paris XII
5	Haddock	7, rue Oberkampf	75012	Paris XII

Anomalie et redondance.

Après

**Client**

#IdClient	NomClient	AdresseRue
1	Tintin	33, rue de la Paix
2	Milou	33, rue de la Paix
3	Casta	5, rue de la Pompe
4	Tournesol	3, rue Roubo
5	Haddock	7, rue Oberkampf

**Ville**

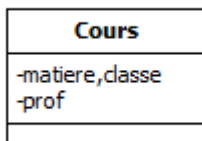
#Cp	NomVille
75011	Paris XI
75012	Paris XII
75011	Lyon

## 2.4.4 - Forme Normale de Boyce-Codd (BCNF)

Tout élément de l'identifiant est indépendant d'un attribut.

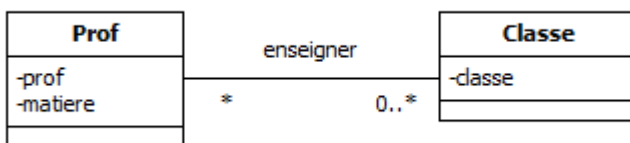
### Exemple

Modélisation qui n'est pas en BCNF



Règle supplémentaire : un prof enseigne une seule matière

Modélisation qui est en BCNF



## Démonstration

Avant

### Cours

#Matiere, classe	Prof
Maths, 1ere	Tintin
Français, 1ere	Milou
Français, Terminale	Tintin

Après

### Profs

#Prof	Matiere
Tintin	Maths
Milou	Français
<del>Tintin</del>	<del>Français</del>

### Classes

#Classe
1ere
Terminale

### Enseigner

#Prof	#Classe
Tintin	1ère
Tintin	Terminale
Milou	1ère

## **CHAPITRE 3 - LE DIAGRAMME DE CLASSES ET LA POO**

### 3.1 - TABLEAU SYNTHÉTIQUE

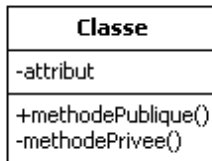
Élément UML	Classe, élément, etc	Commentaire
Classe	class	
Classe abstraite	abstract class	
Attribut	Propriété	
Opération	Méthode	
Association binaire avec une multiplicité de 1 et une autre de *	2 classes 1 liste côté * 1 attribut de type classe côté 1	Ville et Client
Association binaire avec une multiplicité *,*	2 classes 2 listes pour les liaisons	Vendeur et Client
Association binaire avec une multiplicité *,* et une classe association	4 classes : une classe Commande, une classe Produit, une classe LigneDeCommande et une classe IdLigneDeCommande 2 listes pour les liaisons	Commande, LigneDeCommande et Produit
Généralisation	extends	
Agrégation	Comme une association de multiplicité 1	
Composition	Une classe avec une Nested Class	Classe privée interne
Réflexive	Comme une association binaire	
Réalisation	Interface	
Dépendance	Une méthode avec un paramètre de type Class	
Contrainte	Méthode	

cf aussi JAVA\_INTRO\_PRINCIPES\_OBJET.odt

## 3.2 - QUELQUES EXEMPLES AVEC JAVA ET PHP

### 3.2.1 - Classe

#### UML



#### Java

```
public class Classe {  
    private Object attribut;  
  
    public void methodePublique() {  
    }  
    private void methodePrivee() {  
    }  
}
```

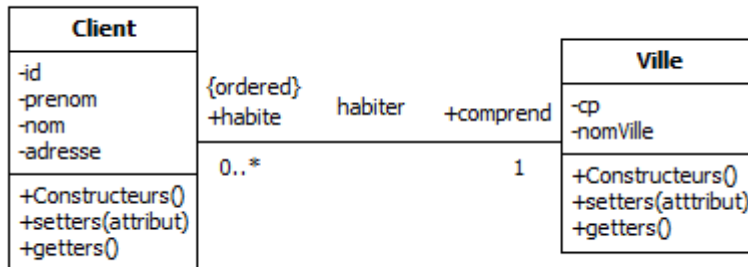
#### PHP

```
class Classe {  
    private $attribut;  
  
    public function methodePublique() {  
    }  
    private function methodePrivee() {  
    }  
}
```



### 3.2.2 - Association 1,\*

UML



**Java**

```
public class Ville {  
    private String cp;  
    private String nomVille;  
    private List<Client> clients;  
  
    public Ville() {  
    }  
    public void setter(Object attribut) {  
    }  
    public type getter() {  
    }  
}
```

```
public class Client {  
    private int id;  
    private String prenom;  
    private String nom;  
    private String adresse;  
    private Ville ville;  
  
    public Client() {  
    }  
    public void setter(Object attribut) {  
    }  
    public type getter() {  
    }  
}
```

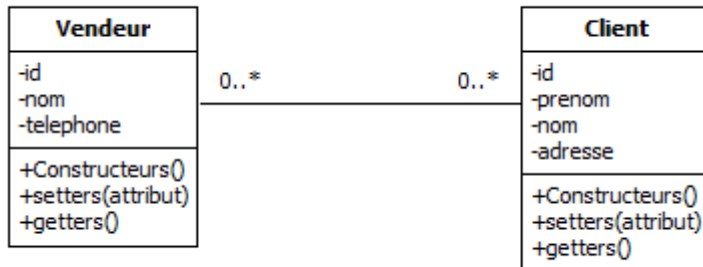
## PHP

```
class Ville {  
    private $cp;  
    private $nomVille;  
    private $clients;  
  
    public __construct() {  
    }  
    public __set($var, $valeur) {  
    }  
    public __get($var) {  
    }  
}
```

```
class Client {  
    private $id;  
    private $prenom;  
    private $nom;  
    private $adresse;  
    private $ville;  
  
    public __construct() {  
    }  
    public __set($var, $valeur) {  
    }  
    public __get($var) {  
    }  
}
```

### 3.2.3 - Association \*\*,\*

#### UML



#### Java

```

public class Client {
    private int id;
    private String nom;
    private String prenom;
    private String adresse;
    private List<Vendeur> vendeurs;

    public Client() {
    }
    public void setter(Object attribut) {
    }
    public type getter() {
    }
}
  
```

```

public class Vendeur {
    private int id;
    private String nom;
    private String telephone;
    private List<Client> clients;

    public Vendeur() {
    }
    public void setter(Object attribut) {
    }
    public type getter() {
    }
}
  
```

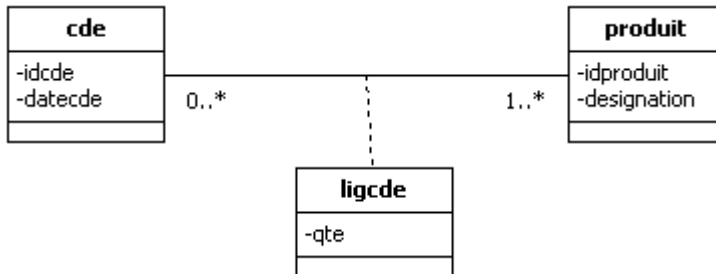
## PHP

```
class Vendeur {  
    private $id;  
    private $nom;  
    private $telephone;  
    private $clients;  
  
    public __construct() {  
    }  
    public __set($var, $valeur) {  
    }  
    public __get($var)  
    {  
    }  
}
```

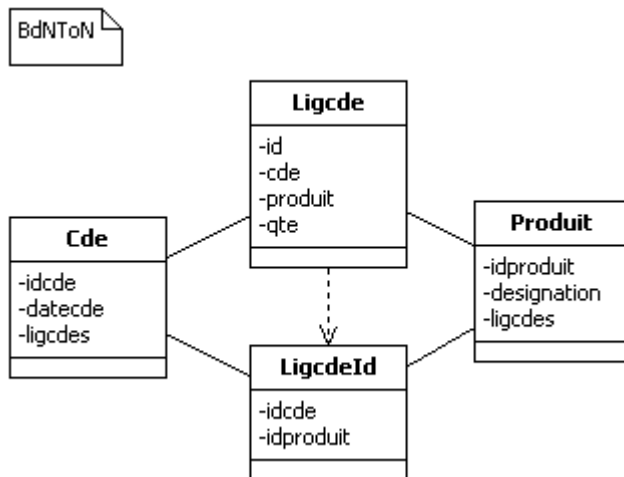
```
class Client {  
    private $id;  
    private $prenom;  
    private $nom;  
    private $adresse;  
    private $vendeurs;  
  
    public __construct() {  
    }  
    public __set($var, $valeur) {  
    }  
    public __get($var) {  
    }  
}
```

### 3.2.4 - Association \*,\* avec une classe association

UML



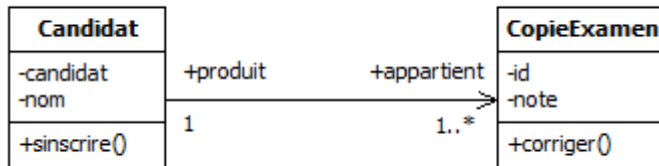
Les classes Java



Notes : id de Ligcde est composé de cde + produit, produit de Ligcde est de type Produit et cde de Ligcde est de type Cde.

### 3.2.5 - Association orientée

#### UML



#### Java

```

public class Candidat {
    private String candidat;
    private String nom;
    public List<CopieExamen> copie;

    public void sinscrire() {
    }
}
  
```

```

public class CopieExamen {
    private int id;
    private int note;

    public void corriger() {
    }
}
  
```

## **PHP**

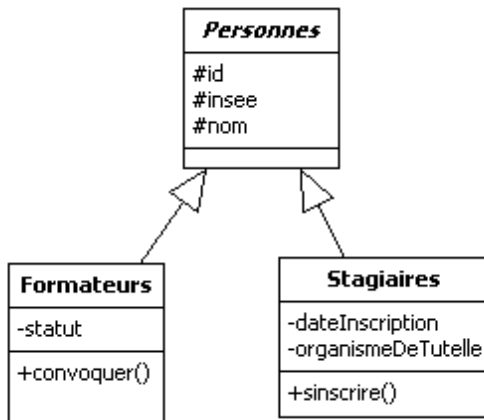
```
| class Candidat {  
|     private $candidat;  
|     private $nom;  
|     private $copie;  
  
|     public function inscrire() {  
|         }  
| }
```

```
| class CopieExamen {  
|     private $id;  
|     private $note;  
  
|     public function corriger() {  
|         }  
| }
```



### 3.2.6 - Héritage

#### UML



#### Java

```

public abstract class Personnes {
    protected int id;
    protected String insee;
    protected String nom;
}

```

```

public class Formateurs extends Personnes {
    private String statut;

    public void convoquer() {
    }
}

```

```

public class Stagiaires extends Personnes {
    private Date dateInscription;
    private String organismeDeTutelle;

    public void sinscrire() {
    }
}

```

## PHP

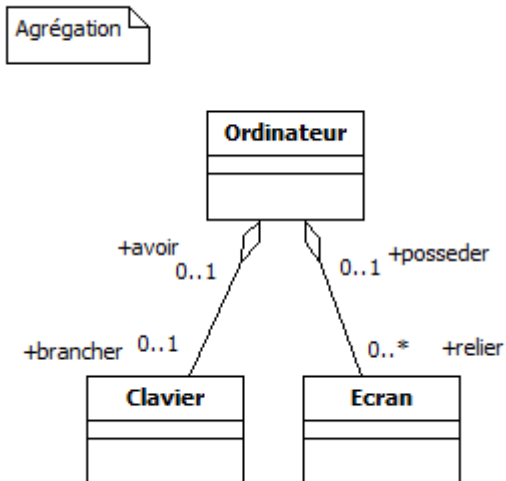
```
| abstract class Personnes {  
|     protected $id;  
|     protected $insee;  
|     protected $nom;  
|  
| }
```

```
| class Formateurs extends Personnes {  
|     private $statut;  
|  
|     public function convoquer() {  
|         }  
| }
```

```
| class Stagiaires extends Personnes {  
|     private $dateInscription;  
|     private $organismeDeTutelle;  
|  
|     public function inscrire() {  
|         }  
| }
```

### 3.2.7 - Agrégation

#### UML



#### Java

```

public class Ordinateur {
    public Clavier a;
    public List<Ecran> possede;
}

```

```

public class Ecran {
    public Ordinateur relier;
}

```

```

public class Clavier {
    public Ordinateur brancher;
}

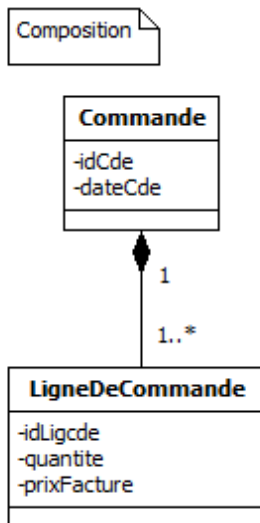
```

**PHP**

|

### 3.2.8 - Composition

#### UML



#### Java

```

public class Commande {
    private Object idCommande;
    private Object dateCommande;
    public LigneDeCommande Unnamed1;

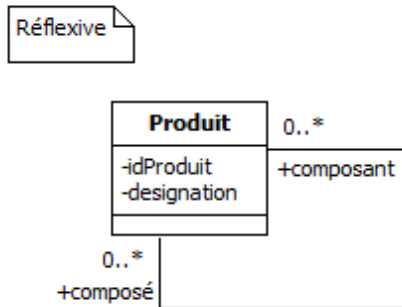
    private class LigneDeCommande {
        private Object idLigCommande;
        private Object quantite;
        private Object prixFacture;
        public Commande Unnamed1;
    }
}
  
```

#### PHP

```

|
  
```

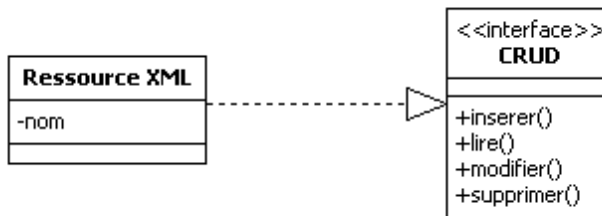
### 3.2.9 - Réflexive



```
public class Produit {
    private Object idProduit;
    private Object designation;
    public List<Produit> compose;
    public List<Produit> composant;
}
```

### 3.2.10 - Interface

#### UML



#### Java

```

public interface CRUD {
    public void inserer();
    public void lire();
    public void modifier();
    public void supprimer();
}
  
```

```

public class RessourceXML implements CRUD {

    private String nom;

    public void inserer() {
        // Du code
    };
    public void lire() {
        // Du code
    };
    public void modifier() {
        // Du code
    };
    public void supprimer() {
        // Du code
    };
}
  
```

## **PHP**

```
interface CRUD {  
    public function inserer();  
    public function lire();  
    public function modifier();  
    public function supprimer();  
}
```

```
class RessourceXML implements CRUD {  
  
    private $nom;  
  
    public function inserer() {  
        // Du code  
    };  
    public function lire() {  
        // Du code  
    };  
    public function modifier() {  
        // Du code  
    };  
    public function supprimer() {  
        // Du code  
    };  
}
```

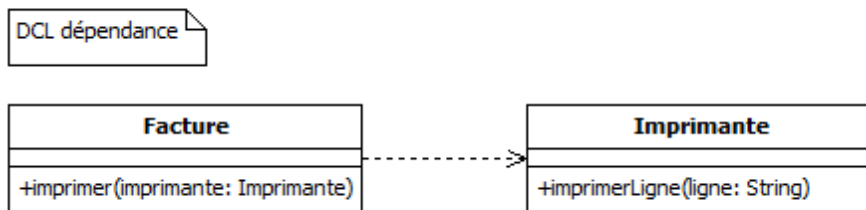


### 3.2.11 - Dépendance

Un élément A dépend d'un élément B, lorsque A utilise des services de B.

On utilise souvent une dépendance quand une classe en utilise une autre comme argument dans la signature d'une opération.

#### UML



#### Java

```
public class Facture {
    public void imprimer(Imprimante imprimante) {
    }
}

public class Imprimante {
    public void imprimerLigne(String ligne) {
    }
}
```

#### PHP

```
|
```

## **CHAPITRE 4 - LE DIAGRAMME DE CLASSES ET SQL**

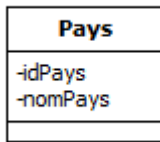
## 4.1 - QUELQUES RÈGLES DE PASSAGE DU MODÈLE OBJET AU MODÈLE RELATIONNEL

Christian SOUTOU et Frédéric BROUARD, "UML 2 pour les Bases de Données", Eyrolles, 2012.

Élément UML	Élément relationnel	Commentaire
Classe	Table	
Attribut <i>identifiant</i>	Clé primaire	
Attribut	Colonne	
Association de multiplicité 1-1..*	Clé étrangère	
Association de multiplicité 1-1	Clés étrangères « réciproques »	
Association de multiplicité *-*	Table de jointure avec une clé primaire composée de 2 clés étrangères	
Classe association	Table de jointure avec une clé primaire composée de 2 clés étrangères et autant de colonnes que d'attributs de la classe-association	
Agrégation	Tables, clé étrangère	Suppression avec affectation de NULL
Composition	Tables, clé étrangère	Suppression en cascade
Réflexive	Comme une association binaire	
Méthode	Fonction ou procédure stockée	
Contrainte	Fonction ou procédure stockée ou trigger	

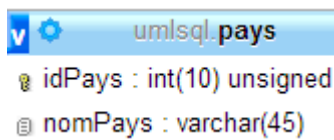
## 4.2 - UNE CLASSE

### UML



### SQL

Une classe = une table.

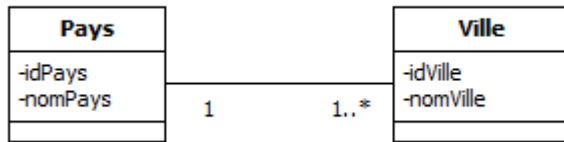


```

DROP TABLE IF EXISTS pays;
CREATE TABLE pays (
  idPays int(10) unsigned NOT NULL AUTO_INCREMENT,
  nomPays varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (idPays) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

### 4.3 - ASSOCIATION BINAIRE 1-\*

UML



SQL

Deux classes en association 1-\* = deux tables dont une avec une FK.

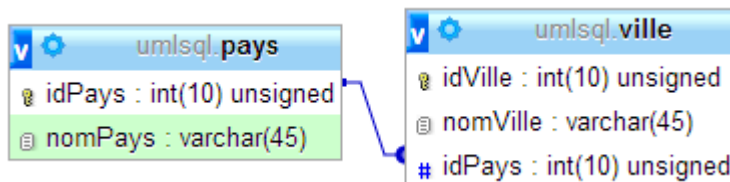


Table pays : la même qu'en 1.2.

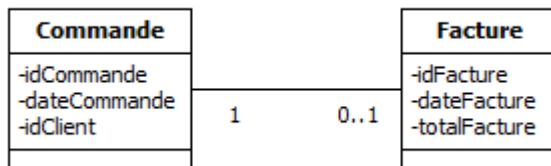
DROP TABLE IF EXISTS ville;

```

CREATE TABLE ville (
  idVille int(10) unsigned NOT NULL AUTO_INCREMENT,
  nomVille varchar(45) COLLATE utf8_general_ci NOT NULL,
  idPays int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (idVille) USING BTREE,
  KEY FK_ville_pays (idPays) USING BTREE,
  CONSTRAINT FK_ville_pays FOREIGN KEY (idPays) REFERENCES pays (idPays) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

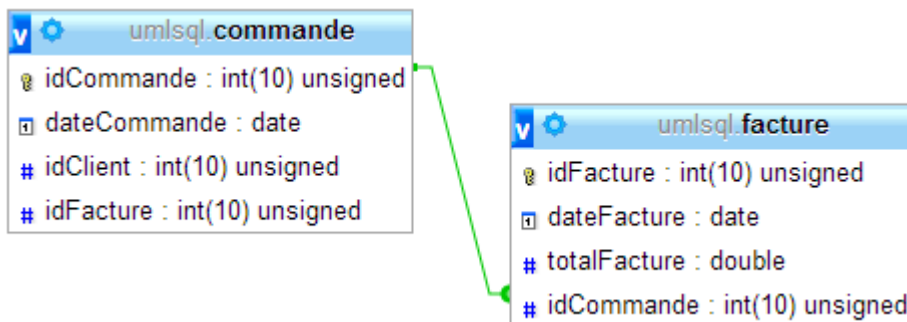
## 4.4 - ASSOCIATION BINAIRE 1-1

### UML



### SQL

Deux classes en association 1-1 = deux tables + 2 FK d'auto-référencement.



```

DROP TABLE IF EXISTS commande;
CREATE TABLE commande (
  idCommande int(10) unsigned NOT NULL AUTO_INCREMENT,
  dateCommande date NOT NULL,
  idClient int(10) unsigned NOT NULL,
  idFacture int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (idCommande)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

```

DROP TABLE IF EXISTS facture;
CREATE TABLE facture (
  idFacture int(10) unsigned NOT NULL AUTO_INCREMENT,
  dateFacture date NOT NULL,
  totalFacture double NOT NULL,
  idCommande int(10) unsigned NULL,
  PRIMARY KEY (idFacture),
  KEY FK_facture_commande (idCommande),
  CONSTRAINT FK_facture_commande FOREIGN KEY (idCommande) REFERENCES
commande (idCommande) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

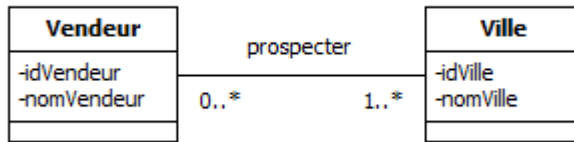
Notez le NULL autorisé dans la colonne `idFacture` de la table `commande`.

Notez le NOT NULL dans la colonne `idCommande` de la table `facture`.

Notez la FK dans la table `Facture` et l'absence de FK dans la table `Commande`.

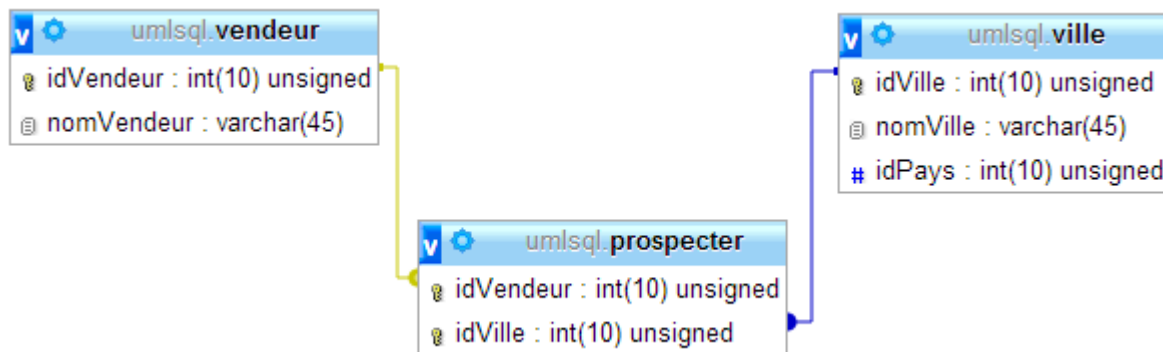
## 4.5 - ASSOCIATION BINAIRE \*-\*

UML



SQL

Deux classes en association \*-\* = trois tables dont une table de jointure avec une PK composée de 2 FK.



La table ville : comme en 1.3.

```

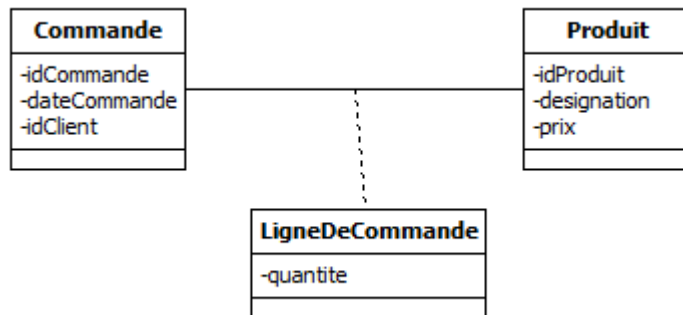
DROP TABLE IF EXISTS vendeur;
CREATE TABLE vendeur (
  idVendeur int(10) unsigned NOT NULL AUTO_INCREMENT,
  nomVendeur varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (idVendeur)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

```

DROP TABLE IF EXISTS prospecteur;
CREATE TABLE prospecteur (
  idVendeur int(10) unsigned NOT NULL,
  idVille int(10) unsigned NOT NULL,
  PRIMARY KEY (idVendeur, idVille),
  KEY FK_prospecteur_ville (idVille),
  CONSTRAINT FK_prospecteur_ville FOREIGN KEY (idVille) REFERENCES ville (idVille),
  CONSTRAINT FK_prospecteur_vendeur FOREIGN KEY (idVendeur) REFERENCES vendeur (idVendeur)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

## 4.6 - UNE CLASSE-ASSOCIATION

### UML



### SQL

Deux classes en association \*-\* et une classe-association = trois tables dont une table de jointure avec une PK composée de 2 FK et des colonnes correspondantes aux attributs de la classe-association.



La table commande : comme en 1.4.

```

DROP TABLE IF EXISTS produit;
CREATE TABLE produit (
  idProduit int(10) unsigned NOT NULL AUTO_INCREMENT,
  designation varchar(45) COLLATE utf8_general_ci NOT NULL,
  prix double NOT NULL,
  PRIMARY KEY (idProduit)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

```

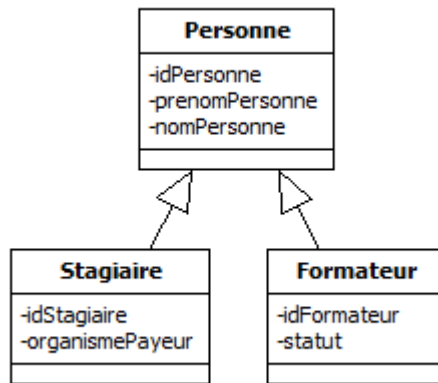
DROP TABLE IF EXISTS linedecommande;
CREATE TABLE linedecommande (
  idCommande int(10) unsigned NOT NULL,
  idProduit int(10) unsigned NOT NULL,
  quantite int(10) unsigned NOT NULL,
  PRIMARY KEY (idCommande,idProduit),
  KEY FK_lignedecommande_produit (idProduit),
  CONSTRAINT FK_lignedecommande_produit FOREIGN KEY (idProduit) REFERENCES
produit (idProduit),
  
```



```
CONSTRAINT FK_lignedecommande_commande FOREIGN KEY (idCommande)  
REFERENCES commande (idCommande)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

## 4.7 - HÉRITAGE

### 4.7.1 - UML



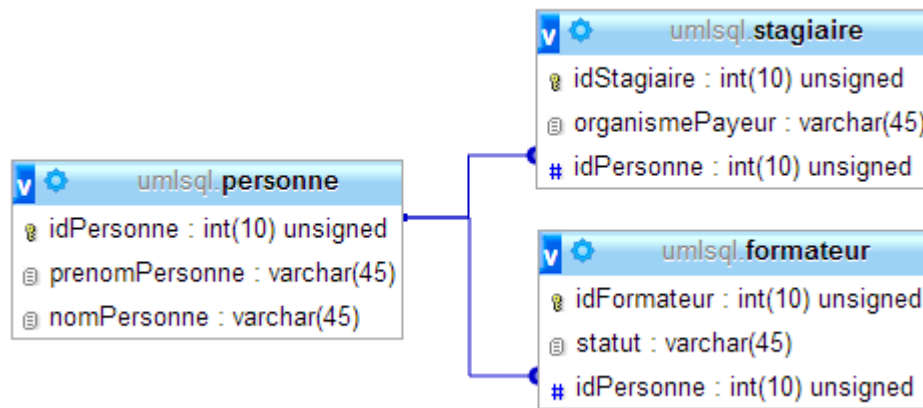
#### Note

Deux types de contraintes sont liées à l'héritage :

- ✓ la distinction (disjonction),
- ✓ la complétude (couverture).

## 4.7.2 - SQL

### 4.7.2.1 - Distinction



```

DROP TABLE IF EXISTS personne;
CREATE TABLE personne (
  idPersonne int(10) unsigned NOT NULL AUTO_INCREMENT,
  prenomPersonne varchar(45) COLLATE utf8_general_ci NOT NULL,
  nomPersonne varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (idPersonne)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

```

DROP TABLE IF EXISTS stagiaire;
CREATE TABLE stagiaire (
  idStagiaire int(10) unsigned NOT NULL AUTO_INCREMENT,
  organismePayeur varchar(45) COLLATE utf8_general_ci NOT NULL,
  idPersonne int(10) unsigned NOT NULL,
  PRIMARY KEY (idStagiaire),
  KEY FK_stagiaire_personne (idPersonne),
  CONSTRAINT FK_stagiaire_personne FOREIGN KEY (idPersonne) REFERENCES personne
(idPersonne)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

```

DROP TABLE IF EXISTS formateur;
CREATE TABLE formateur (
  idFormateur int(10) unsigned NOT NULL AUTO_INCREMENT,
  statut varchar(45) COLLATE utf8_general_ci NOT NULL,
  idPersonne int(10) unsigned NOT NULL,
  PRIMARY KEY (idFormateur),
  KEY FK_formateur_personne (idPersonne),
  CONSTRAINT FK_formateur_personne FOREIGN KEY (idPersonne) REFERENCES personne
(idPersonne)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
  
```

## 4.7.2.2 - Push-Down

umlsql stagiaire
idStagiaire : int(10) unsigned
organismePayeur : varchar(45)
prenomStagiaire : varchar(45)
nomStagiaire : varchar(45)

umlsql formateur
idFormateur : int(10) unsigned
statut : varchar(45)
prenomFormateur : varchar(45)
nomFormateur : varchar(45)

```

DROP TABLE IF EXISTS stagiaire;
CREATE TABLE stagiaire (
  idStagiaire int(10) unsigned NOT NULL AUTO_INCREMENT,
  organismePayeur varchar(45) COLLATE utf8_general_ci NOT NULL,
  prenomStagiaire varchar(45) COLLATE utf8_general_ci NOT NULL,
  nomStagiaire varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (idStagiaire)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

```

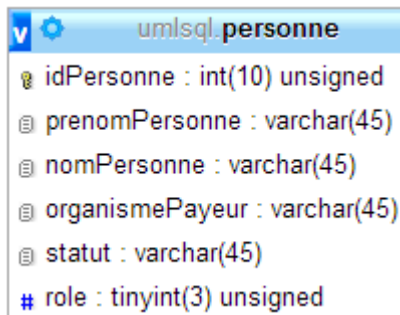
```

DROP TABLE IF EXISTS formateur;
CREATE TABLE formateur (
  idFormateur int(10) unsigned NOT NULL AUTO_INCREMENT,
  statut varchar(45) COLLATE utf8_general_ci NOT NULL,
  prenomFormateur varchar(45) COLLATE utf8_general_ci NOT NULL,
  nomFormateur varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (idFormateur)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

```

Notez la redondance des prénoms et noms !

## 4.7.2.3 - Push-Up



```
DROP TABLE IF EXISTS personne;
CREATE TABLE `personne` (
  `idPersonne` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `prenomPersonne` varchar(45) COLLATE utf8_general_ci NOT NULL,
  `nomPersonne` varchar(45) COLLATE utf8_general_ci NOT NULL,
  `organismePayeur` varchar(45) COLLATE utf8_general_ci DEFAULT NULL,
  `statut` varchar(45) COLLATE utf8_general_ci DEFAULT NULL,
  `role` tinyint(3) unsigned NOT NULL,
  PRIMARY KEY (`idPersonne`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

Notes :

la colonne [role] est facultative,  
notez la haute fréquence de valeurs NULL dans les colonnes,  
notez les NULL autorisés pour les colonnes [organismePayeur] et [statut].

Il serait aussi possible de créer en plus deux vues (CREATE VIEW ...).

## **4.8 - AGRÉGATION ET COMPOSITION**

La même qu'en 1.5 ie avec Commande et Lignedecommande.

Mais avec un DELETE CASCADE pour la Composition.

## 4.9 - QUELQUES CONTRAINTES

### 4.9.1 - Partition stagiaire-formateur

Nous partons du modèle issu de la distinction.

Personne(idPersonne, prenomPersonne, nomPersonne, role)

Formateur(idFormateur, statut, prenomFormateur, nomFormateur, idPersonne)

Stagiaire(idStagiaire, organismePayeur, prenomStagiaire, nomStagiaire, idPersonne)

Ajout dans Formateur ou dans Stagiaire avec une gestion d'exception.

Ce pourrait être avec un WHERE EXISTS.

**Ajout d'un formateur :**

DELIMITER \$\$

DROP PROCEDURE IF EXISTS `umlsql`.`formateurAjout` \$\$

CREATE PROCEDURE `umlsql`.`formateurAjout` (asPrenom VARCHAR(50), asNom VARCHAR(50), asStatut VARCHAR(50))  
BEGIN

DECLARE codeErreur INT DEFAULT 0;

DECLARE compte INT DEFAULT 0;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET codeErreur = -1;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET codeErreur = -1;

SELECT COUNT(\*) INTO **codeErreur** FROM stagiaire WHERE prenomStagiaire = asPrenom AND nomStagiaire = asNom;

IF codeErreur = 0 THEN

START TRANSACTION;

INSERT INTO personne(prenomPersonne, nomPersonne, role)  
VALUES(asPrenom, asNom, 1);

INSERT INTO formateur(statut, prenomFormateur, nomFormateur, idPersonne)  
VALUES(asStatut, asPrenom, asNom, LAST\_INSERT\_ID());

IF codeErreur = 0 THEN  
COMMIT;

ELSE  
ROLLBACK;

END IF;

END IF;

END \$\$

DELIMITER ;

**Le test :**

CALL formateurAjout('Bianca', 'Castafiore', 'Salarié');



**Ajout d'un stagiaire :**

DELIMITER \$\$

DROP PROCEDURE IF EXISTS `uysql`.`stagiaireAjout` \$\$

CREATE PROCEDURE `uysql`.`stagiaireAjout` (asPrenom VARCHAR(50), asNom VARCHAR(50), asOrganisme VARCHAR(50))

BEGIN

DECLARE codeErreur INT DEFAULT 0;

DECLARE compte INT DEFAULT 0;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET codeErreur = -1;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET codeErreur = -1;

SELECT COUNT(\*) INTO **codeErreur** FROM formateur WHERE prenomFormateur = asPrenom  
AND nomFormateur = asNom;

IF codeErreur = 0 THEN

START TRANSACTION;

INSERT INTO **personne**(prenomPersonne, nomPersonne, role)  
VALUES(asPrenom, asNom, 2);

INSERT INTO **stagiaire**(organismePayeur, prenomStagiaire, nomStagiaire, idPersonne)  
VALUES(asOrganisme, asPrenom, asNom, LAST\_INSERT\_ID());

IF codeErreur = 0 THEN

COMMIT;

ELSE

ROLLBACK;

END IF;

END IF;

END \$\$

DELIMITER ;

**Le test :**

-- Ca casse

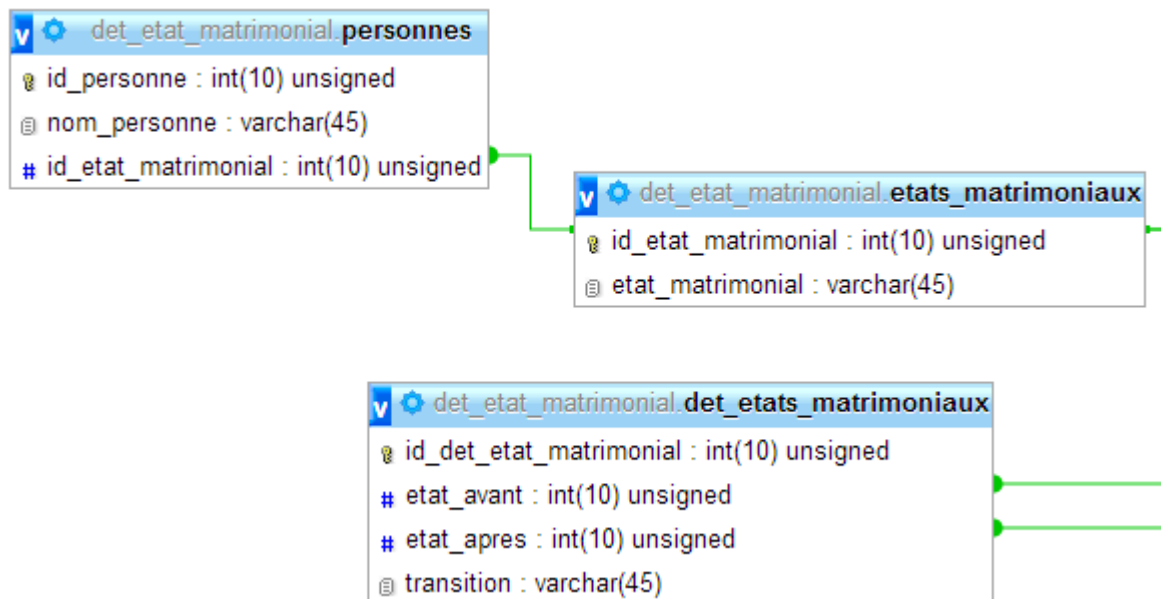
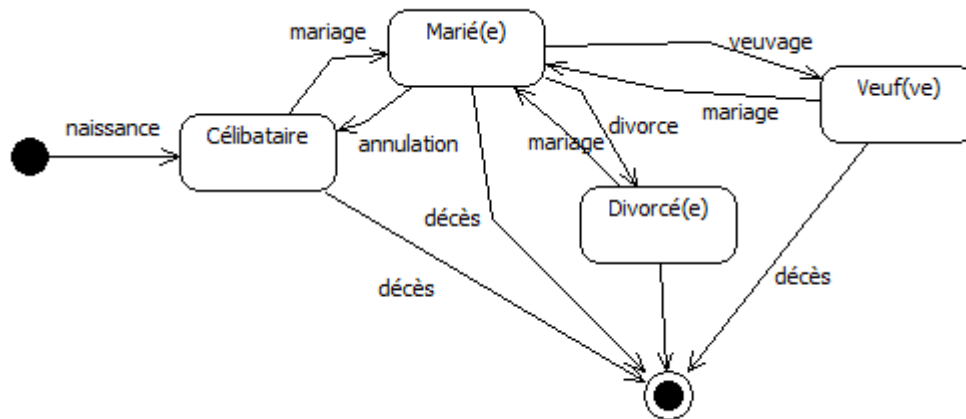
CALL stagiaireAjout('Bianca', 'Castafiore', 'Region');

-- Ca passe

CALL stagiaireAjout('Milou', 'Le chien', 'Region');

### 4.9.2 - DET état-civil

Contrôler le passage d'un état à une autre.



**Le script de création de la BD avec 2 insertions.**

```
DROP DATABASE IF EXISTS det_etat_matrimonial;
```

```
CREATE DATABASE IF NOT EXISTS det_etat_matrimonial
DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;
```

```
USE det_etat_matrimonial;
```

```
SET FOREIGN_KEY_CHECKS=0;
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT=0;
START TRANSACTION;
```

```
DROP TABLE IF EXISTS det_etats_matrimoniaux;
CREATE TABLE IF NOT EXISTS det_etats_matrimoniaux (
  id_det_etat_matrimonial int(10) unsigned NOT NULL AUTO_INCREMENT,
  etat_avant int(10) unsigned NOT NULL,
  etat_apres int(10) unsigned NOT NULL,
  transition varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (id_det_etat_matrimonial),
  KEY FK_det_etats_matrimoniaux_1 (etat_avant),
  KEY FK_det_etats_matrimoniaux_2 (etat_apres)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

```
INSERT INTO det_etats_matrimoniaux (id_det_etat_matrimonial, etat_avant, etat_apres, transition)
VALUES
(1, 1, 2, 'Mariage'),
(2, 2, 1, 'Annulation du mariage'),
(3, 2, 3, 'Divorce'),
(4, 3, 2, 'Mariage'),
(5, 2, 4, 'Veuvage'),
(6, 4, 2, 'Mariage');
```

```
DROP TABLE IF EXISTS etats_matrimoniaux;
CREATE TABLE IF NOT EXISTS etats_matrimoniaux (
  id_etat_matrimonial int(10) unsigned NOT NULL AUTO_INCREMENT,
  etat_matrimonial varchar(45) COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY (id_etat_matrimonial),
  KEY Index_em_etat_matrimonial (etat_matrimonial) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

```
INSERT INTO etats_matrimoniaux (id_etat_matrimonial, etat_matrimonial) VALUES
(1, 'Célibataire'),
(3, 'Divorcé(e)'),
(2, 'Marié(e)'),
(4, 'Veuf(ve)');
```

```
DROP TABLE IF EXISTS personnes;
CREATE TABLE IF NOT EXISTS personnes (
  id_personne int(10) unsigned NOT NULL AUTO_INCREMENT,
```

```
nom_personne varchar(45) COLLATE utf8_general_ci NOT NULL,  
id_etat_matrimonial int(10) unsigned NOT NULL,  
PRIMARY KEY (id_personne),  
KEY FK_personnes_ec (id_etat_matrimonial)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

```
INSERT INTO personnes (id_personne, nom_personne, id_etat_matrimonial) VALUES  
(1, 'Tintin', 1),  
(3, 'Milou', 1);
```

```
ALTER TABLE `det_etats_matrimoniaux`  
  ADD CONSTRAINT FK_det_etats_matrimoniaux_1 FOREIGN KEY (etat_avant) REFERENCES  
etats_matrimoniaux (id_etat_matrimonial),  
  ADD CONSTRAINT FK_det_etats_matrimoniaux_2 FOREIGN KEY (etat_apres) REFERENCES  
etats_matrimoniaux (id_etat_matrimonial);
```

```
ALTER TABLE `personnes`  
  ADD CONSTRAINT FK_personnes_ec FOREIGN KEY (id_etat_matrimonial) REFERENCES  
etats_matrimoniaux (id_etat_matrimonial);  
SET FOREIGN_KEY_CHECKS=1;  
COMMIT;
```

**Le trigger de contrôle :**

```
DROP TRIGGER IF EXISTS `personnes_before_update`;

DELIMITER //

CREATE TRIGGER `personnes_before_update` BEFORE UPDATE ON `personnes`
FOR EACH ROW BEGIN
    DECLARE nCount INT;

    SELECT COUNT(*) INTO nCount
    FROM det_etats_matrimoniaux
    WHERE etat_avant = OLD.id_etat_matrimonial AND etat_apres = NEW.id_etat_matrimonial;

    -- Si on ne trouve pas la transition
    IF nCount = 0 THEN
        SIGNAL SQLSTATE '10001';
        -- SET NEW.id_etat_matrimonial = OLD.id_etat_matrimonial;
    END IF;
END
//

DELIMITER ;
```

**La procédure stockée de modification (changement d'état).**

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `det_etat_matrimonial`.`changerEtat` $$
CREATE PROCEDURE `det_etat_matrimonial`.`changerEtat` (asNom VARCHAR(50), aiEtat INT)
BEGIN
    UPDATE personnes
    SET id_etat_matrimonial = aiEtat
    WHERE nom_personne = asNom;
END $$

DELIMITER ;
```

**Le test de la procédure stockée.**

```
KO : affiche Unhandled user-defined exception condition
CALL changerEtat('Tintin', 1);

OK
CALL changerEtat('Tintin', 2);
```

## **4.10 - RÉTRO-INGÉNIERIE**

Penser à la rétro-ingénierie ou rétro-conception (reverse engineering) pour créer un Diagramme de classes à partir d'une BD ou d'un script SQL de création de tables (PowerAMC fait cela très bien).

## **CHAPITRE 5 - ANNEXES**

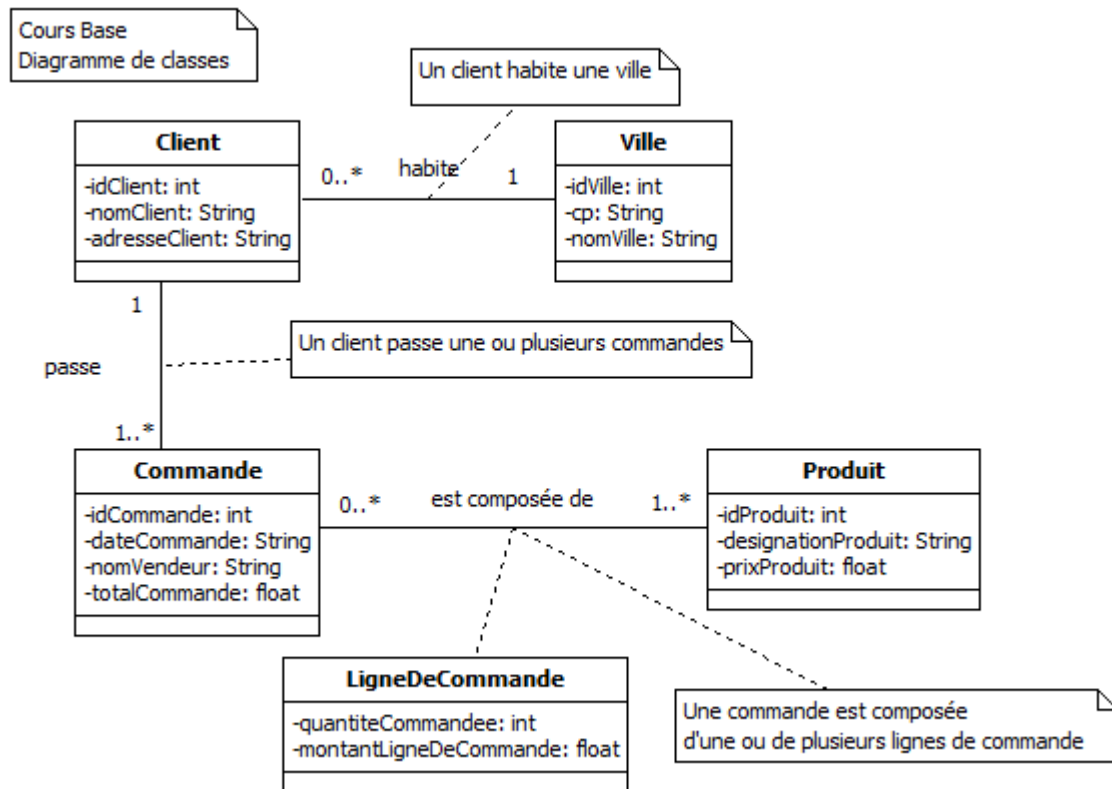
## **5.1 - QUELQUES TYPOLOGIES DE CLASSES**

### 5.1.1.1 - Boundary / Control / Entity



## 5.2 - LE MODÈLE COURS

### 5.2.1 - DCL (Diagramme de classes)

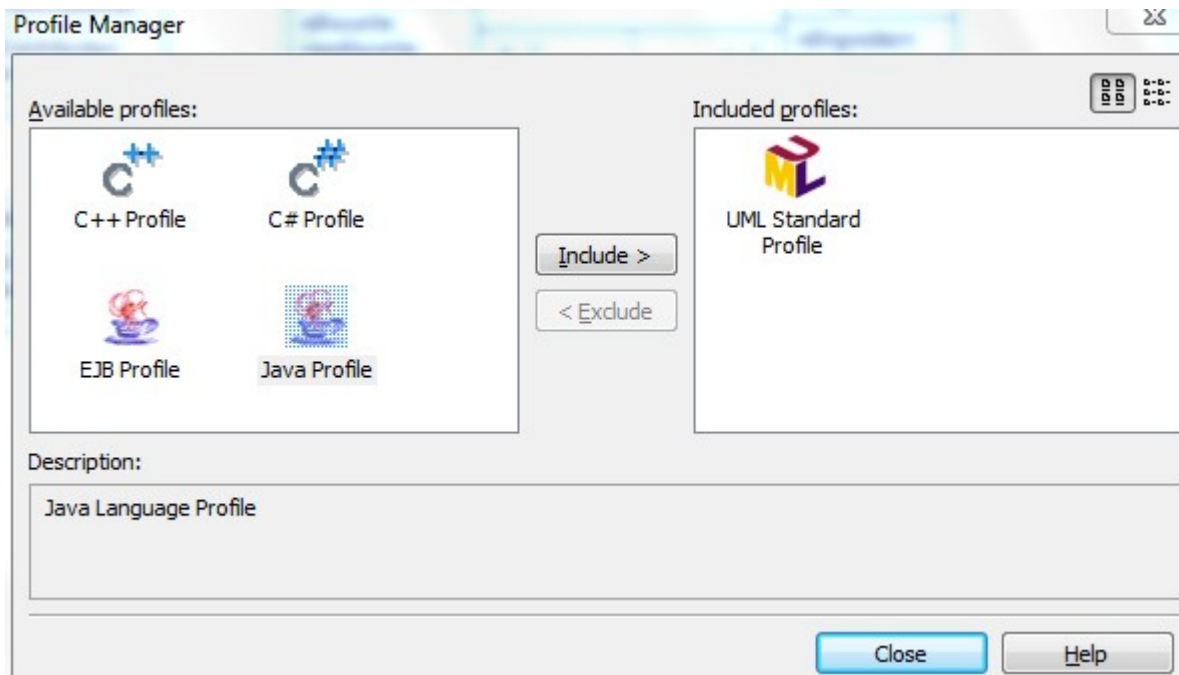


## 5.2.2 - Codes java

### 5.2.2.1 - Utiliser StarUML pour générer du code Java.

**Première étape** : Ajouter le profil Java pour avoir les types Java.

Model/Profiles/Sélectionnez Java Profile/Cliquez sur Include



**Deuxième étape** : générer le code Java.

Tools/Java/Generate Code

Sélectionnez le package,  
sélectionnez les éléments,  
sélectionnez l'Output Directory,

The screenshot shows the 'Java Code Generation' dialog box, specifically the 'Option Setup' tab. The dialog is titled 'Option Setup' and has a subtitle 'Configure options for code generation.' It is divided into several sections:

- Generation Options:** Contains three checkboxes: 'Generate unnamed AssociationEnd' (checked), 'Generate the Documentation by JavaDoc' (checked), and 'Generate empty Java Doc' (unchecked).
- Code Style:** Contains two checkboxes: 'Place opening curly brace "{" in the new line' (unchecked) and 'Insert tab as space' (unchecked). Below these is a 'Tab width:' field set to '4'.
- File Header Comment:** A text area containing the following text:

```
///  
Generated by StarUML(tm) Java Add-In  
///  
@@ Project : @p  
@@ File Name : @f  
@@ Date : @d  
@@ Author : @a
```
- Description:** A list of character codes and their meanings:
  - @p : Title
  - @d : Date
  - @c : Company
  - @a : Author
  - @r : Copyright
  - @f : File name
  - @e : Element name
  - @@ : Character@

At the bottom of the dialog are three buttons: '< Back', 'Next >', and 'Cancel'.

### 5.2.2.2 - Code généré par StarUML !!!

```
public class Ville {
    private int idVille;
    private String cp;
    private String nomVille;
    public Client Unnamed1;
}

public class Client {
    private int idClient;
    private String nomClient;
    private String adresseClient;
    public Ville Unnamed1;
    public Commande Unnamed2;
}

public class Commande {
    private int idCommande;
    private String dateCommande;
    private String nomVendeur;
    private float totalCommande;
    public Client Unnamed1;
    public Produit Unnamed2;
}

public class LigneDeCommande {
    private int quantiteCommandee;
    private float montantLigneDeCommande;
}

public class Produit {
    private int idProduit;
    private String designationProduit;
    private float prixProduit;
    public Commande Unnamed1;
}
```

## 5.3 - AUTRES EXERCICES

### 5.3.1 - Chevaux

#### Modélisation des courses de chevaux.

Vous établirez le Diagramme de Classes à partir des DF et du GDF en partant du DD et des règles de gestion qui suivent.

- 1 - Type de course (Tiercé, Quarté, ...)
- 2 - Numéro de la course
- 3 - Désignation de la course
- 4 - Nom du champ de course
- 5 - Date de la course
- 6 - Catégorie de la course (Trot attelé, Trot monté, Obstacle, ...)
- 7 - Dotation de la course
- 8 - Nom du cheval
- 9 - N° du dossard du jockey et du cheval pour la course
- 10 - Nom du propriétaire du cheval
- 11 - Gains du cheval depuis le début de la saison
- 12 - Sexe du cheval
- 13 - Nom du jockey
- 14 - Date de naissance du cheval

#### Règles complémentaires

Un champ de courses est équipé pour une ou des catégories de courses.

Un cheval a une parenté ascendante et éventuellement descendante.

Une course est d'une catégorie et d'une seule.

Les données sont conservées une seule année.

### 5.3.2 - Bateaux

#### Modélisation de la tarification de location de bateaux en Méditerranée pour la société X.

Vous établirez le Diagramme de Classes à partir des DF et du GDF en partant du DD.

##### GRECE

Athènes, Rhodes, Kos, Corfou, Porto Carras

	Nbre de Personnes	Hiver		Mi-Saison		Haute-Saison		Automne	
		1/1/2002 - 25/4/2002		26/4/2002 - 26/6/2002		27/6/2002 - 27/8/2002		28/8/2002 - 31/12/2002	
		1 sem.	2 sem.	1 sem.	2 sem.	1 sem.	2 sem.	1 sem.	2 sem.
	Gamme "EXCLUSIVE" - Monocoques								
Moorings 444 & S.Od.45	8 à 10	5000	9000	6000	11000	7000	12000	6000	11000
Moorings S.Od.42	8 à 10	4500	8000	5000	10000	6000	11000	5000	10000
Moorings 405	6 à 8	4275	7600	4750	9500	5700	10450	4750	9500
Gamme "CLUB" - Monocoques									
Moorings 500	10 à 12	5000	9000	6000	11000	7000	12000	6000	11000
Moorings 444 & S.Od.44	8 à 10	4500	8000	5000	10000	6000	11000	5000	10000
Moorings 45	8 à 10	4275	7600	4750	9500	5700	10450	4750	9500
Moorings 405	6 à 8	4061	7220	4513	9025	5415	9928	4513	9025
Moorings Sun Od.37	6 à 8	3858	6859	4287	8574	5144	9431	4287	8574
Moorings 353 & S.Od.36	6 à 8	3665	6516	4073	8145	4887	8960	4073	8145
Moorings Sun Od.33	6 à 8	3482	6190	3869	7738	4643	8512	3869	7738
Moorings Sun Od.31	6 à 8	3308	5881	3675	7351	4411	8086	3675	7351
Gamme "VALUE" - Monocoques									
Moorings 500	10 à 12	4500	8000	5000	9000	6000	11000	5000	9000
Moorings S. Mag 44	8 à 10	4365	7760	4850	8730	5820	10670	4850	8730
Moorings 430	8 à 10	4234	7527	4705	8468	5645	10350	4705	8468
Moorings 390	6 à 9	4107	7301	4563	8214	5476	10039	4563	8214
Sun Light 30	4 à 6	3984	7082	4426	7968	5312	9738	4426	7968

##### TURQUIE

Marmaris, Finike (Transit Log 50 € non compris)

MEMES DATES ET SAISONS QUE LA GRECE									
<b>Gamme "EXCLUSIVE" - Monocoques</b>									
Moorings 44	8 à 10	3000	5000	4000	7000	5000	9000	4000	7000
<b>Gamme "CLUB" - Monocoques</b>									
Moorings Sun Od.51	10 à 12	4000	7000	5000	9000	6000	10000	5000	9000
Moorings Sun Od.44	8 à 10	3800	6650	4750	8550	5700	9500	4750	8550
Moorings 405	6 à 8	3610	6318	4513	8123	5415	9025	4513	8123
Moorings 353 & S.Od.36	6 à 8	3430	6002	4287	7716	5144	8574	4287	7716
<b>Gamme "CLUB" - Catamarans</b>									
Moorings 4100	8 à 10	3665	6516	4073	8145	4887	8960	4073	8145
<b>Gamme "VALUE" - Monocoques</b>									
Moorings 500	10 à 12	3000	5000	4000	7000	5000	9000	4000	7000
Moorings 430	8 à 10	2850	4750	3800	6650	4750	8550	3800	6650
Moorings 390	6 à 9	2708	4513	3610	6318	4513	8123	3610	6318
Moorings 350	6 à 8	2572	4287	3430	6002	4287	7716	3430	6002

### **5.3.3 - MusikScope**

Cf pariscopes rubrique Musique.

### 5.3.4 - Élections

Modélisation de l'organisation des élections politiques à Paris.

Code	Règle
R1	Lors d'une élection des partis politiques se présentent avec un candidat ou un candidat tête de liste en fonction du type de scrutin.
R2	Les opérations de votes sont réalisées dans des bureaux de votes.
R3	Un bureau de vote est caractérisé par un numéro (numéro d'arrondissement + numéro de bureau), un lieu (Mairie, école, ...), une adresse, ...
R4	Le personnel du bureau de votes appartient le plus souvent à un parti politique qu'il représente. Il peut faire partie du personnel de mairie (En tant qu'assesseur). Il peut être représentant du Maire (En tant que Président du bureau).
R5	Le personnel doit être électeur de la commune donc résident et inscrit sur les listes électorales parisiennes, sachant que les listes électorales sont gérées par arrondissement.
R6	Le personnel tient un rôle particulier et un seul au sein du bureau (Président de bureau, assesseur, délégué politique, suppléant, ...).
R7	Il est nécessaire de connaître le nom, le prénom, l'adresse et la date de naissance du personnel.
R8	A l'issue du dépouillement le président du bureau de vote communique le nombre d'inscrits, le nombre de votants, le nombre de bulletins nuls, le nombres d'exprimés, et le nombre de procurations.
R9	A l'issue du dépouillement le président du bureau de vote communique le nombre de voix par parti.

Pas de DF ni de GDF. Démarche via la méthode de l'analyse du discours et/ou des règles de gestion.



## 5.4 - STARUML ET LES DESIGN PATTERNS

Tools/Apply Pattern ...

