UML

5 - Diagrammes de classes

diagrammes d'objets - packages

Diagrammes de structure

Analyse des données et analyse organique

Bertrand LIAUDET

SOMMAIRE

Edition novembre 2017.

I - LE DIAGRAMME DE CLASSES

1 - Notions générales sur les classes et formalisme UML

Définition

Une classe est une description abstraite d'un ensemble d'objets qui partagent les mêmes propriétés (attributs et associations) et les mêmes comportements (mêmes opérations, c'est-à-dire les mêmes en-têtes de méthodes).

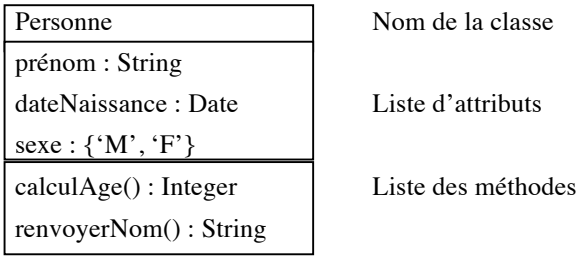
Instance

Un objet est une instance d'une classe.

L'instance est une notion générale :

- un lien est une instance d'une association ;

Représentation UML



Le nom de la classe doit être significatif. Il commence par une majuscule.

Le nom de la classe peut être préfixé par son ou ses paquetages d'appartenance.

Si personne est dans le paquetage A, lui même dans le paquetage B, on écrira

B ::A ::Personne

Encapsulation

L'occultation des détails de réalisation est appelée : encapsulation.

L'encapsulation présente un double avantage :

- Les données encapsulées dans les objets sont protégées.
- Les utilisateurs d'une abstraction ne dépendent pas de sa réalisation, mais seulement de sa spécification, ce qui réduit le couplage dans les modèles.

Le degré d'encapsulation peut être paramétré : c'est la notion de visibilité.

Visibilité des attributs et des méthodes

Par défaut, les attributs d'une classe sont « private » : les attributs d'un objet sont encapsulés dans l'objet. Les attributs ne sont manipulables que par les méthodes de l'objet.

Par défaut, les méthodes d'une classe sont « public » : les méthodes d'un objet sont accessibles par tous les « clients » de l'objet.

UML définit quatre niveaux de visibilité

La visibilité des attributs et des méthodes est précisée par des mot-clés ou des symboles.

Symbole	Mot-clé	Signification
+	Public	Visible **partout**
		Visible **dans tout le paquetage** où la classe est définie
#	Protected	Visible dans la classe, **dans ses sous-classes** et par les amis.
-	Private	Visible uniquement **dans la classe** et par les amis.

Attribut de Classe

Certains attributs ont une valeur identique pour tous les objets de la classe. Ce sont des sortes de constantes définies au niveau de la classe. Ce sont les attributs de classe.

En UML, ces attributs sont listés avec les autres, mais ils sont soulignés.

Symbole	Mot-clé	Signification
Souligné		Valeur identique pour tous les objets de la classe

Attribut dérivé

Certains attributs peuvent être calculés à partir d'autres attributs de la classe.

En UML, ces attributs sont listés avec les autres, mais ils sont précédés d'un « / ».

Symbole	Mot-clé	Signification
/		Attribut dont la valeur est calculée à partir de celle d'autres attributs

Méthode de Classe

Quand une méthode ne porte pas sur les attributs d'objet de la classe mais uniquement sur des attributs de classe ou sur des valeurs constantes, cette méthode est dite méthode de classe.

Une méthode de classe peut être utilisée sans avoir à instancier d'objet. On peut se contenter d'un objet déclaré pour y accéder.

En UML, ces méthodes sont listées avec les autres, mais elles sont <u>soulignées</u>.

On y accède en préfixant le nom de la méthode par le nom de la classe.

Compartiment des responsabilités

Le compartiment des « responsabilités » liste l'ensemble des tâches que la classe doit réaliser. C'est une façon de lister les méthodes dont on n'a pas encore défini l'entête.

Quand la conception sera achevée, ce compartiment devra disparaître.

Compartiment des exceptions

Le compartiment des « exceptions » liste les situations exceptionnelles devant être gérées par la classe.

Là encore, quand la conception sera achevée, ce compartiment devra disparaître.

2 - Associations entre classes

Comme dans le modèle entité-association (MEA, MCD MERISE), UML permet d'établir des relations (des associations) entre les classes.

Il y a trois types d'associations entre classes :

- Les associations simples
- Les associations d'agrégation de composition
- Les associations d'héritage

Les deux premières sont de même nature et expriment des relations entre les objets de deux classes (ou plus).

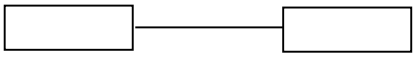
L'association d'héritage permettent de définir des sous-ensembles dans un ensemble ou inversement de définir un ensemble à partir de sous-ensembles. Elle exprime une relation entre les classes et pas entre les objets.

3 - Les associations simples

Association

Une association représente une relation structurelle entre des classes d'objets.

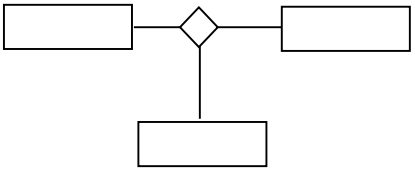
Une ligne entre deux classes représente une association.



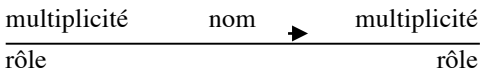
Association binaire (arité = 2)

La plupart des associations sont binaires : elles ne réunissent que deux classes.

Association ternaire et plus (arité >2)

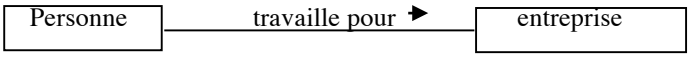


L'association et ses différents ornements

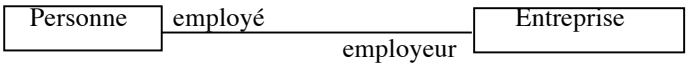


Nom et sens de lecture des associations

L'association peut avoir un nom. Le nom explicite le lien entre les deux classes. C'est souvent un verbe. Le sens de lecture ▶ permet de préciser dans quel sens il faut lire le nom.



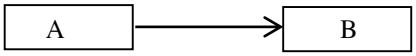
Rôles des extrémités des associations



Le rôle est un pseudo-attribut. On peut préciser sa visibilité (+, -, #)

La plupart du temps, le nommage des rôles est limité en fonction de la navigabilité.

Navigabilité des associations



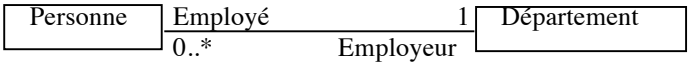
L'association n'est navigable que dans le sens de A vers B.

Une association navigable uniquement dans un sens peut être vue comme une demi-association.

Multiplicité (cardinalité) des associations

Chaque extrémité de l'association peut porter une indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe.

1	Un et un seul
0..1	Zéro ou un
N	N (entier naturel qui peut être précisé)
M .. N	De M à N (entiers naturels qui peuvent être précisés)
*	De zéro à plusieurs
0..*	De zéro à plusieurs
1..*	De un à plusieurs

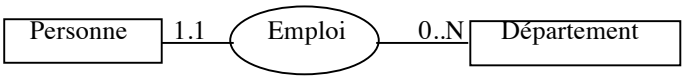


Chaque personne est employée dans un département et un seul.

Les départements sont employeur de 0 ou plusieurs personnes.

Attention :

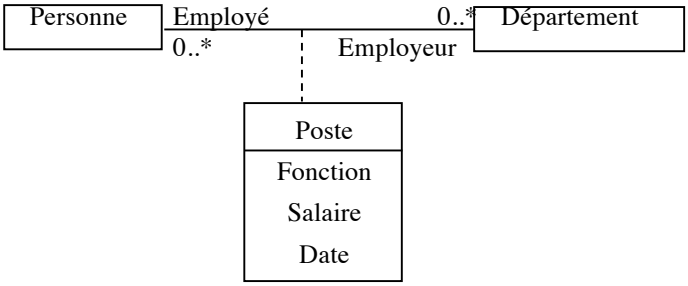
Les cardinalités sont mises à l'inverse du modèle entité-association (MEA) :



Classe-Association

Une association peut avoir ses propriétés qui ne sont disponibles dans aucune des classes qu'elle relie.

Pour cela, on relie une classe à une association :



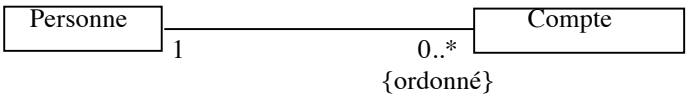
La classe-association correspond à l'association non-hiérarchique avec attributs du MEA .

Contraintes sur les associations

Toutes sortes de contraintes peuvent être définies sur les associations.

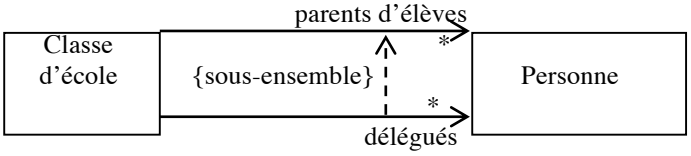
Certaines contraintes s'appliquent à une seule association :

- {ordonné} : précise qu'une collection (0..*) doit être ordonnée.



Certaines contraintes s'appliquent à plusieurs associations :

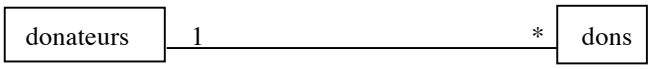
- {sous ensemble} : précise qu'une collection est incluse dans une autre collection
- {ou exclusif} : précise pour un objet donné qu'une association et une seule est possible parmi les associations contraintes par le ou exclusif.



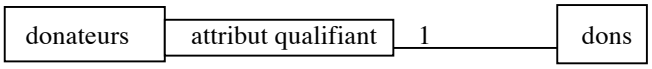
Qualification des associations (restriction)

La qualification des associations consiste à réduire le nombre d'occurrences d'une association.

Elle ne s'applique qu'à des associations dont la multiplicité est supérieure à 1 (sinon, le nombre d'occurrences ne peut pas être réduit).



Dans ce cas, la qualification ne peut s'appliquer qu'aux donateurs (un donateur peut faire plusieurs dons).

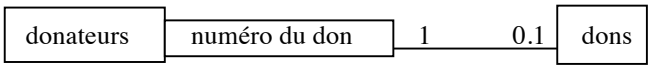


L'attribut qualifiant peut être extrait de la classe des dons.

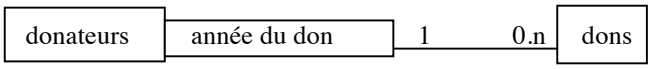
La qualification restreint la relation aux seuls couples concernés par un qualifieur.

La qualification des associations est une forme d'association contrainte.

Exemples



La classe « donateurs » contient un objet « dons », instancié ou pas, et filtré par numéro de don.

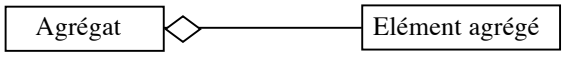


La classe « donateurs » contient une collection d'objets « dons » filtrés par année de don.

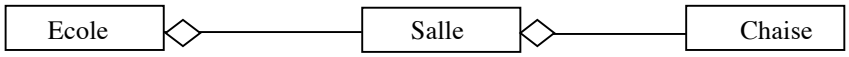
4 - Agrégation et composition

Agrégation

L'agrégation est une association non symétrique dans laquelle une des extrémités, l'agrégat, joue un rôle prédominant par rapport à l'autre extrémité, l'élément agrégé.



L'agrégation représente en général une relation d'inclusion structurelle ou comportementale.



Une école est composée de plusieurs salles, qui elles-mêmes sont composées de plusieurs fenêtres et plusieurs chaises.

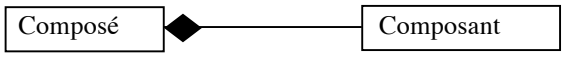
En général, les agrégations sont navigables uniquement dans le sens Agrégat vers Eléments agrégés.

En général, les agrégations sont des associations 1..*.

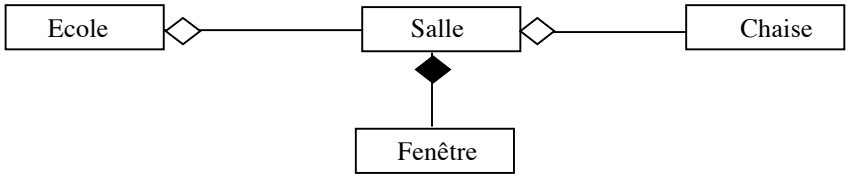
En général, quand on ne précise pas la cardinalité et la navigabilité, on sous-entend une cardinalité 1..* et une navigabilité limitée au sens Agrégat vers Agrégés.

Composition

La composition est un cas particulier d'agrégation.



La composition implique une coïncidence des durées de vie des composants et du composite : la destruction du composite implique la destruction de tous ses composants.



Si on supprime la salle, les fenêtres sont aussi supprimées. Ce qui n'est pas le cas des chaises.

De ce fait, un composant n'est pas partageable et la cardinalité est obligatoirement 1 (ou 0..1) du côté du composé.

En général, les compositions sont des associations 1..*

En général, les compositions sont navigables uniquement dans le sens Composé vers Composants.

En général, quand on ne précise pas la cardinalité et la navigabilité, on sous-entend une cardinalité 1..* et une navigabilité limitée au sens Composé vers Composants.

Remarques sémantiques

La notion de « rôle prédominant » est assez subjective.

Elle se traduit en général par la navigabilité et la cardinalité par défaut.

Souvent, ce sont les classes « control » (classes de gestion) qui porteront le plus d'agrégations.

La notion de composition a une valeur sémantique plus forte que celle d'agrégation du fait que la destruction du composant implique celle du composé.

Remarques techniques

Techniquement, si le langage gère un « ramasse-miettes » (garbage collector), tout est composition !

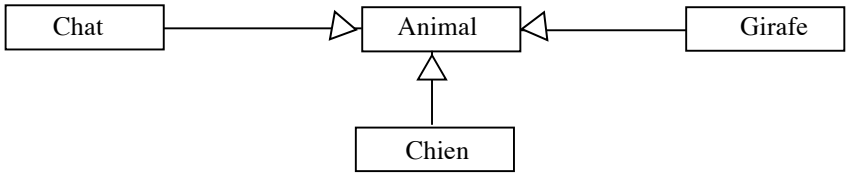
Quand on supprime un objet, les attributs sont supprimés. Si l'attribut est un autre objet ou une collection d'objets, ce sont des références vers l'objet attribut. Si l'objet attribut n'est plus référencé par personne, il sera détruit par le « ramasse-miettes ». Si l'objet attribut est référencé par une autre référence, il ne sera pas détruit par le « ramasse-miettes ».

5 - Généralisation

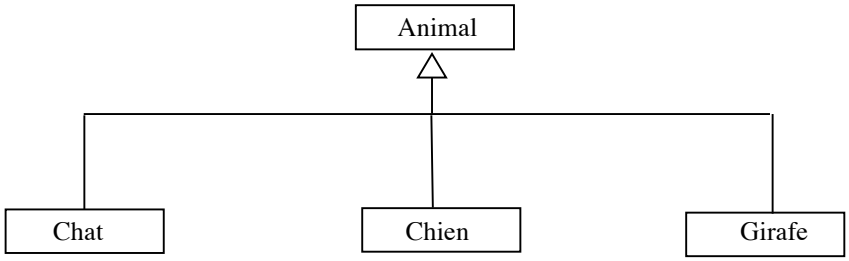
Définition

Le terme généralisation désigne une relation de classification entre un ensemble plus général (le genre) et un ensemble plus spécifique (l'espèce). La classe Chat est une espèce de la classe Animal.

On parle de classe spécifique, ou sous-classe, ou **classe enfant** et de classe générale, ou sur-classe, ou super classe, ou **classe parent**, ou classe mère.



Ou encore :



Relation « est un » ou « is a »

La généralisation correspond à une relation « est un » : un chat est un animal.

Toute relation « est un » peut être transformée en généralisation.

Principe : rendre toute similitude explicite à l'aide de l'héritage

L'héritage consiste à :

- Choisir les classes.
- Fournir un ensemble complet d'opérations pour chaque classe.
- **Rendre toute similitude explicite à l'aide de l'héritage**

Distinction entre généralisation et composition

Un chat est un animal : c'est une généralisation. La classe chat est une partie de la classe animal. Chaque objet « chat-x » appartient à la classe « chat » et à la classe « animal ».

Un chat a deux oreilles : c'est une composition. L'objet « oreille » est une partie de l'objet « chat ». Pour tout objet « chat », il existe deux objets « oreille ».

Héritage

La classe spécifique contient des attributs et des méthodes qui lui sont propres.

- **La classe spécifique hérite de tous les attributs, méthodes et associations de la classe générale**, sauf pour ceux qui sont privées. L'attribut « protected », symbole « # », permet de limiter la visibilité aux enfants.
- **Une classe spécifique peut redéfinir une ou plusieurs méthodes**. C'est le principe de la surcharge des opérations. Un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes.
- **Un objet spécifique peut être utilisé partout où un objet général est attendu** : partout où on attend un animal, on peut utiliser un chat.

Généralisation multiple

Une sous-classe peut avoir plusieurs sur-classes

Cependant, c'est à éviter, puisque c'est interdit en Java par exemple.

Par contre, une classe peut avoir plusieurs interfaces.

6 - Les relations de dépendance

Présentation

Les relations de dépendances sont utilisées quand il existe une relation sémantique entre plusieurs éléments qui n'est pas de nature structurelle (association, composition, agrégation ou héritage.

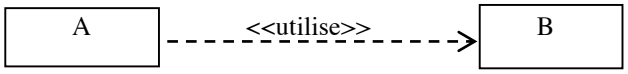
Type de dépendance

Type de dépendance	Stéréotype	Signification
Permission	<<Ami>>	La source a accès à la destination, quelle que soit la visibilité.
Abstration	**<<Réalise>>** <<Raffine>> <<Trace>>	Un même concept à des niveaux d'abstraction différents. Réalisation d'une spécification (opération). Hiérarchie sémantique (l'analyse raffine la conception) Historique des constructions de différents modèles.
Utilisation	**<<Utilise>>** <<Appelle>> <<Crée>> <<Instancie>>	La source requiert la cible pour son bon fonctionnement. Une opération de la source invoque une opération de la cible. La source crée une instance de la cible. Idem que <<Crée>>.
Liaison	<<Lie>> <<Dérive>>	Liaison entre une classe paramétrée et une classe paramétrable Elément calculé à partir d'autres.

Principales dépendances

<< Utilise >> et << Réalise >> sont les principales dépendances.

Représentation UML



Remarque sur la dépendance d'utilisation

Dans la déclaration d'une classe, on peut faire référence à une autre classe de 3 manières :

- En déclarant un attribut
- En déclarant un paramètre formel
- En déclarant une variable locale

Dans chacun de ces cas, la classe d'origine utilise la classe à laquelle elle fait référence pour une déclaration.

Il y a toujours dépendance quand il y a navigabilité.

Le principe général de la modélisation est de minimiser au maximum des dépendances, quelles qu'elles soient.

7 - Bilan des relations entre les classes : composition, utilisation, héritage

Présentation

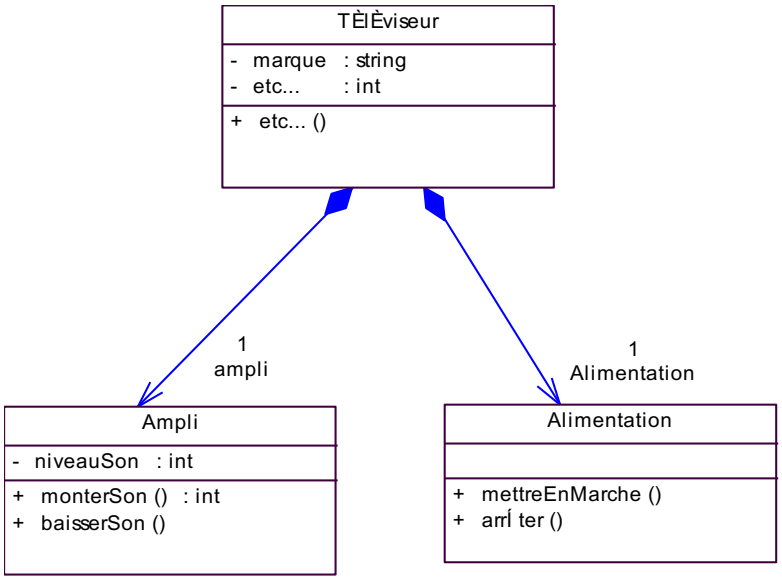
Finalement, dans le code, il y a 3 relations entres les classes : la composition, l'utilisation et l'héritage.

Ces relations génèrent des dépendances entre les objets.

La composition

Un objet 2 est composant d'un objet 1 si c'est un **attribut** de l'objet 1, autrement dit si un attribut de la classe 1 est de type classe 2.

La relation de composition génère une **dépendance** : la classe 1 dépend de la classe 2, donc l'objet 1 dépend de l'objet 2.

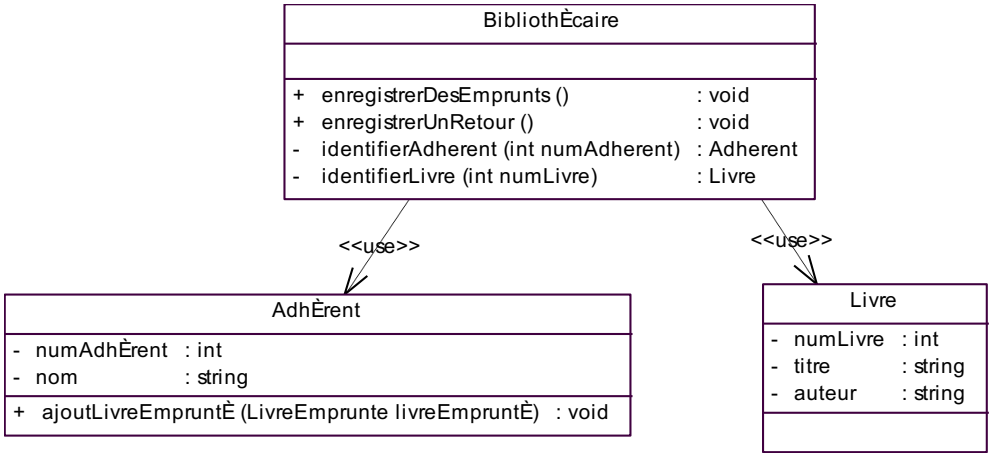


L'utilisation (ou dépendance)

La classe 2 est utilisée par la classe 1 si :

- un objet de la classe 2 est **déclaré localement** dans une opération de la classe 1
- un objet de la classe 2 **passé en paramètre formel** d'une opération de la classe 1

La relation de composition génère une **dépendance** : la classe 1 dépend de la classe 2, donc l'objet 1 dépend de l'objet 2.



L'héritage

Principes

Une classe peut hériter des attributs et des opérations d'une autre classe. La classe héritière est appelée **sous-classe**, la classe dont elle hérite est appelée **classe parente**. Une classe parente peut à son tour hériter. Si une classe a une seule classe parente, on parle d'**héritage simple** sinon d'**héritage multiple**.

Héritage d'attributs : l'ensemble des **attributs d'un objet** d'une classe est constitué de l'ensemble de ses attributs et de l'ensemble des attributs de ses classes parentes.

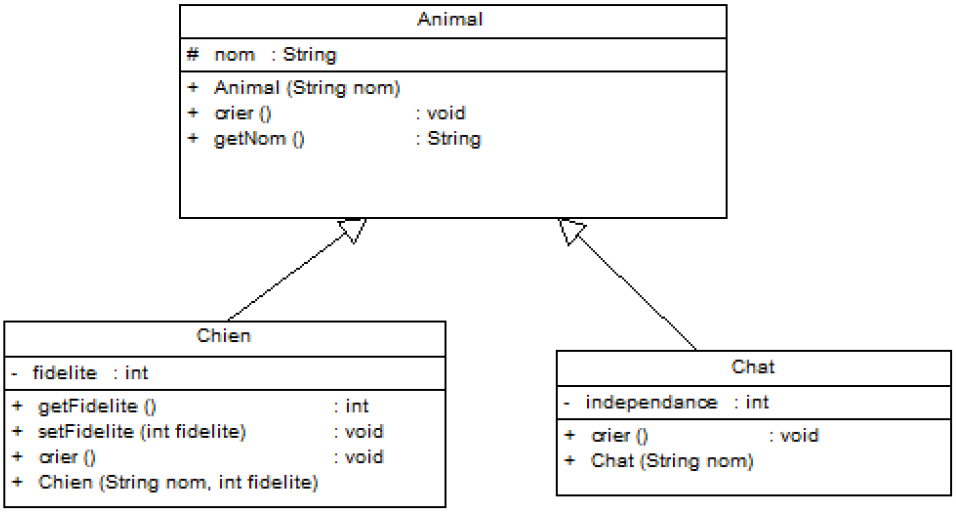
En ce sens, l'héritage est une **façon d'organiser les concepts** dans un rapport d'espèce à genre : l'espèce (sous-classe) est une sorte du genre (classe parente).

Héritage d'opérations : l'ensemble des **opérations applicables à un objet** est constitué de l'ensemble de ses opérations et de l'ensemble des opérations de ses classes parentes.

En ce sens l'héritage est une **technique de partage et de réutilisation du code existant**.

Polymorphisme : la manière d'associer du code à un message est un mécanisme dynamique qui n'est pas établi à la compilation. L'envoi d'un message ne correspond pas à un débranchement vers une adresse de code prédéfinie comme c'est le cas en programmation procédurale classique. Le message choisi dépend de l'objet émetteur concrètement instancié.

Formalisme UML



Polymorphisme

Si une méthode reçoit un animal en paramètre, et qu'elle envoie le message crier(), le crier() envoyé dépendra de l'animal concret passé en paramètre. Le comportement sera donc polymorphe.

Les 3 relations entre les objets

En étudiant les relations entre classes, on vient de voir les 3 relations entre objets :

- un objet peut être un attribut d'un autre objet (composition)
- un objet peut être un paramètre formel d'une méthode d'un autre objet (dépendance)
- un objet peut être une variable locale d'une méthode d'un autre objet (dépendance)

8 - Classes et méthodes abstraites

Définition

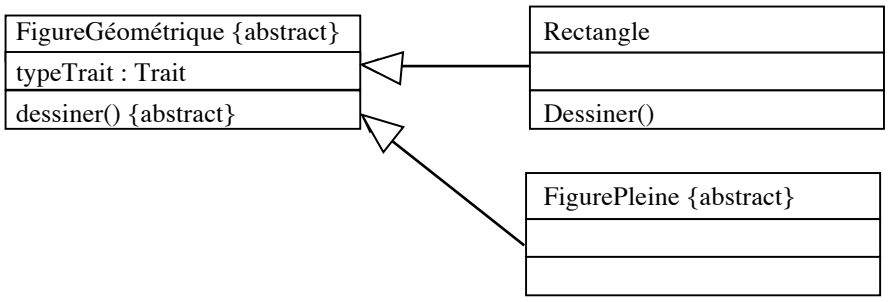
Une classe abstraite est une classe qui ne s'instancie pas directement, par opposition à une classe concrète : c'est une classe qui ne contient pas d'objets.

Les classes abstraites sont en général des super-classes qui contiennent des sous-classes qui, elles, contiendront des objets.

Ces classes abstraites servent surtout pour la classification et le polymorphisme.

En phase de modélisation, une classe est aussi dite abstraite quand elle, ou une de ses classes parents, définit au moins une méthode abstraite (et donc qu'elle ne peut pas, au moins provisoirement, être instanciée).

Formalisme



La classe « figure géométrique » est abstraite parce quelle contient une méthode abstraite, c'est-à-dire une méthode dont on connaît l'entête mais pas le corps.

La classe « Figure pleine » est abstraite parce qu'elle hérite de la méthode abstraite « dessiner » qu'elle ne redéfinit pas comme la classe « Rectangle ».

Méthode et opération

La spécification d'une méthode correspond à son en-tête : on l'appelle aussi : **l'opération**.

L'implémentation de la méthode (l'algorithme), c'est ce qu'on appelle la méthode.

Méthode abstraite

Une méthode est dite abstraite quand on connaît son en-tête mais pas la manière dont elle peut être réalisée.

Utilisation

Les classes et les méthodes abstraites sont souvent associées au polymorphisme.

9 - Les classes « interface » ou « boundary »

Présentation

Une classe « interface » décrit le comportement visible d'une classe autrement dit la communication entre la classe et son environnement, que ce soit un utilisateur ou une autre classe.

Les classes interfaces modélisent les interfaces du système.

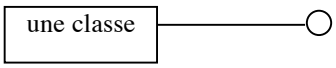
L'interface est la partie visible d'une classe ou d'un package. Elle est parfois synonyme de spécifications, ou vue externe, ou vue publique.

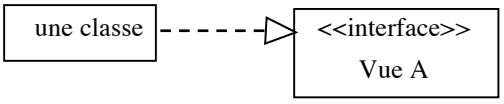
Caractéristiques techniques

Une interface est une classe avec le stéréotype « interface ».

- C'est une classe abstraite qui ne contient aucun attribut et ne contient que des opérations (autrement dit des méthodes abstraites) ayant une visibilité « public ».
- Ces opérations seront réalisées par les classes qui réalisent l'interface.
- Une classe qui réalise une interface réalise toute les opérations de l'interface.
- Une classe peut réaliser plusieurs interfaces.
- Une interface peut être réalisée par plusieurs classes.

Représentation UML des interfaces



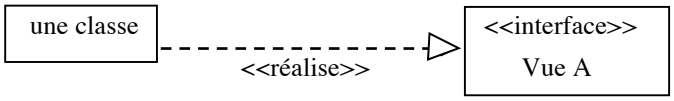


Le symbole ——○ montre l'existence d'une interface, mais ne précise pas les opérations.

La flèche - - - - -▷ signifie que la source réalise la destination. La flèche triangulaire est de type « héritage ».

<<interface>> est le nom d'un stéréotype

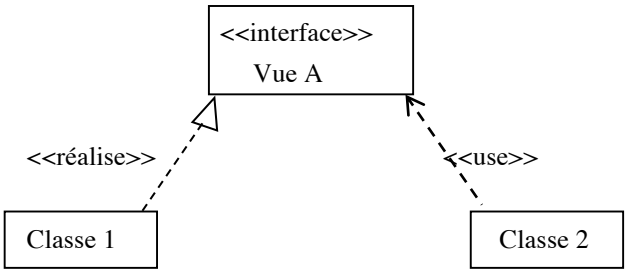
On peut préciser le stéréotype du lien : « réalise » (ce n'est pas obligé).



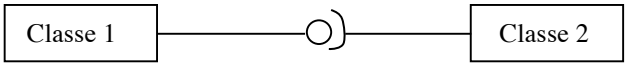
Utilisation des interfaces

Une classe peut utiliser tout ou partie des opérations proposées par une interface. Cette classe est alors dépendante de l'interface.

La relation d'utilisation <<use>> signifie que la source requiert la cible pour son bon fonctionnement.



ou encore :



Méthode de d'analyse des interfaces

Pour trouver les interfaces du système, il faut **examiner toutes les paires : acteurs physiques – scénario.** On a intérêt à créer dans un premier temps au moins une classe interface par cas d'utilisation.

Les interfaces se trouvent à un haut niveau d'abstraction. On commence par renseigner les besoins en interfaces utilisateur sans les implémenter. Ces classes seront affinées au fur et à mesure.

On a intérêt à regrouper les interfaces dans un paquetage à part.

En utilisant des interfaces plutôt que des classes, on sépare les traitements de leur interface. Ainsi, on facilité l'évolution des applications et leur adaptations à différents environnement d'interface.

10 - Quelques classes particulières

Les classes de variables et d'opérations globales : stéréotype <<utility>>

<<utility>> Math
PI :Real=3,14
Sinus (angle): Real Cosinus(angle): Real Tangente(Angle):Real

Les énumérations : stéréotype << énumération >>

Un type énuméré peut se traduire par une classe UML (toute classe est un type).

On utilise le stéréotype <<énumération>> et on liste les valeurs de l'énumération dans la classe.

<< énumération >> Couleurs
rouge vert bleu jaune

Les classes actives

Par défaut, les classes sont passives.

Une classe active possède son propre flot d'exécution. Les classes actives peuvent être des <<processus>> ou des <<thread >>.

Un **processus** est un flot de contrôle lourd qui s'exécute dans un espace d'adressage indépendant.

Un **thread** est un flot de contrôle léger qui s'exécute à l'intérieur d'un processus.

Une classe active choisit un ou plusieurs flots de contrôle.

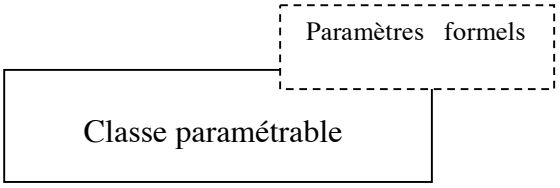
Représentation UML

Classe et objet actif sont représentés par un **cadre plus épais** ou avec un trait double.

@Les classes paramétrables : template (généricité)

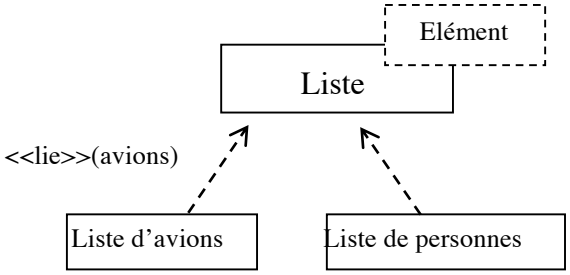
Une classe paramétrable est un modèle de classe. Elle traduit la notion de généricité c'est-à-dire de type variable.

Une classe paramétrable a des paramètres formels. Chaque paramètre possède un nom, un type et une valeur par défaut optionnelle.



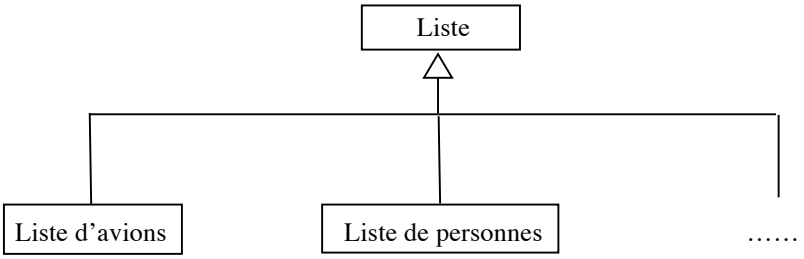
Une classe paramétrable est liée à des paramètres effectifs pour devenir une classe paramétrée qui pourra être instanciée.

Exemple



Ce type de classe apparaît rarement au début de la modélisation.

A noter que les classes paramétrables peuvent être remplacées, de façon moins élégante, par des généralisations :

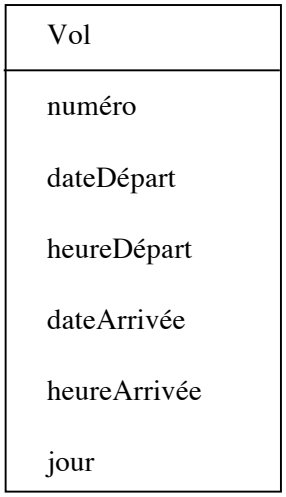


11 - Amélioration de l'analyse des classes : notion de métaclasse

Une métaclasse est une classe (2) d'une classe (1). La métaclasse est telle qu'un objet de cette classe contient des données et des opérations qui caractérisent plusieurs objets de la classe dont elle est issue : classe (1).

Exemple :

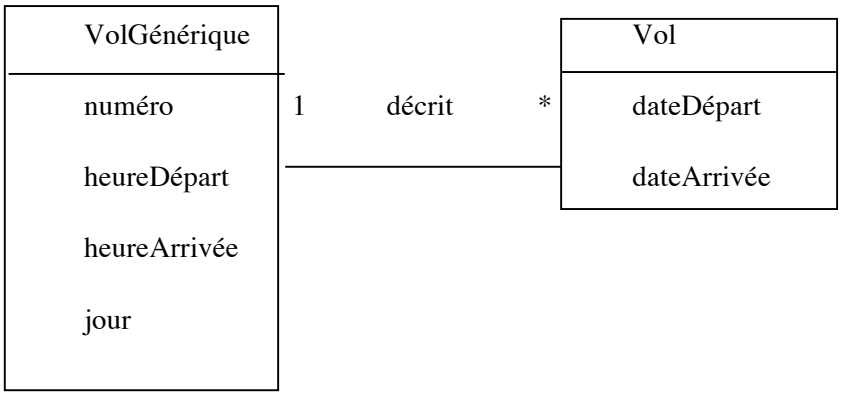
Soit la classe des vols d'une compagnie aérienne. Cette classe contient les attributs suivants :



Le numéro du vol caractérise tous les vols qui ont lieu le même jour de la semaine à la même heure.

En réalité, cette classe modélise deux classes : la métaclasse des « genres de vols » - VolGénérique - et la classe des vols concrets.

La classe des vols concrets ne contient que les attributs : dateDépart et dateArrivée.



12 - Notion de pattern

Présentation

Les patterns sont des micro-architectures finies.

Comme les frameworks (architectures semi-finies) ils permettent d'élever la granularité de la modélisation, et d'éviter de réinventer la roue à chaque projet !

Classes **Pattern** **Framework**

→ Granularité croissante

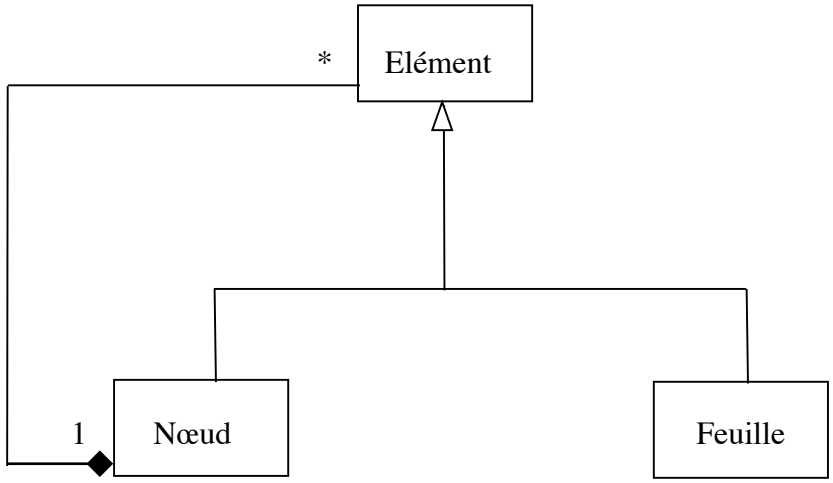
Caractéristiques des patterns

- Les patterns synthétisent l'élaboration itérative de solutions éprouvées, qui ont évolué au cours du temps pour fournir des structures toujours plus flexibles et plus facilement réutilisables.
- Les patterns forment un vocabulaire de très haut niveau et très expressif. Les informaticiens habitués aux patterns identifient, nomment et parlent des problèmes et des solutions en utilisant ce vocabulaire.
- Les patterns sont indépendants de tout langage de programmation, ce qui leur confère une grande généricité. En contrepartie, il n'y a pas de réutilisation de code et un effort de programmation est requis pour les mettre en œuvre.

Les différents types de patterns

- **Les patterns de conception (design patterns)** : ce sont les plus populaires car ce sont les patterns des informaticiens.
- **Les patterns d'analyse ou patterns métier** : les deux notions sont équivalentes. Ils dépendent d'un secteur d'activité particulier.
- **Les patterns d'architecture :** ils décrivent la structure des architectures logicielles, comme la structuration en couches.
- **Les patterns organisationnels** : ils fournissent des solutions à des problèmes organisationnels, telle l'organisation des activités de développement logiciel.
- **Les patterns d'implémentation :** ils expliquent comment exploiter un langage de programmation pour résoudre certains problèmes typiques.
- **Les patterns pédagogiques :** ils décrivent des solutions récurrentes dans le domaine de l'apprentissage.

Exemple de design pattern : Le pattern « composite »



Ce pattern est décrit dans « Design Patterns : Elements of Reusable Object-Oriented Software, E. Gamma et al., 1995, Addison-Wesley.

Il fournit une solution pour modéliser les arbres.

13 - Notion de framework

Présentation

Les frameworks sont des architectures semi-finies.

Comme les patterns (micro-architectures finies) ils permettent d'élever la granularité de la modélisation, et d'éviter de réinventer la roue à chaque projet !

Classes **Pattern** **Framework**

→ Granularité croissante

Caractéristiques des frameworks

- Les frameworks définissent des ossatures pour des familles d'applications.
- L'architecture d'un framework est générique, volumineuse et complexe.
- Un framework est une architecture semi-finie dictant l'architecture générale de l'application ou du sous-système à construire.
- Un framework peut être représenté par : les cas d'utilisation (le comportement externe), les paquetages (la macrostructure statique) et les collaborations (les patterns).
- Il faut fournir un effort important pour comprendre l'architecture générale des classes qui composent un framework et ainsi pour le maîtriser (cas des MFC : IHM sous Windows :

même si la création d'une fenêtre est une tâche aisée, il faut compter 6 mois pour maîtriser les MFC).

- Contrairement aux patterns, un framework est implémenté afin de permettre le développement rapide de nouvelles applications. Il dépend ainsi du langage de programmation utilisé pour son implémentation.

Classification des frameworks selon leur domaine d'utilisation

- **Les frameworks métier (framework verticaux)** : ils encapsulent l'expertise d'un domaine particulier.
- **Les frameworks d'application (framework horizontaux)** : ils encapsulent l'expertise technique applicable à plusieurs domaines.

Classification des frameworks selon leur structure

- **Les frameworks boîte blanche** : ce sont des applications semi-finies dont la spécialisation se fait par définition de sous-classes. Il faut connaître la structure interne des super-classes pour les spécialiser.
- **Les frameworks boîte noire**

Intérêt des frameworks

Du fait de l'architecture générique qu'il propose, le framework permet :

- **Productivité accrue**
- **Prototypage rapide**
- **Évolution facilitée**
- **Maintenance facilitée**

II - LES 3 GRANDS TYPES DE CLASSE

0 – Les 3 points de vue de la modélisation

3 points de vue guident la modélisation du diagramme des classes :

- Le point de vue statique
- Le point de vue fonctionnel (au sens de l'analyse fonctionnelle)
- Le point de vue dynamique

Ces trois points de vue aboutissent à trois types de classes :

- Les classes entités
- Les classes interfaces
- Les classes « control » ou « manager »

1 - Le point de vue statique : les classes « entité » ou « métier »

- Il met l'accent sur les concepts du domaine et les associations qui les relient.
- Les classes correspondantes sont parfois stéréotypées par le nom « entité » ou « entity »: c'est le point de vue qui se rapproche le plus du MCD (ou de la modélisation des données persistantes).

Méthode de construction du diagramme des classes « entité » :

- Analyse du type de celle faite pour un MCD. Il s'agit de trouver les classes (entités), les attributs et les associations à partir de la description « métier » du domaine.
- En même temps ou à la suite, ajouter les méthodes (les responsabilités au sens large) en s'appuyant sur l'analyse fonctionnelle des cas d'utilisation.
- En même temps ou à la suite, organiser et simplifier le diagramme en utilisant l'héritage.
- En même temps ou à la suite, ajouter les classes « interface » et « control »
- Itérer et affiner le modèle

2 - Le point de vue fonctionnel : les classes « interface »

- Il met l'accent sur les interfaces : interfaces utilisateur ou interfaces avec d'autres systèmes.
- Les classes correspondantes sont stéréotypées par le nom « interface » ou « boundary ».
- Les classes « interface » constituent la partie du système qui dépend de l'environnement.

Méthode pour trouver les classes « interface » :

- Examiner les paires acteur-scénario (occurrence de cas d'utilisation).
- Les classes correspondent en gros à une étape de l'interaction avec l'utilisateur.
- Ne pas hésiter à travailler à un haut niveau d'abstraction : ces classes seront affinées pendant la conception.

3 - Le point de vue dynamique : les classes « control »

- Il met l'accent sur l'implémentation.
- Les classes correspondantes sont stéréotypées par le nom « control ».
- Les classes « control » modélisent le séquençage comportemental (une séquence est une suite ordonnée d'opérations ; le séquençage est la détermination de la séquence).

Méthode pour trouver les classes « control » :

- Ajouter une classe « control » pour chaque paire acteur / cas d'utilisation.
- Les classes « control » définies par le couple acteur/cas d'utilisation ne sont qu'un point de départ : au cours de l'avancement de l'analyse, elle pourront être éliminées, éclatées ou fusionnées.

4 - Attention !!!

L'importance des classes entités

Les classes « interface » et « control » ne doivent pas conduire à dissocier le comportement et les données :

- Elles ne doivent pas conduire à revenir à la programmation procédurale !
- Elles ne doivent pas appauvrir le diagramme des classes « entité ».

Le point de vue essentiel reste le point de vue des classes « entité ». Les deux autres points de vue servent :

- A rendre plus concrète la modélisation
- A partir d'un niveau d'abstraction élevé
- A affiner le diagramme des classes « entité ».

L'importance des méthodes

L'analyse des méthodes est centrale dans la modélisation si on ne veut pas en rester à un modèle type « base de données ».

A noter que cette analyse est difficile et qu'elle sera facilitée par le point de vue dynamique (les classes « control ») et par l'usage de diagramme de séquence objet.

Documentation des méthodes

Dans le diagramme de classe, il faudra préciser pour chaque méthode (ou au moins pour chaque opération des interfaces) ce qu'on précise classiquement pour chaque fonction :

- Nom de la fonction
- Liste des paramètres : nom de la variable, type, mode de passage (entrée, sortie, entrée-sortie), signification (usage).
- Liste des attributs de classe utilisées par la méthode : nom de l'attribut, mode de passage
- But de la méthode

Une zone de commentaires permet d'ajouter ces informations dans les ateliers de génie logiciel (avec power AMC par exemple). Les commentaires se retrouvent ensuite dans le code généré automatiquement.

5 - Quelques étapes de la modélisation

1 : Faire le MEA et le transformer en diagramme de classes UML

On arrive ainsi aux classes métier. On peut analyser les méthodes de ces classes en réfléchissant aux usages associés à ces classes.

2 : Ajouter les classes interfaces

A partir des cas d'utilisation, UC, les plus généraux, on va retrouver les méthodes dont on a besoin dans les classes métier. On pourra ainsi définir les interfaces utiles.

3 : Ajouter les classes de gestion (de contrôle).

Chaque acteur peut être considéré comme porteur d'une gestion particulière et ainsi donner lieu à une classe de gestion particulière qui utilisera une ou plusieurs interfaces.

4 : A partir des diagrammes de séquence système de chaque UC

Pour chaque UC, on imagine avec quels objets concrets on va communiquer, puis comment les objets vont collaborer entre eux. Imaginer la communication consiste à imaginer les méthodes (au sens de fonction) qui vont être appelées et imaginer à quels objets on va les attribuer. On peut ainsi mettre à jour les classes métier et les interfaces.

III - LES DIAGRAMMES D'OBJETS

1 - Rappels techniques sur les objets

Définition

Un objet est un représentant concret d'une classe.

Déclaration

Pour pouvoir utiliser un objet, c'est-à-dire utiliser une de ses opérations publiques, il faut le déclarer une variable ayant comme type la classe de l'objet en question. Techniquement, la déclaration d'un objet est toujours la **déclaration d'un pointeur** sur un objet (ou une référence). La déclaration ne crée pas l'objet mais seulement le pointeur (la référence) vers un futur objet.

Construction

On distingue donc entre déclaration et construction de l'objet. La construction s'effectue par l'opérateur « **new** » qui va allouer dynamiquement un objet et renvoyer l'adresse de cette allocation, adresse qui sera affectée à une variable ayant comme type la classe de l'objet en question.

Comme toute variable allouée dynamiquement, l'objet n'a donc pas à proprement parler de nom. Quand on déclare un objet, le nom de la variable est le nom du pointeur qui permet d'accéder à l'objet. Le pointeur pourra référencer un autre objet : on peut donc changer d'objet sans changer de variable.

Initialisation

Un objet peut être initialisé lors de sa construction, via des opérations particulières appelées « **constructeurs** ».

En conception, la construction-initialisation n'est pas abordée. C'est cependant un élément central de la programmation objet.

Lieux de déclaration – construction des objets

La déclaration et la construction d'un objet peut se faire à deux endroits différents :

- Au niveau des attributs d'une classe
- Au niveau d'une variable locale d'une opération

Provenance des objets d'une méthode

Un objet, dans le corps d'une méthode, provient de trois lieux :

- C'est un **attribut de l'objet** de la méthode : il a été construit avec l'objet.
- C'est une **variable locale** à la méthode : il est construit dans la méthode.
- C'est un **paramètre formel** de la méthode : il a été fournit par la fonction appelante. Toutefois, en dernière analyse, on retombera sur les deux premiers cas.

2 - Syntaxe UML

Objets

Les objets sont soulignés et placés dans un rectangle. Le nom de l'objet commence par une minuscule. Le nom de la classe commence par un majuscule.

Objets nommés : olivier bertrand

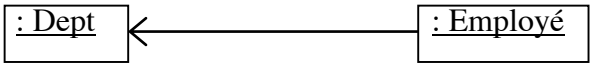
Objets sans noms : : Eleve : Professeur

Lien entre objets

Tous les liens entre classes sont représentables pour les objets. Toutefois, c'est l'association qui est le principal lien.

- Association
- Instanciation
- Héritage
- Dépendance

Exemple :



On représente les liens entre les objets : ici une association.

On peut flécher l'association pour signifier que le lien n'est que dans un seul sens.

En général, les associations entre objets sont 1-1. On peut toutefois préciser la cardinalité si on veut montrer une collection.

On peut aussi montrer la représente les liens entre les objets : ici une association.

3 - Diagramme d'objets

Principes

Le diagramme d'objets montre **les objets et leurs liens à un moment** de l'exécution du programme.

C'est **un instantané, une photo**, d'un sous-ensemble d'objets d'un système à un instant de la vie du système.

Il permet de rendre plus concrètes et plus claires certaines parties du diagramme de classe.

C'est un diagramme orienté développeur.

Représentation UML

La représentation s'apparente à celle d'un diagramme de classes.

On représente les objets et pas les classes, donc pas les liens d'héritage.

On représente les liens du diagramme de classes entre les objets tels qu'il sont représentés dans un diagramme de classes.

4 - Communication entre les objets : envoi de message

Principe général de l'envoi de message

Envoyer un message à un objet c'est utiliser une de ses opérations.

Un objet 1 envoi un message à un objet 2 quand une opération de l'objet 1 utilise une opération de l'objet 2.

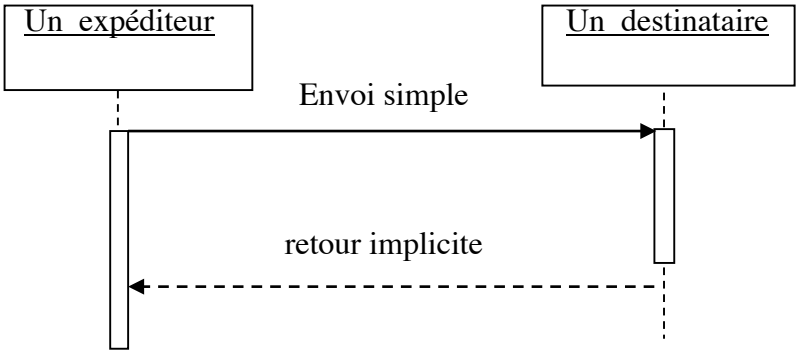
Pour cela, l'objet 2 doit exister dans l'environnement de l'opération appelante de l'objet 1 :

- soit c'est une variable locale de l'opération 1,
- soit c'est un objet passé en paramètre de l'opération 1,
- soit c'est un attribut de l'objet 1.

Envoi de message et héritage

En programmation objet, **la manière d'associer du code à un message** est un mécanisme dynamique qui n'est pas établi à la compilation. Autrement dit, l'envoi d'un message **ne correspond pas à un débranchement vers une adresse de code prédéfinie** comme c'est le cas en programmation procédurale classique. **C'est la relation d'héritage entre classes qui permet de gérer ce mécanisme**.

Syntaxe UML du diagramme de séquence



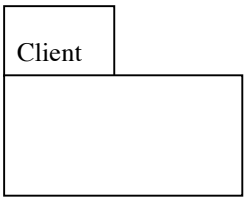
IV - PAQUETAGES ET VUE LOGIQUE

1 - Présentation et syntaxe UML

Un paquetage permet de regrouper les classes pour faciliter leur emploi, leur maintenance et leur réutilisation.

Le paquetage est un élément clé de l'architecture. Les principaux paquetages apparaissent donc très tôt dans la phase d'analyse.

Dans la version 2 d'UML, un paquetage peut regrouper n'importe quels éléments de modélisations (les cas d'utilisation, les classes, etc.).

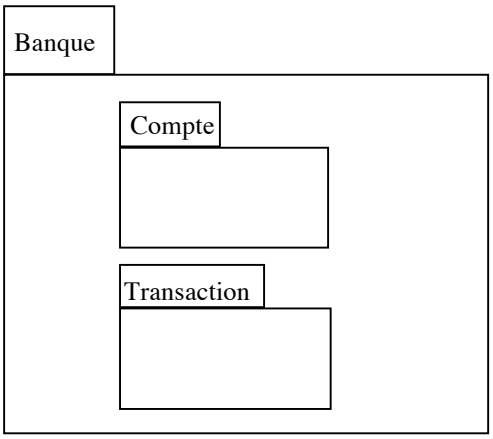


Inclusion de paquetage

Un paquetage peut inclure un autre paquetage.

Un paquetage peut être inclus dans deux paquetages différents.

Les paquetages de cas d'utilisation définissent des sous-domaines fonctionnels du système étudiés. Ces sous-domaines peuvent partager des cas d'utilisation.



Nommage des paquetages

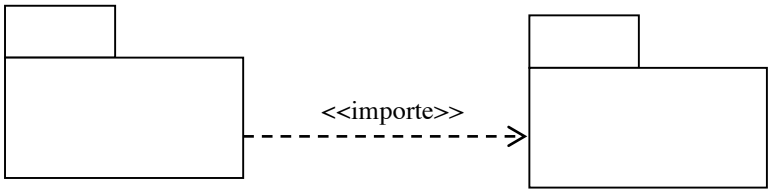
Banque ::Compte désigne le paquetage Compte défini dans le paquetage Banque

Si le cas d'utilisation « Retirer argent » se trouve dans le paquetage transaction, on écrira : Banque ::Compte ::Retirer argent pour le désigner.

Dépendances entre paquetages

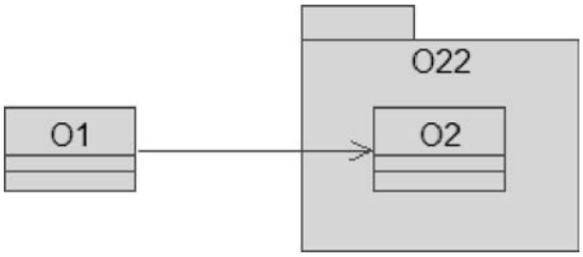
<<importe>> : ajoute les éléments du paquetage destination au paquetage source.

Si on ne précise pas le stéréotype, c'est « importe » par défaut.

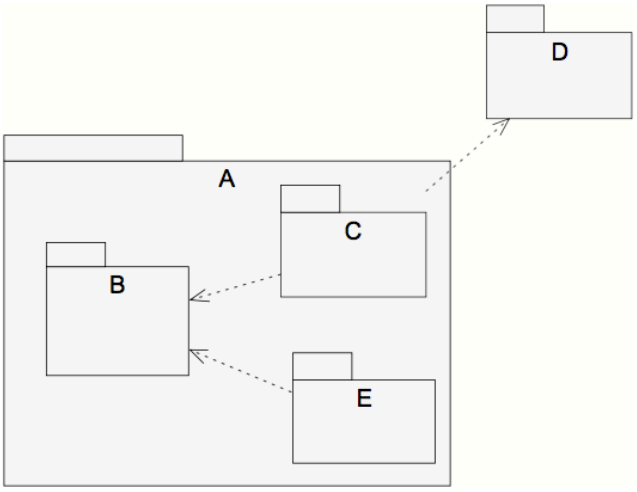


Les éléments du paquetage destination sont visibles dans le paquetage source. Toutefois, il n'y a pas de relation de possession.

La classe O1 utilise la classe O2 :

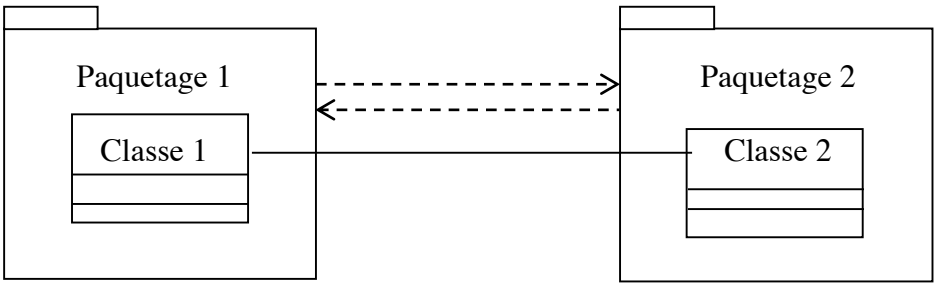


Dépendances multiples :

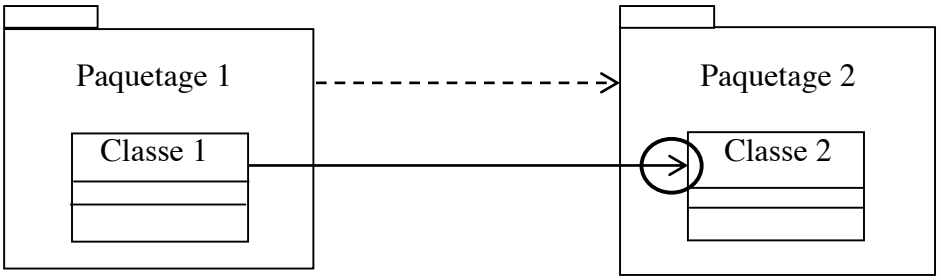


Réduction du couplage entre paquetage

Si deux classes sont liées, les deux paquetages seront liés.



Réduction de la navigabilité entre les classes et du couplage entre les paquetages :



Le choix de la navigabilité est fait en fonction de l'utilisation prévue pour les classes. Ce choix doit être justifié.

Paquetage réutilisable

Un paquetage est réutilisable quand il ne dépend d'aucun autre paquetage.

2 - Paquetages métiers, interfaces et techniques

Paquetages « métier »

La vue logique est élaborée au début de la phase d'analyse avec les classes « métier ».

L'analyse des classes métiers permet de les regrouper dans différents paquetages métiers.

Les paquetages métiers permettent de définir des ensembles de classes portables d'un projet à un autre.

Le choix des frontières entre les paquetages se fera aussi en fonction du couplage entre les paquetages c'est-à-dire de la navigabilité entre les classes. Le but est d'obtenir le maximum de paquetage réutilisable (indépendant des autres paquetages).

Paquetage Interface

Le paquetage interface permet de regrouper les classes interfaces des différentes classes métier.

Paquetages techniques

Ensuite, en fonction des choix de programmation effectués (langage, gestion d'une base de données, gestion des erreurs, etc.), on pourra avoir des paquetages techniques dans l'architecture de la vue logique :

- **Paquetage BaseDeDonnées :** il permet de gérer le problème de la permanence des données.
- **Paquetage IHM :** il contient des classes fournies par le langage de programmation.
- **Paquetage ClassesDeBase (global) :** c'est un paquetage qui est utilisé par tous les autres paquetages du système.
- **PaquetageGestionDesErreurs (global) :** c'est un paquetage spécialisé dans la gestion des erreurs et qui est utilisé par tous les autres paquetages du système.

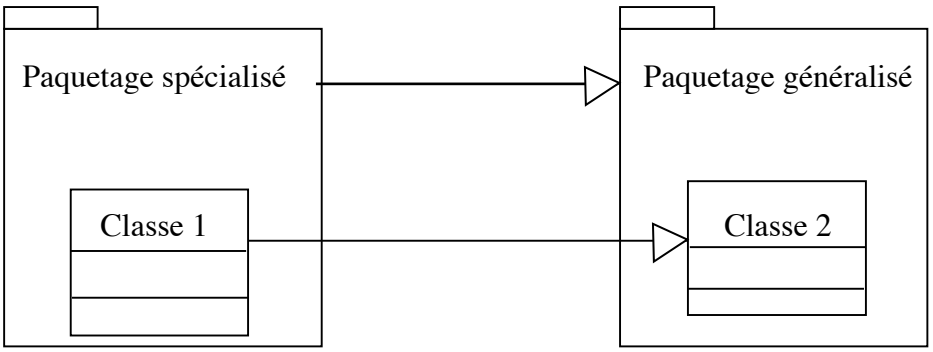
3 - Paquetage généralisé - paquetage spécialisé

Si on est amené à déterminer des classes abstraites qui seront donc spécialisées par d'autres classes, on peut créer une frontière de paquetage entre la classe abstraite et ses classes spécialisées.

Le paquetage contenant la classe abstraite est alors dit « paquetage généralisé ». Il n'est pas dit paquetage abstrait car il peut aussi, par ailleurs, contenir des classes concrètes.

Le paquetage contenant les classes concrète est alors dit « paquetage spécialisé ».

La relation entre les deux paquetages est une relation de généralisation :



4 - Paquetage paramétré (ou générique ou template) et *bound package*

UML 2 introduit la notion de paquetage paramétré (ou paquetage générique, ou paquetage template).

Les paquetages liés au paquetage paramétré sont appelés *bound package*.

Cette notion est peu mise en œuvre.

5 - Paquetage de cas d'utilisation

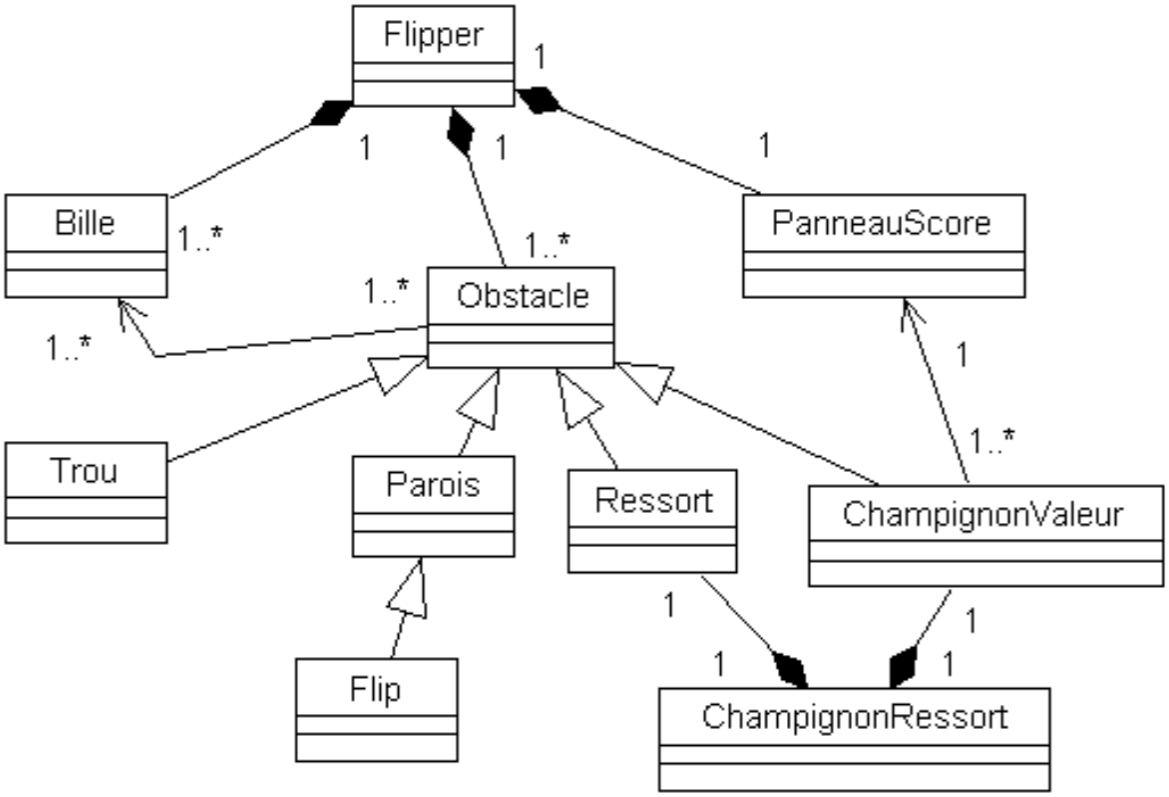
On peut aussi utiliser les paquetages pour regrouper des cas d'utilisation qui vont ensemble ou pour présenter les cas d'utilisation abstraits.

V - EXEMPLES

Flipper

5 catégories d'obstacle peuvent se trouver dans le flipper : la paroi, le trou qui fait disparaître la bille, le ressort qui la fait rebondir, le champignon de valeur incrémente le score de la valeur indiquée sur l'obstacle et le champignon à ressort qui se comporte comme un champignon et un ressort.

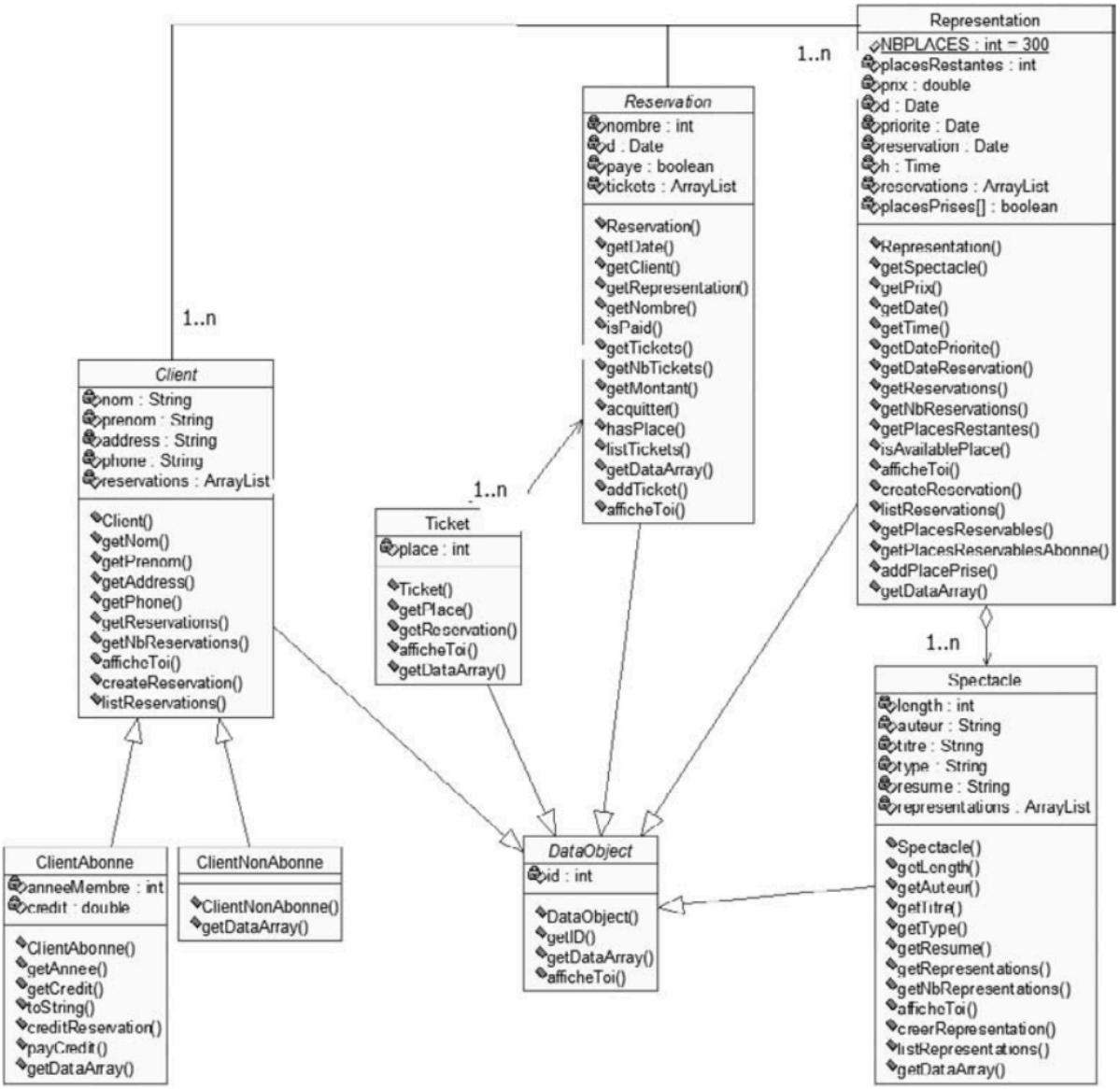
Notez que le ChampignonRessort n'hérite pas directement de l'obstacle mais est composé d'un ressort et d'un champigonValeur qui eux hérite de l'obstacle.



Réservations de spectacle

Problème d'interprétation

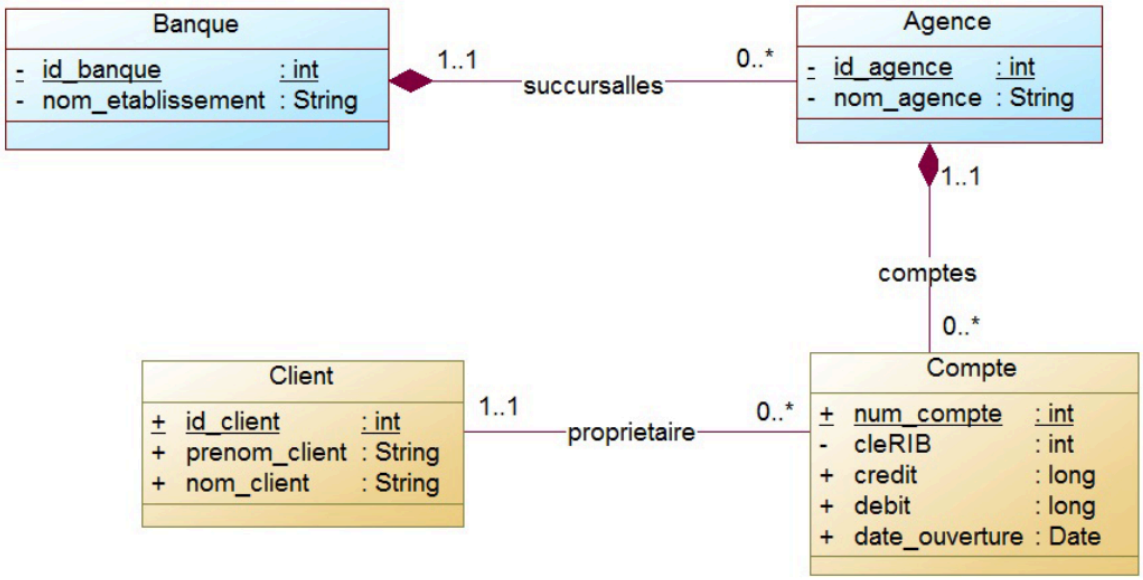
Le modèle ci-dessous manque d'informations sur les associations. Il rend son interprétation difficile sans précisions sur les données métier.



Banque, agence, compte, client

Utilisation d'une composition entre Banque et Agence (pas de banque, pas d'agence), entre agence et compte (pas d'agence, pas de compte).

Par contre entre les comptes et les clients, on n'a ni composition, ni agrégation.



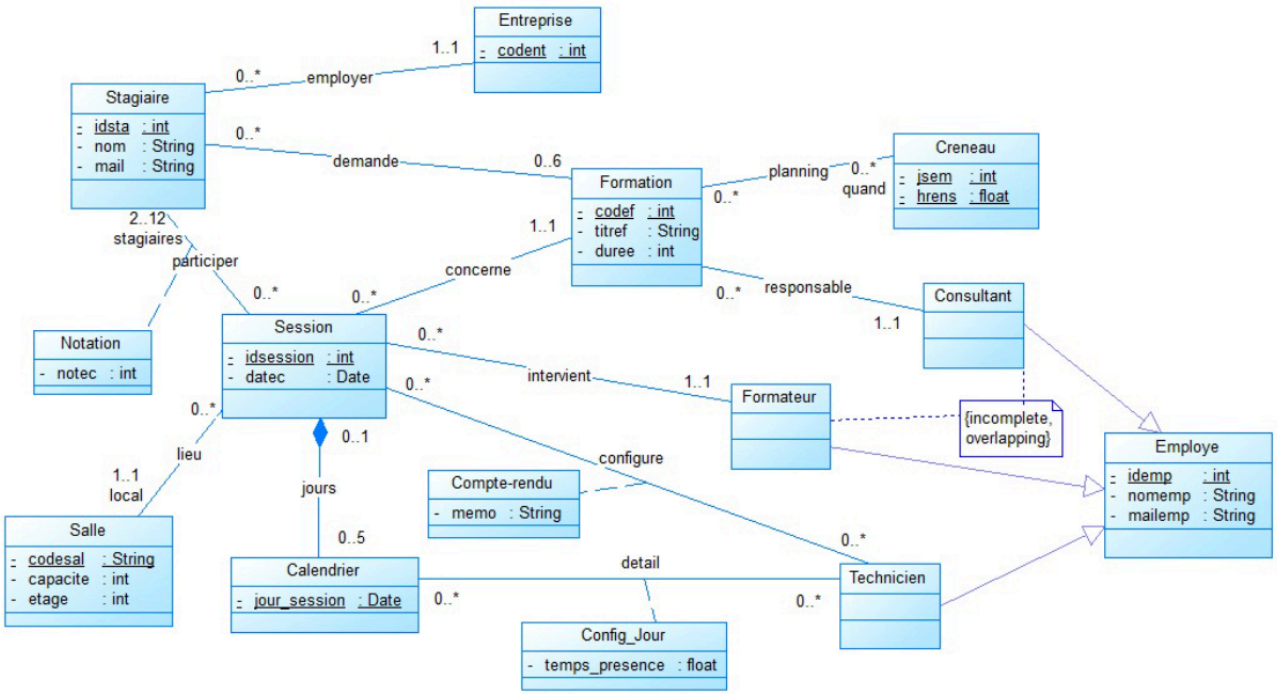
Organisme de formation

A noter ici :

La formation est une formation type avec ses créneaux type. Elle est gérée par un consultant.

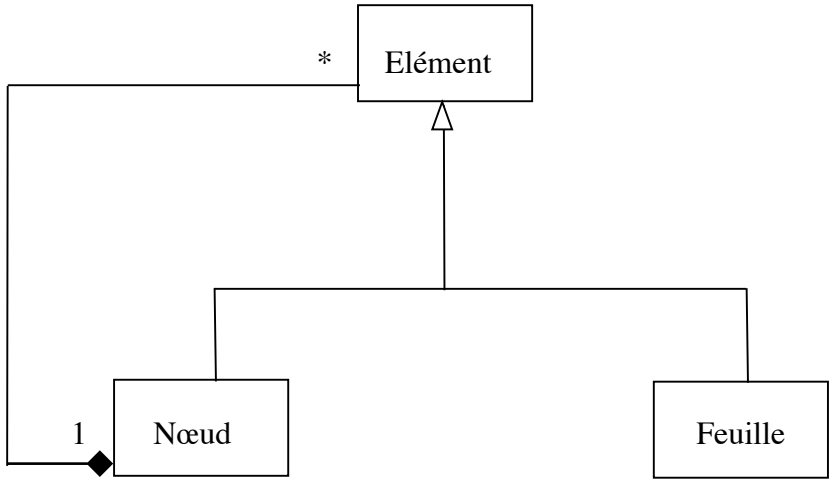
Elle donne lieu à des sessions de formations qui sont des instances concrètes avec leur calendrier concret. Si la session est supprimée, son calendrier l'est aussi.

Le formateur est rattaché à la session. Le fonctionnement est qu'il n'y a qu'un formateur par formation.



VI – DESIGN PATTERN

Exemple de design pattern : Le pattern « composite »



Ce pattern est décrit dans « Design Patterns : Elements of Reusable Object-Oriented Software, E. Gamma et al., 1995, Addison-Wesley.

Il fournit une solution pour modéliser les arbres.

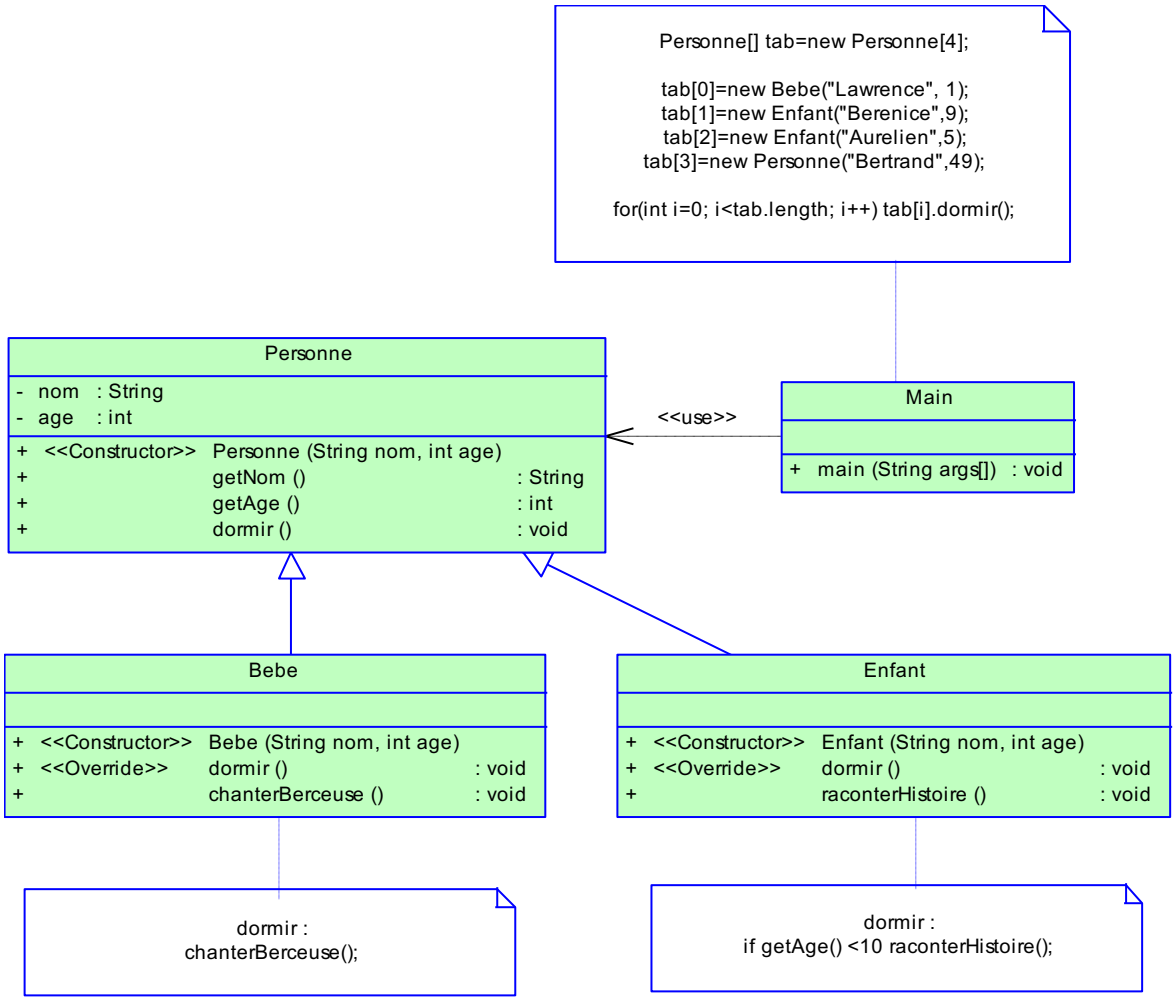
Héritage *vs*. délégation

La délégation offre un mécanisme de réutilisation aussi puissant que celui offert par la généralisation.

L'héritage est une construction rigide mais la propagation des attributs et des méthodes est automatique. L'héritage correspond à une relation « est-a » stricte, c'est-à-dire pas à une relation « a-un-comportement-de ».

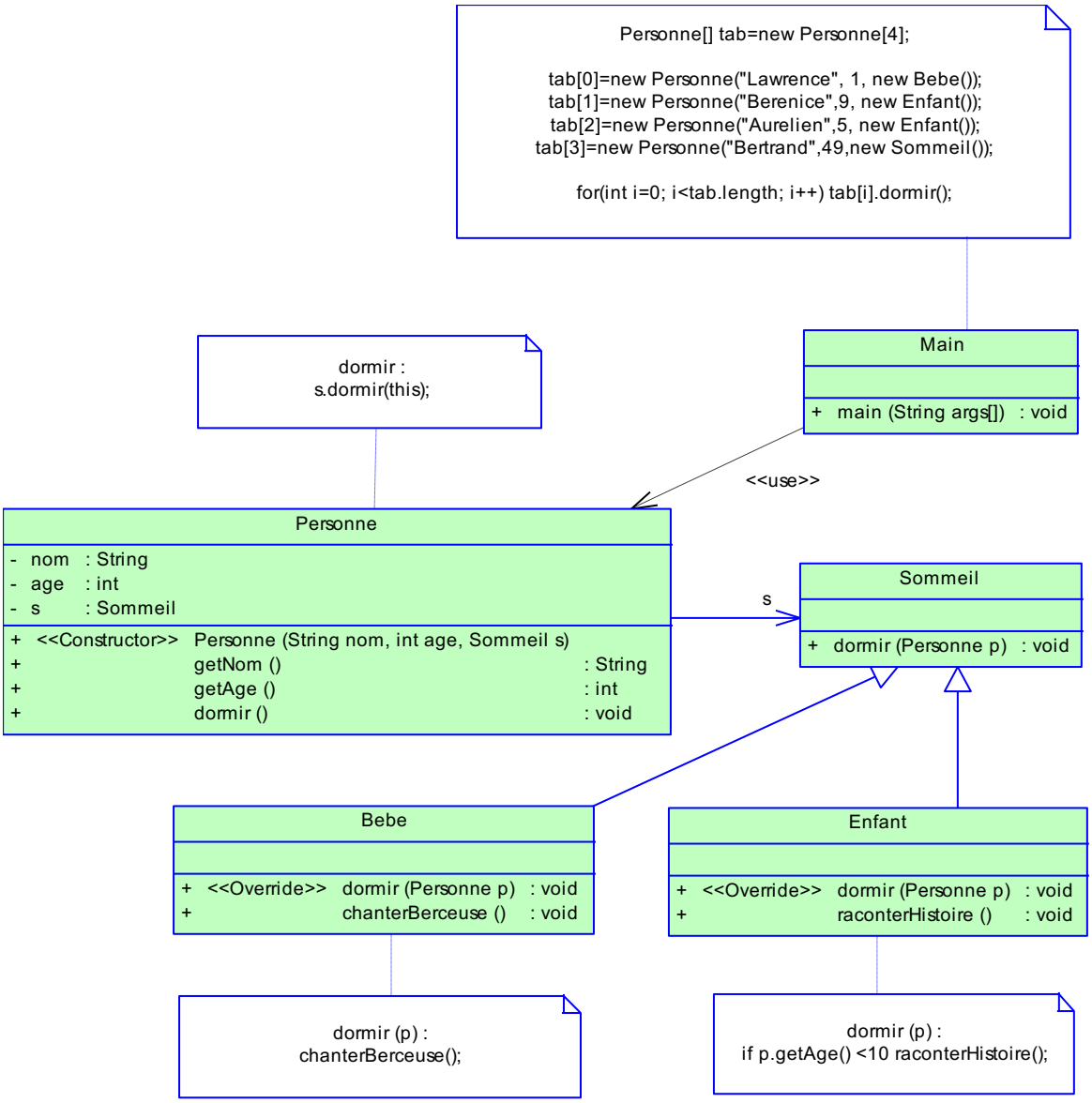
La délégation est une construction plus souple basée sur l'agrégation. La propagation des propriétés doit être réalisée manuellement. Elle permet généralement la mise en œuvre de généralisation multiple.

Héritage



Ici le mécanisme est un mécanisme classique d'héritage et de polymorphisme.

Délégation



Ici on obtient le polymorphisme par la délégation.

On fournit l'objet « p » à la méthode dormir pour accéder aux méthodes de Personne, si nécessaire (ici : if p.getAge <10…).

On peut considérer qu'on « sette » une méthode spécifique à la personne au moment de son instanciation : « new Personne("Lawrence", 1, new Bebe()); » consiste à setter la méthode « dormir » des Bébé à la personne, comme on lui a setté son prénom et son âge.

Cette technique sera particulièrement utile en cas d'héritage multiple car cette fois-ci elle offrira une solution plus performante que la solution de l'héritage classique. On verra ça particulièrement dans le design pattern Stratégie.

Design pattern : pattern stratégie

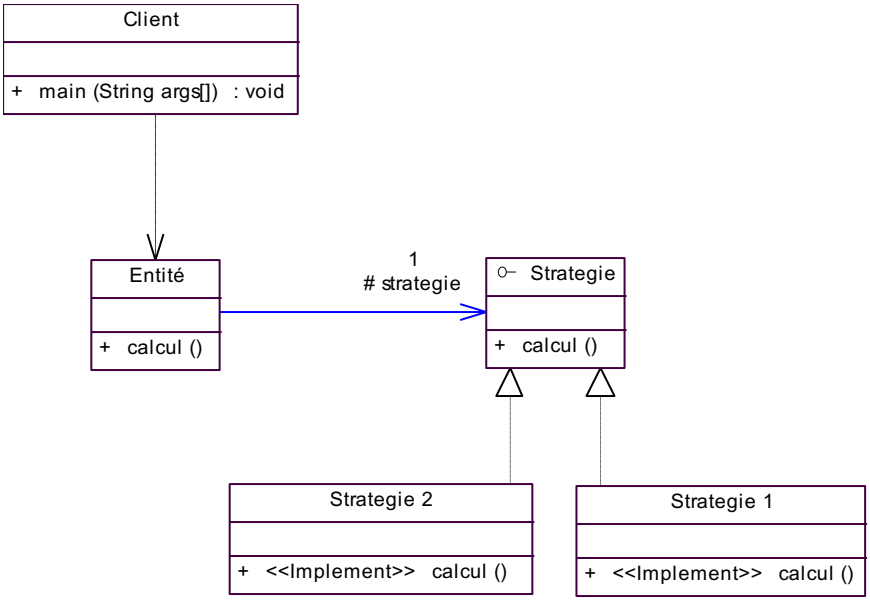
Les objets de la classe qui utilise la classe « Stratégie » contiennent un objet de la classe « stratégie » et sont instanciés en passant en paramètre un objet correspondant à une stratégie concrète.

Ainsi, ces objets accèdent de façon polymorphe aux méthodes de la classe « Stratégie »

La classe Stratégie est une interface qui définit les comportements.

Les stratégies concrètes réalisent cette interface.

A noter la « sucette » à coté du nom de l'interface Strategie. Elle dit graphiquement que c'est une interface. On pourrait aussi utilise un stéréotype <<interface>>.



VII - ANNEXE

1. Génération de C++

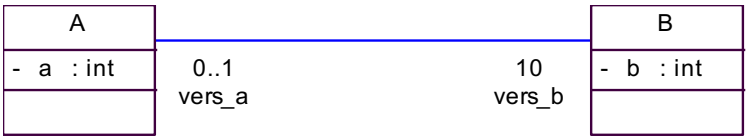
Classe et association

Association 1 - *

```
class B;
class A
{
private:
   int a;
   B** vers_b;
};
```

```
class A;
class B
{
private:
   int b;
   A* vers_a;
};
```

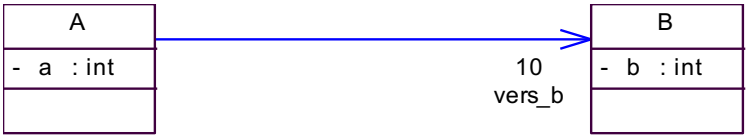
Association 0.1 - 10




```
class B;
class A
{
private:
   int a;
   B* vers_b[10];
};
```

```
class A;
class B
{
private:
   int b;
   A* vers_a;
};
```

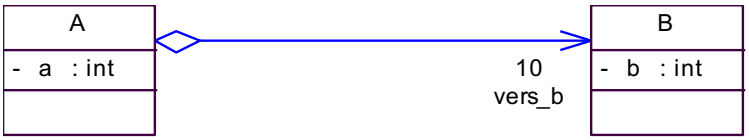
Association mono-navigable




```
class B;
class A
{
private:
   int a;
   B* vers_b[10];
};
```

```
class B
{
private:
   int b;
};
```

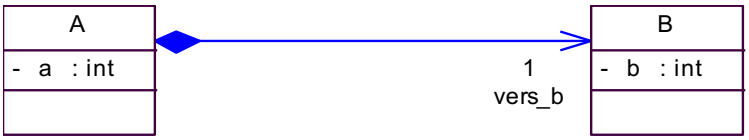
Classe et agrégation




```
class B;
class A
{
private:
   int a;
   B* vers_b[10];
};
```

On retrouve la même chose qu'avec une simple association.

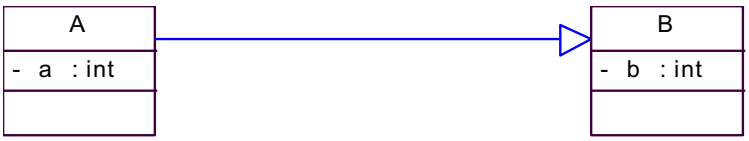
Classe et composition




```
class B;
class A
{
private:
   int a;
   B* vers_b;
};
```

On retrouve la même chose qu'avec une simple association.

Héritage

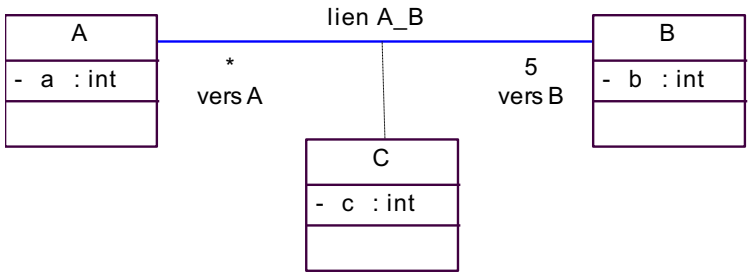



```
#include <B.h>
class A : public B
{
private:
   int a;
};
```

```
class B
{
private:
   int b;
};
```

On retrouve la même chose qu'avec une simple association.

Classe association




```
class C;
class A
{
private:
  int a;
 C* lien_A_B;
};
```

```
class C;

class A

{

private:

   int a;

   C* lien_A_B;

};
```

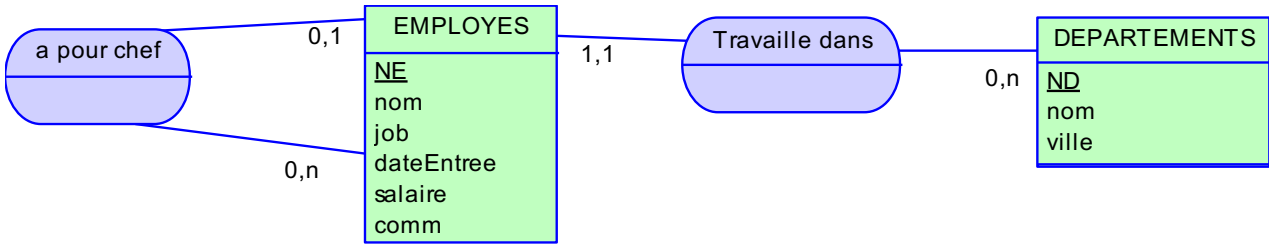
```
class A;

class B;

class C

{

private:

   int c;

   B* vers B[5];

   A** vers A;

};
```

A noter que la production de code par Power AMC est fausse : la classe C devrait pointer que 1 A et 1 B et les classes A et B vers plusieurs C.

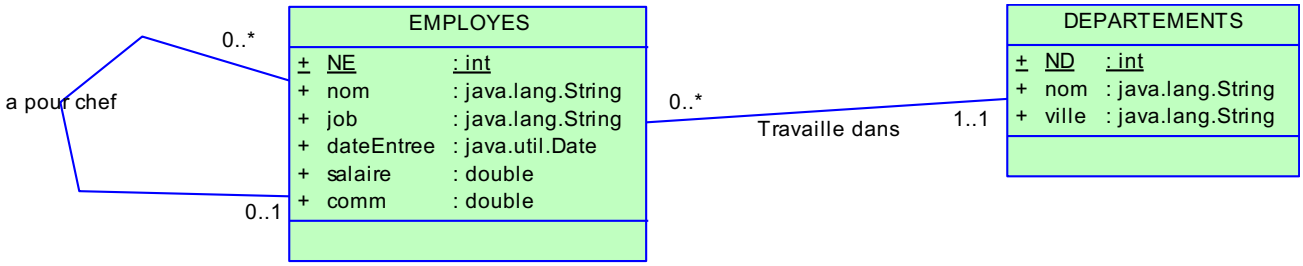
2. Traduction de MEA en UML

Les employés et les départements

MEA



UML

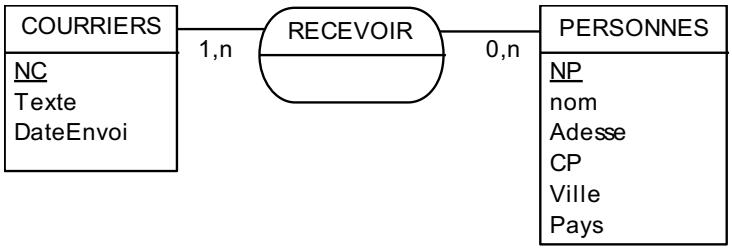


Explications

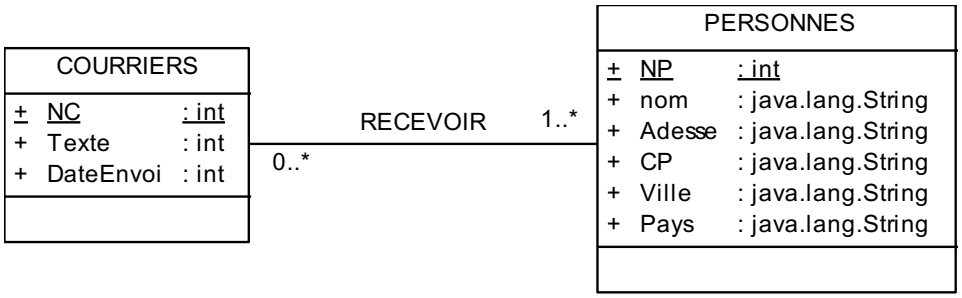
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- La notion de clé primaire n'a pas de signification dans un diagramme de classe. En effet, tout objet (instance d'une classe) est caractérisé par ses attributs, ses méthodes et son identifiant qui est son adresse. Cependant, on peut préciser la notion d'identifiant primaire pour les attributs.
- Les cardinalités des associations UML peuvent reprendre le même formalisme que dans le MEA : 0.1, 1.1, 0.N, 1.N.
- La position des cardinalités est inversée par rapport au formalisme MEA : un employé travaille dans 1 et 1 seul département. La cardinalité 1.1 est du coté du département.
- Les associations UML sont orientées : il peut y avoir des flèches dans un sens ou un autre. Cette notion n'ayant pas de sens dans le MEA, l'association sera rendu navigable dans les deux sens, ce qui conduit à l'élimination des flèches.

Les courriers : association non hiérarchique sans attributs

MEA



UML

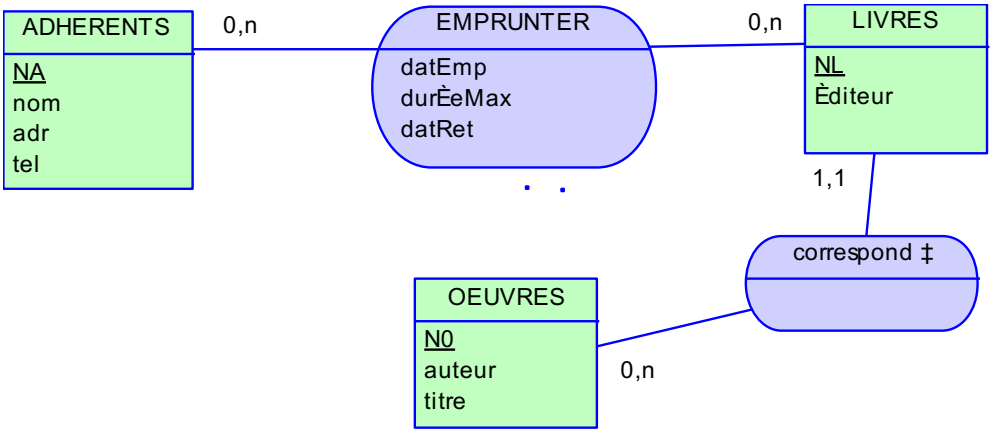


Explications

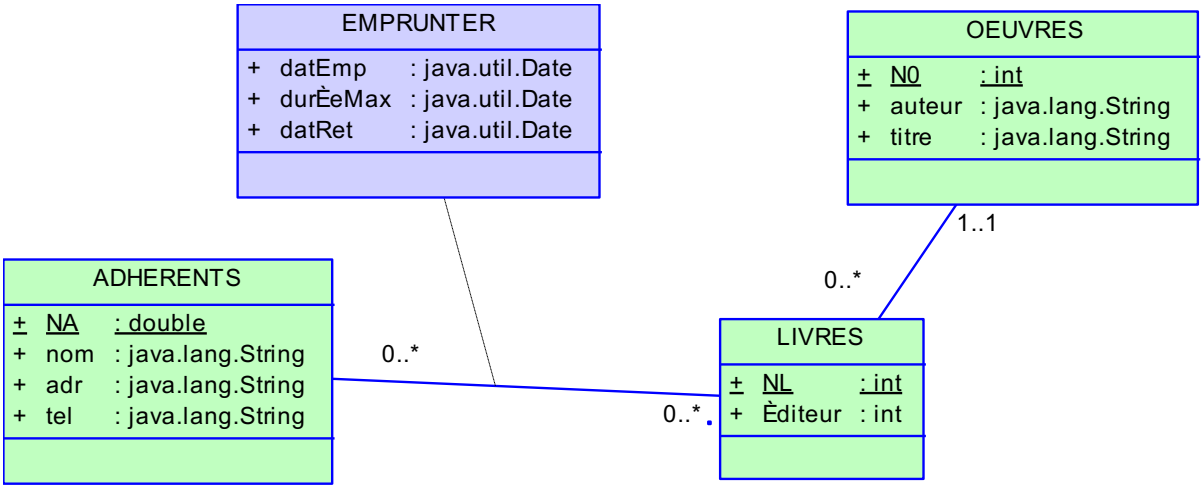
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- Les associations non hiérarchiques sans attributs du MEA sont transformées en association dans le modèle de BD UML. La position des cardinalités est inversée : un personne peut recevoir 0 ou N courriers (ça peut être 0 si on ne lui à jamais écrit). Un courrier est envoyé à 1 ou N personnes. Au moins 1 car il n'y a pas de courriers qui ne soit pas envoyé.

La bibliothèque : association non hiérarchique avec attributs et classe-association

MEA



UML

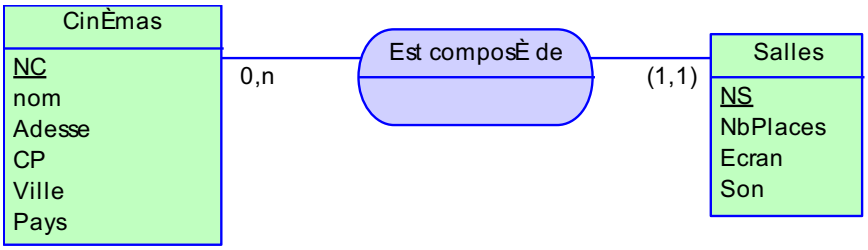


Explications

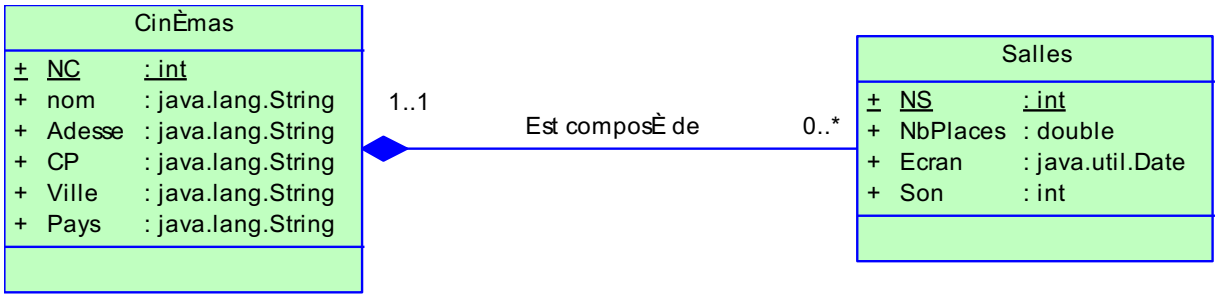
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- En UML, on ne peut pas mettre d'attributs sur les associations. Les associations non hiérarchiques avec attributs du MEA donnent lieu dans le modèle de BD UML à des classes-associations.
- Une classe-association est une association classique à laquelle est raccrochée une classe dont les attributs proviennent de l'association non hiérarchique du MEA.

Les cinémas : identifiant relatif et composition

MEA



UML

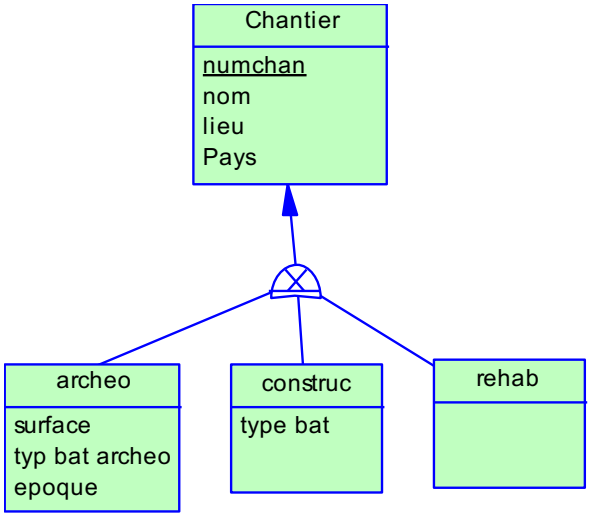


Explications

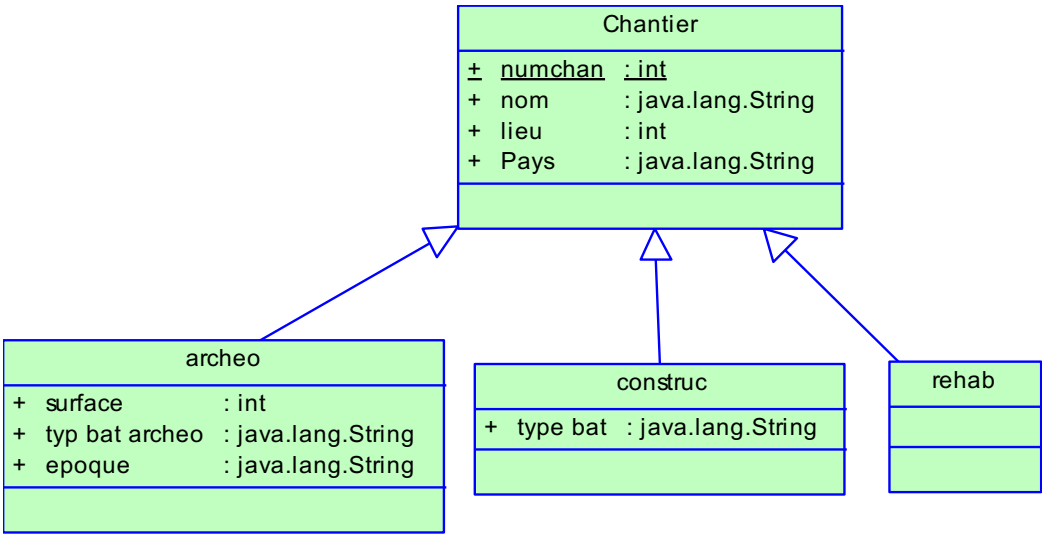
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- L'identifiant relatif du MEA est transformé, dans le modèle de BD UML, en une association de composition : losange plein (et pas creux !) du coté du composé.
- La composition signifie que si on supprime de cinéma, alors on supprime aussi les salles.

Les chantiers : héritage

MEA



UML



Explications

- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- L'héritage dans les MEA se traduit par un héritage dans le modèle de BD UML.
- Dans notre exemple, l'héritage est de type XT, ou + (X souligné dans notre formalisme) : ce qui veut dire qu'il y a couverture et disjonction : un chantier ne peut être que d'une seule espèce (disjonction, X) et il est forcément d'une espèce donnée (couverture, T).
- Les caractéristiques de disjonction, X, et de couverture, T, ne sont pas représentées dans le modèle de BD UML.