

3^{ème} année licence

Option: SCI

Module: CL2

Chapitre 3: Analyse & Conception

Dr. Meriem Kermani



Année 2017/2018

1.INTRODUCTION

- L'analyse et l'expression de besoins sont très dépendants
- Il y a une ambiguïté entre l'analyse et l'expression de besoins
- L'analyse permet de clarifier les besoins d'une manière détaillée
- Il y a une ambiguïté entre l'analyse et la conception.
- L'analyse et la conception répondent à la question «comment »
- L'analyse se focalise sur l'aspect **métier** des fonctionnalités tandis que la conception se focalise sur l'aspect technique.
- Dans UP, le gros de l'analyse se fait durant la phase d'analyse de besoins et d'élaboration

1.INTRODUCTION

- L'analyse produit un modèle appelé *modèle d'analyse*
- Le modèle est représenté par le *vocabulaire*
- Le vocabulaire désigne les *concepts du domaine. Il concerne tous les acteurs et entités manipulées par ou dans le systèmes*
- Le vocabulaire ne concerne pas uniquement les concepts mais aussi les *relations entre concepts*

1.INTRODUCTION

- Le *modèle d'analyse* est composé de deux sous-modèles : le **vocabulaire** et **les interactions**
- Le vocabulaire désigne les **concepts du domaine**. Il concerne tous les acteurs et entités manipulées par ou dans le systèmes
- Les interactions représentent les activités impliquant plusieurs entités ou acteurs (du vocabulaire) afin de réaliser un objectif métier précis. Généralement, les interactions détaillent les cas d'utilisation.

2.EXEMPLE

- *À l'UC2, pour effectuer un stage, l'étudiant doit chercher un encadreur interne ou dans une entreprise externe et un sujet à réaliser. Une fois le sujet trouvé, l'étudiant le dépose au niveau du service des stages. Ce dernier s'occupe de la validation du sujet en faisant appel à un enseignant de l'UC2 ».*

2.EXEMPLE

- Le vocabulaire devra contenir les concepts suivants : enseignant, étudiant, encadreur, entreprise, sujet, service de stage et stage.
- Il y a plusieurs relation à souligner : l'interaction de recherche de sujet qui implique l'étudiant, l'entreprise, le sujet et l'encadreur. L'interaction de validation qui implique l'étudiant, le service de stages et l'enseignant.

3. RÈGLES POUR CONSTRUIRE LE MODÈLE D'ANALYSE

- Se limiter aux concepts métier, s'éloigner des considérations techniques.
- Le modèle capture une vision globale sur un concept ou un processus, ne pas aller trop dans le détail
- Le modèle doit toujours être compréhensible et utile au client
- Plus le modèle est simple, plus il est meilleur

Diagramme d'objet

1. OBJET

- Rumbaugh définit l'objet comme étant une entité discrète ayant une limite bien définie qui possède un **état** et un **comportement**
- Un objet représente une entité du monde réel
- L'état de l'objet est l'ensemble des valeurs de ses attributs
- Le comportement d'un objet est représenté par les opérations qu'il peut effectuer. Souvent les opérations conduisent à un changement de l'état d'un objet
- L'objet a un identifiant unique qui permet de le distinguer des autres objets.

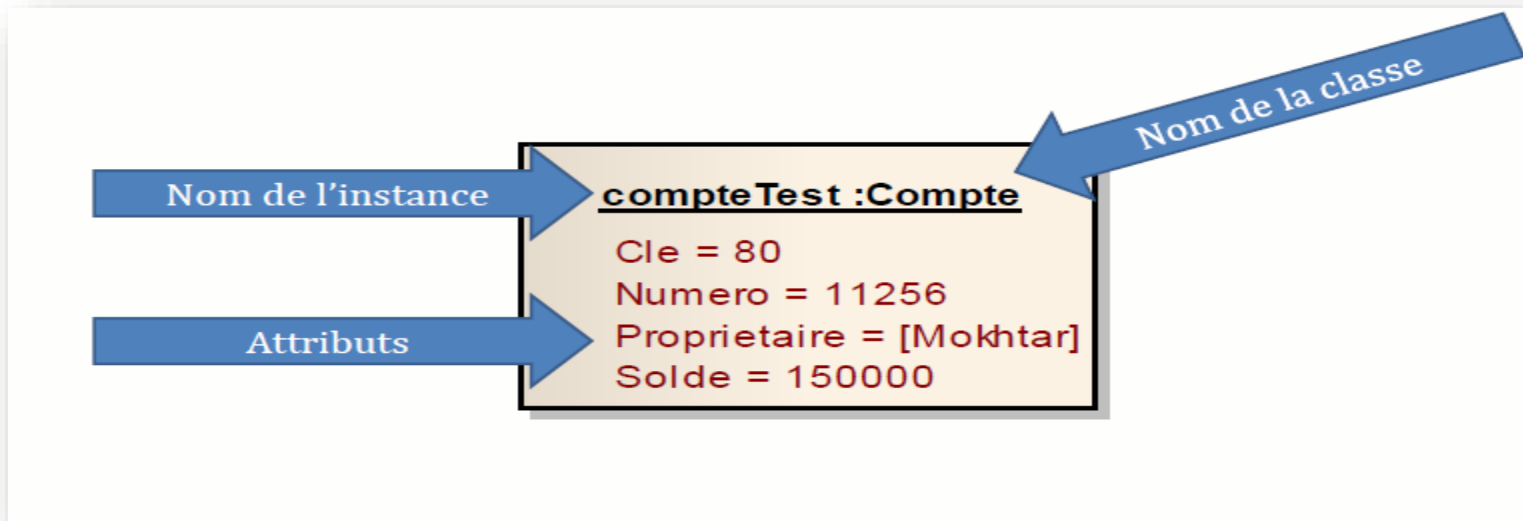
1. OBJET



Etat	Comportement
<ul style="list-style-type: none">• Numéro de série (Identifiant)• Marque• Modèle• Allumée• Mode (Photo / Vidéo)• Connectée à un ordinateur• Liste des photos en mémoire• Capacité• Photo en cours	<ul style="list-style-type: none">• Allumer()• Eteindre()• Connecter()• Filmer()• PrendreUnePhoto()

1.OBJET

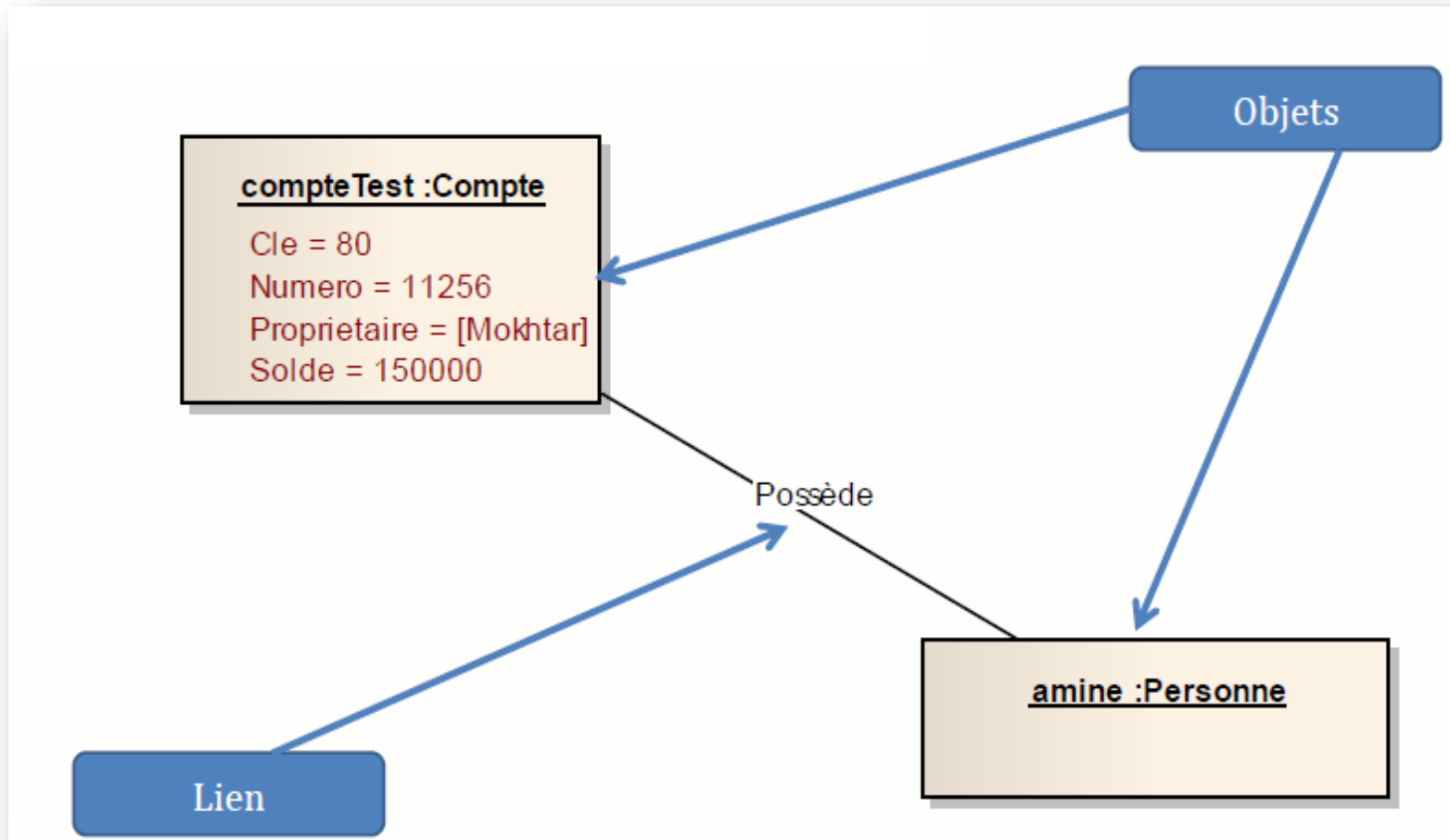
- Le diagramme qui est utilisé pour la représentation des objets est le diagramme d'objets.
- Notation UML des objets



1.OBJET

- Les noms sont écrits en souligné
- Les noms des objets commencent par une minuscule. Si c'est un nom composé, le début de chaque mot suivant commence par une majuscule. Par exemple : *compte11256:Compte ou clientFavori:Client*. Le symbole « : » sépare le nom de l'instance du nom de sa classe.
- Le nom de l'objet peut être anonyme (ne comporte que le nom de la classe. Par exemple : *:Compte ou :Client*).
- L'objet peut ne pas avoir de classe : par exemple *amine ou compteTest*.

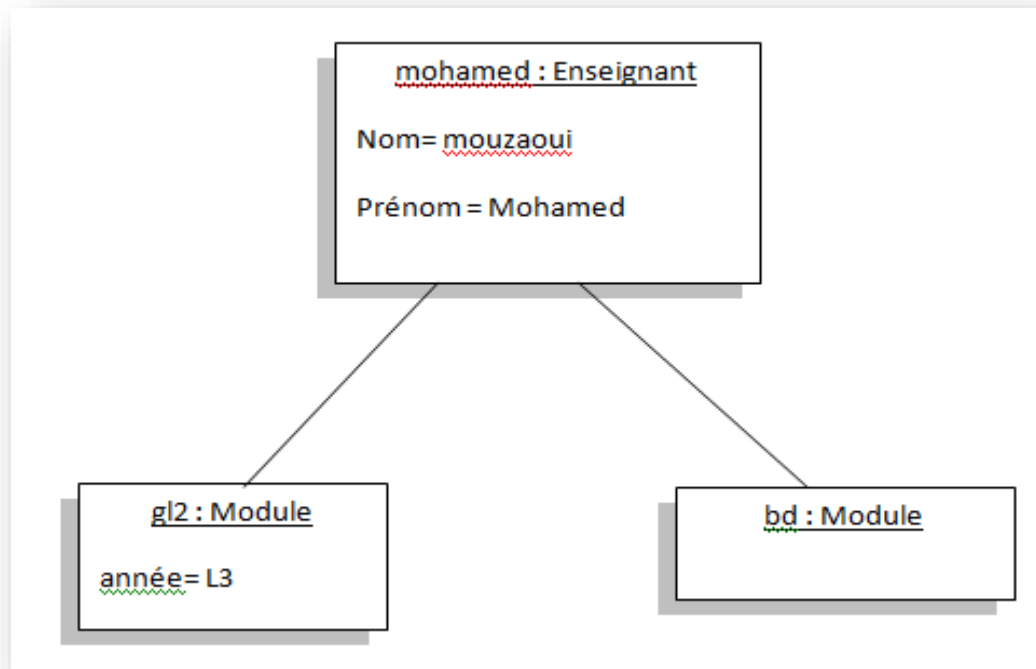
1. OBJET



1. OBJET

- Un diagramme d'objets présente des objets et leurs relations à un instant T
- Le diagramme d'objets est idéal pour donner des exemples sur des situations particulières
- Les liens bidirectionnels entre A B quand A et B s'envoient des messages
- Un lien unidirectionnel entre A vers B indique que A peut envoyer un message vers B mais pas l'inverse

2. OBJET/ EXEMPLE



OBJET

- Les diagrammes d'objets sont utilisés pour donner des ***exemples sur des situations complexes mettant en relation plusieurs concepts***
- Les diagrammes d'objet sont facultatifs et souvent les diagrammes de classes sont ***suffisants pour décrire le domaine.***

DIAGRAMME DE CLASSE

1. DIAGRAMME DE CLASSE

- Rumbaugh définit la classe comme étant un descripteur d'un ensemble d'objets qui partagent les mêmes attributs, méthodes, relations et comportement
- La classe est le modèle d'un ensemble d'objets similaires
- Un objet appartient à une seule classe
- La classe définit la structure d'un objet (aspect statique) , son comportement (aspect dynamique) et ses relations

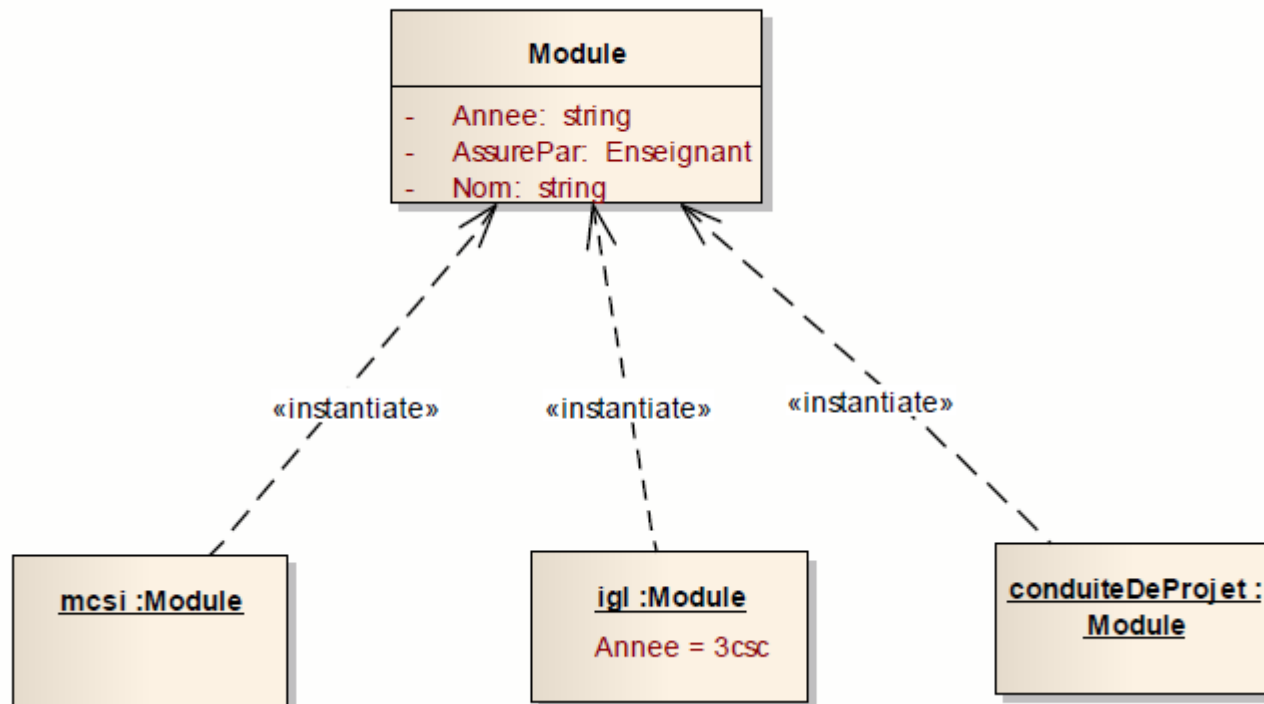
1. DIAGRAMME DE CLASSE

- Le diagramme des classes est un diagramme **structurel (statique) qui permet de représenter :**
 - les classes (*attributs + méthodes*)
 - les associations (*relations*) entre les classes.
- Le diagramme de classes est le plus important des diagrammes UML, c'est le seul qui soit obligatoire lors de la modélisation objet d'un système.

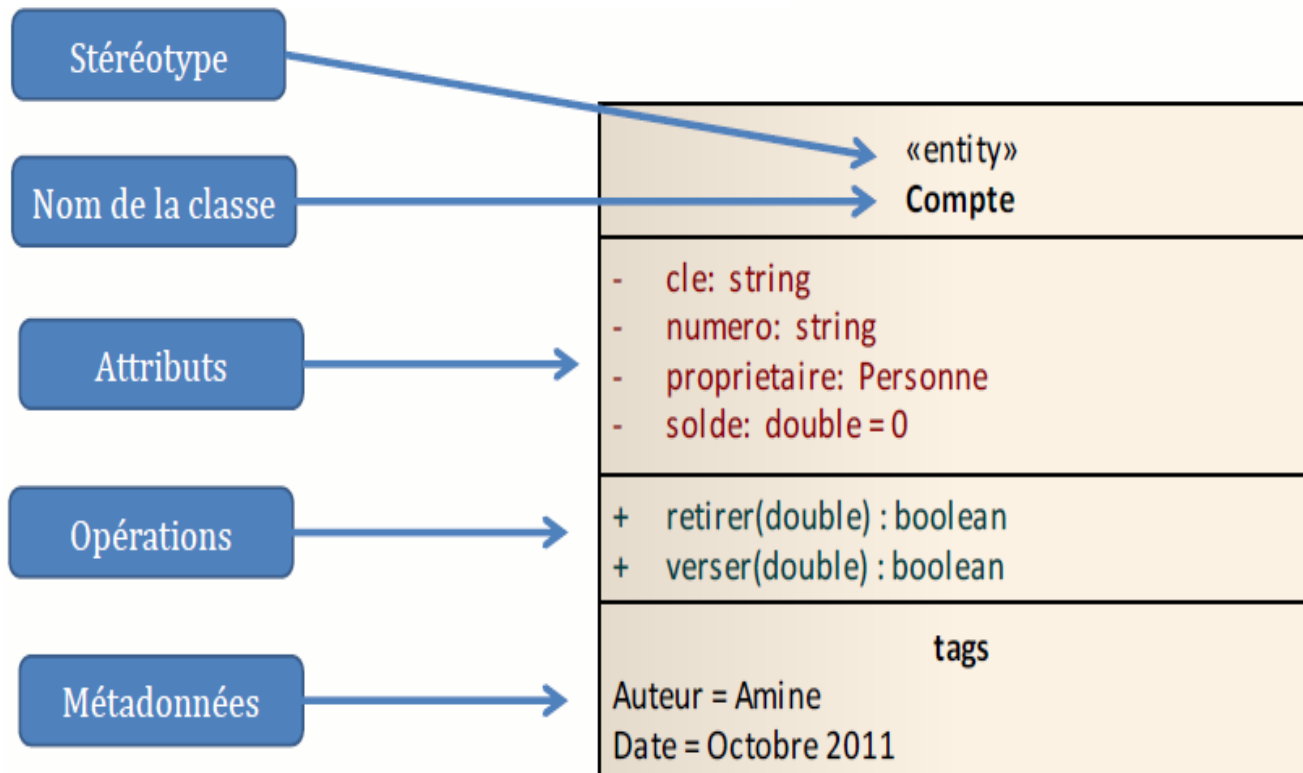
1. DIAGRAMME DE CLASSE

L'objet *instancie la classe*

UML définit l'instanciation comme une relation de dépendance avec le stéréotype « instantiate »



1. DIAGRAMME DE CLASSE



2. VISIBILITÉ

- La visibilité s'applique aussi bien aux attributs qu'aux opérations
- Durant la phase d'analyse, *la visibilité n'est pas importante*
- Les langages de programmation peuvent avoir une interprétation *différente de la visibilité*

2. VISIBILITÉ

Visibilité	Nom	Description
+	Visibilité publique	Accès depuis la même classe et à l'extérieur de la classe
-	Visibilité privée	Accès depuis la même classe uniquement
#	Visibilité protégée	Accès depuis la même classe et les classes descendantes
~	Visibilité paquet	Accès depuis la même classe et toutes les classes appartenant au même paquet

3. ATTRIBUTS

- Les attributs sont formulés en utilisant la syntaxe suivante :

*boolean, byte, char,
double, float, int, long,
short a*

- *Visibilité Nom_Attribut : type [multiplicité]
= valeur_initiale*

3. ATTRIBUTS /MULTIPLICITÉ

- La multiplicité indique les attributs multiples.
- Les multiplicités sont équivalentes aux tableaux mais ont plus de sémantique.
- Exemple 1 : `int valeurs[7]`. La classe contient ***exactement 7 valeurs***.
- Exemple 2 : `int valeurs[2..*]` : la classe contient ***au moins 2 valeurs***.
- Exemple 3 : `int valeurs[2..7]` la classe contient ***au minimum 2 valeurs et au maximum 7 valeurs***.
- Exemple 4 : `int valeurs [0..1]` la classe contient ***une valeur ou null***.

EXEMPLE

Etudiant

```
+ matricule : int
# nom : string
# prenom : string
~modules : int [2..9]
- age :int = 18
- chambreAffectee : Chambre [0..1]
+ tuteursDeSuivi : Tuteur [2]
+ nombreEtudiants : int
+ inscriptions : Inscription [1..*]
```

4. OPÉRATION

- Les opérations sont formulés en utilisant la syntaxe suivante
- *Visibilité Nom_Operation (direction
nom_paramètre: type = valeur_défaut,...) :
type_retour*

4. OPÉRATION/ *DIRECTION DES PARAMÈTRES*

- Devant le nom du paramètre, **il est possible** d'indiquer par un mot clé (**in**, **out**, **inout**, **return**), la direction dans laquelle celui-ci est transmis.

In	La valeur du paramètre est transmise à l'appel de la méthode (par l'appelant de la méthode) et ne peut pas être modifiée (c'est le comportement par défaut si aucune direction n'est spécifiée).
Out	La valeur finale du paramètre est transmise au retour de l'appel de la méthode (à l'appelant de la méthode).
inout	Un paramètre d'entrée / sortie. p est utilisé par l'opération et sa valeur peut être changée par l'opération.
Return	Le paramètre est un paramètre de retour. L'opération doit retourner une valeur.

4. OPÉRATION/ *DIRECTION DES PARAMÈTRES /* *EXEMPLE*

- Considérons la méthode *régler* (*heures : int, minutes : int*) : *void* de la classe *Horloge* de. L'appelant de la méthode veut affecter les attributs *heures et minutes* avec des valeurs qu'il va donner.
- La direction des paramètres sera alors *in* : *régler(in heures : int, in minutes : int) : void*
- Considérons maintenant la méthode *getTime*(*heures : int, minutes : int*) : *void* de la classe *horloge*.
- L'appelant de la méthode veut récupérer les valeurs des attributs *heures et minutes*. La direction des paramètres sera alors *out* : *getTime(out heures : int, out minutes : int) : void*

4. OPÉRATION/ *DIRECTION DES PARAMÈTRES / EXEMPLE*

Horloge

-heures : int =0

-minutes: int = 0

+ régler (in heure: int=0 ; in minute int =0) : void

+getTime (out heures: int ; out minute: int)

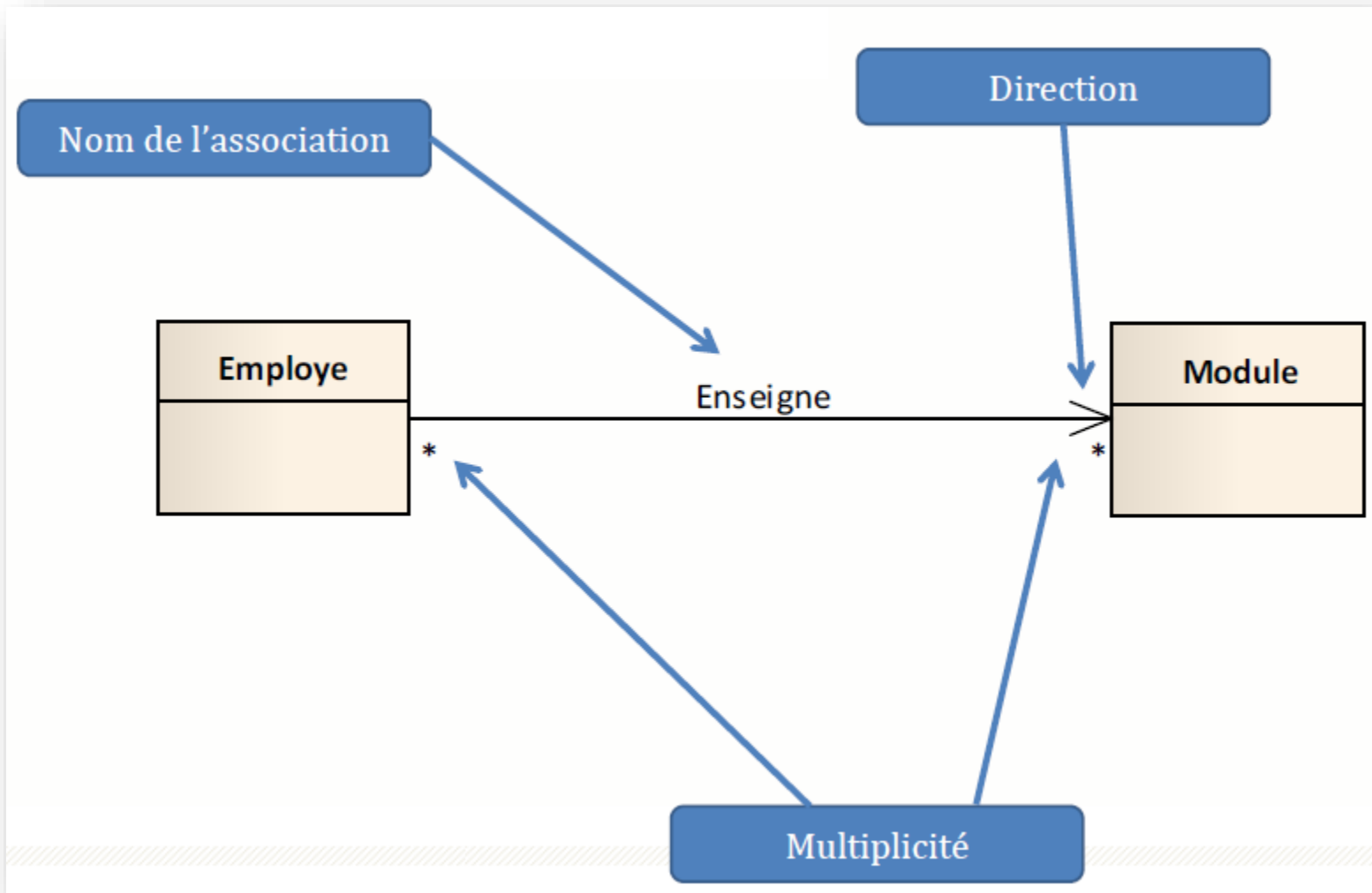
5.RELATION ENTRE CLASSES / ASSOCIATION

- L'association indique qu'une classe est en relation avec une autre.
- La ligne de vie des deux objets concernés ne sont cependant pas associés étroitement (un objet peut être détruit sans que l'autre le soit nécessairement).
- L'association est représentée par un simple trait continu, reliant les deux classes. Le fait que deux instances soient ainsi liées permet **la navigation** d'une instance vers l'autre, et vice versa (chaque classe possède un attribut qui fait référence à l'autre classe).

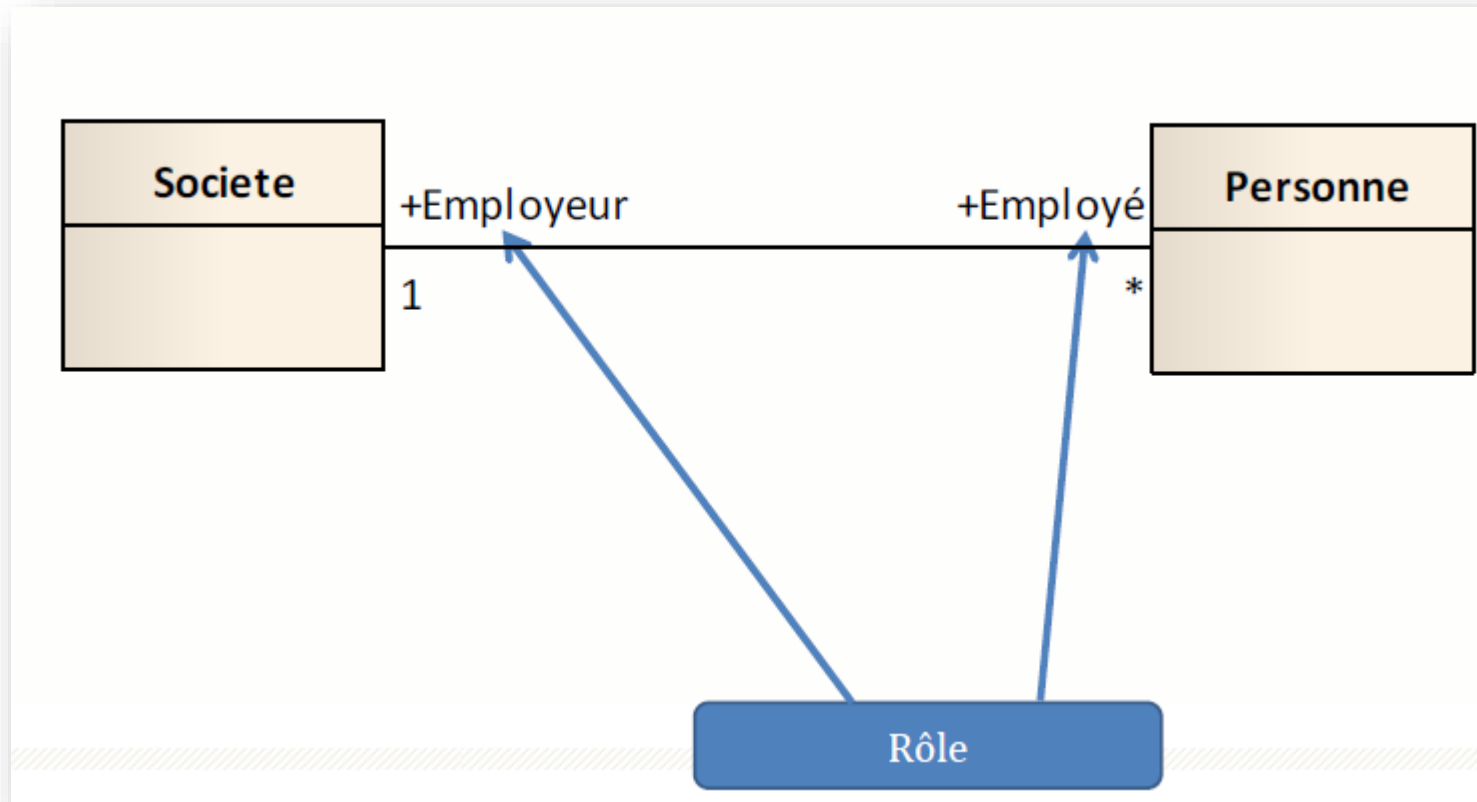
5.RELATION ENTRE CLASSES / ASSOCIATION

- On peut aussi donner un nom de chaque coté de l'association, afin de nommer le rôle de chacun.
- En théorie, l'association se lit de gauche à droite. Si le verbe se lit dans l'autre sens, il faut l'indiquer.
- Des cardinalités expriment le nombre d'instances en jeu dans la relation
- *Il n'est pas recommandé d'utiliser les noms et les rôles. Utiliser soit le nom soit le rôle.*

5. RELATION ENTRE CLASSES / ASSOCIATION



5. RELATION ENTRE CLASSES / ASSOCIATION



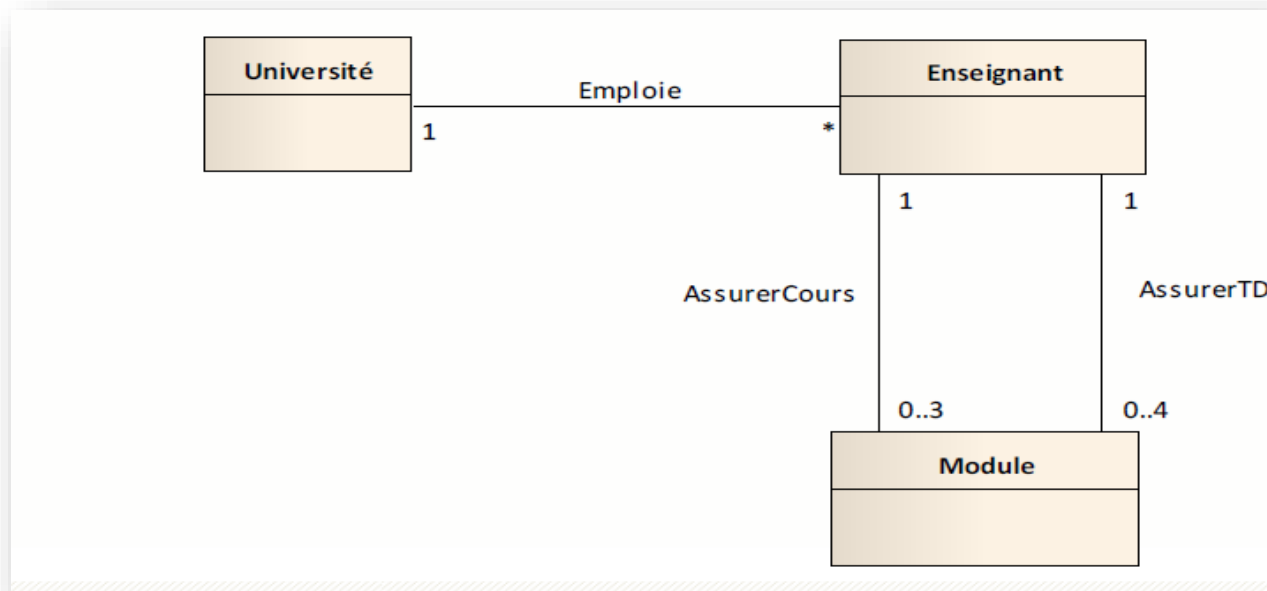
5.RELATION ENTRE CLASSES / ASSOCIATION

Multiplicité	Signification
<i>0..1</i>	<i>Zéro ou 1</i>
<i>1</i>	<i>Exactement 1</i>
<i>0..*</i>	<i>Zéro ou plusieurs</i>
<i>*</i>	<i>Zéro ou plusieurs</i>
<i>1..*</i>	<i>1 ou plusieurs</i>
<i>1..9</i>	<i>1 à 9</i>
<i>1..5, 8, 20..*</i>	<i>1 à 5, exactement 8 ou plus de 20</i>

5. RELATION ENTRE CLASSES / ASSOCIATION

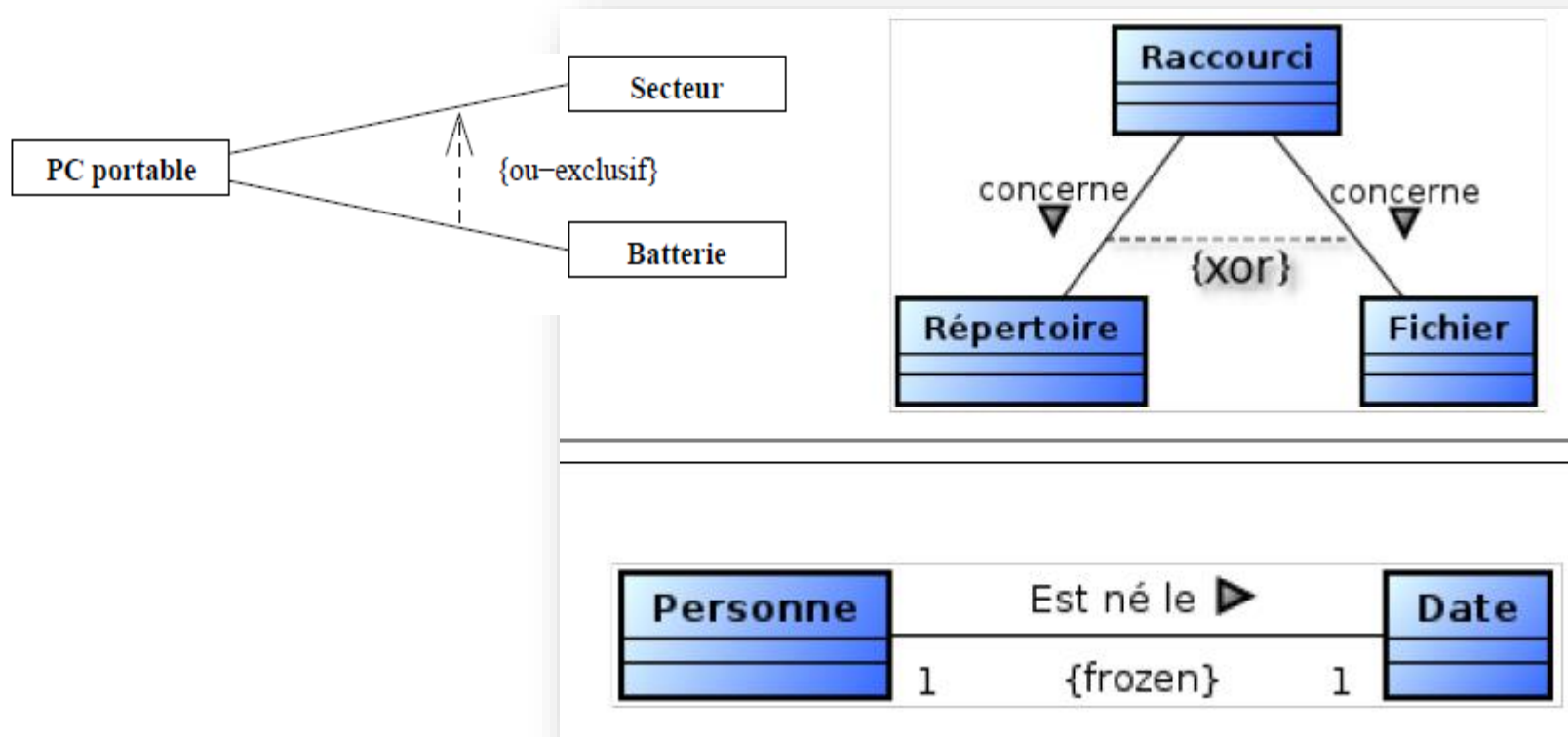


- Deux instances d'une même classe peuvent avoir des rôles différents



5. RELATION ENTRE CLASSES / ASSOCIATION

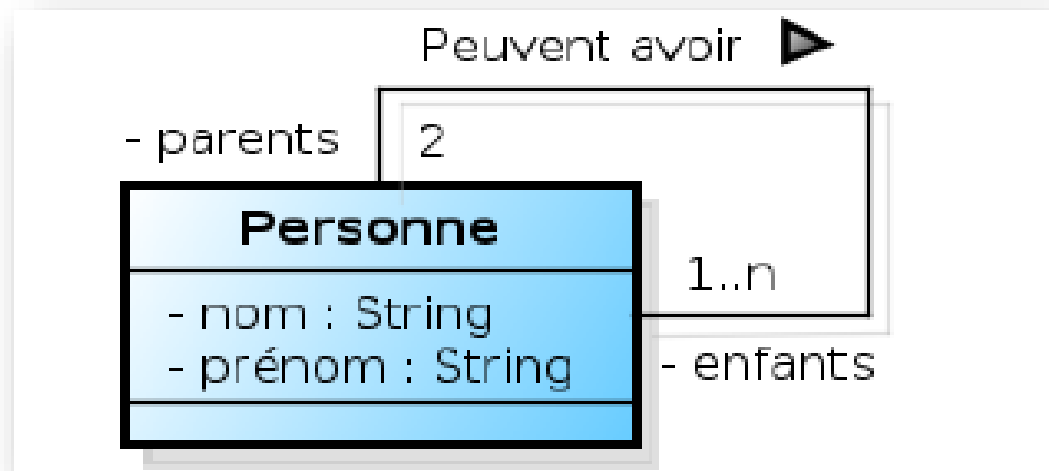
- *Contrainte sur une association*



5.RELATION ENTRE CLASSES / ASSOCIATION

Association réflexives (ou récursive) :

Une association qui lie une classe avec elle-même est une association réflexive.

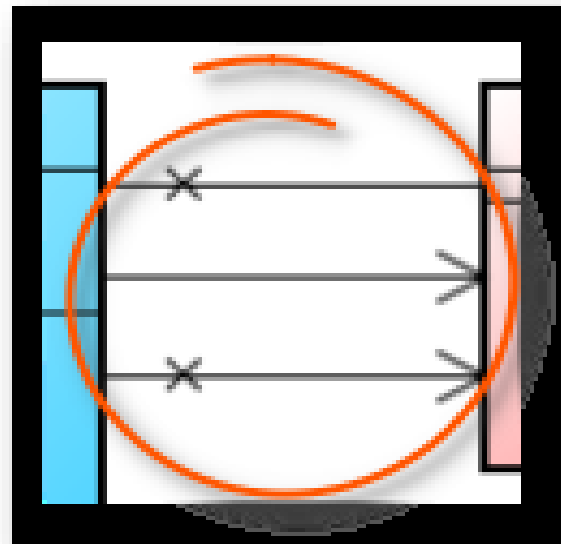


5.RELATION ENTRE CLASSES / ASSOCIATION

- **La navigabilité**
- Les associations possèdent une navigation bidirectionnelle par défaut, c'est-à-dire qu'il est possible de déterminer les liens de l'association depuis une instance de chaque classe d'origine. Cela suppose que chaque classe possède un attribut qui fait référence à l'autre classe en association.
- Une navigation bidirectionnelle est du coup plus complexe à réaliser ; il convient de l'éviter dans la mesure du possible.

5.RELATION ENTRE CLASSES / ASSOCIATION

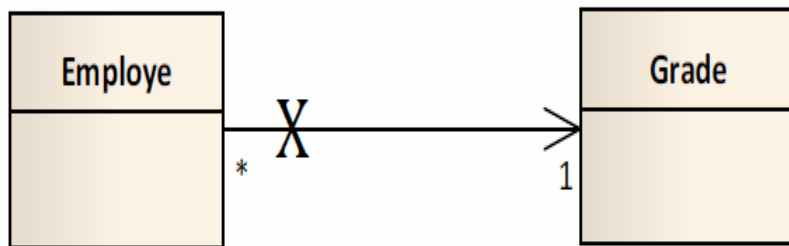
- La navigabilité
- Il est plus fréquent d'avoir besoin d'une navigabilité *unidirectionnelle*. Dans ce cas, une seule classe possède un attribut qui fait référence à l'autre classe.



5. RELATION ENTRE CLASSES / ASSOCIATION

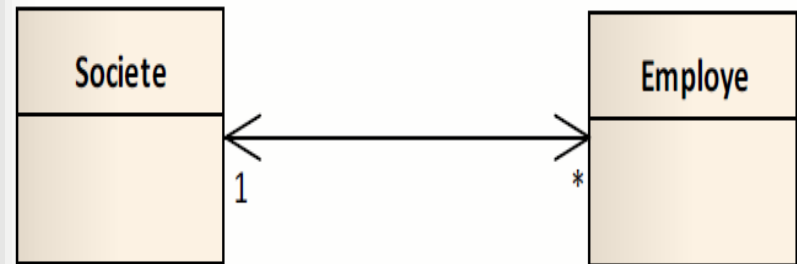
○ La navigabilité/ Exemple

Employé vers Grade est navigable



Grade vers Employé n'est pas navigable

Société vers Employé est navigable



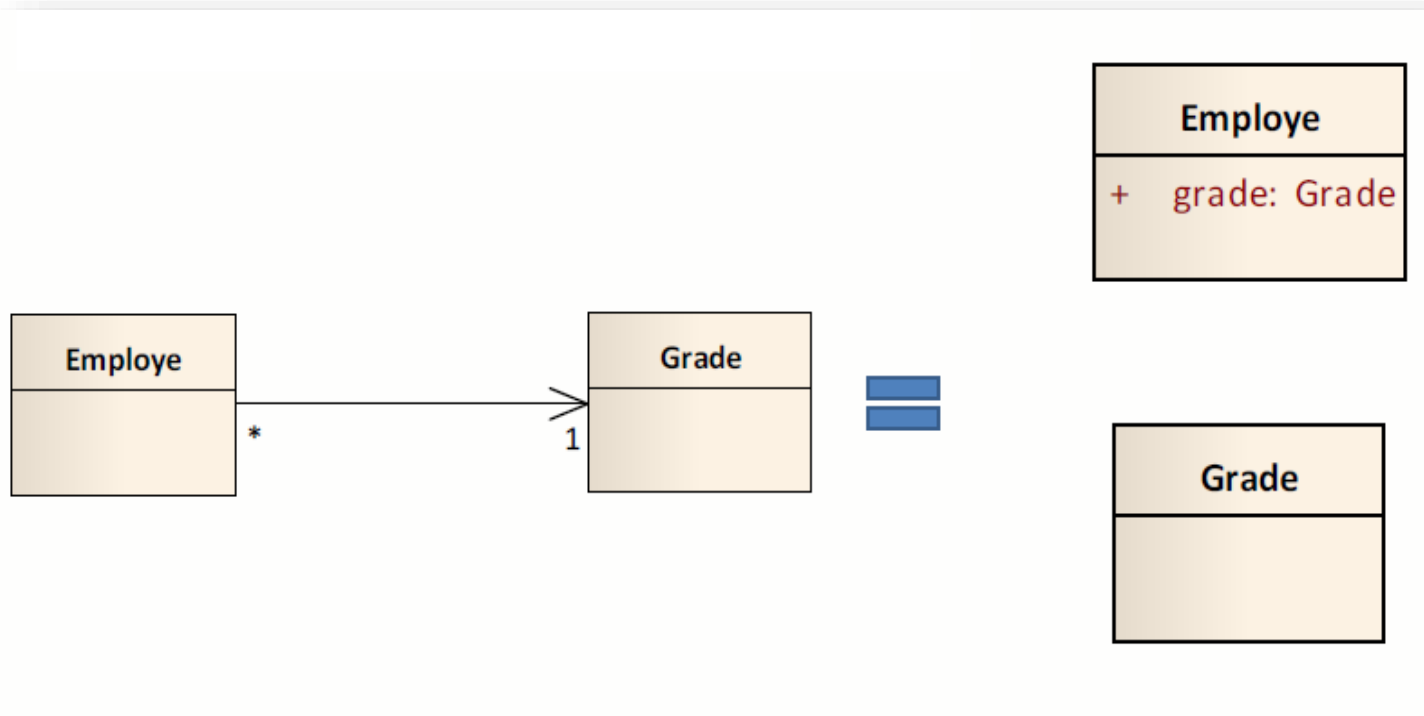
Employé vers Société est navigable

5.RELATION ENTRE CLASSES / ASSOCIATION

- **Attributs et association**
- Les attributs peuvent être un autre moyen de représenter une association
- Les associations à multiplicité multiple peuvent être représentées par des tableaux ou des collections
- Lors de la génération de code, les associations sont générées en tant qu'attributs

5. RELATION ENTRE CLASSES / ASSOCIATION

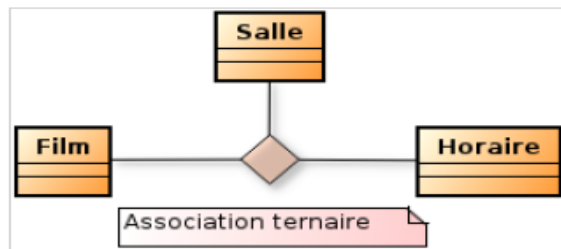
Attributs et associations / Exemple



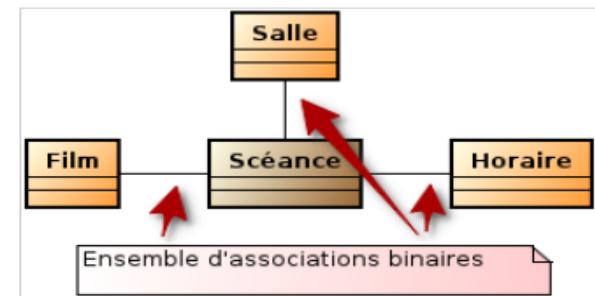
5. RELATION ENTRE CLASSES / ASSOCIATION

- Association n-aire

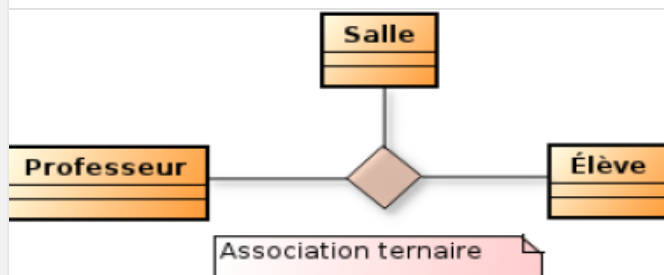
- Il est possible que **plusieurs Classes participent** à l'association. Ce n'est alors plus une association binaire, mais **n-aire**. On l'indique par un losange.



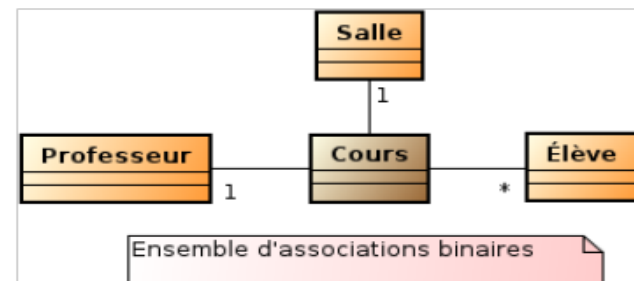
À remplacer par :



Exemple 2 : Un cours peut correspondre à l'association ternaire de 3 classes.



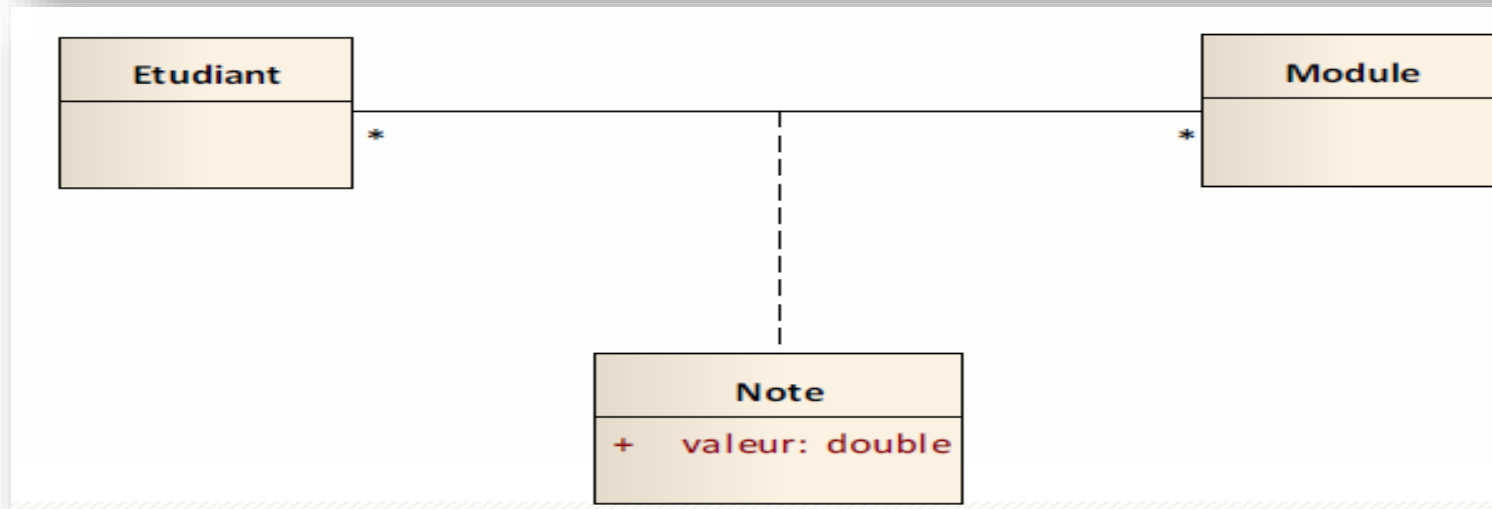
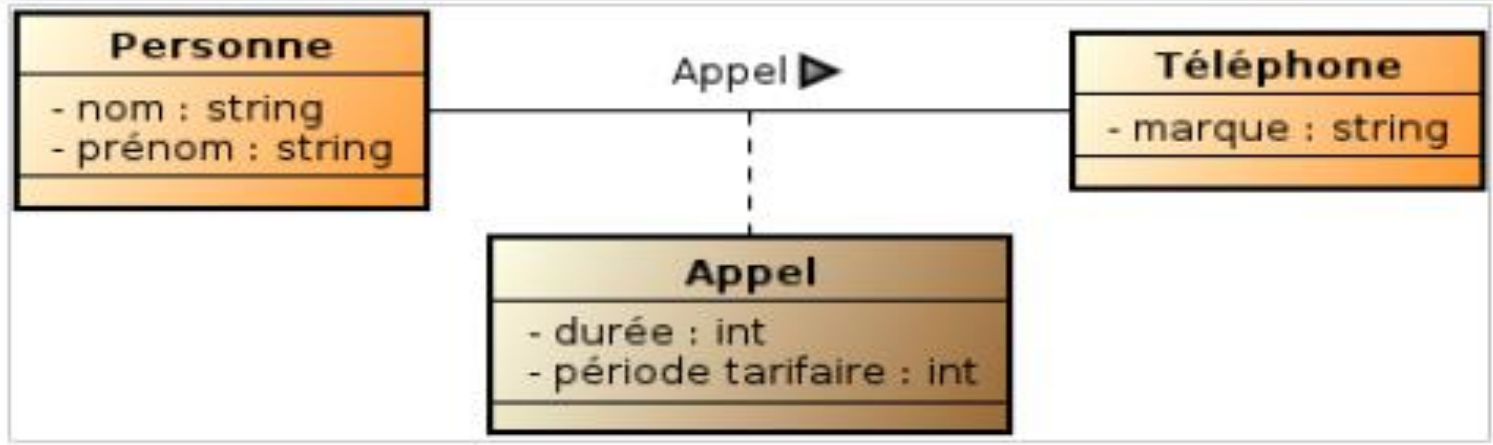
À remplacer par :



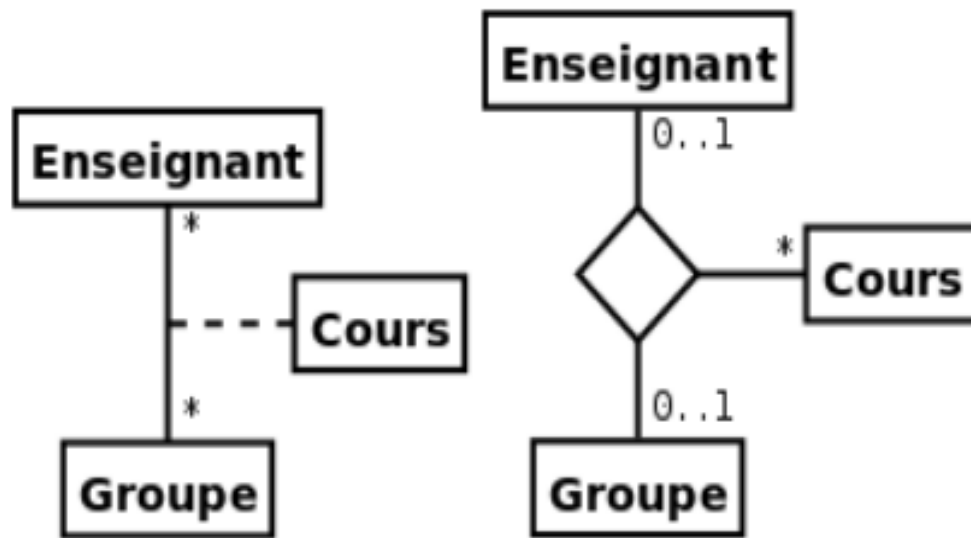
5.RELATION ENTRE CLASSES / CLASSE D'ASSOCIATION

- Une association peut apporter de nouvelles informations (*attributs et méthodes*) **qui n'appartiennent à aucune des deux classes** qu'elle relie et qui sont spécifiques à l'association. Ces nouvelles informations peuvent être représentées par une nouvelle classe attachée à l'association via un trait en pointillés.

5. RELATION ENTRE CLASSES / CLASSE D'ASSOCIATION



5. RELATION ENTRE CLASSES / CLASSE D'ASSOCIATION & ASSOCIATION N-AIRE



5.RELATION ENTRE CLASSES / ASSOCIATION

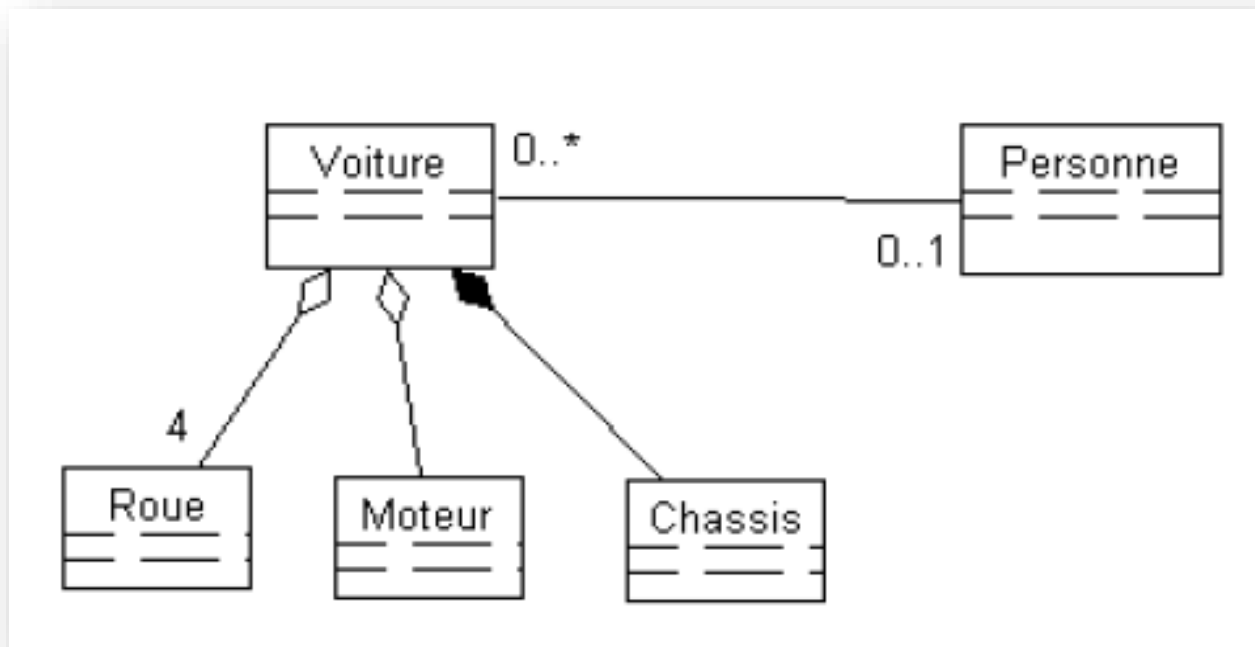
- **Composition – Agrégation**
- La composition et l'agrégation sont des cas particuliers d'association.
- **La composition** indique qu'un objet A (appelé conteneur) est constitué d'un autre objet B. Cet objet B n'appartient qu'à l'objet A et ne peut pas être partagé avec un autre objet C'est une relation très forte, si l'objet A disparaît, alors l'objet B disparaît aussi.
- *Elle se représente par un losange plein du coté de l'objet conteneur*

5.RELATION ENTRE CLASSES / ASSOCIATION

- **Composition – Agrégation**
- **L'agrégation** indique qu'un objet A possède un autre objet B, mais contrairement à la composition, l'objet B peut exister indépendamment de l'objet A. La suppression de l'objet A n'entraîne pas la suppression de l'objet B. L'objet A est plutôt à la fois possesseur et utilisateur de l'objet B
- *Elle se représente par un losange vide du côté de l'objet conteneur.*

5. RELATION ENTRE CLASSES / ASSOCIATION

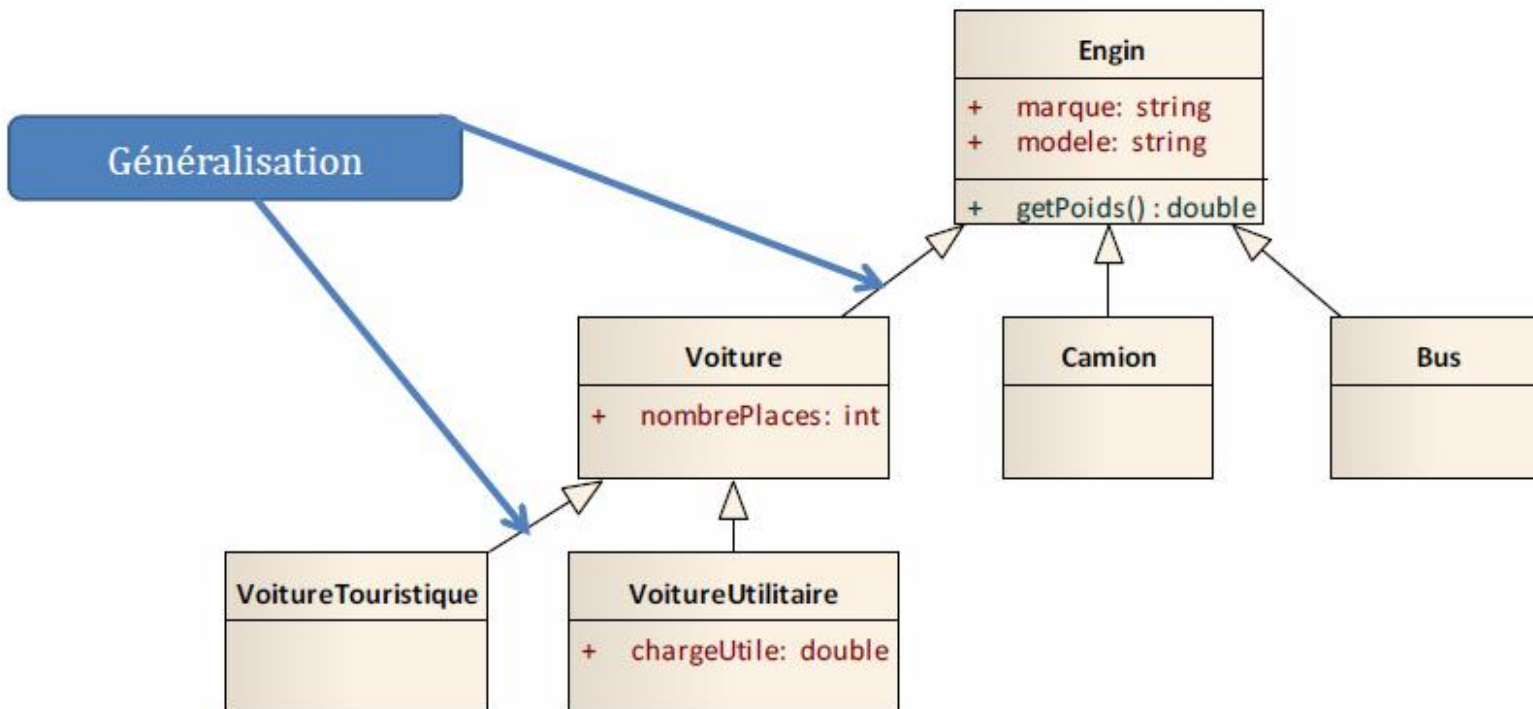
- Composition – Agrégation



5.RELATION ENTRE CLASSES / HÉRITAGE

- L'héritage indique qu'une classes B est une spécialisation d'une classe A. La classe B (appelé classe fille, classe dérivée ou sous classe) hérite tous les **attributs et des méthodes** de la classe A (appelée classe mère, classe de base ou super classe).
- *Il se représente par un triangle vide*

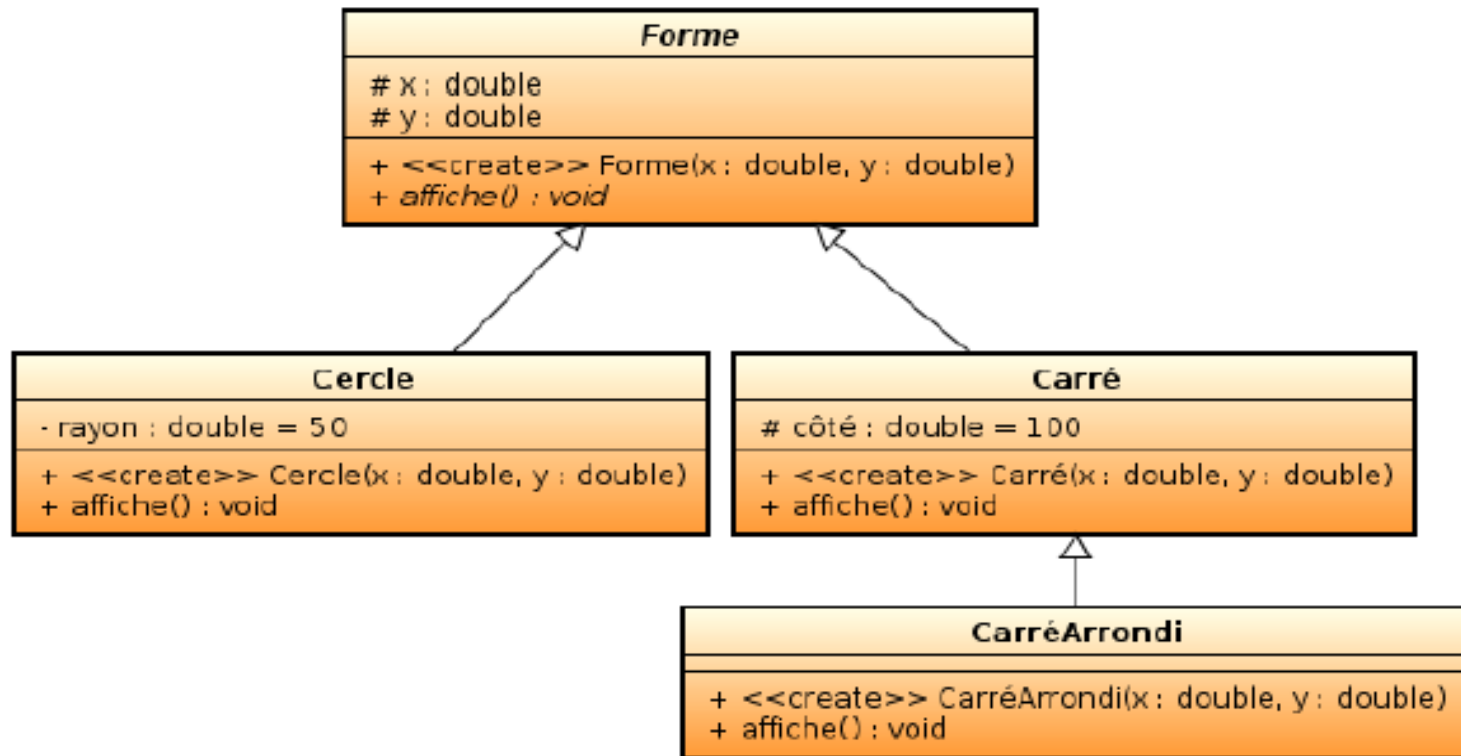
5. RELATION ENTRE CLASSES / HÉRITAGE



5.RELATION ENTRE CLASSES / CLASSES CONCRÈTES ET ABSTRAITES

- Une classe concrète possède des instances. Elle constitue un modèle complet d'objet (tous les attributs et méthodes sont complètement décrits).
- À l'opposé, une classe abstraite ne peut pas posséder d'instance directe car elle ne fournit pas une description complète. Elle a pour vocation de posséder des sous-classes concrètes et sert à factoriser des attributs et
- des méthodes à ses sous-classes.
- Une classe abstraite possède généralement des méthodes communes aux sous-classes qui sont uniquement déclarées (sans codage interne).
- En UML, une classe ou une méthode abstraite sont représentées avec une mise en italique du nom de la classe ou de la méthode.
- Tout classe possédant au moins une méthode abstraite est une classe abstraite.

5. RELATION ENTRE CLASSES / CLASSES CONCRÈTES ET ABSTRAITES



REMARQUES

- De préférence, utiliser la convention UpperCamelCase. Le nom de classe est en minuscules et la première lettre en majuscules. Si le nom de la classe est composite, le nom de chaque mot composant est en minuscule et la première lettre en majuscule. Par exemple : *Agent*, *Compte*, *LigneFacture*, *MandatPostalValide*.
- Eviter les abbréviations : par exemple utiliser *FactureValideDetaillée* au lieu de *FactureVD*.
- Ne pas utiliser des noms verbaux car les classes représentent des « choses ».

PHASE DE CONCEPTION


1. INTRODUCTION

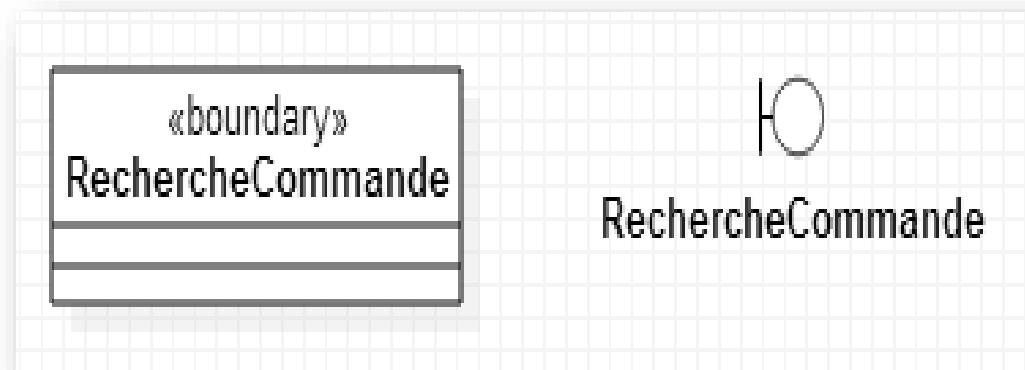
- Le modèle d'analyse définit les fonctionnalités *du système* à développer
- La conception s'intéresse à *comment* ces fonctionnalités seront implémentées
- La conception se base sur le *modèle de besoins* et le *modèle d'analyse*
- Les solutions proposées par la conception repose sur le *domaine métier* et le *domaine technique*

2. STÉRÉOTYPES


- Pour rendre les modèles plus précis et plus lisibles, Ivar Jacobson, un des fondateurs d'UML, a proposé de catégoriser les classes d'analyse/conception.
- Trois catégories de classes sont proposées :
 - Les classes «Boundary» (ou dialogue) qui représentent les interactions entre un utilisateur ou un système externe avec le système modélisé.
 - Les classes «Control» (ou contrôle) qui contiennent la cinématique du système construit,
 - Les classes « Entity » (ou entité) qui représentent les concepts métier manipulés.

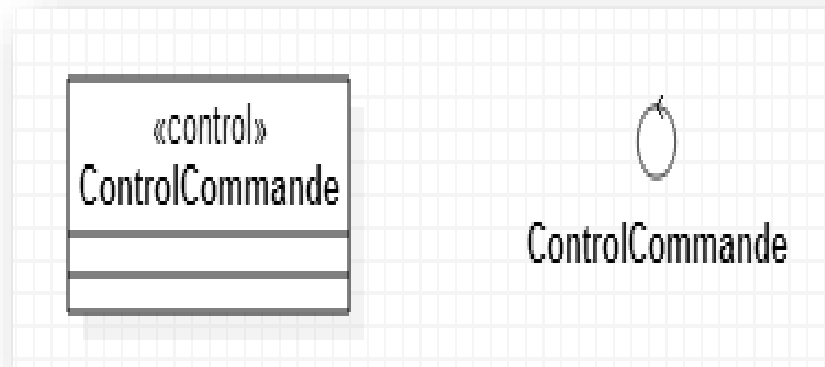
2. STÉRÉOTYPES

- Les classes « **Boundary** » sont identifiées lors de la spécification des interfaces utilisateurs (maquettes écran, ...).
- Représentation graphique : symbole  ou classe avec stéréotype
- Exemple*

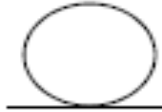


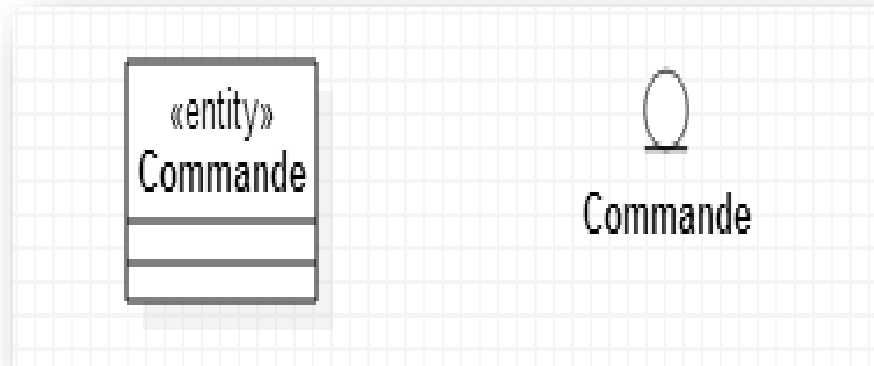
2. STÉRÉOTYPES

- Les classes «**Control**» sont chargées de la coordination entre les classes Boundary et les classes Entity.
- Représentation graphique :  ou classe avec stéréotype
- Exemple**



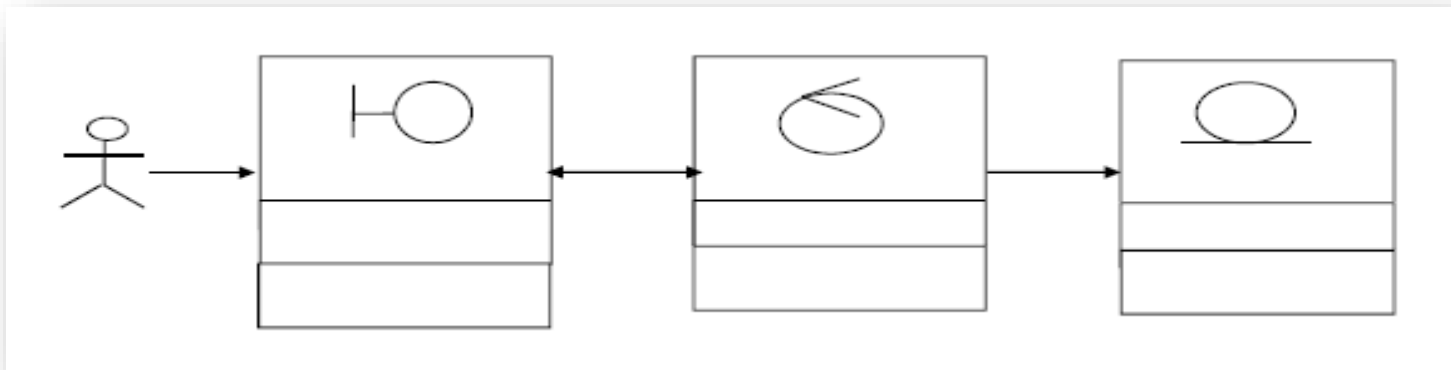
2. STÉRÉOTYPES

- Les classes « Entity » représentent les concepts métier ou classes du domaine.
- Représentation graphique :  ou classe avec stéréotype
- Exemple



2. STÉRÉOTYPES

- Règles d'interactions entre les classes :
- Les classes « Boundary » **ne peuvent être reliées** qu'aux classes « Control »,
- Les classes « Control » ont accès aux classes «Boundary », aux classes « Entity » et aux autres contrôles,
- Les classes « Entity » ont accès aux autres classes «Entity» et **ne sont reliées** qu'aux classes «Control »

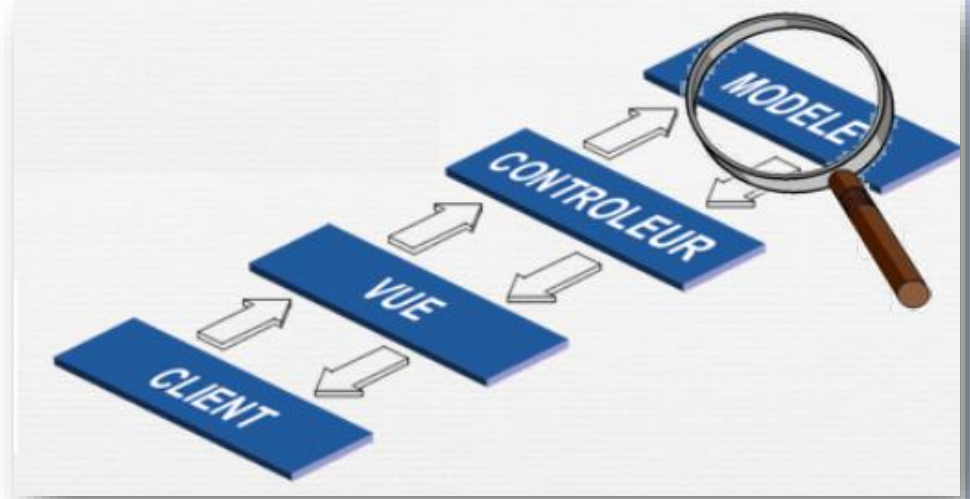


2. STÉRÉOTYPES

- L'utilisation des stéréotypes de Jacobson doit aider à améliorer la cohérence des spécifications et faciliter le passage à l'implémentation.
- Les classes « Boundary » doivent être définies en cohérence avec le diagramme de navigation.
- Les classes « Entity » doivent être utilisées pour définir le modèle de stockage des données en base
- L'utilisation des stéréotypes de Jacobson facilite la mise en œuvre du pattern MVC :
 - Classes « Entity » → Modèle.
 - Classes « Boundary » → Vue.
 - Classes « Control » → Contrôleur.

3. MVC

Modèle



- décrit et contient les données manipulées par l'application, ainsi que des traitements propres à ces données
- les résultats renvoyés par le modèle sont dénués de toute présentation
- le modèle contient la logique métier de l'application

3.MVC

VUE



- Interface avec laquelle l'utilisateur interagit
- Reçoit toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, soumission de formulaire ...)<
- Envoie les événements au contrôleur
- Présentation des résultats renvoyés par la couche modèle, après le traitement du contrôleur
- La vue n'effectue aucun traitement

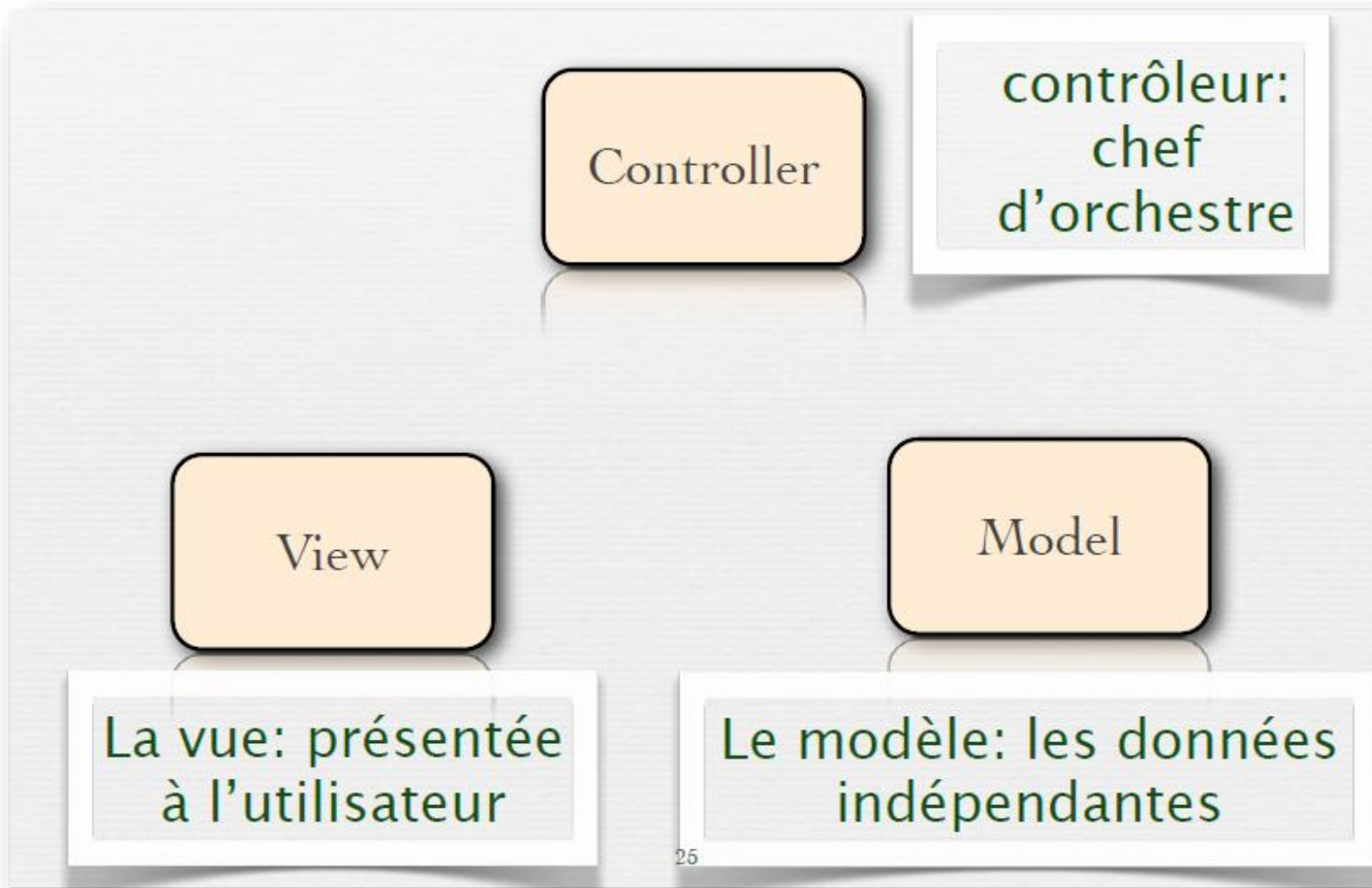
3.MVC

Contrôleur

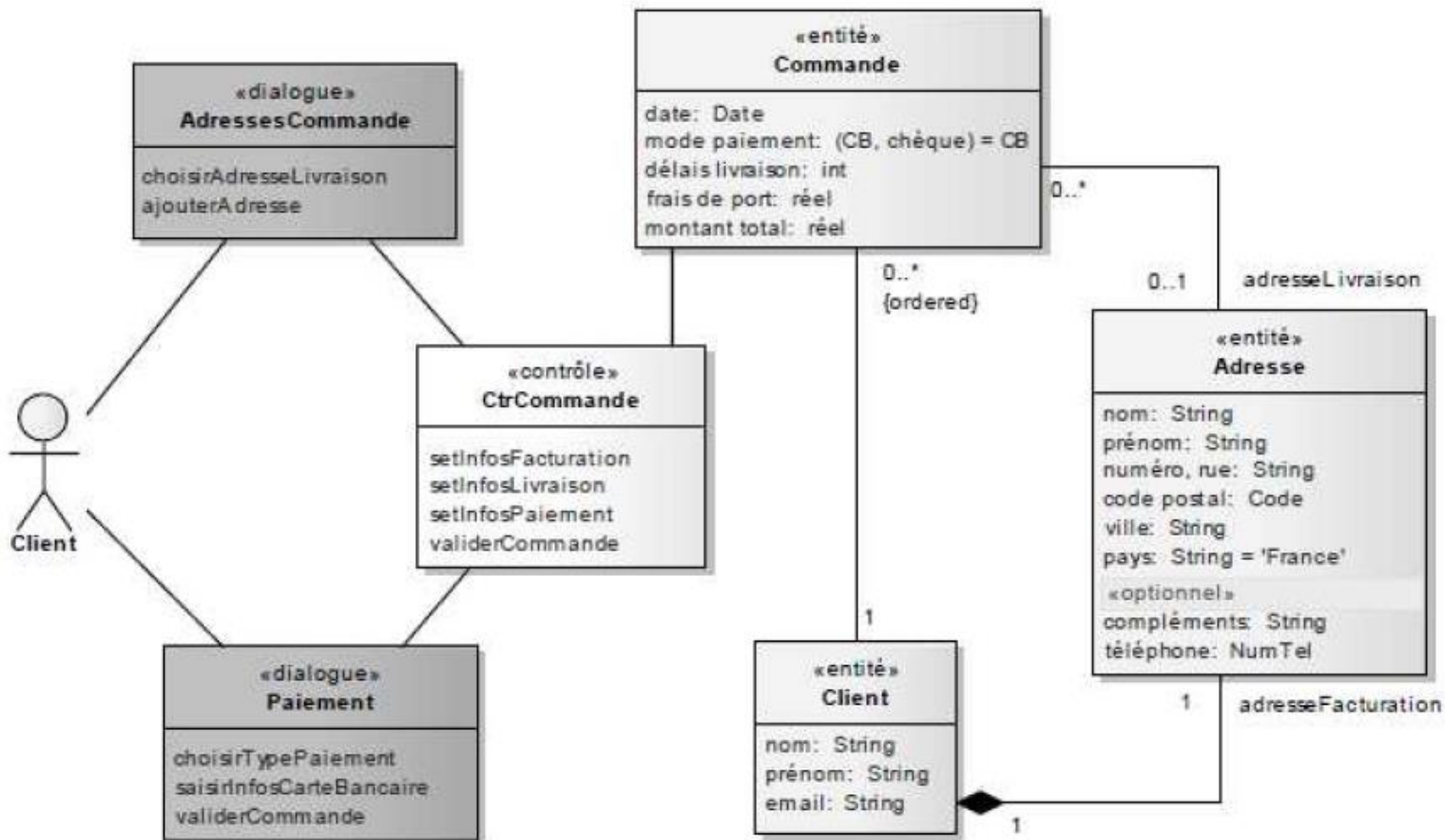


- Gestion des événements de synchronisation entre modèle et vue
- Détermine l'action à réaliser
- Si une action nécessite un changement des données
 - ❑ demande la modification des données au modèle
- Ne fait qu'appeler des méthodes
 - ❑ n'effectue aucun traitement directement
 - ❑ ne modifie aucune donnée directement

3. MODÈLE- VUE –CONTRÔLEUR (MVC)



3. EXEMPLE



3.EXEMPLE

- Les «entités» vont seulement posséder des attributs. Ces attributs représentent en général des informations persistantes de l'application.
- Les «contrôles» vont seulement posséder « des opérations ». Ces opérations montrent la logique de l'application, les règles métier, les comportements du système informatique.
- Les «dialogues » vont posséder des attributs et « des opérations ». Les attributs vont représenter des champs de saisie ou des résultats. Les opérations représenteront des actions de l'utilisateur sur l'IHM.

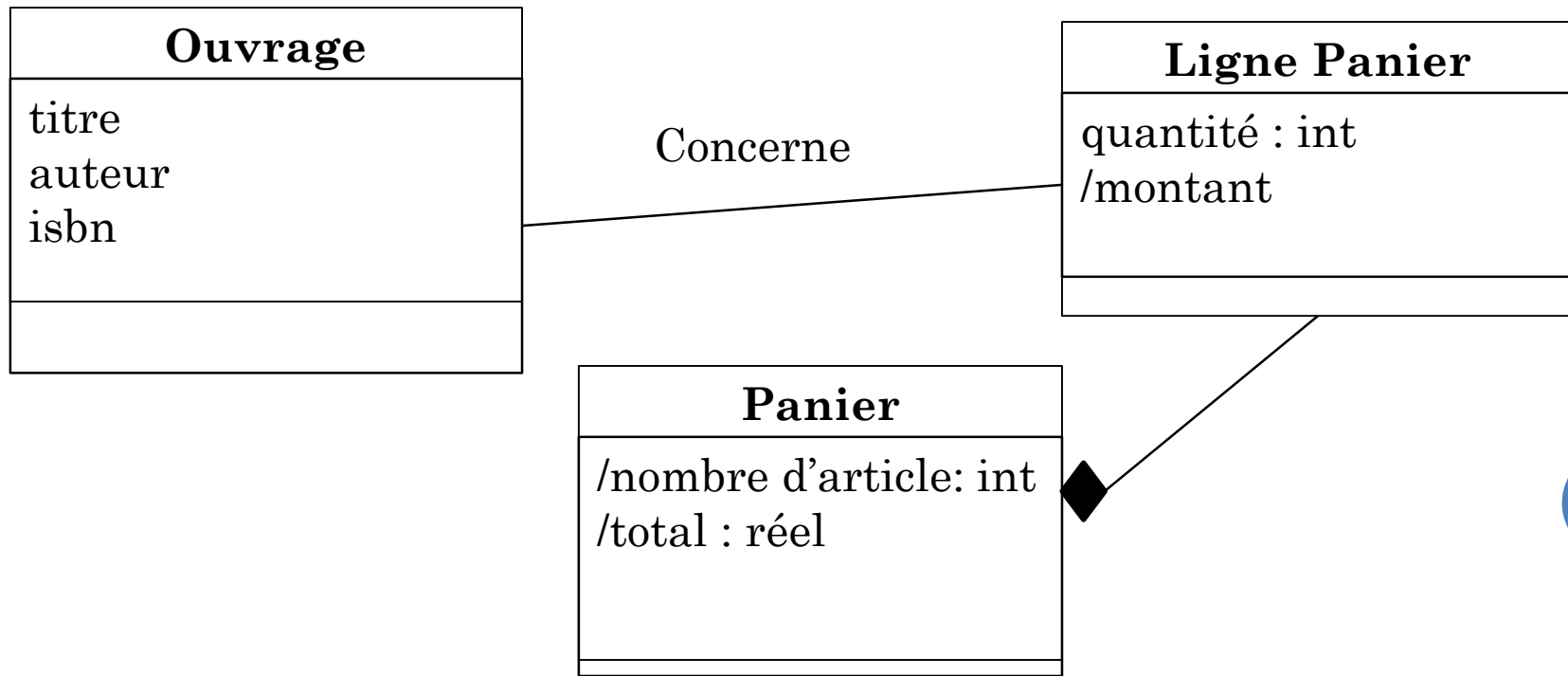
3.EXEMPLE

Exemple

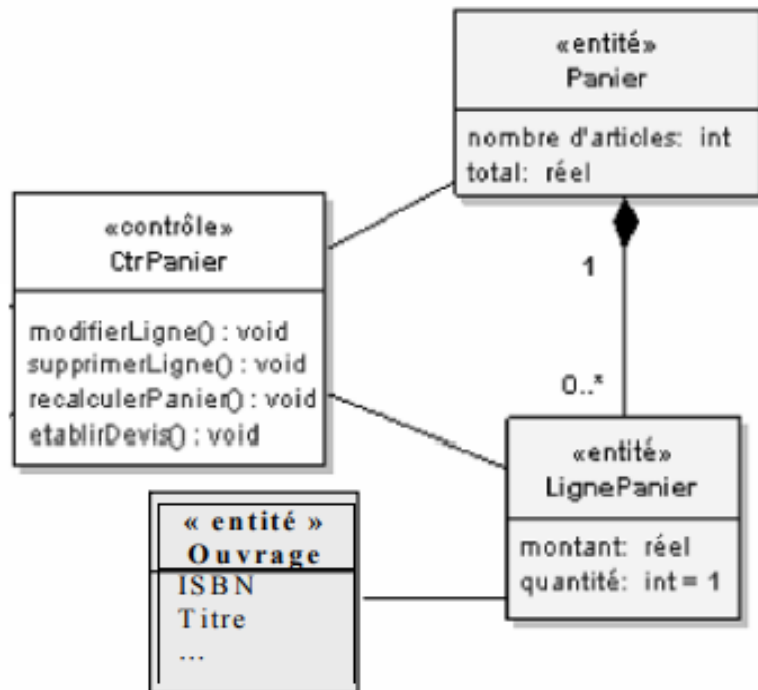
- Modélisation d'une librairie en ligne :
- L'objectif est de permettre aux internautes de rechercher des ouvrages par thème, auteur, mot-clé, etc., de constituer un panier virtuel, puis de pouvoir les commander et les payer directement sur le Web.
- Dans cet exemple, nous nous restreindrons à la fonctionnalité de gestion du panier virtuel

3.EXEMPLE

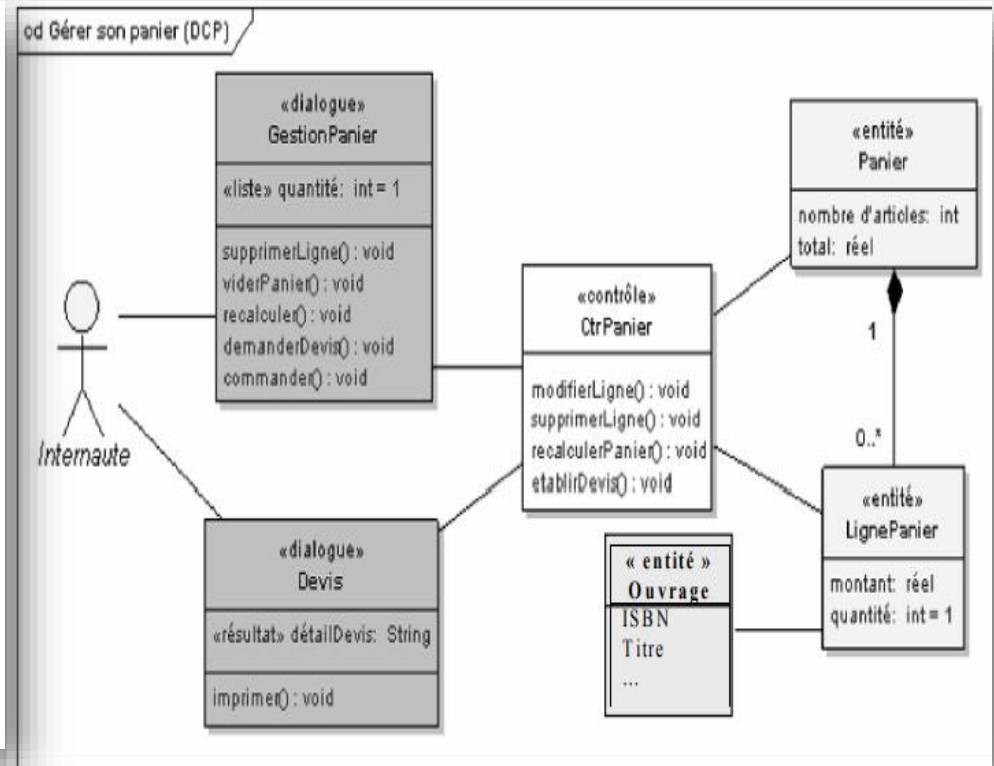
- Les concepts métier « Panier » et « ouvrage »
- 1- Modéliser ces concepts sous forme de diagrammes de classes contenant uniquement des attributs et des associations.



3. EXEMPLE

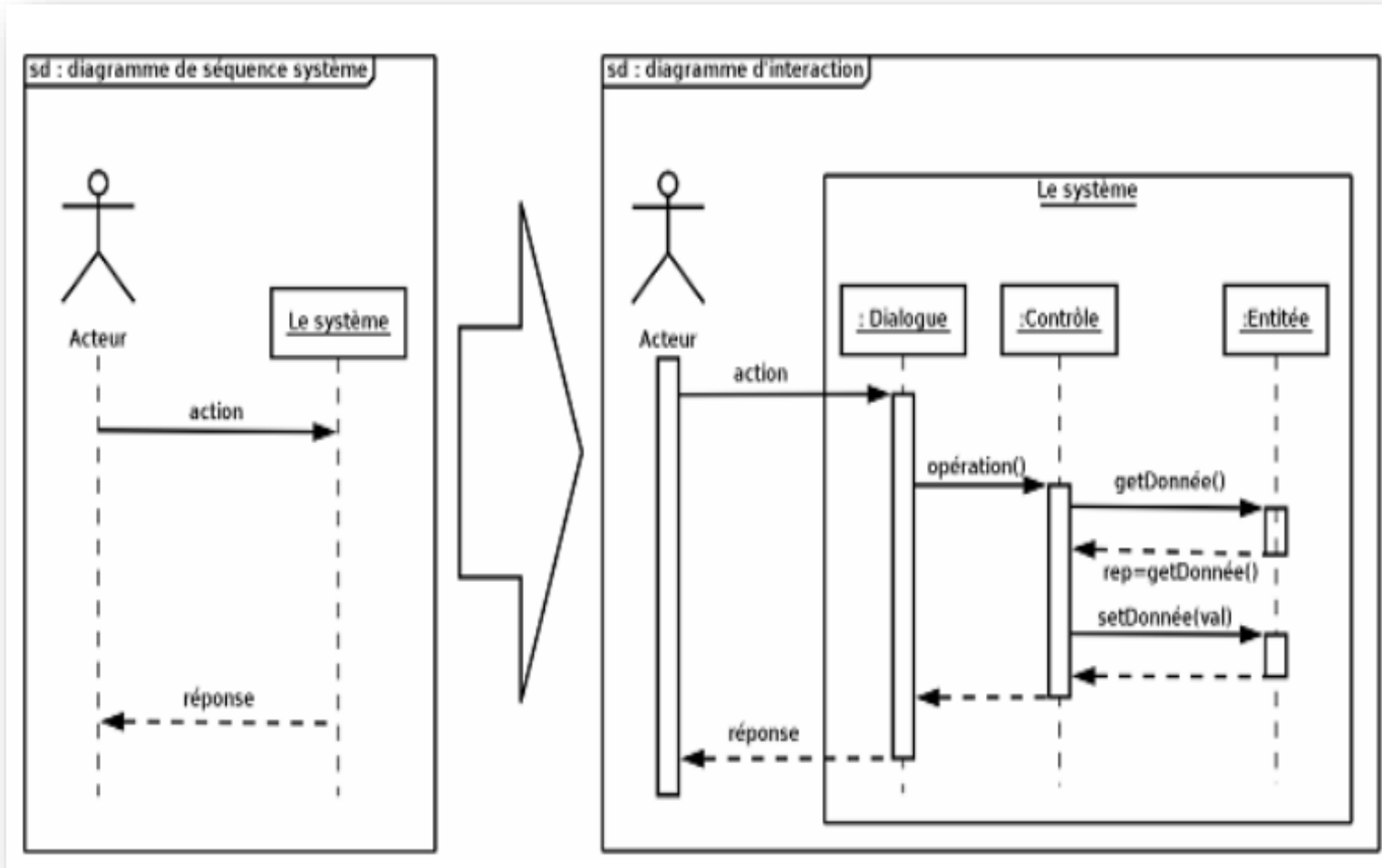


DC Contrôleur



DCP

4. DIAGRAMME DE SÉQUENCE DÉTAILLÉ

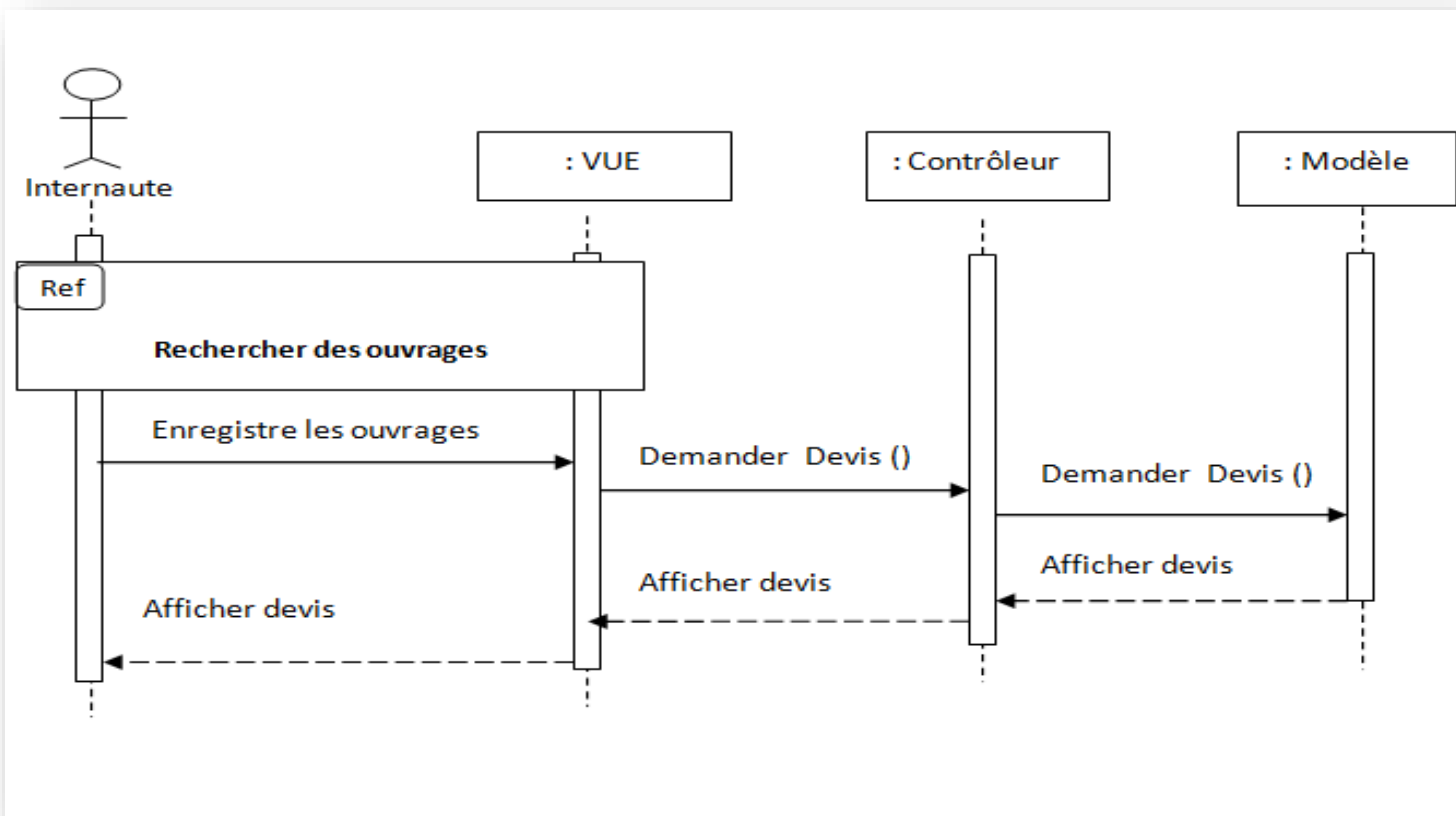


4. DIAGRAMME DE SÉQUENCE DÉTAILLÉ

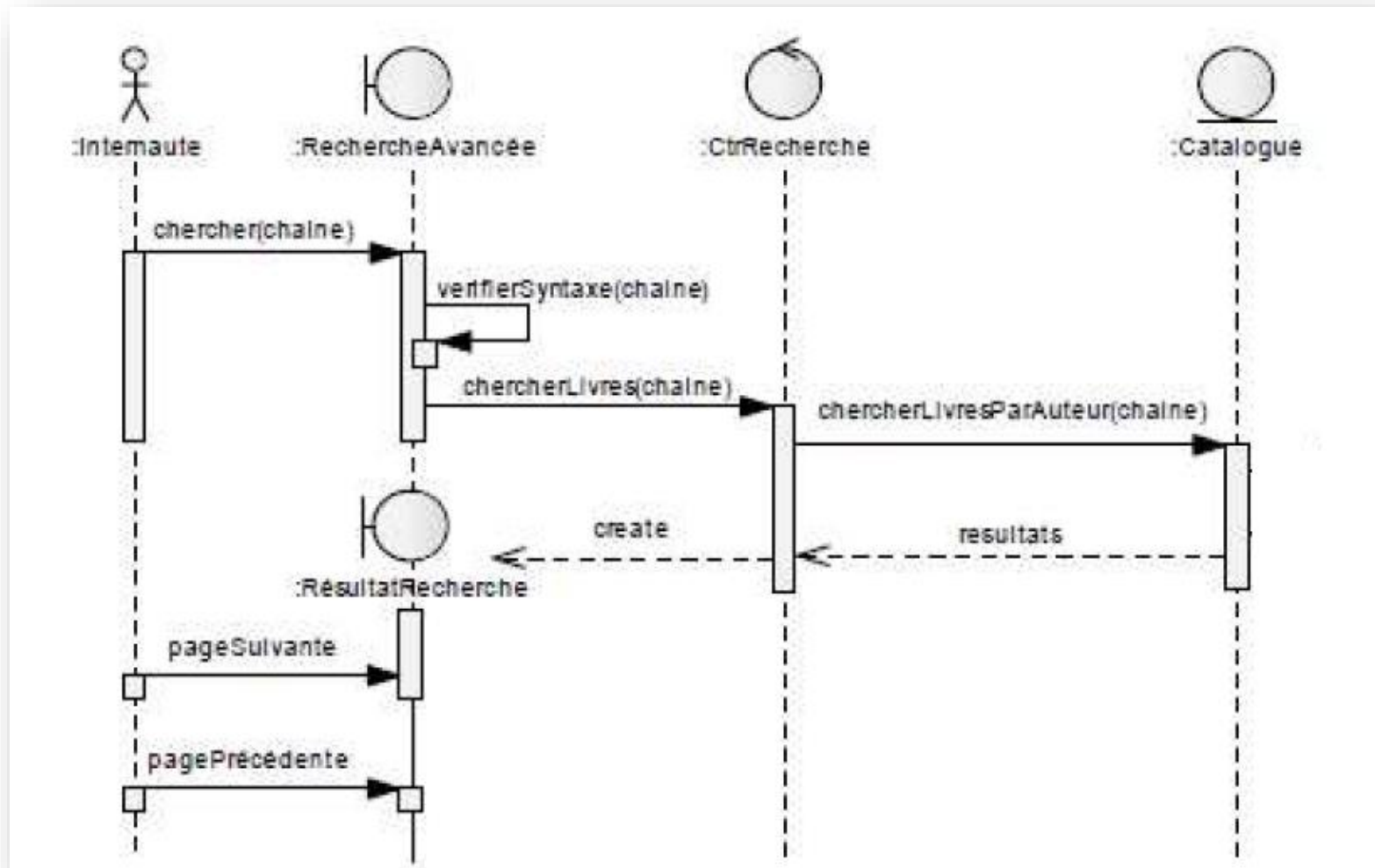
- **Exemple**
- Scénario nominal :
- 1. L'Internaute enregistre les ouvrages qui l'intéressent dans un panier virtuel
- 2. Le Système lui affiche l'état de son panier. Chaque ouvrage qui a été préalablement sélectionné est présenté sur une ligne, avec son titre, son auteur son prix unitaire et le prix total de la ligne est calculé. Le total général est calculé par le Système et affiché en bas du panier avec le nombre d'articles.
- 3. L'Internaute demande un devis pour commander
- 4. Le système valide le paiement.

4. DIAGRAMME DE SÉQUENCE DÉTAILLE

○ Exemple



4. DIAGRAMME DE SÉQUENCE DÉTAILLÉ



4. DIAGRAMME DE SÉQUENCE DÉTAILLÉ

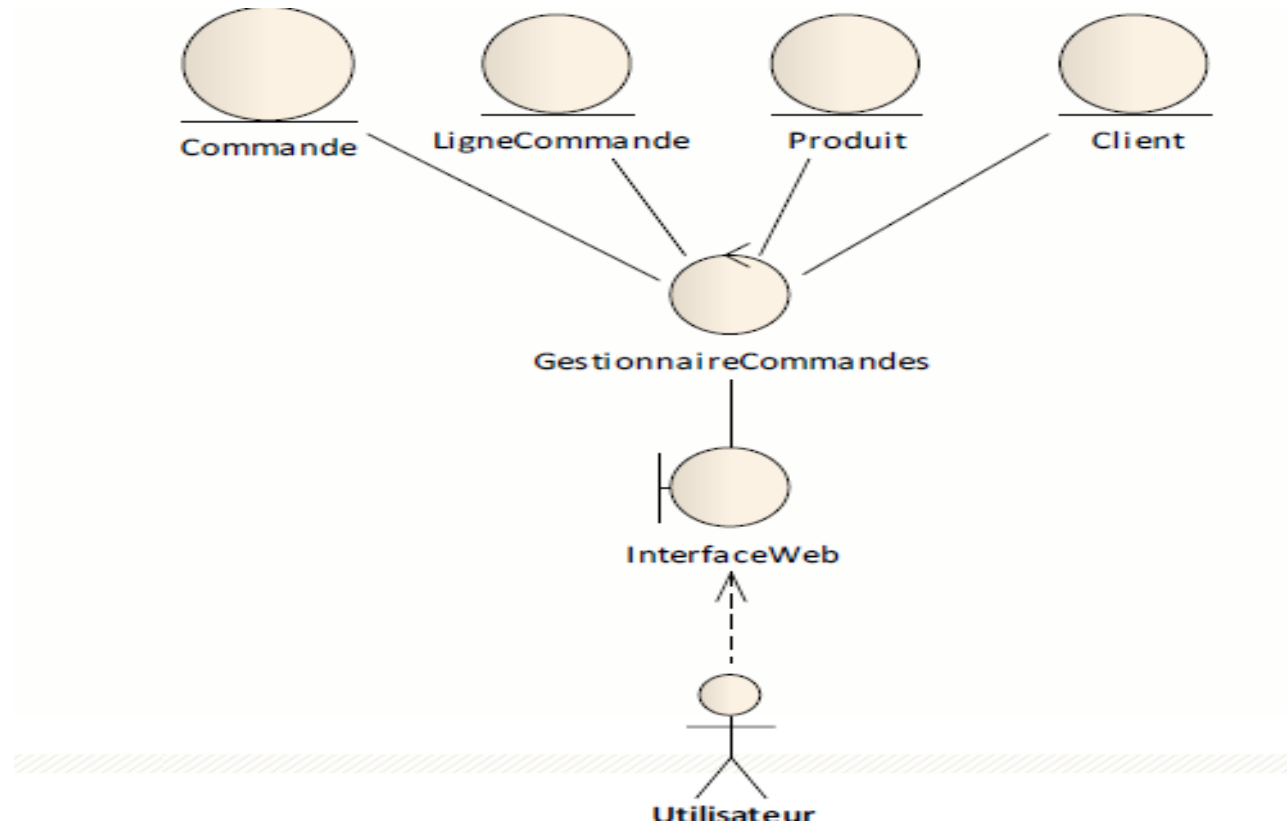
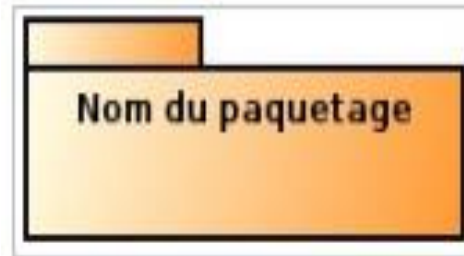


Diagramme de package

1. DIAGRAMME DE PACKAGE

- Lorsque nous sommes en présence d'un système **de grande taille**, il peut être intéressant de le décomposer en plusieurs parties (appelées paquetage).
- Un paquetage est un regroupement de différents éléments d'un système (regroupement de classes, diagrammes, fonctions, interfaces...). Cela permet de clarifier le modèle en l'organisant.



1. DIAGRAMME DE PACKAGE

- Un paquetage est un regroupement d'éléments de modèle et de diagrammes. Il peut contenir tout type d'élément de modèle: des classes, des cas d'utilisations, des interfaces... et même des paquetages imbriqués.
- Les éléments contenus dans un paquetage doivent être cohérentes entre elles et ils sont généralement de même nature et de même niveau sémantique.
- les paquetages permettent d'**encapsuler** des éléments de modélisation en proposant des interfaces, cela empêche l'accès aux données par un autre moyen que les services proposés par l'interface.

2. DÉPENDANCES ENTRE PAQUETAGES

- **Visibilité** : Chaque éléments d'un paquetage est soit :
 - **Privé**: c'est-à-dire encapsulé dans le paquetage et invisible à l'extérieur de celui-ci. Un élément privé est désigné par un signe – devant lui.
 - **Public**: c'est-à-dire visible et accessible de l'extérieur du paquetage. Un élément public est désigné par un signe + devant lui.
- *Par défaut, les éléments d'un paquetage sont publics.*

2. DÉPENDANCES ENTRE PAQUETAGES

Dépendance de type « **import** » : Elle correspond à l'importation par un paquetage B de tous les éléments **publics** d'un paquetage A. Ces éléments

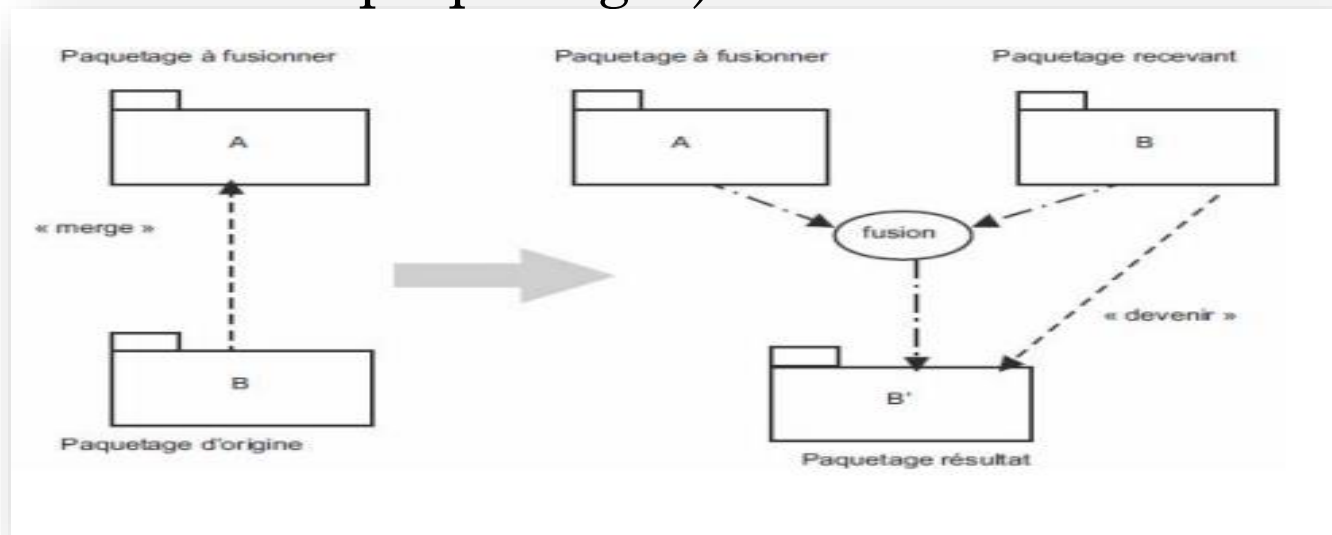
- ❑ Auront la visibilité « public » dans le paquetage B (et seraient donc aussi transmis à un paquetage C qui ferait une importation du paquetage B).
- ❑ Seront accessibles au paquetage B sans avoir à utiliser explicitement le nom du paquetage A.

2. DÉPENDANCES ENTRE PAQUETAGES

- **Dépendance de type « access » :**
- Elle correspond à l'accès par un paquetage B de tous les éléments **publics** d'un paquetage A. Ces éléments auront la visibilité privé dans le paquetage B, ils ne peuvent donc pas être transmis à un paquetage C qui ferait une importation ou un accès au paquetage B (pas de transitivité).

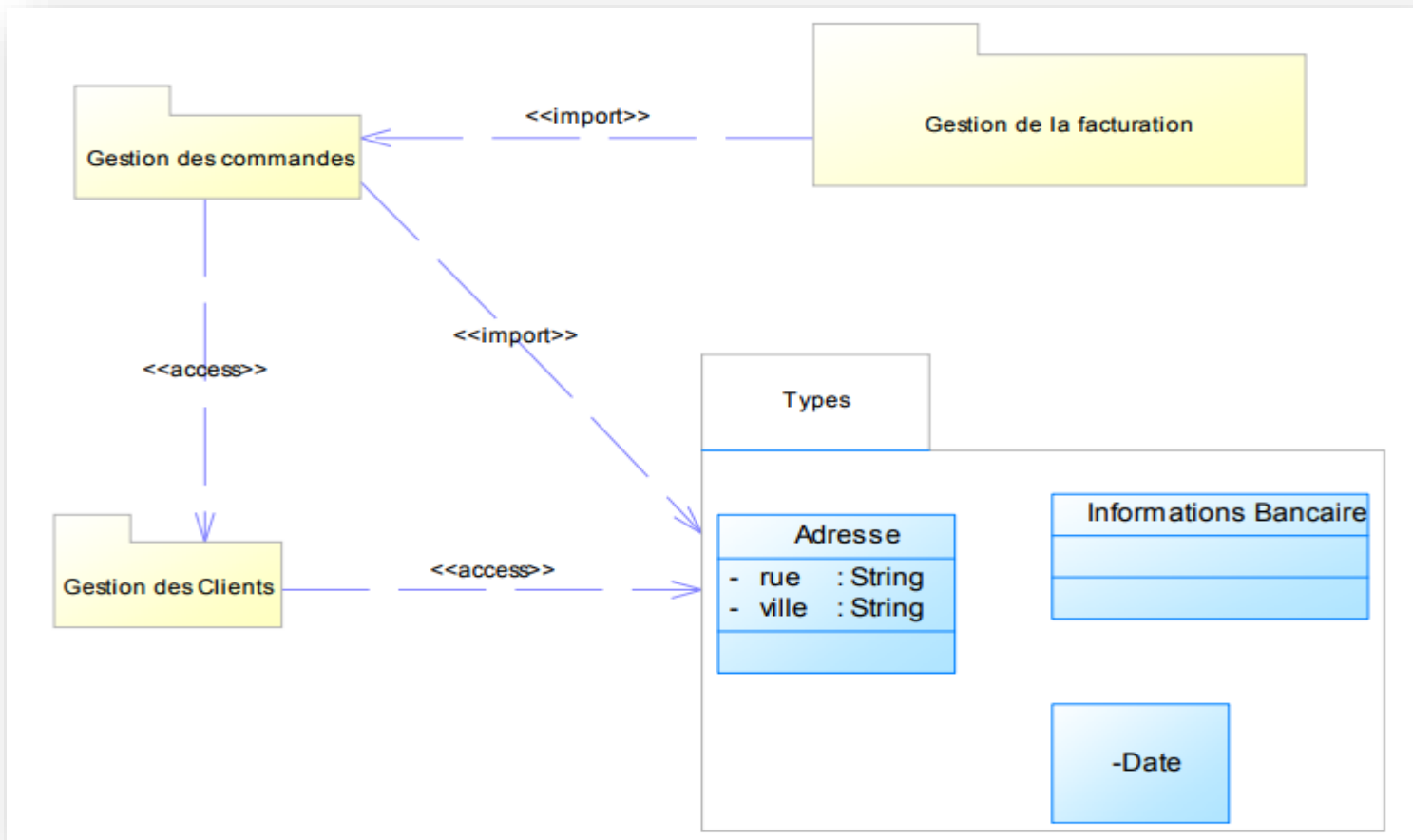
2. DÉPENDANCES ENTRE PAQUETAGES

- **Dépendances de type « merge »**
- Elle correspond à la fusion de 2 paquets en un seul.
- ($b \rightarrow A$): Le paquetage A est fusionné dans le paquetage B (le paquetage A n'est pas modifié alors que le paquetage B est écrasé pour accueillir la fusion des 2 paquetages).



2. DÉPENDANCES ENTRE PAQUETAGES

○ Exemple



2. DÉPENDANCES ENTRE PAQUETAGES

- « Date » est un élément privé dans le package « Types » -
- « Adresse » et « Informations bancaires » sont visibles dans « Gestion des commandes »
- « Date » n'est pas visible dans « Gestion des commandes ».
- « import » est transitive, donc « Adresse » et « Informations bancaires » sont aussi visibles dans « Gestion de la facturation ».
- Les éléments du package « Gestion des clients » sont visibles dans « Gestion des commandes », mais pas dans le package « Gestion de la facturation ».

3. STRUCTURATION EN PACKAGES

- La structuration d'un modèle statique est une activité délicate. Elle doit s'appuyer sur deux principes fondamentaux : *cohérence et indépendance*.
- Le premier principe consiste à regrouper les classes proches d'un point de vue sémantique. Pour cela, il faut que les critères de cohérence suivants soient réunis :
 - ❑ Finalité : les classes doivent rendre des services de même nature aux utilisateurs ;
 - ❑ Evolution : on isole ainsi les classes réellement stables de celles qui vont vraisemblablement évoluer au cours du projet.
 - ❑ cycle de vie des objets : ce critère permet de distinguer les classes dont les objets ont des durées de vie très différentes.
- Le second principe consiste à minimiser les dépendances entre les packages.