



# DIAGRAMME DE CLASSE

**Présenté par : Dr. Lamia BERKANI**

**Section :L3 ACAD « B »**

**Année: 2024-2025**

# CONCEPTION: APPROCHE FONCTIONNELLE VS OBJET

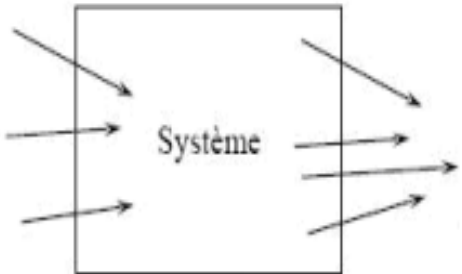
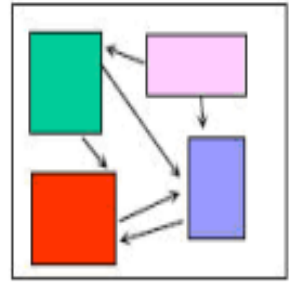
- **Modularité:** partitionner le logiciel en modules
- **Cohésion d'un module:** mesurée aux nombres de ses sous-modules qui effectuent des tâches similaires et ont des interactions continues
- **Interdépendance (couplig)** entre deux modules A et B est mesuré à la quantité de modifications à faire sur B lorsque A est modifié (et réciproquement)

# CONCEPTION: APPROCHE FONCTIONNELLE VS OBJET

- Pour qu'un logiciel soit extensible et réutilisable, il faut qu'il soit découpé en modules :
  - Faiblement couplés, et
  - À forte cohésion
- **Couplage**: une entité (fonction, module, classe, package, composant) est couplée à une autre si elle dépend d'elle (couplage faible / fort)
- **Forte cohésion**: l'idée est que nous rassemblons bien dans une classe des méthodes cohérentes, qui visent à réaliser des objectifs similaires,

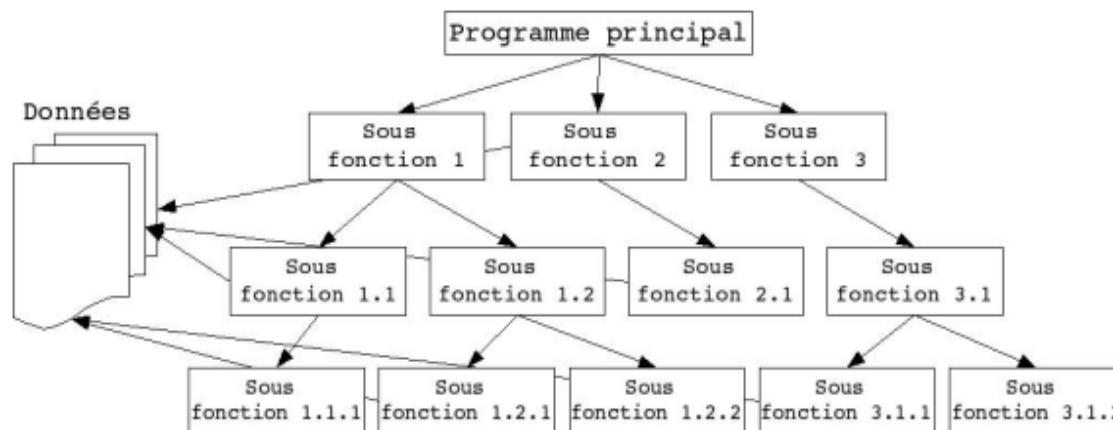
# CONCEPTION: APPROCHE FONCTIONNELLE VS OBJET

- Deux approches:
  - Fonctionnelles (descendante)
  - Objet (ascendante)

LA DEMARCHE FONCTIONNELLE	LA DEMARCHE OBJET
<p>Accent mis sur ce que fait le système (ses fonctions)</p>  <p>«Que fait le système ?»</p>	<p>Accent mis sur ce qu'est le système (ses composants) : les objets</p>  <p>«De quoi se compose le système ?»</p>

# MODÉLISATION PAR DÉCOMPOSITION FONCTIONNELLE DESCENDANTE

- La forme la plus immédiate pour décrire un travail à effectuer est de **lister les actions à réaliser**.
- On découpe une tâche complexe à effectuer en une **hiérarchie d'actions** à réaliser de plus en plus simples, petites et précises
- L'implémentation en code source d'une solution décrite en terme d'actions est un code organisé en fonctions (**programmation procédurale**) :



# MODÉLISATION PAR DÉCOMPOSITION FONCTIONNELLE DESCENDANTE

- **L'analyse** est une découpe fonctionnelle descendante des fonctionnalités à pourvoir.
- **La conception** est une découpe du logiciel en une hiérarchie descendante d'actions permettant de satisfaire les fonctionnalités à pourvoir.
- **L'implémentation** est une programmation procédurale.

# DÉMARCHE OBJET

- On ne raisonne plus en termes d'actions mais plutôt en concepts du monde physique.
  - Puisqu'ils appartiennent au monde physique, ces concepts peuvent être stables et **réutilisables**.

# OBJET

## IDENTITÉ + ETAT + COMPORTEMENT

### ○ Une identité

- deux objets différents ont des identités différentes
- on peut désigner l'objet (y faire référence)

### ○ Un état (attributs)

- ensemble de propriétés/caractéristiques définies par des valeurs
- permet de le personnaliser/distinguer des autres objets
- peut évoluer dans le temps

### ○ Un comportement (méthodes)

- ensemble des traitements que peut accomplir un objet (ou que l'on peut lui faire accomplir)



# CLASSES ET INSTANCES

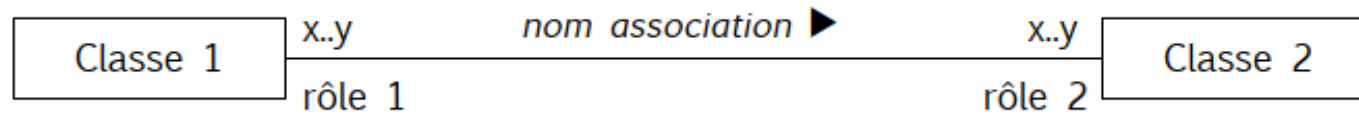
- Les objets possédant la même structure de données (attributs) et le même comportement (opérations) sont les représentants d'une même classe.
- Une classe est une abstraction qui décrit les propriétés pertinentes pour une application.
- Données et opérations traitant les données ne sont pas séparées, mais réunies au sein d'un même module → Cohésion,
- Chaque objet est une instance d'une classe.

# DIAGRAMME DE CLASSE

- Les diagrammes de cas d'utilisation modélisent à QUOI sert le système.
- Le système est composé d'objets qui interagissent entre eux et avec les acteurs pour réaliser ces cas d'utilisation.
- Les diagrammes de classes permettent de spécifier la *structure statique d'un système*, en termes de classes et de relations entre ces classes.

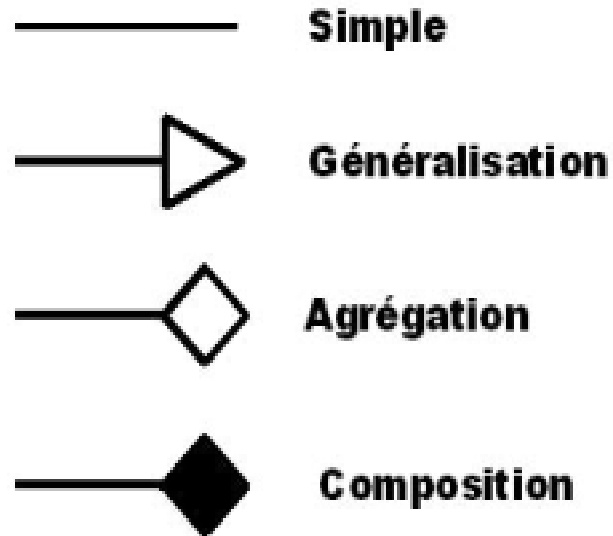
# LES ASSOCIATIONS

- Connexion sémantique bidirectionnelle entre classes
- Représentation des associations :
  - Nom : forme verbale, avec un sens de lecture
  - Rôles : forme nominale, décrit une extrémité de l'association
  - Multiplicité : 1, 0..1, 0..\*, 1..\*, n..m



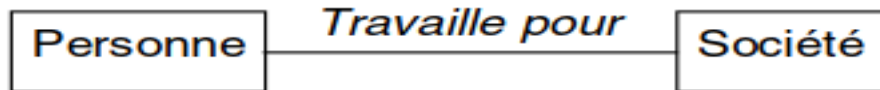
# LES ASSOCIATIONS

- Types d'associations:

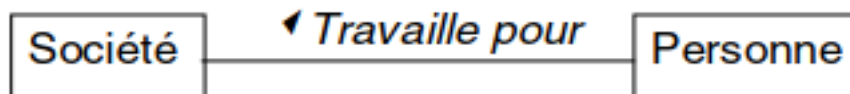


# LES ASSOCIATIONS

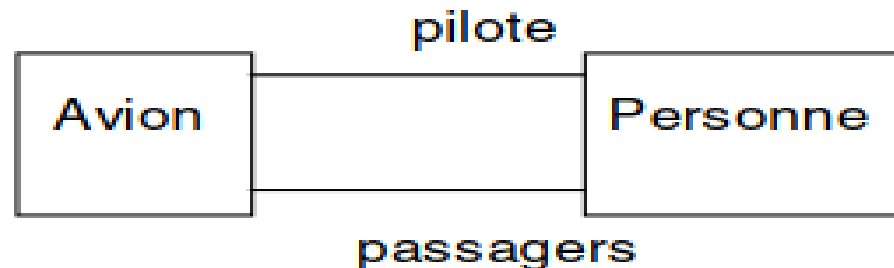
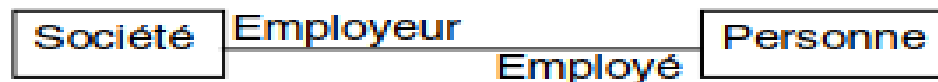
- Exemple d'une association



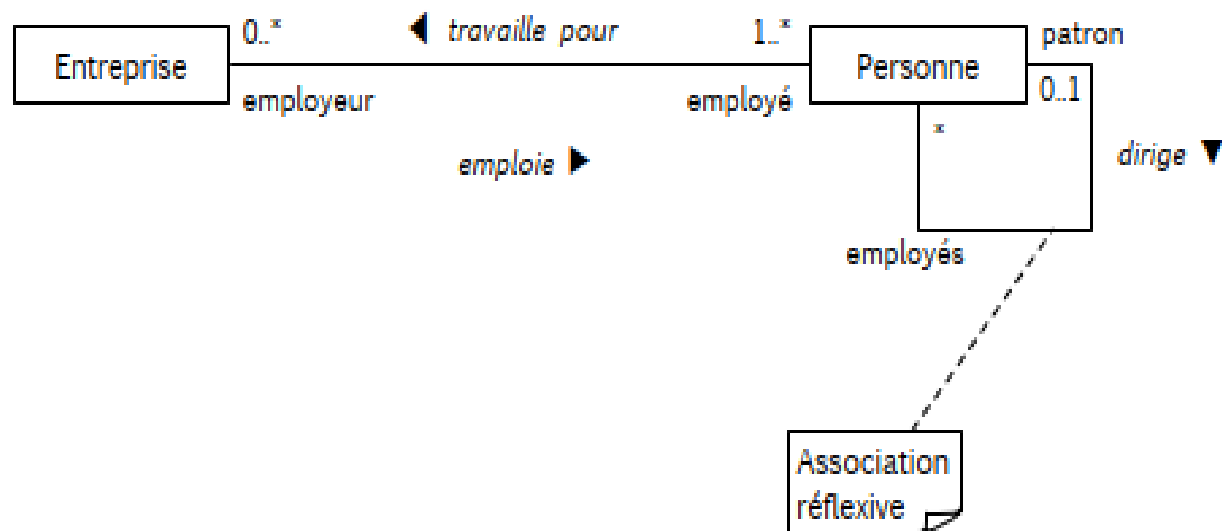
- Possibilité d'ajouter un sens de lecture



- Notion de Rôles:

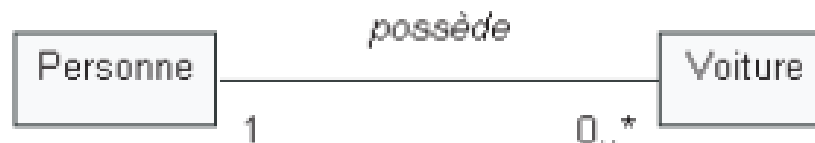


# EXEMPLE



# MULTIPLICITÉ DES ASSOCIATIONS

- La multiplicité spécifie le nombre d'instances d'une classe pouvant être liées à une seule instance d'une classe associée.
- Exemple : une personne peut posséder plusieurs voitures (entre zéro et un nombre quelconque); une voiture est possédée par une seule personne.

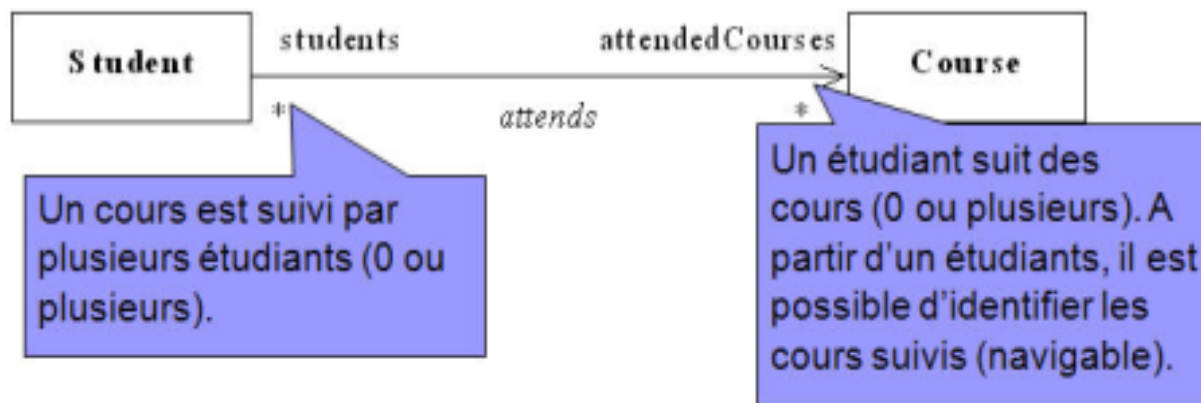
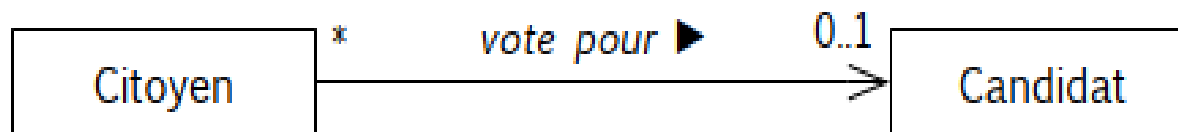


Cardinalités:

<b>1</b>	Un et un seul
<b>0..1</b>	Zéro ou un
<b>N</b>	N (entier naturel)
<b>M .. N</b>	De M à N (entiers naturels)
<b>*</b>	De zéro à plusieurs
<b>0 .. *</b>	De zéro à plusieurs
<b>1 .. *</b>	D'un à plusieurs

# NAVIGABILITÉ D'UNE ASSOCIATION

- Par défaut, les associations sont navigables dans les deux directions.
- la navigation peut être restreinte à une seule direction : les instances d'une classe ne "connaissent" pas les instances d'une autre.
- On restreint la navigabilité d'une association à un seul sens à l'aide d'une flèche.

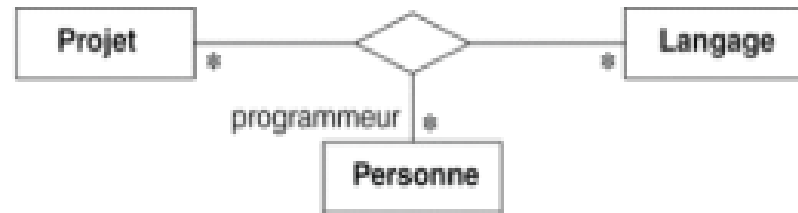




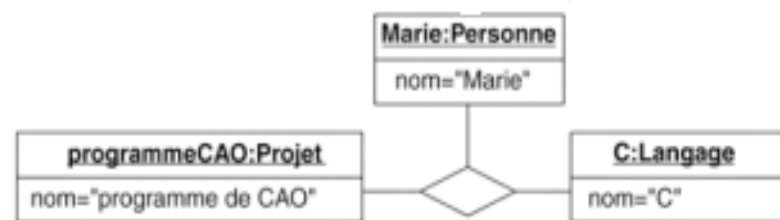
# ASSOCIATION N-AIRE

- En général, les associations sont binaires
- N-aires : au moins trois instances impliquées
- A n'utiliser que lorsqu'aucune autre solution n'est possible!

- Diagramme de classe:

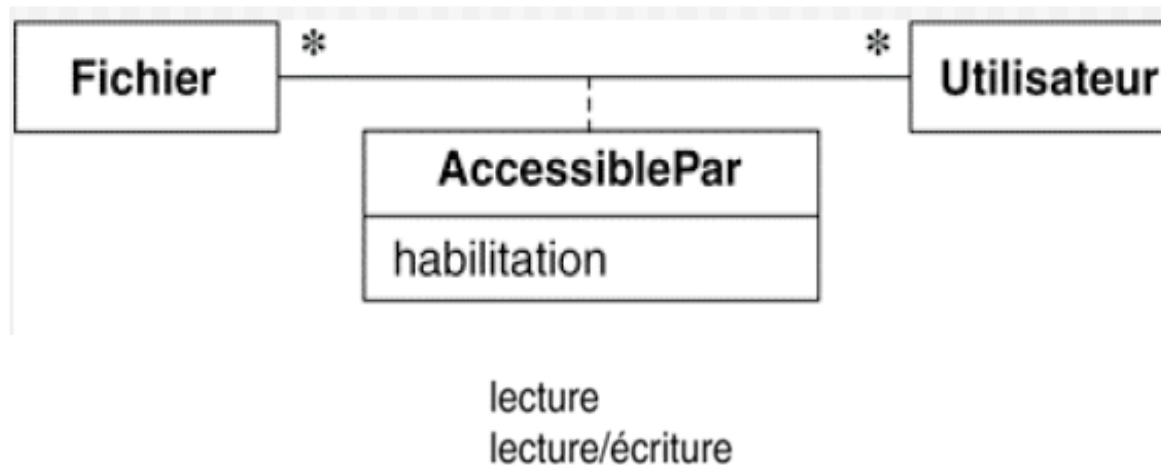


- Diagramme d'objet:  
(on souligne les noms des objets)



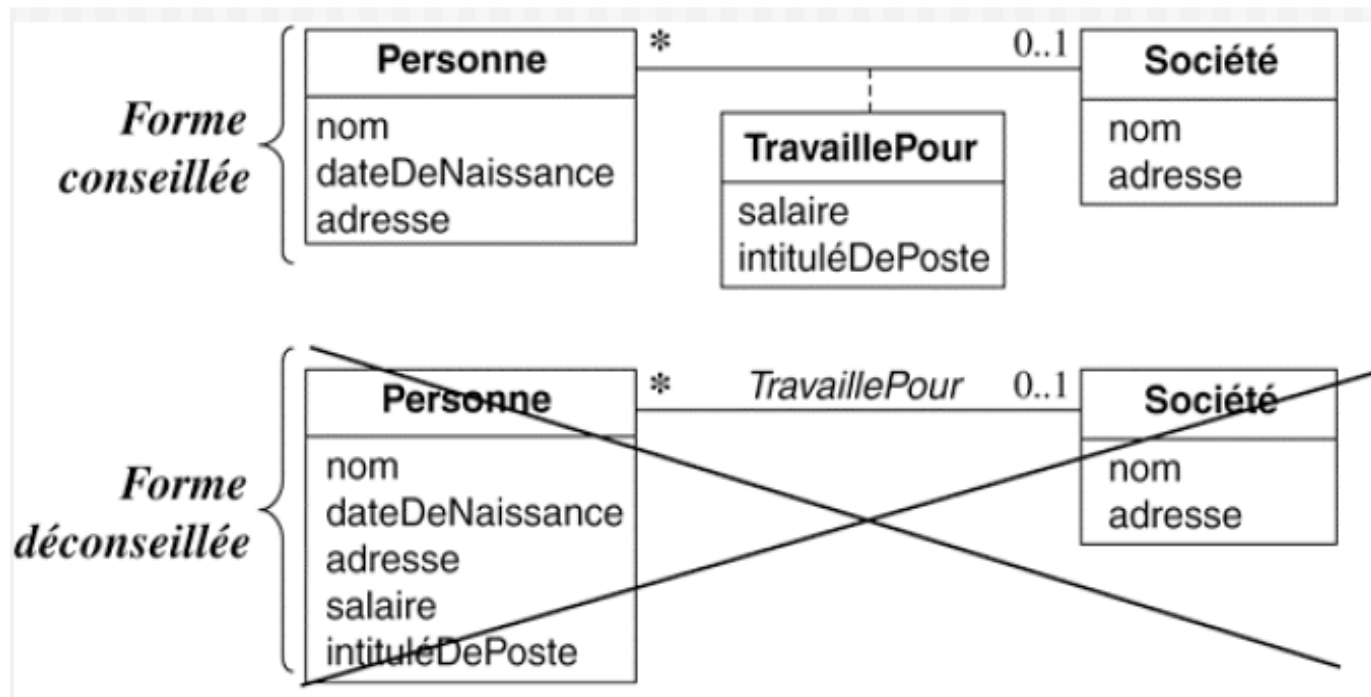
# CLASSE D'ASSOCIATION

- Une **classe d'association** est une association qui est aussi une classe,



# CLASSE D'ASSOCIATION

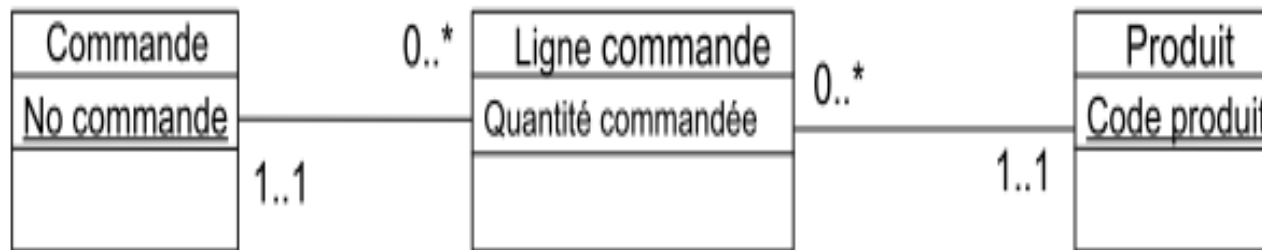
- Ne placez pas les attributs d'une association dans une classe.



# CLASSE D'ASSOCIATION

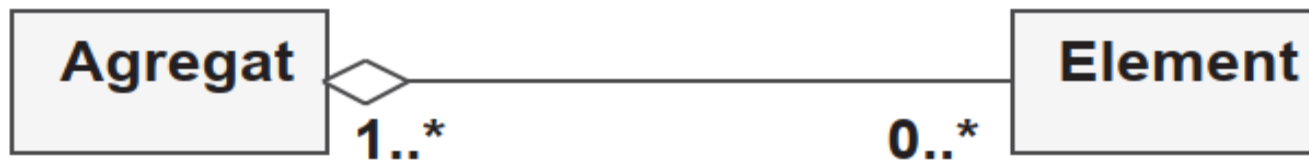
- Toute classe-association peut être remplacée par une **classe intermédiaire** et qui sert de **pivot** pour une paire d'association.

*Exemple :*



# AGRÉGATION

- Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance d'un élément dans un ensemble.
- Les agrégations n'ont pas besoin d'être nommées : implicitement elles signifient « **contient** », « **est composé de** ».
- On représente l'agrégation par l'ajout d'un losange vide du côté de l'agregat (l'ensemble).



# COMPOSITION

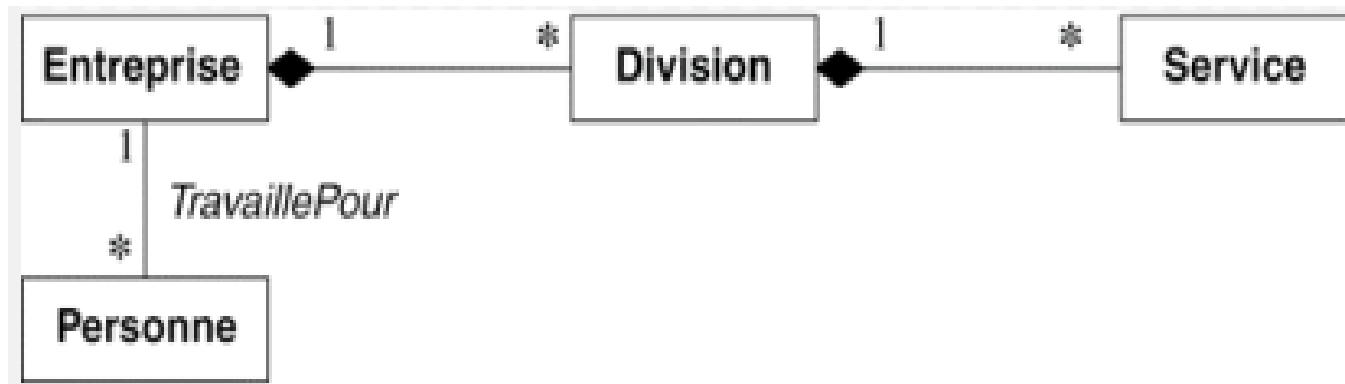
Une composition est une agrégation plus forte impliquant que :

- un élément ne peut appartenir qu'à un seul agrégat composite (agrégation non partagée);
- la destruction de l'agrégat composite (l'ensemble) entraîne la destruction de tous ses éléments (les parties)
- le composite est responsable du cycle de vie des parties.



# COMPOSITION - EXEMPLES

- Une partie constituante ne peut appartenir à plus d'un assemblage;
- Une fois une partie constituante affectée à un assemblage, sa durée de vie coïncide avec ce dernier.



# QUAND METTRE UNE COMPOSITION PLUTÔT QU'UNE AGRÉGATION ?

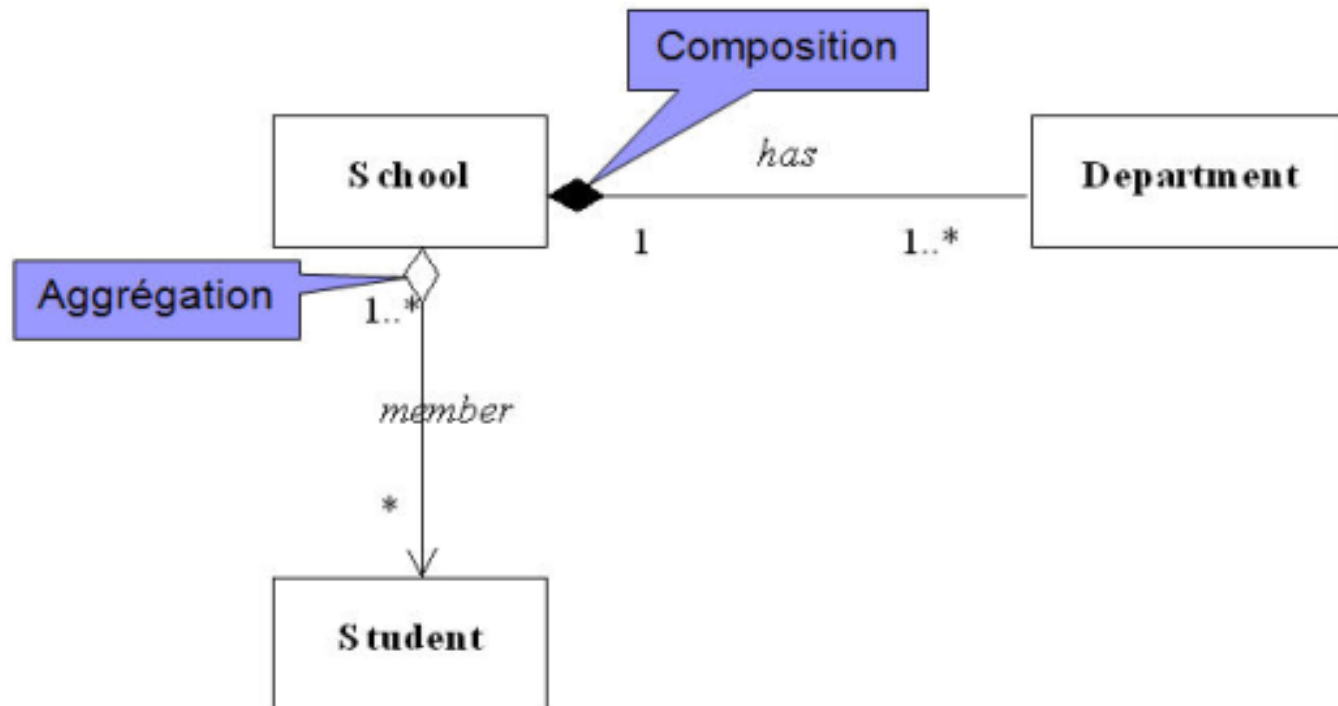
Pour décider de mettre une composition plutôt qu'une agrégation, on doit se poser les questions suivantes :

- Est-ce que la destruction de l'objet composite (du tout) implique nécessairement la destruction des objets composants (les parties)?  
C'est le cas si les composants n'ont pas d'autonomie vis-à-vis des composites.
- Lorsque l'on copie le composite, doit-on aussi copier les composants, ou est-ce qu'on peut les «réutiliser», auquel cas un composant peut faire partie de plusieurs composites?
- Si on répond par l'affirmative à ces deux questions, on doit utiliser une composition.



# AGRÉGATION VS. COMPOSITION

- Exemples:

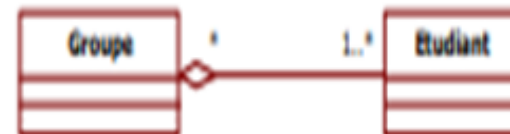




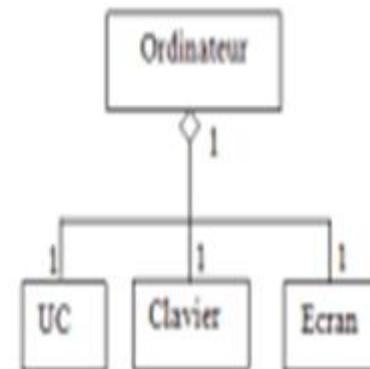
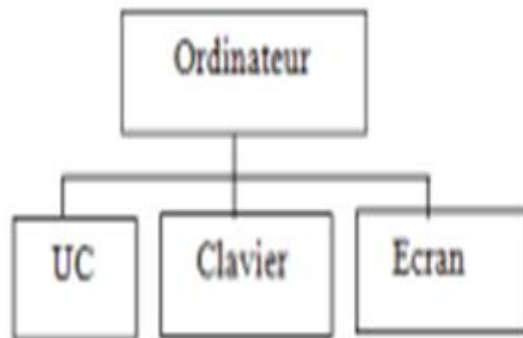
Composition :



Agrégation

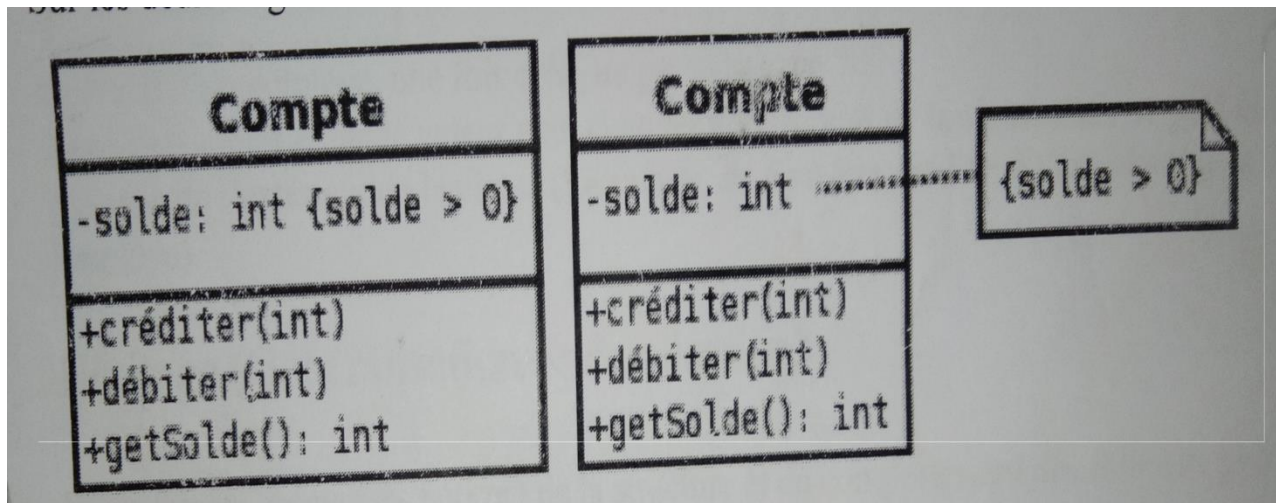


Agrégation :



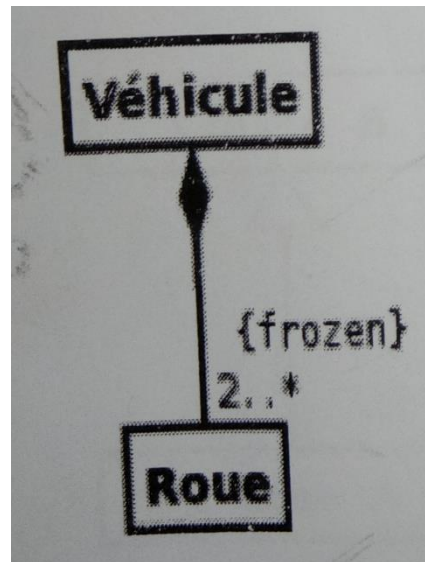
# CONTRAINTES

- UML permet d'associer une contrainte à un élément de modèle de plusieurs façons:
- **Exemple1:** Contrainte sur un attribut qui doit être positif



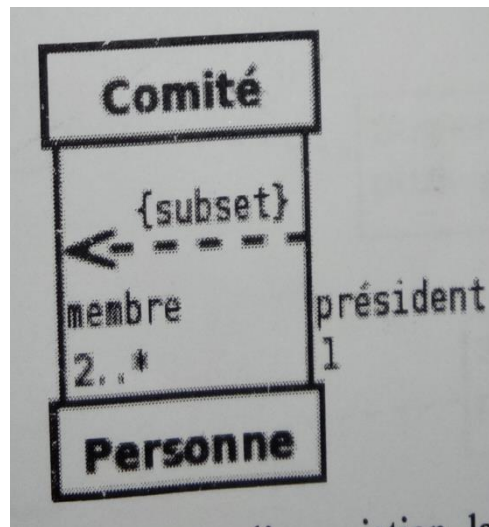
# CONTRAINTES

- **Exemple 2** : La contrainte **frozen** signifie que l'association, la classe ou l'attribut, une fois créé, ne changera jamais (ici frozen précise que le nombre de roues d'un véhicule ne peut pas varier).



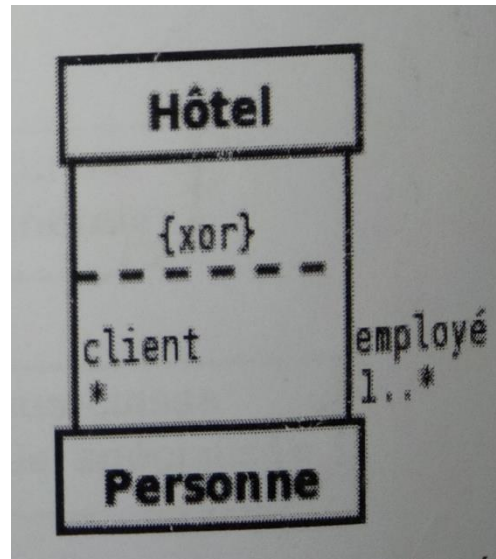
# CONSTRAINTES

- **Exemple3:** Contrainte **subset**, précise que le président est également un membre du comité:



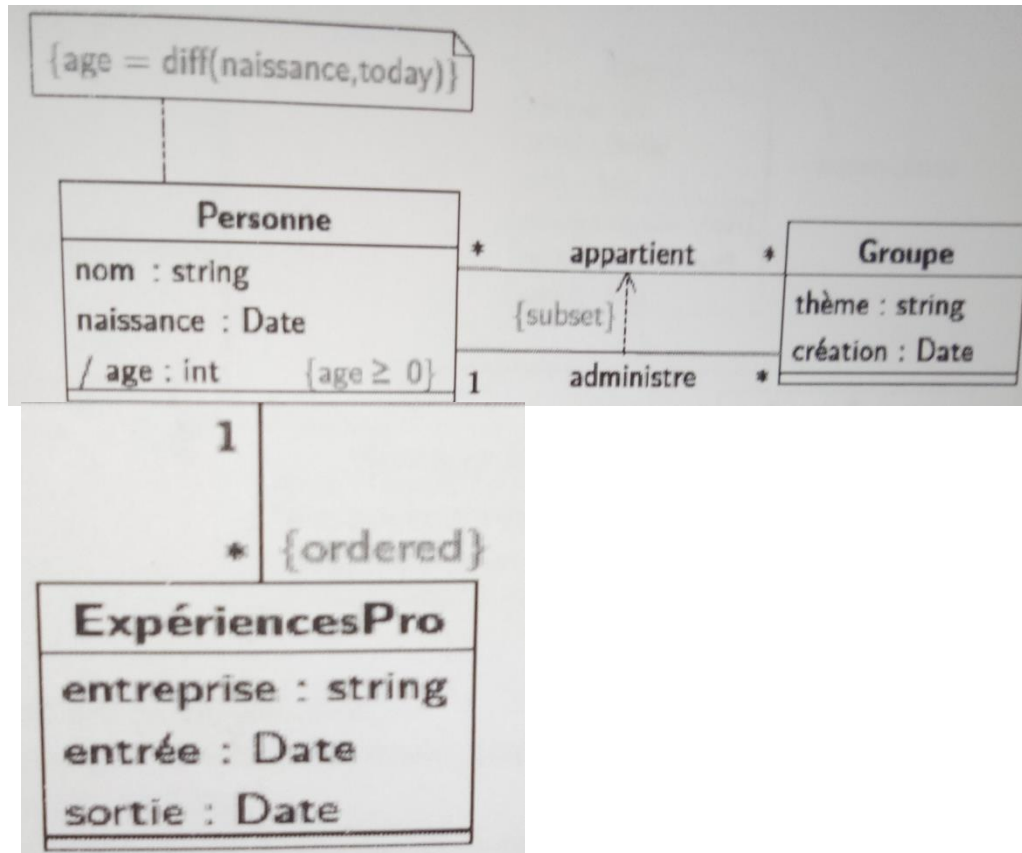
# CONSTRAINTES

- **Exemple 4:** La contrainte XOR (ou exclusif) précise que les employés de l'hôtel n'ont pas le droit de prendre une chambre dans ce même hôtel:



# EXEMPLE

- L'âge est toujours positif
- L'âge est calculé comme la différence entre la date de naissance et la date d'aujourd'hui
- L'administrateur d'un groupe en est un membre
- On a accès aux expériences professionnelles dans l'ordre



# EXEMPLE

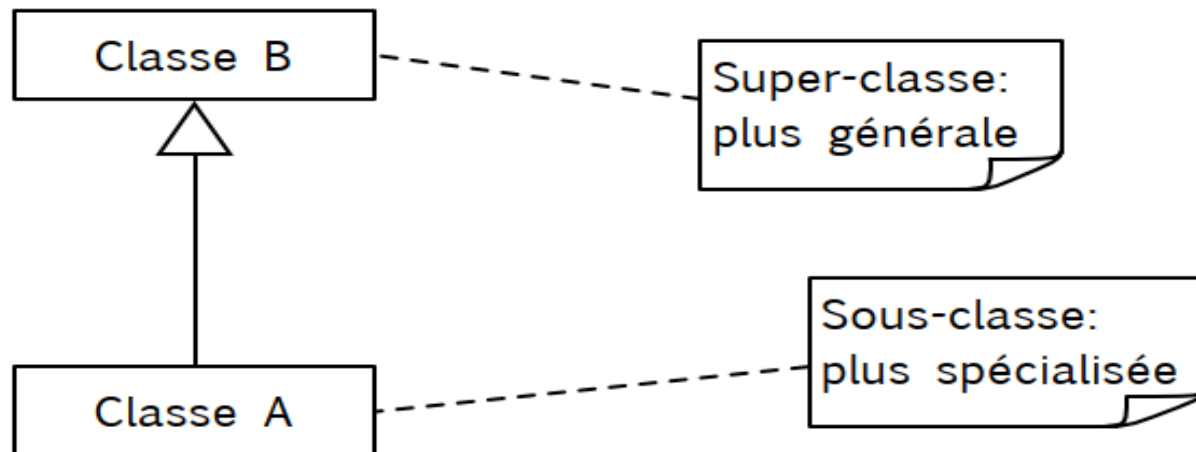
- Le NAS d'un employé, une fois créé, ne peut changer
- Read only signifie que la valeur peut changer à l'intérieur de la classe mais ne peut être changée de l'extérieur de la classe (le salaire ne peut être changé à l'extérieur de la classe employé)

Employé
NAS: string {unique}{frozen} Poste: string /Salaire: real {read only}



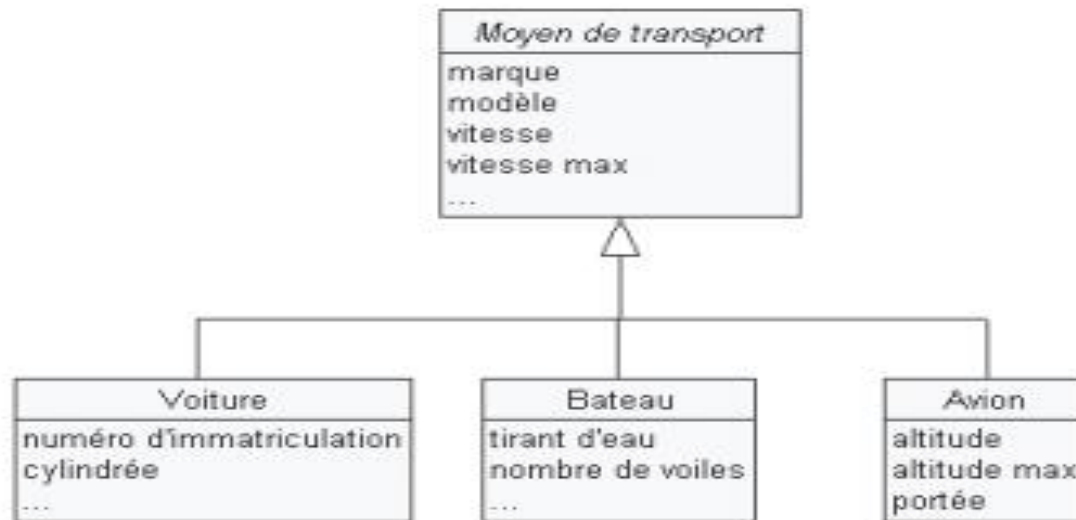
# HÉRITAGE (GÉNÉRALISATION/SPÉCIALISATION)

- L'héritage est une relation de spécialisation / généralisation.
- Les éléments spécialisés héritent de la **structure** et du **comportement** des éléments plus généraux (**attributs** et **opérations**)



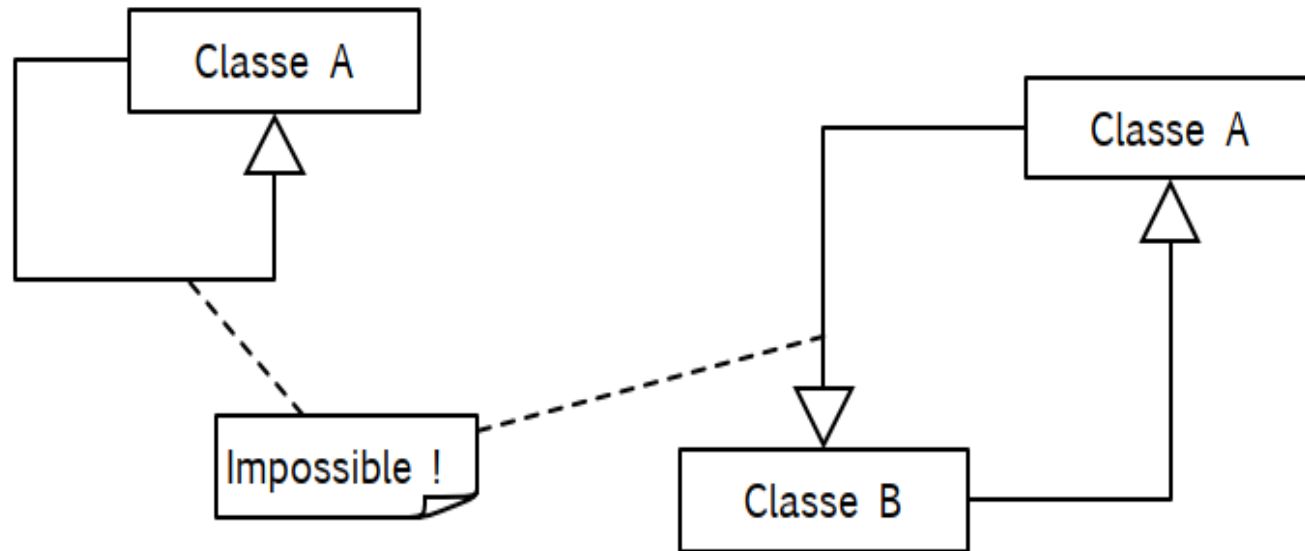
# HÉRITAGE

- **Principe de substitution** : toutes les propriétés de la classe parent doivent être valables pour les classes enfant
- Principe du « **A est un B** » ou « **A est une sorte de B** » : toutes les instances de la sous-classe sont aussi instances de la super-classe.
- Par exemple, toute opération acceptant un objet d'une classe Animal doit accepter tout objet de la classe Chat (l'inverse n'est pas toujours vrai).



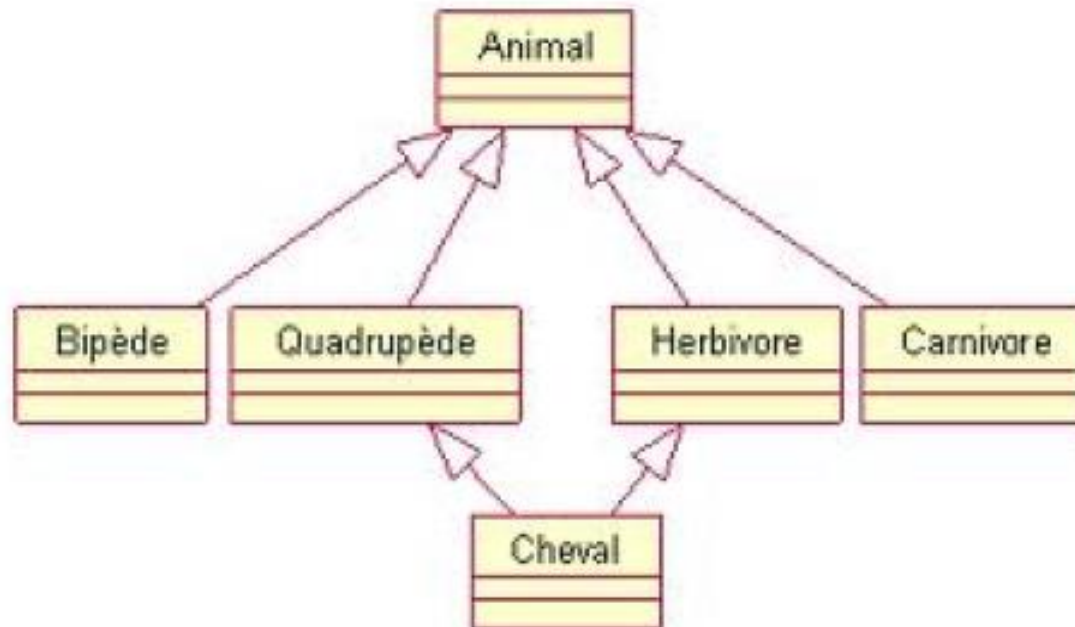
# HÉRITAGE

- L'héritage: Relation non-réflexive, non-symétrique



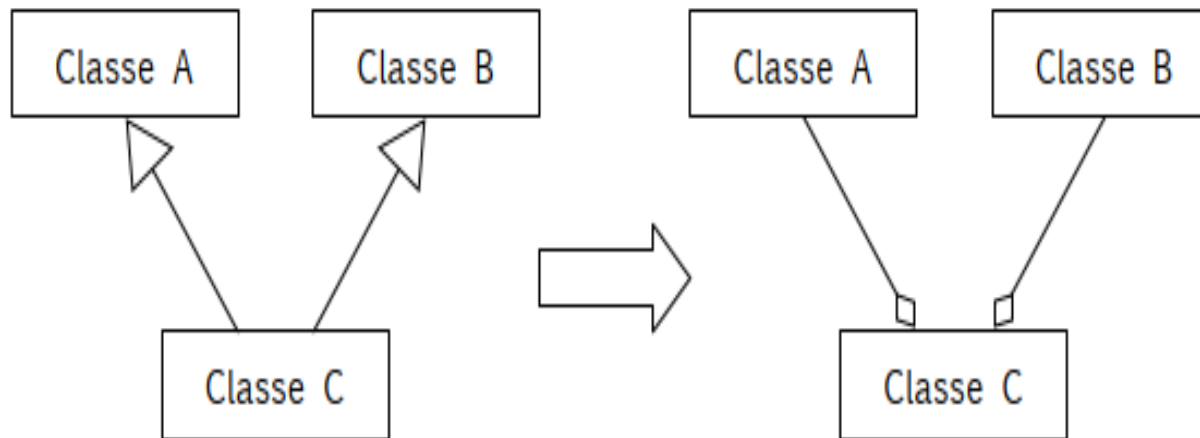
# HÉRITAGE MULTIPLE

- Une classe peut avoir plusieurs classes parents..



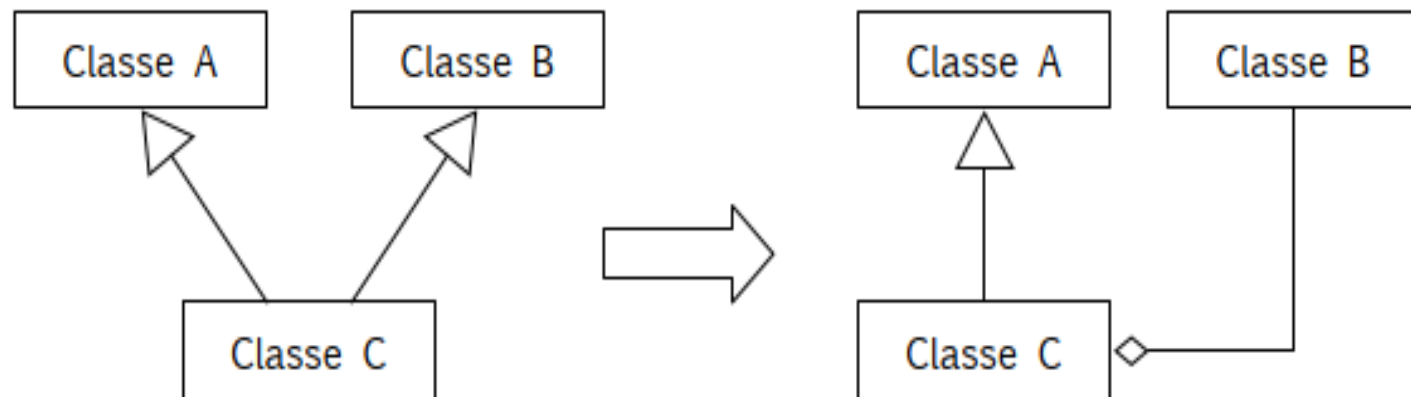
# COMMENT ÉVITER L'HÉRITAGE MULTIPLE ?

- Première solution : déléguer



# COMMENT ÉVITER L'HÉRITAGE MULTIPLE ?

- Deuxième solution : hériter de la classe la plus importante et déléguer les autres



# ENUMÉRATIONS

- Structures qui définissent une **liste de valeurs possibles**, permettant de créer des types de données personnalisés.
- Une énumération ne se définit pas par une classe, mais par un classeur stéréotypé « énumération ».
- Il s'agit d'un type de données, possédant un nom, et utilisé pour énumérer un ensemble de littéraux correspondant à toutes les valeurs possibles que peut prendre une expression de ce type.

# EXAMPLE:

Enseignant
CodeEns: string NomEns: string PrénomEns: string <b>GradEns: GradeE</b>

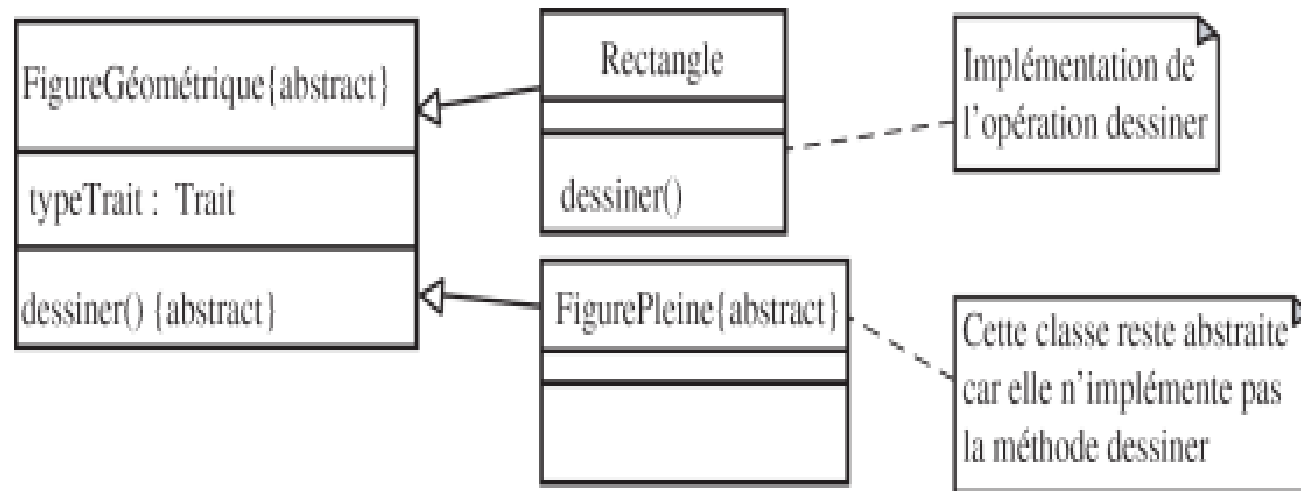
<<enumeration>> <b>GradeE</b>
Professeur MCA MCB MAA MAB

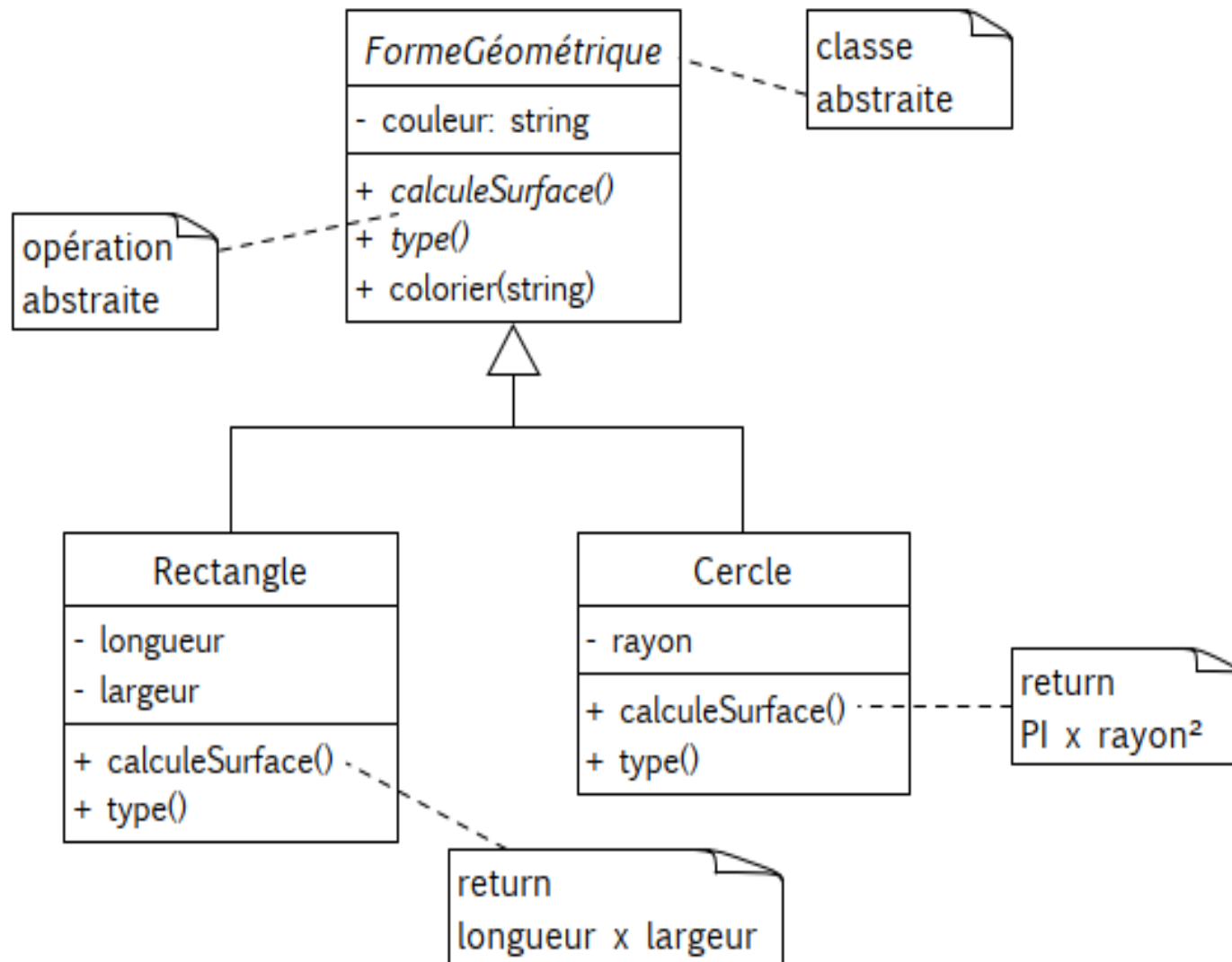


# CLASSES ABSTRAITES

- Une méthode est dite abstraite lorsqu'on connaît son entête (signature) mais pas la manière dont elle peut être réalisée.
  - Il appartient aux classes enfant de définir les méthodes abstraites.
- Une classe est dite abstraite lorsqu'elle définit au moins **une méthode abstraite** ou lorsqu'une **classe parent contient une méthode abstraite non encore réalisée**.

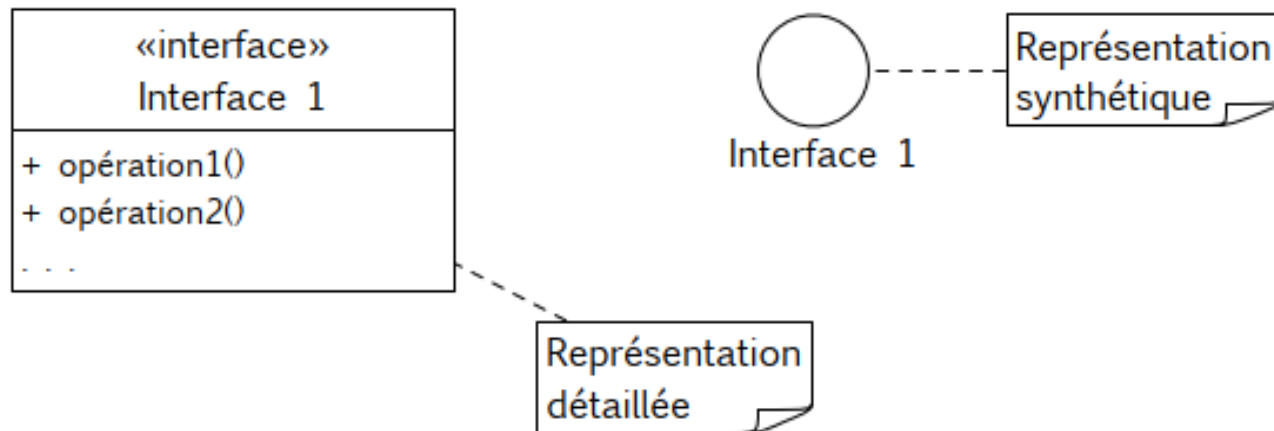
# CLASSES ABSTRAITES - EXEMPLES





# LES INTERFACES

- Une interface spécifie un ensemble d'opérations (comportement)
- C'est un contrat:
  - Les classes liées s'engagent à respecter le contrat
  - elles doivent mettre en œuvre les opérations de l'interface

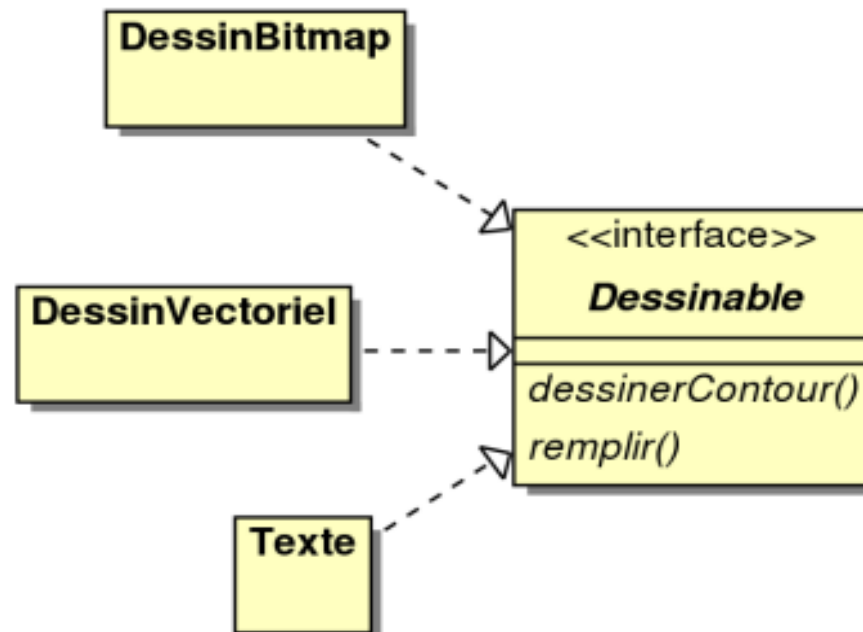


# LES INTERFACES

- Le rôle d'une interface est de grouper un ensemble d'opérations assurant un service cohérent offert par une classe,
- Une interface est définie comme une classe, avec les mêmes compartiments. Les principales différences sont:
  - La non utilisation du mots clé « Abstract », car l'interface et toutes ses méthodes sont, par défaut abstraites;
  - L'ajout du stéréotype « interface » avant le nom de l'interface

# LES INTERFACES

- On utilise une relation de type réalisation entre une interface et une classe qui l'implémente.
- Les classes implémentant une interface doivent implémenter toutes les opérations décrites dans l'interface



# PACKAGES

- Un package permet de grouper des éléments
- Un package sert d'espace de désignation
- Un package peut inclure d'autres package
- Un package peut importer d'autres package
- L'héritage entre package est possible

