UML
Diagramme de classes
Diagramme d'objets – Packages

Bertrand LIAUDET

SOMMAIRE

Juin 2019 – mise à jour juin 2022

LE DIAGRAMME DE CLASSES

Il est facile de décrire la méthode ou le langage, encore que son application exige à coup sûr savoir et pratique.

	Point de vue	Diagramme UML
ANALYSE TECHNIQUE	Statique – objet	Classes
		Objets
	Dynamique - objet	Séquence
		Communication
	Dynamique Plutôt non objet	Etats-transitions
		Activités

1 - Notions générales sur les classes et formalisme UML

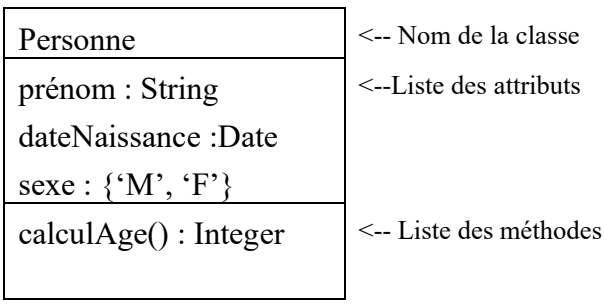
Définition

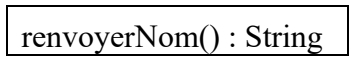
- **Une classe** est une description abstraite d'un **ensemble d'objets qui partagent les mêmes propriétés** (attributs et associations) **et les mêmes comportements** (mêmes opérations, c'est-à-dire les mêmes en-têtes de méthodes).

Notion d'instance

- Une instance est une réalisation concrète d'un modèle abstrait.
- Un **objet** est une instance d'une **classe**.
- Une **entité** est une instance d'une **entité-type** (MCD Merise).
- 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 P1, lui même dans le paquetage P2, on écrira : **P2 :: P1 :: Personne**

Encapsulation

- L'occultation des détails de réalisation est appelée : **encapsulation**.
- **Avantage** : **les données** encapsulées dans les objets sont **protégées**.
- **Avantage** : Les utilisateurs d'une abstraction ne dépendent pas de sa réalisation, mais seulement de sa spécification, ce qui **réduit le couplage dans les modèles**.

Visibilité des attributs et des méthodes

- **Paramétrage** possible du degré d'encapsulation : c'est la notion de **visibilité**.
- **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.

Les 4 niveaux de visibilité

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**
-	Private	Visible uniquement **dans la classe**

Attribut et méthode de Classe

Principes

- Certains **attributs** ont une **valeur identique pour tous les objets** de la classe. Ce sont des attributs qui concernent toutes la classe.
- Certaines **méthode** ne porte pas sur les attributs d'un objet particulier mais sur des attributs de classe ou sur des valeurs constantes. Ce sont des méthodes **qui concernent toute la classe**.
- On y accède en préfixant le nom de la méthode par le nom de la classe.

Représentation UML

- En UML, ces attributs et ces méthodes sont listés avec les autres, mais sont soulignés.

	Symbole	**Mot-clé**	**Signification**
Attribut :	Souligné	**static**	Valeur identique pour tous les objets de la classe
Méthode :	Souligné	**static**	Accès à partir de la classe et pas d'un objet.

Classe stéréotypée <<utility>>

- On utilise le stéréotype <<utility>> pour dire que c'est une classe avec uniquement des attributs et des méthodes de classes.

<<utility>> Math
<<static>> PI :Real=3,14
<<static>> sinus (angle): float <<static>> cosinus(angle): float <<static>> tangente(Angle): float

Attribut dérivé (ou calculé)

- 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
/	<< calculé >>	Attribut dont la valeur est calculée à partir de celle d'autres attributs

Compartiment des responsabilités : 3ème compartiment

- 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.
- C'est un de compartiment théorique de commentaires, de brouillon. Quand la conception sera achevée, ce compartiment devra disparaître.

Compartiment des exceptions : 4ème compartiment

- Le compartiment des « exceptions » liste les situations exceptionnelles devant être gérées par la classe.
- C'est un de compartiment théorique de commentaires, de brouillon. Là encore, quand la conception sera achevée, ce compartiment devra disparaître.

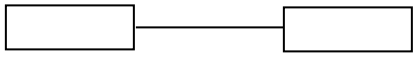
2 - Associations entre classes

- Comme dans le modèle entité-association (MEA, MCD MERISE), UML permet d'établir des relations (des associations) entre les classes.
- Il y a 4 types d'associations entre classes :
  - Les **associations simples**
  - Les associations d'**agrégation** et de **composition**
  - L'**énumération**
  - Les associations d'**héritage**
- Les associations simples, agrégations et compositions sont de même nature et expriment des relations entre les objets de deux classes (ou plus).
- **L'énumération** est une façon de **gérer des petites listes** de valeurs possibles pour un attribut.
- **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 simple ligne entre deux classes représente une association.

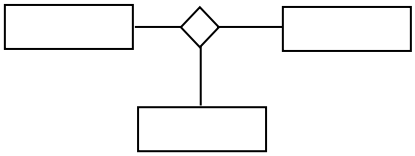


Association binaire (arité = 2)

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

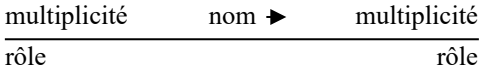
Association ternaire et plus (arité >2)

- Le carré posé sur un sommet : ◇ permet d'associer plus de 2 classes.



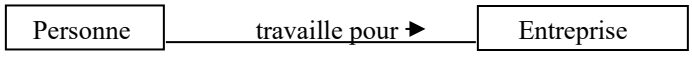
- On peut avoir plus de 3 classes associées mais c'est rare et plutôt le signe d'une erreur..

L'association et ses différents ornements



Nom et sens de lecture des associations

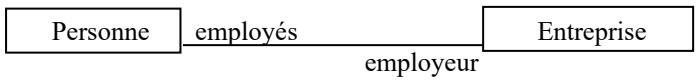
- L'association peut avoir un nom. Le nom explicite le lien entre les deux classes. C'est souvent un verbe. Le sens de lecture : ► permet de préciser dans quel sens il faut lire le nom.



- Une personne travaille pour une entreprise.
- Inversement, dans une entreprise travaillent des personnes.

Rôles des extrémités des associations

- L'association peut avoir des rôles. Les rôles explicitent le lien entre les deux classes. Ce sont souvent des noms, au singulier ou au pluriels. Le nom du rôle commence par une minuscule.



- La lecture est « automatique » dans les deux sens :
- Une personne **a un** « employeur » qui **est une** entreprise.
- Une entreprise **a des** « employés » qui **sont des** personnes.

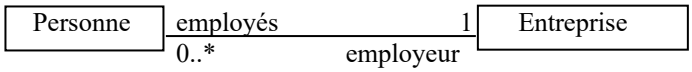
Multiplicité (cardinalité) des associations

Principe

- 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.

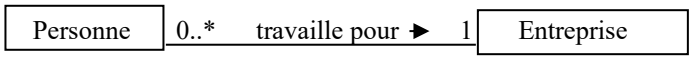
1	Un et un seul
0..1	Zéro ou un
N -- 3	N (entier naturel qui peut être précisé, par ex : 4)
M .. N -- 2..4	De M à N (entiers naturels qui peuvent être précisés, par ex : 2..4)
*	De zéro à plusieurs
0 .. *	De zéro à plusieurs
1 .. *	De un à plusieurs

Exemple 1



- Une personne **a un et un seul** « employeur » qui **est une** entreprise
- Une entreprise **a 0 ou plusieurs** « employés » qui **sont des** personnes.

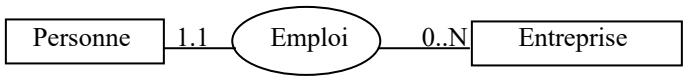
Exemple 2



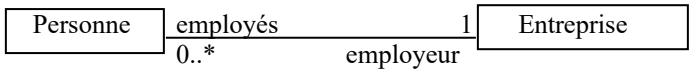
- Une personne travaille pour une entreprise et une seule.
- Dans une entreprise travaille de 0 à plusieurs personnes.

Attention !!! inverse du MEA

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



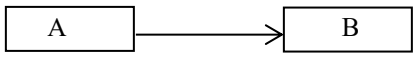
Signification de l'association pour les classes



- Dans la classe Personne, un attribut « employeur » est un objet de la classe Entreprise.
- Dans la classe Entreprise, un attribut « employes » est une collection de 0 à plusieurs objets de la classe Personne.

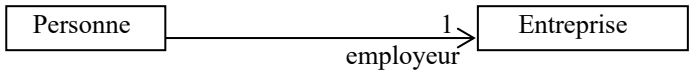
Navigabilité des associations

Principe



- L'association n'est **navigable que dans le sens de A vers B**, autrement dit **n'existe que dans le sens de A vers B**.

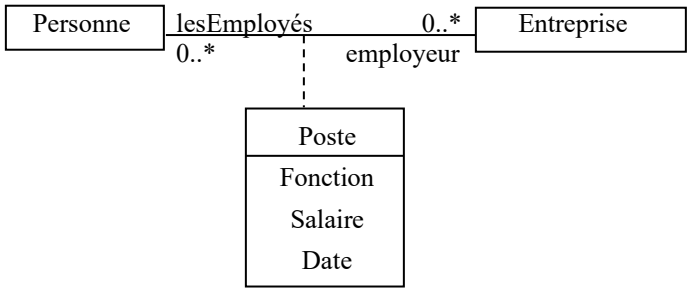
Exemple



- Dans la classe Personne, il existe un attribut employeur qui est un objet de la classe Entreprise.
- **Dans la classe Entreprise, il n'y a pas de personnes**.

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 :



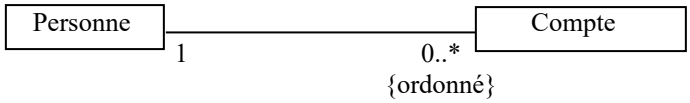
- La classe-association correspond à l'association non-hiérarchique avec attributs du MEA.
- Dans la classe Poste, il y a une personne et une entreprise.
- Dans la classe Personne, il y a plusieurs postes.
- Dans la classe Entreprise, il y a plusieurs postes.

Contraintes sur les associations (théorique)

Toutes sortes de contraintes peuvent être définies sur les associations.

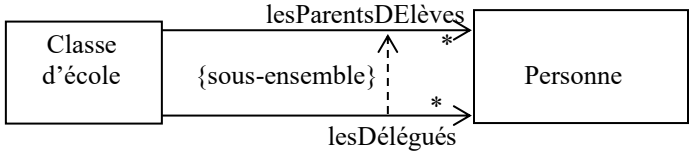
Certaines contraintes s'appliquent à une seule association :

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



Certaines contraintes s'appliquent à plusieurs associations :

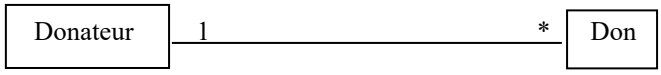
- {sous ensemble} : précise qu'une collection est incluse dans une autre collection
- {ou exclusif} : précise pour un objet donné qu'une association et une seule est possible parmi les associations contraintes par le ou exclusif.



Qualification (restriction) des associations (théorique)

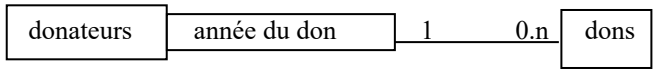
Principe

- La qualification des associations consiste à sélectionner les objets dans une association multiple.
- Exemple d'association multiple : le donateur peut faire plusieurs dons : il a 0 ou plusieurs dons.

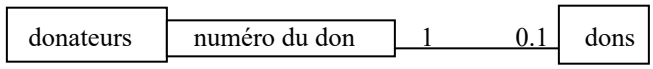


Exemples de qualification

- On peut préciser qu'on ne veut que les dons d'une année donnée :



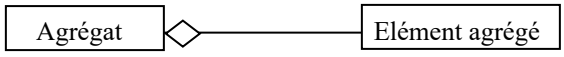
- On peut préciser qu'on ne veut qu'un seul don défini par son numéro de don :



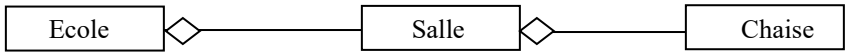
4 - Agrégation et composition

Agrégation

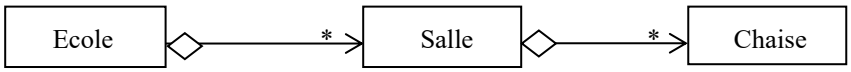
- L'agrégation est une association non symétrique dans laquelle **l'agrégat joue un rôle prédominant** par rapport à ou aux **éléments agrégés**.



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



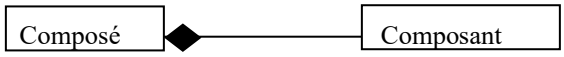
- Une école est composée de plusieurs salles, qui elles-mêmes sont composées de plusieurs fenêtres et plusieurs chaises.
- **En général**, les agrégations sont **navigables uniquement de agrégat vers agrégés** et sont des **associations « * »**



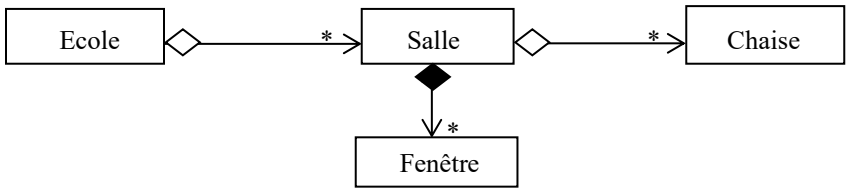
- Dans une agrégation, si on supprime l'agrégat, ça ne supprime pas les éléments agrégés.

Composition

- La composition est un cas particulier d'agrégation.



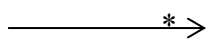
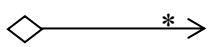
- La composition implique une coïncidence des durées de vie des composants et du composite : la destruction du composite implique la destruction de tous ses composants.
- On a en général les mêmes navigabilité et cardinalités que pour les agrégations.



- 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 : la fenêtre appartient à une salle et une seule.

Remarques

Notion de « rôle prédominant »

- Quelle différence y a-t-il entre :  et  ???
- La notion de « rôle prédominant » est assez subjective.
- Souvent, ce sont les classes « control » (classes de gestion) qui porteront le plus d'agrégations.

Notion de composition

- La notion de composition est plus facile à discerner que celle d'agrégation du fait que la destruction du composant implique celle du composé.

Remarques techniques

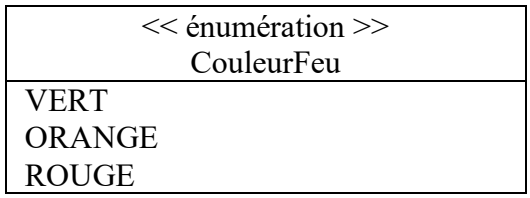
- Techniquement, si les langages orienté objet gère un « ramasse-miettes » (garbage collector). Un ramasse miette s'occupe de supprimer les objets qui ne sont plus référencé par personne.
- Donc si une fenêtre n'est référencée que par sa salle, la destruction de sa salle fera que la fenêtre ne sera plus référencée par personne et donc l'objet sera supprimé.
- Mais quand on supprime la salle, si les chaises ne sont pas référencées par un autre objet, elles seront supprimées par le ramasse-miettes.
- Autrement dit, pour le langage, il n'y a pas de différence entre agrégation et composition et toute les associations sont des compositions !

5 – Enumération

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.



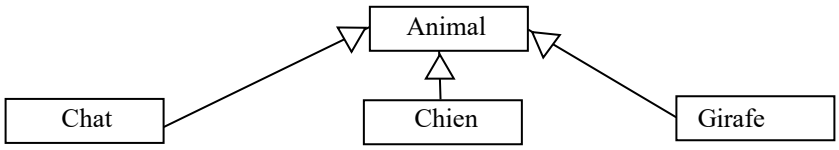
6 – Généralisation - héritage

Principes généraux

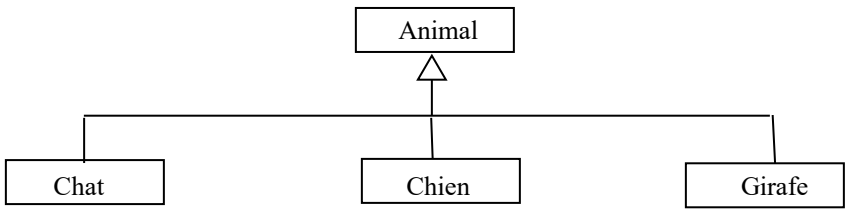
Présentation

- 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 dit que le Chat **hérite** de l'Animal.
- On parle de couple (**classe dérivée – classe de base**) ou (classe spécifique – classe générale) ou (sous-classe – super-classe) ou (**classe enfant – classe parent**) ou (classe fille – classe mère).
- Dès que 2 classes ont **des attributs ou des méthodes en commun**, on retire ces attributs et méthodes de leurs classes de départ pour **créer une nouvelle classe** avec les attributs et méthodes communes. On fait ensuite « **hériter** » les classes de départ de la nouvelle classe.

Formalisme UML



Ou encore :



Précisions

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**.

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 » 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 multiple

- **On évite l'héritage multiple** : une classe enfant n'a qu'un seul parent.
- Ca évite les conflits : si on a plusieurs parents qui ont des attributs ou des méthodes de même nom, lequel ou laquelle l'enfant devra-t-il choisir ?

Héritage d'association

- Une association se traduit finalement par un attribut. Une association sera donc **héritée comme un attribut**.

Principes techniques

Encapsulation

- **La classe enfant hérite** des **attributs**, **associations** et **méthodes de la classe parent**.
- **Le principe d'encapsulation continue à fonctionner** : les attributs, associations ou méthodes privés ne seront pas hérités.
- **L'attribut « protected »**, symbole « # », dans la classe parent permet de limiter la visibilité aux enfants.

Redéfinition

- **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.

Principe de substitution et polymorphisme

- **Un objet spécifique peut être utilisé partout où un objet général est attendu** : partout où on attend un animal on peut mettre un chat. C'est le principe de substitution.
- Ce principe permet le polymorphisme

7 – UML + Relations de dépendance

Principes généraux

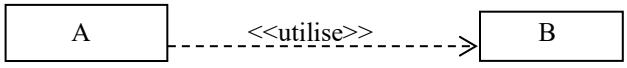
Présentation

- 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.
- Elle traduit une présence qu'on retrouve dans le code.

Exemple

- Si une méthode d'une classe A instancie un objet d'une classe B, alors la classe A « crée » un objet de la classe B.
- Si une méthode d'une classe A reçoit en paramètre un objet de la classe B, alors la classe A « utilise » la classe B.

Formalisme UML



Principales dépendances

<< utilise >> << crée >> << réalise >> sont les principales dépendances.

La réalisation correspond à l'instanciation d'un objet à partir d'une interface.

Type de dépendance : théorique !!!

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.

8 – UML + Classe abstraite - méthode abstraite

Présentation

Principes

- Une classe abstraite est une classe à partir de laquelle on ne peut pas instancier d'objets.
- Les classes abstraites sont en général des super-classes qui contiennent des sous-classes qui, elles, permettront d'instancier des objets.
- Les classes abstraites servent surtout pour la **classification** et le **polymorphisme**.

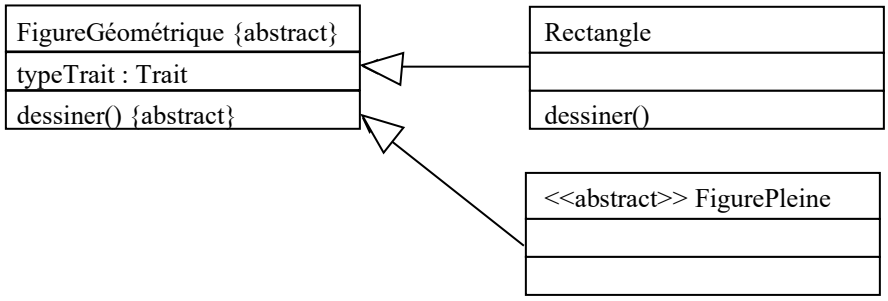
Méthode abstraite

- Une méthode abstraite est une méthode dont le corps (le code) n'est pas défini. Elle n'a que son en-tête de définie.
- Toute classe qui possède une méthode abstraite est une classe abstraite : on ne peut pas instancier d'objet puisqu'une méthode n'est pas définie.

Distinction entre méthode et opération

- **L'opération c'est l'en-tête de la méthode, sans le corps**. Une méthode abstraite ne définit que son opération.
- **La méthode c'est l'en-tête et l'implémentation** du corps (le corps).

Formalisme UML



- La classe « FigureGoémétrique » est abstraite. Elle contient une méthode abstraite (ce n'est pas obligé).
- La classe Rectangle n'est pas abstraite : elle réalise la méthode abstraite « dessiner() ».
- La classe « Figure pleine » est abstraite parce qu'elle hérite de la méthode abstraite « dessiner » mais qu'elle ne la réalise pas. La méthode reste abstraite et donc la classe aussi.
- « réaliser » une méthode abstraite c'est implémenter son corps.
- A noter le formalisme UML : { } ou << >> pour dire que la classe ou la méthode sont abstraites. << >> c'est le symbole commun pour tous les stéréotypes.

9 – UML + Interface

Présentation générale

Principe d'une interface en général

- **L'interface d'un système**, c'est **ce par quoi on accède** au système. Ce sont les **fonctionnalités** qui permettent de l'utiliser.

Présentation théorique

- Les **méthodes publiques** de toute classe définissent **l'interface de la classe**, autrement dit le **système de communication entre la classe et son environnement**, que ce soit un utilisateur ou une autre classe.
- Une classe « **interface** » permet de définir des **spécifications de méthodes** (les opérations = l'en-tête des méthode), autrement dit des **méthodes abstraites**. Ces opérations définissent un système de communication entre toute classe réalisant l'interface et son environnement, que ce soit un utilisateur ou une autre classe.
- L'interface est donc la **partie publique d'une classe** ou d'un **package**. Elle est parfois synonyme de **spécifications**, ou **vue externe**, ou **vue publique**.
- **Autour d'une interface** il y a **celui qui l'utilise**, que ce soit un utilisateur ou une classe, et **celui qui la réalise** : la classe qui implémente les méthodes abstraites de l'interface.

Présentation technique

- Une interface est une classe avec le **stéréotype « interface ».**
- Une interface est **constituée de méthodes abstraites uniquement et pas d'attributs**.
- Une classe qui réalise une interface réalise toutes les méthodes abstraites de l'interface.
- **Réalisation** :
  - Une **classe** peut réaliser **plusieurs interfaces**.
  - Une **interface** peut être réalisée par **plusieurs classes**.
- Utilisation :
  - Une **classe** peut utiliser **plusieurs interfaces**.
  - Une **interface** peut être utilisée par **plusieurs classes**.

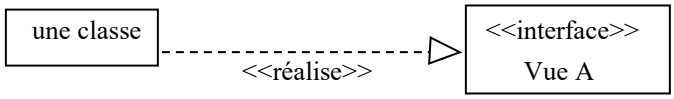
Représentation UML

- Le symbole ——○ montre l'existence d'une interface, mais ne précise pas les opérations.
- On appelle ce symbole « sucette » ou « lolipop »

- La flèche - - - - - -▷ en pointillé signifie que la source réalise la destination.
- La flèche est triangulaire : de type « héritage ». Les méthodes abstraites de l'interface sont héritées par la classe qui réalise l'interface.

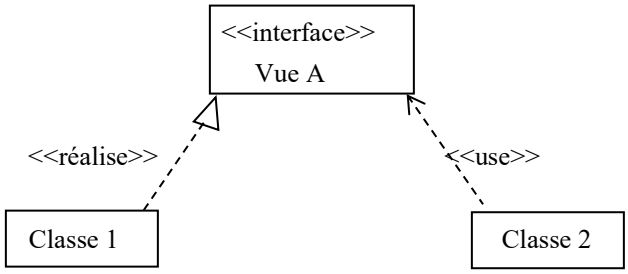
- <<interface>> est le nom d'un stéréotype

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

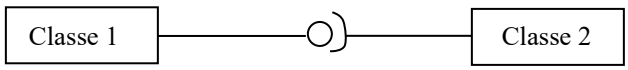


Utilisation des interfaces

- 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 :



- Le <<use>> se traduit concrètement par l'utilisation dans la classe 2 d'un objet de la classe 1. Il y a donc une dépendance entre la classe 2 et la classe 1.
- En utilisant des interfaces plutôt que des classes, on sépare les traitements de leur interface. Ainsi, **on facilite l'évolution des applications** et leur adaptation à différents environnement d'interface.

Méthode de d'analyse des interfaces

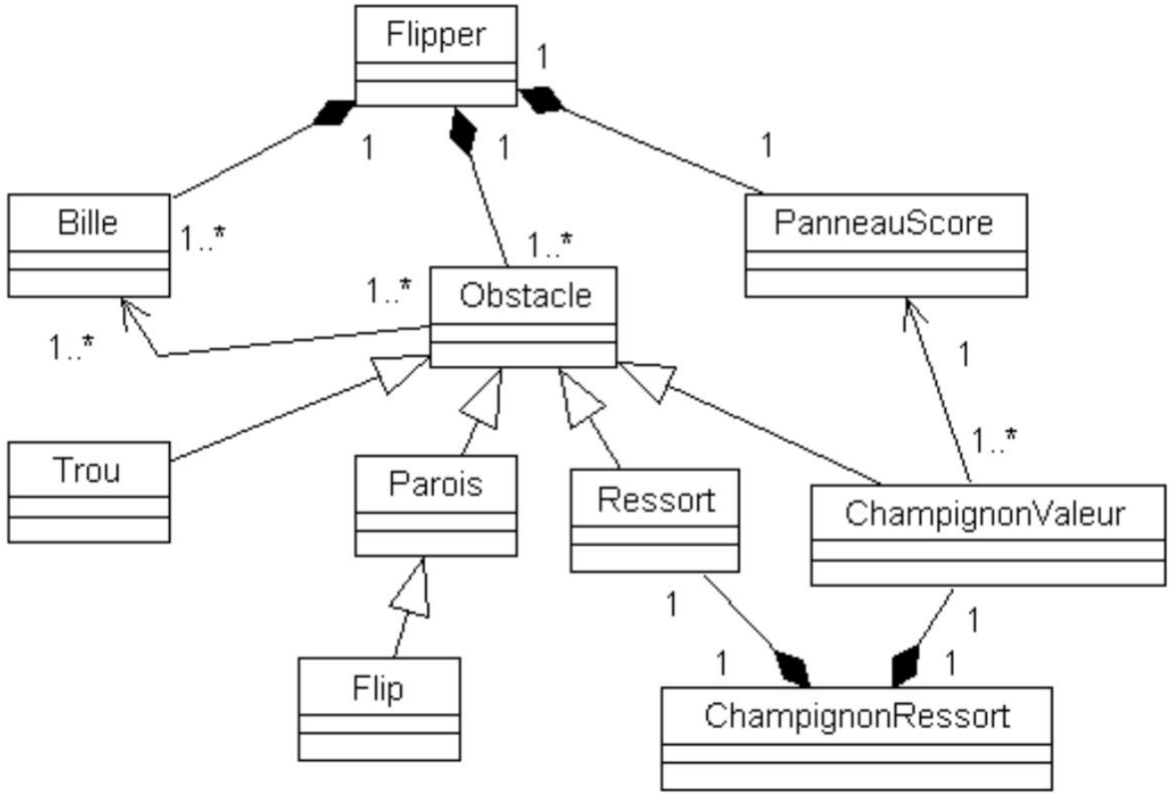
- Pour trouver les interfaces du système, il faut examiner toutes les paires : **acteurs actif – scénario**.
- On a peut dans un premier temps au moins **une classe interface par cas d'utilisation**.
- **Les interfaces se trouvent à un haut niveau d'abstraction. On commence par renseigner les besoins 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.

10 – Exemples de diagrammes

Le flipper

Éléments d'interprétation

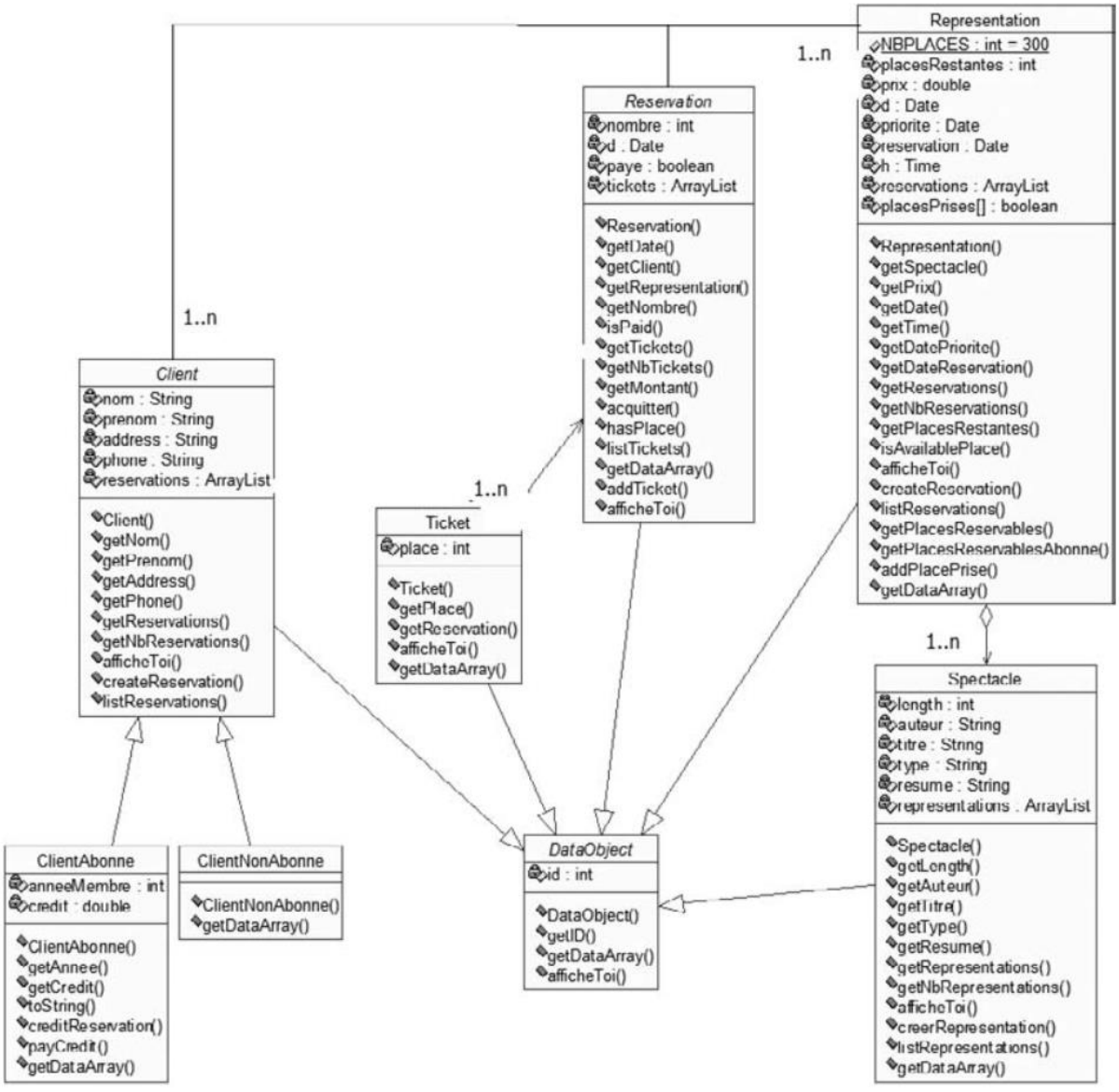
- **5 catégories d'obstacle** peuvent se trouver dans le flipper :
  - la **paroi**,
  - le **trou** qui fait disparaître la bille,
  - le **ressort** qui la fait rebondir,
  - le **champignon** de valeur incrémente le score de la valeur indiquée sur l'obstacle,
  - le **champignon** à ressort qui se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

Éléments d'interprétation

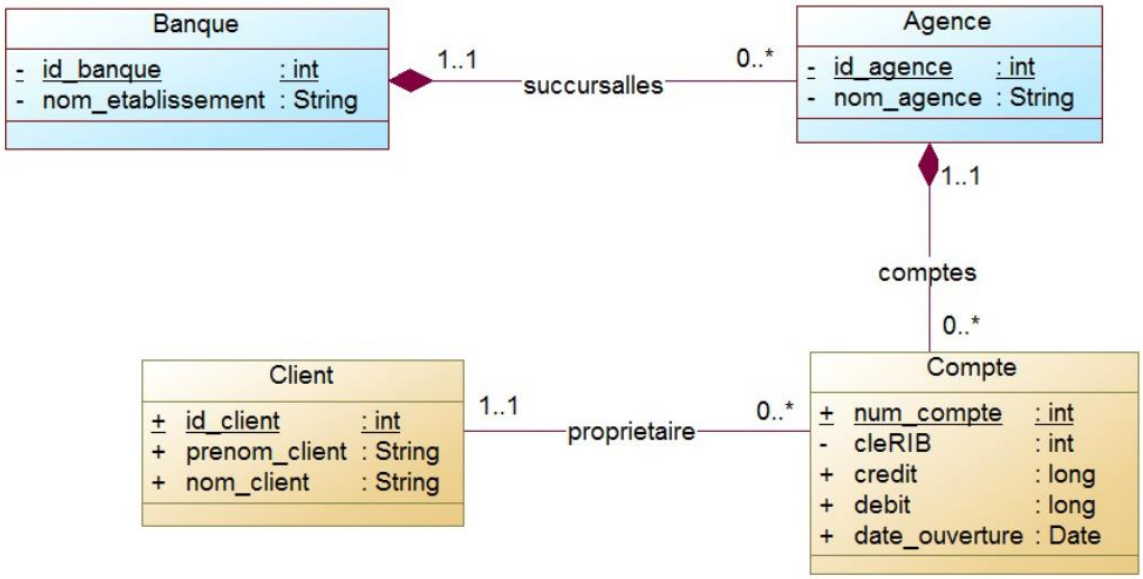
- Dans le modèle ci-dessous, la classe « Reservation » est une classe association.
- On y trouve par exemple : getRepresentation() qui permet de récupérer l'objet « représentation » qui se trouve dans la classe du fait du lien de classe-association.
- Beaucoup de gettter. Pas de setter.
- DataObject est une classe du framwork .NET qui définit un mécanisme indépendant du format pour transférer des données



Banque, agence, compte, client

Éléments d'interprétation

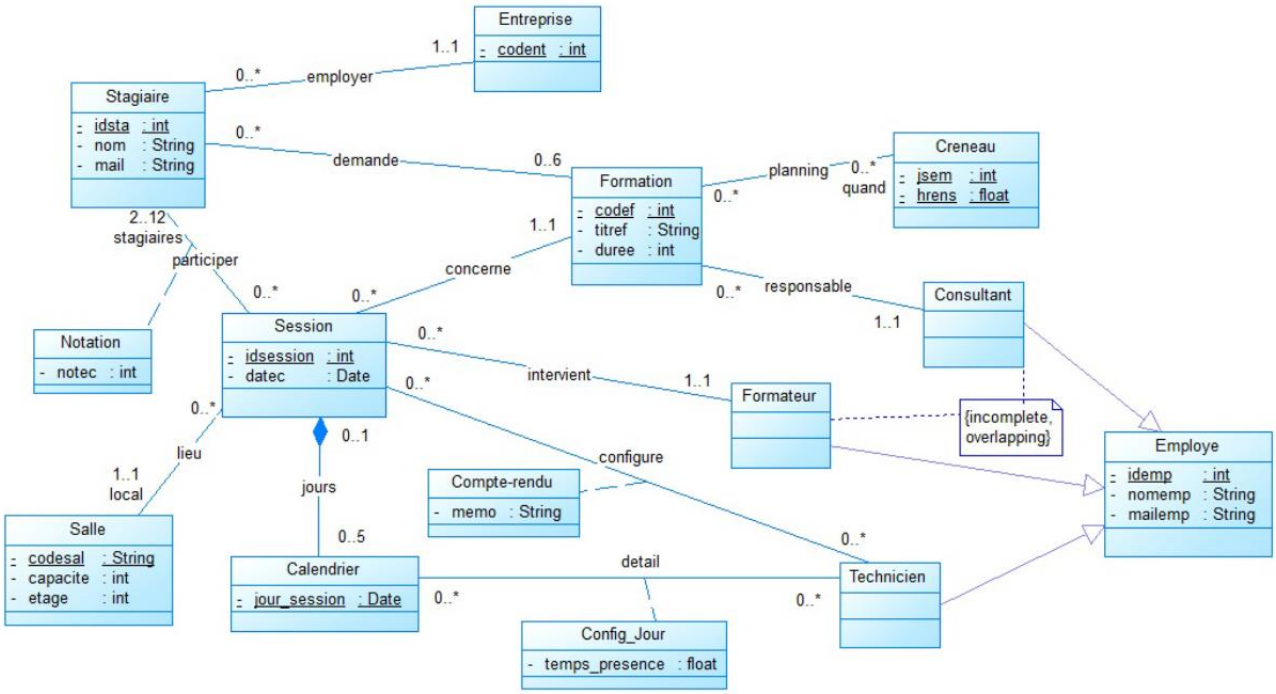
- Utilisation d'une composition entre Banque et Agence (pas de banque, pas d'agence), entre agence et compte (pas d'agence, pas de compte).
- Par contre entre les comptes et les clients, on n'a ni composition, ni agrégation. Si le compte disparaît, le client peut rester. Si le client disparaît on gardera au moins la trace du compte.
- Notez les pluriels : surcursalleS, compteS.
- La nom surcurssales devrait être un rôle côté Agence. La banque a un attribut « sucurssales » qui est une collection d'objet « Agence ».
- Même logique pour les autres noms d'association.



Organisme de formation

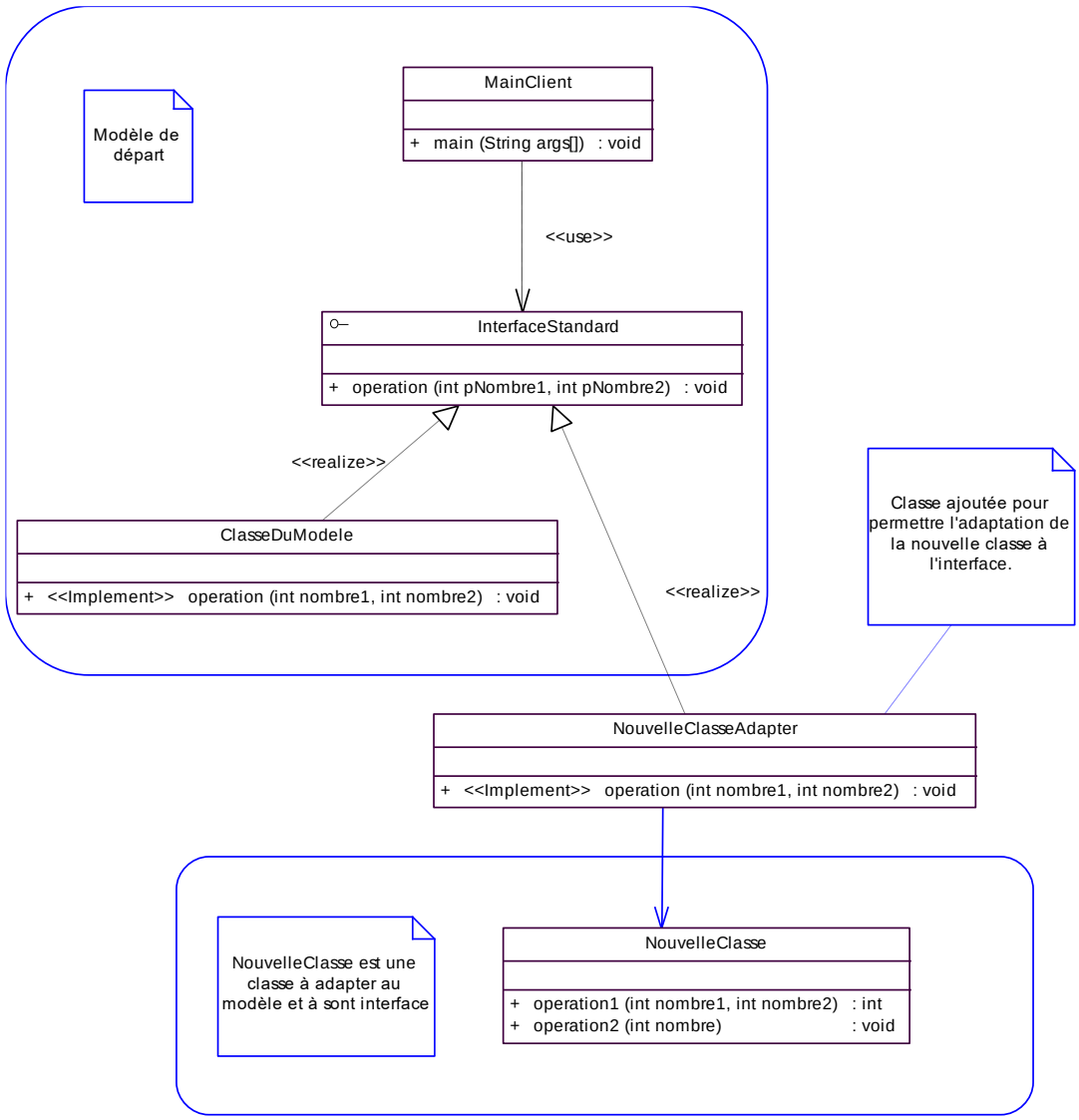
Éléments et difficultés d'interprétation

- On gère des formations et leurs sessions concrètes.
- On arrive à interpréter beaucoup d'éléments. Mais la méconnaissance des données métier et particulièrement des créneaux et du calendrier rend l'interprétation difficile.



Pattern Adapter

- C'est un design pattern.
- On a une dépendance d'utilisation, des réalisations d'interface, une association.
- Les rectangles arrondis n'ont pas de signification UML. Ils permettent ici de montre 2 groupes de classes et la classe qui correspond au patten à part.



11 - Traduction UML vers Java

Classe et association

Association 0..1 – 0..*


```
public class A {
  private int a;
  private B[] les B;

  public int getA() {
    // TODO: implement
  }
}
```

```
public class B {
  private int b;
  private A a;

  public int getB() {
    // TODO: implement
  }
}
```

Association mono-navigable


```
public class A {
  private int a;
  private B[] lesB;

  public int getA() {
    // TODO: implement
  }
}
```

```
public class B {
  private int b;

  public int getB() {
    // TODO: implement
  }
}
```

Agrégation = composition = association simple

Association simple :

Agrégation :

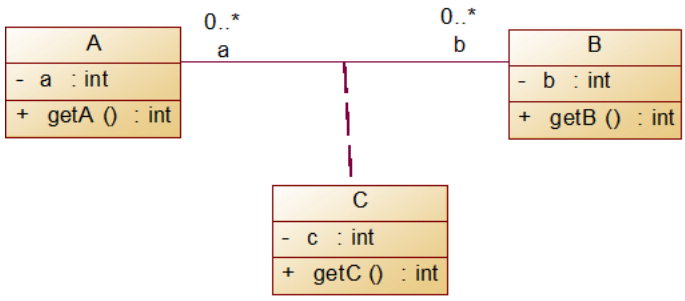
Composition :

Résultats identiques :

```
public class A {
  private int a;
  private B[] lesB;

  public int getA() {
    // TODO: implement
  }
}
```

Classe-association




```
public class A {
  private int a;
  private C[] lesC;

  public int getA() {
    // TODO: implement
  }
}
```

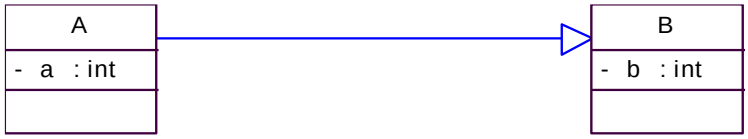
```
public class B {
  private int b;
  private C[] lesC;

  public int getB() {
    // TODO: implement
  }
}
```

```
public class C {
  private int c;
  private A a;
  private B b;

  public int getC() {
    // TODO: implement
  }
}
```

Héritage




```
public class A extends B {
  private int a;

  public int getA() {
    // TODO: implement
  }
}
```

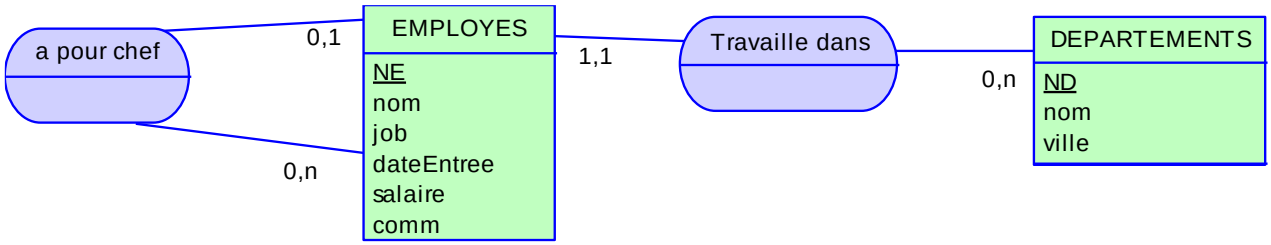
```
public class B {
  private int b;

  public int getB() {
    // TODO: implement
  }
}
```

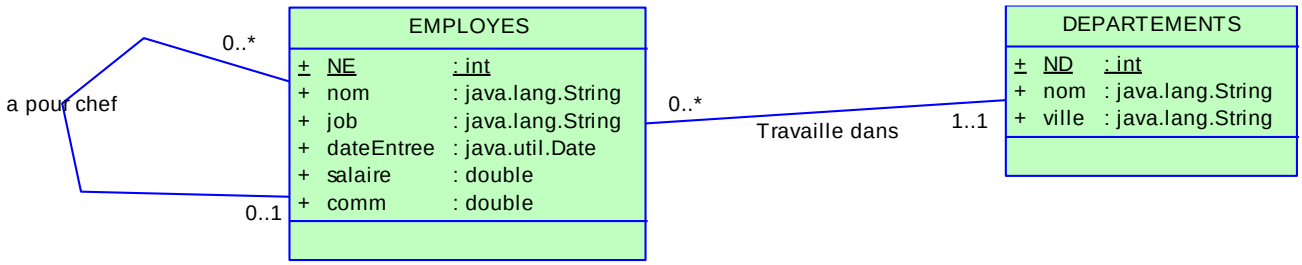
12 - Traduction de MEA en UML

Les employés et les départements

MEA



UML

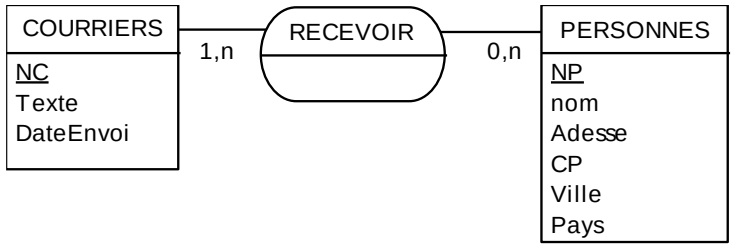


Explications

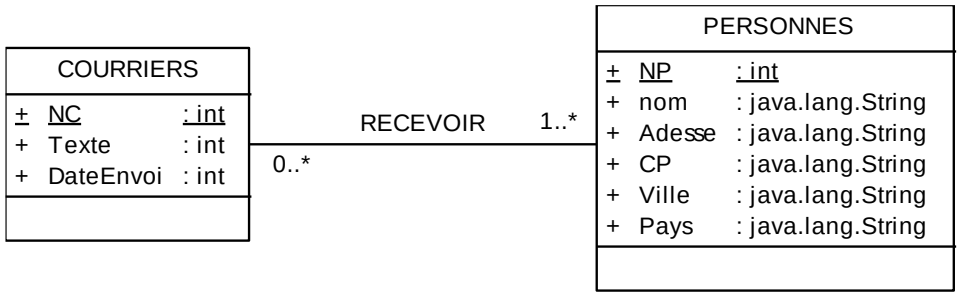
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- La notion de clé primaire n'a pas de signification dans un diagramme de classe. En effet, tout objet (instance 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

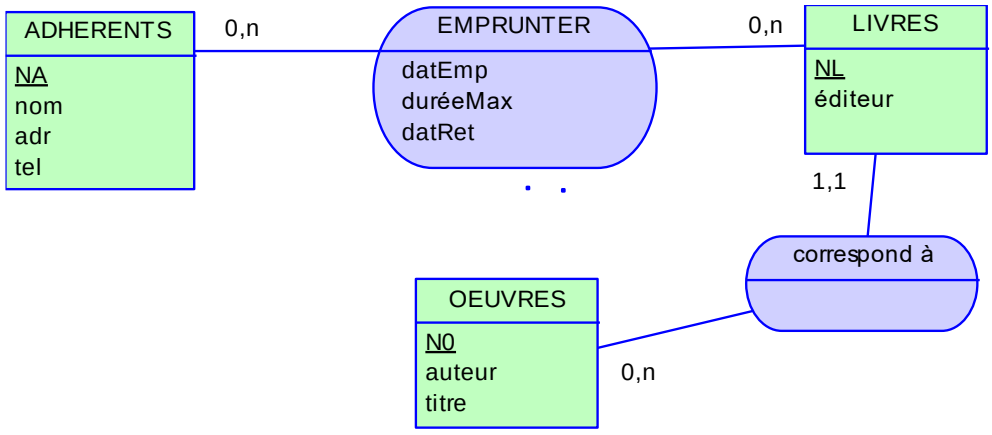


Explications

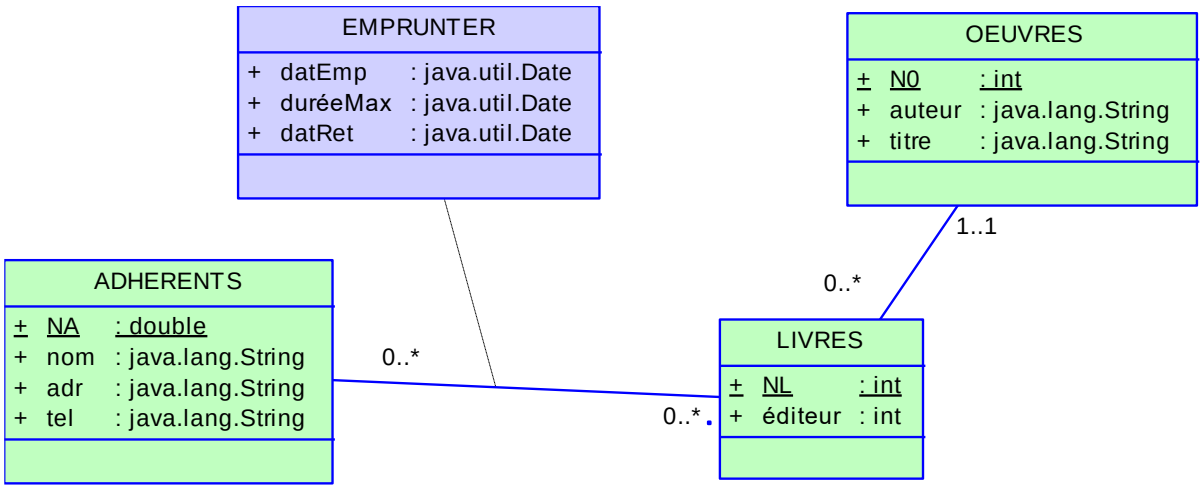
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- Les associations non hiérarchiques sans attributs du MEA sont transformées en association dans le modèle de BD 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

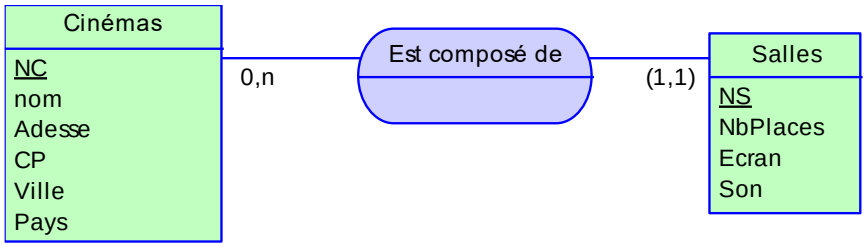


Explications

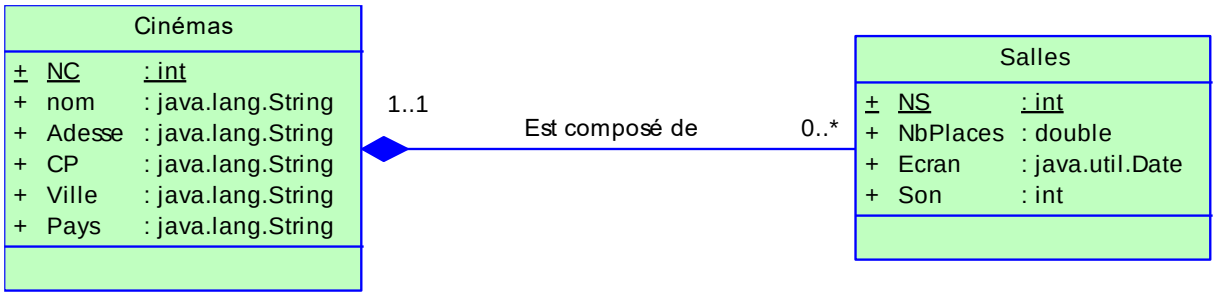
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- En UML, on ne peut pas mettre d'attributs sur les associations. Les associations non hiérarchiques avec attributs 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

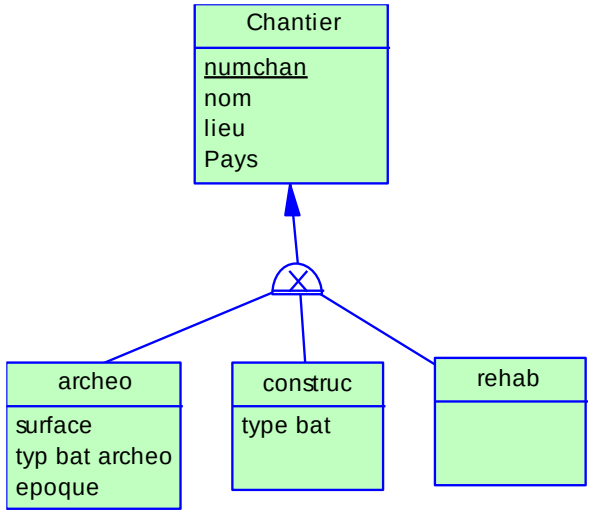


Explications

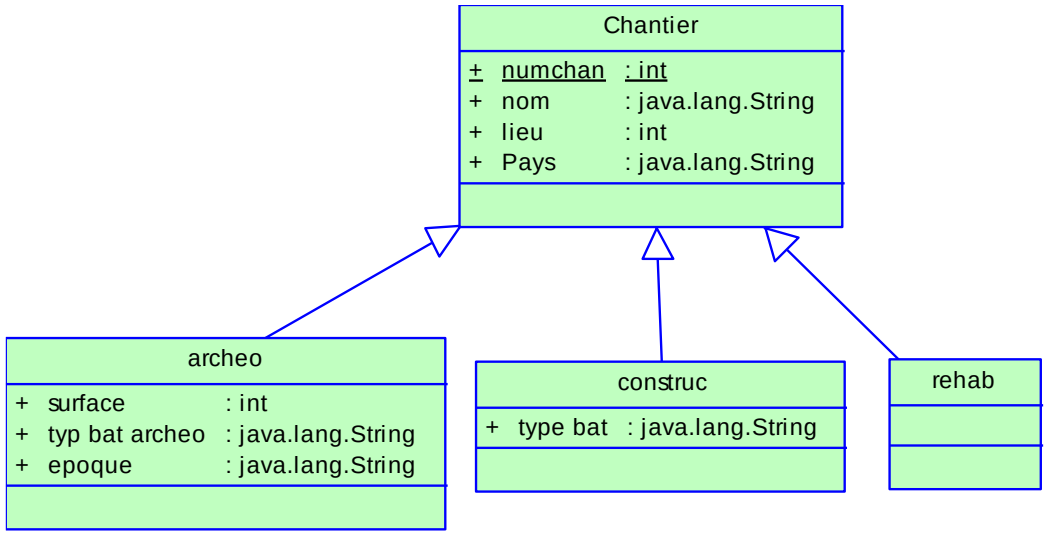
- Tous les attributs sont « + » c'est-à-dire public. En réalité, ils devraient passer « - » , c'est-à-dire privés. Il faut donc faire attention aux résultats des traductions automatiques !
- L'identifiant relatif du MEA est transformé, dans le modèle de BD UML, en une association de composition : losange 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



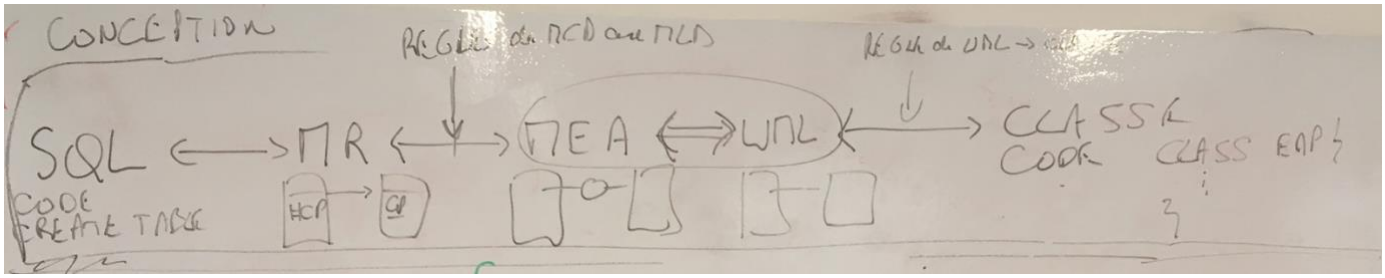
UML



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.

13 – Circulation complète en BD et Classes



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

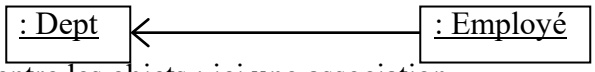
Objets sans noms : : Eleve : Professeur

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.

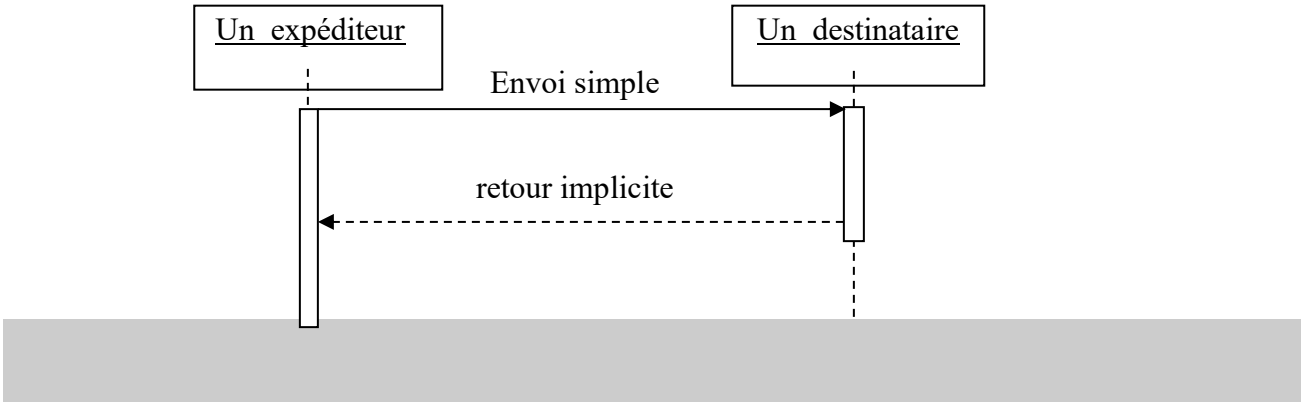
Pour cela, l'objet 2 doit exister dans l'environnement de l'opération appelante de l'objet 1 :

- 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



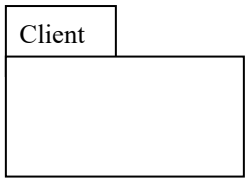
PAQUETAGES ET VUE LOGIQUE

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.).

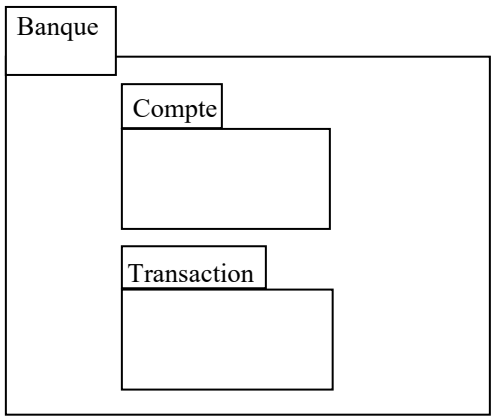


Inclusion de paquetage

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

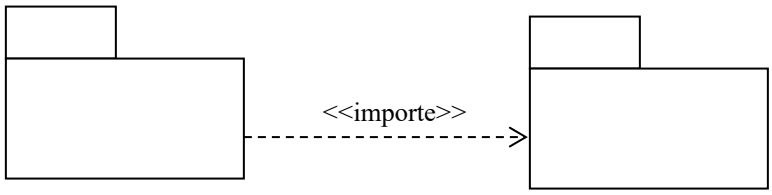
Banque ::Compte désigne le paquetage Compte défini dans le paquetage Banque

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

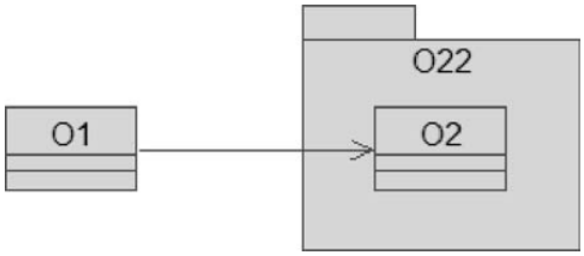
<<importe>> : ajoute les éléments du paquetage destination au paquetage source.

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

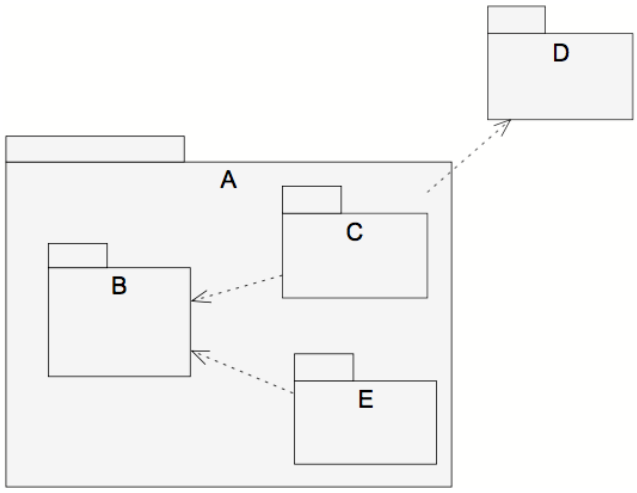


Les éléments du paquetage destination sont visibles dans le paquetage source. Toutefois, il n'y a pas de relation de possession.

La classe O1 utilise la classe O2 :

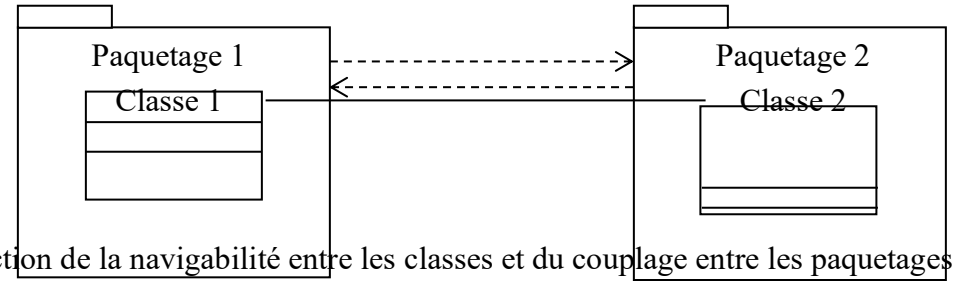


Dépendances multiples :

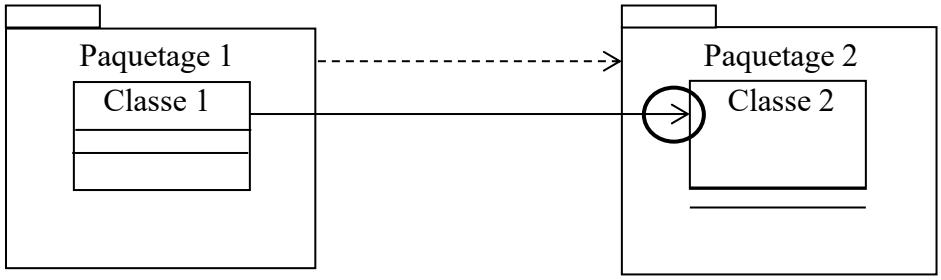


Réduction du couplage entre paquetage

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.

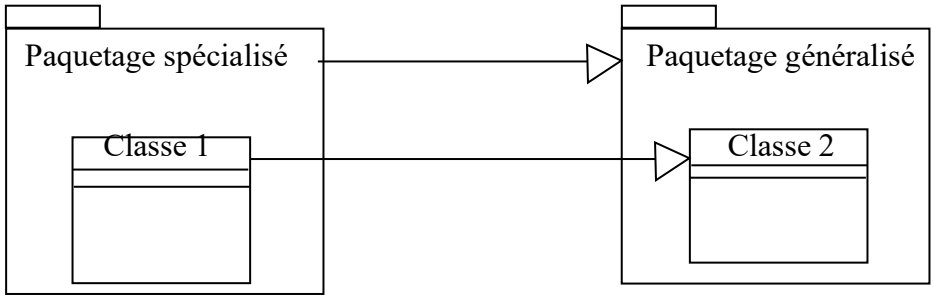
3 - Paquetage généralisé - paquetage spécialisé

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 :



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.