NOVEI FORMATION

LA MANU

L'ÉCOLE DES MÉTIERS DU NUMÉRIQUE
Manufacture de compétences

SUPPORT APPRENANT

UML

CONFIDENTIEL

Ce document est strictement confidentiel et ne doit pas être diffusé sans accord préalable écrit

UML

Sommaire

Préambule

UML est l'acronyme d'**U**nified **M**odeling **L**anguage (ou Langage de Modélisation Unifié en français). C'est un langage de modélisation graphique basé sur des pictogrammes. Il fournit une méthode normalisée pour visualiser la conception d'une application. Sa version actuelle est la 2.5. Il est couramment utilisé dans en conception orientée objet.
L'UML est constitué de schémas appelés *diagrammes*, chacun répondant à un besoin spécifique. Ils permettent de faire une description graphique et de mieux visualiser ce qu'on l'on essaye de documenter. Il en existe 14 dont 7 structurels (qui définissent la structure de tout ou partie de l'application) et 7 fonctionnels (qui aident à prévoir le comportement de l'application).

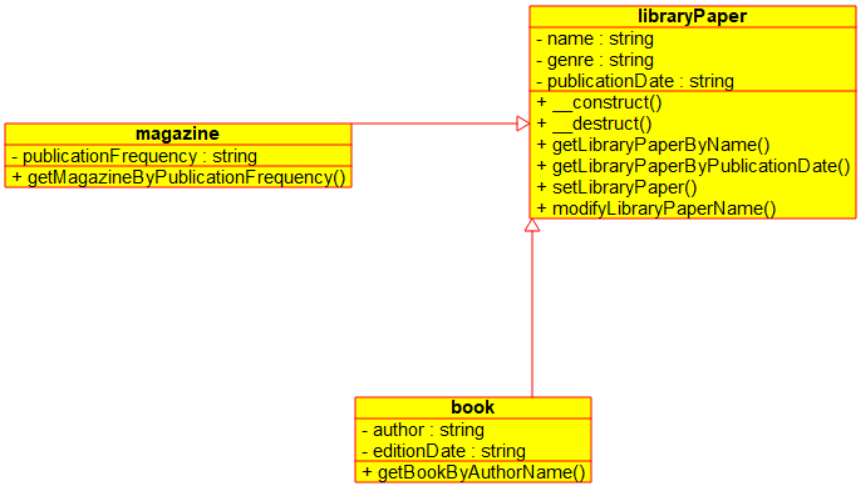
Diagrammes structurels	Diagrammes comportementaux
Diagramme de classes	Diagramme des cas d'utilisation
Diagramme d'objets	Diagramme états-transitions
Diagramme de composants	Diagramme d'activité
Diagramme de déploiement	Diagramme de séquence
Diagramme des paquets	Diagramme de communication
Diagramme de structure composite	Diagramme global d'interaction
Diagramme de profils	Diagramme de temps

I – Les diagrammes les plus utilisés

Les 4 diagrammes **les plus utilisés** sont le **diagramme de classe, le diagramme de déploiement, le diagramme de cas d'utilisation et le diagramme de séquence**.

A - Diagramme de classes (*class diagram*)

Il permet de construire **une classe avec ses attributs et ses méthodes**. Il peut également aider à définir une hiérarchie et les relations entre les classes. Certains logiciels comme *Umbrello* permettent de générer ces classes en codes à partir d'un diagramme de classe.



1 - Les classes

Une **classe** est représenté par un rectangle (ex : *libraryPaper*) et peut-être divisée en plus plusieurs parties :

- La première contient le nom de la classe (ici : *libraryPaper*)
- La deuxième contient les attributs de la classes (*name*, *genre*) suivi de son type (*string*, int)
- La troisième contient les opérations ou méthodes de la classe (*getLibraryPaperByName*()) ainsi que ses méthodes magiques (*__contruct*()). Elles peuvent être suivies du type de la valeur qu'elles retournent.

Chaque élément dans cette classe est précédé d'un symbole (+,-) qui définissent son niveau de visibilité (+ → Public, - → Privé, # → Protégé).

2 – Les relations

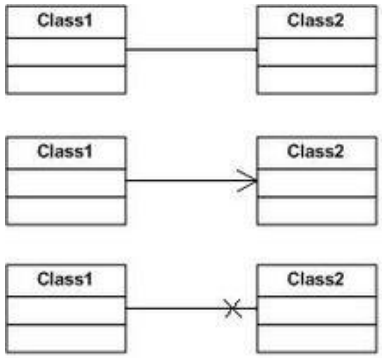
Les classes sont reliées entre elles par des **relations**. Il en existe plusieurs types :

- L'association
- L'agrégation
- La composition
- La dépendance
- L'héritage

a – l'association

L'association est une relation logique entre deux classes et peut-être nommée. Elle peut être binaire (représentée par un simple trait entre les classes) ou n-aire (les classes sont reliées par à des losanges par des traits simples). Elle est caractérisée par :

- la *multiplicité* (comparable aux cardinalités – nombre minimum et maximum d'instance de chaque classe reliée dans la relation)
- la *navigabilité*, qui indique si l'on peut passer d'une classe à l'autre.



b – l'agrégation

L'agrégation est une association qui relie une classe d'origine a une classe subordonnée. Elle est représentée par le symbole ci-dessous. Le losange se trouvant du côté de la classe d'origine.

c – la composition

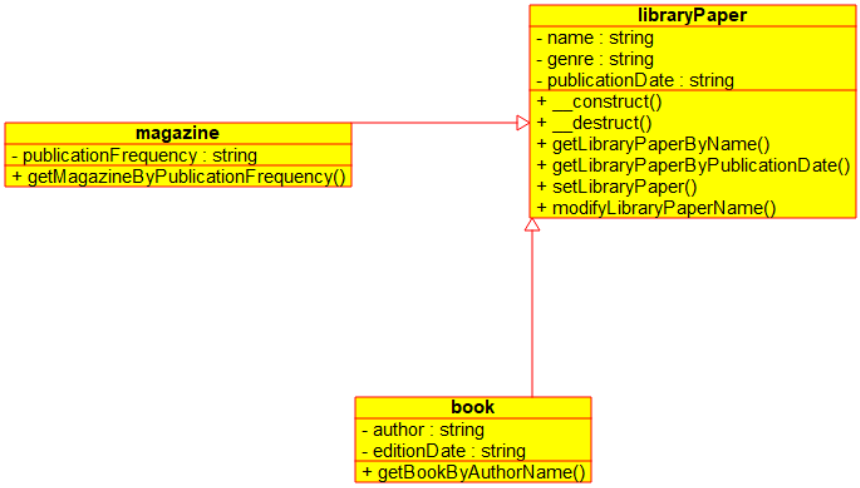
La **composition** est une agrégation qui relie une classe dite mère à une classe composée. Cette dernière est dépendante de la classe-mère. Elle est symbolisée par un trait simple se terminant par un losange plein.

d – la dépendance

La **dépendance** est la forme la plus faible de relation. Elle signifie qu'une des classes utilise l'autre.

e – l'héritage

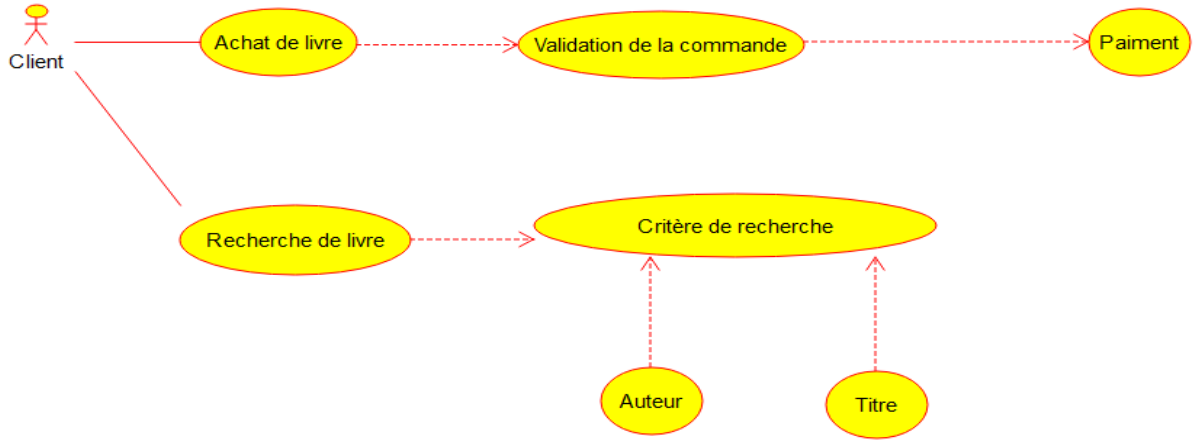
L'héritage permet de mettre en relation des classes ayant des attributs ou des comportements communs en respectant une hiérarchie. Il s'agit de la relation représentée sur notre premier exemple :



Ici, *libraryPaper* est la classe mère et *book* et *magazine* héritent de la classe mère. *Book* et *magazine* ont comme attributs communs *name*, *genre* et publicationDate. Ils peuvent également faire appel aux méthodes de la classe-mère.
Les enfants peuvent également avoir leurs propres attributs : par exemple, *magazine* a un attribut *publicationFrequency* qui ne concerne qu'elle, *book* n'y a pas accès. Il en est de même pour les méthodes (exemple : *getMagazineByPublicationFrequency*()).

B - Diagramme des cas d'utilisation (*use-case diagram*)

Le diagramme des cas d'utilisation représente les possibilités d'interaction entre le système et les acteurs (les utilisateurs, application externe), c'est-à-dire de toutes les fonctionnalités que doit fournir le système.



1 - Les acteurs

Ce sont les entités externes qui peuvent interagir avec l'application. Ce n'est pas forcément un être humain. Ils sont décrits par rôle et ne représente pas une seule personne. Par exemple, l'acteur ne sera pas Jean DUPONT mais « *client* », il peut ainsi représenter tous les clients qui utiliseront l'application et donc leurs besoins et objectifs. On dit que l'acteur agit sur le système. Le système doit parvenir à satisfaire le besoin. Il est représenté par ce pictogramme :

L'acteur est dit principal quand le cas d'utilisation chercher à répondre à son besoin. Si d'autres acteurs doivent être ajoutés, ce sont des acteurs secondaires, on ne répond pas à leurs besoins mais ils contribuent à répondre à celui de l'acteur principal.

2 - Les cas d'utilisation

Un cas d'utilisation est une fonctionnalité de l'application. Dans notre exemple : *l'achat d'un livre*. Il est représenté par un ovale.

3 – Association

C'est la relation qui va relier un acteur à un cas d'utilisation. Il est représenté par un trait continu. Un acteur peut agir plusieurs fois avec un cas d'utilisation. On peut donc y ajouter une multiplicité.

4 - Les relations

Les cas d'utilisations sont reliés entre eux par des relations. Les deux principales sont les dépendances stéréotypées et la généralisation/spécialisation.

a - Les dépendances stéréotypées

- *L'inclusion* : Relie deux cas dont l'une est dépendante de l'autre. Cette relation (représentée par une flèche en pointillé) peut également permettre de découper une fonctionnalité en d'autres fonctionnalités plus petites.

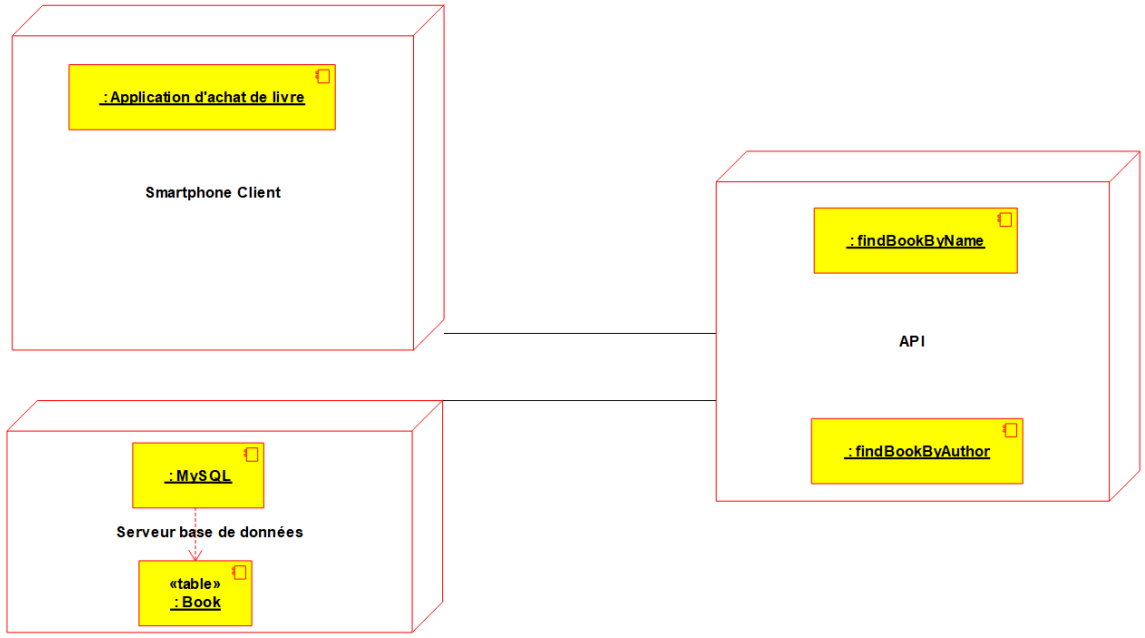
- *L'extension* : Relie deux cas d'utilisation qui ne dépendent pas obligatoirement l'un de l'autre. Un premier cas A peut étendre un deuxième cas B si le cas B peut appeler A au cours de son exécution.

b - La généralisation/spécialisation

Un premier cas est une généralisation d'un deuxième si celui-ci est un cas particulier de ce-dernier.

C - Diagramme de déploiement (*deployment diagram*)

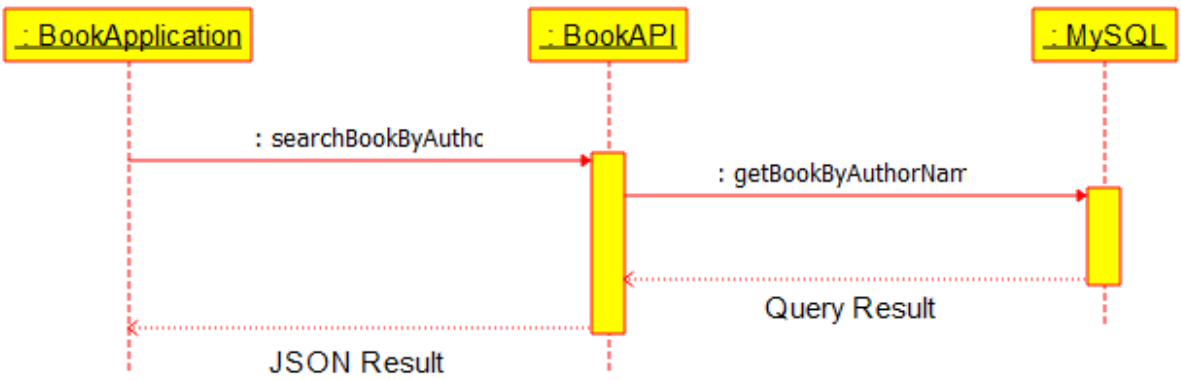
C'est une représentation des éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage…) et de la manière dont les composants de l'application sont répartis sur ces éléments. Il décrit également la manière dont ils interagissent entre eux.



Les ressources sont appelées nœuds et sont représentés par des cubes (ici : *Smartphone Client, API, Serveur base de données*).
Un composant est une partie de l'application ou du système (ici : *Application d'achat de livre*, *findBookByName*).
Pour affecter un composant à un nœud, il faut le placer à l'intérieur ou les relier par une relation de dépendance stéréotypée.
Les artéfacts sont des éléments concret (document, script, table de base de données). Ici la *table book* du nœud *Serveur de base de données* est un artéfact.

D - Diagramme de séquence (*sequence diagram*)

Il représente de façon séquentielle le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.



L'acteur principal du diagramme est à gauche (ici : *BookApplication*). Il est représenté, comme les acteurs secondaires par un rectangle. Les différentes actions sont aussi appelés messages et relient

les différents acteurs. Par exemple, ici *BookApplication* envoie le message *searchBookByAuthor* à *BookAPI*. La dimension verticale du diagramme représente le temps. Ici, *searchBookByAuthor* intervient avant *getBookByAuthorName*.

II – Autres diagrammes

A - Diagramme d'objets (*object diagram*)

Là où le diagramme de classe représente la classe en elle-même, le digramme d'objet (ou *object diagram*) représente une instanciation de cette classe.
Dans l'exemple ci-dessous, le diagramme d'objet représente Jean (instanciation de la classe auteur) et les relations avec Brouillon et final qui sont des instanciations de la classe Document.
Le diagramme d'objet est utilisé pour montrer l'état d'un ou plusieurs objets avant et après une interaction.

B - Diagramme de composants (*component diagram*)

C'est une représentation des composants de l'application d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...). Il permet de mettre en évidence la dépendance des composants entre eux.

C - Diagramme des paquets (*package diagram*)

Ici, un paquet est un ensemble de classes, des fichiers de configuration, fonctions, etc.. qui s'organisent de manières logiques.
Le diagramme de paquets représente les dépendances entre les paquets.

D - Diagramme de structure composite (*composite structure diagram*)

Il définit la structure interne d'une classe ainsi que les interactions ou collaborations que cette dernière rend possibles. Il est composé de parts (ou parties), les ports par lesquels ces parts interagissent entre elles, avec les classes, ou avec l'utilisateur et de connecteurs qui relient les parties.

E - Diagramme de profils (*profile diagram*)

Spécialise et personnalise pour un domaine particulier un meta-modèle de référence d'UML (depuis UML 2.2).

F - Diagramme états-transitions *(state machine diagram*)

Comme son nom l'indique, il définit les différents états d'un système ou d'une application et les transitions qui font passer d'un état à un autre. Il est plus utilisé dans le génie logiciel.

G - Diagramme d'activité (*activity diagram*)

C'est une représentation (sous forme de flux ou d'enchaînement d'activités) du comportement du système ou de ses composants.

H - Diagramme de communication (*communication diagram*)

Représente de façon simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets (depuis UML 2.x).

I - Diagramme global d'interaction (*interaction overview diagram*)

C'est une représentation des enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité) (depuis UML 2.x).

J - Diagramme de temps (*timing diagram*)

Il représente les variations d'une donnée au cours du temps (depuis UML 2.3).

Liens utiles

https://laurent-audibert.developpez.com/Cours-UML/
https://fr.wikipedia.org/wiki/UML_(informatique)#Diagrammes
http://remy-manu.no-ip.biz/