Diagrammes de classes

Objectifs

- Présenter les concepts UML relatifs à la vue structurelle (diagramme de classes)
- Présenter la notation graphique du diagramme de classes UML
- Expliquer la sémantique des classes UML (compatible avec la sémantique des langages de programmation orientés objet)

Vue structurelle du modèle UML

La vue structurelle du modèle UML est la vue la plus utilisée pour spécifier une application. L'objectif de cette vue est de modéliser la structure des différentes classes d'une application orientée objet ainsi que leurs relations.

Paradigme orienté objet

Conçu à l'origine, au cours des années 1990, pour faciliter la construction d'applications orientées objet (OO), le langage UML a ensuite fortement évolué jusqu'à sa version 2.1 actuelle. Néanmoins, UML reste toujours très OO. Les concepts qu'il propose pour modéliser la vue structurelle sont donc les concepts de classe et d'objet.

UML définit cependant sa propre sémantique OO, laquelle ressemble à la sémantique des langages de programmation objet Java ou C++. Il est donc important de considérer UML comme un langage à part entière, et non comme une couche graphique permettant de dessiner des applications Java ou C++.

L'objectif de l'ensemble de ce cours étant de présenter UML pour le développeur, un minimum de connaissances du paradigme orienté objet est requis. Les concepts élémentaires suivants du paradigme objet seront donc supposés connus :

- objet
- classe
- instance
- héritage
- polymorphisme
- encapsulation

Concepts élémentaires

Les concepts élémentaires que nous présentons dans cette section sont les plus employés pour la réalisation de la vue structurelle d'un modèle UML.

Classe

Sémantique

En UML, une classe définit la structure commune d'un ensemble d'objets et permet la construction d'objets instances de cette classe. Une classe est identifiée par son nom.

Graphique

Une classe se représente à l'aide d'un rectangle, qui contient le nom de la classe. La figure 2.1 illustre la classe nommée `Personne`.

Figure 2.1
Représentation graphique d'une classe

Interface

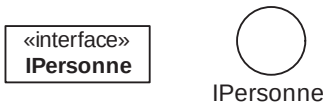
Sémantique

En UML, une interface définit un contrat que doivent respecter les classes qui réalisent l'interface. Une interface est identifiée par son nom. Les objets instances des classes qui réalisent des interfaces sont aussi des instances des interfaces. Une classe peut réaliser plusieurs interfaces, et une interface peut être réalisée par plusieurs classes.

Graphique

Une interface se représente de deux façons : soit à l'aide d'un rectangle contenant le nom de l'interface, au-dessus duquel se trouve la chaîne de caractères `«interface»`, soit à l'aide d'un cercle, au-dessous duquel se trouve le nom de l'interface *(voir figure 2.2)*.

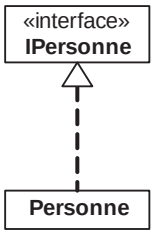
Figure 2.2
Représentations graphiques d'une interface



La relation de réalisation entre une classe et une interface est représentée par une flèche pointillée à la tête en forme de triangle blanc.

La figure 2.3 représente la classe `Personne` qui réalise l'interface `IPersonne`.

Figure 2.3
Représentation graphique d'une relation de réalisation



Propriété (anciennement appelée attribut) d'une classe ou d'une interface

Sémantique

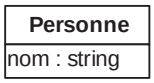
Les classes et les interfaces peuvent posséder plusieurs propriétés. Une propriété a un nom et un type. Le type peut être soit une classe UML, soit un type de base (`integer`, `string`, `boolean`, `char`, `real`). Un objet instance de la classe ou de l'interface doit porter les valeurs des propriétés de sa classe.

Graphique

Les propriétés d'une classe ou d'une interface se représentent dans le rectangle représentant la classe ou l'interface. Chaque propriété est représentée par son nom et son type.

La figure 2.4 présente la classe `Personne`, avec sa propriété `nom` de type `string`.

Figure 2.4
Représentation graphique d'une propriété d'une classe



Opération d'une classe ou d'une interface

Sémantique

Les classes et les interfaces peuvent posséder plusieurs opérations. Une opération a un nom et des paramètres et peut lever des exceptions. Les paramètres sont typés et ont un sens (`in`, `out`, `inout`, `return`).

Un objet instance de la classe ou de l'interface est responsable de la réalisation des opérations définies dans la classe ou dans l'interface.

Si le sens d'un paramètre de l'opération est `in`, l'objet appelant l'opération doit fournir la valeur du paramètre. Si le sens d'un paramètre de l'opération est `out`, l'objet responsable de l'opération doit fournir la valeur du paramètre. Si le sens d'un paramètre de l'opération est `inout`, l'objet appelant l'opération doit fournir la valeur du paramètre, mais celle-ci peut être modifiée par l'objet responsable de l'opération.

Un seul paramètre peut avoir `return` comme sens, et il n'est alors pas nécessaire de préciser le nom de ce paramètre. Si une opération possède un paramètre dont le sens est `return`, cela signifie que l'objet responsable de l'opération rend cette valeur comme résultat de l'opération. L'apport de la direction `return` par rapport à la direction `out` est de faciliter la combinaison de fonction.

Pour finir, les exceptions d'une opération sont typées.

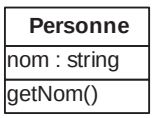
Il est important de souligner que les opérations UML ne définissent pas le comportement qui sera réalisé lors de l'invocation de l'opération. Nous verrons dans la suite du cours comment ce comportement est intégré dans le modèle.

Graphique

Les opérations d'une classe ou d'une interface se représentent dans le rectangle représentant la classe ou l'interface. Chaque opération est représentée par son nom et ses paramètres. Il est aussi possible de masquer les paramètres de l'opération.

La figure 2.5 présente la classe `Personne` avec son opération `getNom`.

Figure 2.5
Représentation graphique d'une opération d'une classe



Héritage entre classes

Sémantique

En UML, une classe peut hériter d'autres classes. L'héritage entre classes UML doit être considéré comme une inclusion entre les ensembles des objets instances des classes. Les objets instances des sous-classes sont des objets instances des superclasses. En d'autres termes, si une classe A hérite d'une classe B, l'ensemble des objets instances de A est inclus dans l'ensemble des objets instances de B.

Ce faisant, tout objet instance de A doit posséder les valeurs des propriétés définies dans A et dans B et doit être responsable des opérations définies dans A et dans B.

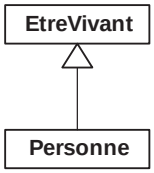
Nous verrons dans la suite de ce cours que la relation d'héritage entre deux classes appartenant à des packages différents dépend de certaines règles.

Graphique

La relation d'héritage entre deux classes est représentée par une flèche à la tête en forme de triangle blanc.

La figure 2.6 représente la classe `Personne`, qui hérite de la classe `EtreVivant`.

Figure 2.6
Représentation graphique de la relation d'héritage entre classes



Package

Sémantique

Un package permet de regrouper des classes, des interfaces et des packages. Classes, interfaces et packages ne peuvent avoir qu'un seul package dans lequel ils sont regroupés. La possibilité d'établir un lien entre des classes et des interfaces dépend du lien qui existe entre les packages qui les contiennent.

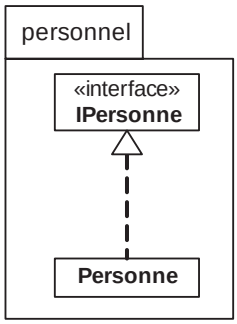
Nous détaillons ces concepts avancés à la section suivante.

Graphique

Les packages se représentent à l'aide d'un rectangle possédant un onglet dans lequel est inscrit le nom du package. Les éléments contenus se représentent dans le rectangle. La taille du rectangle s'adapte à la taille de son contenu.

La figure 2.7 représente le package nommé `personnel`, qui contient la classe `Personne`.

Figure 2.7
Représentation graphique d'un package



Import de package

Sémantique

Afin que les classes d'un package puissent hériter des classes d'un autre package ou y être associées *(voir section suivante)*, il faut préciser une relation d'import entre ces deux packages. La relation d'import est monodirectionnelle, c'est-à-dire qu'elle comporte un package source et un package cible. Les classes du package source peuvent avoir accès aux classes du package cible.

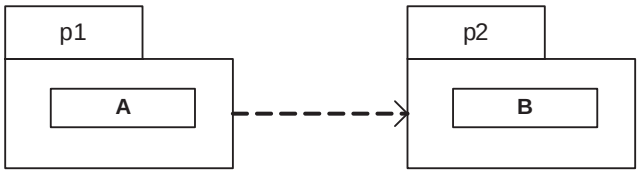
Nous revenons sur cette sémantique au chapitre 4 de ce cours, mais nous pouvons déjà mentionner que nous considérons comme interdits les cycles de relations d'import entre plusieurs packages.

Graphique

La relation d'import entre deux packages s'exprime à l'aide d'une flèche en pointillé. La chaîne de caractères `access element` inscrite au-dessus de cette flèche peut être optionnellement positionnée.

La figure 2.8 présente la relation d'import entre deux packages P1 et P2 contenant respectivement les classes A et B. Nous considérons ici que la classe A a besoin d'accéder à la classe B.

Figure 2.8
Représentation graphique de la relation d'import entre deux packages



Note

Sémantique

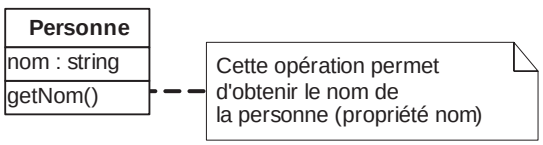
Une note UML est un paragraphe de texte qui peut être attaché à n'importe quel élément du modèle UML (package, classe, propriété, opération, association). Le texte contenu dans la note permet de commenter l'élément ciblé par la note.

Graphique

Les notes se représentent à l'aide d'un rectangle contenant le texte et dont un des coins est corné. Une ligne discontinue permet de relier la note à l'élément du modèle qu'elle cible.

La figure 2.9 représente une note attachée à l'opération nommée `getNom()`.

Figure 2.9
Représentation graphique d'une note associée à une opération



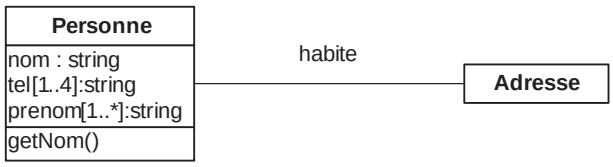
Associations entre classes

Le langage UML définit le concept d'association entre deux classes. Ce concept très intéressant, qui ne fait pas partie des concepts élémentaires du paradigme objet, permet de préciser les relations qui peuvent exister entre plusieurs objets.

En UML, une association se fait entre deux classes. Elle a un nom et deux extrémités, qui permettent de la connecter à chacune des classes associées. Lorsqu'une association est définie entre deux classes, cela signifie que les objets instances de ces deux classes peuvent être reliés entre eux.

La figure 2.10 présente l'association nommée `habite`, qui associe les classes `Personne` et `Adresse`. Cette association signifie que les objets instances de la classe `Personne` et les objets instances de la classe `Adresse` peuvent être reliés. En d'autres termes, cela signifie que des personnes habitent à des adresses.

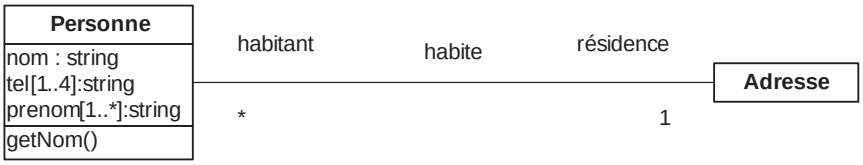
Figure 2.10
Représentation graphique d'une association nommée



Chaque extrémité de l'association a un nom de rôle, qui permet d'identifier chacune des classes liées dans le contexte de l'association.

La figure 2.11 représente la même association en précisant le nom des rôles de chaque classe liée. Dans le contexte de cette association, la classe `Personne` représente l'habitant alors que la classe `Adresse` représente la résidence. En d'autres termes, cette association signifie que les personnes habitent à des adresses et qu'ils sont les habitants de ces résidences.

Figure 2.11
Représentation graphique d'une association et de ses rôles



En UML, il est possible de spécifier à chaque extrémité les nombres minimal et maximal d'objets devant être reliés.

La figure 2.12 représente la même association en précisant les nombres minimal et maximal d'objets devant être reliés. La lecture de ce diagramme est la suivante :

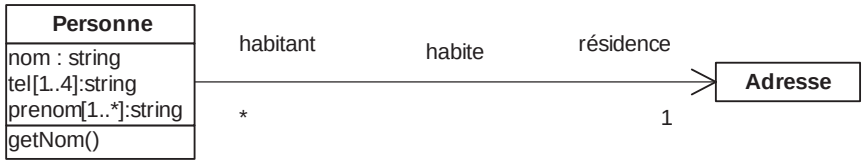
- `résidence 1` : pour 1 habitant, il y a au minimum 1 résidence et au maximum 1 résidence.

- `habitant *` : pour 1 résidence, il y a au minimum 0 habitant et au maximum une infinité d'habitants.

En UML, il est possible de rendre chacune des extrémités navigable ou non. Si une extrémité est navigable, cela signifie que l'objet peut naviguer vers l'autre objet auquel il est relié et ainsi obtenir les valeurs de ses propriétés ou invoquer les opérations dont il est responsable.

À la figure 2.12, les habitants peuvent naviguer vers leurs résidences (et pas l'inverse), ce qui permet d'obtenir, par exemple, le numéro de rue.

Figure 2.12
Représentation graphique d'une association navigable



Pour finir, il est possible en UML de préciser des sémantiques de contenance sur les associations. Par exemple, il est possible de préciser sur une extrémité qu'une classe associée joue un rôle de conteneur, l'autre classe jouant le rôle de contenu.

UML propose deux sémantiques de contenance : une sémantique de contenance faible, dite d'agrégation, qui permet de préciser que les éléments contenus peuvent être partagés entre plusieurs conteneurs, et une sémantique de contenance forte, dite composition, qui permet de préciser que les éléments contenus ne peuvent être partagés entre plusieurs conteneurs.

Du point de vue graphique, la relation de contenance se représente à l'aide d'un losange sur l'extrémité. Le losange est blanc pour l'agrégation et noir pour la composition.

La figure 2.13 précise que la classe `Personne` joue un rôle de conteneur pour la classe `CompteBancaire` dans le cadre de l'association `moyens de paiement`, ce qui signifie qu'un compte bancaire ne peut être le « moyen de paiement » que d'une seule personne.

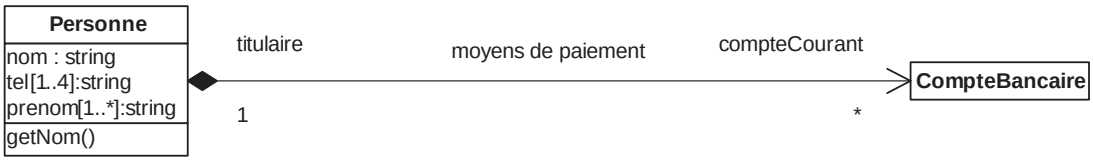


Figure 2.13
Représentation graphique d'une association de composition

Concepts avancés

Les concepts élémentaires que nous venons de présenter sont largement utilisés pour modéliser la vue comportementale d'une application. Les concepts avancés que nous présentons le sont moins mais permettent cependant de faciliter la réalisation des opérations de Reverse Engineering et de génération de code que nous présenterons dans les chapitres suivants de ce cours.

Classe abstraite

Sémantique

Une classe UML peut être abstraite. Dans ce cas, elle ne peut pas directement instancier un objet.

Dans une classe abstraite, il est possible de préciser que certaines propriétés et certaines opérations sont abstraites. Ce sont précisément les valeurs de ces propriétés et les responsabilités de ces opérations que les objets ne peuvent pas supporter directement. C'est la raison pour laquelle aucun objet ne peut être directement instance d'une classe abstraite.

Pour que des objets soient instances d'une classe abstraite, il faut obligatoirement qu'ils soient instances d'une classe non abstraite, laquelle hérite de la classe abstraite et rend non abstraites les propriétés et les opérations abstraites.

Graphique

En UML 2.0, il n'existe pas de représentation graphique particulière pour les classes abstraites. En UML 1.4, il fallait mettre le nom de la classe en italique.

Multiplicité des propriétés et des paramètres

Sémantique

Il est possible de préciser qu'une propriété ou un paramètre peut porter plusieurs valeurs. UML permet de préciser les nombres minimal et maximal de ces valeurs. Préciser qu'une propriété peut avoir au minimum une valeur et au maximum une infinité de valeurs revient à préciser que la propriété est un tableau infini.

Graphique

Pour les propriétés et les paramètres, les nombres minimal et maximal des valeurs apparaissent entre crochets. Le caractère `*` est utilisé pour préciser que le nombre maximal de valeurs est infini.

La figure 2.14 présente différentes propriétés en précisant des nombres minimal et maximal de valeurs.

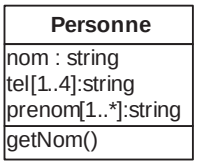


Figure 2.14
Représentation graphique des multiplicités des propriétés

Visibilité des classes, des propriétés et des opérations

Sémantique

Il est possible de préciser la visibilité des propriétés et des opérations des classes. Les visibilités portent sur les accès aux propriétés et aux opérations. On dit qu'une classe A accède à la propriété d'une classe B si le traitement associé à une opération de A utilise la propriété de B. On dit qu'une classe A accède à l'opération d'une classe B si le traitement associé à une opération de A fait un appel à l'opération de B.

Les visibilités proposées par UML 2.0 sont les suivantes :

- `public` : la propriété ou l'opération peuvent être accédées par n'importe quelle autre classe.
- `private` : la propriété ou l'opération ne peuvent être accédées que par la classe elle-même.
- `protected` : la propriété ou l'opération ne peuvent être accédées que par des classes qui héritent directement ou indirectement de la classe qui définit la propriété ou l'opération.

Graphique

Dans la représentation graphique de l'élément, les visibilités sont représentées de la façon suivante :

- Le caractère + est utilisé pour préciser la visibilité `public`.
- Le caractère - est utilisé pour préciser la visibilité `protected`.
- Le caractère # est utilisé pour préciser la visibilité `private`.

Propriétés et opérations de classe

Sémantique

Il est possible de préciser que la valeur d'une propriété définie par une classe est portée directement par la classe elle-même (et non par chacun des objets). De même, il est possible de préciser qu'une classe est directement responsable d'une opération qu'elle définit. On appelle ces propriétés et ces opérations des éléments de niveau « classe ».

Graphique

Dans la représentation graphique de la classe, les propriétés et les opérations de niveau classe sont soulignées.

Synthèse

Dans ce deuxième chapitre, nous avons présenté le diagramme de classes UML qui permet de représenter la vue structurelle des applications informatiques. Ce chapitre ne se veut pas un guide de référence du diagramme de classes. Nous avons simplement présenté les concepts relatifs à ce diagramme dont nous aurons besoin dans la suite du cours.

En introduisant ces concepts, nous avons détaillé aussi bien leur sémantique propre que la façon de la représenter graphiquement. Rappelons que la sémantique de ces concepts est proche de celle des langages de programmation orientés objet sans lui être équivalente.

Pour finir, soulignons le fait que les diagrammes de classes UML peuvent être employés à tout niveau d'abstraction. Rien n'empêche de représenter une application informatique à l'aide d'une seule classe (haut niveau d'abstraction) ou de représenter tous les composants de cette application comme des classes (bas niveau d'abstraction).

Travaux dirigés

TD2. Diagrammes de classes

Question 12 *Définissez la classe UML représentant un étudiant, caractérisé, entre autres, par un identifiant, un nom, un prénom et une date de naissance.*

Question 13 *Définissez la classe UML représentant un enseignant, caractérisé, entre autres, par un identifiant, un nom, un prénom et une date de naissance.*

Question 14 *Définissez la classe UML représentant un cours, caractérisé par un identifiant, un nom, le nombre d'heures de cours magistral, le nombre d'heures de travaux dirigés et un nombre d'heures de travaux pratiques que doit suivre un étudiant.*

Question 15 *Définissez les associations qui peuvent exister entre un enseignant et un cours.*

Question 16 *Définissez la classe UML représentant un groupe d'étudiants en utilisant les associations.*

Question 17 *Définissez l'association possible entre un groupe d'étudiants et un cours.*

Question 18 *Pensez-vous qu'il soit possible de définir un lien d'héritage entre les classes UML représentant respectivement les étudiants et les enseignants ?*

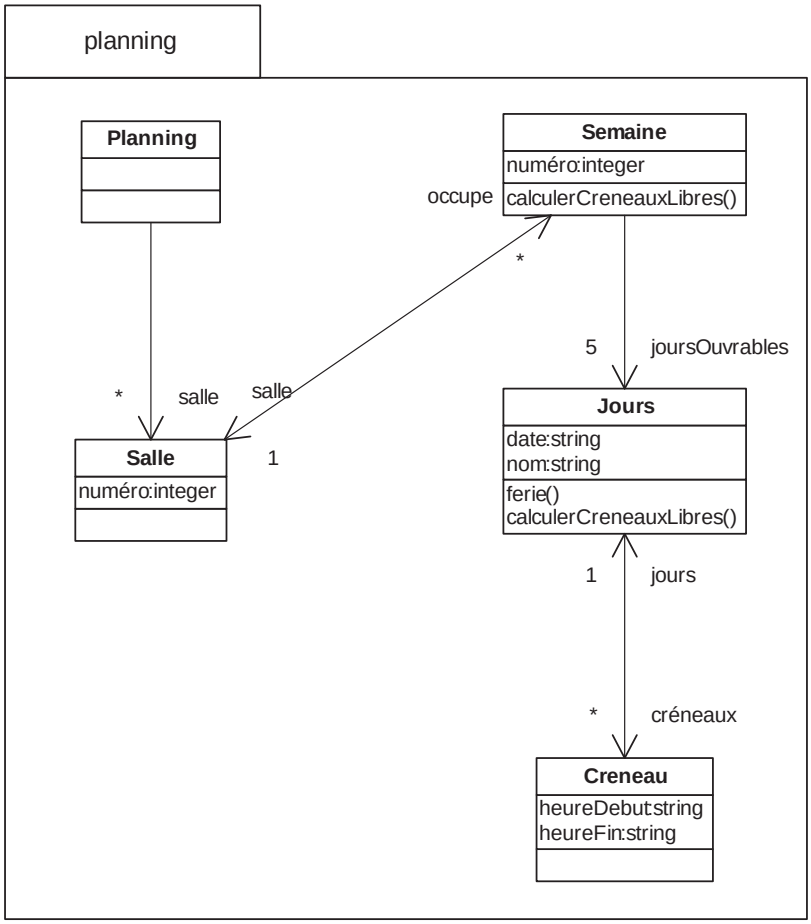
Question 19 *Pensez-vous qu'il soit possible de définir un lien d'héritage entre les classes UML représentant respectivement les étudiants et les groupes d'étudiants ?*

Question 20 *On nomme* `coursDeLEtudiant()` *l'opération permettant d'obtenir l'ensemble des cours suivis par un étudiant. Positionnez cette opération dans une classe, puis précisez les paramètres de cette opération, ainsi que les modifications à apporter aux associations préalablement identifiées pour que votre solution soit réalisable.*

Question 21 *On nomme* `coursDeLEnseignant()` *l'opération permettant d'obtenir l'ensemble des cours dans lesquels intervient un enseignant. Positionnez cette opération dans une classe, puis précisez les paramètres de cette opération, ainsi que les modifications à apporter aux associations préalablement identifiées pour que votre solution soit réalisable.*

Question 22 *Expliquez le diagramme de classes représenté à la figure 2.15.*

Figure 2.15
Package planning



Question 23 *Positionnez toutes vos classes (*`Etudiant`*,* `Enseignant`*,* `Cours`*,* `GroupeEtudiant`*) dans un package nommé* `Personnel`*.*

Question 24 *Liez vos classes pour faire en sorte qu'un créneau soit lié à un cours !*

Ce TD aura atteint son objectif pédagogique si et seulement si :

- Vous maîtrisez la notion de classe.
- Vous maîtrisez la signification des associations et héritages entre classes.
- Vous avez compris la répercussion des associations entre classes sur les dépendances entre packages.