LES CONCEPTS OBJETS

I Objet et Classe

Un objet est une entité du monde réel qui a
- très souvent un identifiant
- des propriétés
- des comportements (actions qu'il peut effectuer).

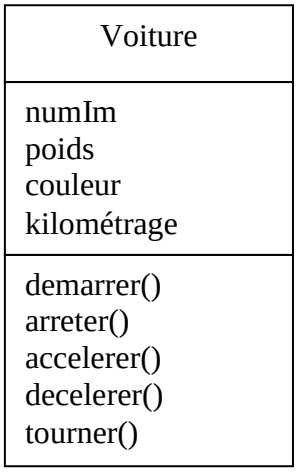
La voiture de Clément a pour numéro d'immatriculation AB-165-RE, elle pèse 1,5T, elle est verte. Elle a 55000 kilomètres au compteur. La voiture peut démarrer, s'arrêter, accélérer, décélérer, et tourner.

La voiture de Nicolas a pour numéro d'immatriculation AC-271-BB, elle pèse 2T, elle est noire, elle a 90000 kilomètres au compteur. Cette voiture peut démarrer, s'arrêter, accélérer, décélérer, et tourner.

Un objet est responsable de ses comportements (actions). Par exemple, si la voiture de Nicolas reçoit l'ordre d'accélérer il peut réagir de deux façons : exécuter l'accélération ou refuser le message/ordre et envoyer un message d'erreur (par exemple, en cas de dépassement de la vitesse limite).

I.a Les classes

On regroupe les objets ayant les mêmes types de propriétés et de comportements en une classe.



L'*instanciation* de la classe VOITURE est la création d'un objet de la classe VOITURE. Une instance de la classe VOITURE est une voiture donnée (objet « de type » VOITURE).

II Encapsulation

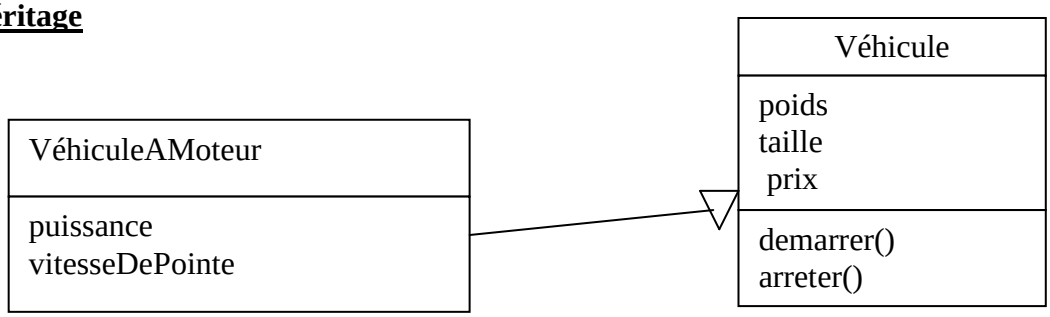
Objet/Classe a deux aspects :
- Interface : vue externe de l'objet
- Corps : implémentation des comportements (opérations) et des attributs.

D'un objet, un utilisateur ne connaît que l'interface. L'implémentation est masquée et non accessible à l'utilisateur.

- Un utilisateur/programmeur peut utiliser une classe (créer/manipuler des objets de la classe) sans connaître le corps de la classe [sans être concepteur de la classe]. Si la classe est correctement définie, l'utilisateur « bridé » ne peut pas violer l'intégrité de l'objet [par exemple, transformer une liste ordonnée en une liste non-ordonnée].
- Le concepteur peut modifier le corps d'une classe. Si l'interface de la classe n'est pas modifiée alors les utilisateurs (programmes) de la classe ne doivent pas être pas affectés par les modifications du corps.

Les deux dernières propriétés sont très utiles pour l'usage et la définition de « librairie de composants de logiciels » : les librairies qui peuvent évoluer ; et les utilisateurs (qui sont des programmes) ne peuvent pas violer l'intégrité des objets des classes des librairies.

III Héritage



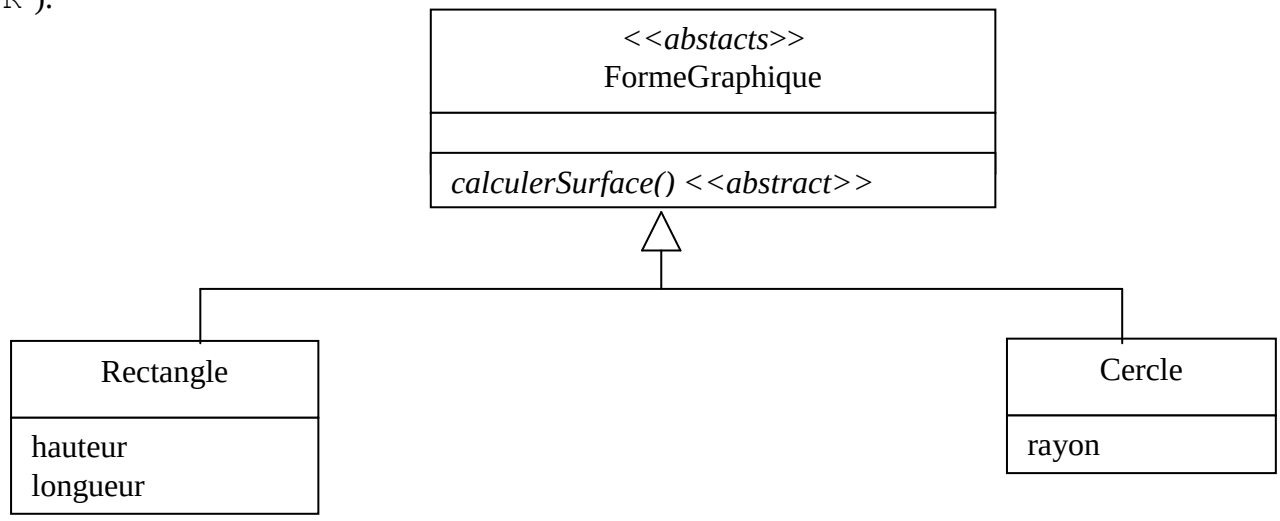
Un objet de la classe « Véhicule à Moteur » hérite des propriétés de la classe « Véhicule » Donc, un tel objet a un poids, une taille et un prix. De plus il peut s'arrêter et démarrer. Par ailleurs, il a une puissance et une vitesse de pointe (propriétés que n'ont pas les véhicules sans moteur à explosition).

On dit que la classe « VéhiculeAMoteur » est dérivée (ou héritée) de la classe « Véhicule ».

IV Polymorphisme

Polymorphisme : plusieurs formes pour la même opération.

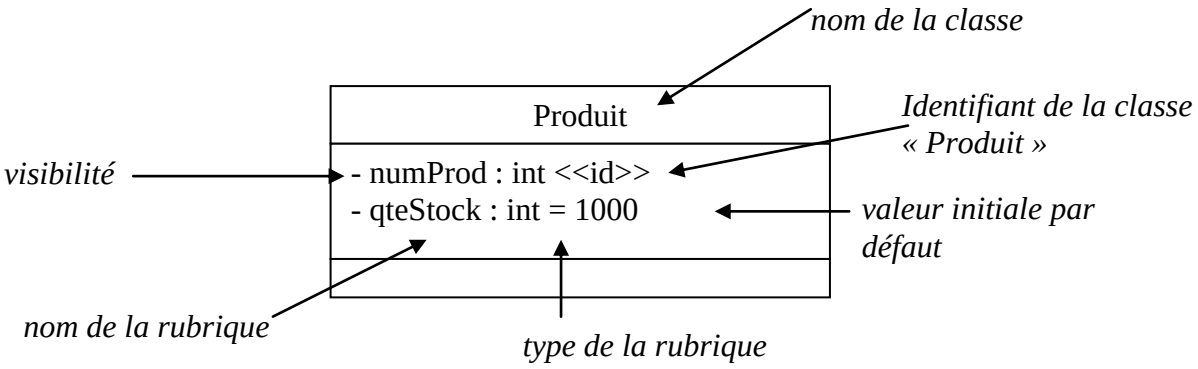
Il est impossible de calculer la surface d'une forme graphique si on a aucune information sur cette forme [la classe forme graphique est donc une classe *abstraite*].
2 formes pour le comportement calculerSurface une pour le rectangle ($H.L$) et une pour le cercle ($\Pi.R^2$).



UML :
DIAGRAMME DE CLASSES

Le diagramme de classes est basé sur les notions fondamentales de classes d'objets et d'associations.

I Classe

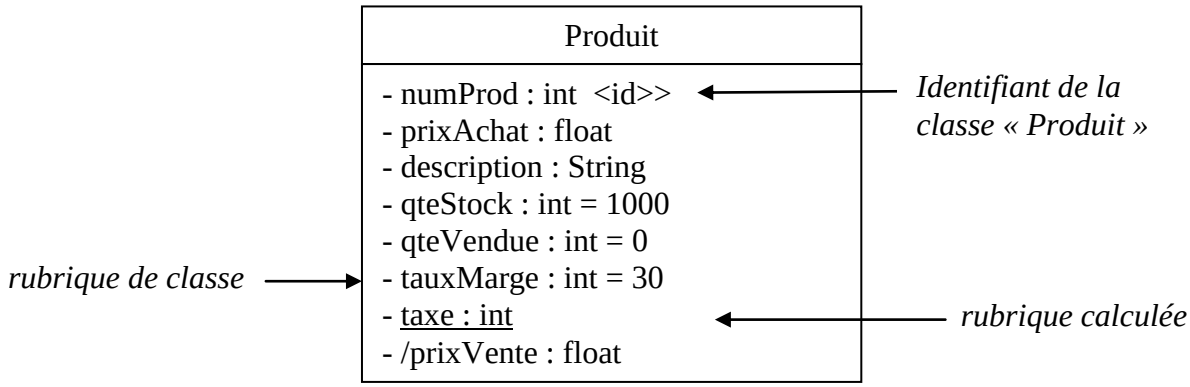


Types Prédéfinis (identique aux types prédéfinis en Java ainsi qu'aux classes des librairies les plus utilisés):

- Boolean (ou bool en C++)
- int
- void
- double ou float (réel)
- Date
- String (ou string en C++)

Visibilité :

- privé : accessibilité uniquement par l'objet. Les attributs sont généralement de visibilité « privé ».
+ publique : accessible par tout autre objet. Les opérations sont généralement « publiques ».
# protégé : accessible par les objets des sous-classes (classes dérivées) de la classe initiale.



taxe : rubrique de classe. Cette rubrique a la même valeur pour tous les objets de la classe [la rubrique est soulignée]. En C++, une rubrique de classe est traduite par un attribut dans la classe de type « static ».

prixVente : rubrique calculée. La valeur de cette rubrique est égale à :

```
prixAchat (100 + tauxMarge) (100*taxe) /10000
```

Le symbole « / » signale que la rubrique est calculée.

I.b Les méthodes

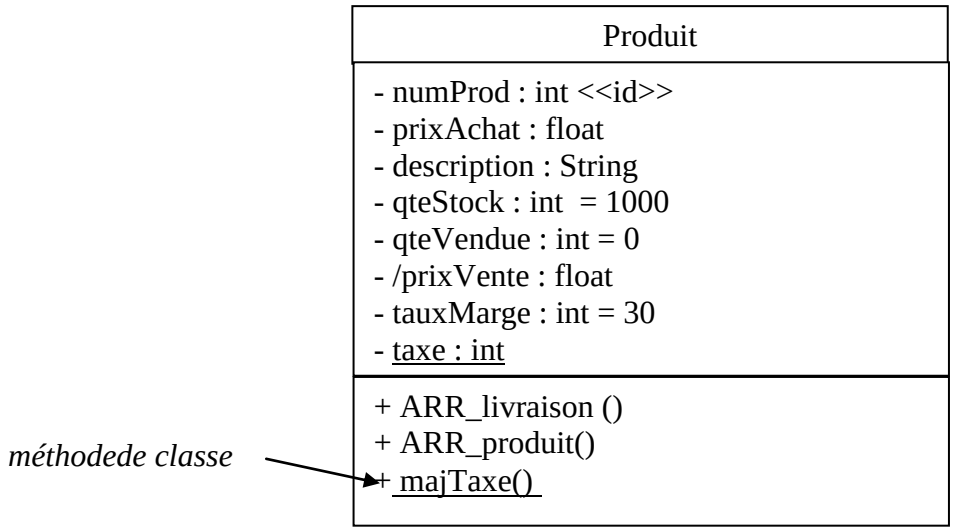
Une description complète d'une classe à trois parties :

- nom,
- liste des attributs avec leur type,
- liste des méthodes (signées)

Une méthode d'objet, ne peut être appelée qu'au travers d'un objet/instance de la classe. L'exécution de cette méthode est faite sur l'objet d'appel : la méthode peut accéder en lecture et écriture à toutes les valeurs des rubriques publiques ou privées de l'objet par lequel l'opération a été appelée et peut appeler toutes les méthodes de cet objet.

Une méthode de classe peut uniquement accéder aux rubriques de classe. Par exemple, majMarge (méthode de classe) est appelée pour modifier le taux de la marge de l'entreprise sur les produits (marge identique pour tous les produits et qui est donc une rubrique de classe). En C++ et Java, une méthode de classe est traduite par une méthode « static » de la classe.

Description incomplète d'une classe (il manque la signature des méthodes).



Par exemple, ARR_livraison est appelé au travers du produit livré : serviette900976.ARR_livraison
(il s'agit de la livraison des serviettes de référence 900976). Le seul paramètre nécessaire à cette opération est la quantité livrée. La méthode va mettre à jour la quantité en stock du produit livré (accès en écriture à l'attribut qteStock).

Signature d'une méthode :

- liste des arguments avec leur type
- type retourné par l'opération

exemple : ***void ARR_livraison (qte : int) : void***
ARR_produit (desc : String, prix : float) : void
La description complète d'une classe inclut la signature des méthode.

Le polymorphisme est la définition de plusieurs signatures pour la même opération Il ne faut confondre le polymorphisme et la surcharge. Par exemple, dans la définition suivante de la classe « Produit », la méthode ARR_produit est polymorphe.

La description complète de la classe « Produit »:

Produit
- numProd : int <<id>> - prixAchat : float - description : String - qteStock : int = 1000 - qteVendue : int = 0 - /prixVente : float - tauxMarge : int = 30 - <u>taxe : int</u>
+ ARR_livraison (qte :int) :void + ARR_produit () : void + ARR_produit (desc : String, prix : float) : void + <u>majTaxe(mg : int) : void</u>

1c Représentation d'une classe

Des parties de la description d'une classe peuvent être omises : liste des rubriques ou listes des méthodes.
Il existe donc plusieurs représentations possibles pour une classe (outre, celle qui est complétée).

La plus simple comporte uniquement le nom de la classe :

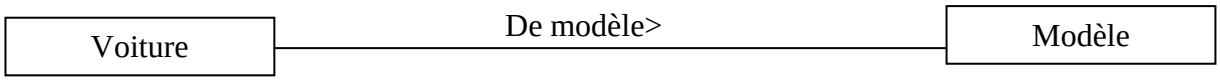
Produit

Dans la seconde, les attributs (typés ou non typés) sont listés, mais pas les méthodes. Généralement, cette notation est utilisée dans les phases préliminaires de l'analyse

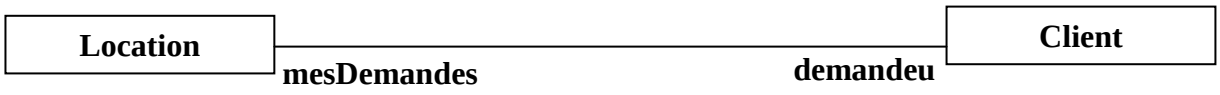
Produit
- numProd <<id>> - prixAchat - description - qteStock - qteVendue -tauxMarge - <u>taxe</u> - /prixVente

II Les Associations

Les associations sont binaires [concerne deux classes].



On peut nommer une association uniquement à l'aide des rôles.



II.a Multiplicités

Un schéma Entité/Association est aussi nommé Modèle Conceptuel des données (MCD) de Merise.

1..1 : multiplicité minimale et maximale sont égale à 1. [dans le schéma Entité/Association c'est noté (1,1)]. Aussi noté 1.

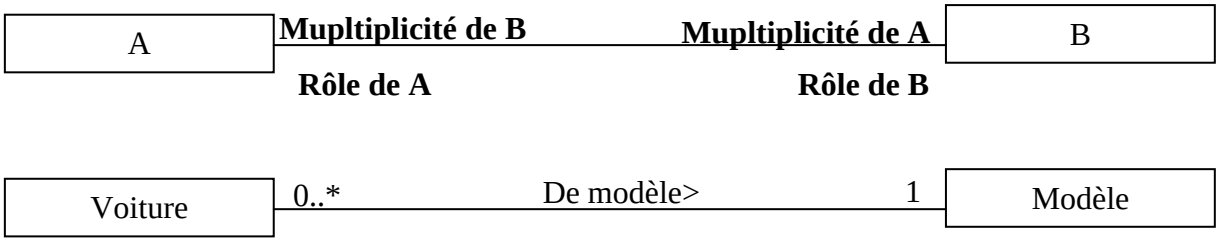
x..x : multiplicité minimale et maximale sont égale à x. [dans le schéma Entité/Association c'est noté (x,x)]. Aussi noté x. x doit être un entier positif et non nul.

0..1 : multiplicité maximale est égale à 1 ; la multiplicité minimale est égale à 0. [dans le schéma Entité/Association c'est noté (0,1)].

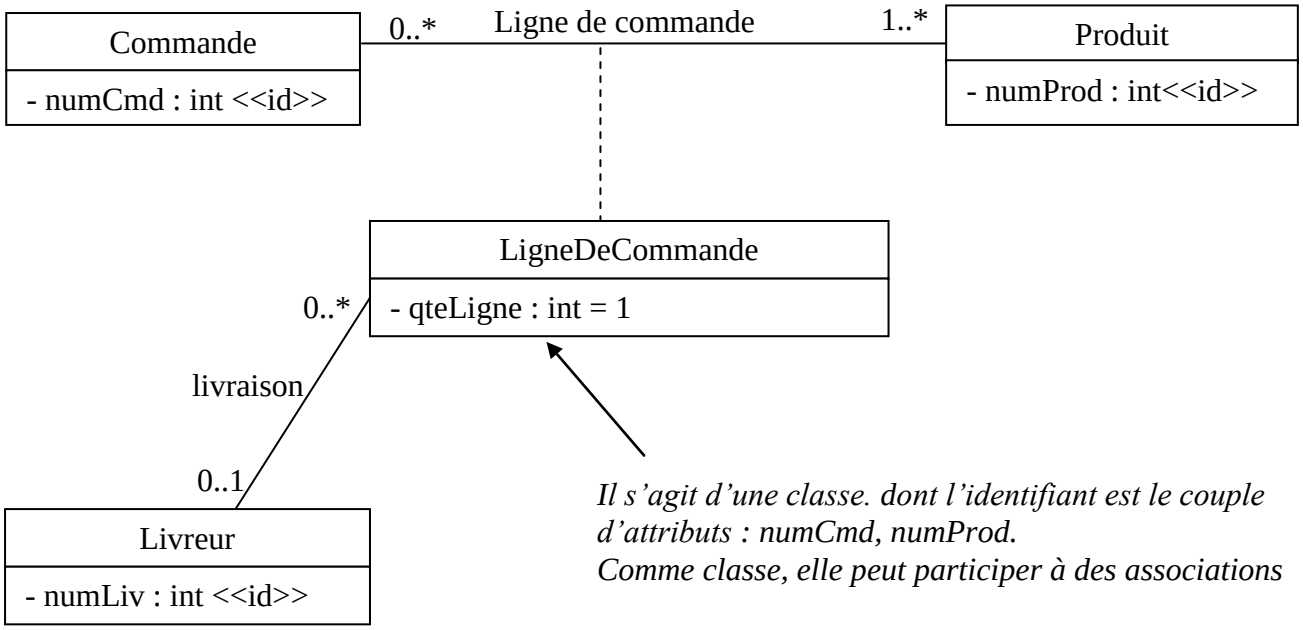
0..* : multiplicité maximale est égale à n ; la multiplicité minimale est égale à 0. [dans le schéma Entité/Association c'est noté (0,n)]. Aussi noté 0..*.

1..* : multiplicité maximale est égale à n ; la multiplicité minimale est égale à 1. [dans le schéma Entité/Association c'est noté (1,n)].

Attention à la place des multiplicités.

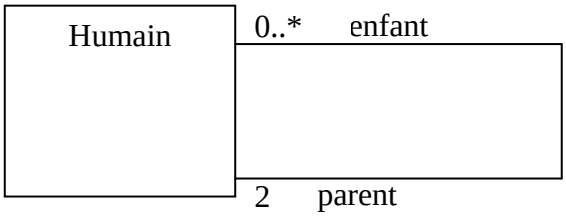


II.c Attribut d'associations

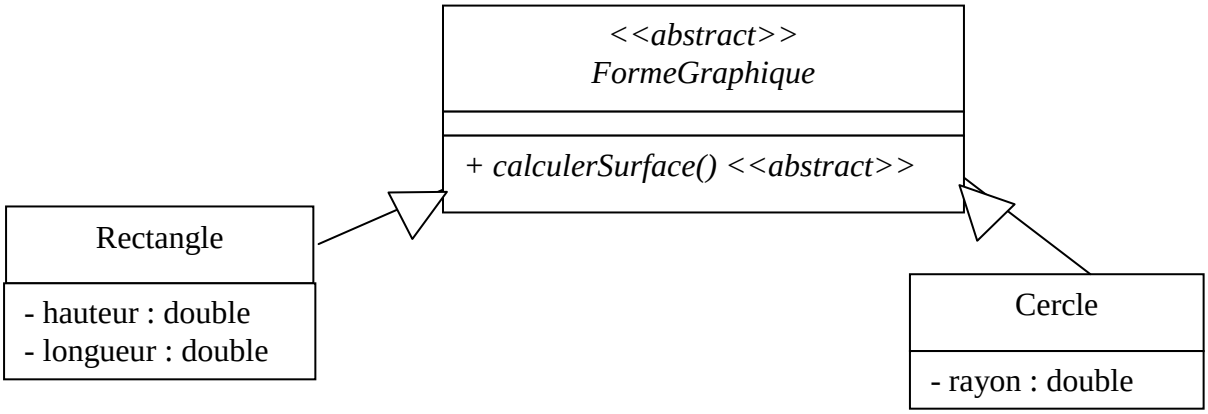


Remarque : On donne à chaque classe un identifiant de référence. UML ne l'impose pas ; mais dans le cadre d'une implémentation sur SGBD relationnel, il est indispensable que les objets aient des identifiants.

II.d Association réflexive



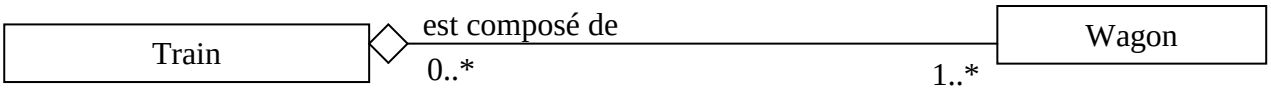
II.e Généralisation/Spécialisation – Héritage



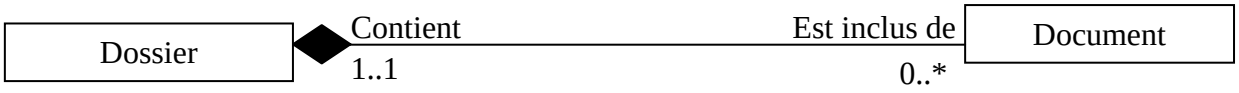
II.f Agrégations et compositions

Une agrégation ou une composition est une association dont la sémantique est de type « composé-composant ». A priori, les objets agrégés n'ont de raison d'être que s'ils sont liés à l'agrégat.

L'agrégation est une « composition faible ». Même si une entreprise fait faillite, les salariés de cet entreprise sont toujours assurés auprès de la sécurité sociale. Le lien entre « Train » et « Wagon » est un lien d'agrégation.



La composition est une « composition forte ». La destruction d'un dossier entraîne la destruction de tous les documents contenus dans le dossier. La composition implique une contrainte sur la valeur de la multiplicité du composant : elle peut prendre pour valeur 0..1 ou 1..1.



III Comparaison avec Merise

Dans Merise, l'aspect statique est modélisé à l'aide de deux schémas :

- MCD - Modèle Conceptuel des Données (schéma Entité/Association) qui modélise la structure abstraite des données persistantes (des données à stocker dans des fichiers ou dans une base de données).
- MLD - Modèle Logique des Données (schémas de relation ou schéma technique) qui modélise la structure des fichiers de données.

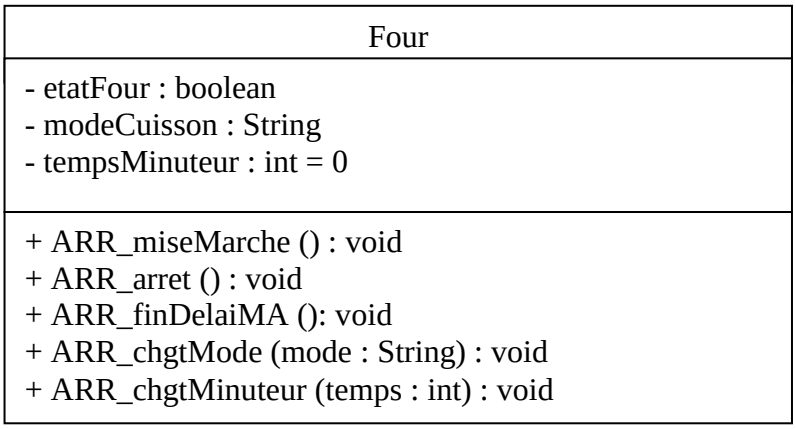
Dans UML, le même diagramme est utilisé dans la phase d'analyse et de conception : diagramme de classes.
La phase d'analyse consiste à comprendre, expliquer, et représenter la nature du système à informatiser. Elle identifie le « quoi faire » et l'environnement sans décrire « comment » le système sera réalisé.
La phase de conception consiste à chercher une solution au problème (concevoir le logiciel) tel qu'il a été défini dans la phase d'analyse. La conception peut être divisée en deux parties : une étape logique indépendante de l'environnement de réalisation et une étape physique qui se préoccupe des particularités des langages de programmation utilisés, des SGBD, de l'environnent d'exécution (matériel, système d'exploitation et réseaux).

Donc, le diagramme de classes inclut des éléments propres à l'étape logique : type des rubriques, signatures des opérations, visibilité, et rubriques calculées ou paramètres.

Le diagramme de classes modélise la structure des données manipulées par le logiciel et pas seulement les données persistantes (données à stocker dans des fichiers ou une BD). Exemple : Dans le cadre de la modélisation du fonctionnement d'un four à micro-onde. Le diagramme de classes contient une seule classe (qui a une seule instance) FOUR.

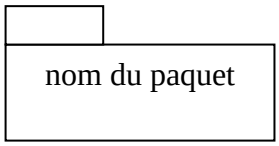
Remarquons que les opérations ARR_mise_marche, et ARR_arrêt n'ont pas de paramètre. Ces opérations n'ont pas besoin d'information une fois qu'elles connaissent le four (à mettre en marche ou à arrêter) concerné par l'opération. Cette information est toujours donnée car ses opérations sont appelées au travers du « four » concerné.

Dans le cas de classes pouvant avoir simultanément plusieurs instances (objets de cette classe), la classe peut contenir un identifiant de référence. UML ne l'impose pas ; mais dans le cadre d'une implémentation sur SGBD relationnel, il est indispensable que les objets aient des identifiants.

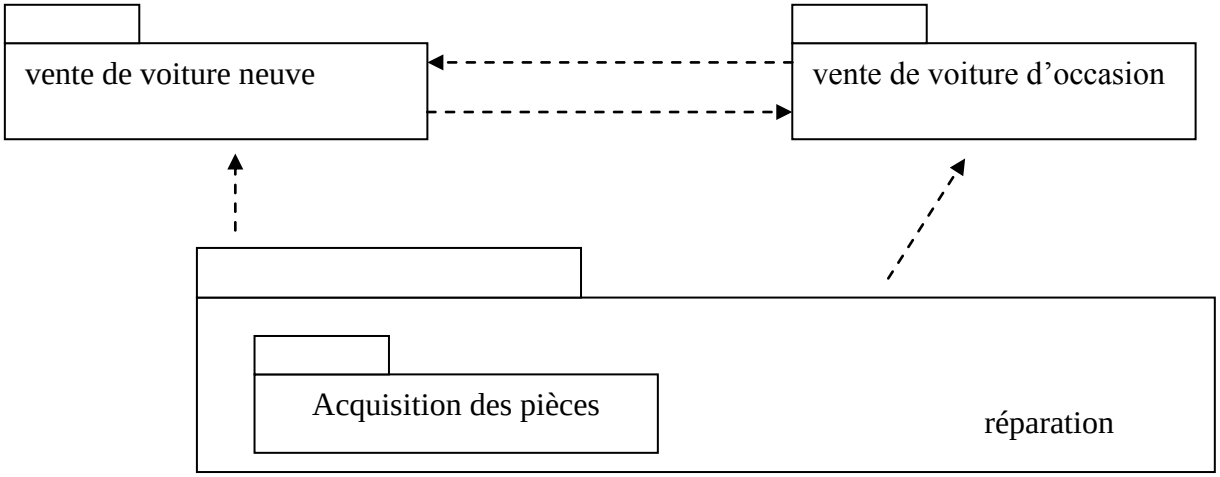


IV Les paquets

Les paquets offrent un mécanisme général pour structurer les éléments de modélisation dans un diagramme. Les paquets organisent les modèles de la même manière que les répertoires organisent les fichiers. Un paquet peut contenir d'autres paquets.

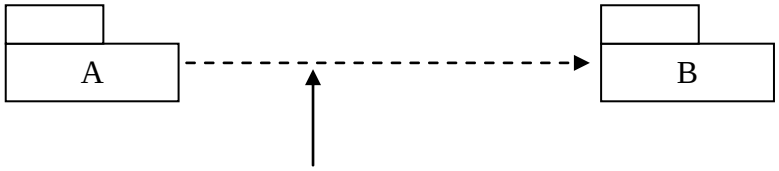


Par exemple, le diagramme de classes de l'étude de cas « Garage DEVINCRE » peut être divisé en 4 paquets :



Une classe est définie dans un seul paquet, mais peut apparaître dans plusieurs paquets (pour aider à la définition des associations). Une association est définie dans un seul paquet et apparaît dans un seul paquet.
Le paquet « vente de voiture neuve » contient la définition des classes : équipement, marque, modèle, commande et client et utilise la classe véhicule.
Le paquet « vente de voiture d'occasion » contient la définition de la classe véhicule et utilise la classe client.
Le paquet « réparation » contient la définition des classes : pièce, travail, catégorie et grossiste et utilise les classes marque, modèle, véhicule et client.

Dans ce cas là, une flèche entre le paquet A et le paquet B signifie qu'une classe définie dans le paquet B apparaît dans le paquet A. De manière générale, une flèche du paquet A vers le paquet B représente une relation de dépendance du paquet A au paquet B.



Une classe définie dans le paquet B est utilisée dans le paquet A.