Chapitre 4 : Diagramme de classes et d'objets

Introduction

Le diagramme de classe constitue l'un des pivots essentiels de la modélisation avec UML. En effet, ce diagramme permet de donner la représentation statique du système à développer. Cette représentation est centrée sur les concepts de classe et d'association.

Diagramme de classe

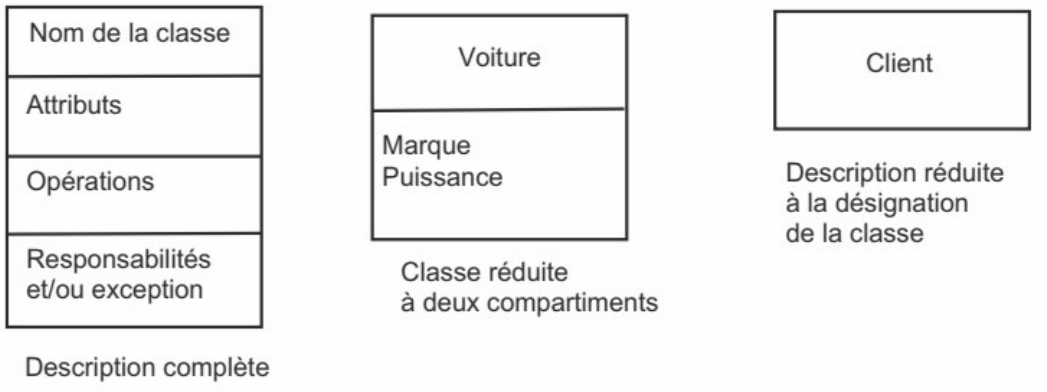
La description du diagramme de classe est fondée sur :

- le concept d'objet,
- le concept de classe comprenant les attributs et les opérations,
- les différents types d'association entre classes.

1. Classe

Une **classe** décrit un groupe d'objets ayant les mêmes propriétés (attributs), un même comportement (opérations), et une sémantique commune (domaine de définition).

Une classe se représente à l'aide d'un rectangle comportant plusieurs compartiments comme c'est illustré par la figure suivante :



Ainsi une classe est représenté par un trois compartiments de base : la désignation de la classe, la description des attributs et la description des opérations.

- **NB :** Deux autres compartiments peuvent être aussi indiqués : la **description des responsabilités** de la classe et la description des **exceptions** traitées par la classe.

1.1. Attribut

Un **attribut** est une propriété élémentaire d'une classe. Il doit doivent respecter le formalisme suivant

Visibilité Nom_attribut : type [= valeur initiale {propriétés}]

– ***Visibilité*** : peut être de type public, protégé, privé ou paquetage. Les symboles + (public), # (protégé), - (privé) et ~ (paquetage) sont indiqués devant chaque attribut ou opération (pour plus de détail voir votre cours de programmation orienté objet).

– ***Nom d'attribut*** : nom unique dans sa classe.

– ***Type*** : type primitif (entier, chaîne de caractères…) dépendant des types disponibles dans le langage d'implémentation ou type classe matérialisant un lien avec une autre classe.

– ***Valeur initiale*** : valeur facultative donnée à l'initialisation d'un objet de la classe.

– {***propriétés***} : valeurs marquées facultatives (ex. : « interdit » pour mise à jour interdite).

NB :

- Un attribut dont la valeur peut être calculée à partir d'autres attributs de la classe est un attribut dérivé qui se note « /nom de l'attribut dérivé ».
- Si on souligne un attribut ou une opération, cela veut que cet attribut ou cet **opération** peut être défini non pas au niveau des instances d'une classe, mais au niveau de la classe. Il s'agit soit d'un attribut qui est une constante pour toutes les instances d'une classe soit d'une opération d'une classe abstraite.
- Rappel sur les modes de visibilité :
    - **Public (+)** – Attribut ou opération visible par tous.
    - **Protégé (#)** – Attribut ou opération visible seulement à l'intérieur de la classe et pour toutes les sous-classes de la classe.
    - • **Privé (-)** – Attribut ou opération seulement visible à l'intérieur de la classe.
    - • **Paquetage (~)** – Attribut ou opération ou classe seulement visible à l'intérieur du paquetage où se trouve la classe.

1.2. Opération

Une **opération** est une fonction applicable aux objets d'une classe. Une opération permet de décrire le comportement d'un objet. Une **méthode** est l'implémentation d'une opération. Voici le formalisme à respecter :

Visibilité Nom_opération (paramètres) [:[type résultat] {propriétés}]

– ***Visibilité*** **:** La même chose que pour la visibilité des attributs.

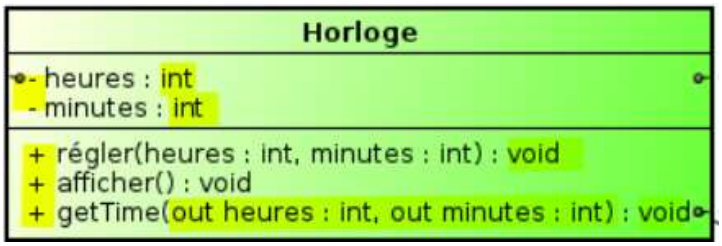
– ***Nom d'opération*** : utiliser un verbe représentant l'action à réaliser.

– ***Paramètres*** : liste de paramètres (chaque paramètre peut être décrit, en plus de son nom, par son type et sa valeur par défaut). L'absence de paramètre est indiquée par ( ).

– ***Type résultat*** : type de valeur retourné dépendant des types disponibles dans le langage d'implémentation. Par défaut, une opération ne retourne pas de valeur, ceci est indiqué par exemple par le mot réservé « ***void*** » dans le langage C++ ou Java.

– ***{propriétés}*** : valeurs facultatives applicables (ex. : {query} pour un comportement sans influence sur l'état du système).

❖ **Exemple d'une classe :**



2. Association

Une **association entre classes** représente les liens qui existent entre les instances de ces classes. La figure suivante représente une association (Posséder) entre deux classes : Personne et Voiture.

2.1. Rôle d'association

Le **rôle** tenu par une classe vis-à-vis d'une association peut être précisé sur l'association. La figure suivante représente les rôles (*employé*) et (*employeur*) :

- Une personne *est employée* chez l'entreprise,
- L'entreprise *est l'employeur* d'une personne.

2.2. Multiplicité

La multiplicité indique un domaine de valeurs pour préciser le nombre d'instance d'une classe vis-à-vis d'une autre classe pour une association donnée.

Le domaine de valeurs est décrit selon plusieurs formes :

• **Intervalle fermé** – Exemple : 2..15.

• **Valeurs exactes** – Exemple : 3, 5, 8.

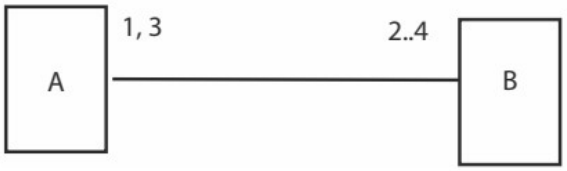
• **Valeur indéterminée notée *** – Exemple : 1..*.

- Dans le cas où l'on utilise seulement *, cela traduit une multiplicité 0..*.

– Dans le cas de multiplicité d'associations, il faut indiquer les valeurs minimale et maximale d'instances d'une classe vis-à-vis d'une instance d'une autre classe.

- À une instance de A correspond 0 ou 1 instance de B.
- À une instance de B correspond 0 à nombre non déterminé d'instances de A.

- À une instance de A correspond 1 à un nombre non déterminé d'instances de B.
- À une instance de B correspond 2 à 10 instances de A.

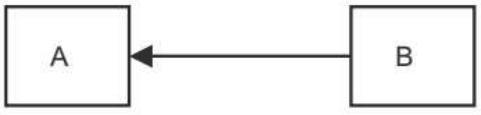


- À une instance de A correspond 2 à 4 instances de B.
- À une instance de B correspond 1 ou 3 instances de A.

2.3. Navigabilité

La **navigabilité** indique si l'association fonctionne de manière unidirectionnelle ou bidirectionnelle, elle est matérialisée par une ou deux extrémités fléchées. La non navigabilité se représente par un « X ». La figure suivante représente les différents cas possible de navigabilité.

- Navigabilité unidirectionnelle de A vers B.
- Pas de navigabilité de B vers A (non définie explicitement).

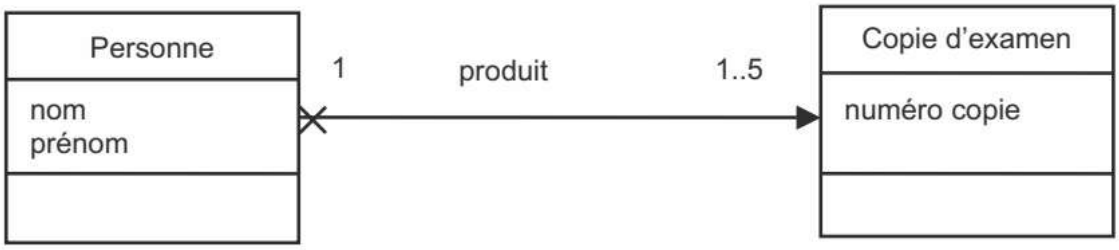


- Navigabilité unidirectionnelle de B vers A.
- Navigabilité de A vers B (non définie explicitement).

- Navigabilité bidirectionnelle entre A et B. Habituellement représentée sans flèche.

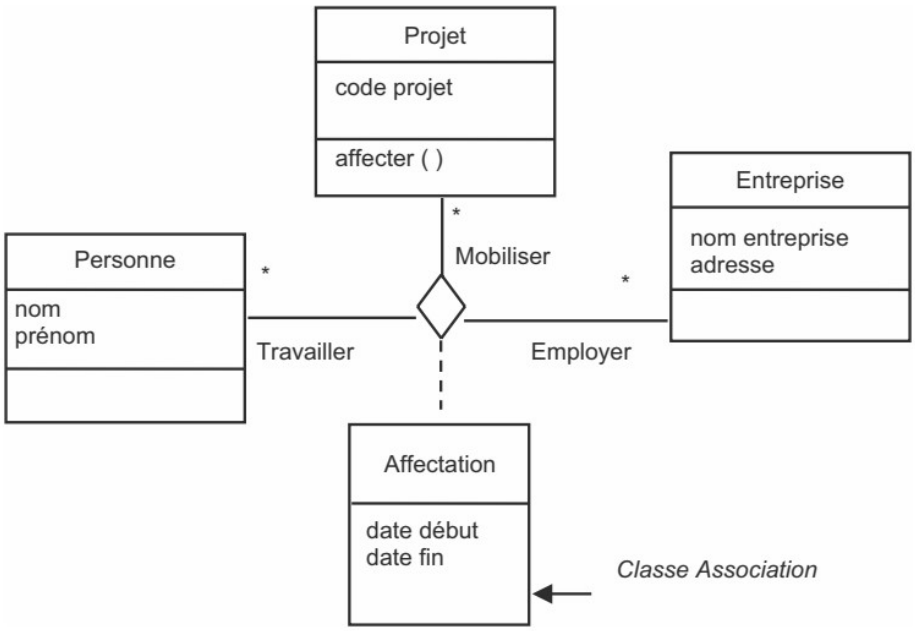
- **Exemple :**

Dans cet exemple à une personne sont associées ses copies d'examen mais l'inverse n'est pas possible (retrouver directement l'auteur de la copie d'examen n'est pas possible notamment avant la correction de la copie).



3. classe-association

Une association de dimension supérieure à 2 se représente en utilisant un losange permettant de relier toutes les classes concernées. Cette classe-association est reliée par un trait en pointillé au losange de connexion.

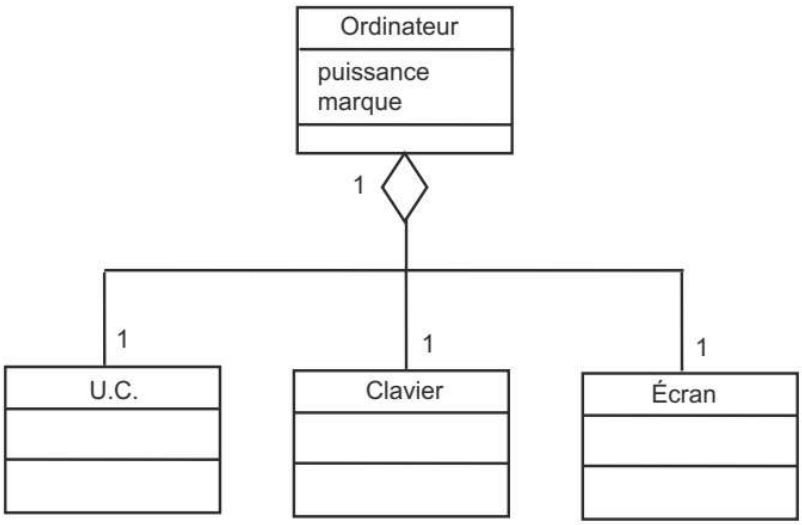


Dans cet exemple une classe association « Affectation » est relie aux 3 autres classes. Elle porte des attributs (*Date_début* et *Date_fin*) pour permettre de décrire ces attributs.

3.1. Agrégation et composition entre classes

3.1.1. Agrégation

L'agrégation est une association qui permet de représenter un lien de type « *ensemble* » comprenant des « *éléments* ». La figure suivante représente un exemple d'agrégation.

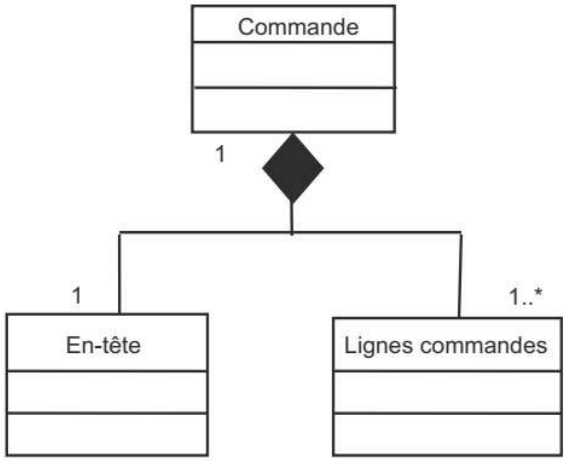


Ici un « *Ordinateur* » est un ensemble composé de « *U.C* », « Clavier » et « Ecran ».

3.1.2. Composition

La composition est une relation d'agrégation dans laquelle il existe une contrainte de durée de vie entre la classe « composant » et la ou les classes « composé ». Autrement dit la suppression de la classe « composé » implique la suppression de la ou des classes « composant ».

.



Dans cet exemple une « *commande* » est composée d'« *entête* » (par exemple N° de commande et sa date) et des « *Ligne de commande* » (représentant les produits commandés). Ici l'association est de type « **compossition** » car la suppression de la commande entrainera la suppression de tous ces entêtes ainsi que la suppression de toutes ces lignes de commandes (les produits commandés).

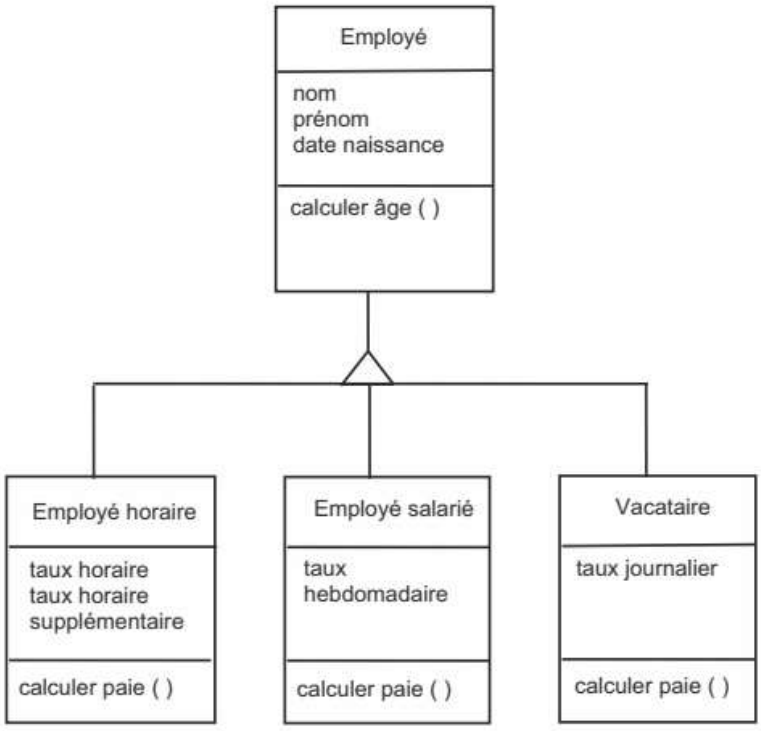
3.1.3. Généralisation et spécialisation

La généralisation est la relation entre une classe (super-classe) et deux autres classes ou plus (sous-classe) partageant un sous-ensemble commun d'attributs et/ou d'opérations. L'opération qui consiste à créer une super-classe à partir de classes s'appelle la **généralisation**. Inversement la **spécialisation** ou **héritage** consiste à créer des sous-classes à partir d'une classe.

L'**héritage** permet à une sous-classe de disposer des attributs et opérations de la classe dont elle dépend.

Une **classe abstraite** est une classe qui n'a pas d'instance directe mais dont les classes descendantes ont des instances. Dans une relation d'héritage, la super-classe est par définition une classe abstraite.

Dans l'exemple suivant trois sous-classe («Employé horaire», «*Employé salarié*», « *vacataire*») hérite de la super-classe « *Employé* » qui est une classe abstraite.



Notant aussi que les attributs *nom*, *prénom* et *date de naissance* et l'opération « *calculer âge* » de « *Employé* » sont hérités par les trois sous-classes.

- **NB :** il existe un autre type d'héritage ou une sous-classe hérite de plusieurs super-classe. Ce type est dit **héritage multiple**. L'héritage multiple n'est pas supporté par tous les langages de programmation en l'occurrence Java et C.

- **Exemple récapitulatif :**

On veut modéliser la vue statique du système d'information d'une entreprise qui assure le suivi des projets. Cette entreprise comprend plusieurs départements où chacun d'entre eux est dirigé par un seul employé comme il comprend plusieurs employés dont on veut garder dans le système, leurs dates de prise de fonction dans ce département. Les employés sont supervisés par un seul employé (et celui-ci supervise plusieurs employés).

Chaque département coordonne plusieurs projets dont on affecte pour chaque projet un ensemble d'employés. Le système doit garder aussi le nombre d'heures assuré par un employé pour chaque projet.

Finalement on veut modéliser, pour chaque employé les personnes à son charge avec le lien de parenté de cette personne avec cet employé.

- **Solution :**

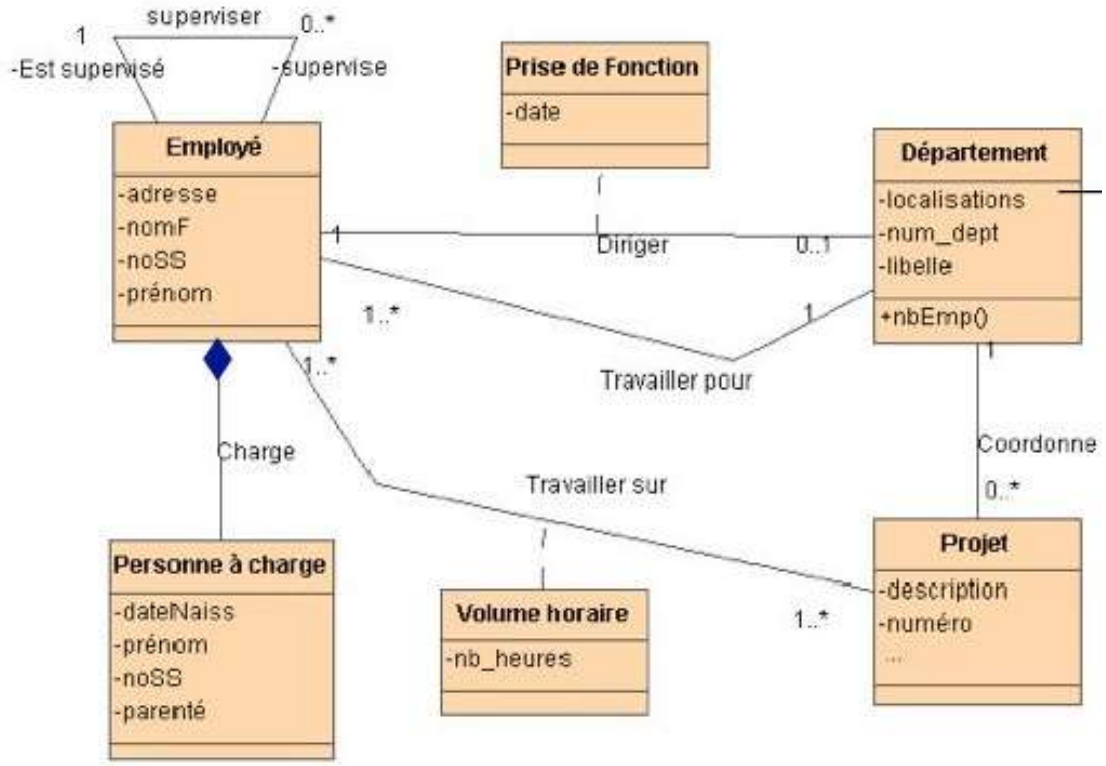
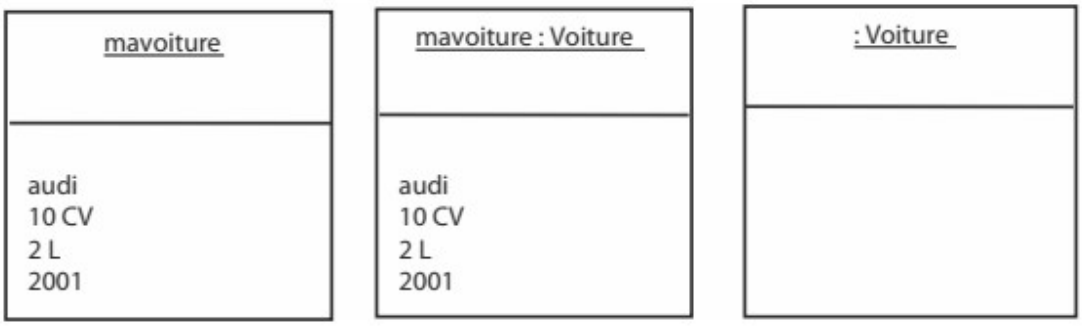

superviser
1
0..*
-Est supervisé
-supervise
Employé
-adresse
-nomF
-noSS
-prénom
Prise de Fonction
-date
Diriger
1
0..1
Département
-localisations
-num_dept
-libelle
+nbEmp()
1..*
1
Travailler pour
1..*
Charge
Travailler sur
Coordonne
1
0..*
Personne à charge
-dateNaiss
-prénom
-noSS
-parenté
Volume horaire
-nb_heures
1..*
Projet
-description
-numéro
...

Diagramme d'objet

1. Les diagrammes d'objet ou d'instance montrent des objets et des liens. Leurs notation est dérivée de celle des diagrammes de classe ; les éléments qui sont des instances sont soulignés. La figure suivante montre les représentations possibles.



Ce type de diagramme est utilisé pour faciliter la compréhension des structures des données complexes, comme les structures récursive ou pour monter un contexte par exemple avant ou après une interaction.

- **Exemple 1 :**

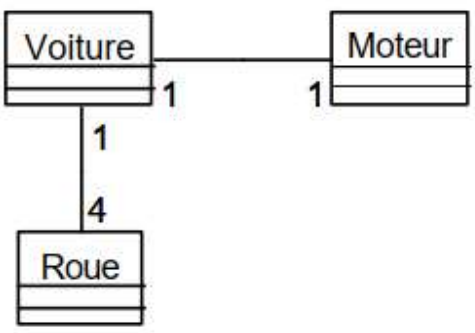


Diagramme de classes

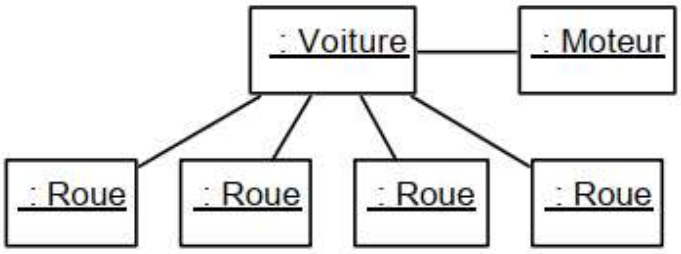
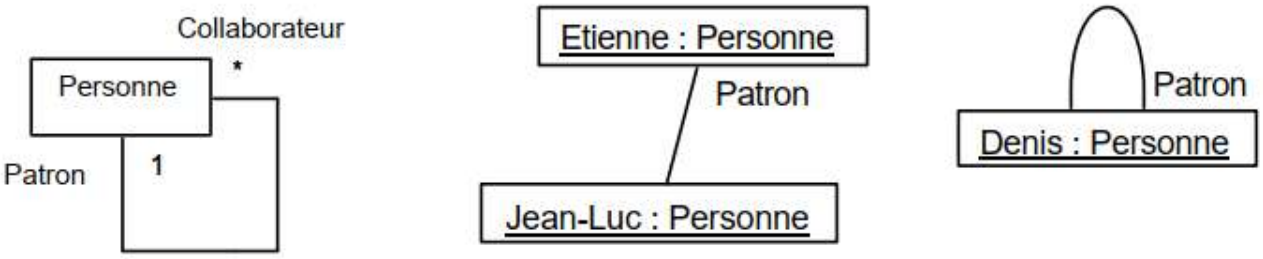


Diagramme d'objets

D'une façon générale, la représentation concrète d'une structure par les objets est plus parlante que celle abstraite représentée par des classes comme c'est montré par l'exemple ci-dessus.

- **Exemple 2 : Relation réflexive**



Les liens d'instance montrent mieux la relation réflexive *Patron*. Les digrammes d'objets montrent qu'Etienne est le patron de *Jean-Luc*, *Denis* est le patron de lui-même.