Université
Internationale
de Casablanca

# Concepts de la Programmation Orientée Objets (POO)

**Pr Imane DAOUDI**
**im.daoudi@yahoo.fr**

# Plan

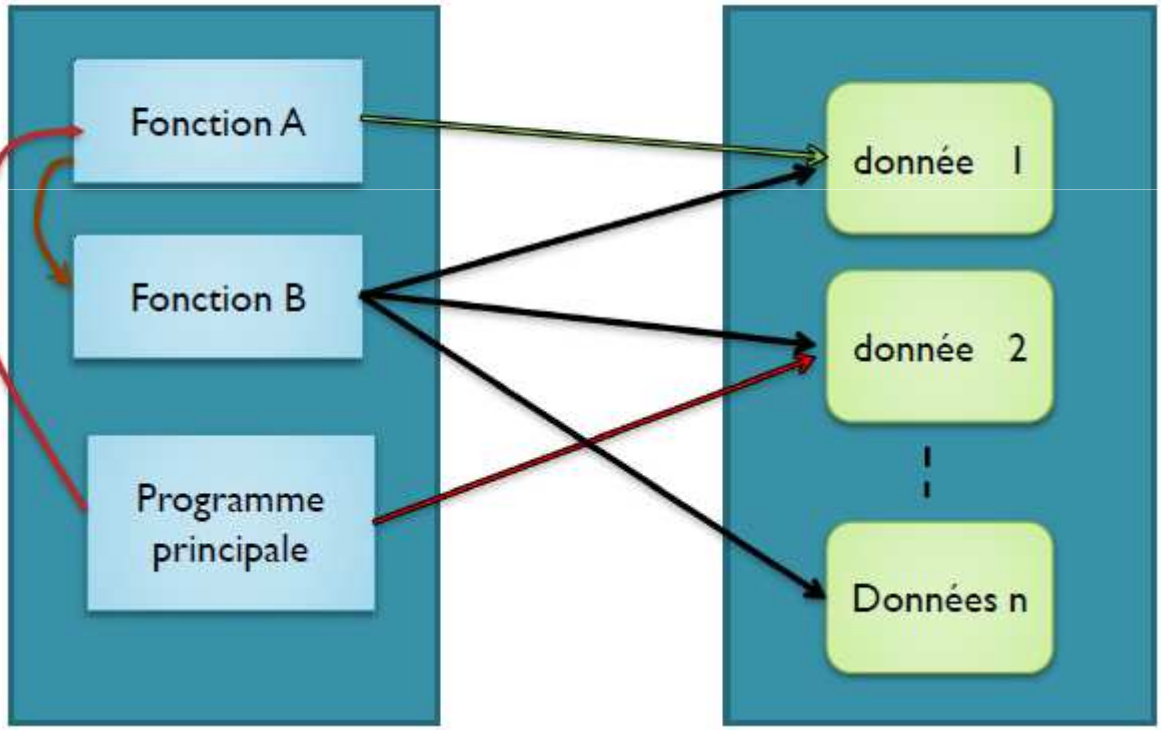
- Introduction
- Programmation procédurale
  - Critiques et limitations
- Programmation orientée objet
  - Concepts objet
  - Les apports de la programmation OO
  - Concepts de la POO
  - Analyse et  conception orientée objet avec UML

# Introduction

- Il existe plusieurs manières pour écrire un programme qui effectue une tâche spécifiée.
  - La manière de programmation dépend du langage utilisé.
  - Le langage utilisé dépend de la manière de programmation
- Paradigme de programmation
  - programmation procédurale : P.P. (Pascal, C, etc.)
  - programmation orientée objet : P.O.O. (C++, Java, python, etc)

# Programmation procédurale

- Rappel du C
  - le programme est composé des fonctions
  - les données (variables) sont créées à l'intérieur des fonctions ou bien passées comme paramètres
  - il y a un programme principal (main)



# Programmation procédurale

- Limitation de la programmation procédurale(1)
  - La programmation procédurale réunit au sein des modules des structures de données et des procédures applicables sur ces dernières.

> Un programme= algorithme + structure de données

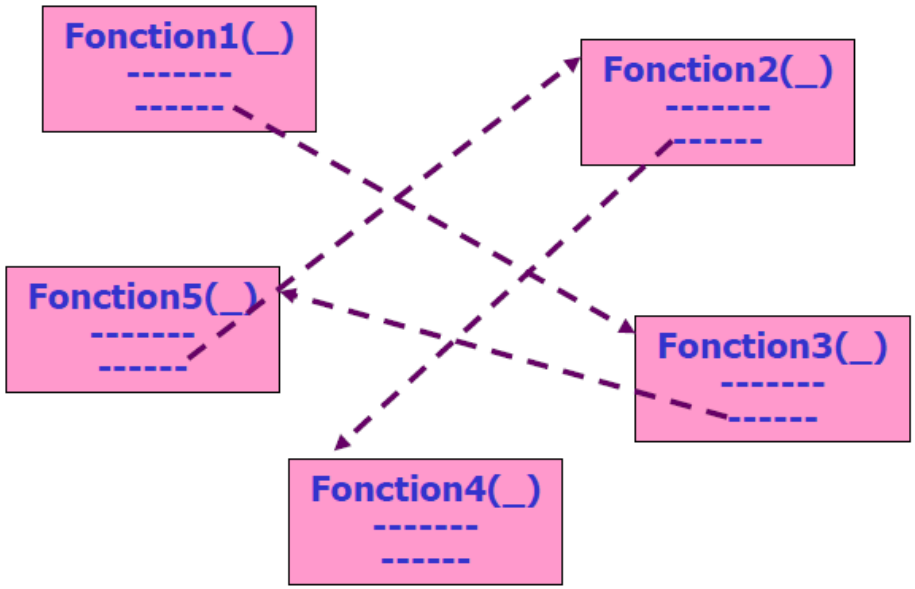
  - Que se passe-t-il si on veut modifier une structure de données ?

> La remise en cause d'une structure de données engendre la remise en cause du module en entier, voire des programmes qui dépendent du module

  - Les modifications d'une fonction entraînent d'autres modifications dans d'autre fonctions, etc.
  - La portée d'une modification est trop grande et difficile à gérer
  - Redondance dans le code (la même chose est codé plusieurs fois)

# Programmation procédurale

- Limitation de la programmation procédurale(2)
  - **Manque de réutilisabilité**
    - On ne peut pas réutiliser les fonctions à cause des structures de données
      - Impossible de réutiliser une fonction existante qui opère sur des structures de données différentes
  - **Conception "plat de spaghettis" des fonctions**



# Programmation procédurale

- Limitation de la programmation procédurale(3)
  - **Conception "plat de spaghettis" des fonctions**

    - **Pb** : Comment identifier la responsabilité d'une fonction? Qui fait quoi?
    - Les fonctions s'appellent entre elles. En cas de modification d'une fonction:
      - Identifier tous les endroits d'appel de cette fonction dans le programme (tâche délicate)

  - **Pas de protection des données**

    - Toute fonction peut accéder et agir sur les données

**Peut – on faire mieux?**

# Programmation Orientée Objet

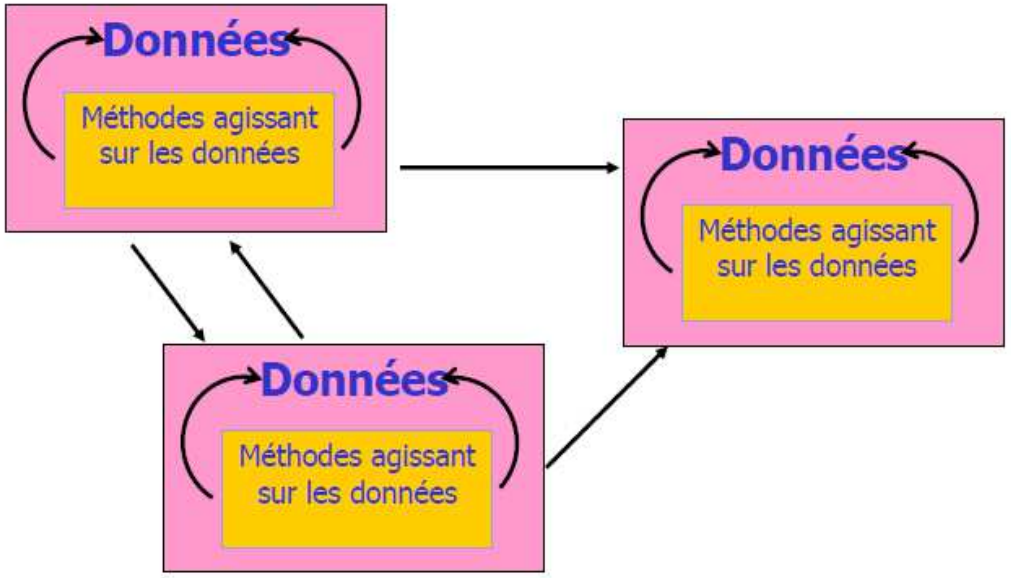
- Solution: la programmation OO

- Intégration des données et des traitements (**méthodes**) au sein d'une même entité : **objet**

Un objet= méthodes + données

- Un programme = ensemble de petites entités (objets) qui interagissent et **communiquent** par messages

- Architecture des objets



# Programmation Orientée Objet

➢ Concepts d'objet

✓ Encapsulation

- La partie données d'un objet est inaccessible (**cachée**) de l'extérieur.
  - Seuls les méthodes de cet objet peuvent y accéder.
  - Offre une meilleure protection des données

✓ Abstraction

- L'interaction d'un objet vis-à-vis de l'extérieur est définie par un ensemble de méthodes (interface).
  - Ceci nous épargne de connaître la structure interne de l'objet et le détail d'implémentation de ses méthodes pour communiquer avec lui.
  - L'objet est défini par son comportement et non par sa structure

# Programmation Orientée Objet

- Les apports de la programmation OO

- Plus de modularité :
  - l'unité de modularité est l'objet
    - plus de réutilisabilité et d'extensibilité
- Plus de sécurité
  - Protection des données de l'objet
  - Interaction avec l'objet est définie par une interface
- Meilleure conception
  - Les données et méthodes sont spécifiées en même temps
- Meilleure lisibilité
  - Les données et méthodes sont spécifiées au même endroit

# Programmation Orientée Objet

- Concepts de la programmation OO
  - Classe et objets
  - Message et méthodes
  - Héritage (simple, multiple)
  - Polymorphisme
  - Agrégation (composition)

# Programmation Orientée Objet

- Notion d'objet

> Un objet est un ensemble des propriétés ayant des valeurs et des actions (méthodes) agissant sur les valeurs de ces propriétés.

- Dans le monde réel un objet :
  - Une voiture, un téléphone, un livre,…
- Chaque objet a deux caractéristiques:
  - Des propriétés → Partie statique
  - Des opérations → Partie dynamique

**Le mécanisme** par lequel l'exécution d'un programme produit un objet à partir d'une classe constitue **« l'instanciation ».**

# Programmation Orientée Objet

- Notion d'objet
- Exemple 1: l'objet fenêtre

  - propriétés d'une fenêtre
    - ouverte/fermée
    - cassée/intacte
    - taille
    - sens d'ouverture
    - type de verre
    - coefficient de réflexion de chaleur

- Remarque :Pour une fenêtre concrète, ces propriétés ont des valeurs.

  - opérations avec une fenêtre donnée
    - ouvrir
    - fermer
    - casser
    - réparer
    - changer le verre

# Programmation Orientée Objet

- Notion d'objet
- Exemple 2: l'objet livre

  - propriétés d'un livre dans une bibliothèque
    - état (emprunté / disponible / perdu)
    - date de la fin de l'emprunt
    - titre
    - auteur
    - nombre de pages

- Remarque : Pour un livre donné, ces propriétés ont des valeurs!

  - opérations sur un livre d'une bibliothèque
    - emprunter
    - rendre
    - perdre
    - voler

# Programmation Orientée Objet

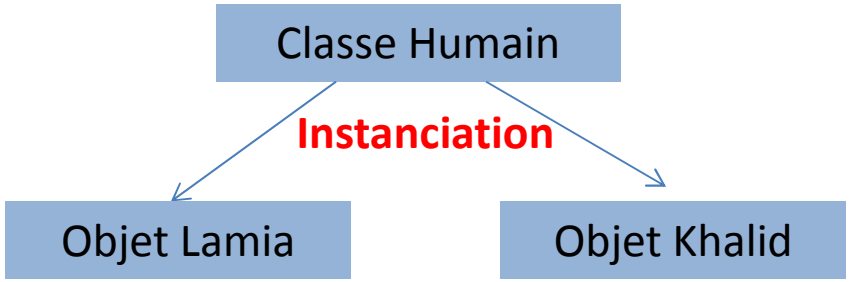
## ➢ Notion de classe

> Une classe est une description d'un ensemble d'objets ayant une structure de données commune et disposant des mêmes méthodes.

– Remarques:

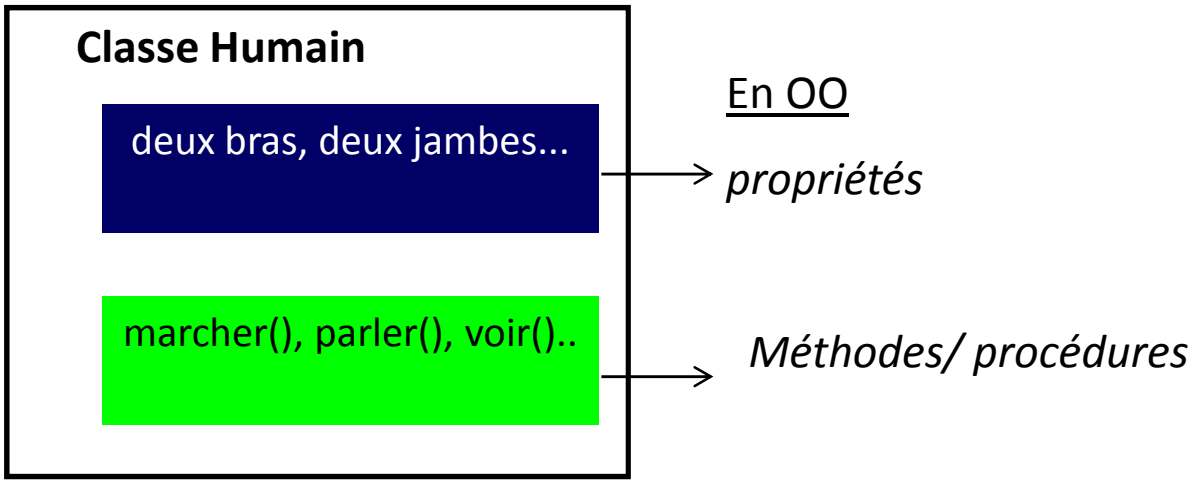
- Les objets ayant les mêmes propriétés et les mêmes méthodes peuvent être mis dans une classe
- Les objets apparaissent alors comme des variables d'un tel type classe. On dit aussi qu'un objet est une **instance** de sa classe.

– Exemple



# Programmation Orientée Objet

- Notion de classe
  - Une classe encapsule des propriétés et des comportements. Par exemple, la classe Humain définit des propriétés ( deux bras, deux jambes...) et des comportements ( marcher, parler, voir...).
  - Elle possède deux composantes :
    - **composante statique** : description des données,
    - **composante dynamique** : description des procédures, appelées méthodes
  - Exemple




⇨ ***Une classe génère une famille d'objets***

# Programmation Orientée Objet

- Classe des livres

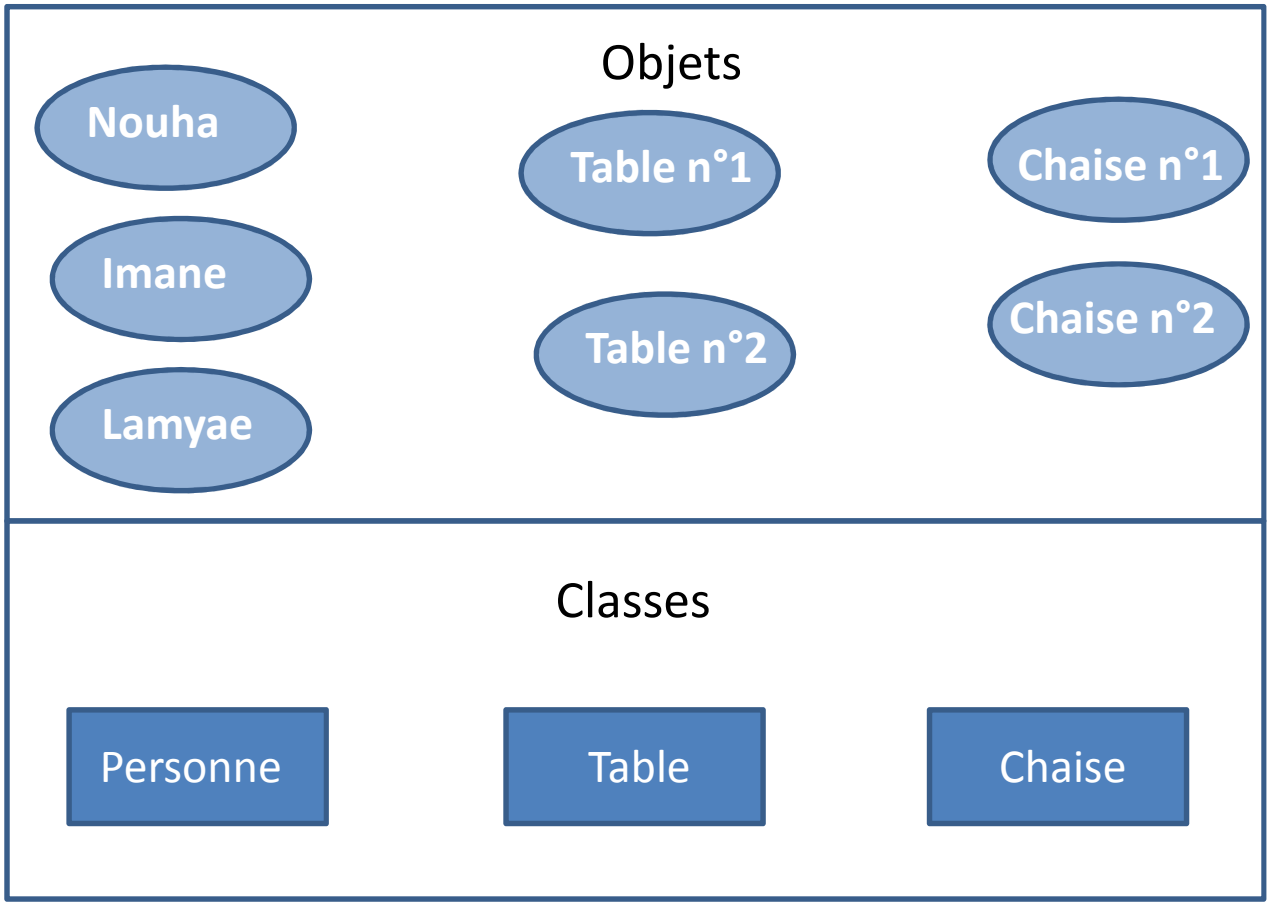
| propriétés | méthodes |
| --- | --- |
| état (emprunté / disponible / perdu)<br>date de la fin de l'emprunt<br>titre<br>auteur<br>nombre de pages | emprunter<br>rendre<br>perdre<br>voler |

- Classe des fenêtres

| propriétés | méthodes |
| --- | --- |
| état d'ouverture (ouverte/fermée)<br>état (cassée/intacte)<br>taille<br>sens d'ouverture<br>type de verre<br>coef de réflexion de chaleur | ouvrir<br>fermer<br>casser<br>réparer<br>changer de verre |

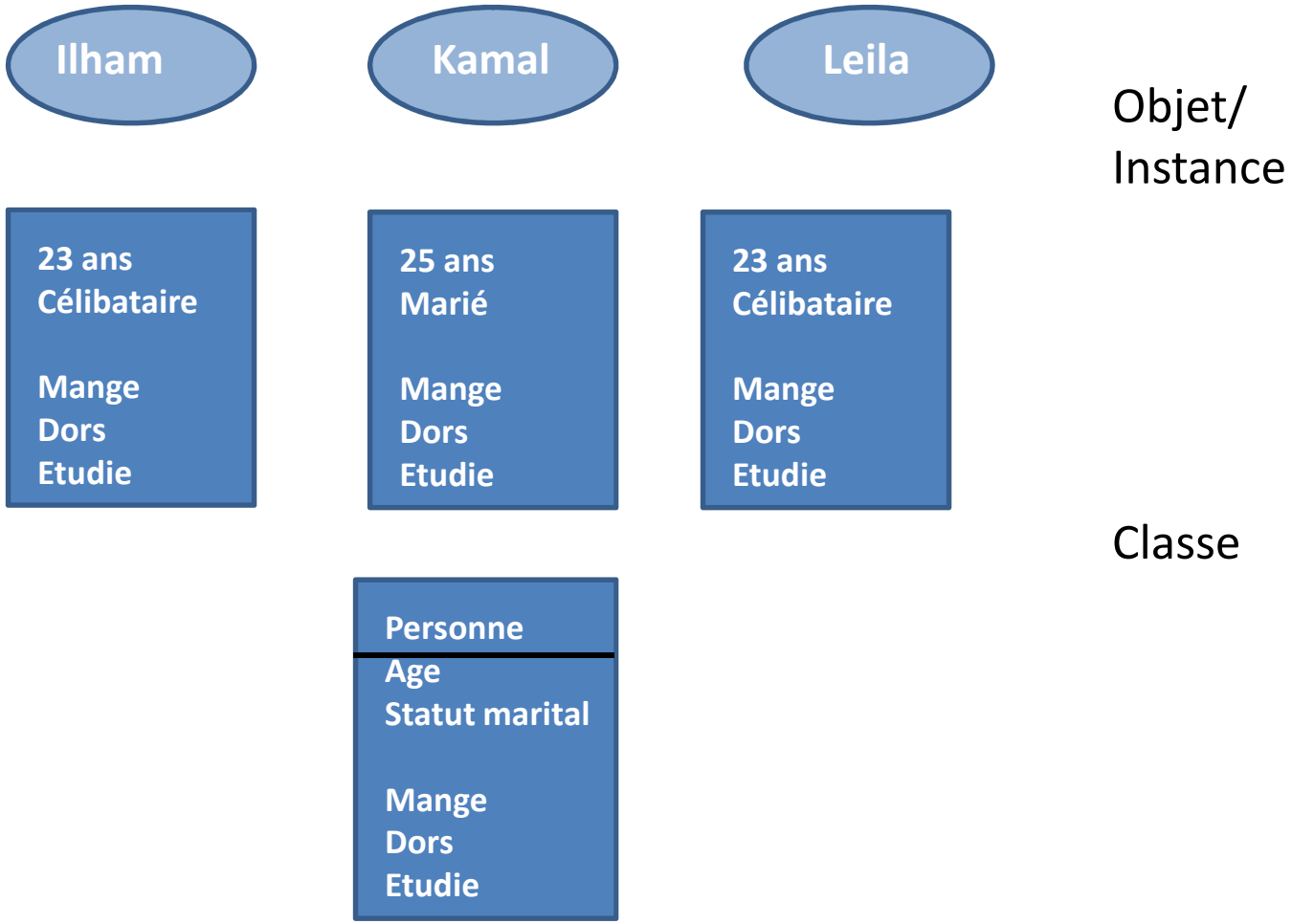
# Programmation Orientée Objet

- Classe/Objet



# Programmation Orientée Objet

- Classe/instance

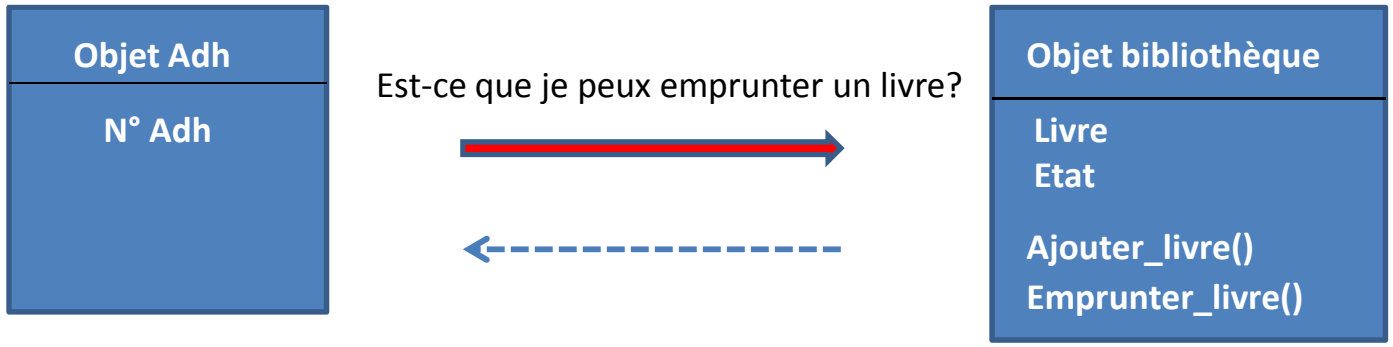


# Programmation Orientée Objet

- Méthodes et envoi de message

  - Les objets communiquent entre eux par échange de messages
  - Le message le plus échangé est la demande de réalisation de traitement

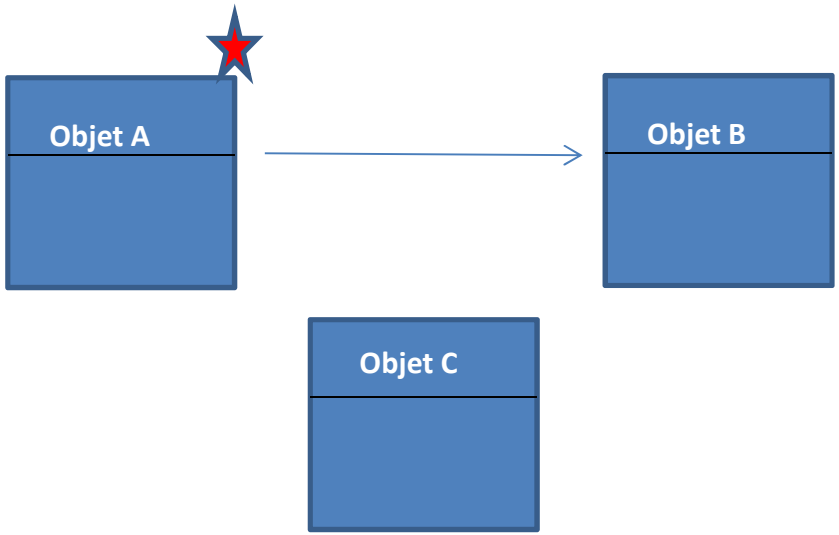
  - Exemple



# Programmation Orientée Objet

- Méthodes et envoi de message

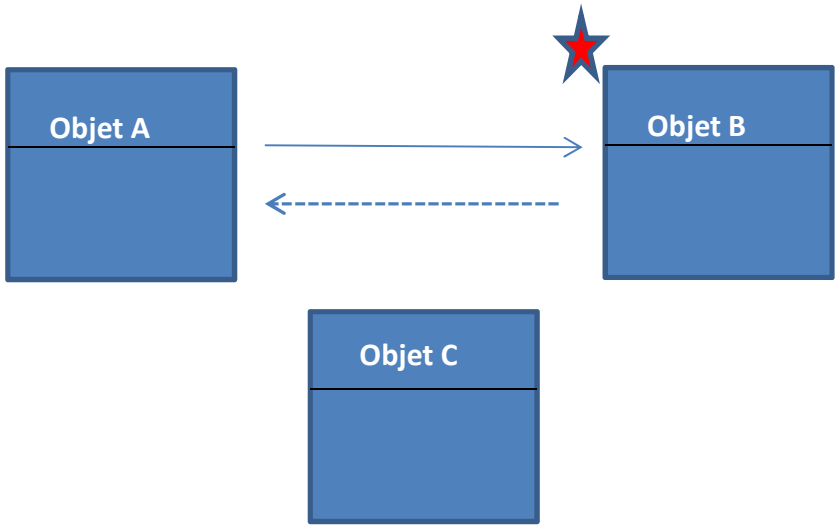
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose



# Programmation Orientée Objet

- Méthodes et envoi de message

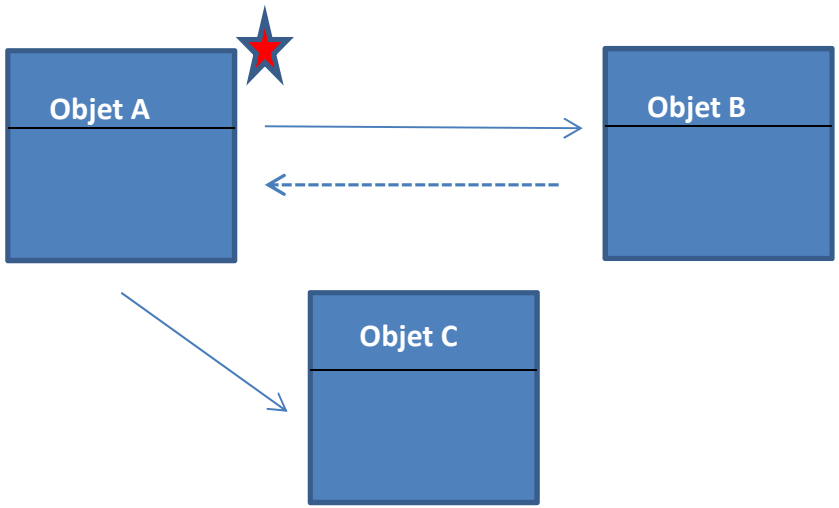
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose



# Programmation Orientée Objet

- Méthodes et envoi de message

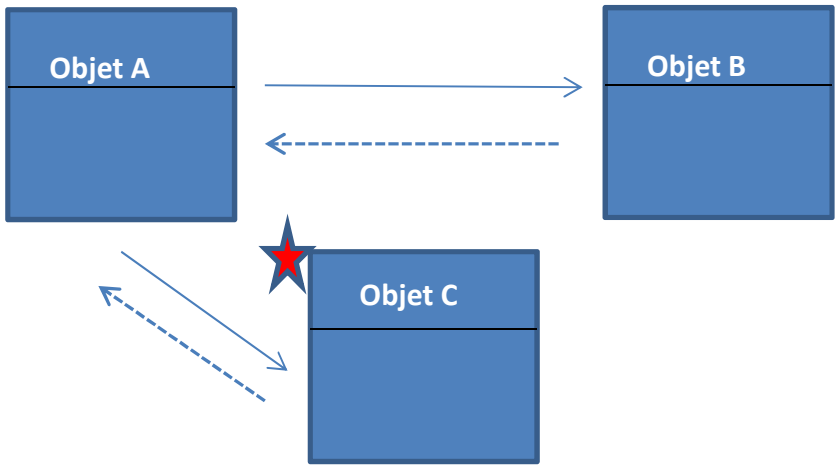
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose



# Programmation Orientée Objet

- Méthodes et envoi de message

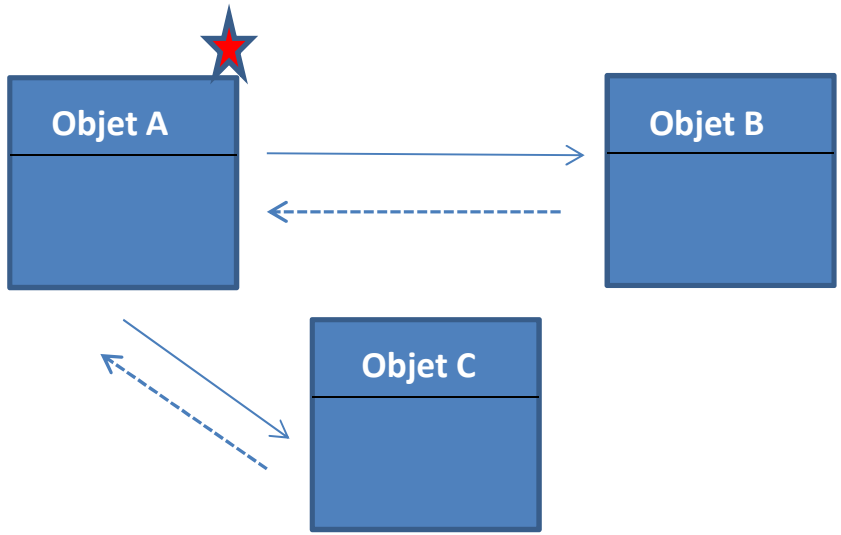
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose



# Programmation Orientée Objet

- Méthodes et envoi de message

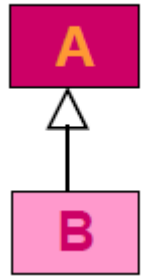
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose



# Programmation Orientée Objet

## ➢ Héritage simple

- L'héritage permet de définir une classe à partir d'une classe existante en lui ajoutant de nouveaux attributs et de nouvelles méthodes
- Une classe B qui hérite d'une classe A dispose implicitement de tous les attributs et de toutes les méthodes de la classe A
- La classe A est dite la super-classe ou la classe de base
- La classe B est dite la sous-classe ou la classe dérivée

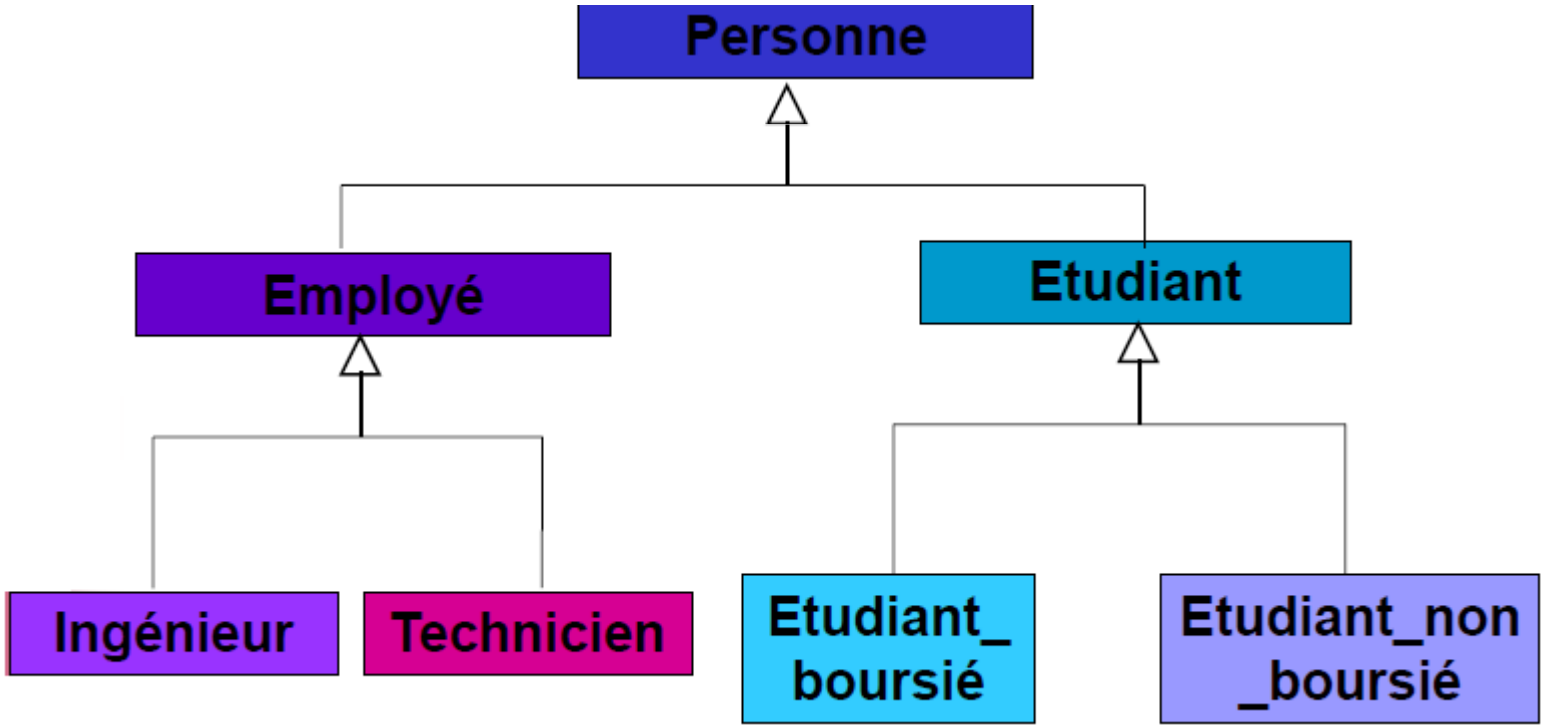


# Programmation Orientée Objet

- Héritage simple

- Le concept d'héritage n'est pas limité à un seul niveau. Une hiérarchie de classe peut être définie

- Exemple

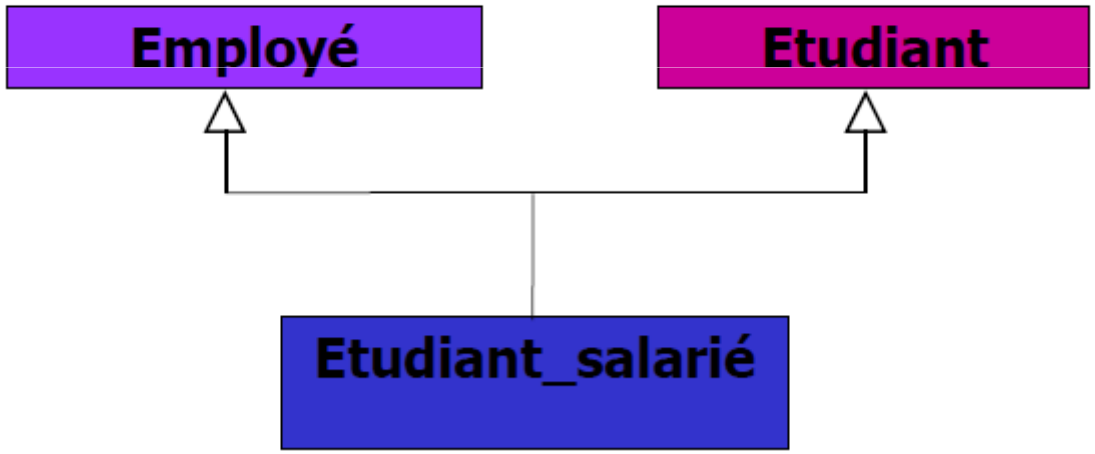


# Programmation Orientée Objet

- Héritage multiple

- Le concept d'héritage n'est pas limité à un seul niveau. Une hiérarchie de classe peut être définie

- Exemple

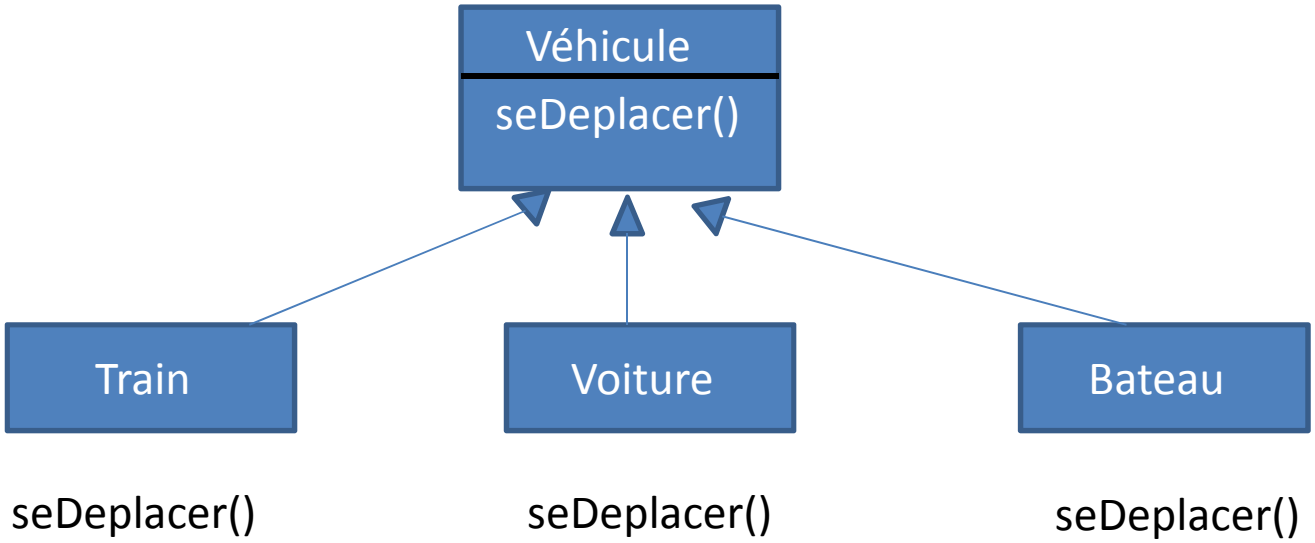


# Programmation Orientée Objet

- Polymorphisme

- Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Le polymorphisme augmente la généricité du code

- Exemple



# Programmation Orientée Objet

## ➢ Agrégation (composition)

### ✓ Définition:

- L'agrégation permet de définir qu'un objet est l'assemblage d'un ou de plusieurs sous-objets

### ✓ En POO

- L'objet contient les sous-objets
- L'Objet communique lui même avec les sous-objets

### ✓ Règles à suivre

- Les sous-objets doivent avoir une relation structurelle ou fonctionnelle avec l'objet dont ils sont les constituants. Ex : un clavier fait partie d'un ordinateur
- L'agrégation respecte les contraintes suivantes: Antisymétrie et transitivité

# Programmation Orientée Objet

- Composition

- ✓ La composition est un cas particulier d'agrégation qui implique une dépendance plus forte de l'objet agrégé dans l'objet agrégeant.
- ✓ Un objet peut être agrégé (partagé) par plusieurs objets, mais ne peut être composé que par un objet unique.
- ✓ Notion de durée de vie : en général les objets qui composent un objet A sont détruits en même temps que A.

# Programmation Orientée Objet

- Composition

- ✓ Exprime la relation " **est composé de …**"
- ✓ L'un des attributs de la classe est une instance d'une autre classe
- ✓ Alternative à l'héritage, mais sémantique différente

- Exemple

- ✓ Un bâtiment est composé d'un certain ensemble de salles

# Plan

- Introduction
- Programmation procédurale
  - Critiques et limitations
- Programmation orientée objet
  - Concepts objet
  - Les apports de la programmation OO
  - Concepts de la POO
  - L'analyse et conception orientée objet avec UML

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

**UML**

**Langage visuel** dédié à la **spécification**, la **construction** et la **documentation** des artefacts d'un système logiciel

– Représentation à travers des digrammes

- Diagrammes de cas d'utilisation
- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de séquence
- Etc.

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

**Diagramme de classe**

Représentent un ensemble de classes, ainsi que leurs relations. Ils présentent la vue de conception statique d'un système.

- Utilisation
  - Lors de L'analyse et de la conception
    - Définition formelle des objets qui composent le système à partir des cas d'utilisation et des diagrammes d'interaction (séquences et collaboration).
  - Lors de l'implémentation
    - Génération automatique des structures statiques du système (classes, relations, ...).

# Programmation Orientée Objet

- Diagramme de classe

**Classe**

Description d'un ensemble d'objets partageant les mêmes attributs, méthodes, et relations .

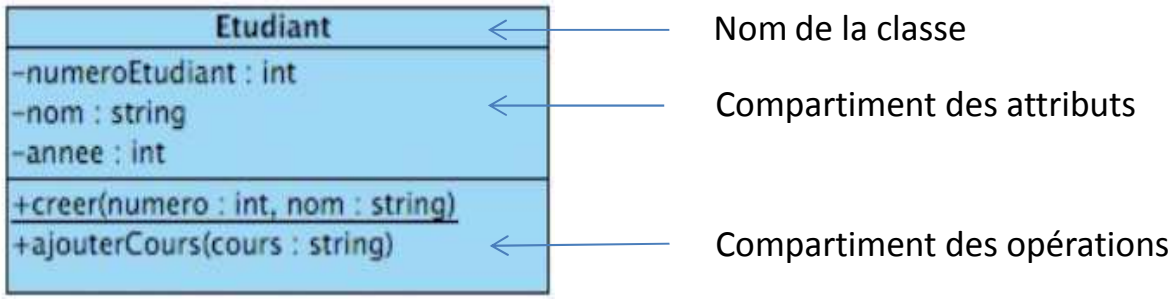
- Exemples

➢ Le client est une classe

➢ La commande est une classe

➢ La classe Personne(nom, prénom), et la classe Voiture(nom , puissance fiscale) ne peuvent pas être groupés en une même classe car ils ne partagent pas les mêmes propriétés

- Représentation graphique d'une classe



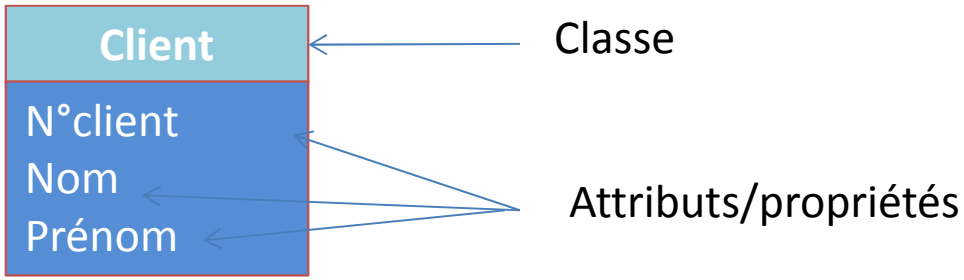
# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

**Attribut (classe)**

**Propriété d'une classe caractérisée par un nom et un type élémentaire.**

- Est un élément d'une classe :
  - a un nom unique,
  - permet de mémoriser une valeur,
  - doit avoir un sens (donc une valeur) pour chacune des instances de la classe.
- Exemple




**Représentation graphique d'une classe comportant trois attributs**

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

**Règles concernant les attributs**

**Règle 1**

**Un attribut ne peut en aucun cas être partagé par plusieurs classes.**

**Règle 2**

**Une entité et ses attributs doivent être cohérents entre eux (i.e. ne traitent qu'un seul sujet).Ex: l'attribut "nomClient" dans la classe "CompteBancaire"**

**Règle 3**

**Se méfier des attributs multi-valués, ils cachent souvent eux-mêmes une classe Ex: l'attribut "enfants" dans la classe « Personne »**

**Règle 4**

**Se méfier des attributs structurés, ils cachent souvent eux-mêmes une classe Ex: l'attribut "adresse" dans la classe « Personne »**

**Règle 5**

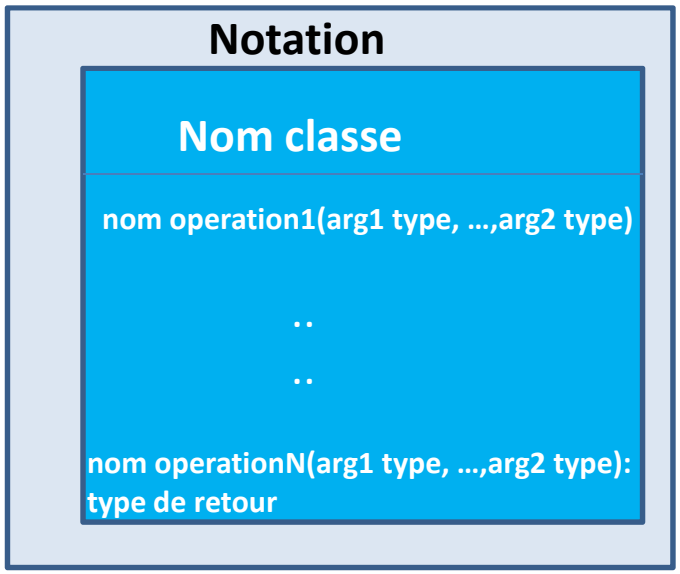
**Un attribut est une donnée élémentaire, ce qui exclut des données calculées .**

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML : opération de la classe

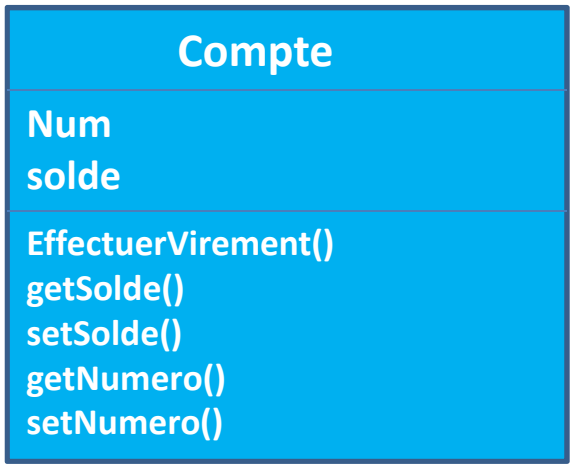
Une opération est un **service** que propose une classe sur son **interface**

- L'opération possède un nom **pas forcément unique** dans la classe
- On peut associer à l'opération ses **arguments**
- On peut associer à l'opération son **type de retour**



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML : opération de la classe
- Exemple de classe




- Les getters sont des fonctions qui permettent la récupération de la valeur d'une variable déclarée « private » dans la classe
- Les setters permettent d'affecter une valeur à l'un des attributs de la classe. Par exemple, pour modifier l'âge d'une personne, on passera par son setter en lui passant en paramètre la nouvelle valeur.

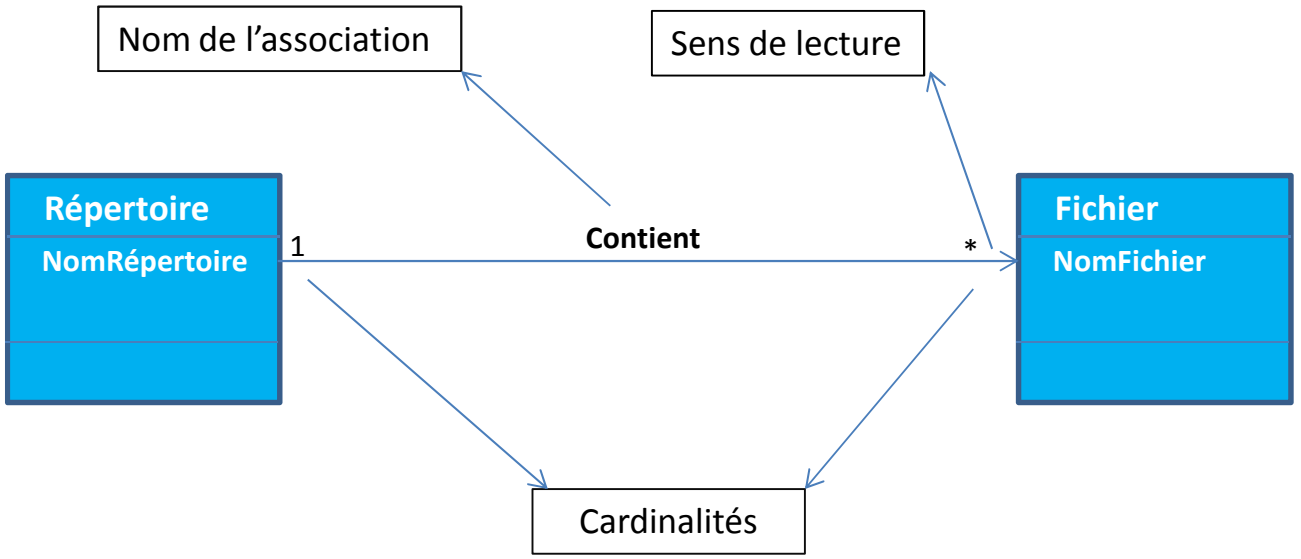
# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

**Association**

Exprime une relation permanente entre instances de 2 ou plusieurs classes.

- Elle est caractérisée par :
  - Nom de l'association
  - Sens de lecture
  - Cardinalités

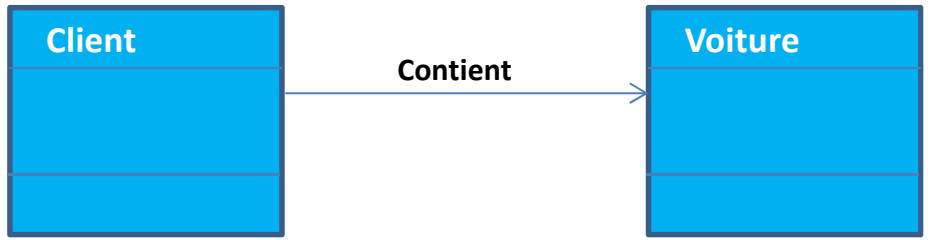


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## Nommage des associations :

- Le nom de l'association est en général une forme verbale active ou passive qui décrit globalement le lien
- Le nom de l'association est facultatif
- Le nom doit apparaitre sur l'association, mais ne doit pas être rattaché à l'une des extrémité

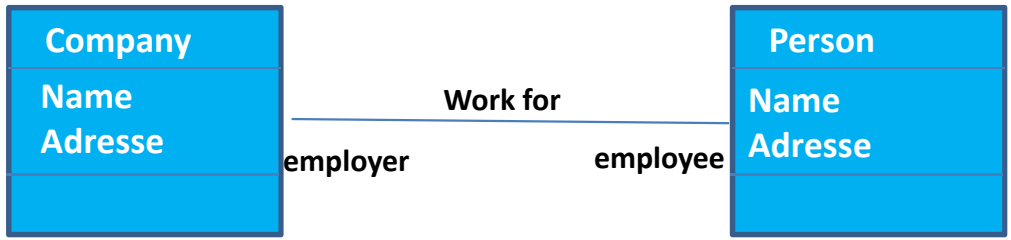


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Rôles de l'association :

- Le rôle permet de décrire, à l'aide d'un nom, comment une classe perçoit une autre classe au travers de l'association
- Un rôle doit figurer à l'extrémité de l'association qu'il qualifie
- Les rôles sont facultatifs
- L'association peut faire figurer les deux, un seul ou aucun des rôles
- Exemple:



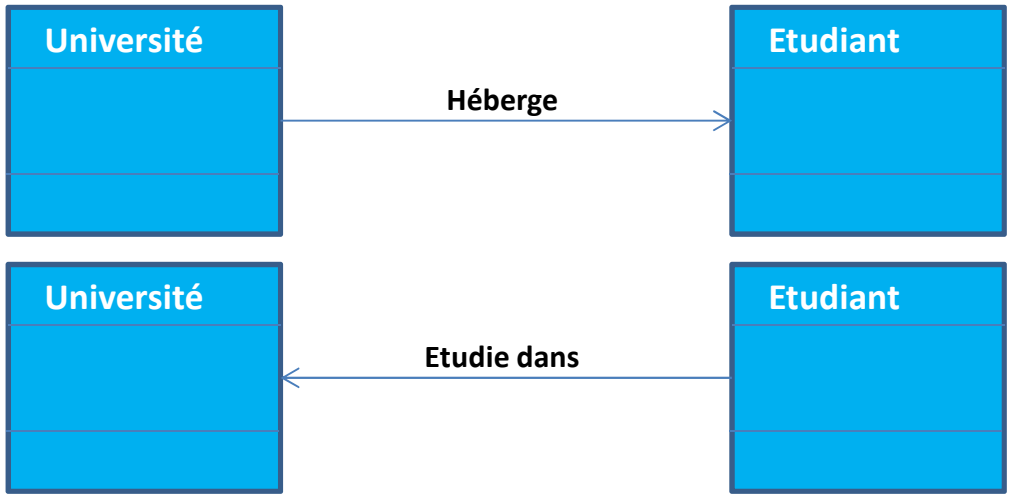

- Une personne peut est considérée comme employé du point de vue de l'entreprise
- L'entreprise est considérée comme employeur du point de vu de la personne
- Rôle : identifie l'extrémité d'une association

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Sens des associations :

- La navigabilité (ou sens) de l'association permet de définir dans quel sens l'association peut être parcourue
- La navigabilité d'une association est modélisée par une flèche sur l'extrémité pouvant être atteinte par navigation
- La navigabilité peut être bidirectionnelle. L'absence de flèche sur les deux extrémités signifie que l'association est bidirectionnelle
- Exemple



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Sens des associations :

- Indication du sens de lecture: exemple 1
  - ✓ étant donné un utilisateur, on désire pouvoir accéder à ses mots de passe
  - ✓ étant donné un mot de passe, on ne souhaite pas pouvoir accéder à l'utilisateur correspondant


- Exemple 2:
  - ✓ Si la flèche est absente, la navigation est bidirectionnelle

# Programmation Orientée Objet

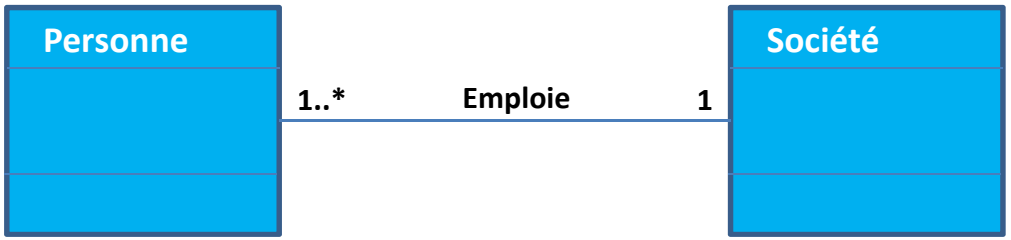
- L'analyse et conception orientée objet avec UML

## ❑ Cardinalités :

**Cardinalité**

**La multiplicité est associée à une extrémité de l'association et indique combien d'instances de la classe considérée peuvent être liées à une instance de l'autre classe**

| Cardinalités | Signification |
|---|---|
| 1 | Un et un seul |
| 0..1 | Zéro ou un |
| N | Exactement N |
| M..N | De M à N |
| * | De zéro à plusieurs |
| 0..* | De zéro à plusieurs |
| 1..* | De 1 à plusieurs |
| N..* | N ou plus |




- Une société emploie de une à plusieurs personnes
- Une personne est employée par une seule société


- Un client achète zéro à plusieurs voitures
- Une voiture peut être achetée par un client au plus

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Cardinalités :

règles

**Règle 7: L'expression de la cardinalité est obligatoire pour chaque patte d'une association**
**Règle 8: Une cardinalité minimal est toujours 0 ou 1, et une cardinalité maximale est toujours 1 ou n**

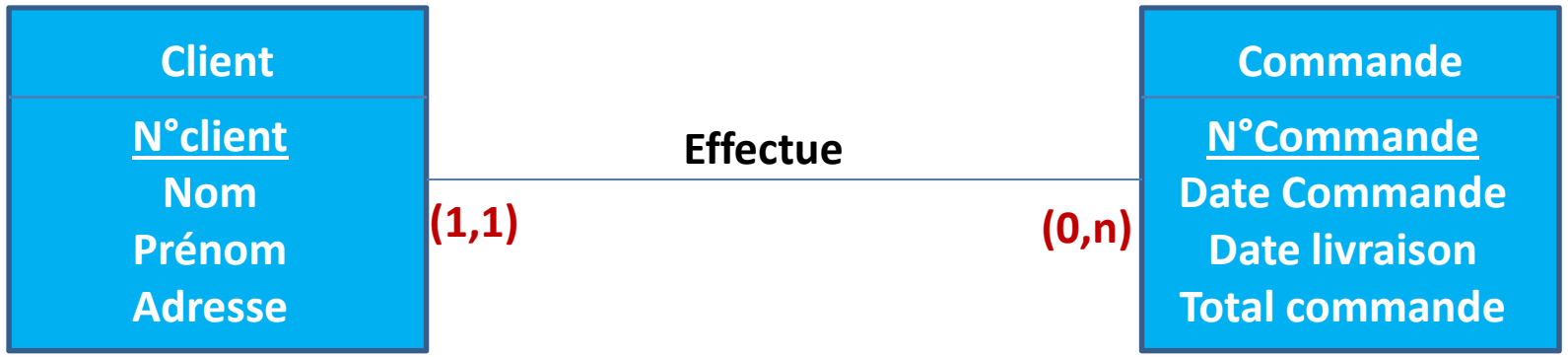
❑ Remarques

- Une cardinalité maximale de 0 n'a pas de sens
- Si une cardinalité maximale est connu et vaut 2, 3 ou plus, alors nous considérons qu'elle est indéterminée et vaut n
- Les cardinalités minimales qui valent plus de 1 sont modélisées par 1

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Cardinalités :




❑ Remarques

- Sur l'extrémité client, le 0 signifie que le client peut ne pas être reliée à la commande lors de sa création.

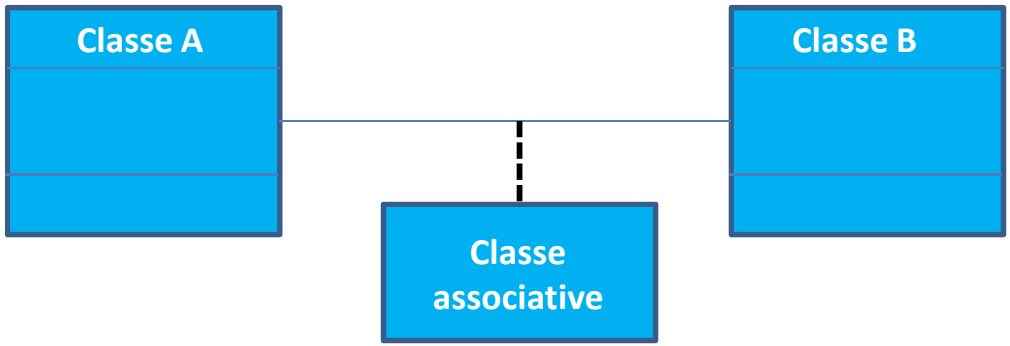
- Le 1 en minimum de l'extrémité commande signifie qu'en aucun cas on ne peut créer une instance de la classe commande sans la relier en même temps à une occurrence de la classe client…Cette dernière doit donc avoir été créée avant !

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## ❑ Classe-association :

- Une association peut être matérialisée par une classe dans une des circonstances suivantes
  - Si l'association est porteuse d'attributs
  - Si l'association se matérialise par un objet concret dans le monde réel
  - Si l'association est de multiplicité M..N



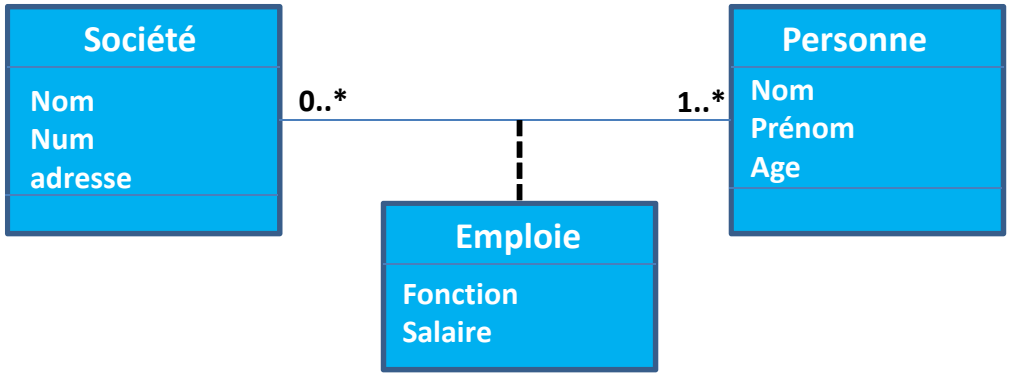

- Une classe-associative est une classe à part entière
- Elle est modélisée par un lien en pointillé allant de la classe vers l'association concernée

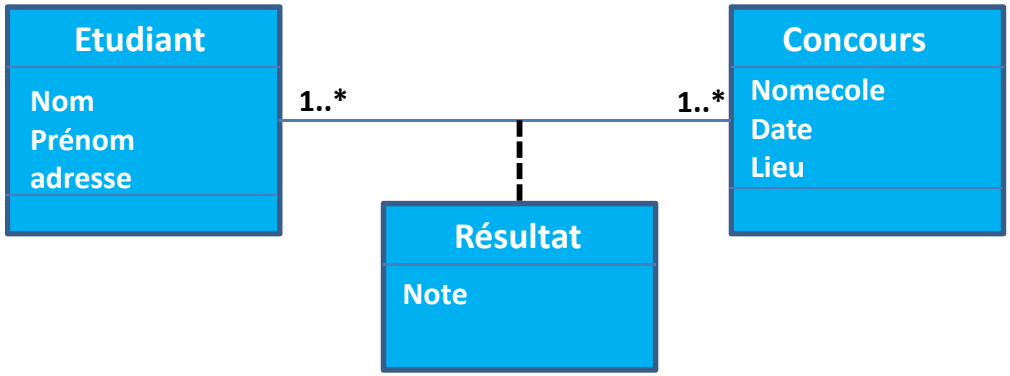
# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## ❑ Classe-association :

- Exemples:



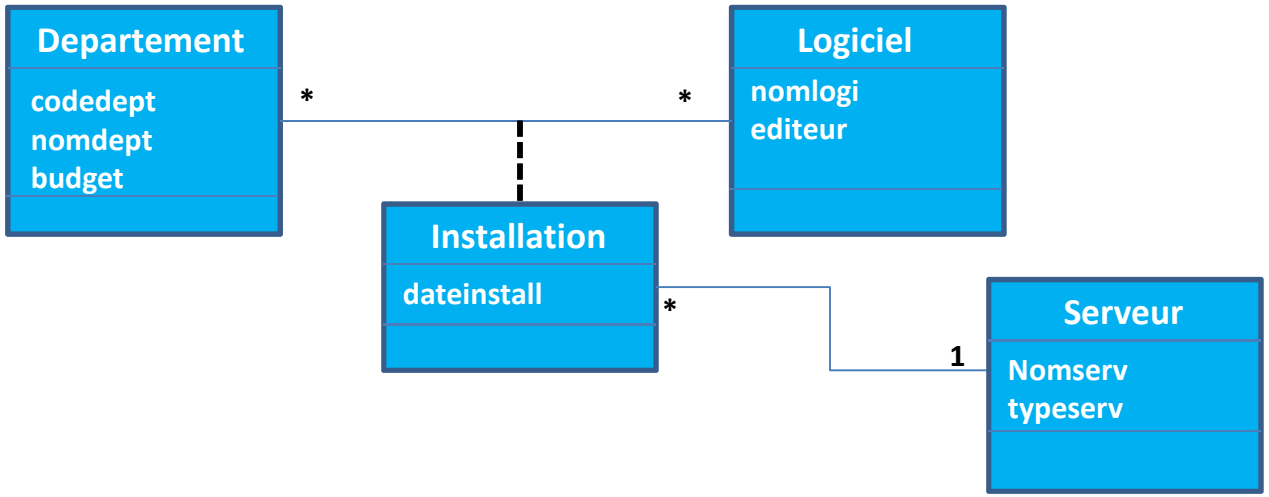
# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## ❑ Utilisation d'une classe-association :

✓ Une classe association peut participer à d'autres relations:

- L'exemple met en œuvre la classe-association **Installation** en liaison avec la classe **Serveur**. Il met en évidence qu'une installation (constituée d'un logiciel et d'un département à une date donnée) est associée à un seul serveur (multiplicité 1 du côté Serveur).
- Contrainte d'unicité: un logiciel d'un département n'est installé que sur un seul serveur.

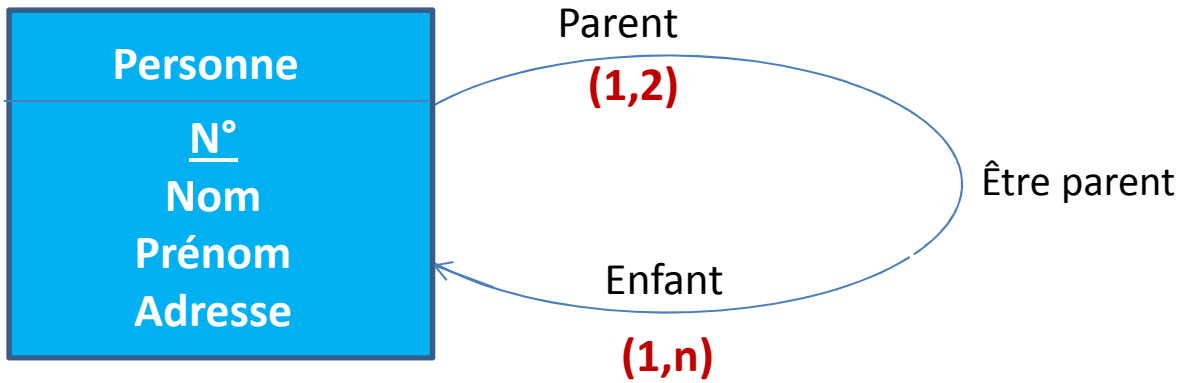


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Association réflexive

**Une association réflexive est une association reliant des instances de la même classe**



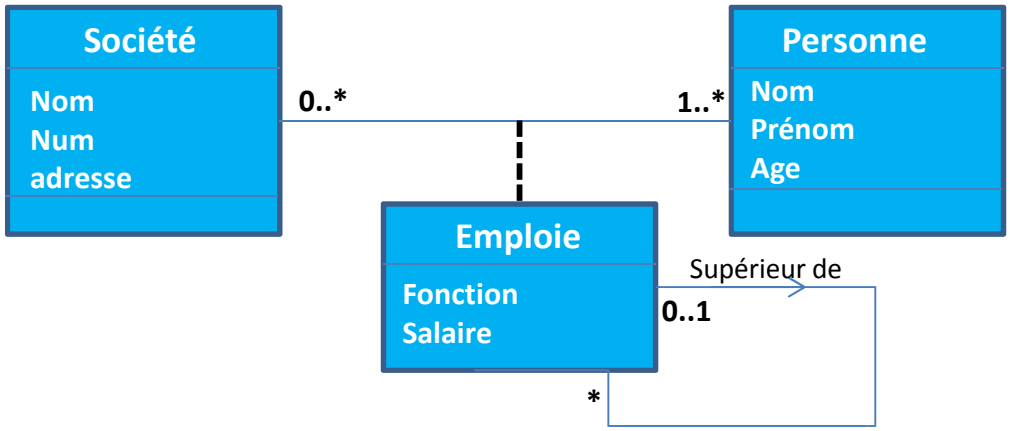
# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Auto-association sur classe-association :

✓ Exemple:

- pour préciser qu'une personne est le supérieur d'une autre personne. On ne peut pas ajouter une association réflexive sur la classe *Personne*. En effet, une personne n'est pas le supérieur d'une autre dans l'absolu. Une personne est, en tant qu'employé d'une entreprise donnée, le supérieur d'une autre personne dans le cadre de son emploi pour une entreprise donnée. Il s'agit donc d'une association réflexive, non pas sur la classe *Personne,* mais sur la classe-association *Emploie*

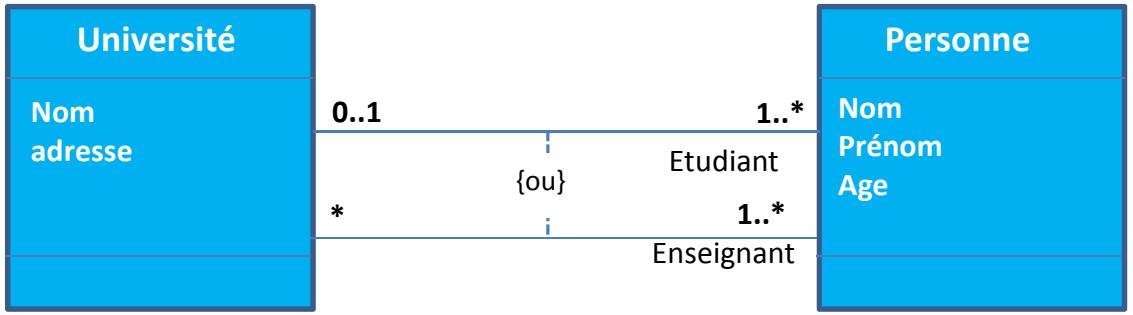


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Contrainte sur les associations :

✓ Il est possible d'exprimer des contraintes sur une association, afin de limiter les objets mis en jeu. Cela permet de mieux cadrer l'architecture de l'ensemble.




Ici, on indique qu'une personne joue soit le rôle d'étudiant, soit le rôle d'enseignant pour une université donnée (ou exclusif)

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

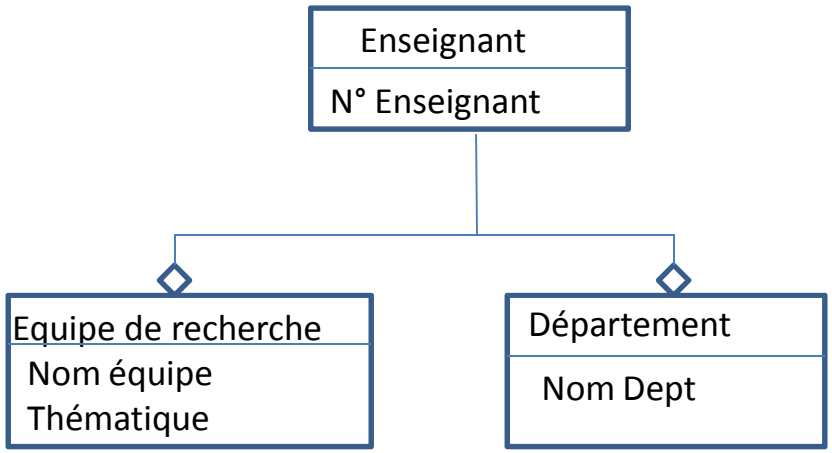
## ❑ Association : agrégation

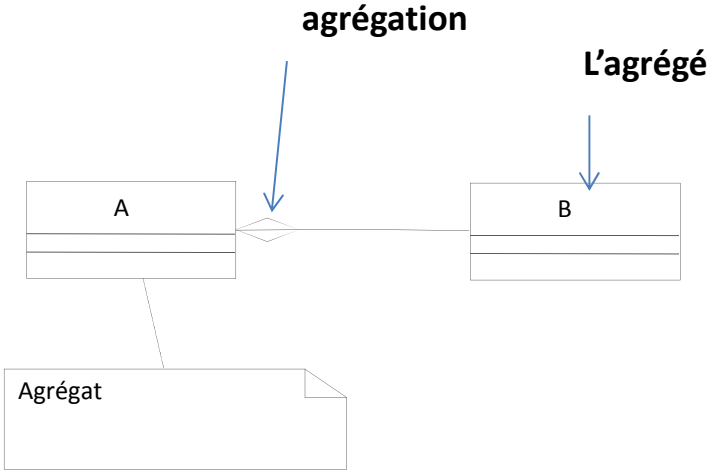
**Association particulière dans laquelle l'une des classes décrit un tout alors que la classe associée décrit des parties. La classe qui représente le tout est appelée composite, la classe qui représente une partie du tout est appelée composant.**

## ❑ Propriétés de l'agrégation

- A « contient » des instances de B
- La suppression de A n'implique pas la suppression de B
- L'élément agrégé peut être partagé

## ❑ Exemple:




**•L'enseignant est un composant d'une (ou plusieurs) équipe de recherche**

**• La disparition d'une équipe de recherche n'entraine pas la disparition d'un enseignant**

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Association : agrégation

❑ Propriétés

**Transitivité**

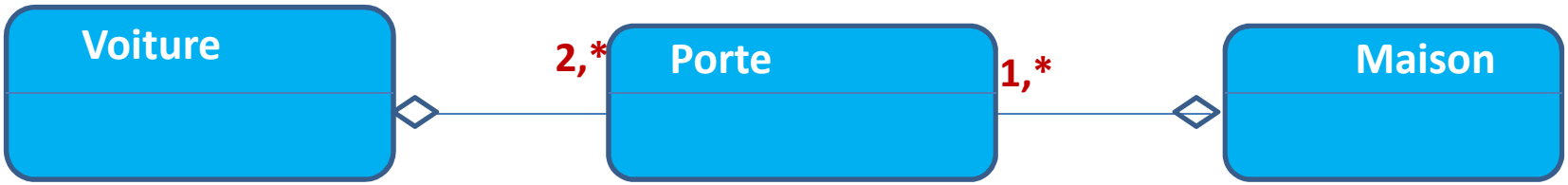
**Si A est composé de B, si B est composé de C, alors A est composé de C**

**Asymétrie**

**Si A est composé de B alors B n'est pas composé de A**

**Propagation des valeurs**

**Les propriétés sont automatiquement propagées du « tout » à « la partie »**
**La voiture est bleue, donc sa porte est bleue.**



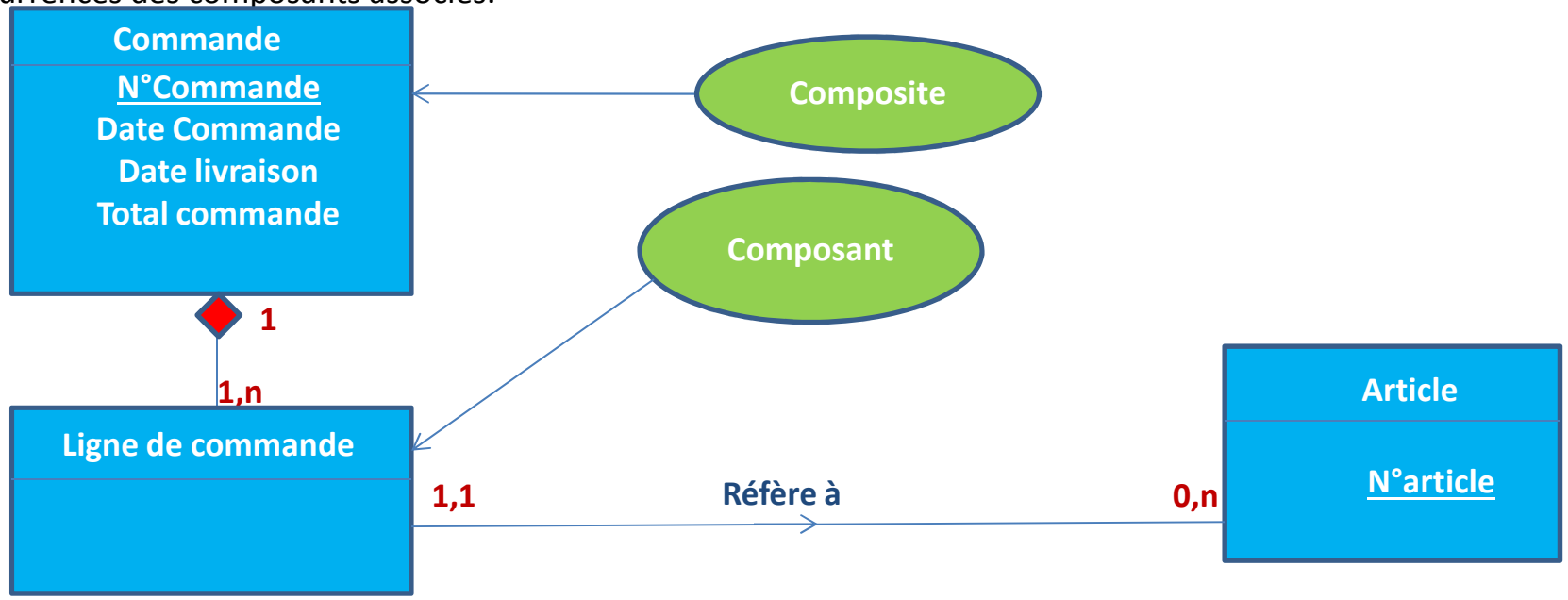
# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## ❑ Association : composition

**Une forme forte d'agrégation avec le composé qui à chaque moment a une possession exclusive des parties. Le temps de vie des parties coïncide avec celui du composé**

- Remarque 1:Dans une composition, la multiplicité du côté du composite est toujours à 1, car un composant doit appartenir à un seul et un seul composite
- Remarque 2: la création d'une occurrence d'un composant exige la présence d'un composite pour s'y associer.
- Remarque 3 : la fin de vie d'une occurrence de composite entraîne en cascade la fin de vie de toutes les occurrences des composants associés.



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## ❑ Association: Héritage

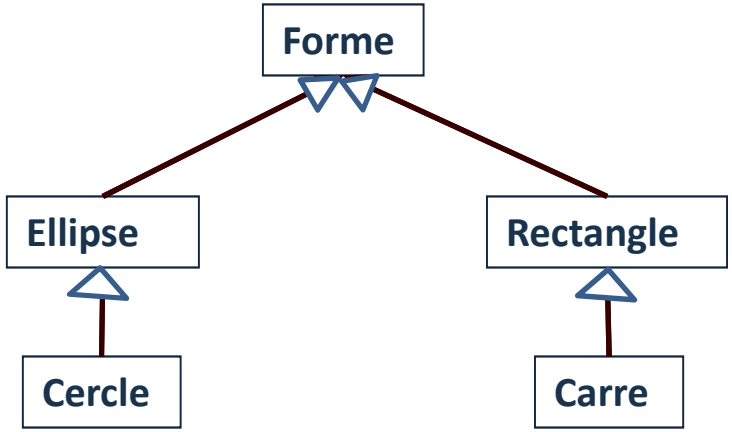
– L'héritage est un mécanisme de transmission des propriétés d'une classe vers une sous-classe. C'est une ***association entre classes***.

– Une sous-classe possède toutes les propriétés de sa super-classe mais elle peut en redéfinir certaines et en posséder d'autres.

– La sous-classe est une spécialisation (ou raffinement) de la super-classe.

## ➢ Exemple

**Un cercle est une spécialisation d'une ellipse, il en possède les propriétés de l'ellipse plus d'autres qui lui sont spécifiques. On dérive donc la classe Cercle de la classe Ellipse.**



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

## ❑ Association: Héritage

➢ L ’héritage a deux objectifs :

– **l'enrichissement** : on adjoint, dans la sous-classe, de nouvelles propriétés à celles issues de la super-classe

– **la substitution** : on redéfinit dans la sous-classe certaines propriétés issues de la super-classe.

– L'héritage est dit **simple** si la sous-classe n'a qu'une super-classe ; il est dit **multiple** si la sous-classe a plusieurs super-classes.

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

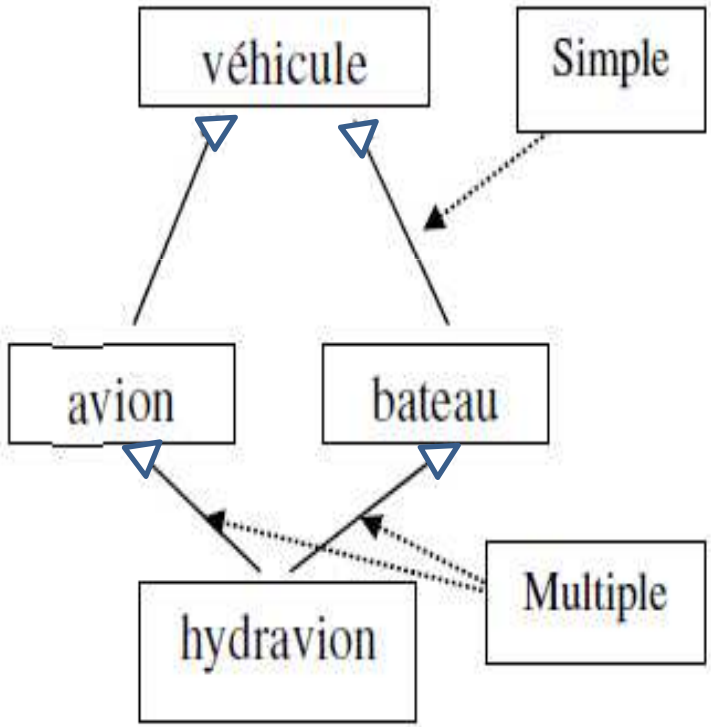
## ❑ Association: Héritage

➢ Classes de base :

- La classe « véhicule » est la classe **supérieure** par rapport à «avion » et « bateau ».
- Les classes «avion» et «bateau» sont les classes supérieures pour «**hydravion** ».

➢ Classes dérivées :

- Les classes « avion » et « bateau » sont des classes **dérivées**, (**sous-classes**) de la classe « véhicule ».
- La classe « hydravion » est une **classe dérivée** de la classe « avion » et « bateau ».
- La classe « hydravion » est considérée aussi comme une **classe dérivée** de la classe « véhicule ».



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

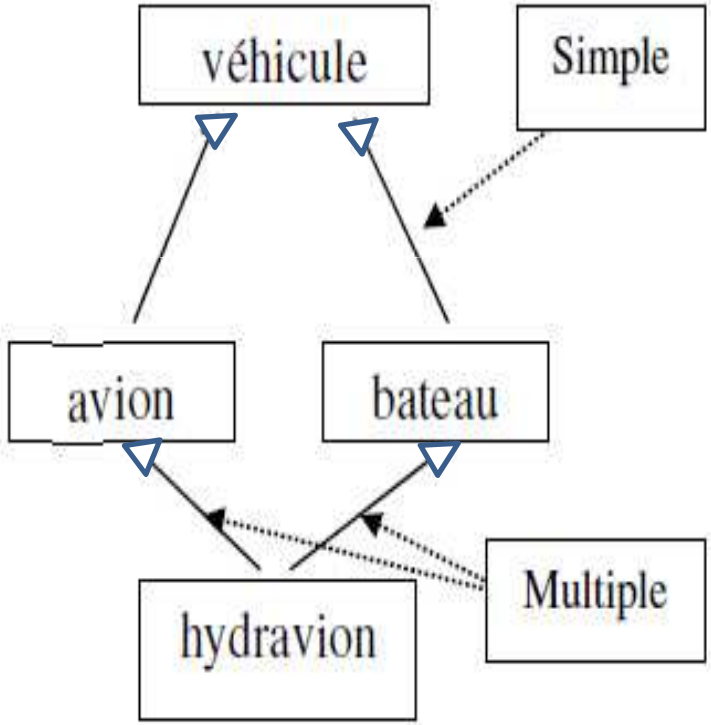
## ❑ Association: Héritage

➢ Héritage simple :

- La classe dérivée n'a qu'une seule classe de base.
- Les classes « avion » et « bateau » n'ont qu'une seule classe de base : la classe « véhicule ».

➢ Héritage multiple :

- Une classe dérivée a plus d'une classe de base.
- La classe « hydravion » hérite de « avion » et « bateau ».

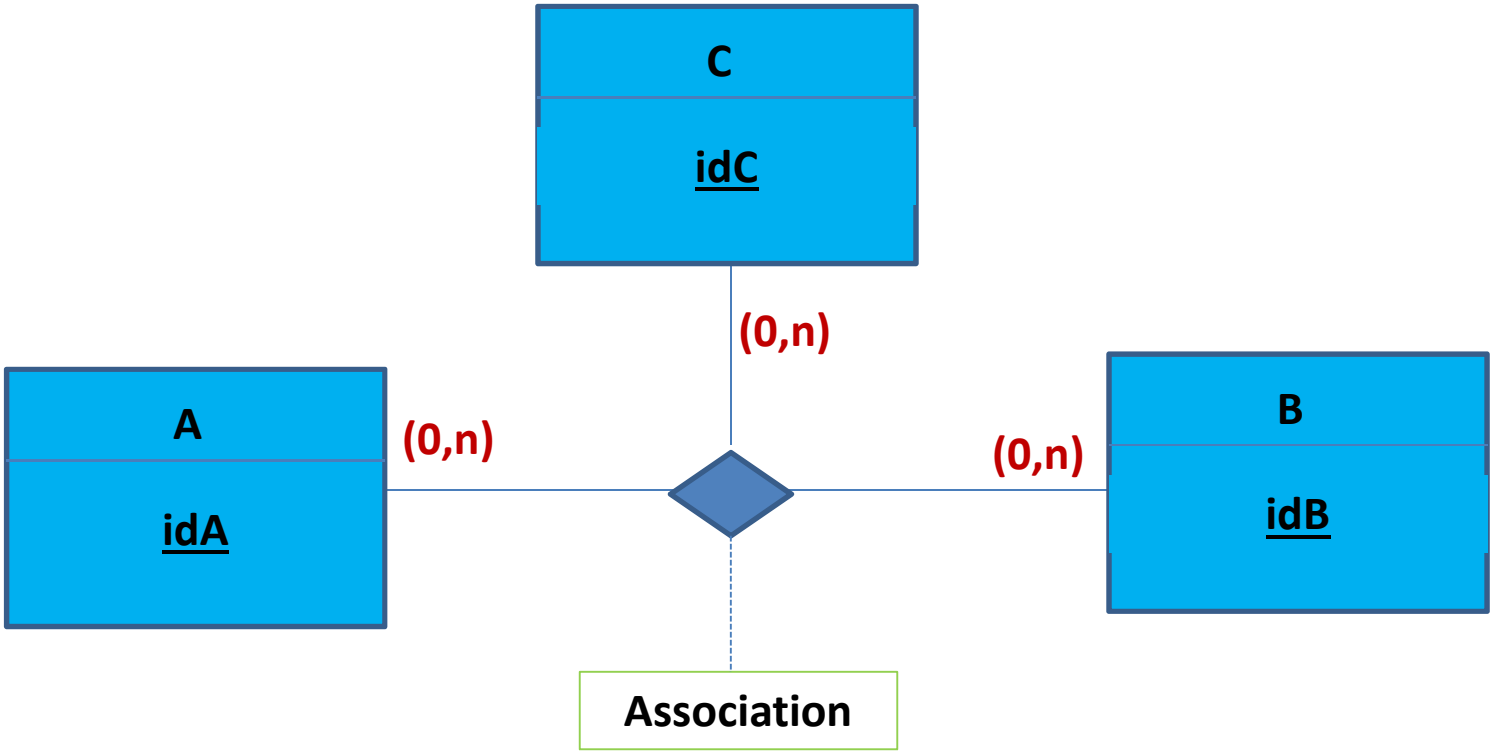


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Association ternaire

**Une association ternaire est une association qui décrit un lien sémantique entre trois classes**




Représentation UML : des associations (n-aire) : losange.

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

- Association ternaire
- Les associations ternaires: décomposition

- On remplace l'association ternaire (ou n-aire) par une classe et on lui attribut un identifiant

- On crée des associations binaires entre la nouvelle classe et toutes les autres classes de la collection de l'ancienne association

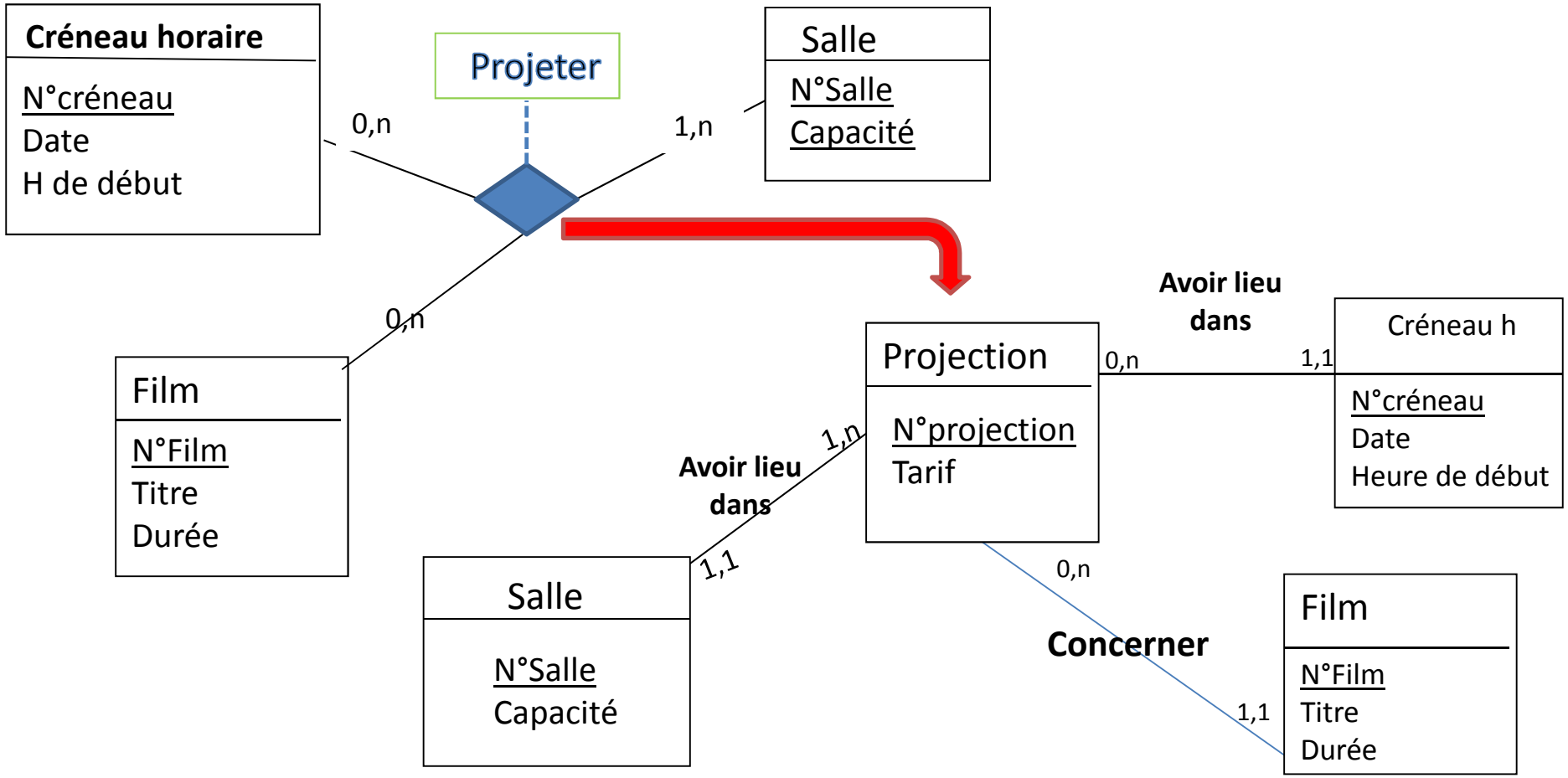
- La cardinalité de chacune des associations binaires crées est 0,n ou 1,n du côté des classes créé et 1,1 du côté des classes de la collection de l'ancienne association

# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Association ternaire

❑ Les associations ternaires: décomposition

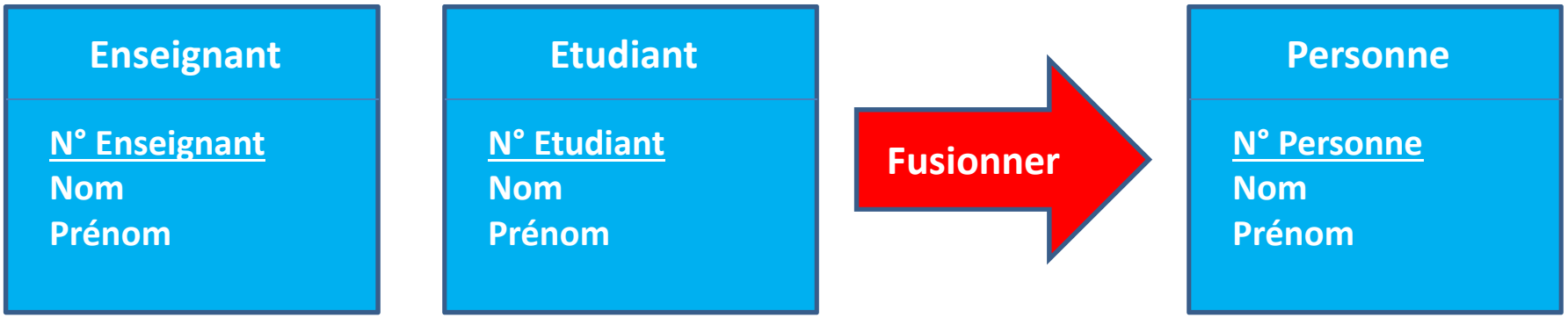


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles portant sur les noms

**Le nom d'une classe, d'une classe-association, ou d'un attribut doit être unique**

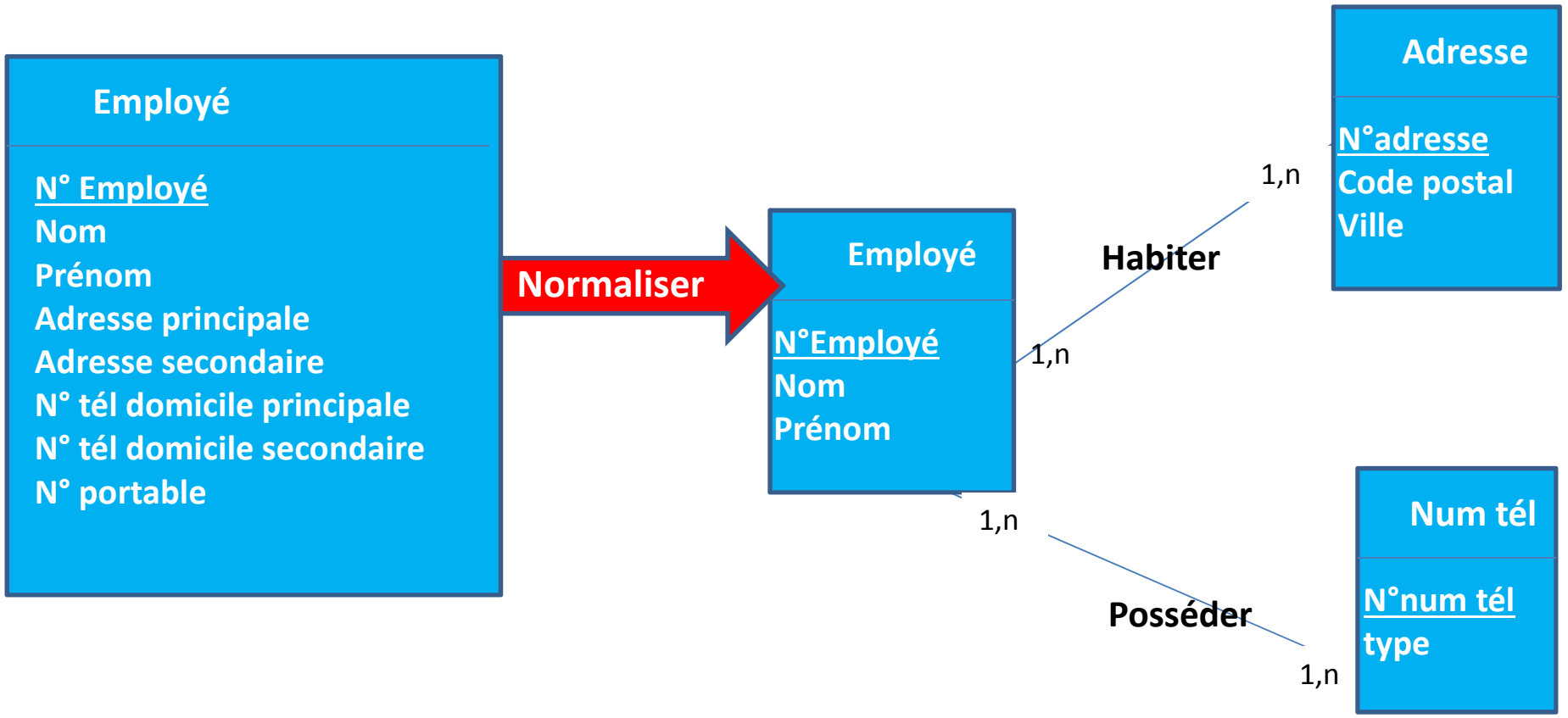



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles de normalisation des attributs

**un attribut multiple doit être remplacé par une classe-association et une classe supplémentaires**

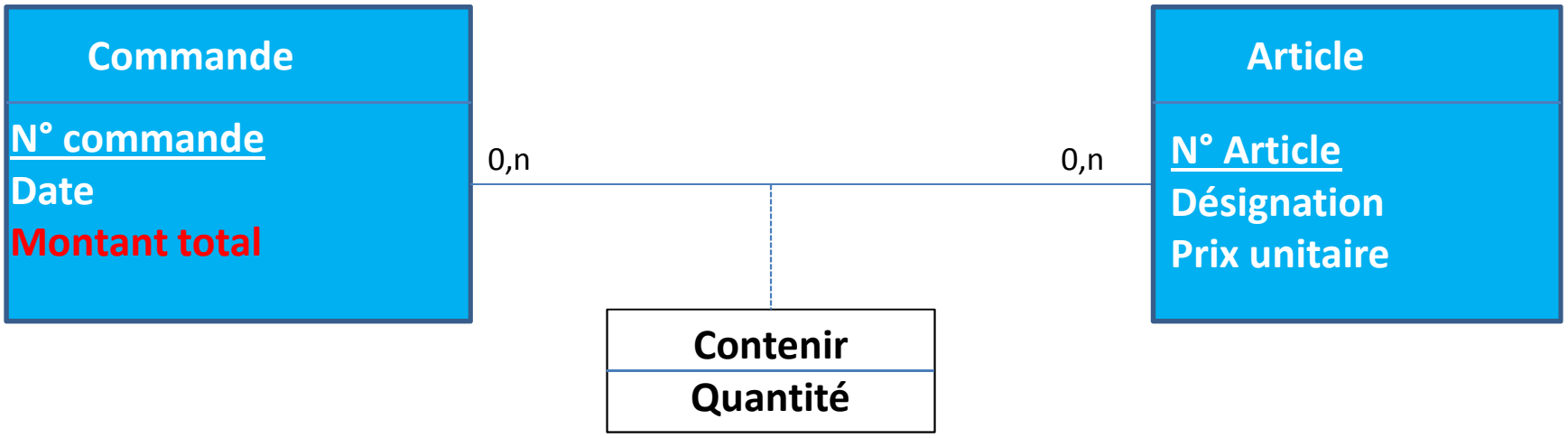


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles de normalisation des attributs

**Un attribut est une donnée élémentaire, ce qui exclut des données calculées ou dérivées**

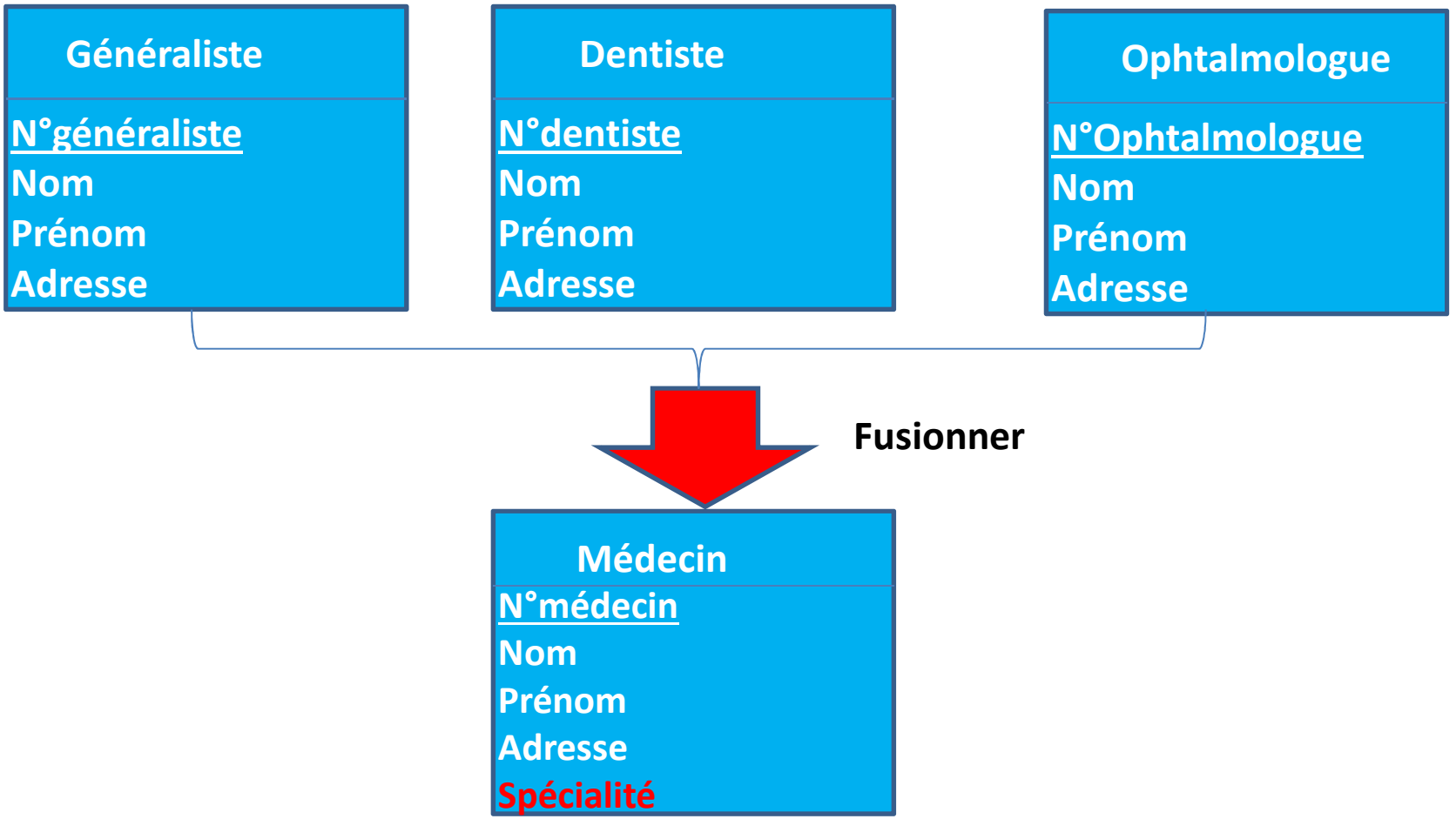


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles de fusion/ suppression classes/classses-associations

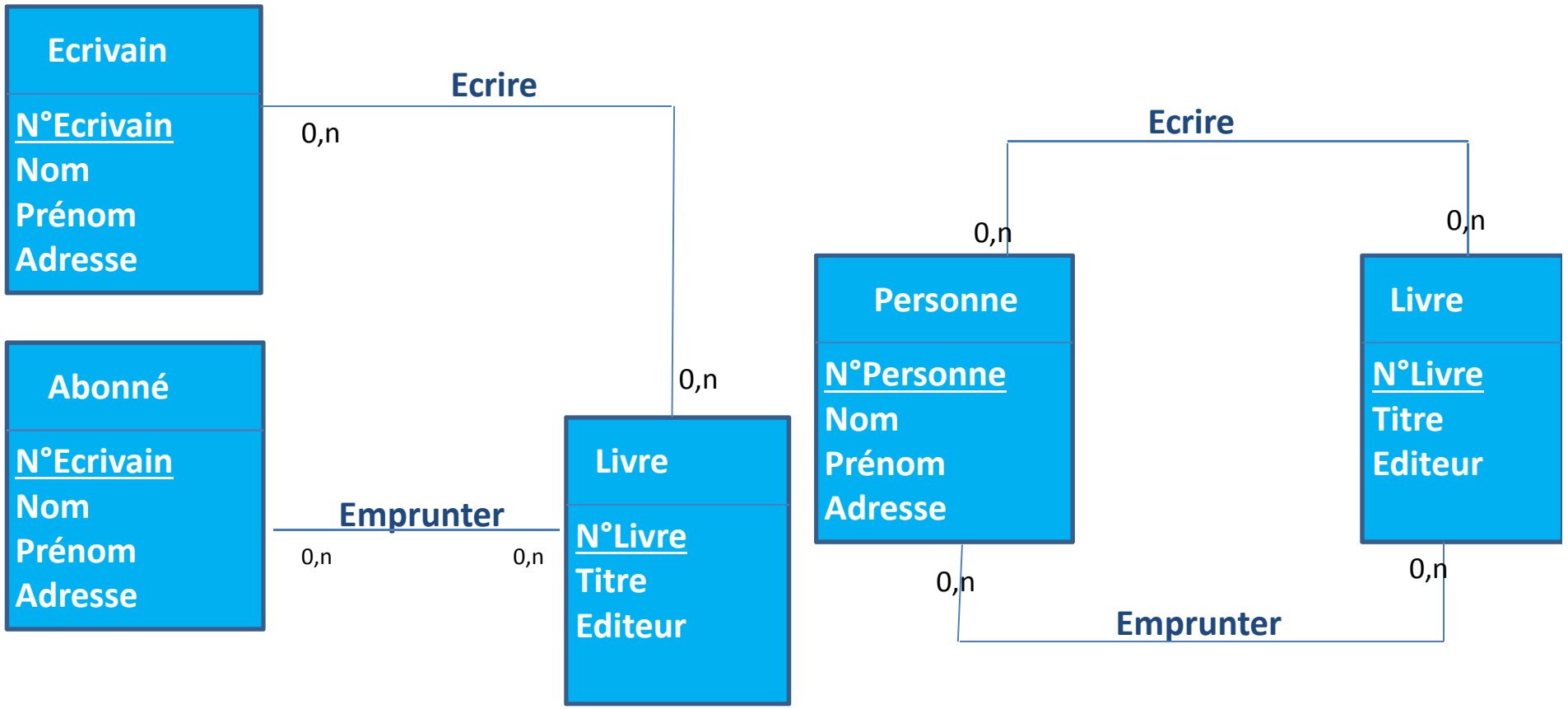
**Il faut factoriser les classes quand c'est possible**



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles de fusion/ suppression classes/classes-associations

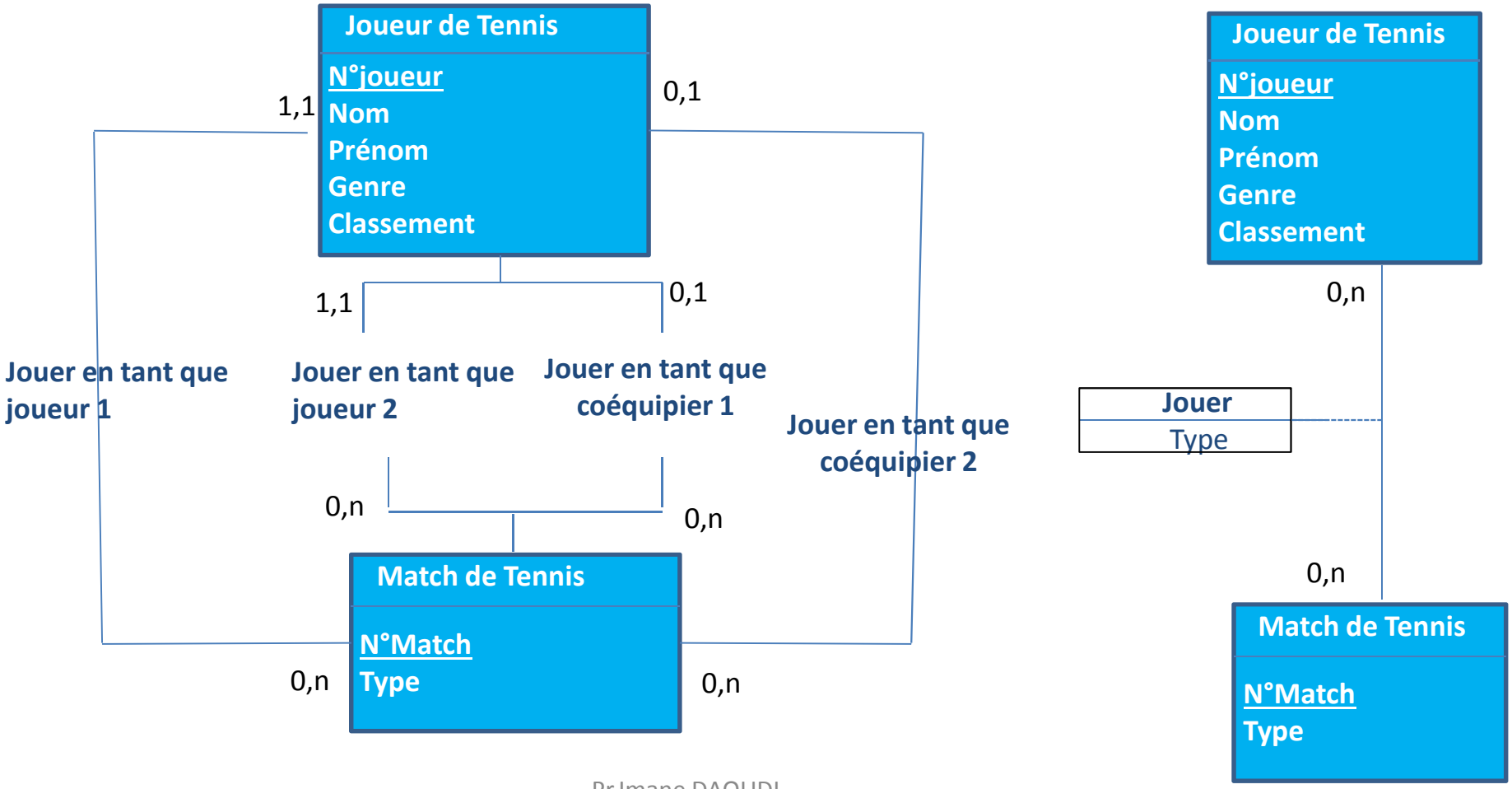


# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles de fusion/ suppression classes/classes-associations

**Il faut factoriser les classes-associations quand c'est possible**



# Programmation Orientée Objet

- L'analyse et conception orientée objet avec UML

❑ Règles de fusion/ suppression classes/classes-associations

**Il faut aussi se poser la question de l'intérêt de l'association quand les cardinalités maximale sont toutes de 1**