Génie logiciel

Notes du cours de 25/11

L3 Informatique appliquée 2022-2023

MABROUK Fayez

25 novembre 2022

1 UML pour modéliser la structure

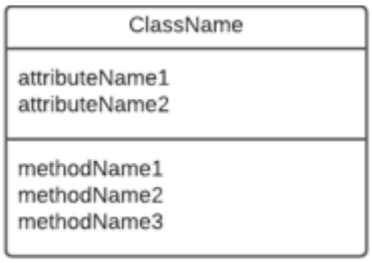
1.1 Découvrir des objets

* Les objets peuvent être représentés à différentes granularités.
* Il est important de choisir la granularité appropriée à l'objectif du diagramme :
  * Diagrammes de cas d'utilisation : diagramme de haut niveau destiné à la discussion : objets à faible granularité.
  * Diagrammes de classes réalisés pour la conception du programme : objets à granularité élevée.
* Exemple : un ordinateur portable peut être vu comme.. :
  * un... objet ordinateur portable (faible granularité).
  * la composition d'un châssis, trackpad, clavier, écran, carte mère, CPU, RAM, ... (granularité élevée).
* Il est possible de découvrir des objets par :
  * dynamique : en regardant quel objet doit recevoir le message.
  * données : en analysant la structure de l'objet.
* Exemple : on peut trouver des objets dans un ordinateur portable :
  * dynamiquement : Je veux exécuter un programme : d'abord, je déplace mon curseur avec le trackpad jusqu'à la barre de recherche ; ensuite, je tape le nom du programme. la barre de recherche ; ensuite je tape le nom du programme avec le clavier ; puis le CPU exécute le programme...
  * à travers des données : quand je regarde les spécifications de mon ordinateur portable, je peux voir qu'il a un intel i7 avec 16Go de RAM, un clavier AZERTY, ...

1.2 Types de diagrammes

* Diagrammes de structure :
  * Diagramme de classes, diagramme d'objets, diagramme de composants, diagramme de structure composite, Diagramme de paquetage et Diagramme de déploiement

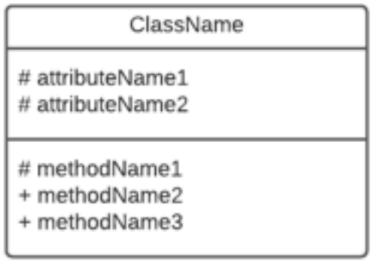
1.3 Représentation d'une classe



* Dans sa forme la plus simple, une classe est représentée comme suit :

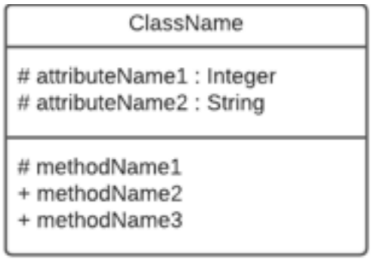
    * Un nom (toujours un substantif, au singulier, représente la nature des objets de cette classe).
    * Une liste d'attributs.
    * Une liste de méthodes.
* Manque d'informations, mais bon pour un premier tour de table de conception.

1.4 Encapsulation



* Rappel : définition de l'encapsulation : cacher certains attributs ou méthodes à d'autres objets.
* En UML, les éléments peuvent être qualifiés de :
    * public (signe : +) : l'élément n'est pas encapsulé ; visible par tout le monde.
    * protected (signe : ♯) : l'élément est encapsulé et visible dans les visible dans les spécialisations de cette classe.
    * private (signe : -) : l'élément est encapsulé (visible uniquement à l'intérieur de la classe).
    * package (signe : ∼) : l'élément est encapsulé et visible à l'intérieur du visible dans le paquetage (nous ne l'utiliserons pas dans cette classe).

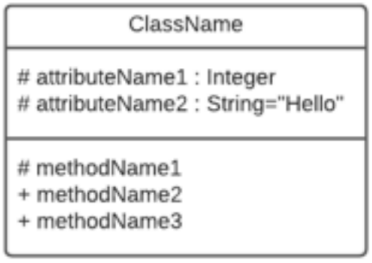
1.5 Types



* Dans un diagramme de classes, il est possible de spécifier le type d'une variable.
* Le type peut être l'un des types standard :
    * Booléen.
    * Entier.

    * Réel.
    * Chaîne.
* Ou il peut s'agir d'une classe (du système ou non).
* Indiqué par " :" après la variable.

1.6 Valeur par défaut

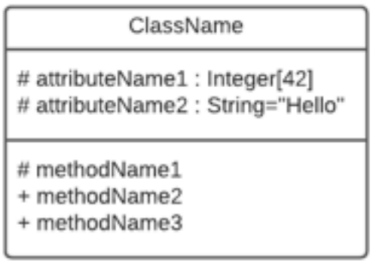


* Un attribut de classe peut avoir une valeur par défaut. par défaut. Dans ce cas, il sera indiqué par "=". valeur par défaut".

1.7 Cardinalité

Syntax	Cardinality
[1]	Exactly one time (default value, can be omitted)
[N]	Exactly N times
[*]	Any number (including 0) of times
[0..1]	0 or one time
[1..*]	One or more times
[M..N]	From M to N times

* Une variable peut avoir plusieurs valeurs.
* Indiqué par la cardinalité.
* Syntaxe : type [M..N].

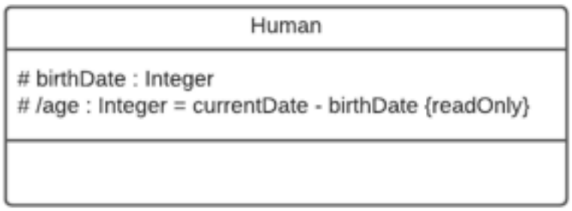


* Dans un langage de programmation, cela serait par un tableau, une liste, un vecteur, ...

1.8 Modificateurs

* Un modificateur peut être utilisé (il est facultatif) pour donner des informations supplémentaires sur une variable.
* Syntaxe modificateur ou m1, m2, ...
* Les plus courants :
  * id : la variable fait partie de l'identifiant de la classe
  * readOnly : la variable ne peut pas être modifiée.
  * rdered (applicable pour les cardinalités ¿ 1) : les valeurs de la variable doivent être ordonnées.
  * unique (applicable pour les cardinalités ¿ 1) : les valeurs de la variable doivent être uniques (par défaut !).
  * nonunique (applicable pour les cardinalités ¿ 1) : les valeurs de la variable ne doivent pas être uniques.
  * redefines "attribute name" : redéfinition de "attribute name" à partir de la superclasse. Si le type est modifié, le nouveau type doit être compatible avec l'ancien type.

1.9 Attributs dérivés

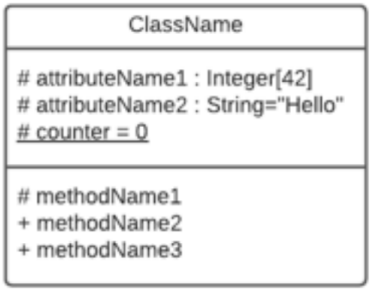


* Un attribut peut être dérivé d'autres informations (souvent : autre(s) attribut(s)).
* indiqué avec "/" devant le nom.
* Peut être suivi d'une expression expliquant comment calculer la valeur.
* Souvent "readOnly" (lecture seule).

1.10 Types 2

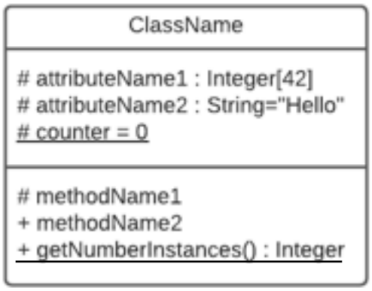
* Pour typer une méthode, nous parlerons de sa ”signature” :
  * Nom de la méthode.
  * Nom des paramètres, leurs types, leurs cardinalités et propriétés, valeur par défaut.
  * Résultat de la méthode : type, cardinalité, propriété.
* Syntaxe :
  name (direction nameParam :type[inf..sup]=Defaultmodifiers, . . .) : returnType[inf..sup] modifiers.
* Le sens peut être :
  * in : la valeur du paramètre est transmise au moment de l'appel.
  * out : la valeur du paramètre est transmise à la fin de l'exécution de la méthode.
  * inout : les deux.

1.11 Attributs dérivés



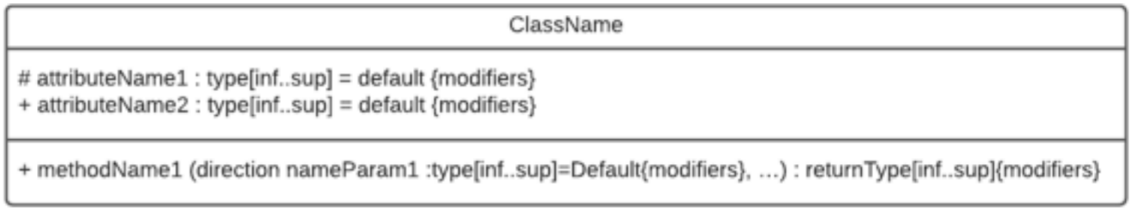
* Une instance d'une classe = une instance de chaque attribut.
* Il est possible d'avoir des attributs qui sont liés à la classe plutôt qu'à une instance. Dans ce cas, on parlera d'”attributs de classe”.
* Un attribut de classe est le même (nom, valeur, type) pour chaque instance. En général, il doit avoir une valeur par défaut.
* Un attribut de classe n'est jamais hérité.
* On y accède par la classe directement (pas par une instance).
* Syntaxe : soulignez l'attribut.

1.12 Méthodes de classe



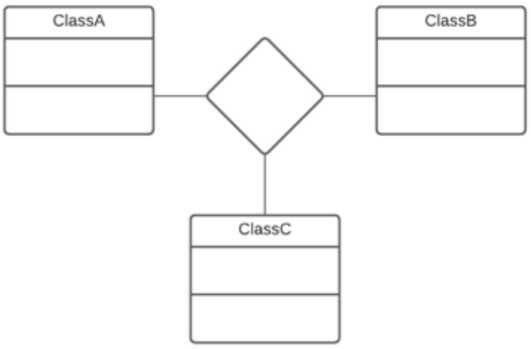
* De même, une méthode peut être liée à une classe, auquel cas nous parlerons de ”méthode de classe”. auquel cas on parlera de ”méthode de classe”.
* ne peut être accédée que par la classe elle-même et ne peut utiliser que les attributs de la classe.

1.13 Représenter une classe - récapitulation



1.14 Association entre les classes

* Une association peut être établie entre deux (ou plusieurs) classes qui sont reliées.
* Montre comment les classes sont liées à travers le système.
* Syntaxe ligne pleine (pour les relations binaires).

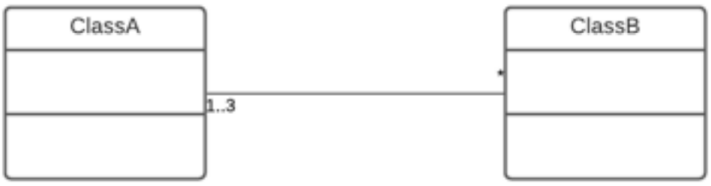


* Relations ternaires (ou N-aires) : avec un losange.

* L'association peut être nommée.
* Le rôle de chaque instance peut être indiqué avec la syntaxe "+role".

1.15 Cardinalité

* Possibilité d'ajouter la cardinalité aux associations.
* La cardinalité sur le côté d'une classe indique combien d'instances de cette classe sont liées à une instance de la classe à l'autre extrémité.
* Exemple : 1 à 3 instances de ClassA sont liées à chaque instance de ClassB. 0 instance ou plus de ClassB est liée à chaque instance de ClasseA



1.16 Composition

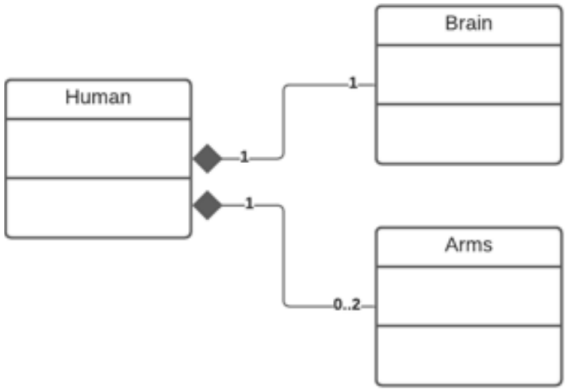
* Rappel : Composition : les objets complexes peuvent être composés d'autres objets.
* Elle est définie au niveau de la classe, mais on ne compose que des instances réelles.
* Elle peut être :

    * une relation forte : les composants ne peuvent pas être partagés ; la destruction de l'objet composé implique la destruction des composants. l'objet composé implique la destruction des composants
    * une relation faible (aussi appelée agrégation) : les composants peuvent être partagés.
* Une composition est une association !
* Relation "a".

1.17 Composition forte (a.k.a. composition)

* Composition forte : les composants ne peuvent pas être partagés -¿ la cardinalité du côté de la composition est toujours 1.

* Exemple : une instance d'un humain a un cerveau et au plus 2 bras.



1.18 Composition faible (alias agrégation)

* Composition faible : les composants peuvent être partagés et la destruction de l'objet l'objet qui compose ne détruit pas le composant.
* Plus fréquente que la composition forte.

1.19 Directionalité

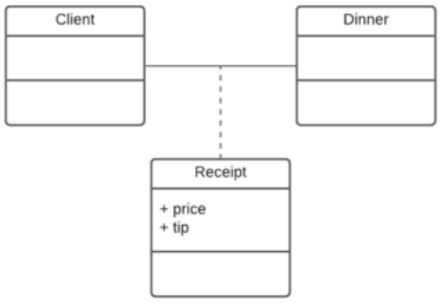
* Par défaut, les associations sont bidirectionnelles : à partir d'une instance de classeA, je devrais pouvoir trouver le lien vers la classeB et à partir d'une instance de la classeB, je devrais pouvoir trouver le lien vers la classeA. et à partir d'une instance de ClassB, je devrais être capable de trouver le lien vers ClassA.
* En général, pas bon en pratique.
* Possibilité d'ajouter une flèche pour indiquer le sens de l'association.
* Exemple : ClassA sait qu'elle est liée à ClassB, mais ClassB ne le sait pas.

1.20 Association réflexive

* Il est possible d'avoir une association d'une classe à elle-même.
* Peut être utilisé pour créer :
  * une hiérarchie.
  * un groupe.

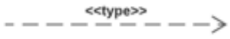
1.21 Classes d'association

* Dans certains cas, l'association peut être complexe.
* Dans ce cas, l'association peut être modélisée comme une classe.
* Syntaxe : ligne en pointillé entre la classe d'association et la classe de association.
* Exemple :



1.22 Association de dépendance

* Association dirigée.
* Indique qu'une classe a besoin d'une autre classe pour sa spécification ou son implémentation.
* Syntaxe :



* Le type indique la dépendance et peut être :
  * call : utilise une méthode.
  * create : crée une instance.
  * derive : indique une redondance.
  * instantiate : classe d'usine qui a besoin de cette autre classe pour sa spécification.
  * permit : permet l'accès à des éléments privés.
  * use : cas général.
  * Exemple : la classe ChocolateFactory est une usine qui a pour seul objectif de produire (instancier) des instances de barres de chocolat. but de produire (instancier) des instances de ChocolateBar. La ChocolateFactory a donc besoin de connaître la spécification de la barre chocolatée pour la produire.

1.23 Associations - récapitulation

* Une association entre deux classes représente un lien entre elles.
* Dans le cas général, l'association est qualifiée par des noms et des rôles et indique un lien simple entre les classes (trait plein).
* Composition : indique un composant qui ne peut pas être partagé.
* Agrégation : indique un composant qui peut être partagé.
* Flèche pointillée : dépendance entre classes.

1.24 Généralisation/spécialisation : rappel

* Spécialisation : une nouvelle classe A peut être créée en tant que sous-classe d'une autre classe B. Dans ce cas, la classe A spécialise la classe B.
* La spécialisation est une relation "est un".
* La généralisation est l'inverse (la superclasse B est une généralisation de la sous-classe A). sous-classe A).
* Héritage : le fait qu'une sous-classe obtienne le comportement et la structure de la superclasse.
* C'est une conséquence de la spécialisation.
* Syntaxe :