Isabelle Le Glaz

Conception Objet - Partie 2

Modeliser une conception objet

UML est une technique de modélisation, représentation, du design d’une application objet.

Elle repose sur des conventions de notations, permettant de représenter tous les cas de figures possibles qu’on peut rencontrer lors du design d’une application.

UML – Le diagramme de classe

- Le diagramme de classe présente un ensemble de classeurs .Il décrit les classes et leurs relations. Il peut aussi décrire leur regroupement en paquetages, interfaces et objets.

- Pour créer un diagramme de classes, nous devons définir les classes elles-mêmes, leurs responsabilités, leurs relations avec d'autres classes.

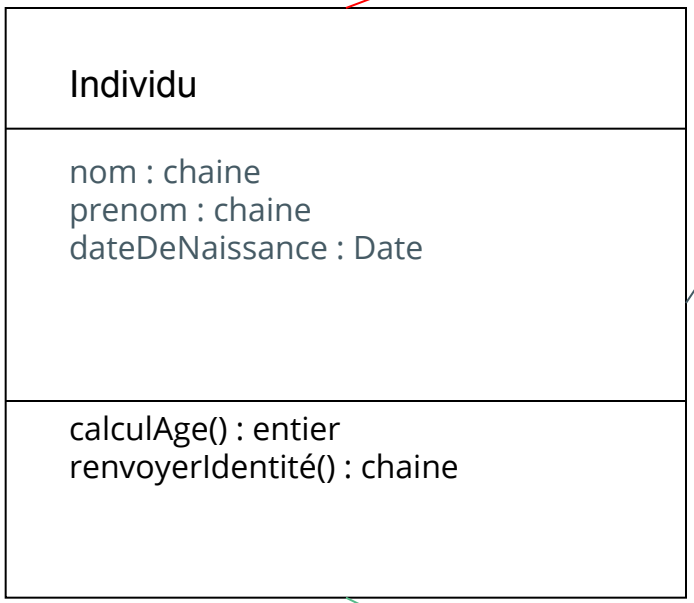
- Beaucoup d'objets informatiques ou réels se ressemblent tant au niveau de leur description que de leur comportement. On regroupe tous les objets qui se ressemblent en classes qui contiennent toutes les caractéristiques communes à ces objets, que ce soient les informations les caractérisant que les traitements applicables à ces objets

- Par exemple, tous les individus sur terre portent un nom, ont une date de naissance. L'âge de chaque individu est un calcul identique pour tous entre la date actuelle et la date de naissance.

UML – Le diagramme de classe

- Une classe est représentée par un rectangle qui contient plusieurs compartiments :
  - Le premier contient le nom de la classe. Celui-ci doit être sans ambigüité sur les objets qui feront partie de cette classe. C'est en général un substantif courant du métier traité
  - Le deuxième contient ses attributs
  - Le troisième contient ses méthodes, c'est-à-dire les fonctions applicables aux objets de la classe et qui permettent d'agir ou de recueillir des informations sur les objets de la classe
- Une classe est donc une description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun. Un objet est une instance de classe.
- Un objet d'une classe doit avoir des valeurs uniques pour chaque attribut de la classe et doit pouvoir exécuter toutes les méthodes de la classe.

Classe Individu



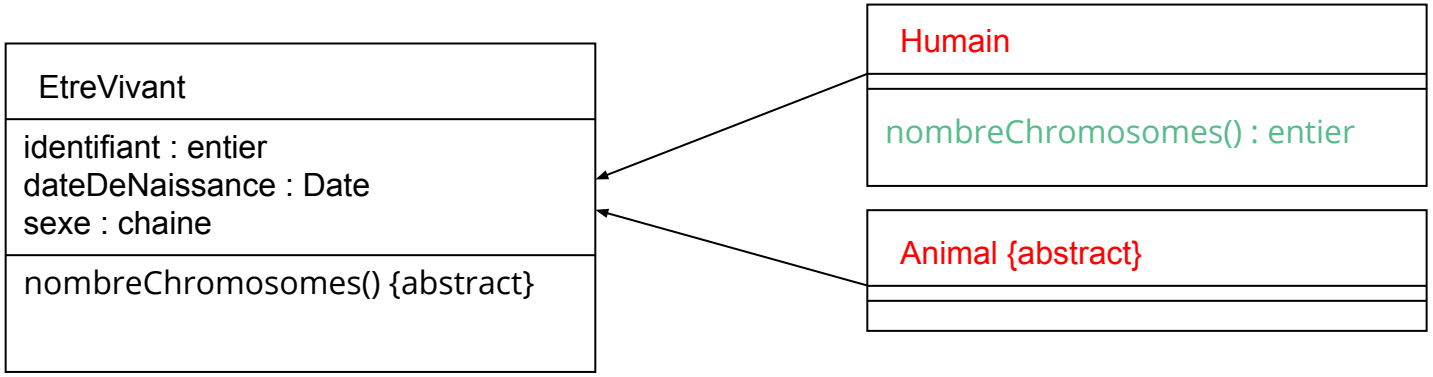
Le nom de la classe doit être significatif et complet.
Il commence par une majuscule. Si le nom est composé de plusieurs mots, chaque mot doit être différencié par une majuscule.

Les attributs sont présentés avec un nom et un type de donnée.
Ce type est , au sens UML, lui-même une classe.
Cependant, la plupart des langages de programmations permettent d'utiliser des types simples (int, float, ...).
Les noms des attributs sont en minuscules. Lorsque le nom est un mot composé, les mots suivants sont différenciés par une majuscule

Les méthodes sont présentées avec un nom et le type de donnée retourné
Ce type est , au sens UML, lui-même une classe.
Cependant, la plupart des langages de programmations permettent d'utiliser des types simples (int, float, ...).
Les noms des méthodes sont en minuscules. Lorsque le nom est un mot composé, les mots suivants sont différenciés par une majuscule

Classe abstraite

- Une classe est dite abstraite lorsqu'on connaît son entête mais pas la manière dont elle peut être réalisée.
- Une méthode est dite abstraite lorsqu'on est capable d'en définir le nom et la responsabilité, mais pas encore le contenu.
- 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.
- Exemple : un être vivant peut être modélisé dans une classe EtreVivant. Cet être vivant peut être un humain ou un animal. Le nombre de chromosomes de cet être vivant ne peut pas encore être connu car il dépend d'informations qui ne sont pas connues de la classe EtreVivant. Par contre, les classes Homme et Animal seront de façon certaine capables de fournir cette information.



L'implémentation de la fonction nombreChromosomes dans la classe Humain implique que cette classe n'est pas abstraite. Par contre, Animal est aussi une classe abstraite car cette méthode n'est pas implémentée.

Nom de classe

- Le nom de la classe doit être significatif du vocabulaire métier de l'entreprise.
- Par défaut une classe est considérée comme concrète. Si elle est abstraite, on ajoute le mot-clé **abstract**.
- Le nom de la classe commence par une majuscule. Quand le nom est composé, chaque mot commence par une majuscule et les espaces blancs sont éliminés.
- Il peut être complété d'informations permettant de mieux comprendre son contenu :
  - Le nom du designer
  - L'état de l'analyse (étude en cours, validé, ...)
  - Le nom du package dans lequel est intégré cette classe

Encapsulation

- Un principe de conception consiste à protéger le cœur d'un système des accès intempestifs venant de l'extérieur. C'est le principe du coffre-fort : seuls les détenteurs d'une clef peuvent l'ouvrir.

- Transposé à la modélisation objet, ce principe s'appelle encapsulation. Il permet de définir les droits d'accès aux propriétés d'un classeur. UML définit quatre niveaux d'encapsulation d'une propriété d'une classe.

- Une propriété peut être visible partout : dans ce cas elle est dite « publique ».

- Si une propriété n'est visible qu'à l'intérieur de sa propre classe, elle est dite « privée ».

- En utilisant le principe de l'héritage, les descendants d'une classe peuvent avoir un accès privilégié aux propriétés des classes parents. Une classe parent qui souhaite autoriser l'accès à l'une de ses propriétés à ses seuls descendants doit définir cette propriété comme « protégée ».

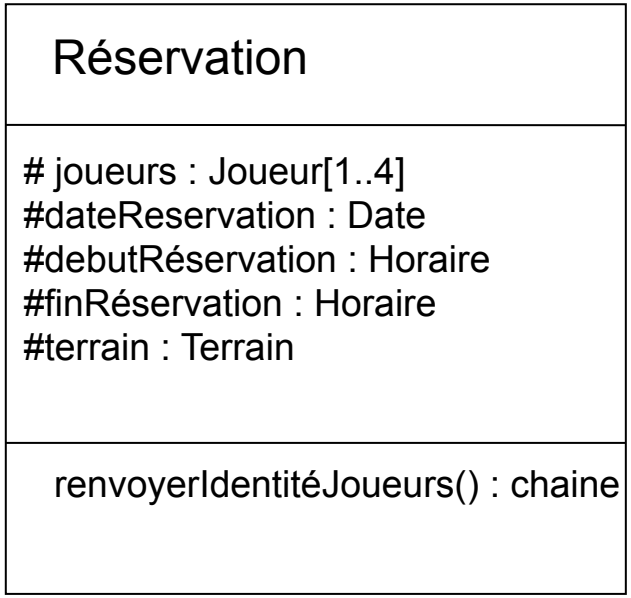
Encapsulation

- Les modificateurs d'accès ou de visibilité associés aux classes ou à leurs propriétés sont notés comme suit :
  - Le mot-clef **public** ou le caractère **+** : propriété ou classe visible de partout.
  - Aucun caractère ni mot-clef : propriété ou classe visible uniquement dans le paquetage où la classe est définie
  - Le mot-clef **protected** ou le caractère **#** : propriété ou classe visible dans la classe et ses descendants
  - Le mot-clef **private** ou le caractère **-** : propriété ou classe visible uniquement dans sa propre classe.

Attributs de la classe

Les attributs définissent les 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 et une visibilité. Le nom de l'attribut doit être unique dans la classe.

NB : En UML, les attributs correspondent à des instances de classes déjà définies. Pour des raisons de programmation, ce principe n'est pas toujours respecté par les langages de programmation objet, comme en témoigne l'existence des types de base (int, float, ...) dans les langages java, C++., C# ...



Attributs de la classe

- Un attribut peut être initialisé et sa visibilité est définie lors de sa déclaration
- La syntaxe de déclaration est la suivante

<modificateur d'accès> [/] <NomAttribut> : <NomClasse> ['[multiplicité']'] [= valeur(s) initiale(s)]

- Un attribut peut avoir une même valeur quelque soit l'objet instancié. Cette valeur est accessible même s'il n'existe aucun objet instancié de cette classe. Dans ce cas, on dit qu'ils 'agit d'un attribut de la classe, et non de ses instances. En UML, Ce type d'attribut est souligné. En C++ ou java, il est précédé du mot-clef static.
- Les attributs dérivés , symbolisés par l'ajout d'un slash (/) devant leur nom, peuvent être calculés à partir d'autres attributs et des formules de calcul. Lors de la conception, un attribut dérivé peut-être utilisé comme marqueur jusqu'à ce qu'on puisse déterminer les règles à lui appliquer. Dans les faits, il sera calculé par une méthode.

Méthodes de la classe

- Le comportement d'un objet est modélisé par un ensemble de méthodes, qui correspondent chacune à une implémentation précise d'un comportement.
- Lors de la conception des classes, on commence par donner les entêtes des méthodes sans préciser la manière de les implémenter (formules, algorithme, ...). Pour chaque entête, plusieurs implémentations seront possibles.
- En UML :
  - La définition de l'entête d'une méthode s'appelle opération.
  - L'implémentation est appelée méthode.
- Dans une classe, une opération (même nom et mêmes types de paramètres) doit être unique. Quand le nom d'une opération apparaît plusieurs fois avec des paramètres différents, on dit que l'opération (et/ou la méthode) est **surchargée**. En revanche, il est impossible que deux opérations ne se distinguent que par la valeur retournée.

Méthodes de la classe

La spécification d'une opération contient les types des paramètres et le type de la valeur de retour.

La syntaxe de la déclaration est la suivante :

<modificateur d'accès><nomDeLaMéthode ([paramètre])>:[<valeurRenvoyée>][{propriété}]

La syntaxe de la liste des paramètres est la suivante :

<NomClasse> nomVariable, <NomClasse> nomVariable, <NomClasse> nomVariable, ...

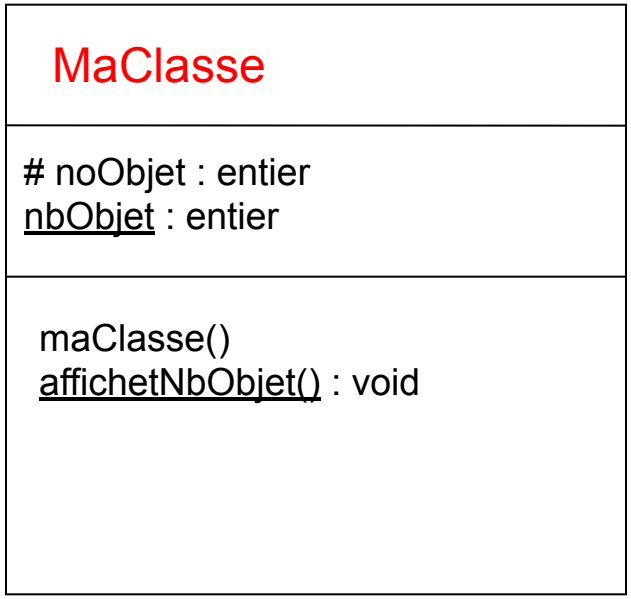
Les propriétés correspondent à des contraintes ou à des informations complémentaires comme les exceptions, les pré-conditions, les post-conditions.

La méthode qui permet d'initialiser un objet instance d'une classe est appelée **constructeur**

La méthode qui permet de détruire un objet est appelée **destructeur**

Opérations de la classe

- Comme pour les attributs, on peut créer des opérations propres à la classe et non à chaque objet de la classe. L'opération n'a pas accès aux attributs des objets.
- Cette méthode apparaît en souligné en UML



L'attribut noObjet est un attribut protégé, accessible aux instances de la classe et aux instances des classes héritées

L'attribut nbObjet est un attribut de classe, accessible uniquement sur la classe

La méthode maClasse est le constructeur de la classe. Elle est appelée à chaque création d'un nouvel objet instancié de la classe.

La méthode afficheNbObjet ne retourne rien (void) et affiche le nombre d'objets.

Compléments sur la classe

- En cours d'analyse, on peut compléter la description des classes par des informations complémentaires qui ne sont pas encore toujours traduisibles en paramètres ou méthodes. Elle le seront en fin d'analyse.

- Les informations complémentaires sont de deux natures :

  - Les responsabilités : description de l'ensemble des tâches devant être réalisées par la classe (contrôles, traitements, ...)

  - Les exceptions : description de tous les cas d'erreur et situations exceptionnelles traitées

Interface

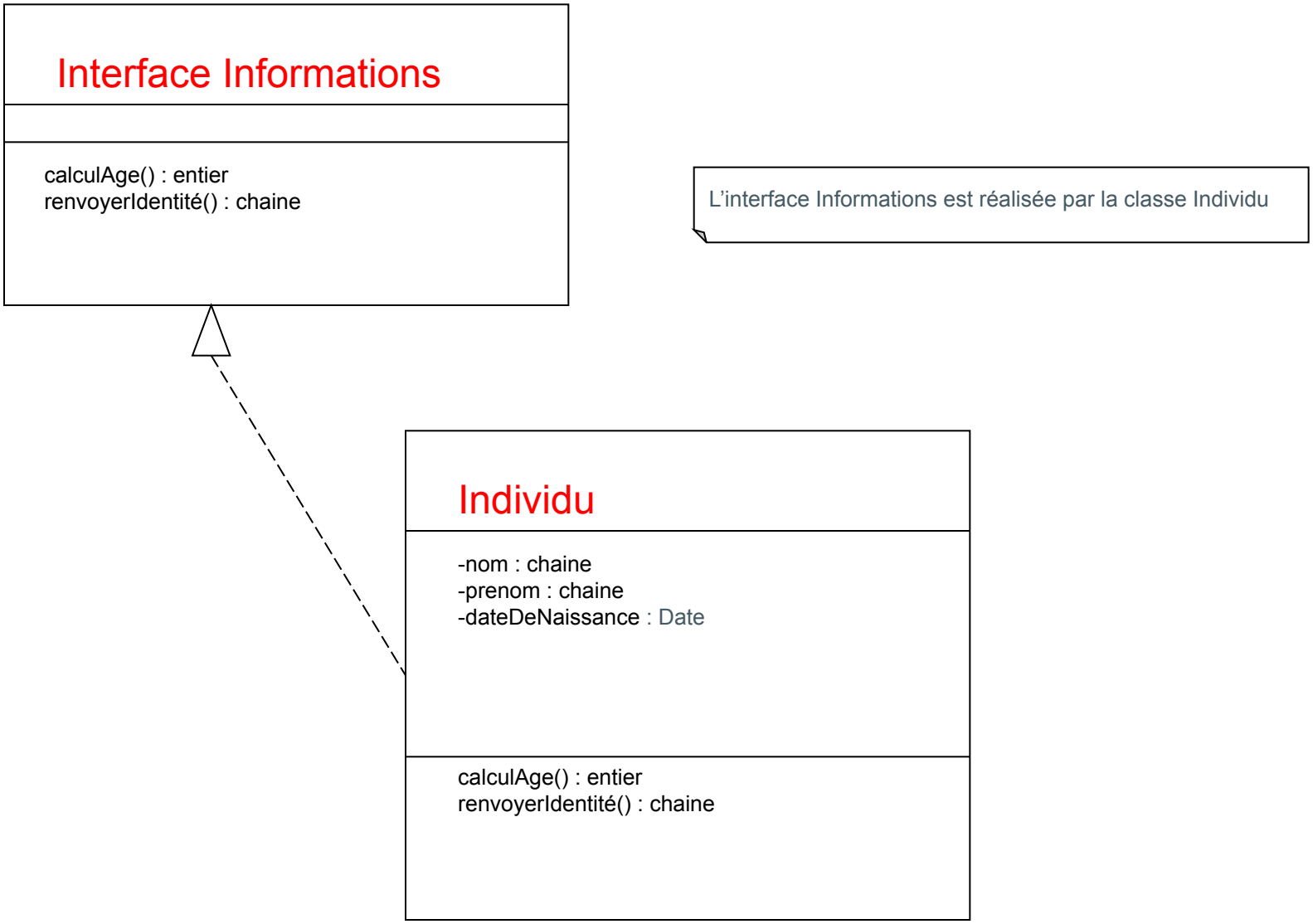
Le rôle d'une interface est de regrouper un ensemble d'opérations assurant un service cohérent offert par un classeur et une classe en particulier. Le concept d'interface est essentiellement utilisé pour classer les opérations en catégories sans préciser la manière dont elles seront implémentées.

La définition d'une interface ne spécifie pas une structure interne et ne précise pas les algorithmes permettant la réalisation de ses méthodes. Elle peut préciser les conditions et les effets de son invocation.

Une interface n'a pas d'instance directe. Pour être utilisée, elle doit être « réalisée » par une classe. Une interface peut être spécialisée et/ou généralisée par d'autres interfaces.

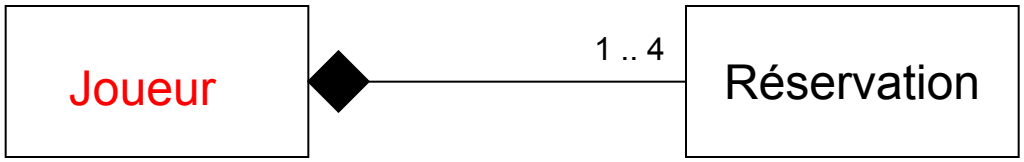
La réalisation d'une interface par une classe est représentée par un trait en pointillé terminé par une flèche vers l'interface L'interface est représentée comme une classe sans attribut, avec le prototype « interface » devant le nom de l'interface.

Interface


Interface Informations
calculAge() : entier
renvoyerIdentité() : chaine
L'interface Informations est réalisée par la classe Individu
Individu
-nom : chaine
-prenom : chaine
-dateDeNaissance : Date
calculAge() : entier
renvoyerIdentité() : chaine

Relations entre classes : multiplicité

La multiplicité indique à chaque extrémité d'une relation entre classes le nombre d'objets de la classe apparaissant à cette extrémité pouvant s'associer à un seul et unique objet de la classe apparaissant à l'autre extrémité.



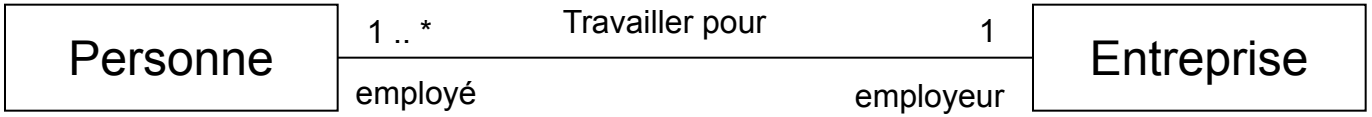
Un objet de la classe réservation est composé d'au moins un joueur et d'au plus 4 joueurs.
Les principales multiplicités normalisées sont:

- plusieurs : *
- Exactement n : n
- Au minimum n : n .. *
- Entre n et m : n..m

Relations entre classes : association

Association

- Une association représente une relation sémantique entre les objets.
- Elle est représentée par un trait plein + le nom de l'association + la multiplicité + le sens de lecture éventuel + le rôle de chaque classe dans l'association de chaque côté du lien



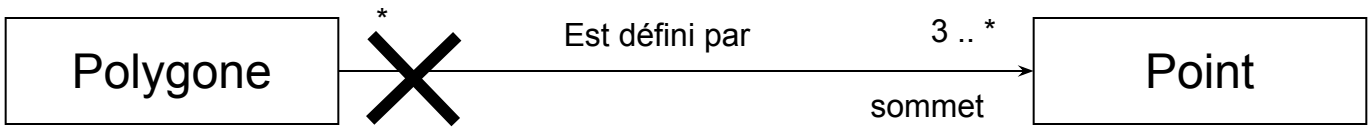
Une personne travaille pour une seule entreprise, l'entreprise emploie au moins une personne.

Relations entre classes

Association

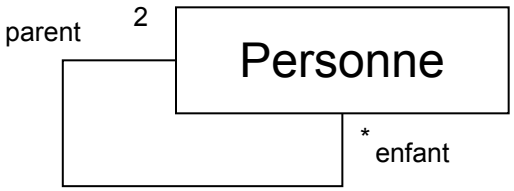
Une association peut aussi

- préciser le sens de lecture
- interdire la réciprocité en ajoutant une croix du coté ou la réciprocité est interdite

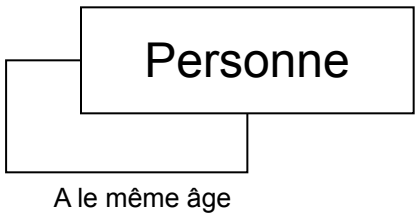


Relations entre classes : association

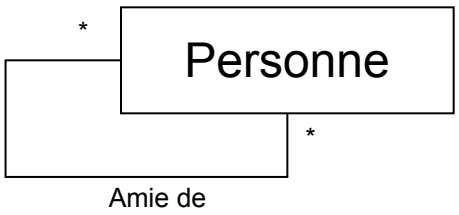
Associations réflexives : Une association peut exister entre instances de la même classe



Association asymétrique : chaque enfant a deux parents, chaque parent à un nombre indéterminé d'enfants



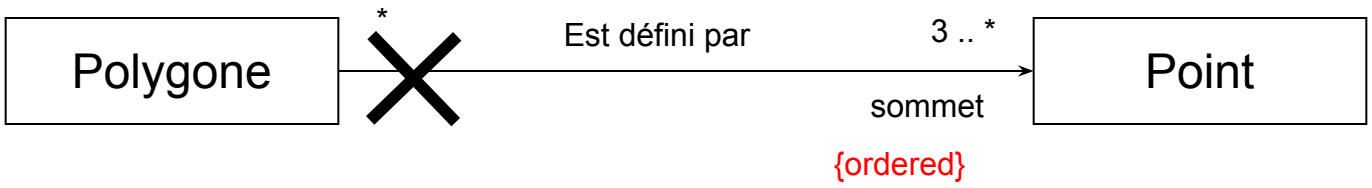
Association symétrique et transitive



Association symétrique et non transitive

Relations entre classes : association avec contraintes

Association avec contraintes : L'ajout de contraintes sur une association permet d'apporter des précisions à l'association. Les contraintes sont mises entre accolades et exprimées dans un langage au choix.



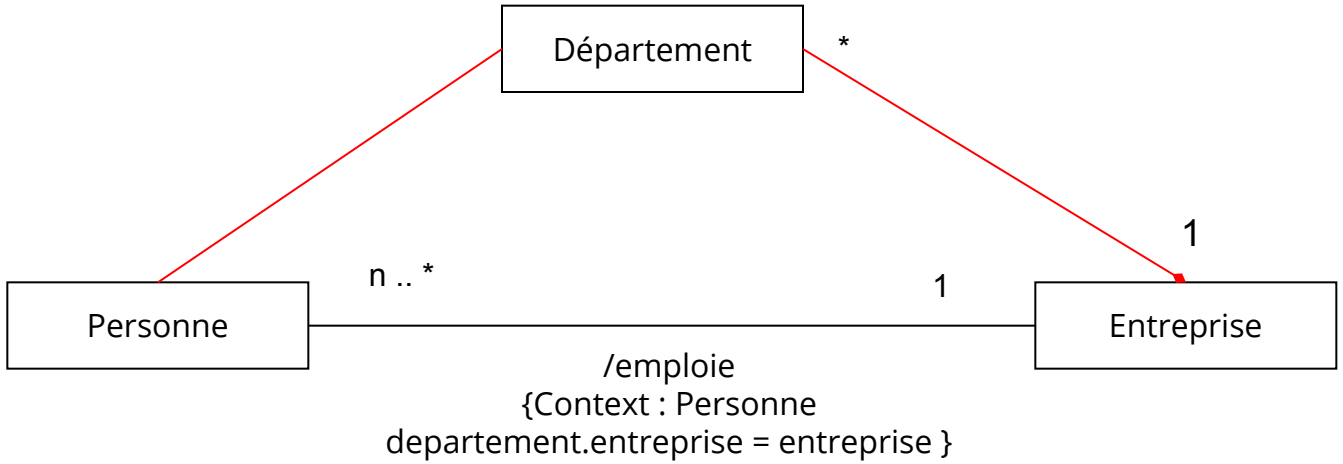
Ici, ordered indique que les sommets sont ordonnés

Les mots clefs peuvent suivre la norme OCL, ou bien être clairement explicités

Relations entre classes : association dérivée

Association dérivée

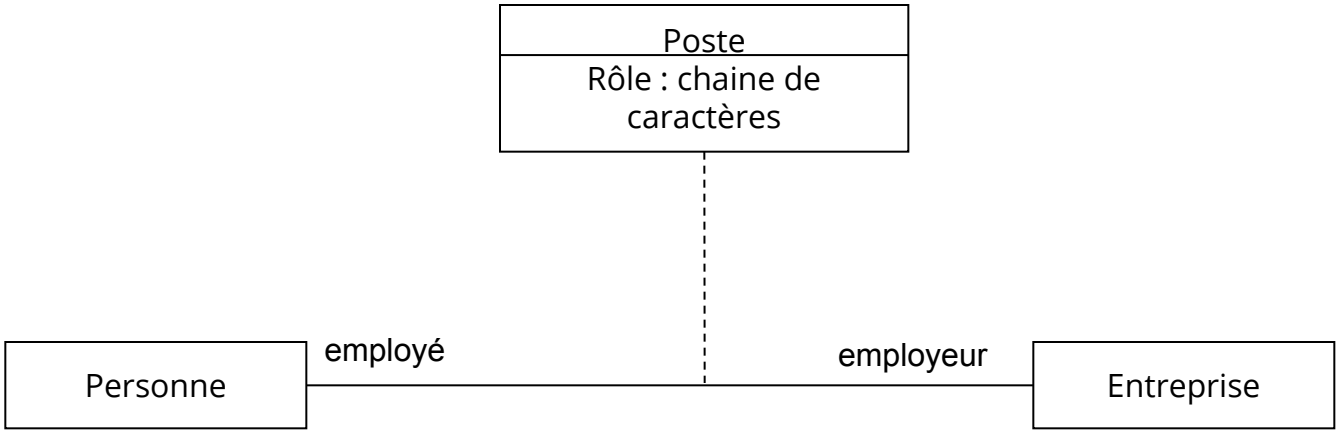
- Une association dérivée peut être déduite d'une autre association.
- L'ajout de contraintes permet de préciser les cas d'existence réelle de l'association déduite



Si une personne travaille dans un département, qui appartient à une entreprise, alors la personne travaille pour l'entreprise (caractère / devant emploie)

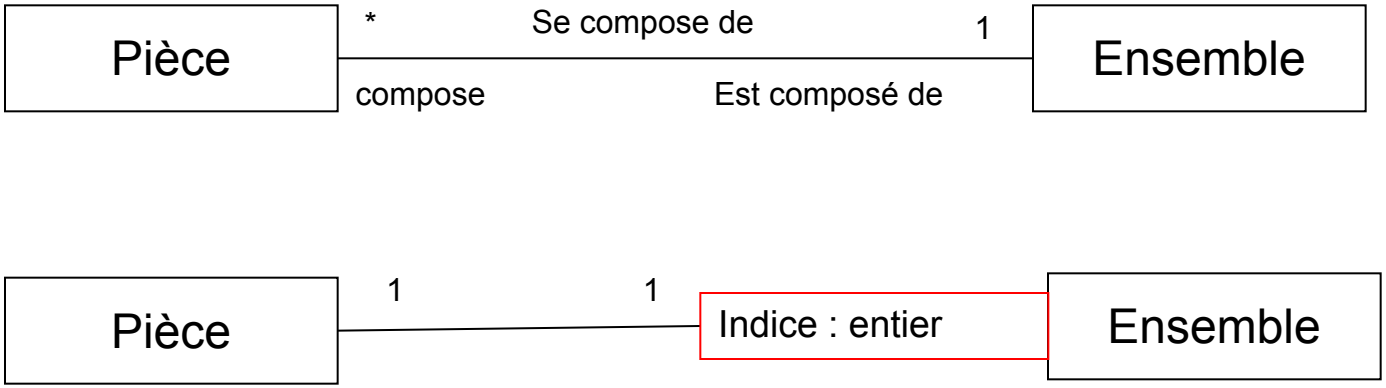
Relations entre classes : association

Classe Association : Une association peut être raffinée et avoir ses propres propriétés, qui ne sont disponibles dans aucune des classes qu'elle lie. Dans ce cas, elle devient une classe-association, et est considérée comme toutes les autres classes du modèle.



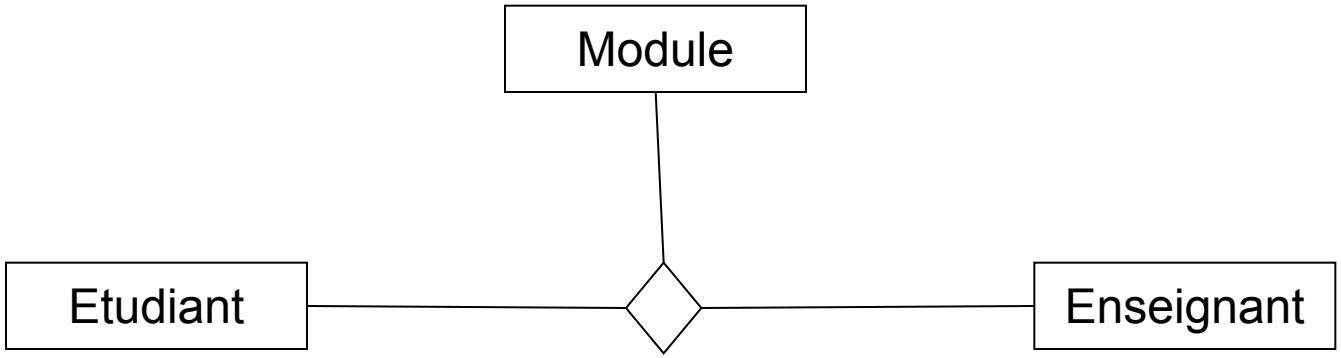
Relations entre classes /qualification

Association qualifiée : La qualification d'une association permet parfois de transformer une multiplicité infinie (*) en une multiplicité finie, en précisant le nombre réel max d'objets de la classe cible en relation avec la classe source



Relations entre classes

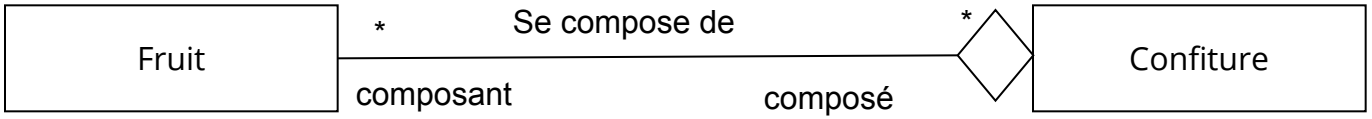
Association n-aire : Une association n-aire lie plus de deux classes. La gestion de ce type d'association est difficile, surtout lorsqu'il faut préciser les multiplicités. On leur préfère des associations binaires



Relations entre classes : Agrégation

Agrégation : Une agrégation est une forme particulière d'association. Elle représente la relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Contrairement à l'association, l'agrégation est une relation transitive. L'agrégation se distingue graphiquement de l'association par l'ajout d'un losange vide du côté de l'agrégat

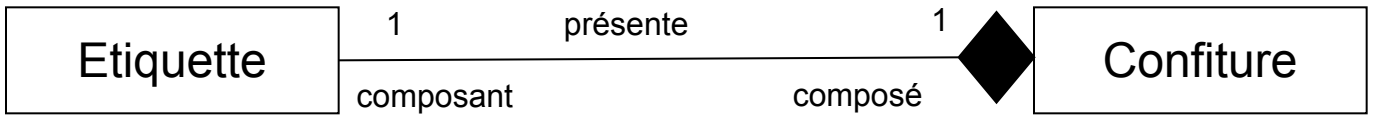
Un composant a sa vie propre. Il peut à un instant t appartenir à plusieurs agrégats



Relations entre classes : Composition

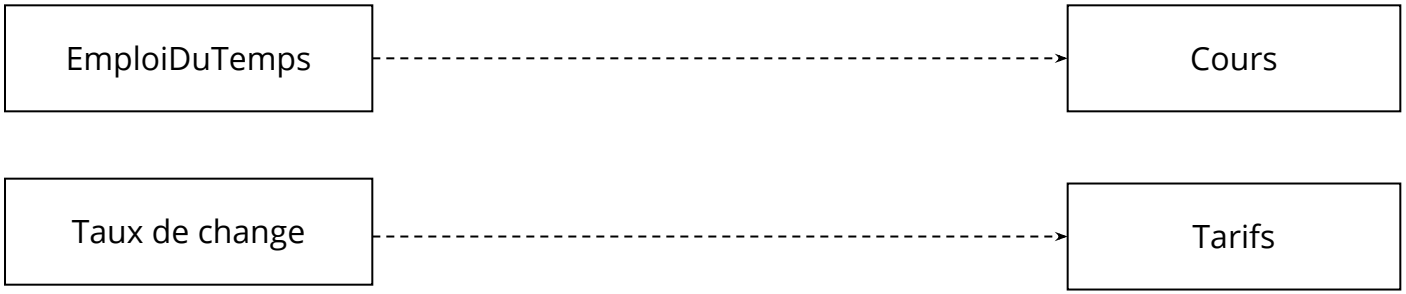
Composition

- La composition est une agrégation particulière : elle peut être remplacée par une agrégation, elle-même pouvant être remplacée par une association. La composition décrit une contenance structurelle entre instances. Ceci implique que l'élément composite est responsable de la création, de la copie et de la destruction de ses composants.
- Le composant ne peut pas appartenir à plusieurs composés en même temps.
- La copie ou la destruction de l'objet composite implique la destruction de ses composants. Une instance de partie appartient toujours à au plus une instance de l'élément composite. Graphiquement, on retrouve un losange plein côté agrégat. La multiplicité du côté de l'agrégat ne peut prendre que deux valeurs : 0 ou 1



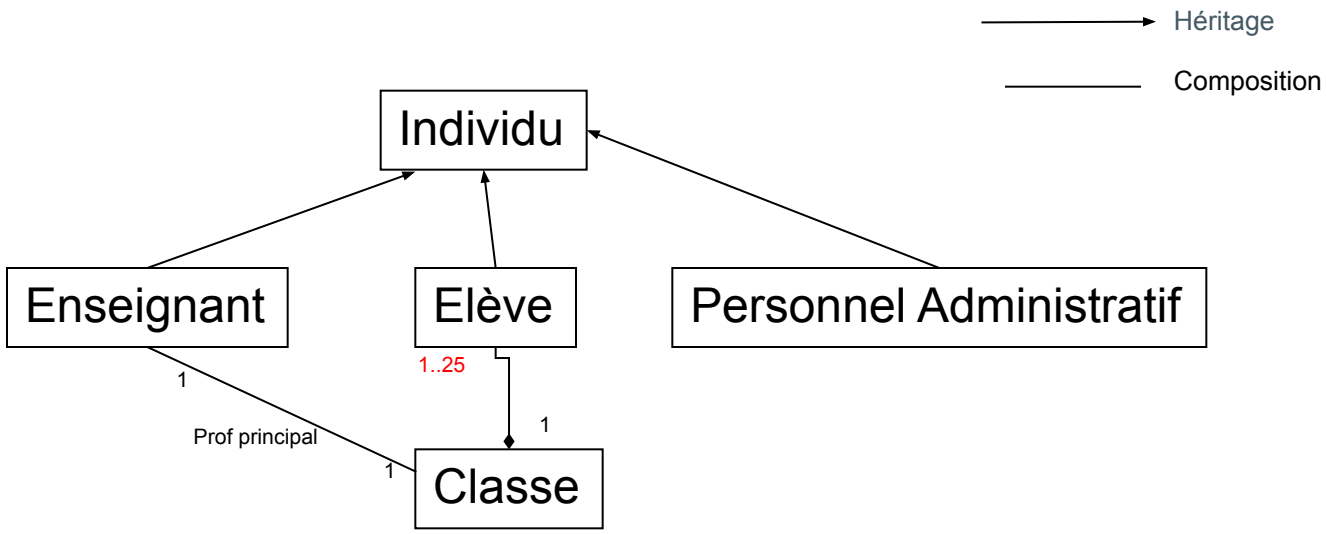
Relations entre classes : dépendance

Dépendance : Une relation de dépendance est une relation entre deux éléments dans laquelle la modification d'un des éléments (élément indépendant) peut affecter la sémantique de l'autre élément (élément dépendant)



Relations entre classes : héritage

Héritage : L'héritage est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe.
Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.
Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.
Une classe peut hériter de plusieurs classes différentes (ce n'est pas vrai dans tous les langages)



Relations entre classes

Héritage

- La spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple.
- L'héritage évite la duplication et encourage la réutilisation.
- Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.
- Le polymorphisme augmente la généricité du code.

Construction du diagramme

- UML ne donne pas de méthode pour construire le diagramme de classe, seulement des conventions de présentation.
- Pour construire le modèle de classe, on opère, comme pour un MCD, en partant du vocabulaire métier qui permet de mettre en évidence les entités identifiables du métier, à travers les concepts et les opérations du métier.
- Dans un premier jet, on peut décrire de nombreuses classes. Un seconde analyse permettra d' éliminer les redondances, de fédérer certaines classes en « super-classes » et en utilisant les notions d'héritage.
- Déterminer le niveau de granularité nécessaire et suffisant pour travailler facilement : chaque classe doit être utile pour mettre en œuvre plusieurs objets. Une classe qui ne verrait son utilisation qu'à travers une seule instance peut souvent être éliminée et transformée en un ensemble de variables de l'application, ou bien il s'agit d'un attribut d'une autre classe.