Chap 4 Diagramme de classes et d'Objet (Vue statique)

-1. Introduction

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation. Alors que le diagramme de cas d'utilisation montre un système du point de vue des acteurs, le diagramme de classes en montre la structure interne. Il permet de fournir une représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation. Il s'agit d'une vue statique, car on ne tient pas compte du facteur temporel dans le comportement du système.

-2. Les objets

Un objet est une abstraction d'un élément du monde réel. Il possède des informations, par exemple nom, prénom, adresse, etc., et se comporte suivant un ensemble d'opérations qui lui sont applicables. De plus, un ensemble d'attributs caractérisent l'état d'un objet, et l'on dispose d'un ensemble d'opérations (les méthodes) qui permettent d'agir sur le comportement de notre objet.

Un objet est l'instance d'une classe, et une classe, est un type de données abstrait, caractérisé par des propriétés (ses attributs et ses méthodes) communes à des objets, qui permet de créer des objets possédant ces propriétés.

Objet = identité + état (attributs) + comportement (méthodes)

• L'identité : L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, ...)

• Les attributs : Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet

• Les méthodes (appelées parfois fonctions membres): Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures.

Remarque : Penser objet oblige à changer de mentalité !

-3. Les classes

-3.1. Notions de classe et d'instance de classe

On appelle classe la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet.

La classe est un type abstrait de données caractérisée par des propriétés (attributs et opérations) communes à ses objets, et un mécanisme permettant de créer des objets ayant ces propriétés.

Classe = instanciation + attributs (variables d'instances)+ opérations

• L'instanciation : L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. L'instanciation représente la relation entre un objet et sa classe d'appartenance qui a permis de le créer.

• Les attributs (appelés aussi variables d'instances): Ils ont un nom et soit un type de base (simple ou construit) soit une classe (l'attribut référence un objet de la même ou une autre classe).

• Les opérations (appelées parfois méthodes): Elles sont les opérations applicables à un objet de la classe. Elles peuvent modifier tout ou en partie l'état d'un objet et retourner des valeurs calculées à partir de cet état.

Tout système orienté objet est organisé autour des classes.

Une classe est la description formelle d'un ensemble d'objets ayant une sémantique et des caractéristiques communes.

3-2. Représentation graphique

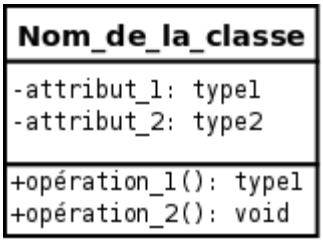

Figure 1 : Représentation UML d'une classe.

Une classe est un classeur. Elle est représentée par un rectangle divisé en trois à cinq compartiments. Le premier indique le nom de la classe, le deuxième ses attributs et le troisième ses opérations. Un compartiment des responsabilités peut être ajouté pour énumérer l'ensemble de tâches devant être assurées par la classe. Un compartiment des exceptions peut également être ajouté pour énumérer les situations exceptionnelles devant être gérées par la classe.

3-3. Encapsulation, visibilité, interface

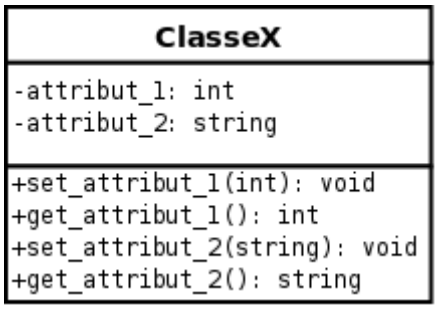

Figure 2 : Bonnes pratiques concernant la manipulation des attributs.

L'encapsulation permet de définir des niveaux de visibilité des éléments d'un conteneur.

Public ou + : tout élément qui peut voir le conteneur peut également voir l'élément indiqué.

Protected ou # : seul un élément situé dans le conteneur ou un de ses descendants peut voir l'élément indiqué.

Private ou - : seul un élément situé dans le conteneur peut voir l'élément.

Package ou ~ ou rien : seul un élément déclaré dans le même paquetage peut voir l'élément.

Dans une classe, le marqueur de visibilité se situe au niveau de chacune de ses caractéristiques (attributs, terminaisons d'association et opération). Il permet d'indiquer si une autre classe peut y accéder.

3-4. Nom d'une classe

Le nom de la classe doit évoquer le concept décrit par la classe. Il commence par une majuscule. Pour indiquer qu'une classe est abstraite, il faut ajouter le mot-clef abstract.

3-5. Les attributs

a. Attributs de la classe

Les attributs définissent des informations qu'une classe ou un objet doivent connaître. Ils représentent les données encapsulées dans les objets de cette classe. Chacune de ces informations est définie par un nom, un type de données, une visibilité et peut être initialisée. Le nom de l'attribut doit être unique dans la classe.

b. Attributs de classe

Un attribut de classe n'est donc pas une propriété d'une instance, mais une propriété de la classe et l'accès à cet attribut ne nécessite pas l'existence d'une instance.

Graphiquement, un attribut de classe est souligné.

c. Attributs dérivés

Les attributs dérivés peuvent être calculés à partir d'autres attributs et de formules de calcul. Lors de la conception, un attribut dérivé peut être utilisé comme marqueur jusqu'à ce que vous puissiez déterminer les règles à lui appliquer. Les attributs dérivés sont symbolisés par l'ajout d'un « / » devant leur nom.

3-6. Les méthodes

a. Méthode de la classe

Dans une classe, une opération (même nom et mêmes types de paramètres) doit être unique. Quand le nom d'une opération apparaît plusieurs fois avec des paramètres différents, on dit que l'opération est surchargée.

b. Méthode de classe

Comme pour les attributs de classe, il est possible de déclarer des méthodes de classe. Une méthode de classe ne peut manipuler que des attributs de classe et ses propres paramètres. Cette méthode n'a pas accès aux attributs de la classe (i.e. des instances de la classe). L'accès à une méthode de classe ne nécessite pas l'existence d'une instance de cette classe.
Graphiquement, une méthode de classe est soulignée.

c. Méthodes et classes abstraites

Une méthode est dite abstraite lorsqu'on connaît son entête, mais pas la manière dont elle peut être réalisée. Une classe est dite abstraite lorsqu'elle définit au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.
Une classe abstraite pure ne comporte que des méthodes abstraites. En programmation orientée objet, une telle classe est appelée une interface.
Pour indiquer qu'une classe est abstraite, il faut ajouter le mot-clef abstract derrière son nom.

3-7. Relations entre classes

Les diagrammes de classes expriment la structure statique du système. Ils décrivent l'ensemble des *classes* et leurs *associations.* Une classe décrit un ensemble d'*objets* (instances de la classe). Une association exprime *une connexion sémantique bidirectionnelle entre classes*.
Une association décrit un ensemble de *liens* (instances de l'association). Le *rôle* décrit une extrémité d'une association. Les *cardinalités* (ou multiplicité) indiquent le nombre d'instances d'une classe pour chaque instance de l'autre classe.

3-7-1. Association

Une association indique qu'il peut y avoir des liens entre des instances des classes associées.
Comment une association doit-elle être modélisée ? Plus précisément, quelle différence existe-t-il entre les deux diagrammes de la figure suivante ?

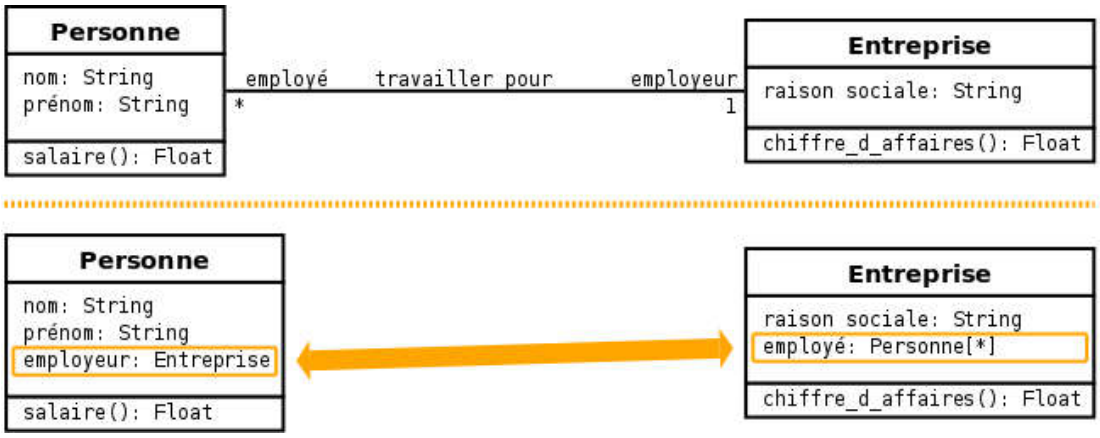

Figure 3 : Deux façons de modéliser une association.

Dans la première version, l'association apparaît clairement et constitue une entité distincte. Dans la seconde, l'association se manifeste par la présence de deux attributs dans chacune des classes en relation. La question de savoir s'il faut modéliser les associations en tant que telles a longtemps fait débat. UML a tranché pour la première version, car elle se situe plus à un niveau conceptuel (par opposition au niveau d'implémentation) et simplifie grandement la modélisation d'associations complexes (comme les associations plusieurs à plusieurs par exemple).

3-7-2. Terminaison d'association et ses Propriétés

La possession d'une terminaison d'association par le classeur situé à l'autre extrémité de l'association peut être spécifiée graphiquement par l'adjonction d'un petit cercle plein (point) entre la terminaison d'association et la classe. Il n'est pas indispensable de préciser la possession des terminaisons d'associations. Dans ce cas, la possession est ambiguë. Par contre, si l'on indique des possessions de terminaisons d'associations, toutes les terminaisons d'associations sont non ambiguës : la présence d'un point implique que la terminaison d'association appartient à la classe située à l'autre extrémité, l'absence du point implique que la terminaison d'association appartient à l'association.

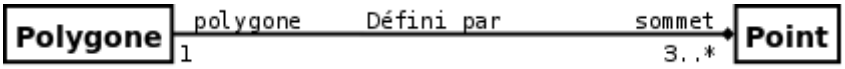

Figure 4 : Utilisation d'un petit cercle plein pour préciser le propriétaire d'une terminaison d'association.

Le diagramme précise que la terminaison d'association sommet appartient à la classe Polygone tandis que la terminaison d'association polygone appartient à l'association Défini par.

Une propriété est une caractéristique structurelle. Dans le cas d'une classe, les propriétés sont constituées par les attributs et les éventuelles terminaisons d'association que possède la classe. Dans le cas d'une association, les propriétés sont constituées par les terminaisons d'association que possède l'association.

Une propriété peut être paramétrée par les éléments suivants nom, visibilité, multiplicité, navigabilité.

3-7-3. Association binaire et n-aire

a. Association binaire

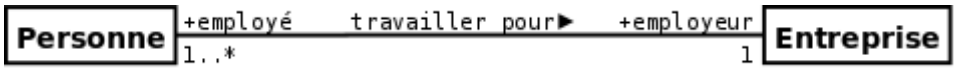

Figure 5 : Exemple d'association binaire.

Une association binaire est matérialisée par un trait plein entre les classes associées.

b. Association n-aire

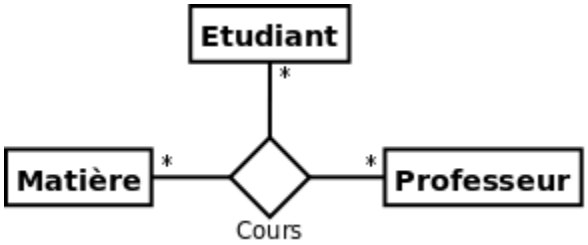

Figure 6 : Exemple d'association n-aire.

Une association n-aire lie plus de deux classes. On représente une association n-aire par un grand losange avec un chemin partant vers chaque classe participante. Le nom de l'association, le cas échéant, apparaît à proximité du losange.

Remarque : La multiplicité associée à une terminaison d'association, d'agrégation ou de composition déclare le nombre d'objets susceptibles d'occuper la position définie par la terminaison d'association.

Voici quelques exemples de multiplicité :

- exactement un : 1 ou 1..1 ;
- plusieurs : * ou 0..* ;
- au moins un : 1..* ;
- de un à six : 1..6.

3-7-4. Classe-association

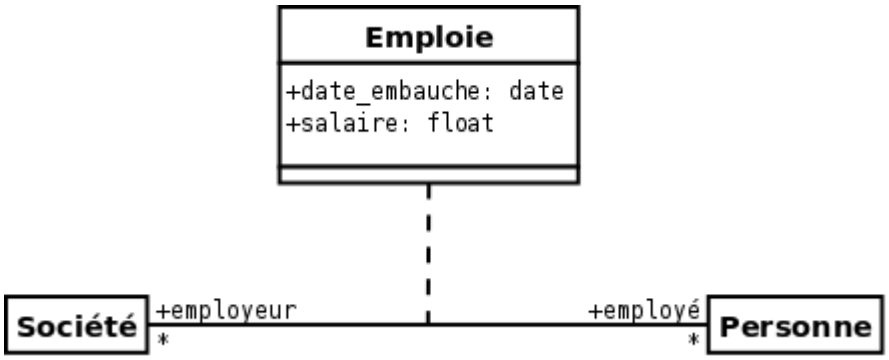

Figure 7 : Exemple de classe-association.

Parfois, une association doit posséder des propriétés. Par exemple, l'association Emploie entre une société et une personne possède comme propriétés le salaire et la date d'embauche. En effet, ces deux propriétés n'appartiennent ni à la société, qui peut employer plusieurs personnes, ni aux personnes, qui peuvent avoir plusieurs emplois. Il s'agit donc bien de propriétés de l'association Emploie. Les associations ne pouvant posséder de propriété, il faut introduire un nouveau concept pour modéliser cette situation : celui de classe-association.

Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

3-7-5. Agrégation et composition

a. Agrégation

Figure 8: Exemple de relation d'agrégation et de composition.

Lorsque l'on souhaite modéliser une relation tout/partie où une classe constitue un élément plus grand (tout) composé d'éléments plus petits (partie), il faut utiliser une agrégation.

Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Graphiquement, on ajoute un losange vide du côté de l'agrégat. Contrairement à une association simple, l'agrégation est transitive.

b. Composition

La composition, également appelée agrégation composite, décrit une contenance structurelle entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants. Une instance de la partie appartient toujours à au plus une instance de l'élément composite : la multiplicité du côté composite ne doit pas être supérieure à 1 (i.e. 1 ou 0..1). Graphiquement, on ajoute un losange plein (✦) du côté de l'agrégat.

Les notions d'agrégation et surtout de composition posent de nombreux problèmes en modélisation et sont souvent le sujet de querelles d'experts et sources de confusions.

3-7-6. Généralisation et Héritage

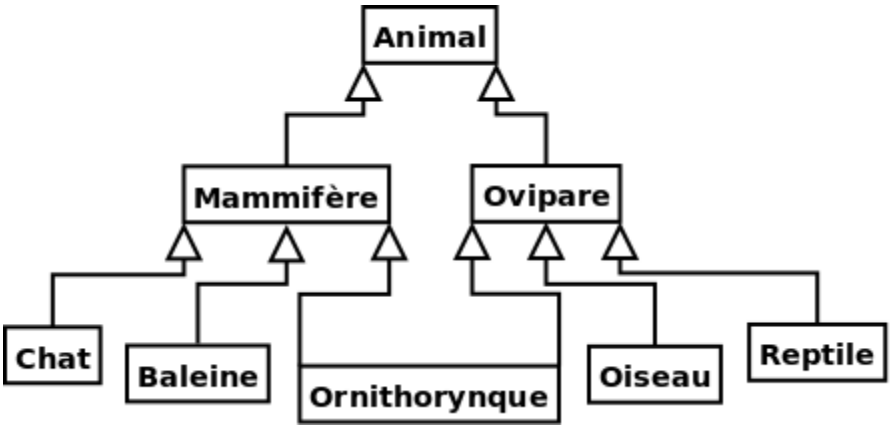

Figure 9 : Partie du règne animal décrit avec l'héritage multiple.

La généralisation décrit une relation entre une classe générale (classe de base ou classe parent) et une classe spécialisée (sous-classe). La classe spécialisée est intégralement cohérente avec la classe de base, mais comporte des informations supplémentaires (attributs, opérations, associations).
Le symbole utilisé pour la relation d'héritage ou de généralisation est une flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.
Les propriétés principales de l'héritage sont :

- la classe enfant possède toutes les caractéristiques de ses classes parents, mais elle ne peut accéder aux caractéristiques privées de cette dernière ;
- une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent. Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes ;
- toutes les associations de la classe parent s'appliquent aux classes dérivées ;
- une classe peut avoir plusieurs parents, on parle alors d'héritage multiple. Le langage C++ est un des langages objet permettant son implémentation effective, le langage Java ne le permet pas.

3-7-7. Dépendance

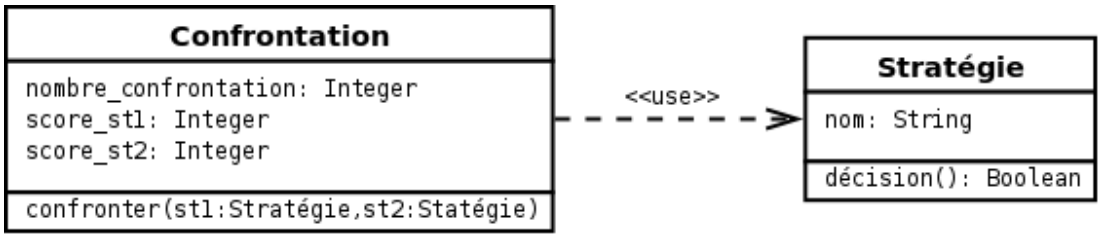

Figure 10 : Exemple de relation de dépendance.

Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle. Elle est représentée par un trait discontinu orienté. Elle indique que la modification de la cible peut impliquer une modification de la source. La dépendance est souvent stéréotypée pour mieux expliciter le lien sémantique entre les éléments du modèle.
On utilise souvent une dépendance quand une classe en utilise une autre comme argument dans la signature d'une opération.

3-7-8. Interfaces

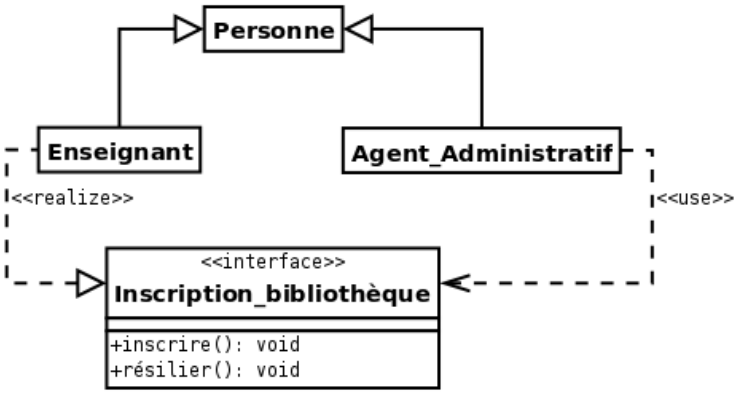

Figure 11 : Exemple de diagramme mettant en œuvre une interface.

Une interface est représentée comme une classe excepté l'absence du mot-clef abstract (car l'interface et toutes ses méthodes sont, par définition, abstraites) et l'ajout du stéréotype << interface >>.
Une interface doit être réalisée par au moins une classe et peut l'être par plusieurs. Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype « realize ». Une classe peut très bien réaliser plusieurs interfaces. Une classe (classe cliente de l'interface) peut dépendre d'une interface (interface requise). On représente cela par une relation de dépendance et le stéréotype « use ».

3-8. Diagramme d'objets (object diagram)

Un diagramme d'objets représente des objets et leurs liens pour donner une vue figée de l'état d'un système à un instant donné.
Le diagramme de classes modélise les règles et le diagramme d'objets modélise des faits.

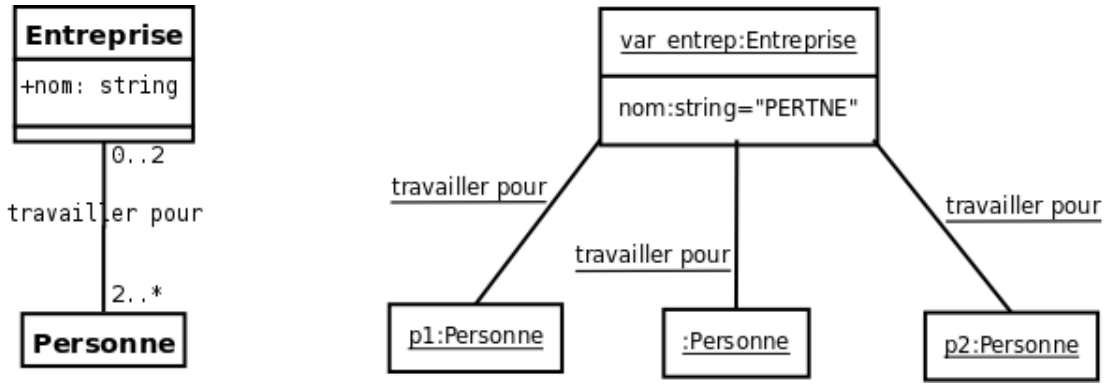

Figure 12 : Exemple de diagramme de classes et de diagramme d'objets associé.

Graphiquement, un objet se représente comme une classe. Cependant, le compartiment des opérations n'est pas utile. De plus, le nom de la classe dont l'objet est une instance est précédé d'un << : >> et est souligné. Pour différencier les objets d'une même classe, leur identifiant peut être ajouté devant le nom de la classe. Enfin les attributs reçoivent des valeurs. Quand certaines valeurs d'attribut d'un objet ne sont pas renseignées, on dit que l'objet est partiellement défini.

Dans un diagramme d'objets, les relations du diagramme de classes deviennent des liens. La relation de généralisation ne possède pas d'instance, elle n'est donc jamais représentée dans un diagramme d'objets. Graphiquement, un lien se représente comme une relation, mais, s'il y a un nom, il est souligné. Naturellement, on ne représente pas les multiplicités.

Exemple

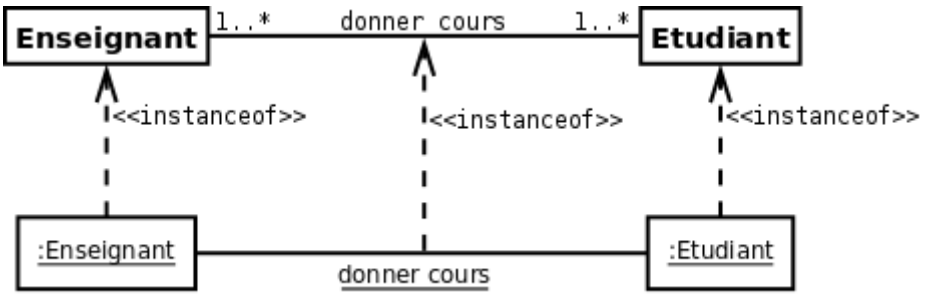

Figure 13 : Dépendance d'instanciation entre les classeurs et leurs instances.

La relation de dépendance d'instanciation (stéréotypée << instanceof >>) décrit la relation entre un classeur et ses instances. Elle relie, en particulier, les liens aux associations et les objets aux classes.

Référence :

1. G. Booch, J. Rumbaugh, I. Jacobson, « The Unified Modeling Language (UML) user guide", Addison-Wesley, 1999.
2. Benoit charroux, aomar Osmani, Yann Therry-Mieg,"UML2 synthèse et exercices",Pearson édition france,ISBN2-7440-7124-2,2005.
3. G. Booch et al., « Object-Oriented Analysis and Design, with applications », Addison- Wesley, 2007.
4. Cours UML 2.0 de Laurent Audibert , site http://www.developpez.com