# Plan

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



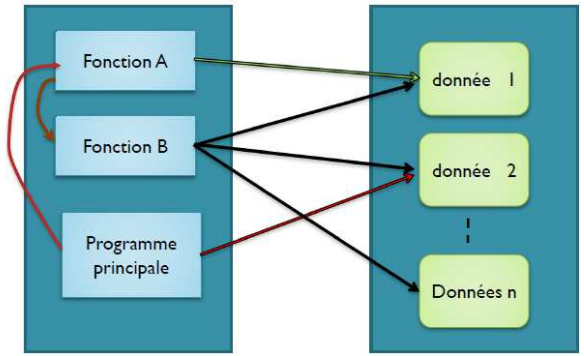
# Introduction

- 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

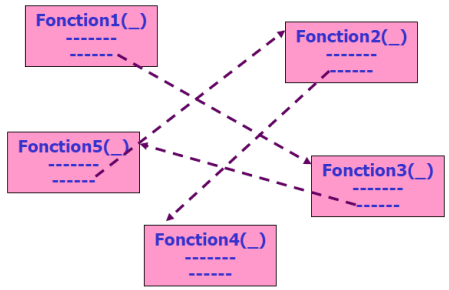
- Limitation de la programmation procédurale(1)
  - La programmation procédurale réunit au sein des modules des structures de données et des procédures applicables sur ces dernières.

    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, car on peut accéder à la représentation interne des données
  - 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)
  - Propagation des erreurs

# 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?**

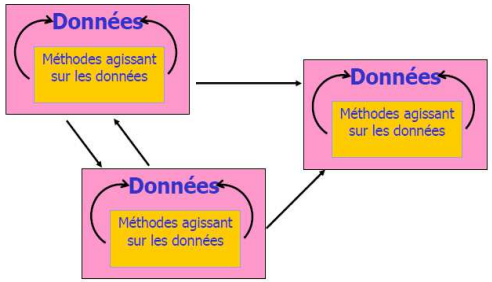


# Programmation Orientée Objet

- 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)
  - 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
- Exemple1: 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
- Exemple2: 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

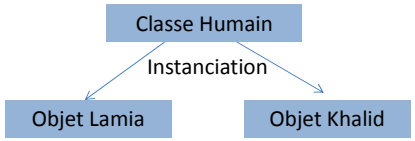


# Programmation Orientée Objet

- Notion de classe

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

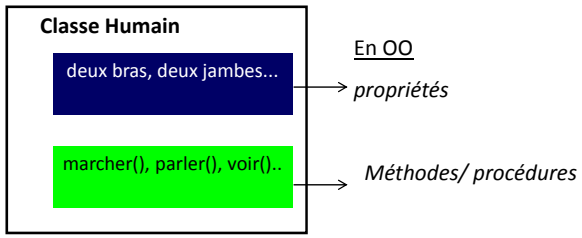
  - Remarques:
    - Les objets ayant des mêmes propriétés et les mêmes méthodes peuvent être mis dans une classe
    - Les objets apparaissent alors comme des variables d'un tel type classe. On dit aussi qu'un objet est une instance de sa classe.
  - Exemple






# 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

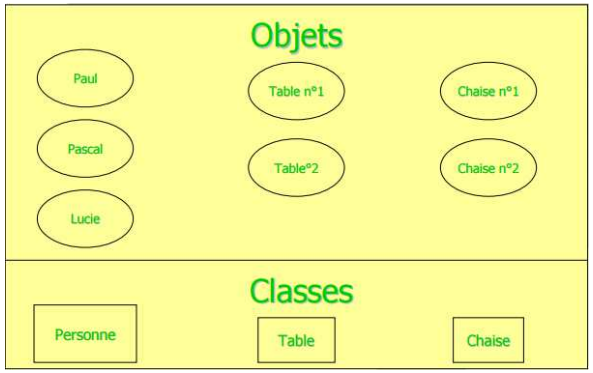
| propriétés | méthodes |
|---|---|
| état (emprunté / disponible / perdu)<br>date de la fin de l'emprunt<br>titre<br>auteur<br>nombre de pages | emprunter<br>rendre<br>perdre<br>voler |

- 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 la verre |

# Programmation Orientée Objet

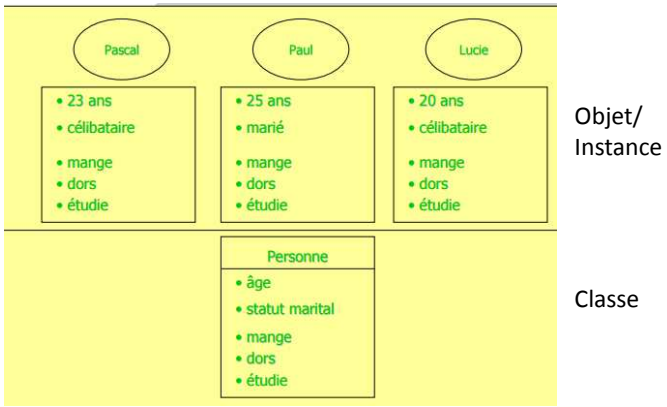
- Classe/Objet






# Programmation Orientée Objet

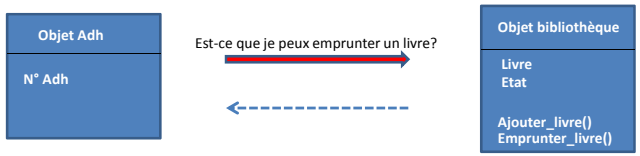
- Classe/instance






# Programmation Orientée Objet

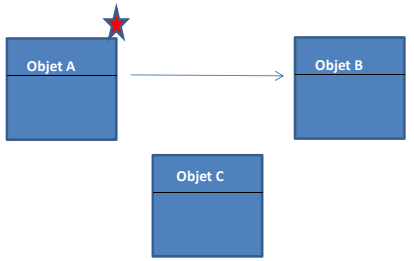
- Méthodes et envoi de message
  - Les objets communiquent entre eux par échange de messages
  - Le message le plus échangé est la demande de réalisation de traitement
  - Exemple





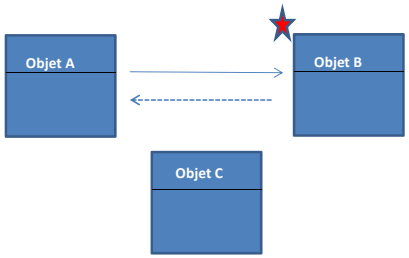

# Programmation Orientée Objet

- 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

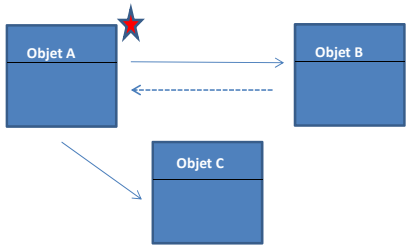
- Méthodes et envoi de message
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose






# Programmation Orientée Objet

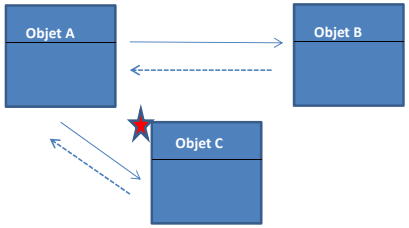
- Méthodes et envoi de message
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose






# Programmation Orientée Objet

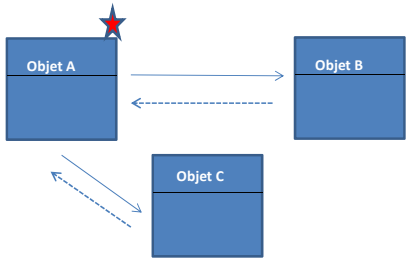
- Méthodes et envoi de message
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose






# Programmation Orientée Objet

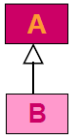
- Méthodes et envoi de message
  - Les communications entre objets sont principalement synchrones
  - L'objet appelant attend la réponse de l'appelé avant de pouvoir faire autre chose



# 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



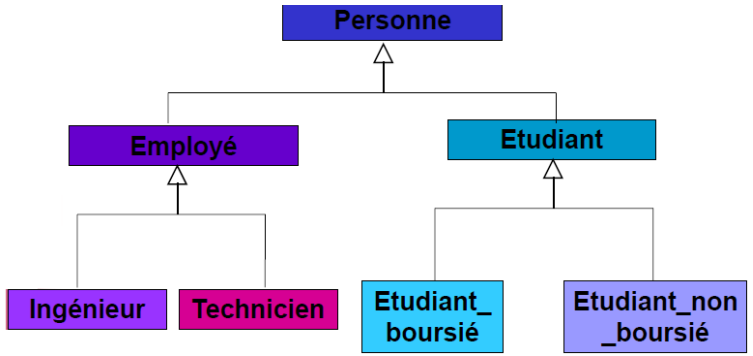



# Programmation Orientée Objet

➢ Héritage simple

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

➢ Exemple



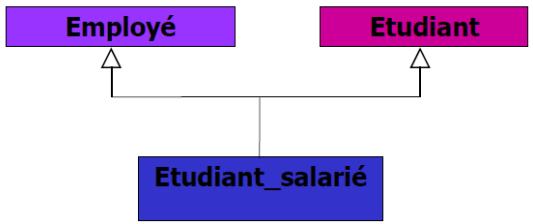



# Programmation Orientée Objet

➢ 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

➢ Agrégation (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

# Plan

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



# Programmation Orientée Objet

- La conception OO 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 d'objets
- **Diagrammes de classes**
- Diagrammes de paquetage
- Diagrammes de composants
- Diagrammes de déploiement



# Programmation Orientée Objet

- La conception OO 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, ...).



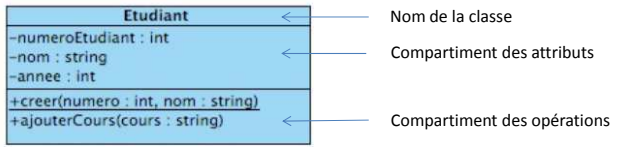
# Programmation Orientée Objet

- La conception OO avec UML

**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

- La conception OO avec UML

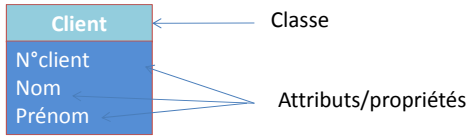
**Attribut (classe)**

**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 clase.

❑ Exemple




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



# Programmation Orientée Objet

- Conception: Modèle conceptuel de données

**Règles concernant les attributs**

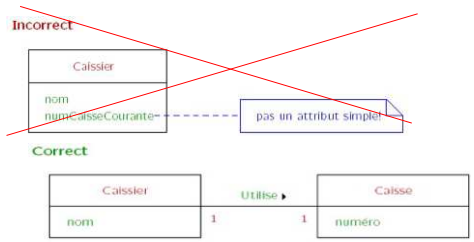
**Règle 1**

**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).**

- Exemple






# Programmation Orientée Objet

- Conception: Modèle conceptuel de données

**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). un attribut est une donnée élémentaire, ce qui exclut des données composées.**

**Règle 3**

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



# Programmation Orientée Objet

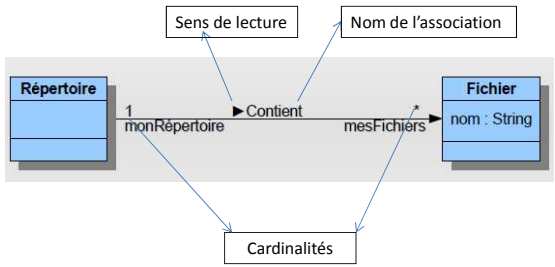
- La conception OO avec UML

**Association**

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

❑ Elle est caractérisée par :

- Nom de l'association
- Sens de lecture
- Cardinalités

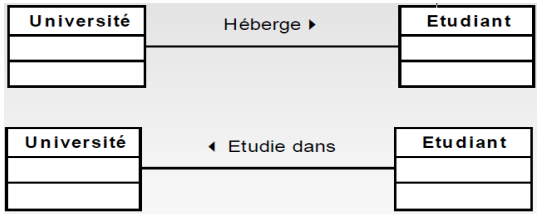


# Programmation Orientée Objet

- La conception OO avec UML

❑ Nommage des associations :

- Indication du sens de lecture




- Par défaut, les associations sont bidirectionnelles. A partir d'un objet d'une des 2 classes, on peut atteindre les objets de l'autre classe
- On peut restreindre la navigabilité en ajoutant une flèche à une extrémité : Navigation uni-directionnelle



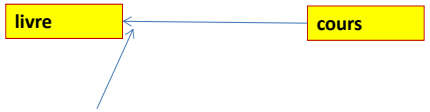
# Programmation Orientée Objet

- La conception OO avec UML

❑ Nommage 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:
  - ✓ Un cours connaît les livres qui lui sont nécessaires mais pas le contraire.
  - ✓ Si la flèche est absente, la navigation est bidirectionnelle




Association directionnelle

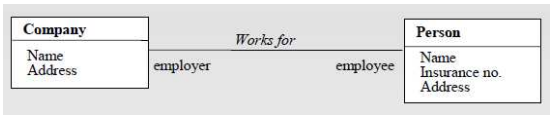


# Programmation Orientée Objet

- La conception OO avec UML

❑ Nommage des rôles :

- Le rôle décrit une extrémité d'une association



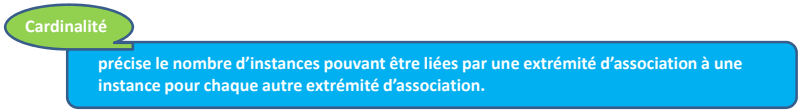

- Une personne peut est considérée comme employé du point de vue de l'entreprise
- L'entreprise est considérée comme employeur du point de vu de la personne
- Rôle : identifie l'extrémité d'une association



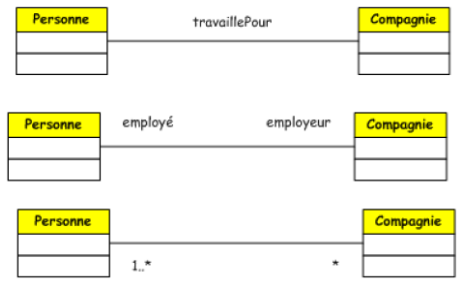
# Programmation Orientée Objet

- La conception OO avec UML

❑ Cardinalités :




| Cardinalités | Signification |
|---|---|
| 0,1 | Au plus un |
| 1,1 (ou 1) | Un seul |
| 0,n (ou *) | Un nombre indéterminé |
| 1,n | Au moins un |



# Programmation Orientée Objet

- La conception OO 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

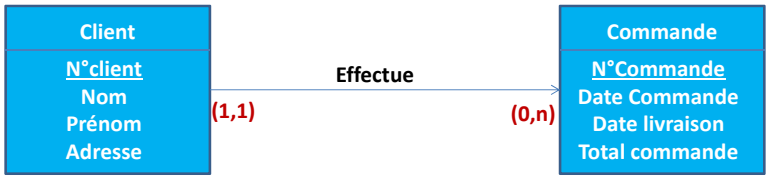
- Une cardinalité maximale de 0 n'a pas de sens
- Si une cardinalité maximale est connu et vaut 2, 3 ou plus, alors nous considérons qu'elle est indéterminée et vaut n
- Les cardinalités minimales qui valent plus de 1 sont modélisées par 1



# Programmation Orientée Objet

- La conception OO 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 !

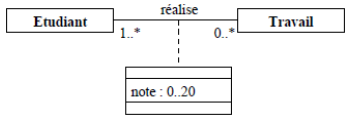


# Programmation Orientée Objet

- La conception OO avec UML

❑ Classe-association :

- Si une association possède des propriétés ou des opérations, il est possible de la qualifier à l'aide d'une **classe-association**.
- Une classe-association possède les mêmes caractéristiques que les associations et les classes.




- Une classe-association qui ne participe pas à d'autres relations avec d'autres classes peut ne pas porter de nom.



# Programmation Orientée Objet

- La conception OO avec UML

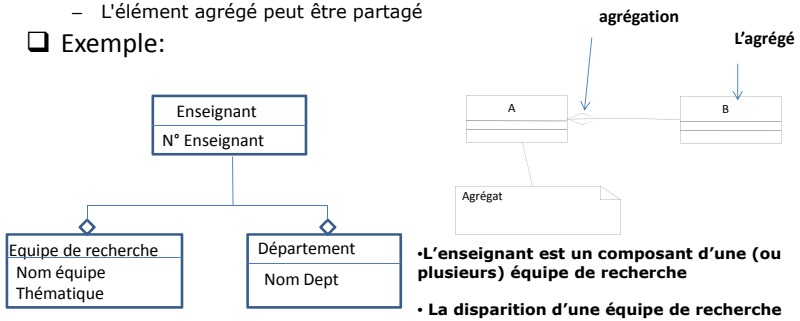
❑ Association : agrégation

**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

- La conception OO avec UML

❑ Association : agrégation

❑ Propriétés

**Transitivité**

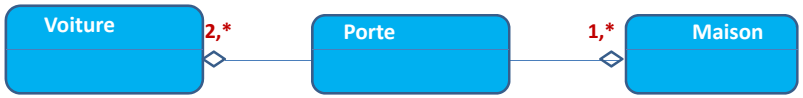
**Si A est composé de B, si B est composé de C, alors A est composé de C**

**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.**





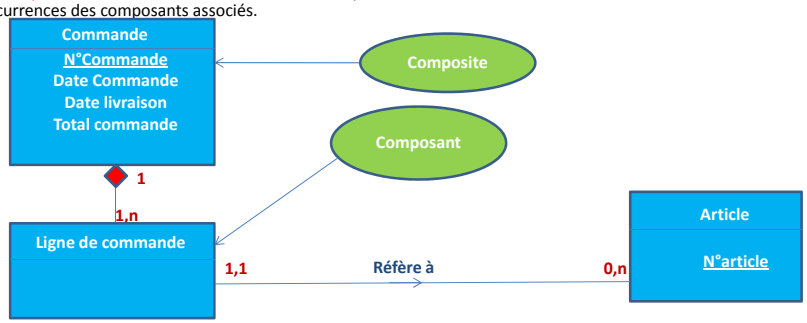

# Programmation Orientée Objet

- La conception OO 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

- La conception OO avec UML

❑ Association: Héritage

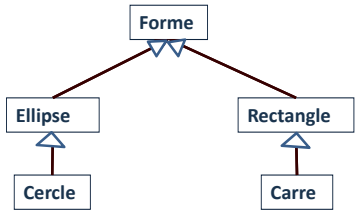
– L'héritage est un mécanisme de transmission des propriétés d'une classe vers une sous-classe. C'est une ***association entre classes***.

– 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

- La conception OO 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

- La conception OO avec UML

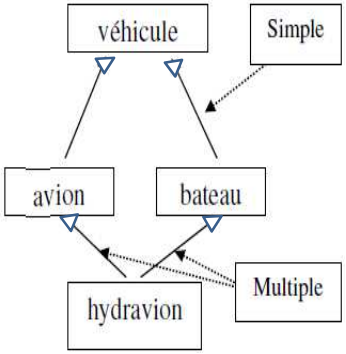
❑ Association: Héritage

➢ 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

- La conception OO avec UML

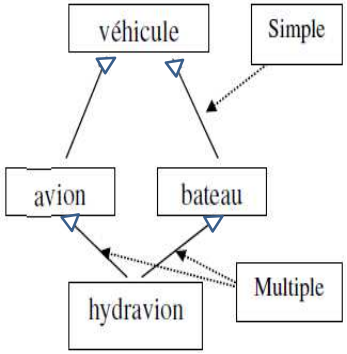
❑ Association: Héritage

➢ 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 ».



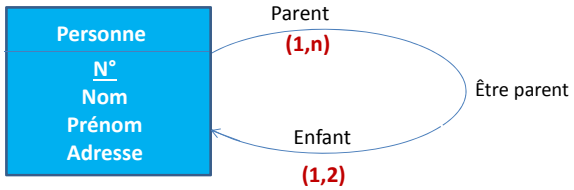



# Programmation Orientée Objet

- La conception OO avec UML

❑ Association réflexive

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



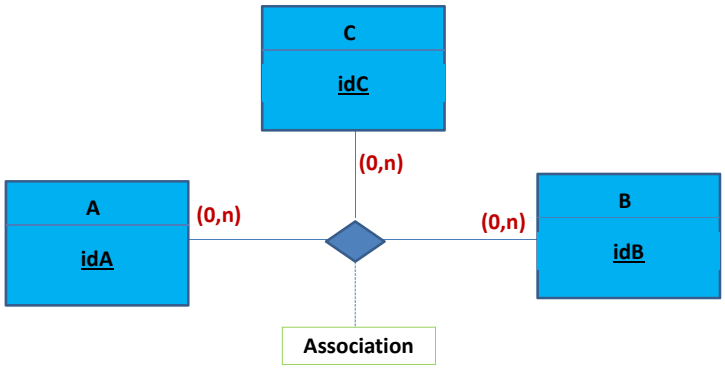



# Programmation Orientée Objet

- La conception OO 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

- La conception OO 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

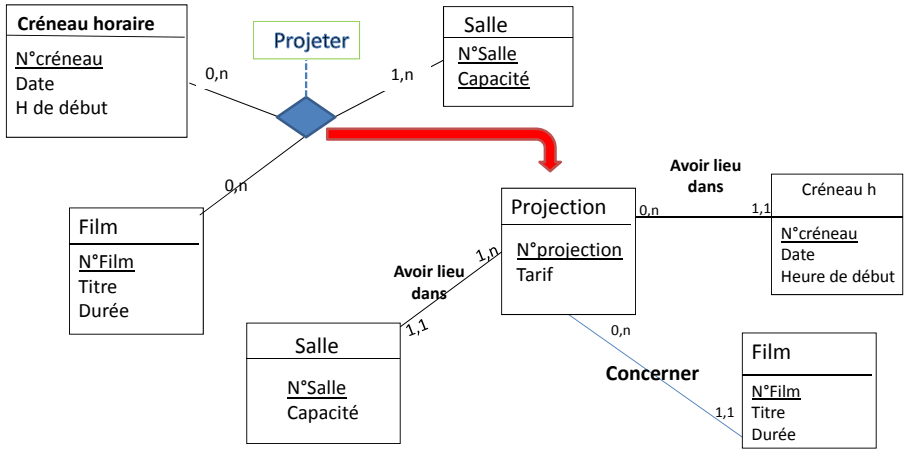


# Programmation Orientée Objet

- La conception OO avec UML

❑ Association ternaire

❑ Les associations ternaires: décomposition






# Programmation Orientée Objet

- La conception OO avec UML

❑ A retenir

**Classes**

**Règle 9** Pour chaque occurrence d'une classe, chaque attribut ne peut prendre qu'une valeur

**Règle 10** Un attribut ne peut en aucun cas être partagé par plusieurs classes

**Règle 11** Un attribut est une donnée élémentaire, ce qui exclut des données calculées ou dérivées

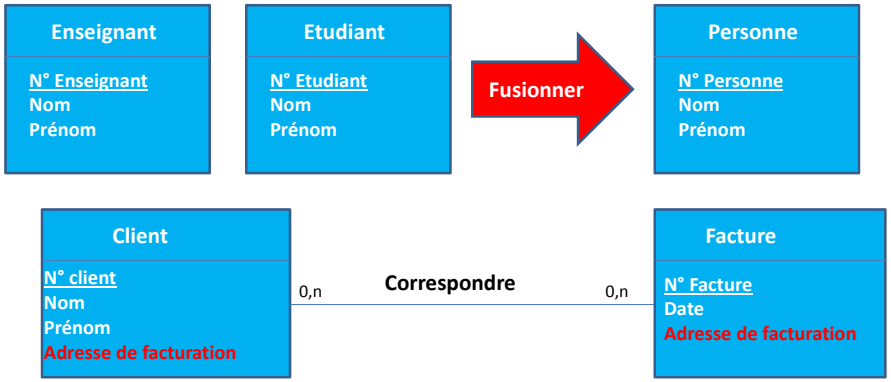


# Programmation Orientée Objet

- La conception OO avec UML

❑ Règles portant sur les noms

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

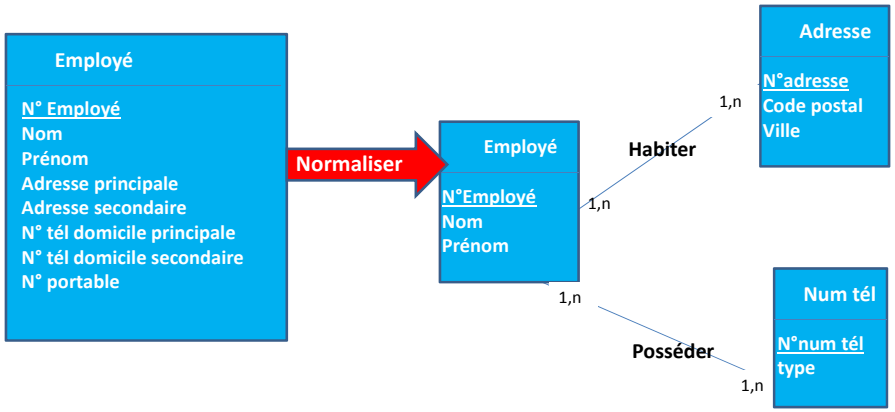


# Programmation Orientée Objet

- La conception OO avec UML

❑ Règles de normalisation des attributs

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



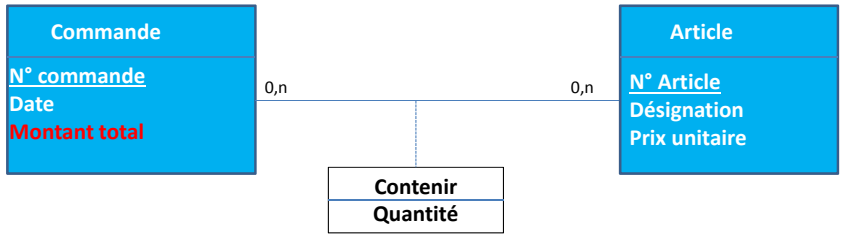



# Programmation Orientée Objet

- La conception OO avec UML

❑ Règles de normalisation des attributs

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



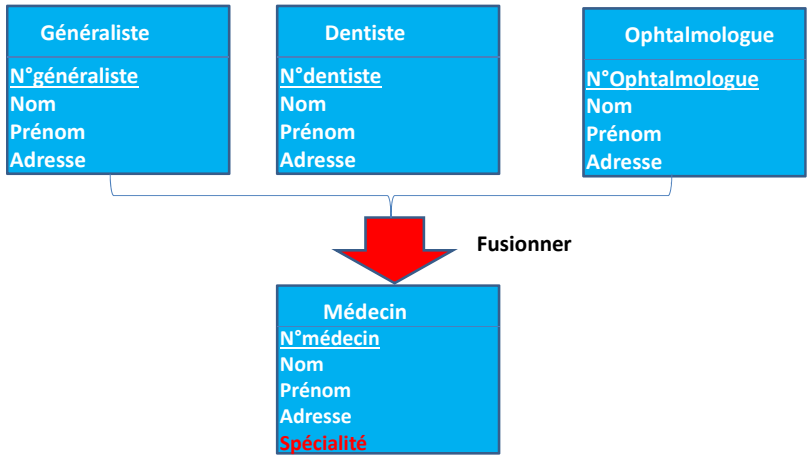



# Programmation Orientée Objet

- La conception OO avec UML

❑ Règles de fusion/ suppression classes/classses-associations

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

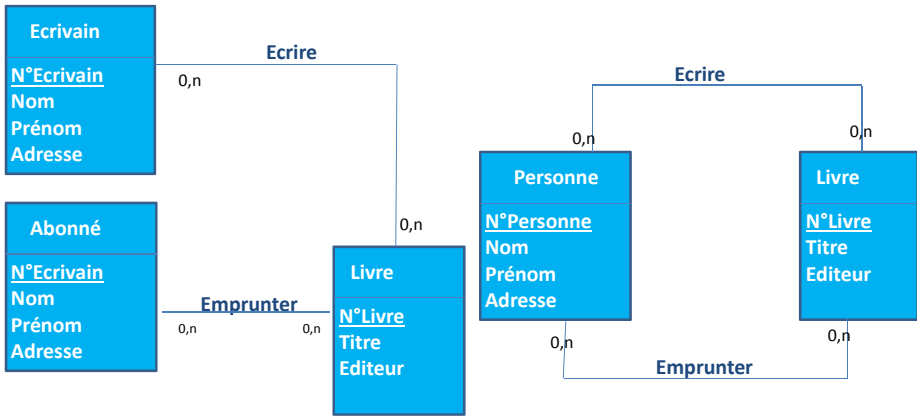





# Programmation Orientée Objet

- La conception OO avec UML

❑ Règles de fusion/ suppression classes/classes-associations

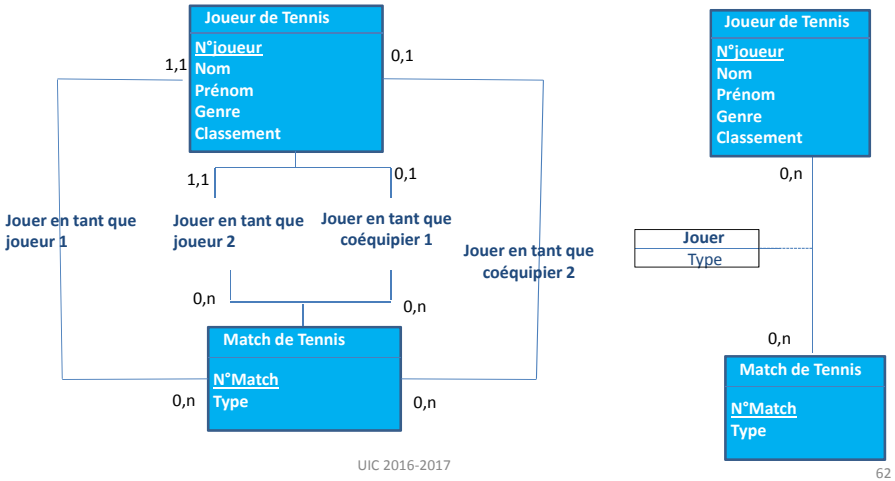


# Programmation Orientée Objet

- La conception OO avec UML

❑ Règles de fusion/ suppression classes/classes-associations

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






# Programmation Orientée Objet

- La conception OO 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**