Chapitre 3

Diagramme de classes (*Class Diagram*)

3.1 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 ensemble pour réaliser les cas d'utilisation. Il est important de noter qu'un même objet peut très bien intervenir dans la réalisation de plusieurs cas d'utilisation. Les cas d'utilisation ne réalisent donc pas une partition[1] des classes du diagramme de classes. Un diagramme de classes n'est donc pas adapté (sauf cas particulier) pour détailler, décomposer, ou illustrer la réalisation d'un cas d'utilisation particulier.

Il s'agit d'une vue statique car on ne tient pas compte du facteur temporel dans le comportement du système. Le diagramme de classes modélise les concepts du domaine d'application ainsi que les concepts internes créés de toutes pièces dans le cadre de l'implémentation d'une application. Chaque langage de Programmation Orienté Objets donne un moyen spécifique d'implémenter le paradigme objet (pointeurs ou pas, héritage multiple ou pas, etc.), mais le diagramme de classes permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier.

Les principaux éléments de cette vue statique sont les classes et leurs relations : association, généralisation et plusieurs types de dépendances, telles que la réalisation et l'utilisation.

3.2 Les classes

3.2.1 Notions de classe et d'instance de classe

Une *instance* est une concrétisation d'un concept abstrait. Par exemple :

- la Ferrari *Enzo* qui se trouve dans votre garage est une instance du concept abstrait *Automobile* ;
- l'amitié qui lie Jean et Marie est une instance du concept abstrait *Amitié* ; Une classe est un concept abstrait représentant des éléments variés comme :
- des éléments concrets (ex : des avions),
- des éléments abstraits (ex : des commandes),
- des structures informatiques (ex : des tables de hachage),
- des éléments comportementaux (ex : des tâches),
- des composants d'une application (ex : les boutons des boîtes de dialogue), etc.

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 propriétés communes.

Un objet est une instance d'une classe. C'est une entité discrète dotée d'une identité, d'un état et d'un comportement que l'on peut invoquer. Les objets sont des éléments individuels d'un système en cours d'exécution.

[1] Une partition d'un ensemble est un ensemble de parties non vides de cet ensemble, deux à deux disjointes et dont la réunion est égale à l'ensemble.

Par exemple, si l'on considère que *Homme* (au sens être humain) est un concept abstrait, on peut dire que la personne Marie-Cécile est une instance de Homme. Si *Homme* était une classe, Marie-Cécile en serait une instance : un objet.

3.2.2 Notions de propriétés

Une classe définit un jeu d'objets dotés de propriétés. Les propriétés d'un objet permettent de spécifier son *état* et son *comportement*. Les propriétés d'un objet étaient soit des attributs, soit des opérations. Ce n'est pas exact dans un diagramme de classe car *les terminaisons d'associations font également partie des propriétés d'un objet au même titre que les attributs et les opérations*.

État d'un objet : Ce sont les attributs et les *terminaisons d'associations* qui décrivent l'état d'un objet. On utilise les attributs pour des valeurs de données pures, dépourvues d'identité, telles que les nombres et les chaînes de caractères. On utilise les associations pour connecter les classes du diagramme de classe. Dans ce cas, la terminaison de l'association (du côté de la classe cible) est une propriété de la classe de base.
Les propriétés décrites par les attributs prennent des valeurs lorsque la classe est instanciée. L'instance d'une association est appelée un lien.

Comportement d'un objet : Les opérations décrivent les éléments individuels d'un comportement que l'on peut invoquer. Ce sont des fonctions qui peuvent prendre des valeurs en entrée et modifier les attributs ou produire des résultats.
Une opération est la spécification (*i.e.* déclaration) d'une méthode. L'implémentation (*i.e.* définition) d'une méthode est également appelée méthode. Il y a donc une ambiguïté sur le terme *méthode*.

Les attributs, les terminaisons d'association et les méthodes constituent donc les propriétés d'une classe (et de ses instances).

3.2.3 Représentation graphique

Une classe est un classeur [2]. Elle est représentée par un rectangle divisé en trois à cinq compartiments (figure 3.1).

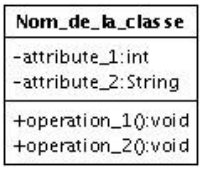

Figure 3.1 : Représentation UML d'une classe

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 mais pour lesquelles on ne dispose pas encore assez d'informations. Un compartiment des exceptions peut également être ajouté pour énumérer les situations exceptionnelles devant être gérées par la classe.

[2] De manière générale, toute boîte non stéréotypée dans un diagramme de classes est implicitement une classe. Ainsi, le stéréotype *class* est le stéréotype par défaut.

3.2.4 Encapsulation, visibilité, interface

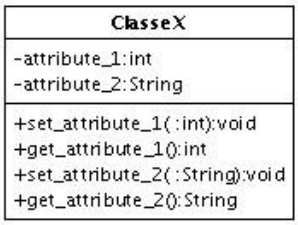

Figure 3.2 : Bonnes pratiques concernant la manipulation des attributs.

L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. Ces services accessibles (offerts) aux utilisateurs de l'objet définissent ce que l'on appelle l'interface de l'objet (sa vue externe). L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.

L'encapsulation permet de définir des niveaux de visibilité des éléments d'un conteneur. La visibilité déclare la possibilité pour un élément de modélisation de référencer un élément qui se trouve dans un espace de noms différents de celui de l'élément qui établit la référence. Elle fait partie de la relation entre un élément et le conteneur qui l'héberge, ce dernier pouvant être un paquetage, une classe ou un autre espace de noms. Il existe quatre visibilités prédéfinies.

***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.

Par ailleur, UML 2.0 donne la possibilité d'utiliser n'importe quel langage de programmation pour la spécification de la visibilité.

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

Dans un paquetage, le marqueur de visibilité se situe sur des éléments contenus directement dans le paquetage, comme les classes, les paquetages imbriqués, etc. Il indique si un autre paquetage susceptible d'accéder au premier paquetage peut voir les éléments.

Dans la pratique, lorsque des attributs doivent être accessibles de l'extérieur, il est préférable que cet accès ne soit pas direct mais se fasse par l'intermédiaire de méthodes.

3.2.5 Nom d'une classe

Le nom de la classe doit évoquer le concept décrit par la classe. Il commence par une majuscule. On peut ajouter des informations subsidiaires comme le nom de l'auteur de la modélisation, la date, etc.
Pour indiquer qu'une classe est abstraite, il faut ajouter le mot-clef *abstract*.

La syntaxe de base de la déclaration d'un nom d'une classe est la suivante :

[<Nom_du_paquetage_1>::...::<Nom_du_paquetage_N>]
<Nom_de_la_classe> [{ [abstract], [<auteur>], [<date>], ... }]

3.2.6 Les attributs

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 initialisé. Le nom de l'attribut doit être unique dans la classe. La syntaxe de la déclaration d'un attribut est la suivante :

<visibilité> [/] <nom_attribut>:
<Type> ['[' <multiplicité> ']' ['{' <contrainte> '}']] [= <valeur_par_défaut>] Le type de l'attribut (<Type>) peut être un nom de classe, un nom d'interface ou un type de donné prédéfini. La multiplicité (<multiplicité>) d'un attribut précise le nombre de valeurs que l'attribut peut contenir. Lorsqu'un multiplicité supérieure à 1 est précisée, il est possible d'ajouter une contrainte (<contrainte>) pour préciser si les valeurs sont ordonnées (*{ordered}*) ou pas (*{list}*).

Attributs de classe

Par défaut, chaque instance d'une classe possède sa propre copie des attributs de la classe. Les valeurs des attributs peuvent donc différer d'un objet à un autre. Cependant, il est parfois nécessaire de définir un *attribut de classe* (*static* en Java ou en C++) qui garde une valeur unique et partagée par toutes les instances de la classe. Les instances ont accès à cet attribut mais n'en possèdent pas une copie. 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.

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.

3.2.7 Les méthodes

Méthode de la classe

Dans une classe, une opération (même nom et même 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. En revanche, il est impossible que deux opérations ne se distinguent que par leur valeur retournée.

La déclaration d'un opération contient les types des paramètres et le type de la valeur de retour, sa syntaxe est la suivante :

<visibilité> <nom_méthode> ([<paramètre> [, <paramètre> [, <paramètre> ...]]]) :
[<valeur_renvoyé>] [{ <propriétés> }]

La syntaxe de définition d'un paramètre (<paramètre>) est la suivante :

[<direction>] <nom_paramètre>:<Type> ['['<multiplicité>']'] [=<valeur_par_défaut>] La direction peut

prendre l'une des valeurs suivante :

in : Paramètre d'entrée passé par valeur. Les modifications du paramètre ne sont pas disponibles pour l'appelant. C'est le comportement par défaut.

out : Paramètre de sortie uniquement. Il n'y a pas de valeur d'entrée et la valeur finale est disponible pour l'appelant.

inout : Paramètre d'entrée/sortie. La valeur finale est disponible pour l'appelant.

Le type du paramètre (<Type>) peut être un nom de classe, un nom d'interface ou un type de donné prédéfini.

Les propriétés (<propriétés>) correspondent à des contraintes ou à des informations complémentaires comme les exceptions, les préconditions, les postconditions ou encore l'indication qu'une méthode est abstraite (mot-clef *abstract*), etc.

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.

Méthodes et classes abstraites

Une méthode est dite abstraite lorsqu'on connaît son entête mais pas la manière dont elle peut être réalisée (*i.e.* on connaît sa déclaration mais pas sa définition).

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.

On ne peut instancier une classe abstraite : elle est vouée à se spécialiser. Une classe abstraite peut très bien contenir des méthodes concrètes.

Une classe abstraite pure ne comporte que des méthodes abstraites. En programmation orientée objet, une telle classe est appelée une interface.

3.2.8 Classe active

Une classe est passive par défaut, elle sauvegarde les données et offre des services aux autres. Une classe active initie et contrôle le flux d'activités.

Graphiquement, une classe active est représentée comme une classe standard dont les lignes verticales du cadre, sur les côtés droit et gauche, sont doublées.

3.3 Relations entre classes

3.3.1 Généralisation et Héritage

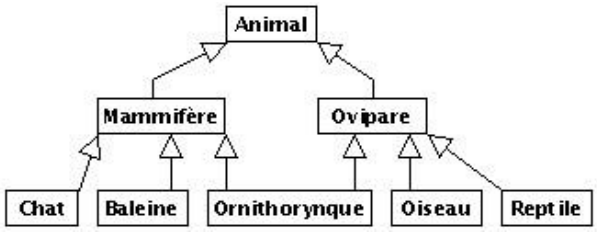

Figure 3.3 : Partie du règne animal décrit avec l'héritage multiple.

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). Un objet de la classe spécialisée peut être utilisé partout où un objet de la classe de base est autorisé.

Dans le langage UML, ainsi que dans la plupart des langages objet, cette relation de généralisation se traduit par le concept d'héritage. On parle également de relation d'héritage. Ainsi, l'héritage permet la classification des objets.

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 propriétés des ses classes parents, mais elle ne peut accéder aux propriétés privées de celle-ci.
- 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 instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue. Par exemple, en se basant sur le diagramme de la figure ci-dessus, toute opération acceptant un objet d'une classe *Animal* doit accepter un objet de la classe *Chat*.
- Une classe peut avoir plusieurs parents, on parle alors d'héritage multiple (cf. la classe *Ornithorynque* de la figure 3.3). Le langage C++ est un des langages objet permettant son implémentation effective, le langage java ne le permet pas.

En UML, la relation d'héritage n'est pas propre aux classes. Elle s'applique à d'autre éléments du langage comme les paquetages, les acteurs ou les cas d'utilisation.

3.3.2 Association

Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions structurelles entre leurs instances.

Terminaison d'association vs. attribut

Un attribut est une association dégénérée dans laquelle une terminaison d'association[3] est détenue par un classeur (généralement une classe). Le classeur détenant cette terminaison d'association devrait théoriquement se trouver à l'autre terminaison, non modélisée, de l'association. Un attribut n'est donc rien d'autre qu'une terminaison d'un cas particulier d'association.

Les terminaisons d'associations et les attributs sont donc deux éléments conceptuellement très proches que l'on regroupe sous le terme de *propriété structurelle*.

Une propriété structurelle peut être paramètrée par les éléments suivant :

nom : Comme un attribut, une terminaison d'association peut être nommée. Le nom est situé à proximité de la terminaison, mais contrairement à un attribut, ce nom est facultatif. Le nom d'une terminaison d'association est appelée *nom du rôle*. Une association peut donc posséder autant de noms de rôle que de terminaisons (deux pour une association binaire et *n* pour une association n-aire).

visibilité : Comme un attribut, une terminaison d'association possède une visibilité. La visibilité est mentionnée à proximité de la terminaison, et plus précisément, le cas échéant, devant le nom de la terminaison.

multiplicité : Comme un attribut, une terminaison d'association peut posséder une multiplicité. Elle est mentionnée à proximité de la terminaison. Il n'est pas impératif de la préciser, mais, contrairement à un attribut dont la multiplicité par défaut est 1, la multiplicité par défaut d'une terminaison d'association est *non spécifiée*. L'interprétation de la multiplicité pour une terminaison d'association est moins évidente que pour un attributs.

navigabilité : Pour un attribut, la navigabilité est implicite, navigable, et toujours depuis la classe vers l'attribut. Pour une terminaison d'association, la navigabilité peut être précisée.

Association binaire

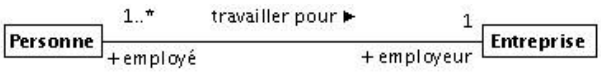

Figure 3.4 : Exemple d'association binaire.

Une association binaire est matérialisée par un trait plein entre les classes associées. Elle peut être ornée d'un nom, avec éventuellement une précision du sens de lecture (◭ ou ◮).

Quand les deux extrémités de l'association pointent vers la même classe, l'association est dite *réflexive*.

Association n-aire

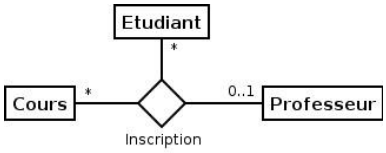

Figure 3.5 : Exemple d'association n-aire.

[3] Une terminaison d'associations est une extrêmité de l'association. Une association binaire en possède deux, une association n-aire en possède *n*

Une association n-aire lie plus de deux classes. La ligne pointillé d'une classe-association peut être reliée au losange par une ligne discontinue pour représenter une association n-aire dotée d'attributs, d'opérations ou d'associations.

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.

3.3.3 Multiplicité ou cardinalité

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

Dans une association binaire, la multiplicité sur la terminaison cible contraint le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association).

Dans une association n-aire, la multiplicité apparaissant sur le lien de chaque classe s'applique sur une instance de chacune des classes, à l'exclusion de la classe-association et de la classe considérée. Par exemple, si on prend une association ternaire entre les classes (A, B, C), la multiplicité de la terminaison C indique le nombre d'objets C qui peuvent apparaître dans l'association avec une paire particulière d'objets A et B.

Remarque

Il faut noter que, pour les habitués du modèle entité/relation, les multiplicités sont en UML « à l'envers » (par référence à Merise) pour les associations binaires et « à l'endroit » pour les n-aires avec $n > 2$.

3.3.4 Navigabilité

La navigabilité indique s'il est possible de traverser une association. On représente graphiquement la navigabilité par une flèche du côté de la terminaison navigable et on empêche la navigabilité par une croix du côté de la terminaison non navigable. Par défaut, une association est navigable dans les deux sens.

Par exemple, sur la figure 3.6, la terminaison du côté de la classe *Commande* n'est pas navigable : cela signifie que les instances de la classe *Produit* ne stockent pas de liste d'objets du type *Commande*.

Figure 3.6 : Navigabilité.

Inversement, la terminaison du côté de la classe *Produit* est navigable : chaque objet commande contient une liste de produits.

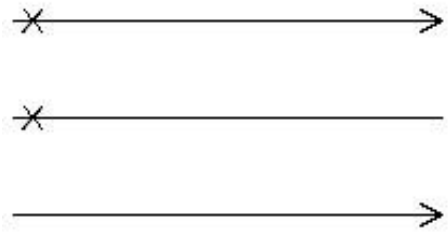

Figure 3.7 : Implicitement, ces trois notations ont la même sémantique.

Lorsque l'on représente la navigabilité uniquement sur l'une des extrémités d'une association, il faut remarquer que, implicitement, les trois associations représentées sur la figure 3.7 ont la même signification : l'association ne peut être traversée que dans un sens.

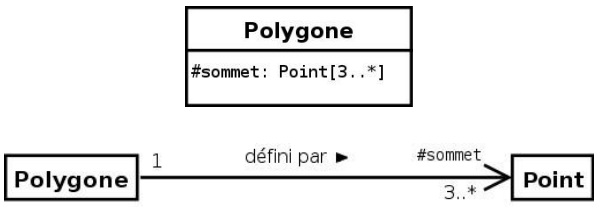

Figure 3.8 : Deux modélisations équivalentes.

Dans la section 3.3.2, nous avons dit que :

> « Un attribut est une association dégénérée dans laquelle une terminaison d'association est détenue par un classeur (généralement une classe). Le classeur détenant cette terminaison d'association devrait théoriquement se trouver à l'autre terminaison, non modélisée, de l'association. Un attribut n'est donc rien d'autre qu'une terminaison d'un cas particulier d'association. »

La figure 3.8 illustre parfaitement cette situation. Attention toutefois, si vous avez une classe *Point* dans votre diagramme de classe, il est extrêmement maladroit de représenter des classes (comme la classe *Polygone*) avec un ou plusieurs attributs de type *Point*. Il faut, dans ce cas, matérialiser cette propriété de la classe en question par une ou plusieurs associations avec la classe *Point*.

3.3.5 Qualification

Généralement, une classe peut être décomposée en sous-classes ou posséder plusieurs propriétés. Une telle classe rassemble un ensemble d'éléments (d'objets). Quand une classe est liée à une autre classe par une association, il est parfois préférable de restreindre la portée de l'association à quelques éléments ciblés (comme un ou plusieurs attributs) de la classe. Ces éléments ciblés sont appelés un *qualificatif*. Un qualificatif permet donc de sélectionner un ou des objets dans le jeu des objets d'un objet (appelé

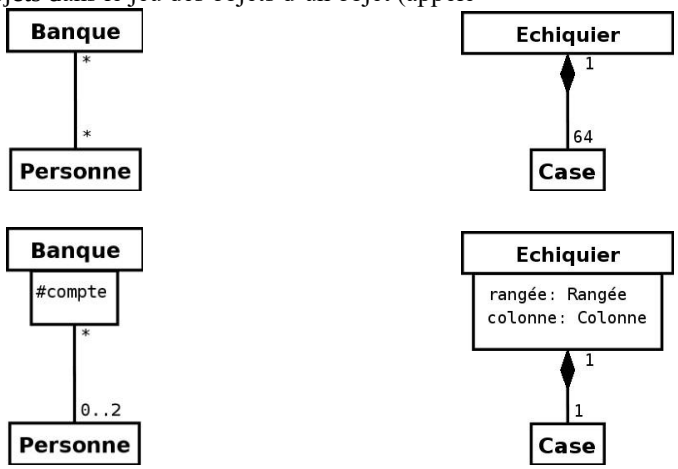

Figure. 3.9 – En haut, un diagramme représentant l'association entre une banque et ses clients (à gauche), et un diagramme représentant l'association entre un échiquier et les cases qui le composent (à droite). En bas, les diagrammes équivalents utilisant des associations qualifiées.

objet qualifié) relié par une association à un autre objet. L'objet sélectionné par la valeur du qualificatif est appelé *objet cible*. L'association est appelé *association qualifiée*. Un qualificatif agit toujours sur une association dont la multiplicité est *plusieurs* (avant que l'association ne soit qualifiée) du côté *cible*.

Un objet qualifié et une valeur de qualificatif génèrent un objet cible lié unique. En considérant un objet qualifié, chaque valeur de qualificatif désigne un objet cible unique.

Par exemple, le diagramme de gauche de la figure 3.9 nous dit que :

- Un compte dans une banque appartient à au plus deux personnes. Autrement dit, une instance ducouple {*Banque , compte*} est en association avec zéro à deux instances de la classe *Personne*.
- Mais une personne peut posséder plusieurs comptes dans plusieurs banques. C'est-à-dire qu'uneinstance de la classe *Personne* peut être associée à plusieurs (zéro compris) instances du couple {*Banque , compte*}.

Le diagramme de droite de cette même figure nous dit que :

- Une instance du triplet {*Echiquier, rangée, colonne*} est en association avec une instance unique de la classe *Case*.
- Inversement, une instance de la classe *Case* est en association avec une instance unique du triplet {*Echiquier, rangée, colonne*}.

3.3.6 Classe-association

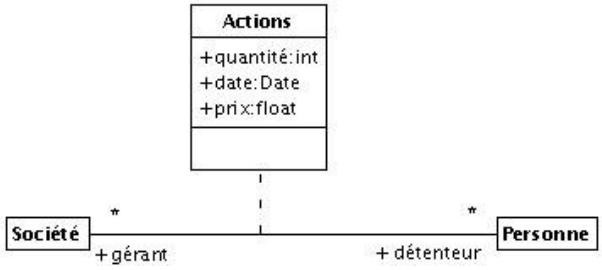

Figure. 3.10 – Exemple de classe-association.

Une classe-association possède les propriétés des associations et des classes : elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations.

Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente (figure 3.10).

Par exemple, dans la figure 3.10, la détention d'actions est modélisée en tant qu'association entre les classes *Personne* et *Société*. Les attributs de la classe-association *Action* permettent de préciser les informations relatives à chaque détention d'actions (nombre d'actions, prix et date d'achat).

3.3.7 Agrégation

Figure. 3.11 – Exemple de relation d'agrégation et de composition.

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 (cf. figure 3.11). Contrairement à une association simple, l'agrégation est transitive.

3.3.8 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. Graphiquement, on ajoute un losange plein () du côté de l'agrégat (cf. figure 3.11).

3.3.9 Dépendance

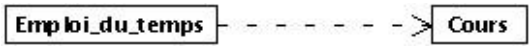

Figure. 3.12 – Exemple de relation de dépendance.

Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre les éléments du modèle. Elle est représentée par un trait discontinu orienté (cf. figure 3.12). Elle indique que la modification de la cible implique une modification de la source. La dépendance est souvent stéréotypée pour mieux expliciter le lien sémantique entre les éléments du modèle (cf. figure 3.15).

3.4 Interfaces

Les classes permettent de définir en même temps un objet et son interface. Le classeur, que nous décrivons dans cette section, ne permet de définir que des éléments d'interface. Il peut s'agir de l'interface complète d'un objet, ou simplement d'une partie d'interface qui sera commune à plusieurs objets.

Le rôle de ce classeur, stéréotypé *« interface »*, est de regrouper un ensemble de propriétés et d'opérations assurant un service cohérent.

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 »* (cf. figure 3.13).

Une interface doit être réalisée par au moins une classe. Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype *« realize »*. 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 »*.

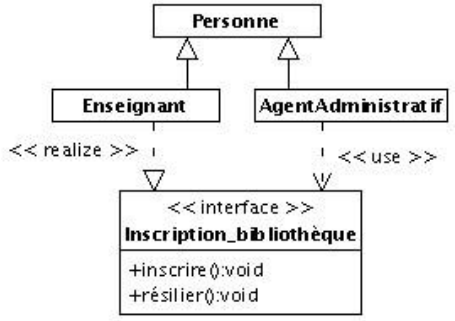

Figure. 3.13 – Exemple de diagramme mettant en œuvre une interface

3.5 Élaboration d'un diagramme de classes

Il y a au moins trois points de vue qui guident la modélisation (Steve Cook et John Daniels) :

- Le point de vue spécification met l'accent sur les interfaces des classes plutôt que sur leurs contenus.
- Le point de vue conceptuel capture les concepts du domaine et les liens qui les lient. Il s'intéresse peu ou prou à la manière éventuelle d'implémenter ces concepts et relations et aux langages d'implémentation.
- Le point de vue implémentation, le plus courant, détaille le contenu et l'implémentation de chaque classe.

En fonction du point de vue adopté, vous obtiendrez des modèles différents.

Une démarche couramment utilisée pour bâtir un diagramme de classes consiste à :

Trouver les classes du domaine étudié. Cette étape empirique se fait généralement en collaboration avec un expert du domaine. Les classes correspondent généralement à des concepts ou des substantifs du domaine.

Trouver les associations entre classes. Les associations correspondent souvent à des verbes, ou des constructions verbales, mettant en relation plusieurs classes, comme « est composé de », « pilote », « travaille pour ». *Attention, méfiez vous de certains attributs qui sont en réalité des relations entre classes.*

Trouver les attributs des classes. Les attributs correspondent souvent à des substantifs, ou des groupes nominaux, tels que « la masse d'une voiture » ou « le montant d'une transaction ». Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs. Vous pouvez ajouter des attributs à toutes les étapes du cycle de vie d'un projet (implémentation comprise). N'espérez pas trouver tous les attributs dès la construction du diagramme de classes.

Organiser et simplifier le modèle en éliminant les classes redondantes et en utilisant l'héritage.

Vérifier les chemins d'accès aux classes.

Itérer et raffiner le modèle. Un modèle est rarement correct dès sa première construction. La modélisation objet est un processus non pas linéaire mais itératif.

3.6 Diagramme d'objets (*object diagram*)

3.6.1 Présentation

Un diagramme d'objets représente des objets (*i.e.* instances de classes) et leurs liens (*i.e.* instances de relations) pour donner une vue de l'état du système à un instant donné. Un diagramme d'objets permet, selon les situations, d'illustrer le modèle de classes (en montrant un exemple qui explique le modèle), de préciser certains aspects du système (en mettant en évidence des détails imperceptibles dans le diagramme de classes), d'exprimer une exception (en modélisant des cas particuliers, des connaissances non généralisables . . .), ou de prendre une image (*snapshot*) d'un système à un moment donné. Le diagramme de classes modélise les règles et le diagramme d'objets modélise des faits.

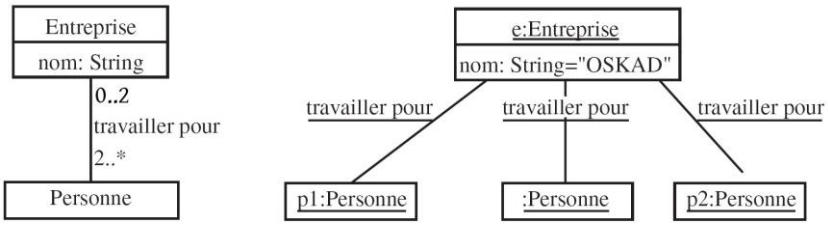

Figure. 3.14 – Exemple de diagramme de classes et de diagramme d'objets associé.

Par exemple, le diagramme de classes de la figure 3.14 montre qu'une entreprise emploie au moins deux personnes et qu'une personne travaille dans au plus deux entreprises. Le diagramme d'objets modélise lui une entreprise particulière (*OSKAD*) qui emploie trois personnes.

Un diagramme d'objets ne montre pas l'évolution du système dans le temps. Pour représenter une interaction, il faut utiliser un diagramme de communication ou de séquence.

3.6.2 Représentation

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 diagrammes d'objets, les relations du diagramme de classes deviennent des liens. 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.

3.6.3 Relation de dépendance d'instanciation

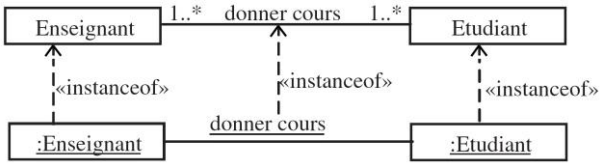

Figure. 3.15 – Dépendance d'instanciation entre les classeurs et leurs instances.

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.