Université Sidi Mohamed Ben Abdellah.
École Supérieure de Technologie de Fès.
Département : Informatique & STID.

UML–Diagramme de classes

Réalisé par : - M.BENSLIMANE.

ANNEE UNIVERSITAIRE : 2022/2023

Plan

I. Introduction

II. Les classes

III. Interfaces

IV. Relations entre classes

V. Généralisation et Héritage

VI. Exemples

I. Introduction

Objectif : Décrire la structure statique du système en modélisant les classes et leurs relations indépendamment d'un langage de programmation particulier.

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet. Il est le seul obligatoire lors d'une telle modélisation.

Les principaux éléments de cette vue statique sont les classes et leurs relations : association, généralisation et plusieurs types de dépendances, telles que la réalisation et l'utilisation.

I. Introduction

N.B.

Le diagramme de cas d'utilisation montre le système du point de vue des acteurs.

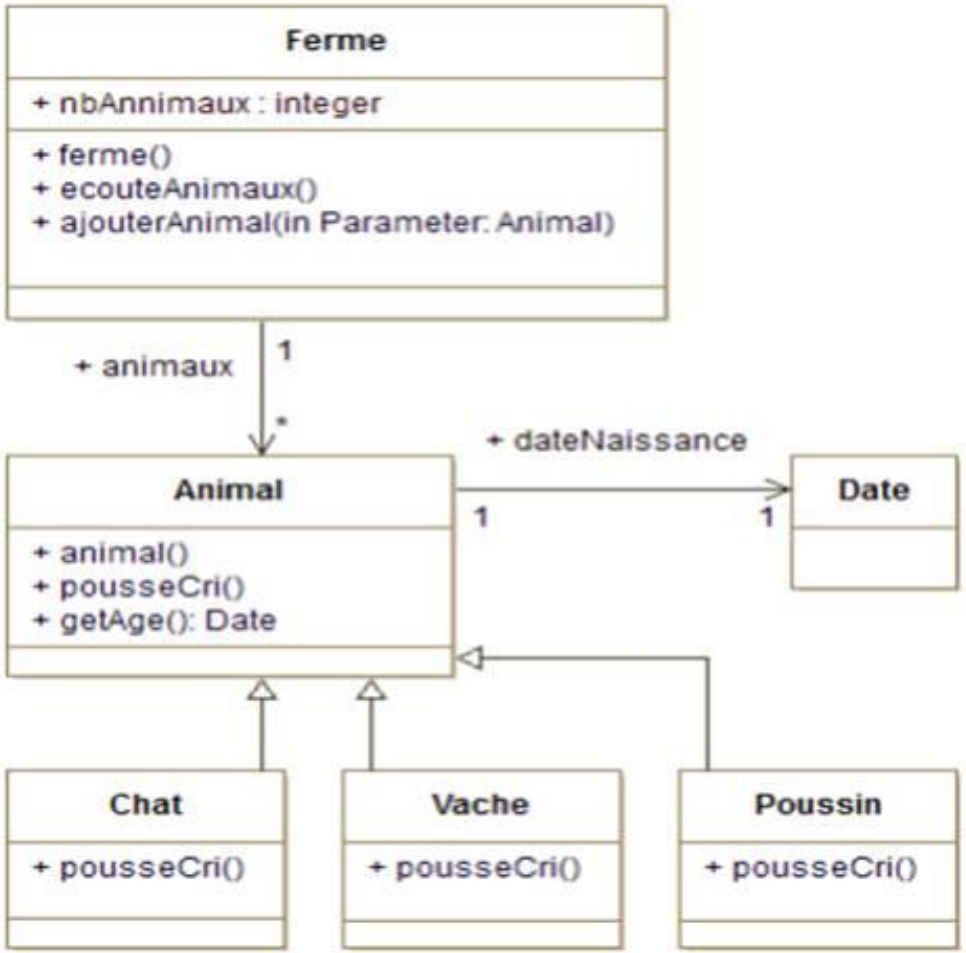
Les cas d'utilisation ne réalisent pas une partition des classes du diagramme de classes.

Le diagramme de classes en montre la structure interne : représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation.

Il s'agit d'une *vue statique car on ne tient pas compte du facteur temporel dans le* comportement du système.

I. Introduction

Exemple.



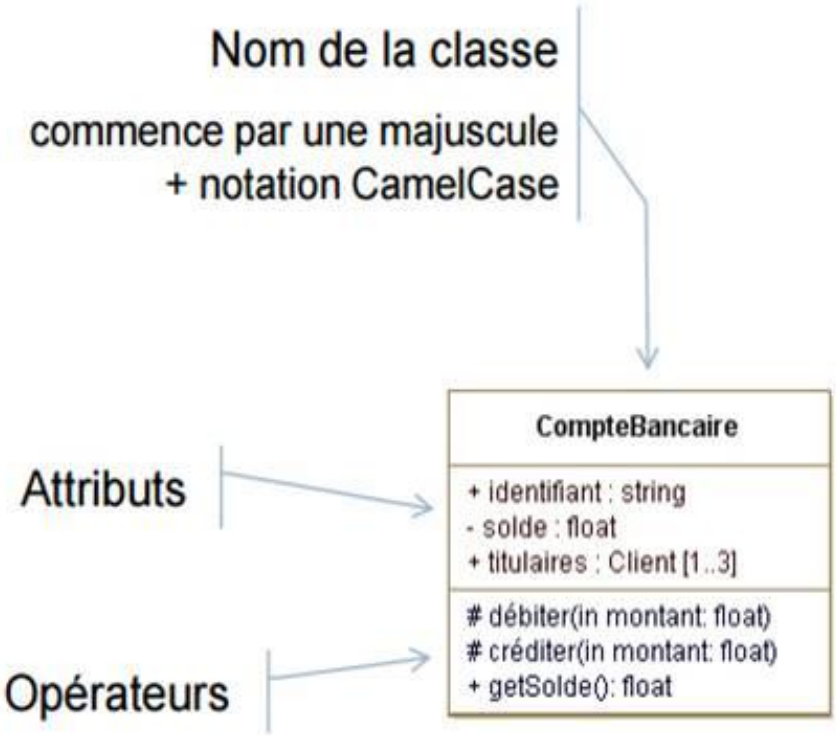
I. Introduction

Exemple :

Classe : entité avec un ensemble d'attributs et d'opérations

```
public class CompteBancaire{
        public String identifiant;
        private Float solde;
        public ArrayList titulaires;

protected void debiter(Float montant)
{}
protected void credit(Float montant)
{}
public Float getbalance () {}
}
```



II. Les classes

II.1. Notion de classe et objet (rappel)

- **Classe :** c'est la description formelle d'un ensemble d'objets ayant en commun :
  - une sémantique,
  - des propriétés (attributs)
  - des relations (associations)
  - et un comportement (méthodes).

- **Objet :** une ***entité concrète*** *avec une identité bien définie qui encapsule un* ***état*** *et* un ***comportement***. *L'état est représenté par des valeurs d'attribut et des* associations. Le comportement est représenté par des méthodes.

- **Instance :** Une instance est une concrétisation d'un concept abstrait.
  - Un objet est une instance d'une classe.
  - Un lien est une instance d'association.

II. Les classes

- **Méthode / Opération :**

• Une méthode est le comportement d'une classe. Ce terme désigne aussi l'implémentation (i.e. la définition) d'une méthode.
• Une opération est la spécification (i.e. la déclaration) d'une méthode.

- **Encapsulation :**

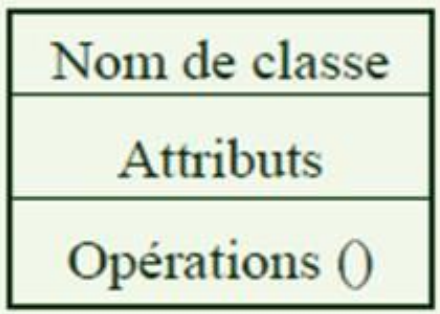
Mécanisme consistant à :

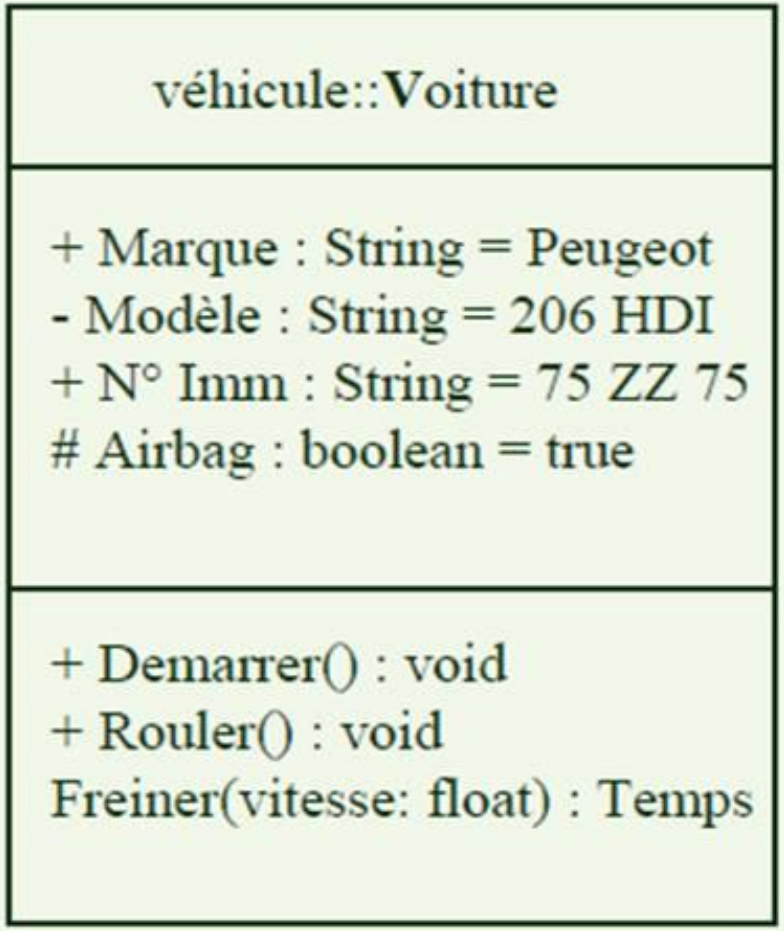
- rassembler les données et les méthodes au sein d'une structure.
- cacher l'implémentation.

L'encapsulation permet de définir ***des niveaux de visibilité*** *(private, protected* et pubic).

II. Les classes

II.2. Exemple de représentation d'une classe





II. Les classes

II.3. Méthode et Classe abstraites

Méthode abstraite : On connaît sa déclaration mais pas sa définition.

Classe abstraite : c'est une classe non instanciable.

Une classe est dite abstraite lorsqu'elle possède au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.

Une classe abstraite peut contenir des méthodes concrètes.

Le terme ***abstract*** est utilisé pour indiquer à la fois les méthodes et les classes abstraites

II. Les classes

II.3. Méthode et Classe abstraites

Méthode abstraite : On connaît sa déclaration mais pas sa définition.

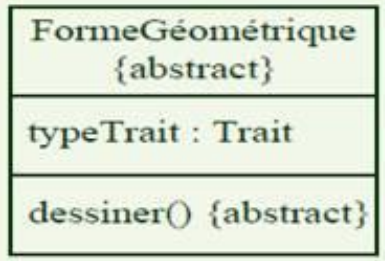
Classe abstraite : c'est une classe non instanciable.

Une classe est dite abstraite lorsqu'elle possède au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.

Une classe abstraite peut contenir des méthodes concrètes.

Le terme ***abstract*** est utilisé pour indiquer à la fois les méthodes et les classes abstraites

Exemple :



II. Les classes

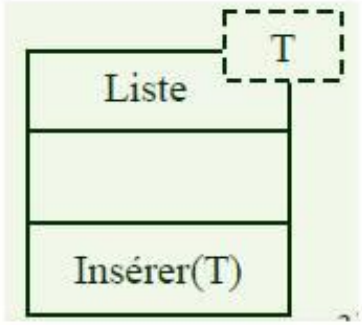
II.4. Classe paramétrée (généricité)

- Une classe paramétrée est un modèle de classe paramétré par des classes et/ou des constantes (paramètre de généricité).

- **Exemple :**

Afin d'éviter de créer une classe "Liste" pour chaque type d'éléments : entier, float, string, Point, etc., on préfère paramétrer la classe "Liste" par un ***type formel***, sans avoir à décider du type d'objet sur lequel elle sera amenée à porter.

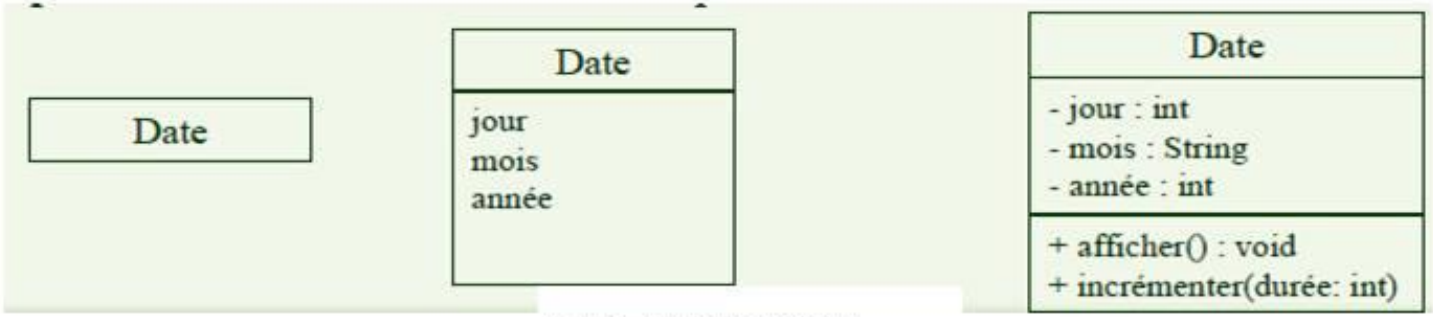
Le ***type effectif*** sera précisé à l'instanciation de la classe.



II. Les classes

II.5. Modélisation d'une classe

- Une classe est un classeur et est représentée par un rectangle divisé en trois à cinq compartiments :
  1. ***Nom de la classe***
  2. ***Attributs***
  3. ***Méthodes***
  4. Responsabilités (tâches devant être assurées mais pas encore bien définies)
  5. Exceptions (Situations exceptionnelles à gérer)
- **Exemple : Différents niveaux de description d'une classe**



II. Les classes

- Nom de la classe :

- Doit évoquer le concept décrit par la classe.
- Commence par une majuscule.
- Mot-clef ***abstract*** pour indiquer qu'une classe est abstraite.

Syntaxe :

[<Nom paquetage 1>::...::<Nom paquetage N>::]
<Nom classe> [{[abstract],[<auteur>],[<date>],[<état>],...}]

II. Les classes

- Description des attributs :

• **Syntaxe :**

<visibilité> [/] <nom attribut> : <nom classe> ['['<multiplicité> ']'] [= <valeur initiale>]

➢ **Visibilité :** quatre niveaux de visibilité prédéfinis :

- public (+) : élément visible partout
- protégé (#) : élément visible dans la classe et ses descendants (les sous-classes)
- privé (-) : élément visible uniquement dans la classe
- package (~ ou rien) : élément visible uniquement dans le paquetage.

➢ **Attributs dérivés :**

- Peuvent être calculés à partir d'autres attributs et de formules de calcul.
- Les attributs dérivés sont symbolisés par l'ajout d'un « / » devant leur nom.
- Exemple : l'âge d'une personne.

II. Les classes

➢ Attribut de classe :

- Un attribut de classe garde une valeur unique et partagée par toutes les instances.
- Les instances ont accès à cet attribut mais n'en possèdent pas une copie.
- L'accès à cet attribut ne nécessite pas l'existence d'une instance.
- Graphiquement, **un attribut de classe est souligné.**

➢ Multiplicité :

Certains attributs peuvent être désignés par plusieurs éléments d'un type donné.

La multiplicité est le nombre d'éléments de ce type, elle est exprimée par un intervalle.

II. Les classes

Exemples de spécification d'attributs :

//seulement le nom

• Origine	
• + origine	*visibilité et nom*
• origine: Point = (0, 0)	*nom, type et initialisation*
• nom [0..1] : String	*nom, multiplicité et type*
• - val [1..50] : Integer	*visibilité, nom, multiplicité et type*
• - */ age: float*	*visibilité, attribut dérivé, nom et type*
• + *nb : integer*	*visibilité, nom, type et attribut de classe*

II. Les classes

- Description des opérations :

Syntaxe :

<visibilité> <nom méthode> ([<listes_de_paramètres>]) : [<type retour>]
[{<propriétés>}]

• Les paramètres sont séparés par des virgules et sont de la forme :

Nom_paramètre : type [= valeur_par_défaut]

• {<propriétés>} : contraintes ou informations complémentaires (méthode abstraite (mot-clef abstract), méthode non polymorphe (leaf) préconditions, post-conditions, . . .).

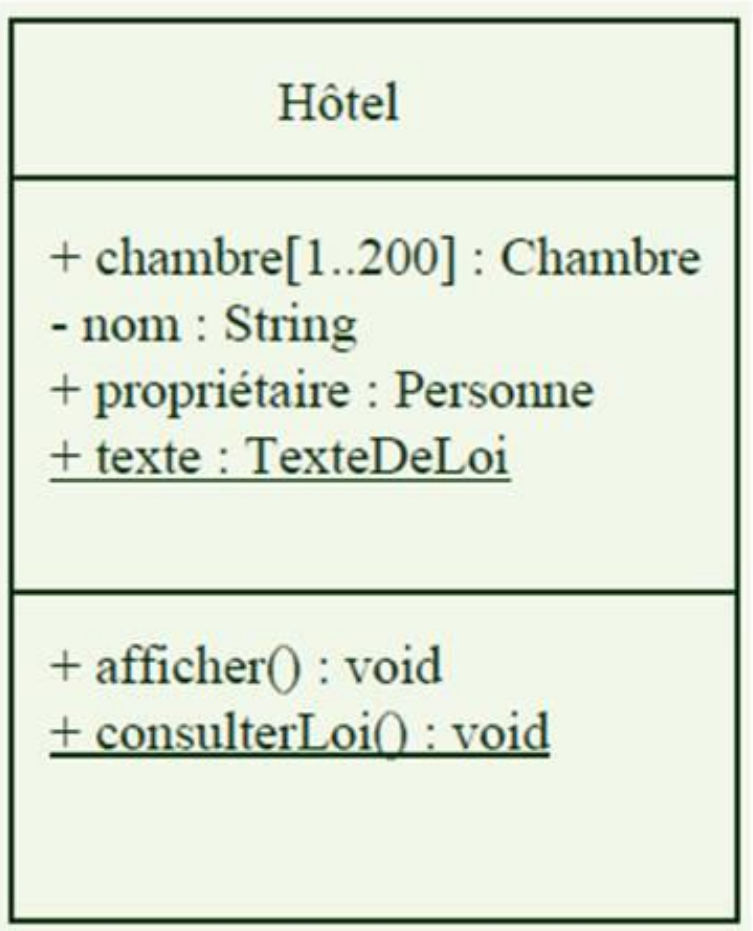
II. Les classes

Méthode de classe :

- Ne peut manipuler que des attributs de classe et ses propres paramètres.
- N'a pas accès aux attributs de la classe.
- L'accès à une méthode de classe ne nécessite pas l'existence d'une instance de cette classe.
- Graphiquement, **une méthode de classe est soulignée.**

II. Les classes

Exemple :

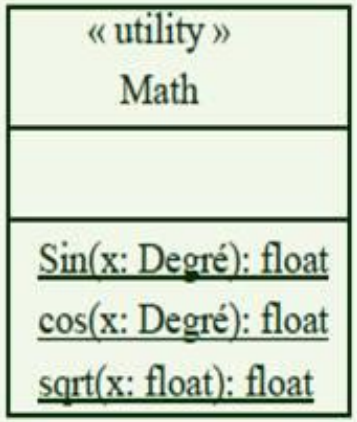


II. Les classes

Remarque :

Des stéréotypes permettent de préciser la signification d'une classe. Il doivent précéder le nom de la classe :

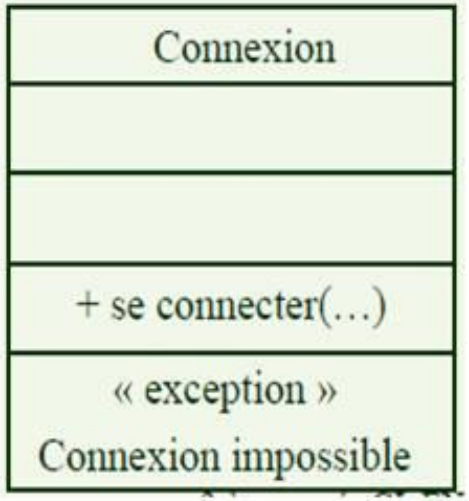
- « **utility** » **:** une classe qui correspond à une bibliothèque et rassemble essentiellement des méthodes de classe.
- « **exception** » **:** une classe qui décrit une exception.
- « **interface** » **:** il s'agit en fait d'une interface et non d'une classe.



II. Les classes

- **Compartiments complémentaires :**

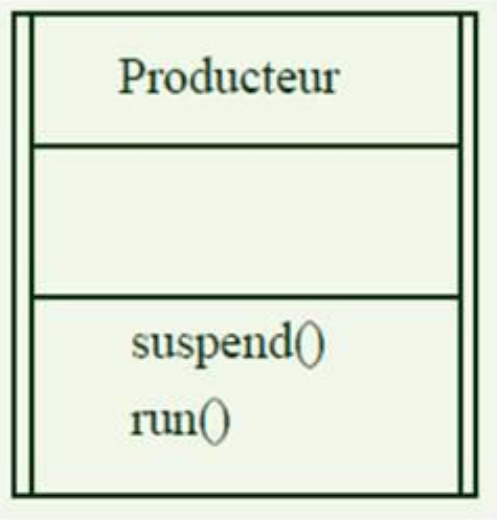
- Permettent d'ajouter des informations complémentaires aux classes telles que les responsabilités et les exceptions.
  - les responsabilités : sont des taches devant être assurés par la classe.
  - les exceptions : sont les situations exceptionnelles devant être gérées par la classe.
- A la phase de réalisation, ces deux compartiments seront transformés en un ensemble d'attributs et de méthodes.



II. Les classes

II.6. Classe active :

- Par défaut, une classe est passive : mémorise des données et offre des services aux autres.
- Classe active : initie et contrôle le flux d'activités.
- Graphiquement : comme une classe standard dont les lignes verticales du cadre sont doublées.

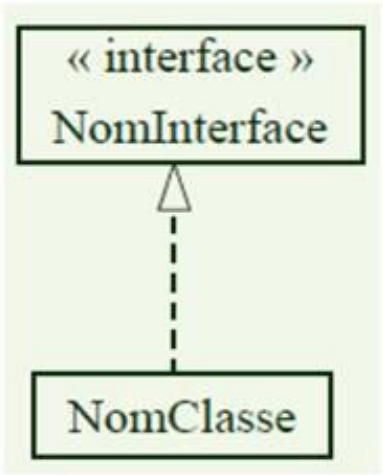


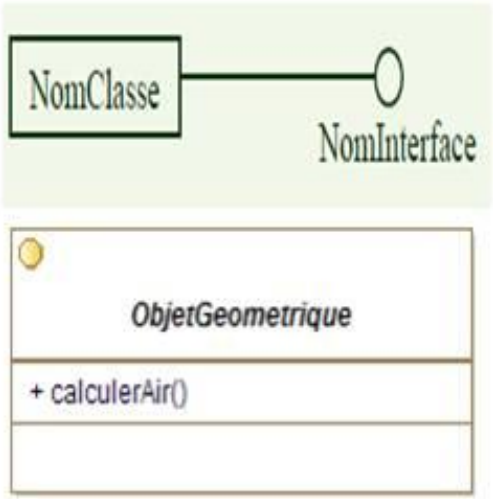
III. Interfaces

Objectif : factoriser un ensemble de propriétés et d'opérations assurant un service cohérent.

Représentation :

- Mêmes compartiments qu'une classe mais on ajoute le stéréotype « interface » avant le nom de l'interface.
- Pas de mot-clef abstract car l'interface et toutes ses méthodes sont, par définition, abstraites.



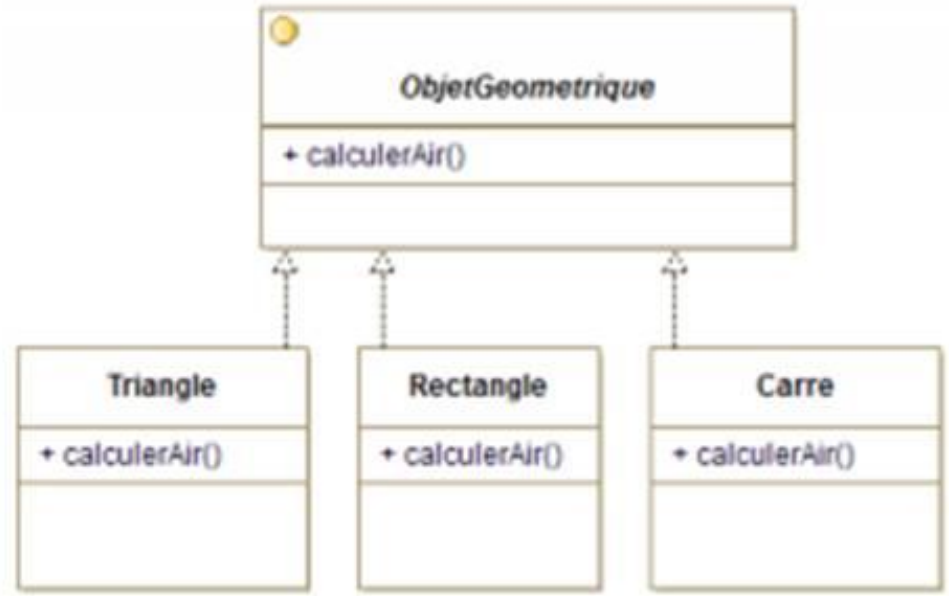



```
public interface ObjectGeometrique{
        public Point pts;
        public float air;

        public float calculerAir() {}
}
```

III. Interfaces

Implémentation : une classe peut implémenter une interface. Pour que l'implémentation soit valide, cette classe doit spécifier le contenu de chaque opération définie par l'interface.



IV. Relations entre classes

IV.1. Association

Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions entre leurs instances.

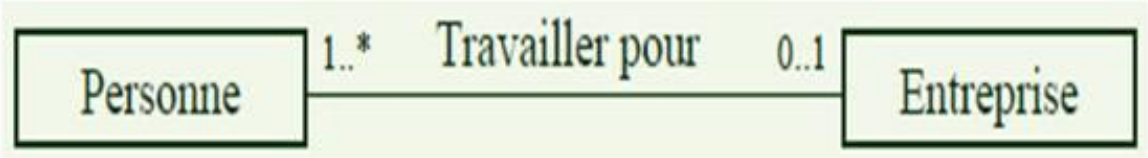
Graphiquement, Une association binaire est représentée par un trait plein entre les classes associées. Elle peut être ornée d'un nom.

La relation d'association est :

- Bidirectionnelle : elle peut être lue dans les deux sens.
- Valuée : sur les extrémités de la relation est précisé le nombre d'objets impliqués dans la relation (la multiplicité).

Quand les deux extrémités de l'association pointent vers la même classe, l'association est dite réflexive.

Exemple :

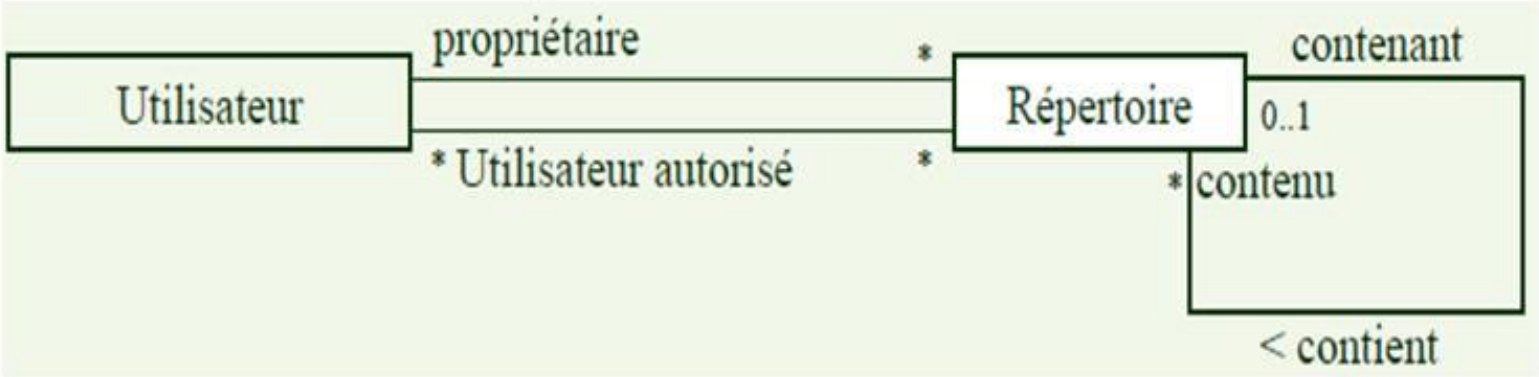


IV. Relations entre classes

- Nom d'une association :

- Le nom de la relation : Par convention, les relations se lisent de gauche à droite et de haut en bas. Si le sens de lecture n'est pas naturel, il faut ajouter une flèche.

- On peut préciser ***le rôle*** que joue un objet dans la relation.



IV. Relations entre classes

- **Multiplicité d'une association :**

Dans une association binaire, la multiplicité sur la terminaison cible contraint le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association).

La multiplicité est notée par une liste de possibilités séparées par des virgules.

Une possibilité est :

– un entier : le nombre exact d'objets (ex : 4) .

– un intervalle (1..4).

– * : un nombre quelconque y compris 0 .

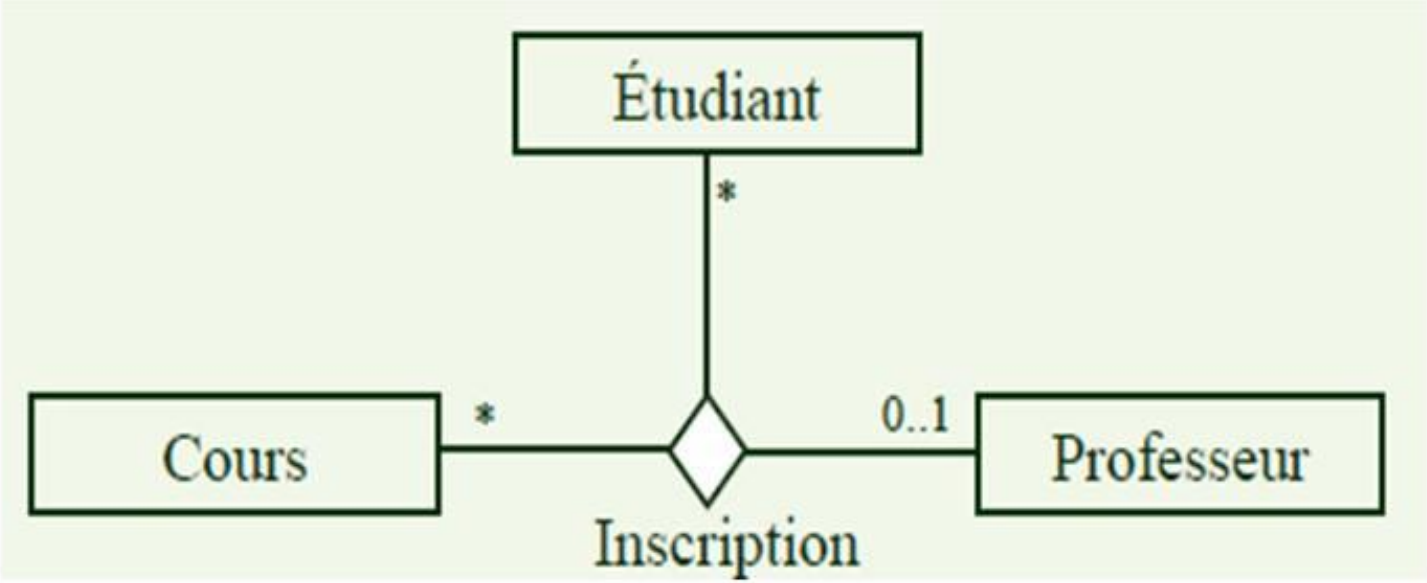
– 1..* : un nombre quelconque hors 0.

La multiplicité par défaut est (exactement) 1.

IV. Relations entre classes

- Association n-aire :

- Une association n-aire lie plus de deux classes.
- On représente une association n-aire par un grand losange avec un chemin partant vers chaque classe participante, le nom de l'association peut apparaître à proximité du losange.



IV. Relations entre classes

- **Navigabilité :**

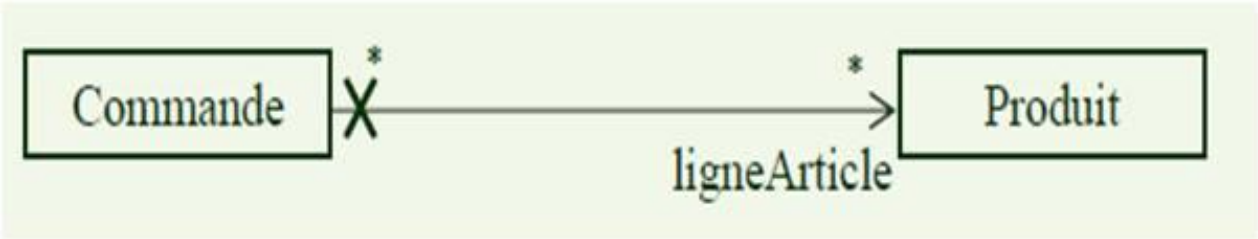
La navigabilité indique s'il est possible de traverser une association.

On représente graphiquement la navigabilité par une flèche du côté de la terminaison navigable.

On empêche la navigabilité par une croix du côté de la terminaison non navigable.

Par défaut, une association est navigable dans les deux sens.

Exemple :

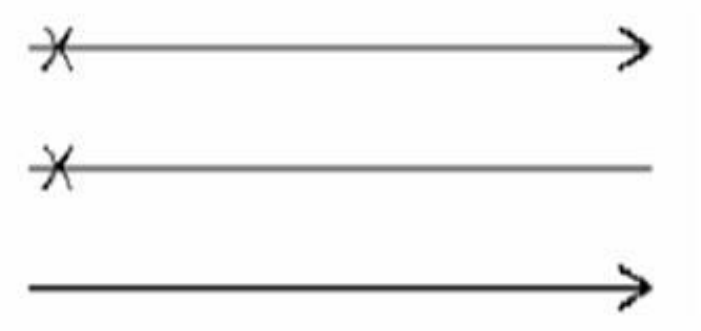


IV. Relations entre classes

La terminaison du côté de la classe Commande n'est pas navigable : les instances de la classe Produit ne stockent pas de liste d'objets du type Commande.

La terminaison du côté de la classe Produit est navigable : chaque objet commande contient une liste de produits.

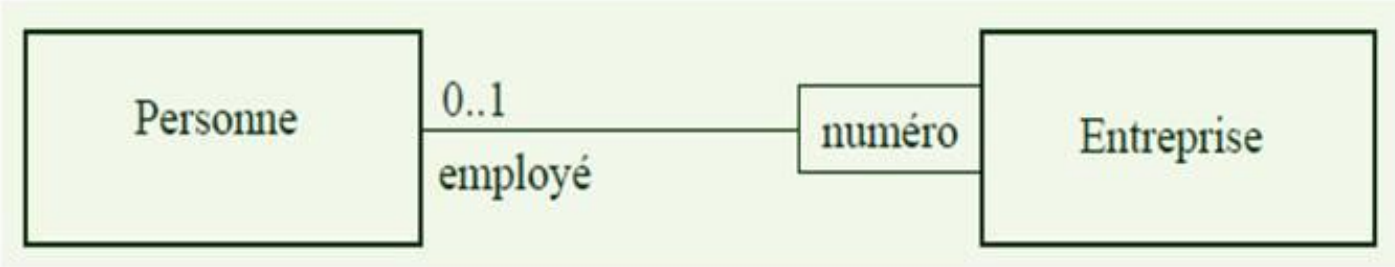
- **N.B.** Les trois notations de navigabilité suivantes sont équivalentes :



IV. Relations entre classes

IV.2. Association qualifiée

Un ***qualificatif*** *est un attribut spécial qui est mis sur une extrémité d'une relation. La* **qualification** d'une association permet en général de transformer une multiplicité indéterminée ou infinie en une multiplicité finie.



Intérêt : La qualification améliore la précision sémantique de la relation :

– elle *réduit la multiplicité effective (* est transformée en 0..1, numéro est* ici équivalent à une clé au sens base de données) ;

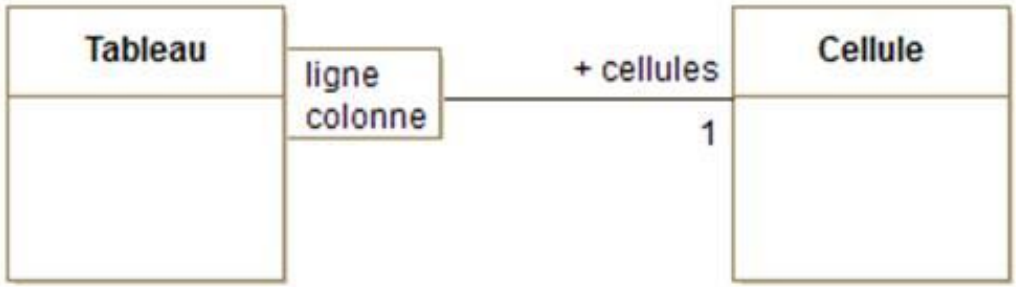
– elle *améliore la navigation dans le réseau des objets (pour désigner une* personne, il suffit d'avoir son numéro, *qui est unique dans l'entreprise).*

IV. Relations entre classes

IV.2. Association qualifiée

Possibilité de sélection d'un sous-ensemble d'objet associé à l'aide d'un ou plusieurs attributs qualificatifs (appelés clés).

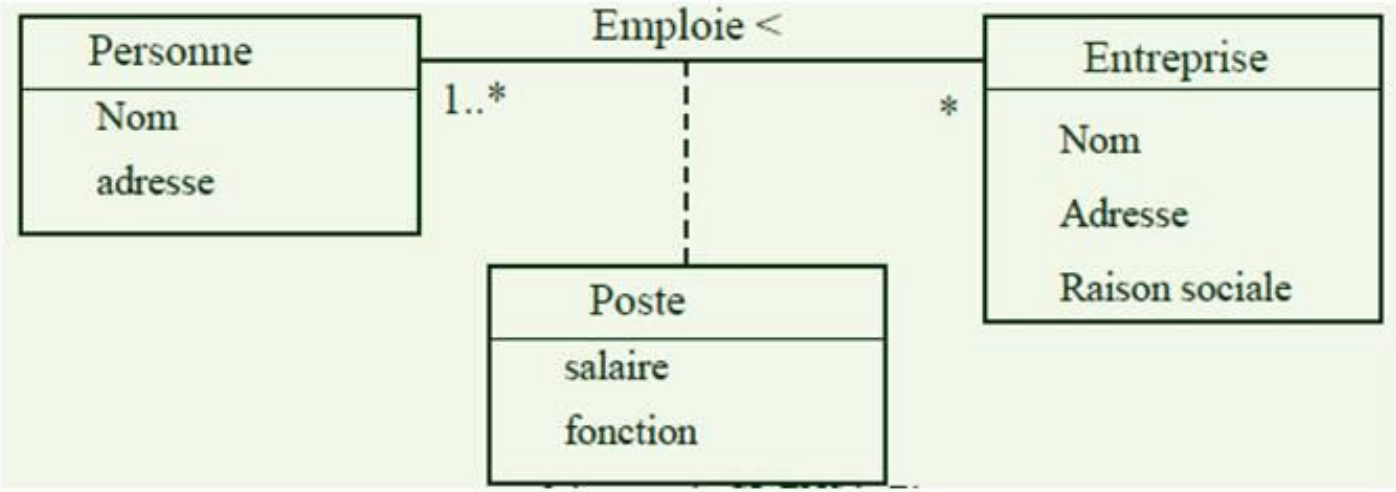
En Java, ce type d'association se traduit souvent par une table d'indexage.



IV. Relations entre classes

IV.3. Classe-association

- Une association peut être raffinée et avoir ses propres propriétés qui ne sont disponibles dans aucune classe qu'elle lie. Cette association devient alors une classe appelée ***classe-association.***
- Un attribut d'une classe d'association (attribut de relation) caractérise la relation et pas seulement une de ses classes extrémités.
- Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

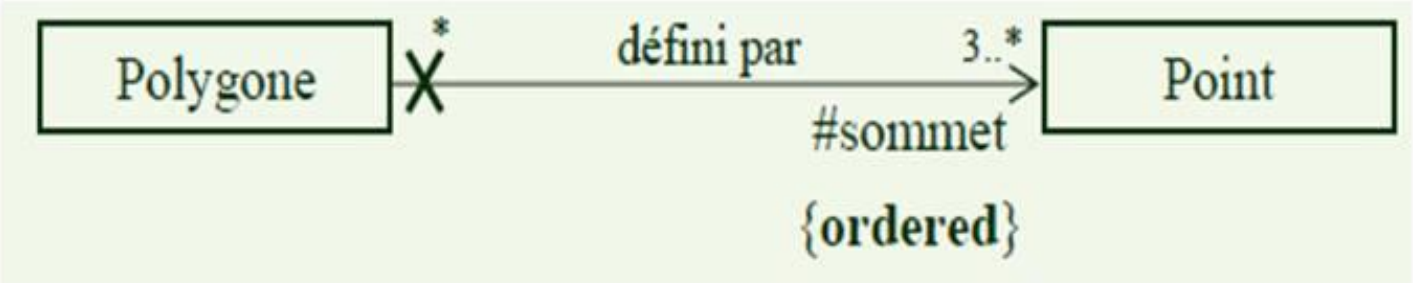


IV. Relations entre classes

IV.4. Association avec contraintes

- L'ajout de contraintes ***à une association*** ou bien ***entre associations*** permet de mieux préciser la portée et le sens de l'association.
- Les contraintes sont mises entre accolades et exprimées de préférence avec le langage OCL (Object Constraint Language).

Exemple :



IV. Relations entre classes

IV.5. Agrégation

C'est une forme particulière d'une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble.

Graphiquement, on ajoute un losange vide du côté de l'agrégat.

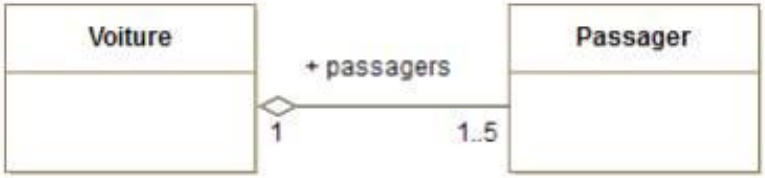
Remarques :

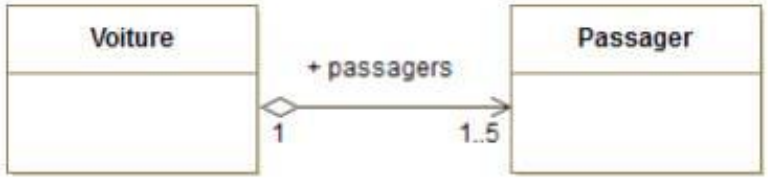
- Contrairement à une association simple, l'agrégation est transitive.
- C'est une association déséquilibrée, où une classe joue un rôle prépondérant.
- La définition d'une opération d'un objet agrégat repose sur les opérations des objets agrégés (ex : segment et points extrémités).

IV. Relations entre classes

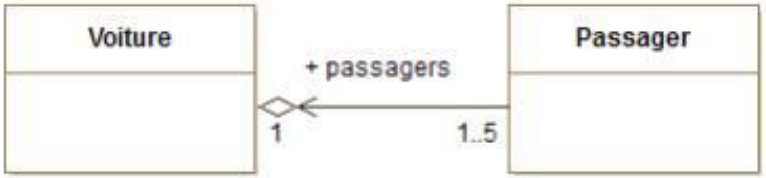
IV.5. Agrégation

Agrégation : association + les éléments existent toujours quand l'association est détruite





En connaissant une voiture, on peut accéder à la liste de ses passagers.



En connaissant un passager, on peut accéder à la voiture dans laquelle il est.

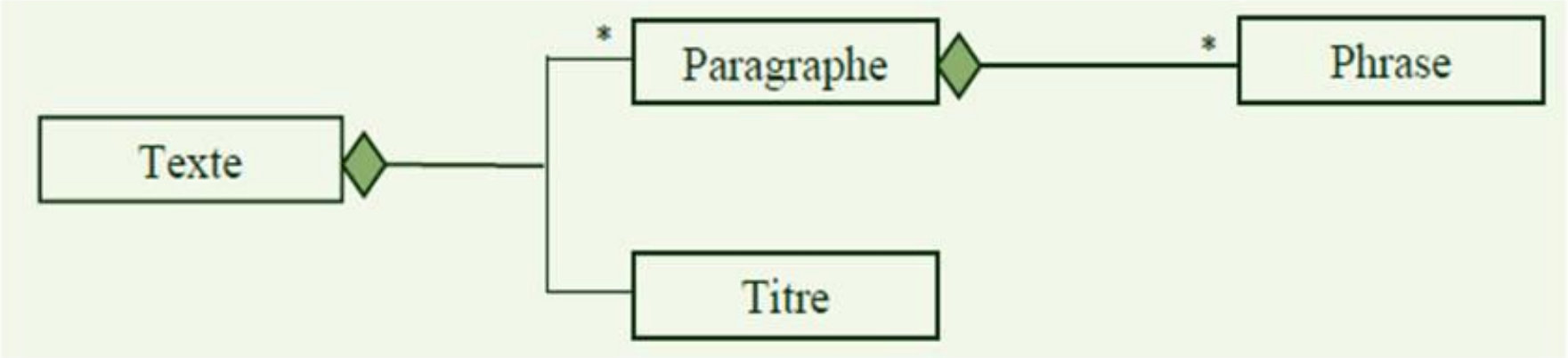
IV. Relations entre classes

IV.6. Composition

- La relation de composition est un cas particulier de la relation d'agrégation avec une sémantique plus forte :
- Les durées de vie du composé et de ses composants sont liées.
- La composition, également appelée ***agrégation composite***, décrit une contenance structurelle entre instances.
- Exemple : voiture/roue, texte/titre et paragraphe.
- Graphiquement, on ajoute un losange plein du côté de l'agrégat.

IV. Relations entre classes

Exemple :



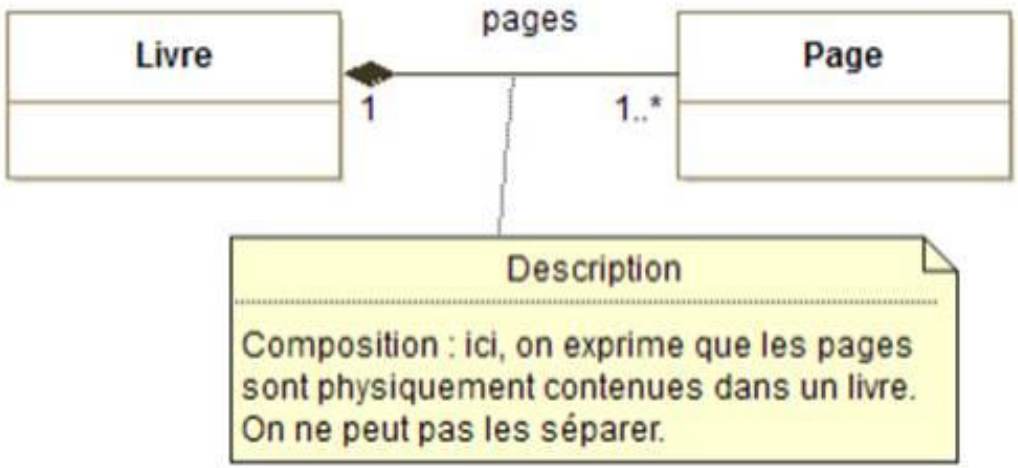
Remarques :

- La destruction de l'objet composite implique la destruction de ses composants.
- Une instance de la partie appartient toujours à au plus une instance de l'élément composite.

IV. Relations entre classes

Exemple :

Composition : les éléments disparaissent quand l'association est détruite.



Si l'objet Livre est effacé, ses pages sont aussi effacées.

IV. Relations entre classes

Association : A est relié à B

Type de la relation non spécifié

Aucune contrainte sur les liens

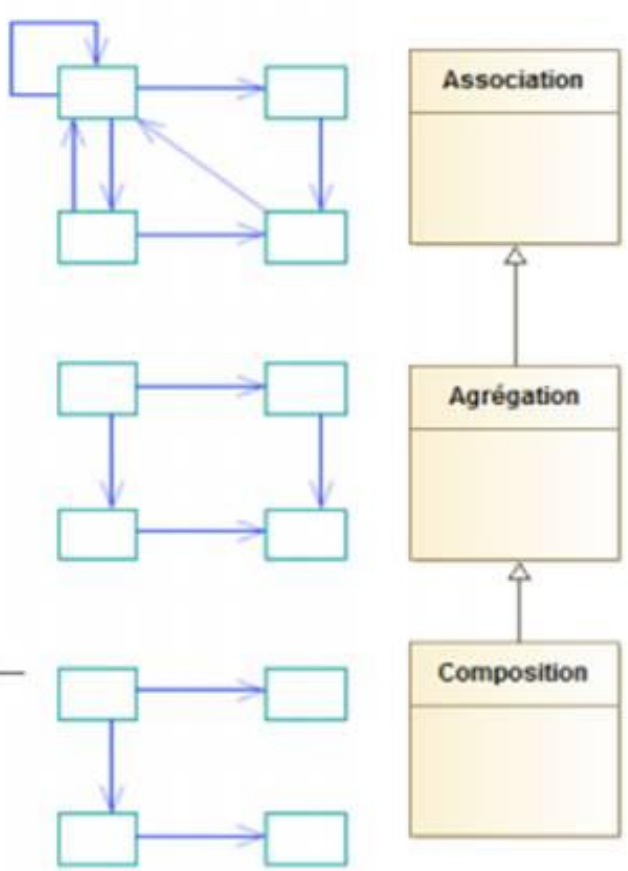
Agrégation : A contient un ou plusieurs B

Association + éléments partageables

Composition : À est composé d'un ou plusieurs B

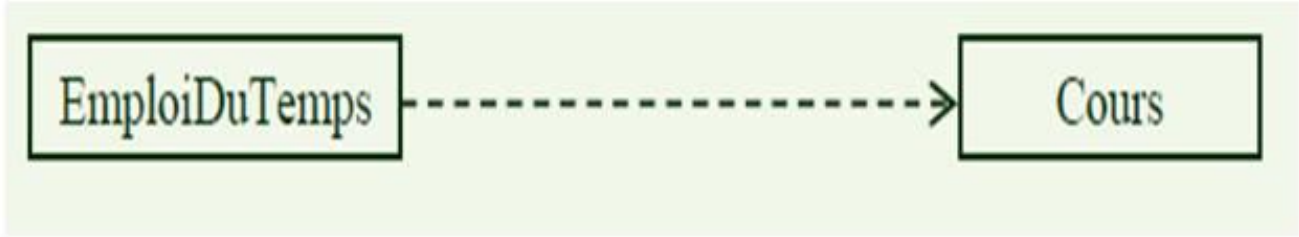
Agrégation + contrainte de durée de vie +

composants non partageables



IV. Relations entre classes

IV.7. Dépendance

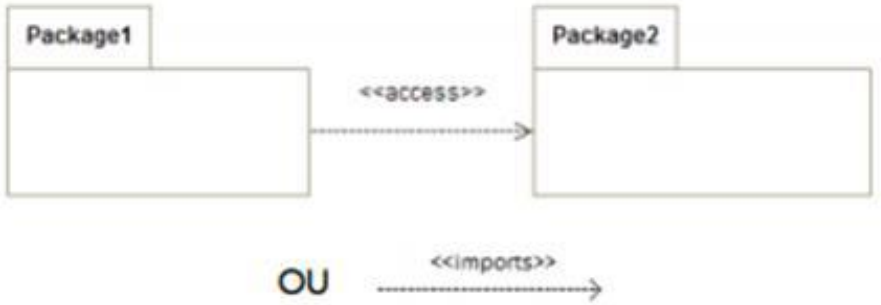


- Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre les éléments du modèle. Dans l'exemple, le déroulement du cours dépend de l'emploi du temps.
- Elle est représentée par un trait discontinu orientée.
- Elle indique que la modification de la cible implique une modification de la source.
- La dépendance est souvent stéréotypée pour mieux expliciter le lien sémantique entre les éléments du modèle : « friend », « derive », « use », « call » ou « bind »

IV. Relations entre classes

Exemple :

Importation : un paquetage ou une classe peut « accéder » un autre paquetage ou classe.



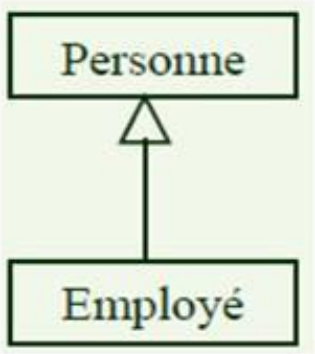
Utilisation : un paquetage ou classe peut utiliser un autre paquetage ou classe.

V. Généralisation et Héritage

- Relation d'héritage :

- Décrit une relation entre une classe générale (classe de base ou classe parent) et une classe spécialisée (sous-classe).
- La classe spécialisée est intégralement cohérente avec la classe de base (héritage), mais comporte des informations (attributs, opérations, associations) supplémentaires (spécialisation).
- Un objet de la classe spécialisée peut être utilisée partout où un objet de la classe de base est autorisé (polymorphisme).
- Graphiquement : flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.

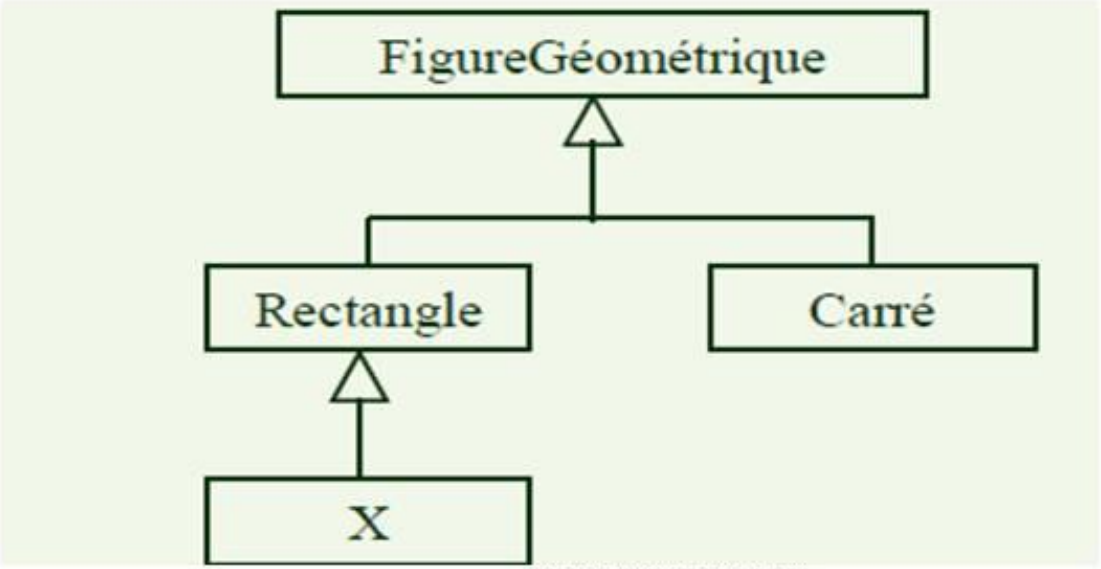
Exemple :



V. Généralisation et Héritage

- **Remarques :**

- Relation d'héritage existe en UML pour : les classes, paquetages, acteurs, cas d'utilisation.
- La classe générale peut être une classe abstraite.
- On peut utiliser {leaf} pour interdire l'héritage (Après le nom d'une classe pour interdire l'héritage à partir d'elle).

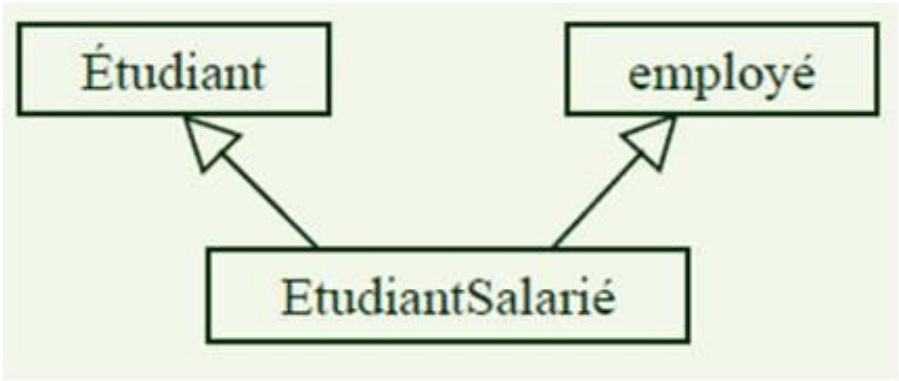


V. Généralisation et Héritage

- Propriétés de l'héritage :

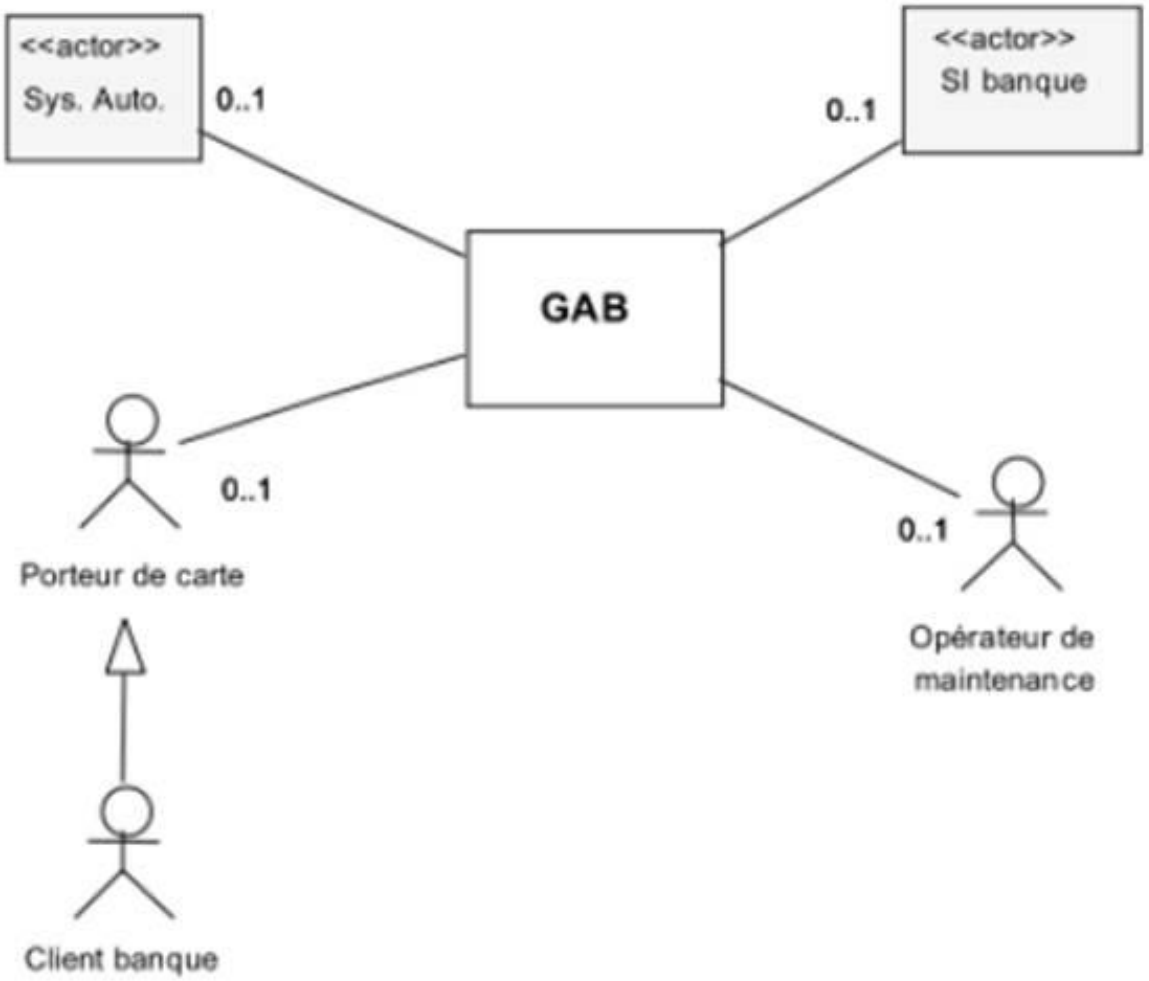
- La classe enfant possède toutes les propriétés des ses classes parents.
- Une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent.
- Un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes.
- Toutes les associations de la classe parent s'appliquent aux classes dérivées.
- Une classe peut avoir plusieurs parents : on parle alors ***d'héritage multiple.***

Exemple :



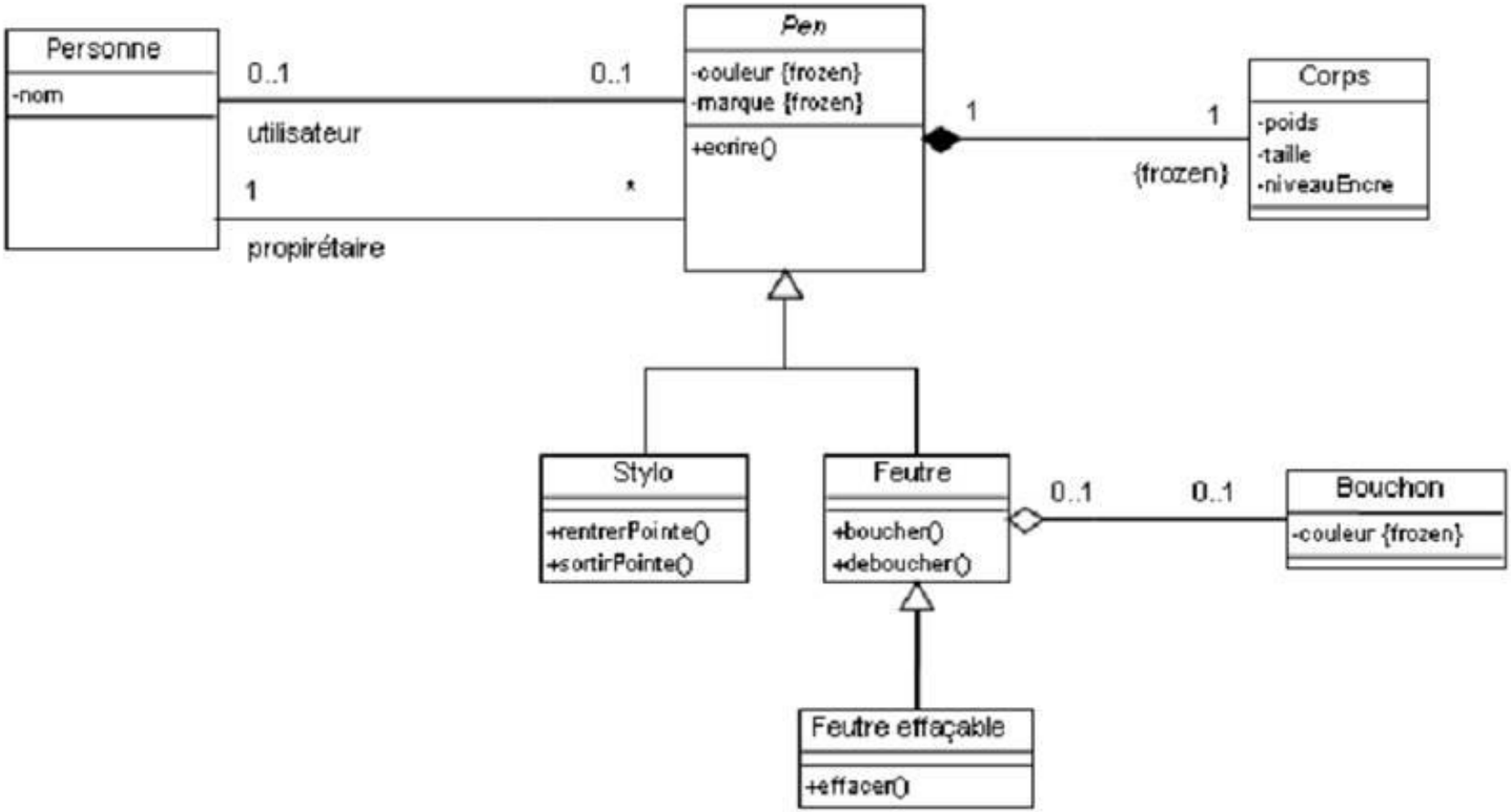
VI. Exemples

Diagramme de classes pour l'étude des besoins (niveau 1)



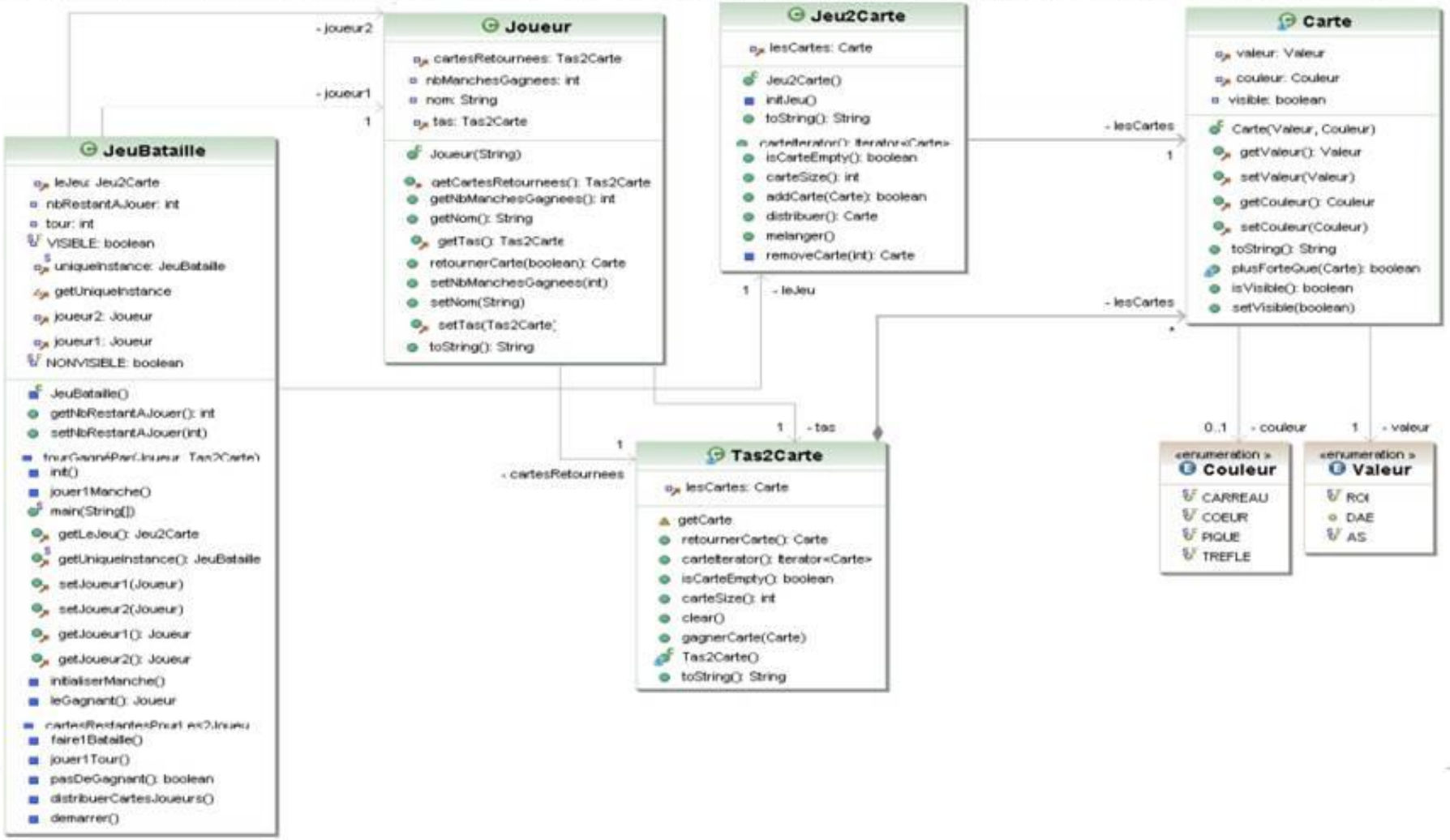
VI. Exemples

Diagramme de classes pour la conception niveau abstrait (niveau 2)



VI. Exemples

Diagramme de classes pour la conception niveau physique (niveau 3)



VI. Exemples

Diagramme de classes pour les bases de données (niveau 2 ou 3)

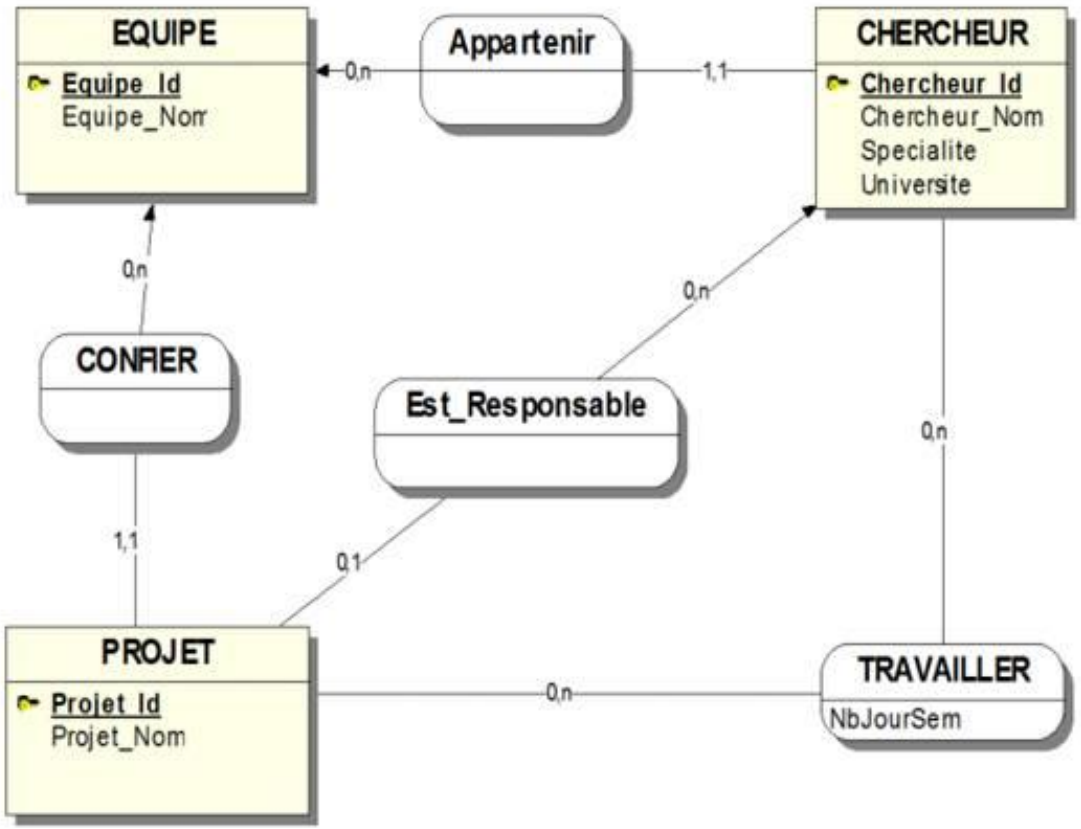


Table => Classes
Colonnes => Attributs

Les associations avec une cardinalité 0 ou 1.. 1 sont traduites en clé secondaire.

Les associations avec une cardinalité multiples sont traduites par une nouvelle table avec 2 clés secondaires.