

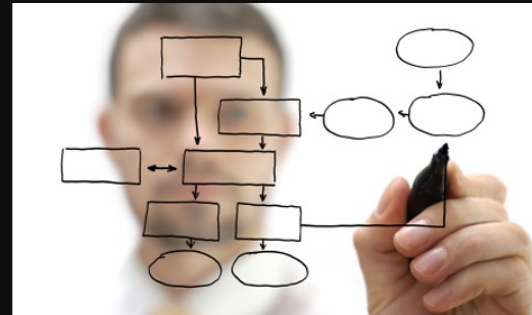


**Université de Corse**  
**2024-2025**  
**L3 Informatique**

# **UE Conception Orientée Objet**

## **Modélisation UML**

### **CH2 – Modèle du Domaine**





# Conception Orientée Objet

## Plan du Cours



CH1 – UML et ACOO



CH2 – MODELE DU DOMAINE

CH3 – MODELE DES CAS D'UTILISATION

CH4 – MODELE D'ANALYSE



# CH2 – MODELE DU DOMAINE

## 2.1 – Présentation

- Objectifs
- Artefacts

## 2.2 – Formalisme des Diagrammes de classe

## 2.3 – Diagrammes d'objets

## 2.4 – Démarche de construction



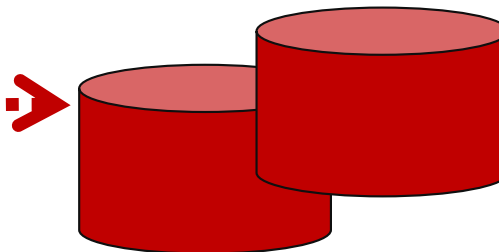


# MODELE DU DOMAINE

## Objectif du modèle du domaine

- Comprendre et décrire les concepts essentiels dans le contexte du système
- « Concepts métiers » : concepts manipulés par les experts du domaine
- Ensemble des données importantes du domaine

*Les données que l'on souhaite  
conserver de manière durable*

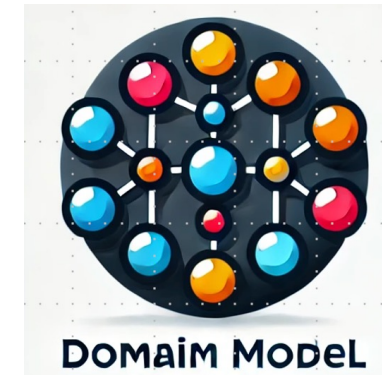
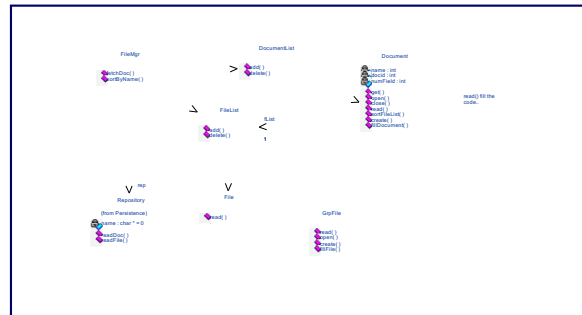




# MODELE DU DOMAINE

## Artefacts du modèle du domaine

+ *Diagrammes d'objets*



## Diagramme de classes du domaine

- **Classes**
- **Relations**
  - Associations, agrégations, composition
  - Généralisation
- **Attributs**

+ *Packages*



# CH2 – MODELE DU DOMAINE

## 2.1 – Présentation

## ➔ 2.2 – Formalisme des Diagrammes de classe

- Définition
- Diagrammes de classe et étapes d'ACOO
- Formalisme



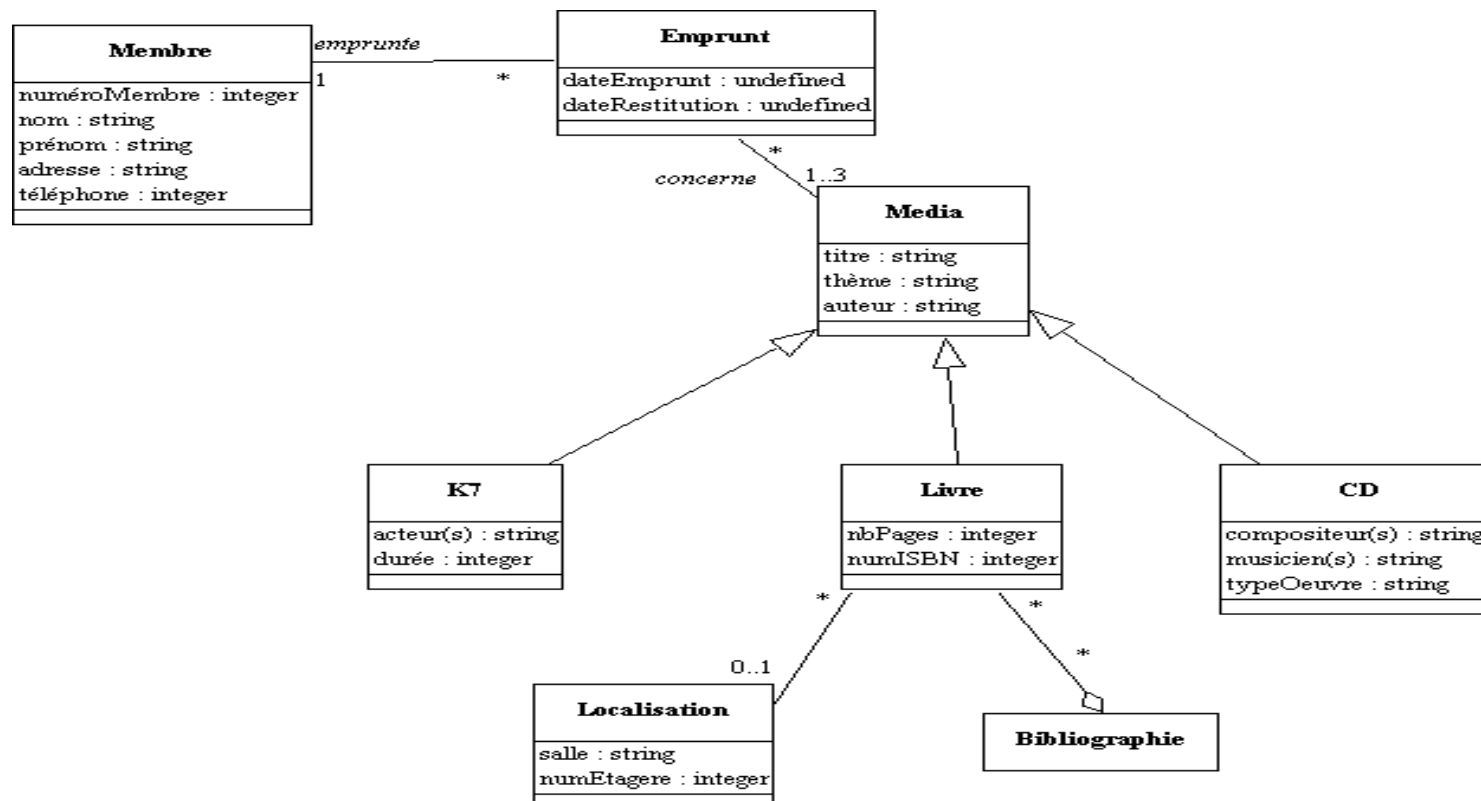
## 2.3 – Diagrammes d'objets

## 2.4 – Démarche de construction



# Définition

- Un **diagramme de classes** exprime la structure d'un système en termes de classes et de relations entre ces classes







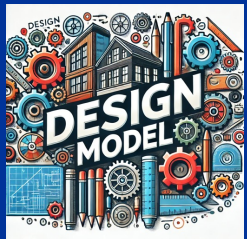
# Diagrammes de classes et étapes d'ACOO

Le **diagramme de classes du domaine** est le premier diagramme de classe défini au cours d'une démarche d'ACOO.

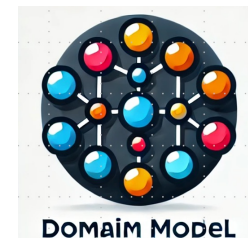
Il est ensuite complété pour obtenir le diagramme de **classes d'analyse** lui-même complété pour aboutir au diagramme de **classes de conception**.

## 3 - Diagramme de classes de Conception

### 2 - Diagramme de classes d'Analyse



### 1 - Diagramme de classes du Domaine

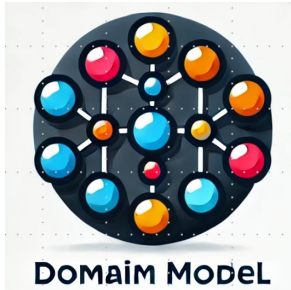








# Diagrammes de classes et étapes d'ACOO

## Description de plus en plus détaillée

	Diagramme de classes du domaine	Diagramme de classes d'analyse	Diagramme de classes de conception
Classes	Attributs (nom:type)  	Opérations Visibilités Classes abstraites Interfaces  	API Classes et interfaces (liées à un environnement de développement)  
Relations	Associations Agrégations Compositions Généralisations	Réalisations Dépendances	



# CH2 – MODELE DU DOMAINE

## 2.1 – Présentation

## → 2.2 – Formalisme des Diagrammes de classe

- Rappel : notion d'objet
- Représentation d'une classe
- Attributs et opérations
- Relations entre classes
- Classes Abstraites et interfaces
- Autres concepts (Conception)



## 2.3 – Diagrammes d'objets

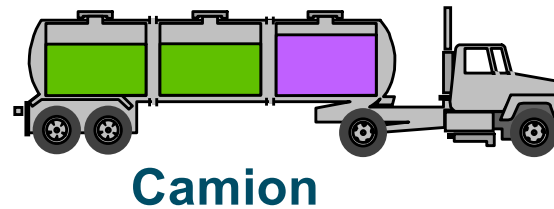
## 2.4 – Démarche de construction



# Notion d'objet

## Qu'est-ce qu'un objet?

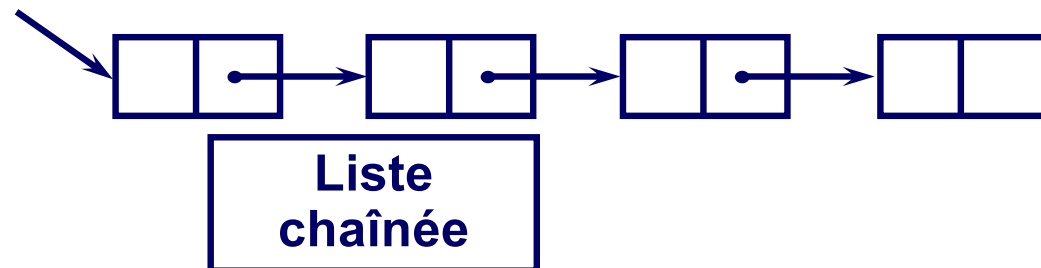
Entité physique



Entité conceptuelle



Entité logicielle



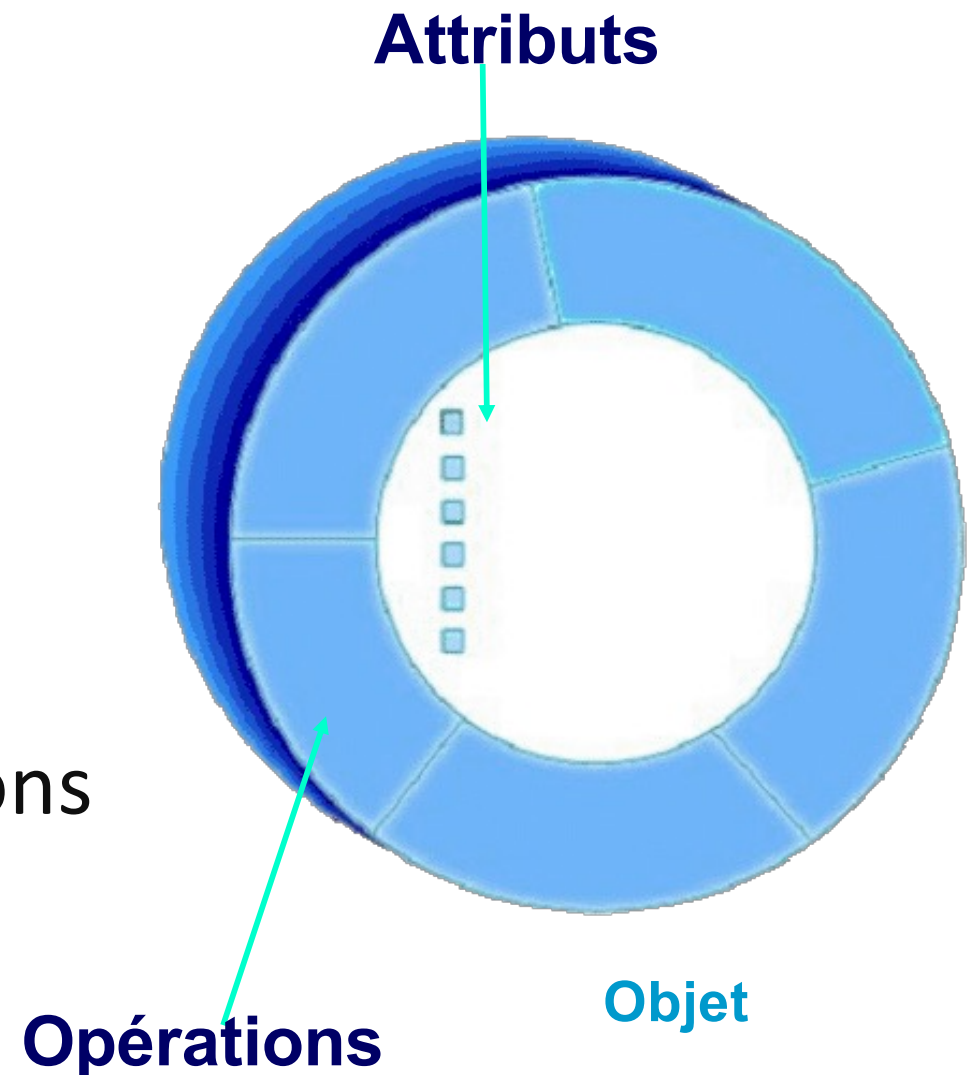


# Notion d'objet

**Objet** = entité aux frontières précises qui possède une identité, un état et un comportement

**Etat** = attributs + *liens*

**Comportement** = opérations





# Notion d'objet en JAVA



nom *Variable définie sur une classe*

age

discipline

grade

●  
**prof001**

Nimbus
30
physique
MCF Ech.6
actif
4

- Une **adresse (ou référence)** en mémoire qui permet d'identifier l'objet
- Un **état** qui est représenté par un ensemble de valeurs attribuées à ses variables d'instances
- Un **comportement** défini par des fonctions ou sous-programmes appelés méthodes

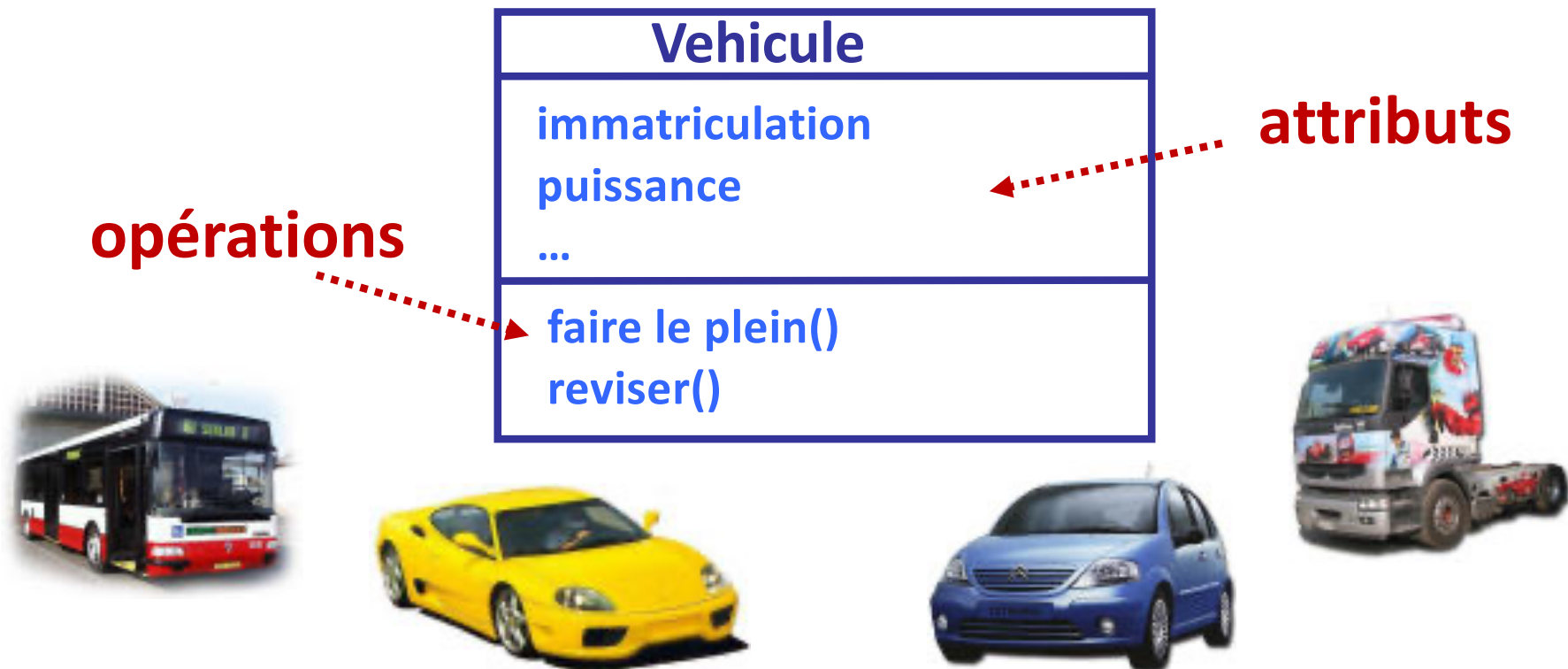


# Formalisme du Diagramme de classes

## Représentation d'une classe

Une **classe** est une abstraction d'un ensemble d'objets

- Mise en valeur des caractéristiques les plus importantes.
- Suppression des autres caractéristiques.





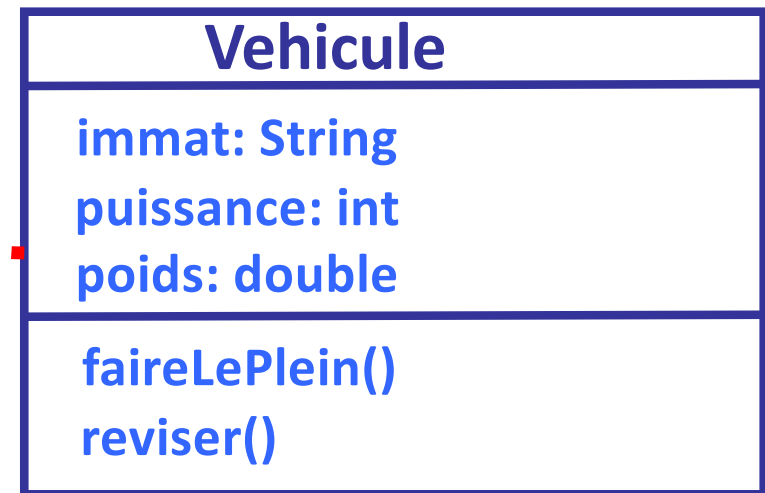
# Classes en JAVA et en UML

## Une classe en JAVA

```
class Vehicule{  
    /** l'immatriculation de ce véhicule */  
    String immat;  
    /** la puissance */  
    int puissance;  
    /** La consommation de ce véhicule. */  
    double poids;  
  
    ...  
    void faireLePlein() {  
        ....  
    }  
    void reviser() {  
        ....  
    }  
}
```

attributs

## Une classe en UML

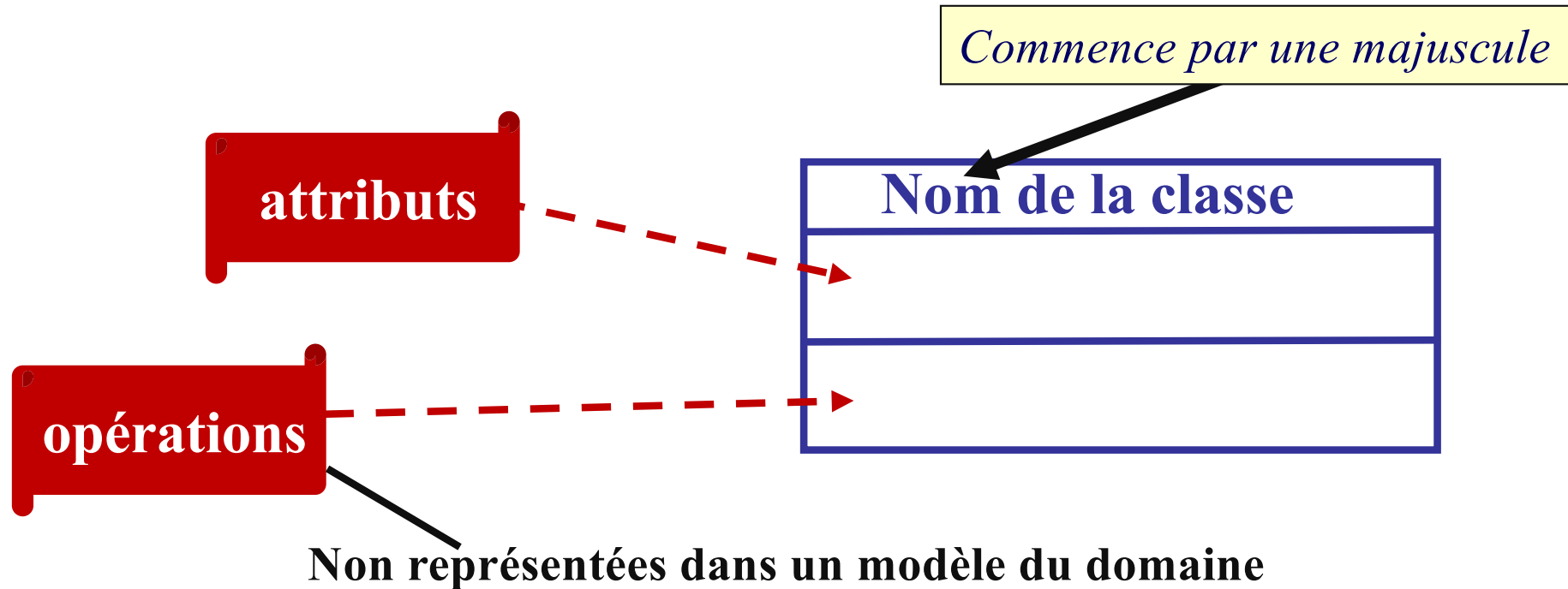


méthodes





# Représentation d'une classe



## Exemples

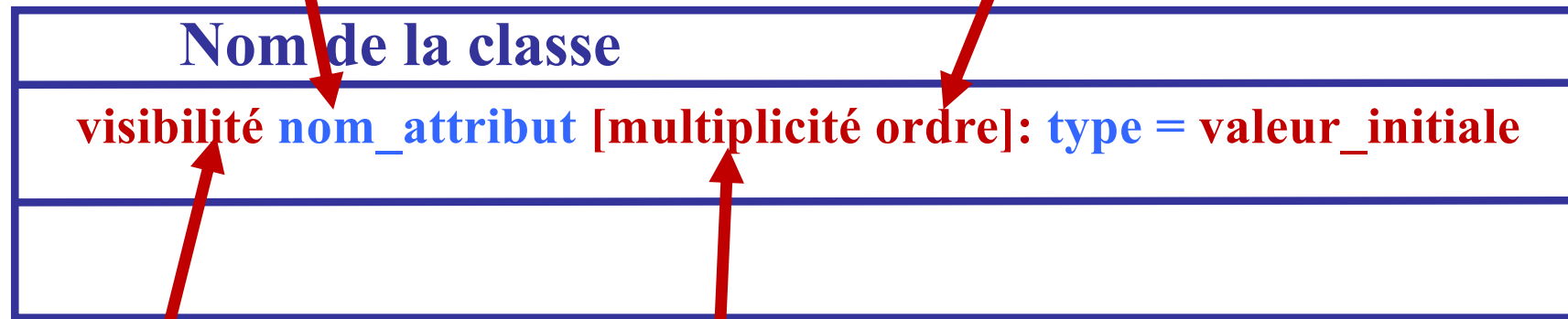




# Représentation d'une classe

*Commence par une minuscule*

trié ou non trié



+ : public  
# : protégé  
- : privé


[I..S] nombre de valeurs compris entre I et S  
[1] exactement 1 valeur  
[\*] nombre quelconque de valeurs  
[0..2] de 0 (aucune) à 2 valeurs  
[2..\*] minimum 2 valeurs




# Représentation d'une classe

## Attribut (*d'instance*) =

- défini par un nom, un type et éventuellement une valeur initiale
- valeur spécifique pour chaque instance

<u>v1:Vehicule</u>
<ul style="list-style-type: none"><li>- immatriculation = « 1000GH2B »</li><li>- puissance = 18</li><li>- Poids = 15</li></ul> 

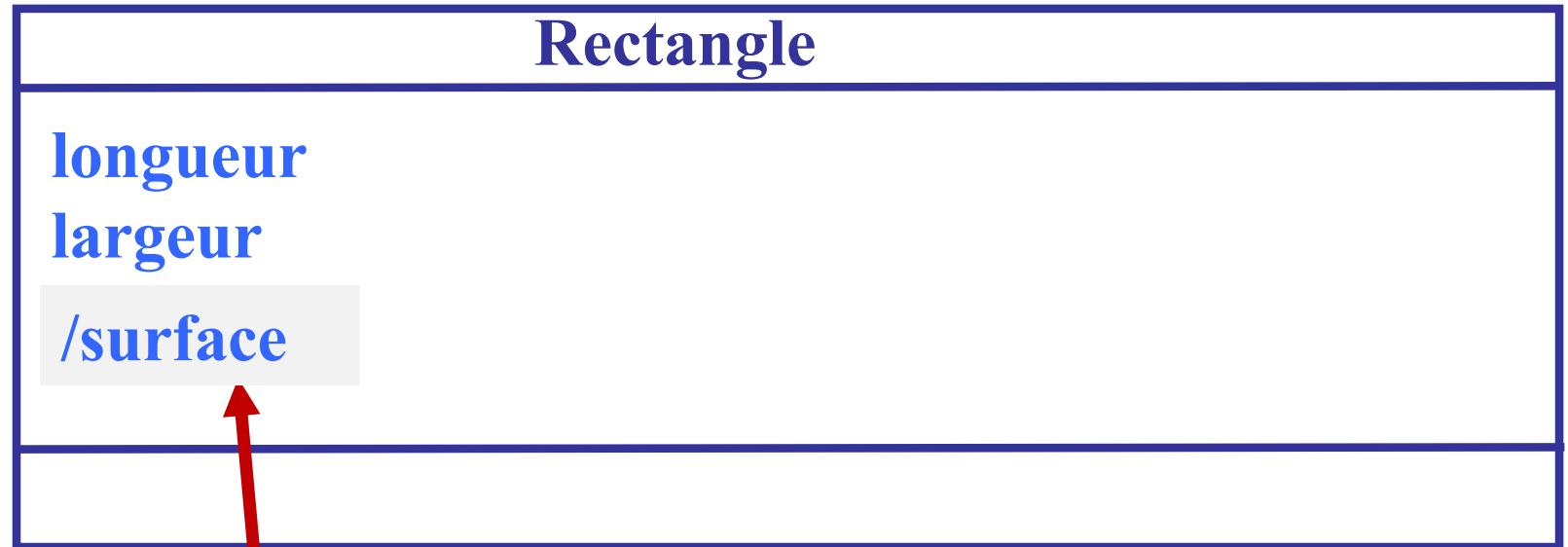
<u>v2:Vehicule</u>
<ul style="list-style-type: none"><li>- immatriculation = « 2222ML2A »</li><li>- puissance = 5</li><li>- Poids = 2,5</li></ul> 

*Valeurs propres à chaque instance*



# Représentation d'une classe

## Exemple



**Attribut dérivé (/nom):**

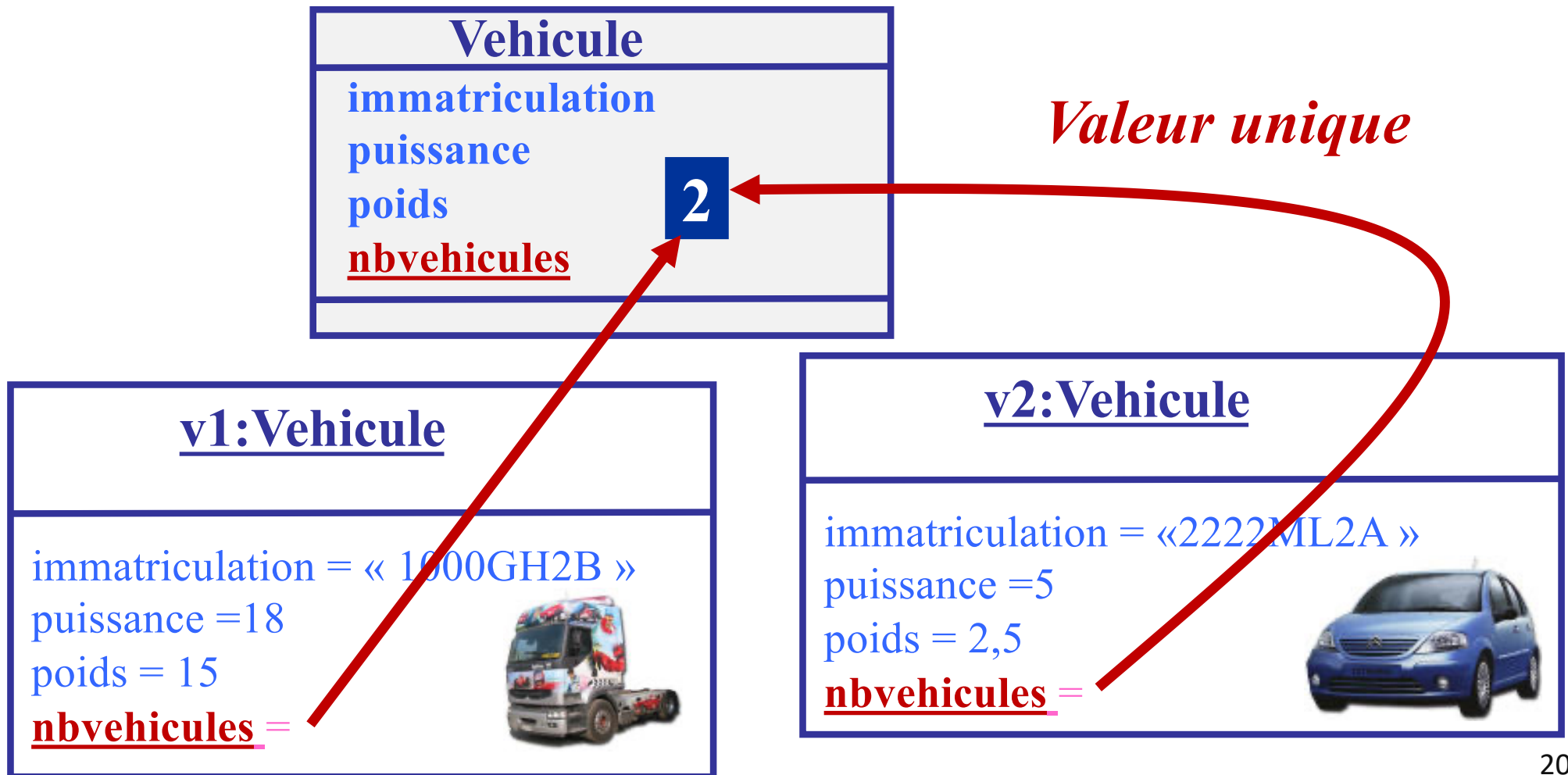
valeur calculée à partir de celles d'autres attributs



# Représentation d'une classe

**Attribut de classe =**

**Valeur partagée par toutes les instances de la classe**





# Attributs et méthodes de classe en Java



```
public class Vehicule{
private String immatriculation;
private int nbPlaces;
Private int age;
private static int nbVehicule=0;
private static int totalNbPlaces=0;
Public Vehicule(String immatriculation,
                int nbPlaces){
    this.immatriculation=immatriculation;
    this.nbPlaces=nbPlaces;
    this.age=0;
    nbVehicule++;
    totalNbPlaces=totalNbPlaces+nbPlaces;
}
public void augmenterAge(){ age++;}
public static void afficherNbVehicule(){
    System.out.println("Nombre de Vehicules "+ nbVehicule);
}
```

instance

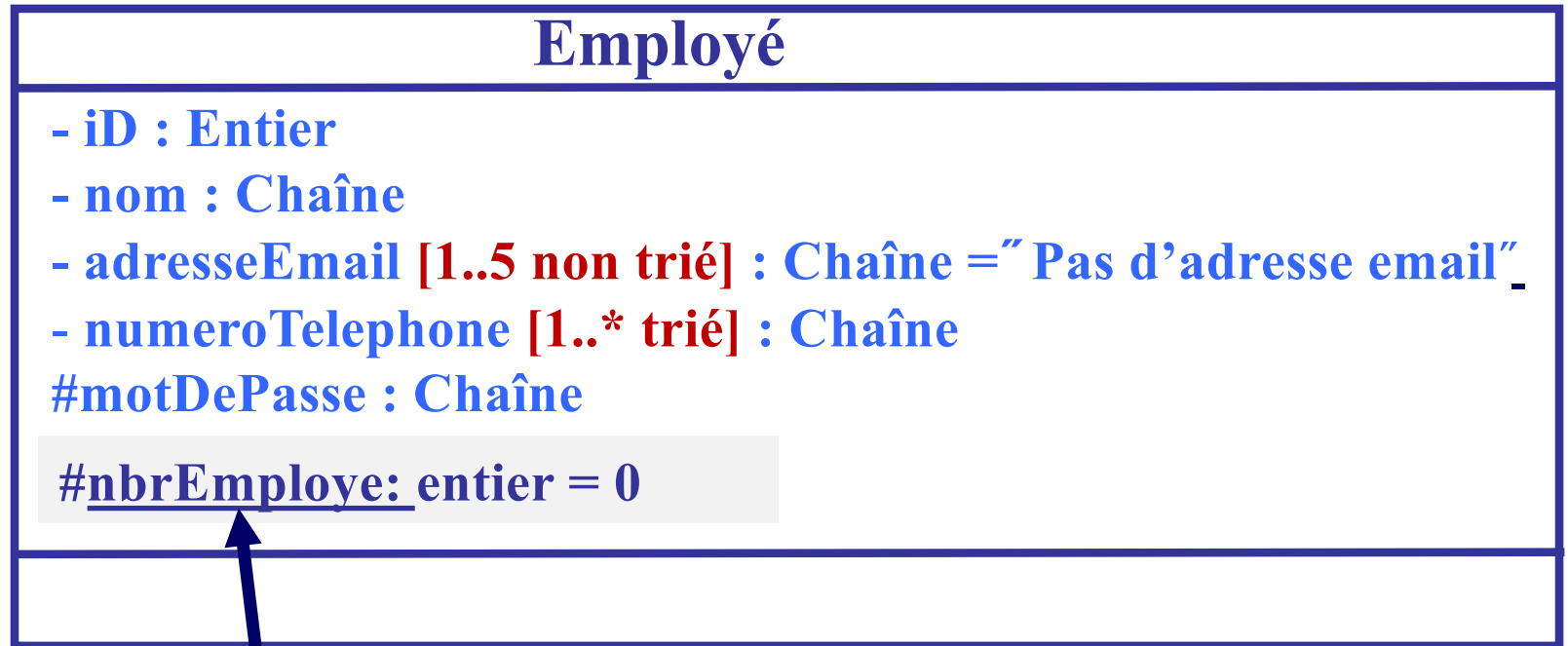
```
class TestVehicule{
    public static void main(String[] args){
        Vehicule v1=new
        Vehicule("2222CX23" , 6);
        v1.augmenterAge();
        Vehicule.afficherNbVehicule();
    }
}
```

classe



# Représentation d'une classe

## Exemple



Attribut de classe (nom souligné):  
valeur partagée par tous les objets d'une classe





# Représentation des opérations



- Les **opérations** sont attachées à une classe.
- L'exécution d'une opération porte sur une instance particulière.

## v3:Vehicule



immatriculation = «1111ML2A »  
nbplaces =5  
date\_derniere\_revision = **16/09/2023**

## v4:Vehicule

immatriculation = «4444MS2B »  
nbplaces =3  
date\_derniere\_revision = **17/09/2022**

## Vehicule

immatriculation  
nbplaces  
date\_derniere\_revision

faire le plein()  
**reviser()**  
augmenternbplaces()

- Déclenchement par envoi d'un message à une instance particulière



# Représentation des opérations



## Opérations de classe =

- Exécution déclenchée par un message envoyé à la classe.
- Opération qui ne concerne pas une instance particulière mais la classe
- Manipulation des attributs de classe uniquement

<u>v1:Vehicule</u>
immatriculation = «1000GH2B » nbplaces =4



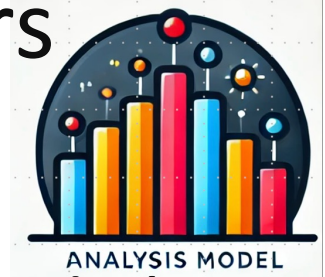
<u>v2:Vehicule</u>
immatriculation = «222ML2A » nbplaces =5

<u>v4:Vehicule</u>
immatriculation = «1111ML2A » nbplaces =3



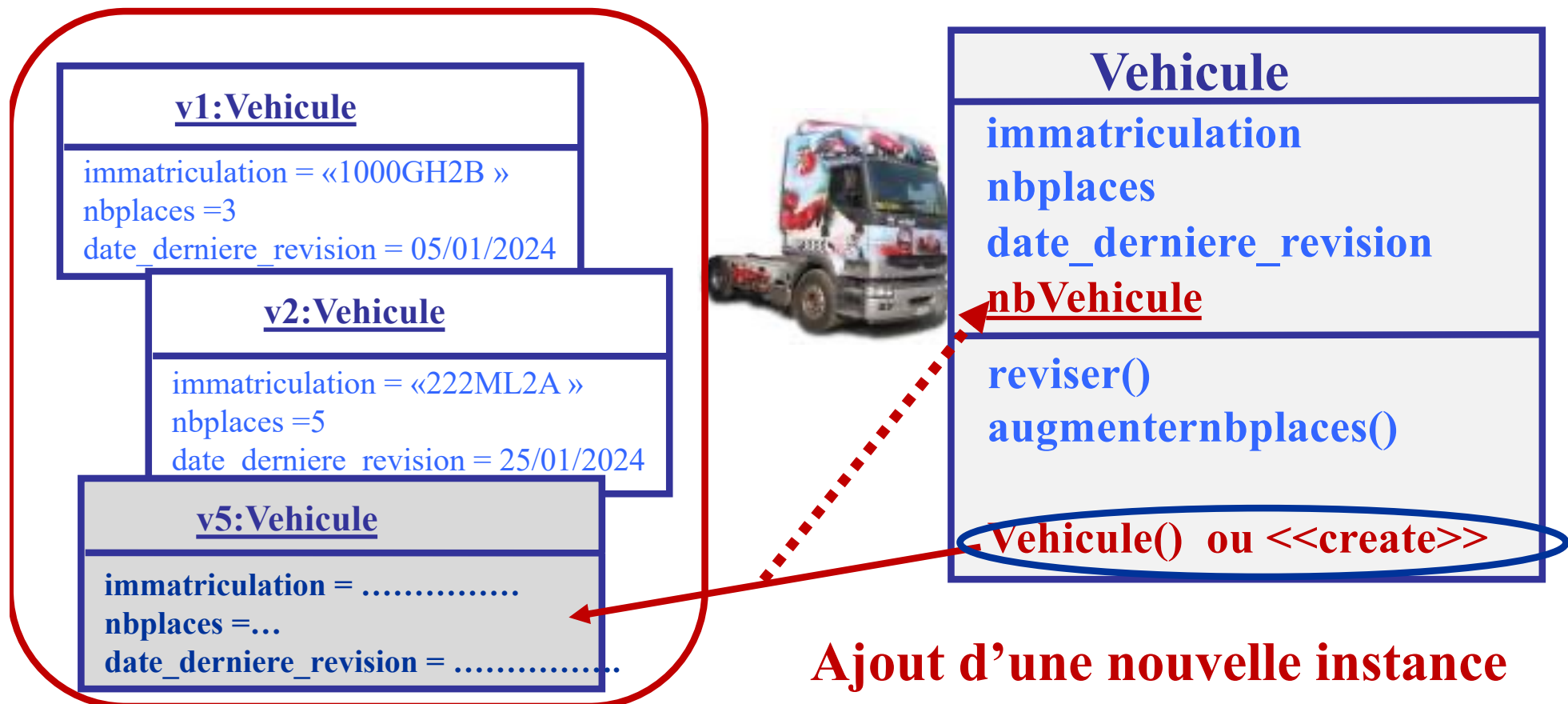
Vehicule	
immatriculation	
nbplaces	
<u>totalNbPlaces</u>	12
<u>nbVehicules</u>	3
<u>augmenternbplaces()</u> <u>calculerNbPlaceMoyen()</u> <u>afficherTotalNbplaces()</u>	

# Représentation des constructeurs



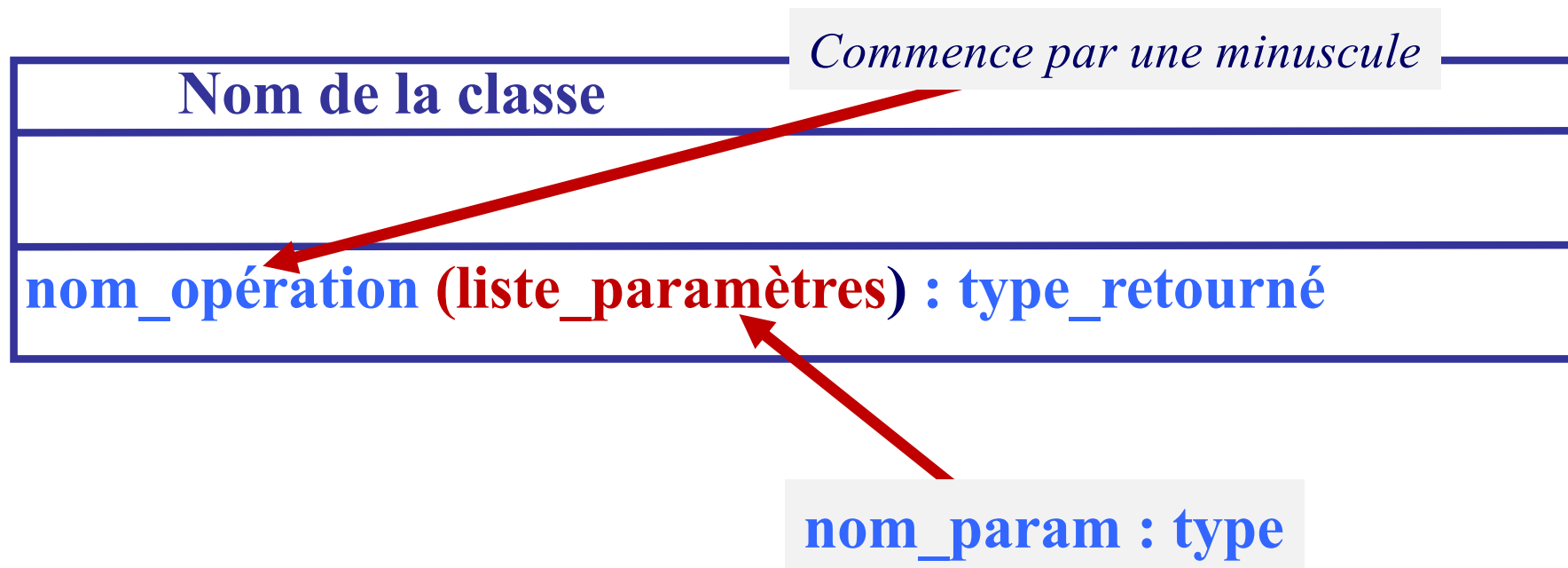
**Mécanisme d'instanciation =**

- Activation de l'opération de création d'instance de la classe: Constructeur





# Représentation des opérations





# Représentation des opérations



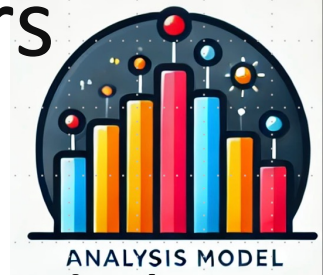
constructeur

**Exemple**

## Employe

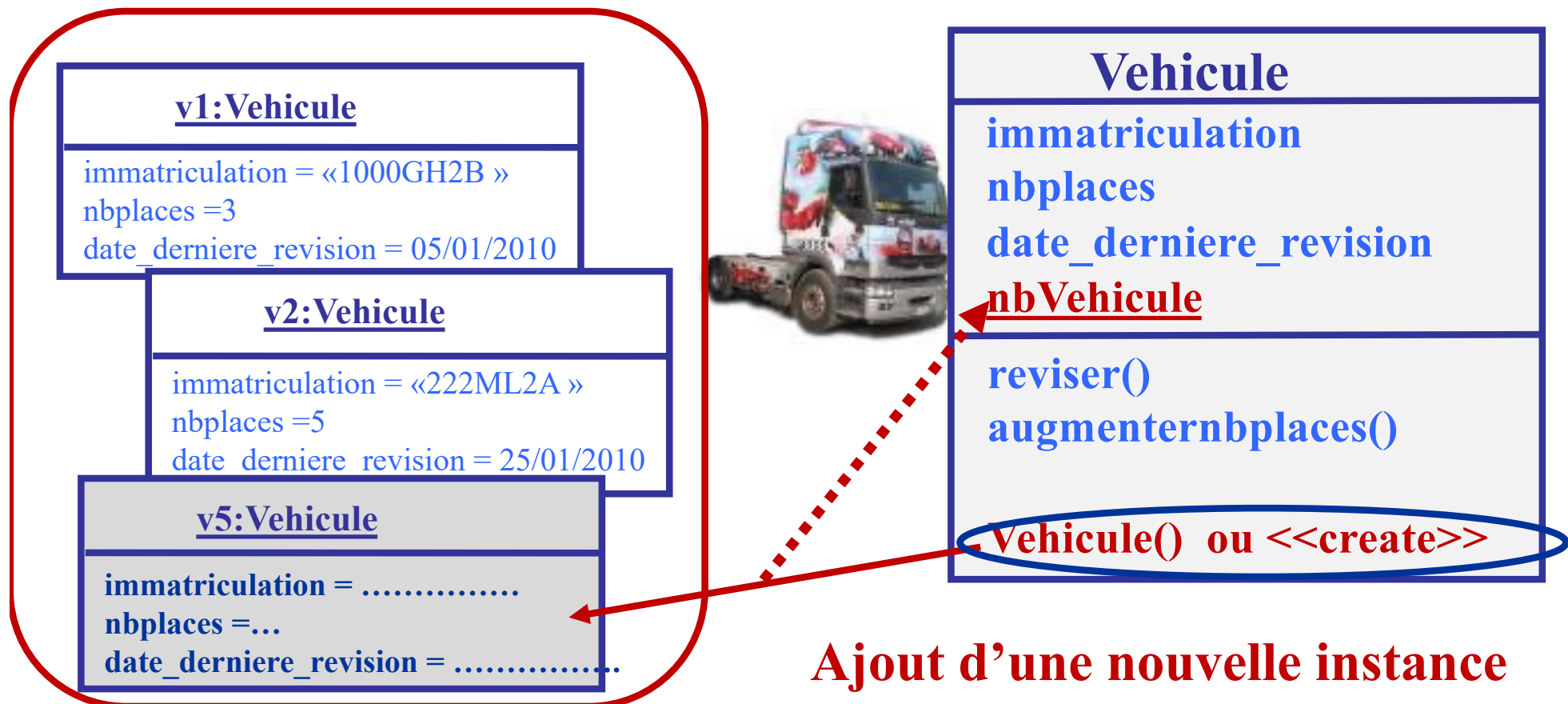
+ <<create>> Employe (nom: Chaîne)  
+ recupererID() : Entier  
+ recupererNom() : Chaîne  
+ definirNom(LeNom:Chaîne)  
+ recupererNumerotelephone(LaPriorite: Entier, Numtel:Chaîne)  
+ ajouterEmail(Iemail: Chaîne ): Booléen  
+ supprimerEmail(Iemail: Chaîne )  
+ recupererEmail(email1: Chaîne, email2: Chaîne,  
email3: Chaîne, email4 : Chaîne,  
email5 : Chaîne)  
# definirMotdepasse(lemotdePasse: Chaîne)

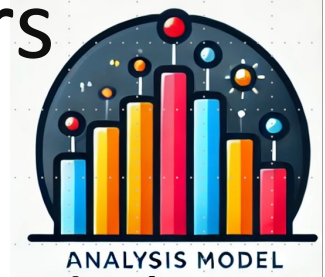
# Représentation des constructeurs



**Mécanisme d'instanciation =**

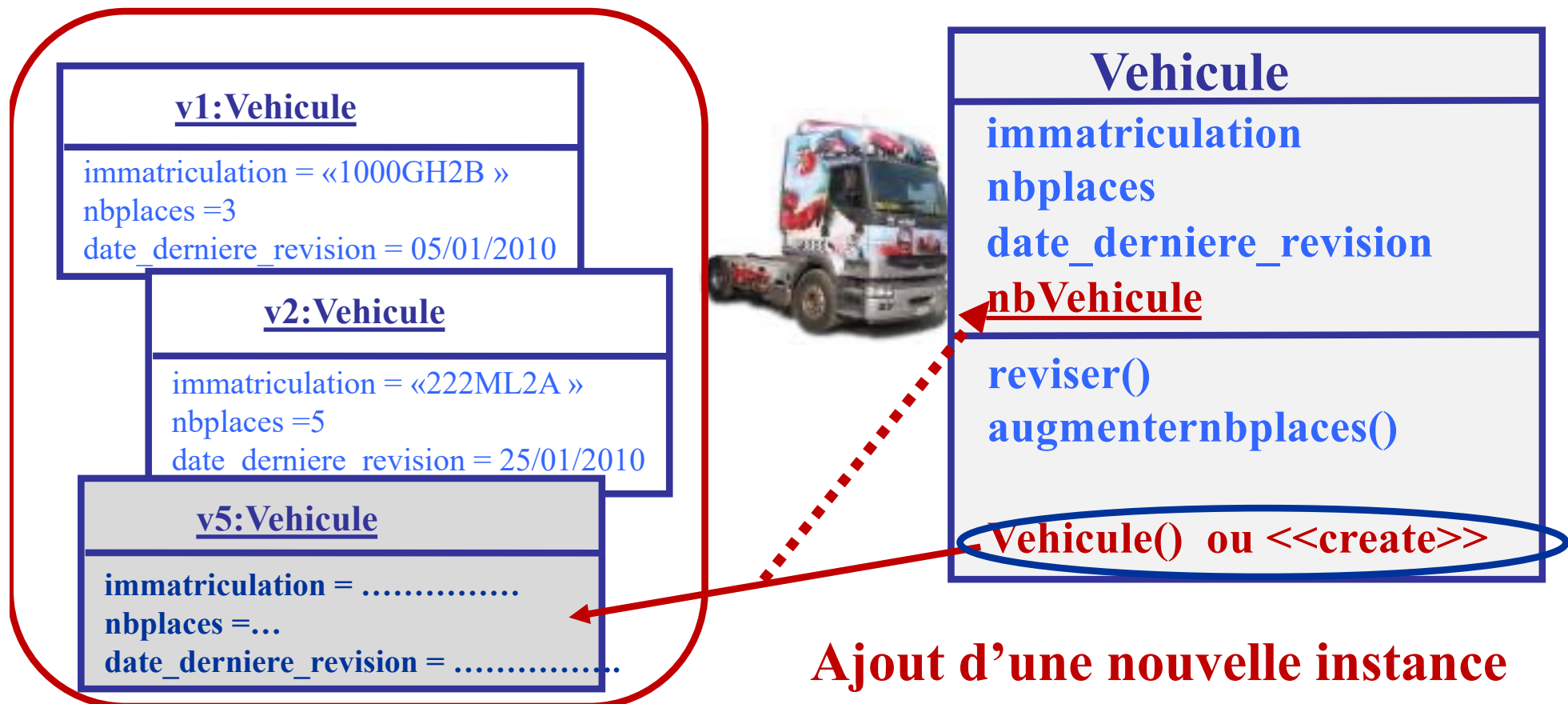
- Activation de l'opération de création d'instance de la classe: Constructeur





**Mécanisme d'instanciation =**

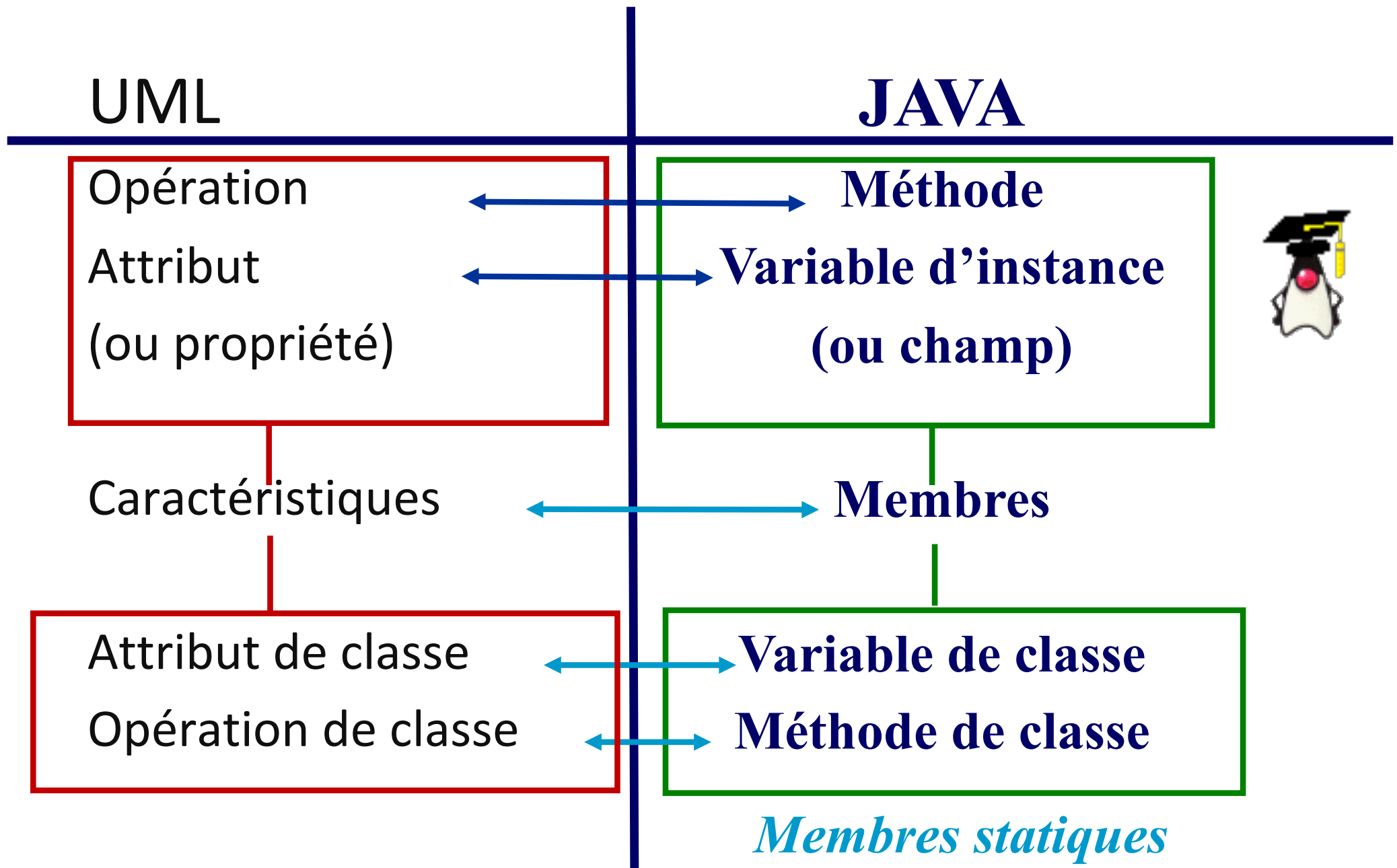
- Activation de l'opération de création d'instance de la classe: Constructeur







# Attributs et Opération - Terminologie





# Modélisation: Distinction Classe/Attribut

Comment savoir si une entité doit être modélisée par une classe ou un simple attribut?

- Attribut = **valeur simple**
  - Entité qui ne peut être caractérisée que par sa valeur (« on ne peut lui demander que sa valeur »)
- Classe = **objet**
  - Entité caractérisée par plusieurs autres entités (« on peut lui poser plusieurs questions »), il s'agit d'un objet qui possède plusieurs attributs et des liens avec d'autres objets.



# Exercice 1 : Classe ou attribut ?



Parmi les éléments, lesquels sont des objets qui appartiennent à des classes, lesquels sont de simples valeurs d'attributs caractérisant d'autres objets ?

- la guerre de 100 ans
- le réfrigérateur dans le coin de la pièce
- la couleur rouge
- une transaction boursière
- Le temps d'exécution d'un programme
- Amadeus Mozart
- l'heure de départ d'un vol
- le chiffre 3



## Exercice 2 : Classe, attribut ou opération?



Parmi les éléments suivants, lesquels peuvent être modélisés par une classe? par un attribut ? par une opération?

	Classe	Attribut	Opération
Un immeuble			
Une longueur			
Une ville			
Une superficie			
Une couleur			
Une personne			
Une date de naissance			
Un age			

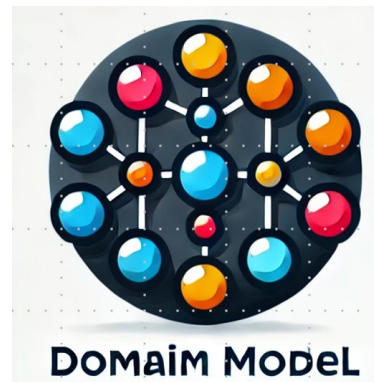
Il y a plusieurs réponses possibles, l'intérêt est le questionnement!!



# Relations entre Classes

UML définit quatre principaux types de relations entre classes

- Association
  - Association simple
  - Agrégation
  - Composition
- Généralisation (Héritage)



- Réalisation
- Dépendance

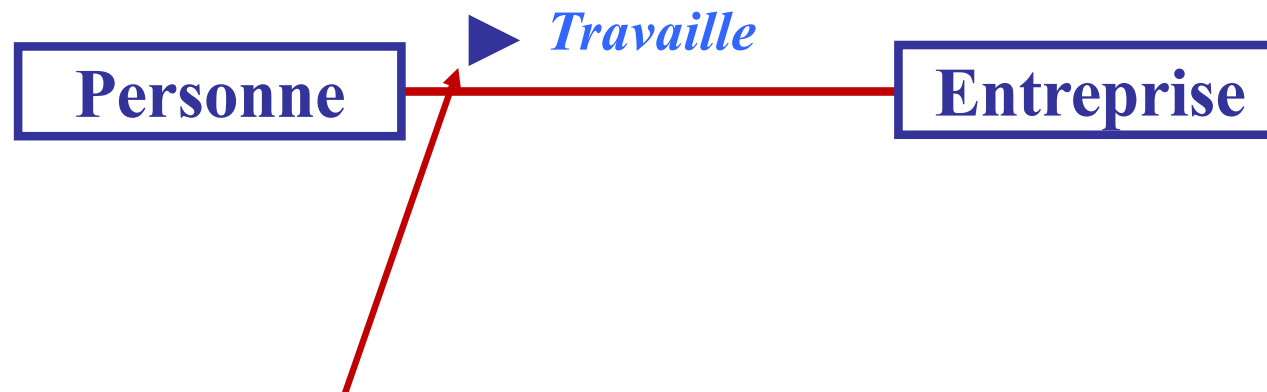
*Concepts de niveau  
analyse/conception*



# Associations

—  
ligne pleine entre classes

## Exemple d'association binaire



Précise le sens de la lecture (optionnel)



# Nommage des Associations

- Nom de l'association en italique au milieu de la ligne
- Forme verbale active ou passive

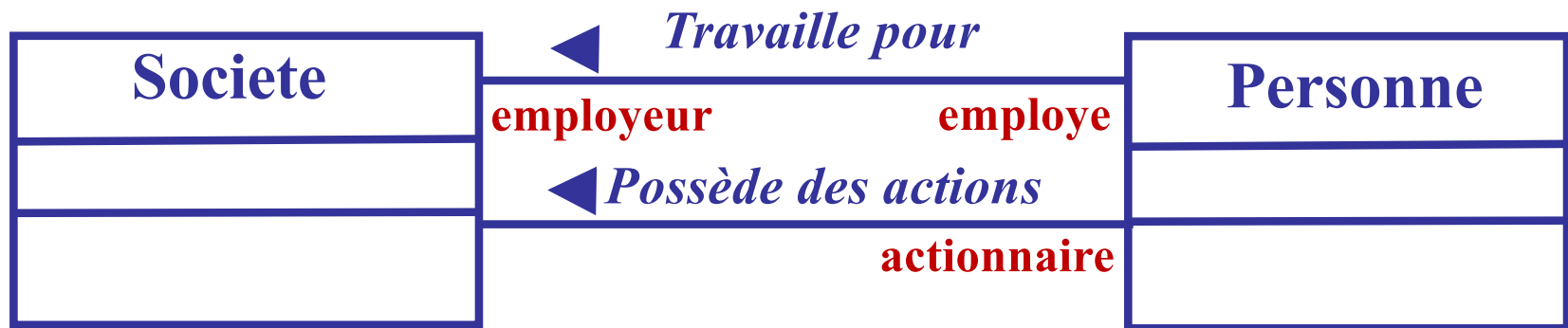






# Nommage des rôles

- Un **rôle** définit la manière dont une classe intervient dans une association

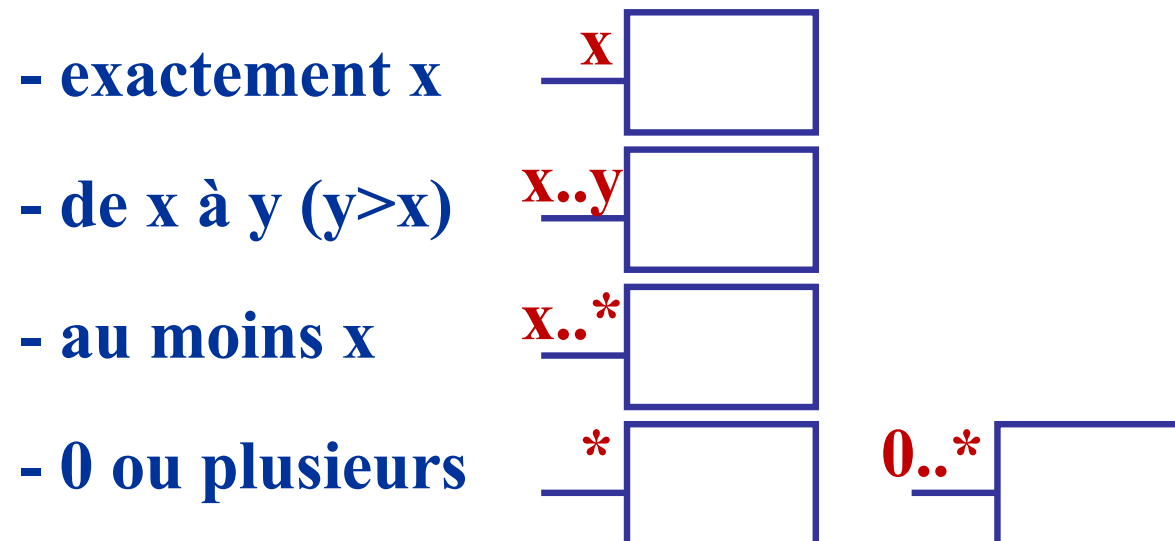


- Le rôle est indispensable lorsqu'il y a plusieurs associations entre 2 classes



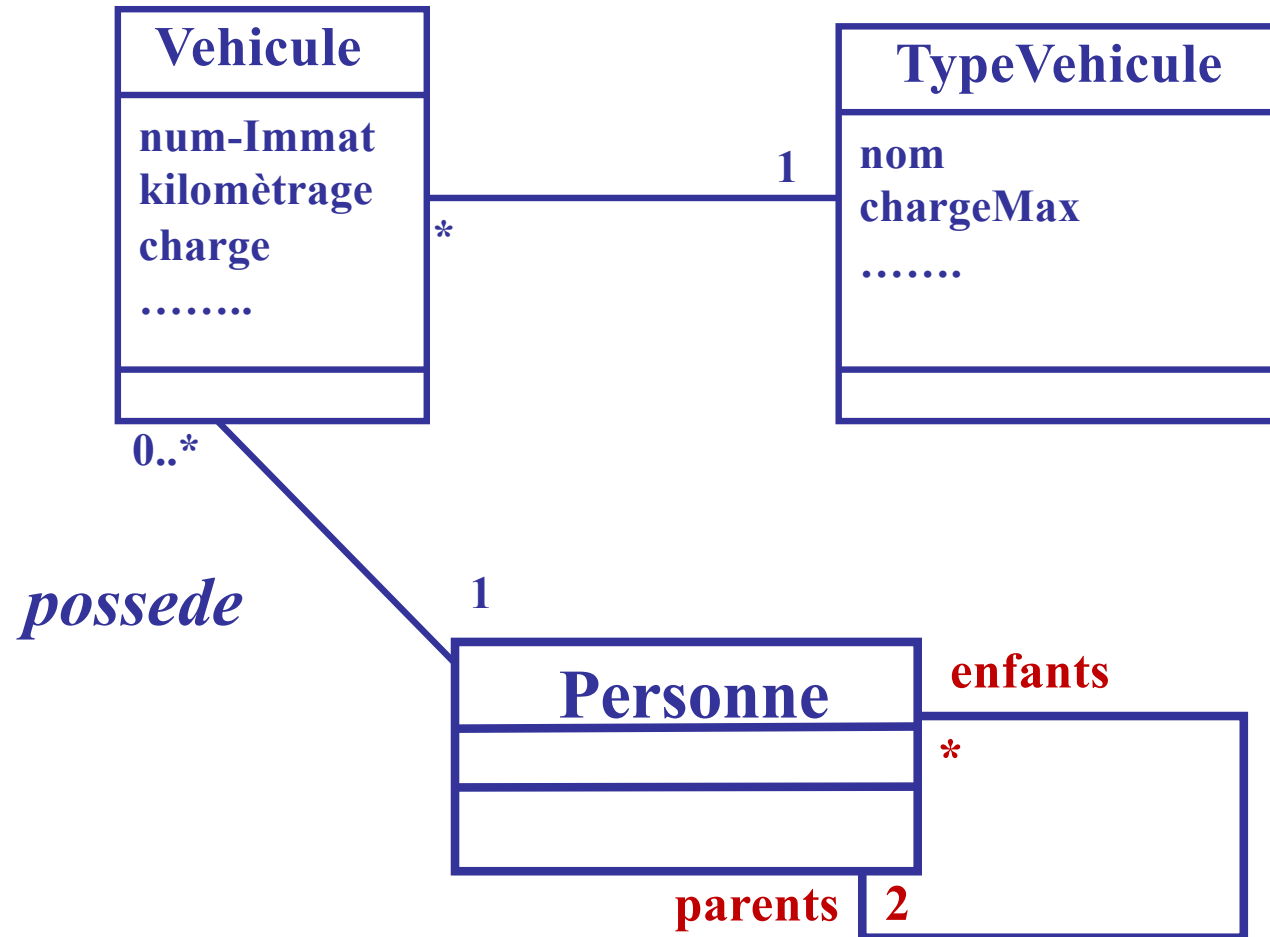
# Cardinalité des associations

- La cardinalité (ou multiplicité) d'une association précise le nombre d'instances qui participent à une relation





# Cardinalité des associations



**Association reflexive**



# Traduction des associations

Les associations sont traduites par l'ajout d'attributs dans les classes lors de la programmation

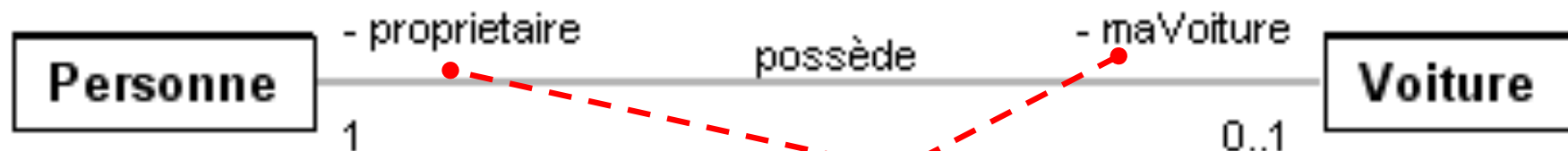
- La multiplicité définit le **style d'attribut**:
  - multiplicité 0 ou 1: références ou pointeurs
  - multiplicité  $> 1$  : listes, ensembles, ...
- La multiplicité définit le **caractère obligatoire ou facultatif** de l'attribut:
  - multiplicité 0 .. : les opérations devront tester la présence de la relation avant de l'utiliser
  - multiplicité  $> 0$ : l'attribut correspondant aura une (ou plusieurs) valeur(s) obligatoire(s).



# Traduction des associations

## Multiplicité $\leq 1$

- Les attributs ajoutés pour représenter l'association sont du type des classes associées.



```
public class Voiture {  
    private Personne proprietaire ;  
}
```

```
public class Personne {  
    private Voiture maVoiture;  
}
```

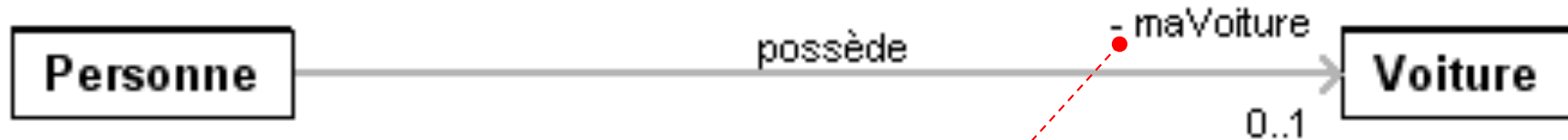




# Traduction des associations

## Multiplicité $\leq 1$

- Association **unidirectionnelle**:  
L'attribut ne sera ajouté que dans une seule classe.



```
public class Personne {  
  
    private Voiture maVoiture;  
  
    ...  
}
```





# Associations entre classes

## Multiplicité >1

- L'attribut ajouté pour représenter l'association est de type tableau ou collection.



```
public class Entreprise {  
    private Collection<Personne> employees;  
    ...  
}
```

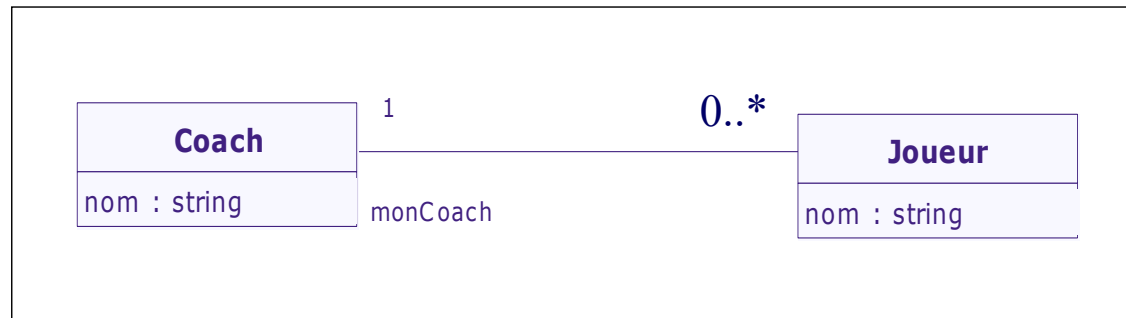




## Exercice 3 : Traduction des associations

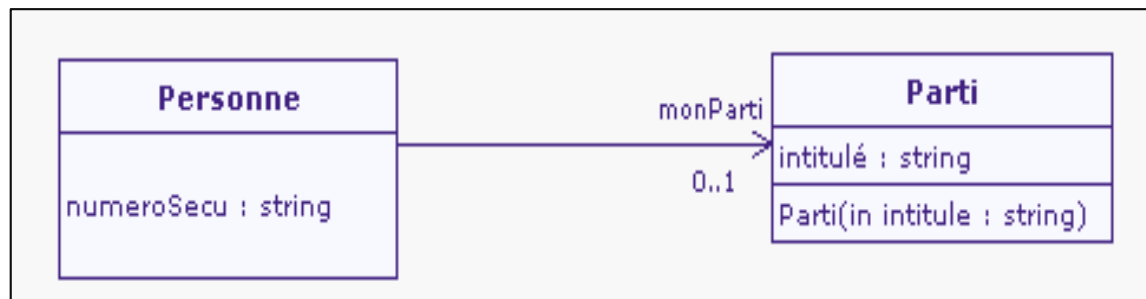


- Pour chacun des diagrammes suivants indiquer les attributs à ajouter dans les classes lors de la phase d'implémentation



```
Classe Coach {  
    String nom;  
    ...  
}
```

```
Classe Joueur {  
    String nom;  
    ...  
}
```



```
Classe Personne {  
    String numeroSecu;  
    ...  
}
```

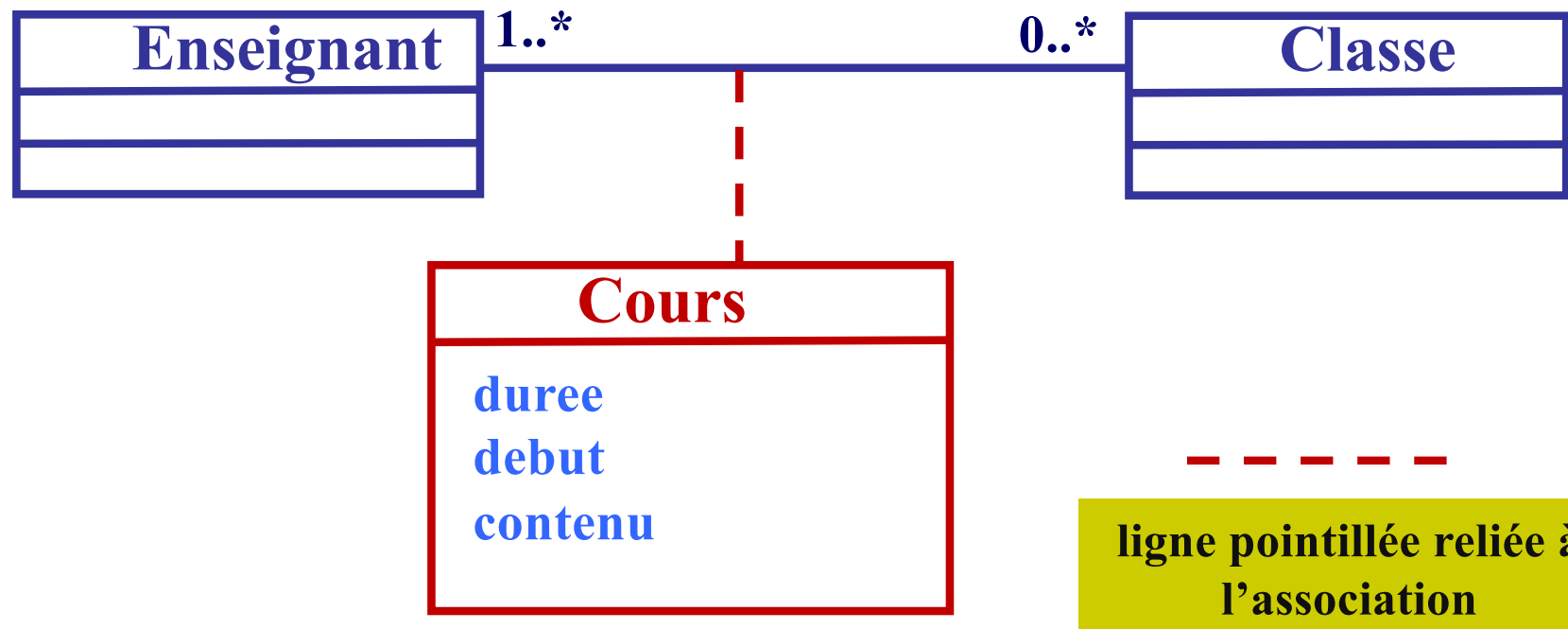
```
Classe Parti {  
    String intitule;  
    ...  
}
```





# Classe-Association

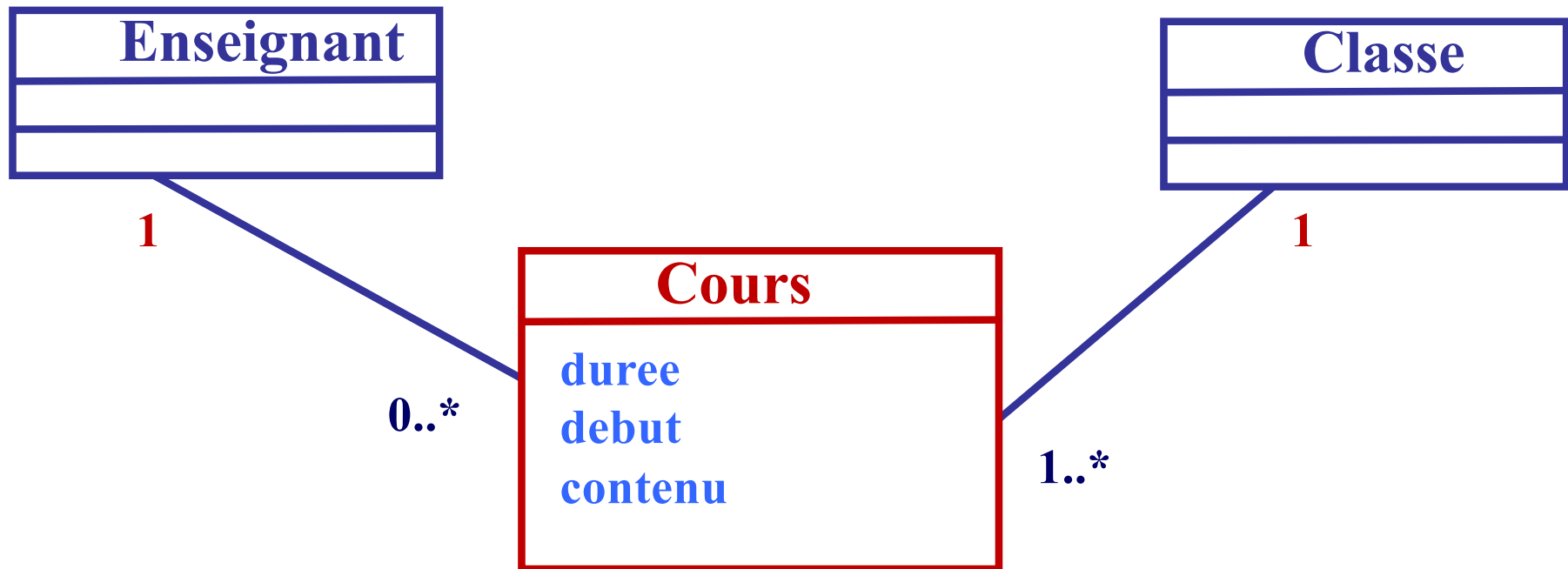
- Une association porteuse d'attributs est représentée par une **classe associative** ou **classe-association**.





# Classe-Association

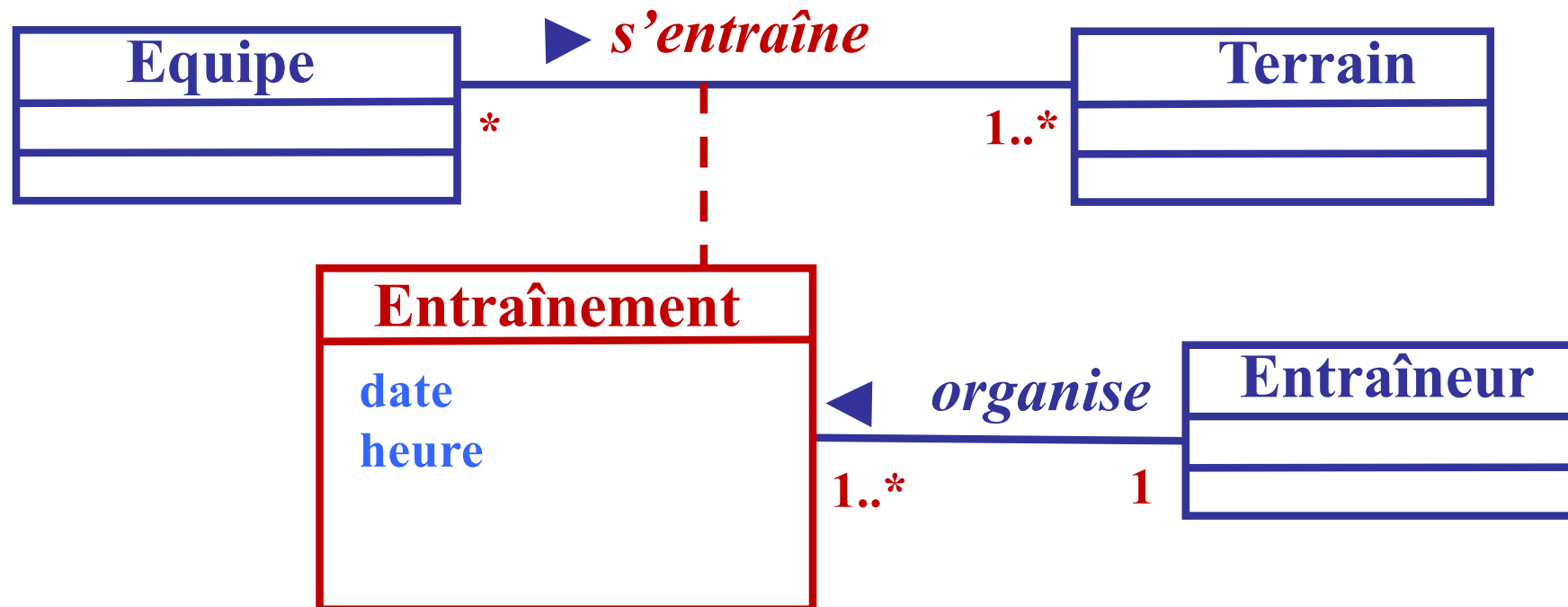
- Un diagramme de classe comportant une classe-association peut toujours être remplacé par un diagramme équivalent sans classe association:





## Exercice 4 : Classe-Association

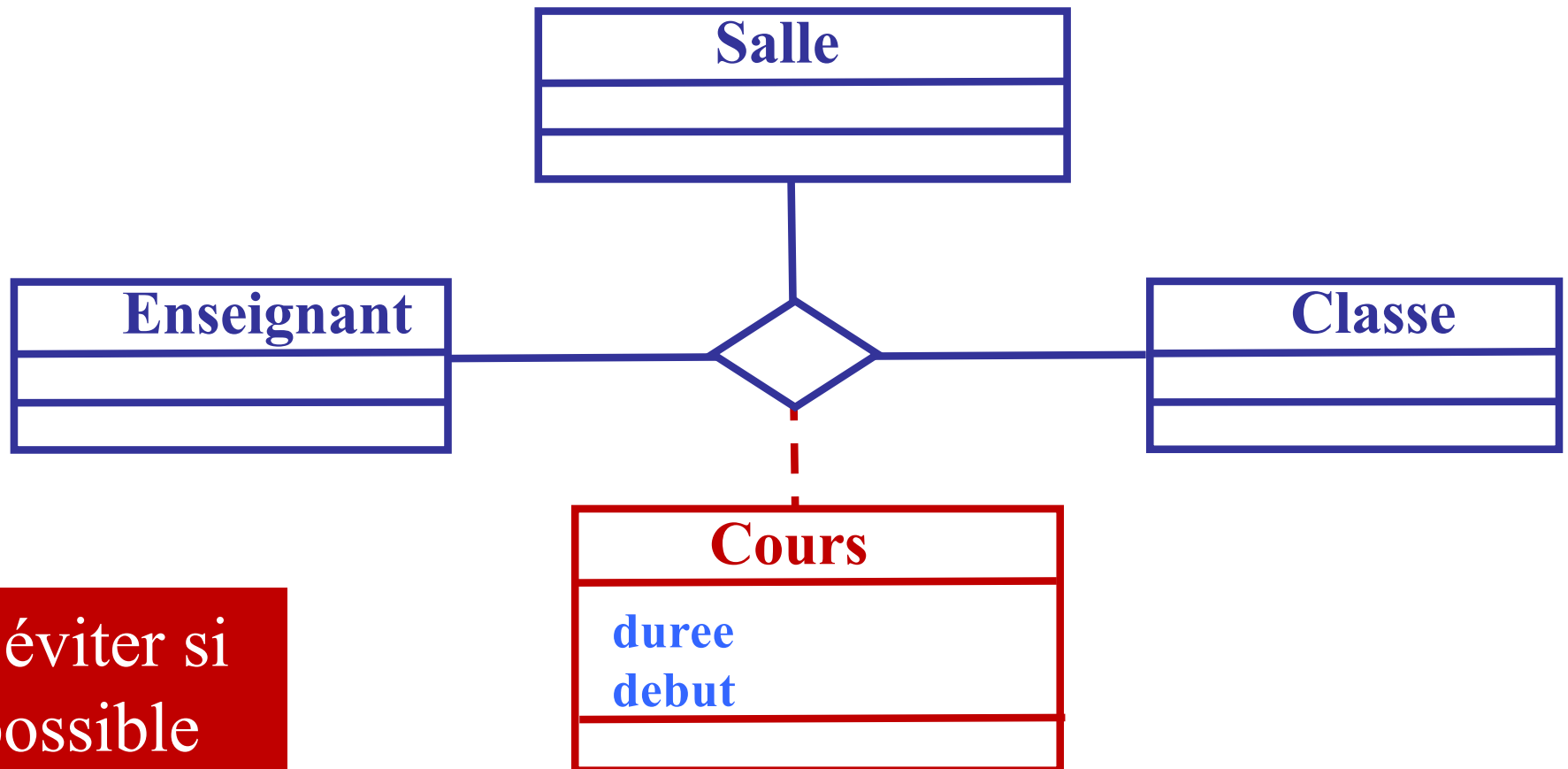
- Définissez un diagramme de classe équivalent au diagramme suivant mais ne comportant pas de classe association.





# Association n-aire

- Une association n-aire  implique au moins 3 classes

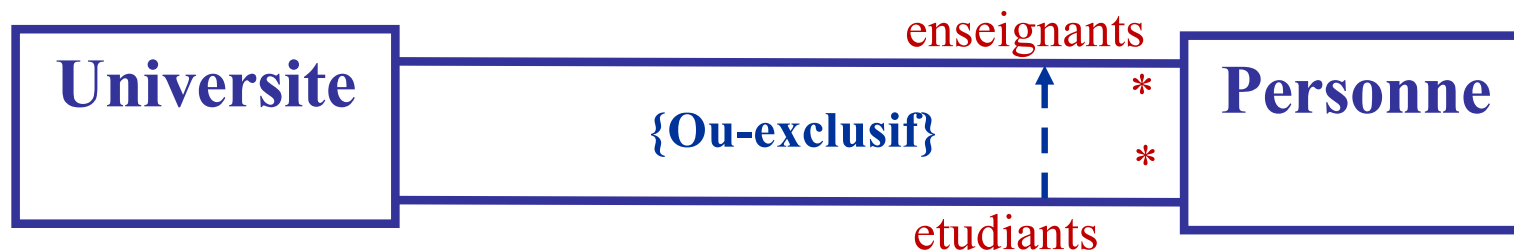
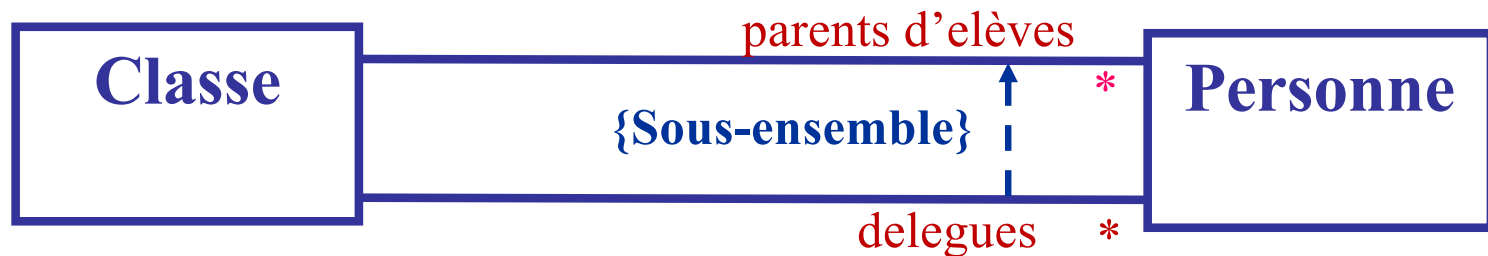


A éviter si possible



# Contraintes sur les associations

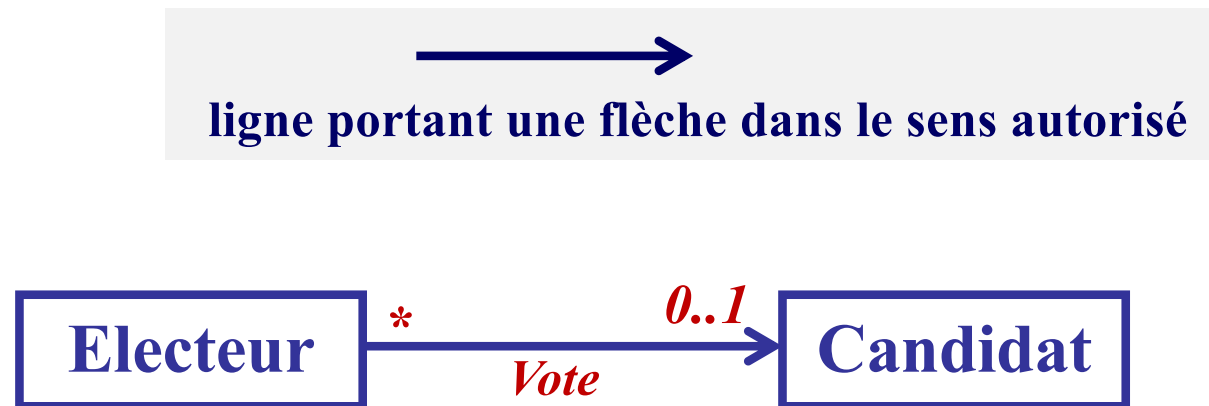
- Une contrainte porte sur une association ou sur un groupe d'associations **{Contrainte}**





# Navigabilité des Associations

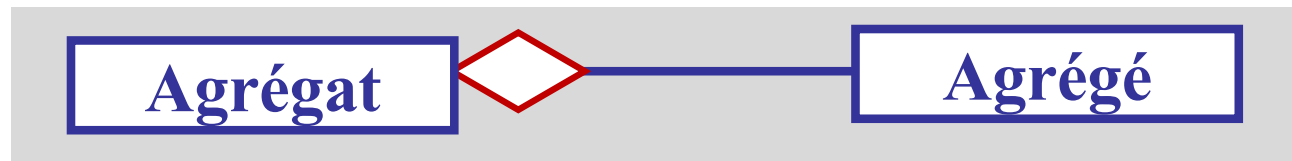
- Par défaut, une association est navigable dans les deux sens
- Lors de l'analyse, on peut exprimer le fait qu'un seul sens ne devra être implémenté :



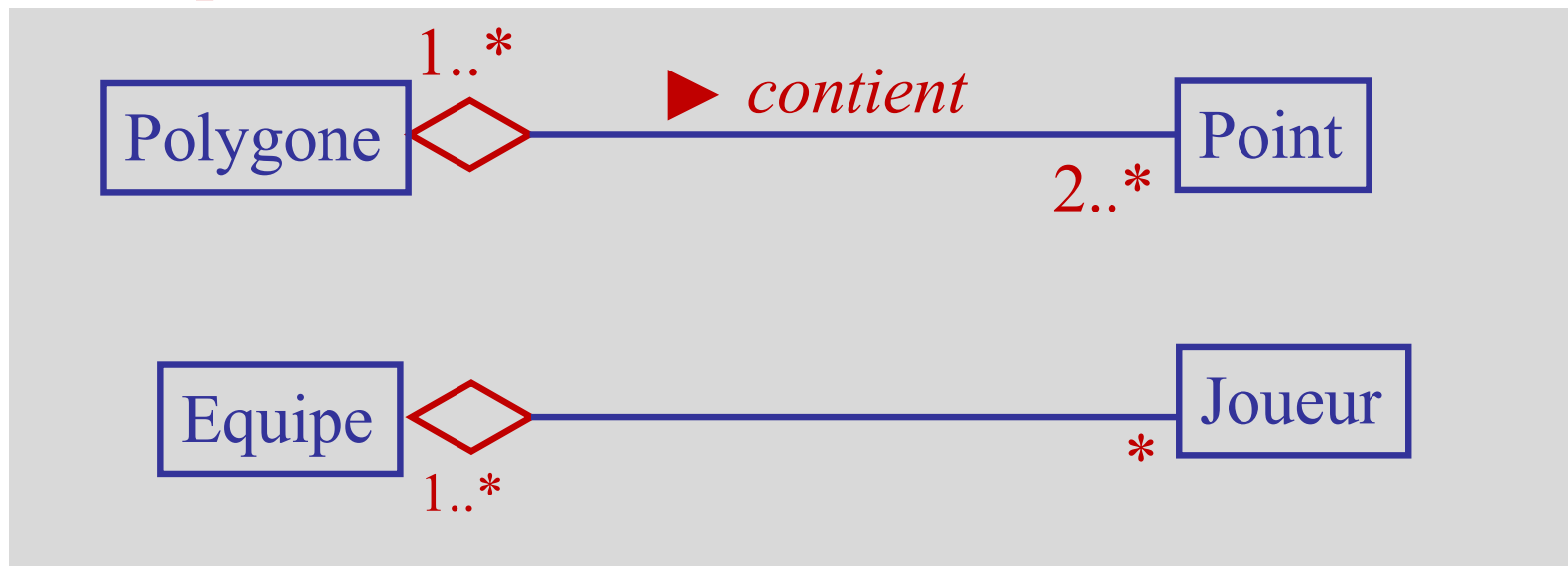


# Agrégation

- Une **agrégation** est une association non symétrique dont la sémantique évoque une relation de contenance



## Exemples



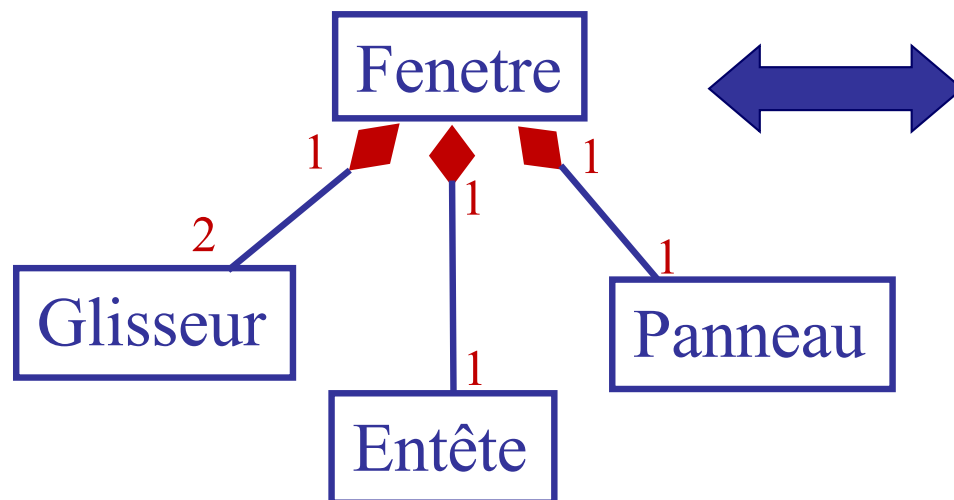


# Composition

- Une **composition** est une agrégation avec des contraintes fortes sur les cardinalités et les durées de vie composant/composé



## Exemple







## Exercice 5 : Association, Agrégation ou Composition



Pour chacune des associations suivantes, indiquez s'il s'agit d'une association simple, d'une agrégation ou d'une composition

- Une université emploie des enseignants
- Une personne possède une voiture
- Une maison comporte des pièces
- Un zoo contient des animaux
- Une voiture possède des roues et un châssis
- Une page web comporte des liens et des images
- Un livre comporte des pages



# Généralisation -Spécialisation

- Généralisation :  
relation « EST UN » ou « EST UNE SORTE DE »
- Factorisation des éléments communs d'un ensemble de classes
- **Super-classe** = abstraction de ses sous classes
- généralisation / spécialisation= deux points de vue antagonistes du concept de classification

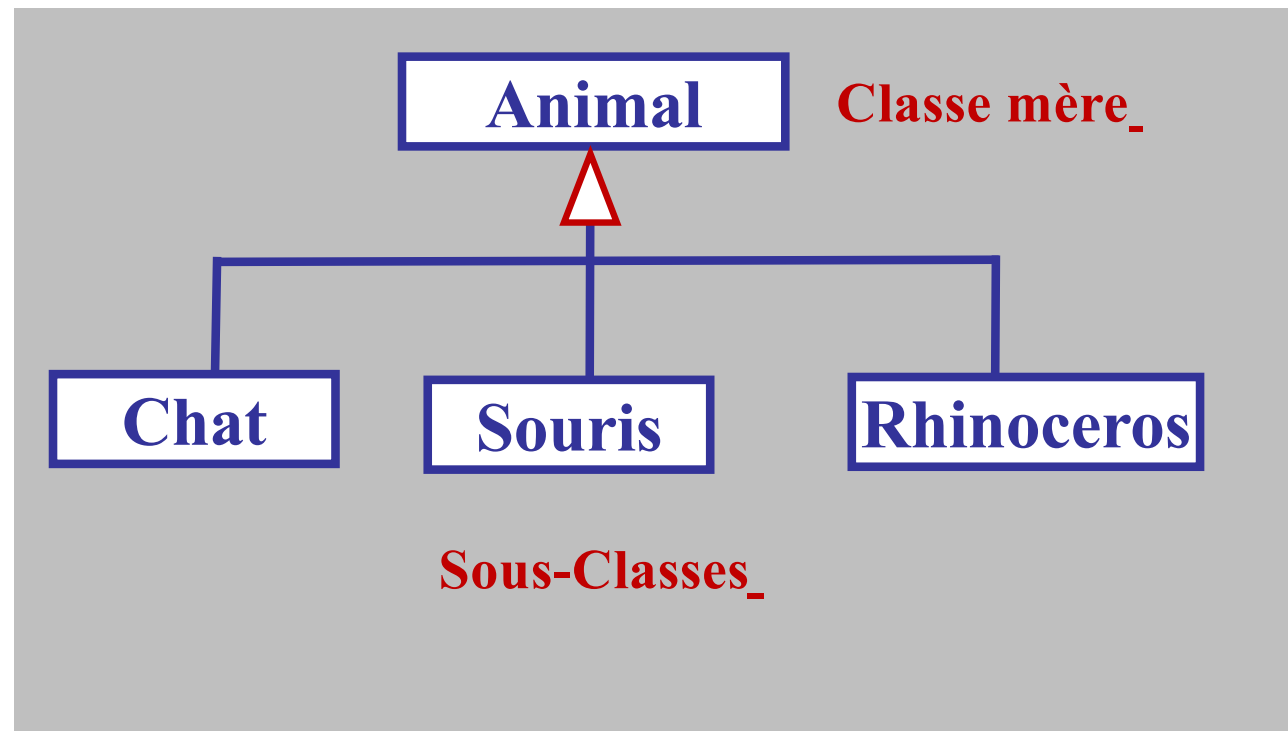


# Généralisation -Spécialisation

- Relation EST DE ou EST UNE SORTE DE  $\triangle$

## Notion d'héritage

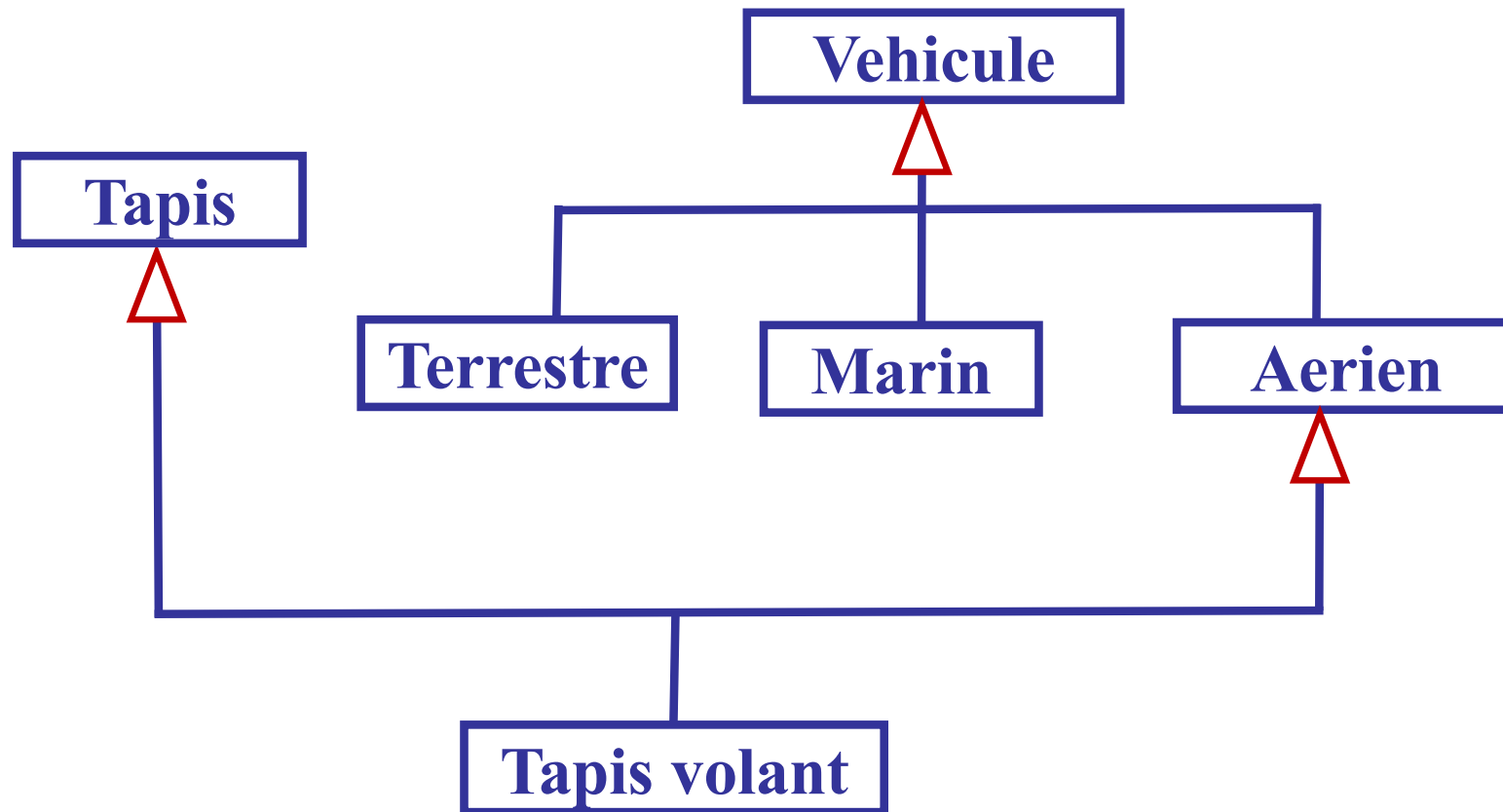
généralisation  $\uparrow$   
spécialisation  $\downarrow$





# Généralisation -Spécialisation

## ■ Notion d'héritage multiple





## Exercice 6 : Instanciation ou spécialisation?

Pour chacune des phrases suivantes, indiquez si la relation décrite est une instanciation ou une spécialisation :

- une toyota est une voiture
- un appartement est une habitation
- Lady est un chien
- un singe est un animal
- Ajaccio est une ville

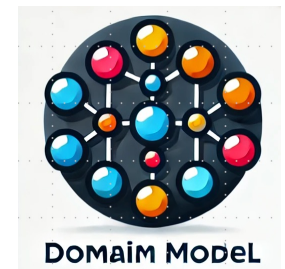




# Dépendances

*Identifiée dans le  
modèle du domaine*

- **L'association** représente une connexion structurelle (représentée par des **attributs** références) entre les objets des classes associées.



*Identifiée dans le  
modèle d'analyse*

- La **dépendance** est une relation d'usage. Elle traduit l'utilisation temporaire (dans une méthode) d'objets de la classe dont on dépend.





# Dépendances



- variable locale
- paramètre de méthode
- résultat de méthode



Attention : pas dans le modèle du domaine  
Modèle d'Analyse



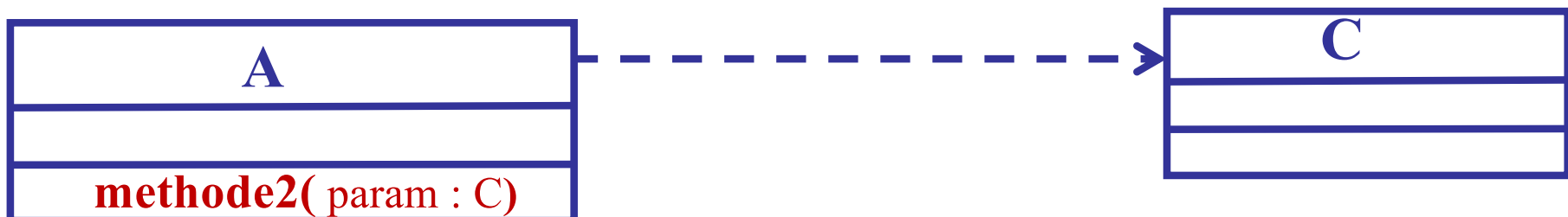
# Dépendances



- Création d'un objet d'une autre classe en tant que variable locale d'une méthode.



- Objet d'une autre classe comme paramètre de méthode.







# Dépendances



- Objet d'une autre classe comme résultat de méthode.





# Dépendances (Résumé)

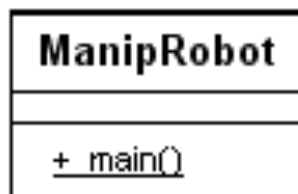


- Une dépendance est une relation non structurée entre classes (communication momentanée, limitée dans le temps):



*Au moins une méthode de A :*

- contient une **variable locale** de type B
- possède un **paramètre** de type B
- renvoie un **résultat** de type B



```
public class ManipRobot {  
    public static void main(String[] args) {  
        Robot r=new Robot("Toto",10,20,Robot.NORD);  
        r.changerOrientation(Robot.SUD);  
        r.déplacer();  
    }  
}
```





# Classes Abstraites et Interfaces

## Classe abstraite

- Une classe abstraite est une classe qui n'a pas d'instances directes.

**Nom en italique**

*Nom de la classe abstraite*  
**{abstraite}**

**Propriété {abstraite=vrai}**





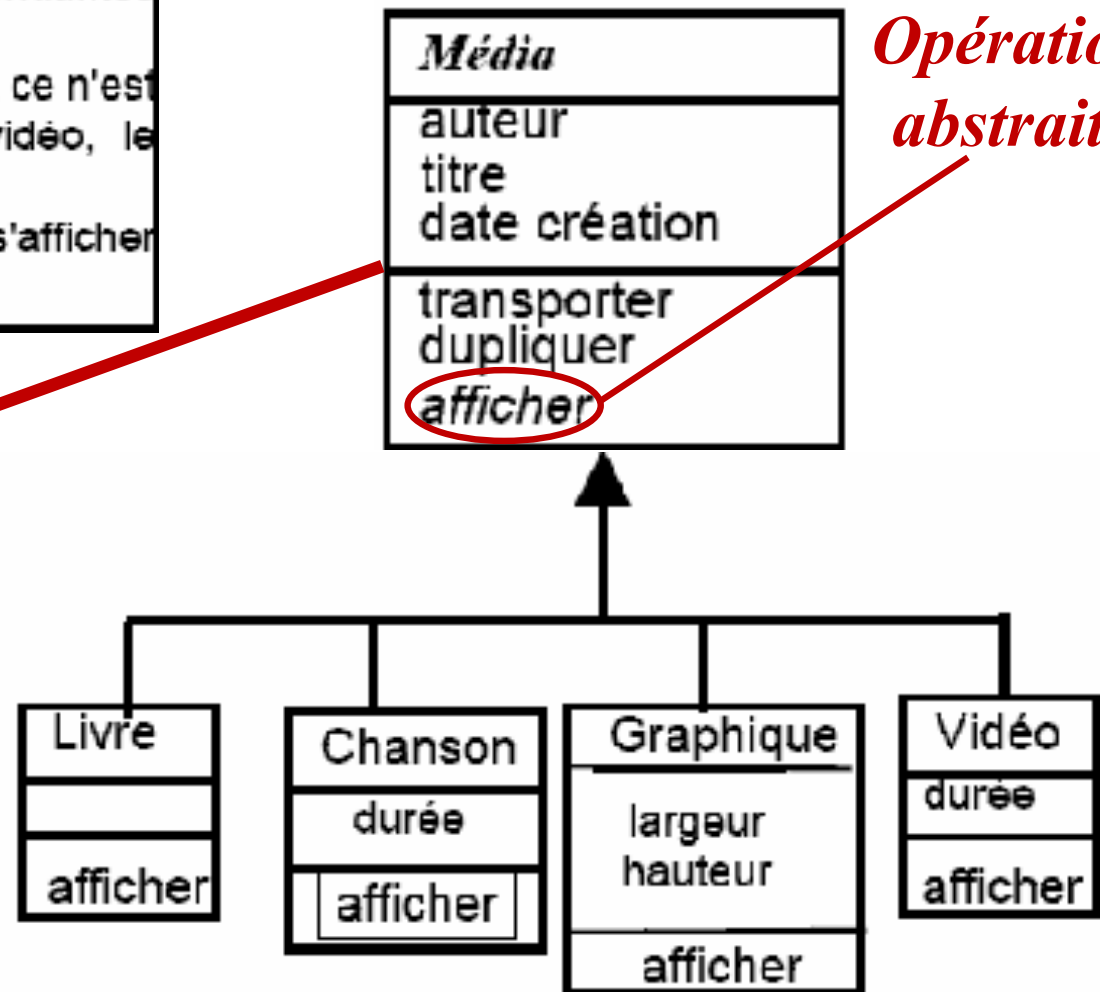
# Classes abstraites



Un média peut être transporté, dupliqué, affiché.  
Le transport et la duplication sont indépendantes  
du type du média (copie de fichiers).  
Par contre, tout média peut être affiché et ce n'est  
pas la même chose pour l'audio, la vidéo, le  
graphisme, le texte.  
Un média ne peut pas définir comment s'afficher  
tant qu'il ne sait pas ce qu'il est.

*Opération  
abstraite*

Il n'y a pas d'instance de la  
classe média.  
Un média existe en tant  
que livre, chanson,  
graphique, vidéo.





# Interfaces



- Une interface est une classe qui ne possède que des opérations abstraites



- ✓ Description du comportement visible d'une classe, d'un composant: *liste d'opérations publiques (services de l'interface)*
- ✓ Généralisation possible entre les interfaces



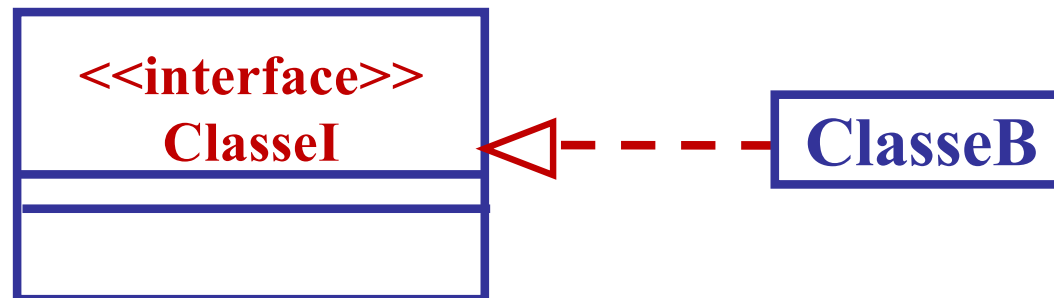
# Interfaces



Les classes peuvent être reliées aux classes interfaces par deux sortes de relations :

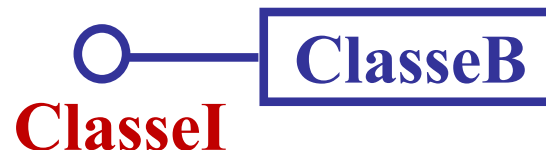
- relation de **réalisation** (implémentation)

Une classe B réalise (ou implémente) une classe I interface si elle fournit un ensemble de méthodes qui implémentent les opérations de l'interface.



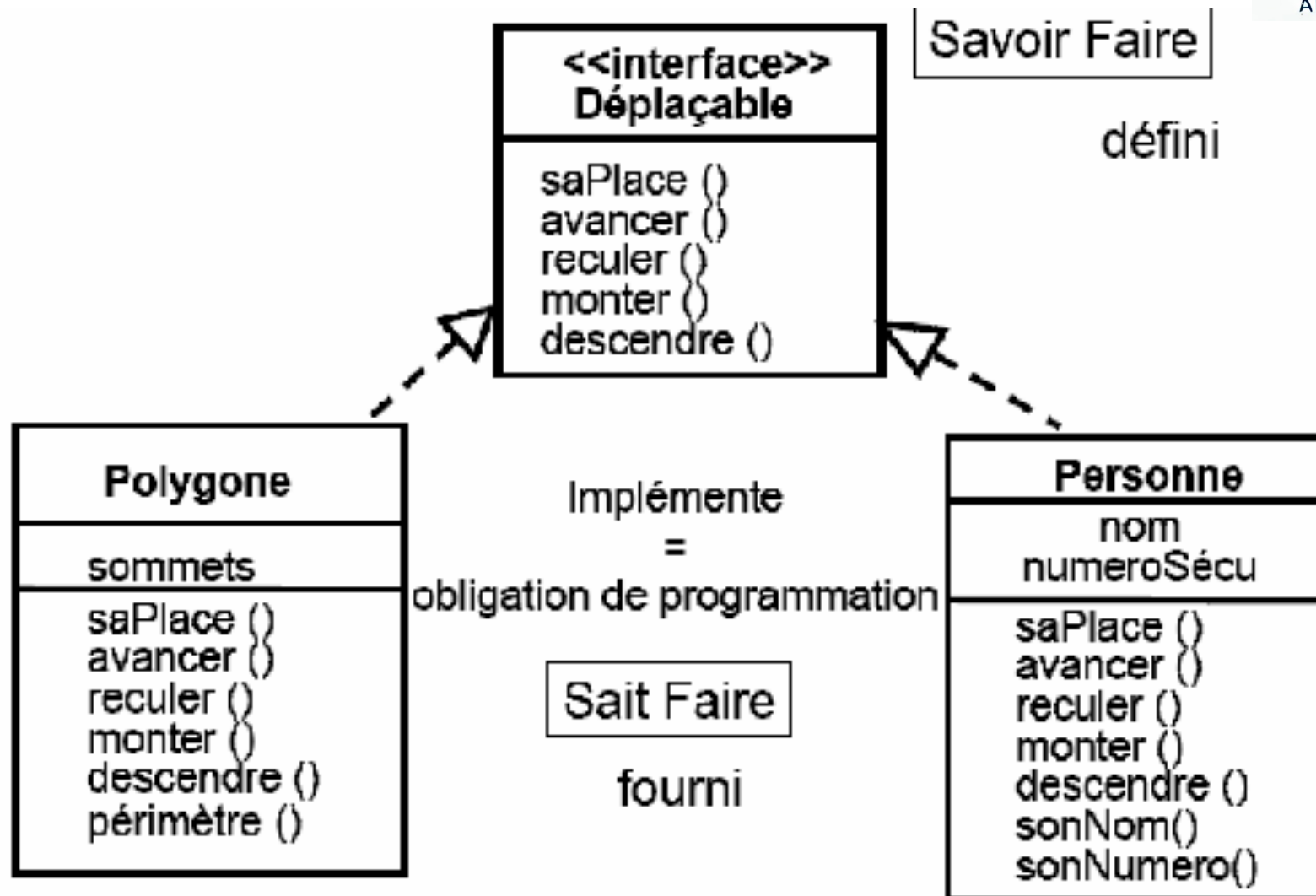
Autre représentation

(*lollipops*)





# Interfaces



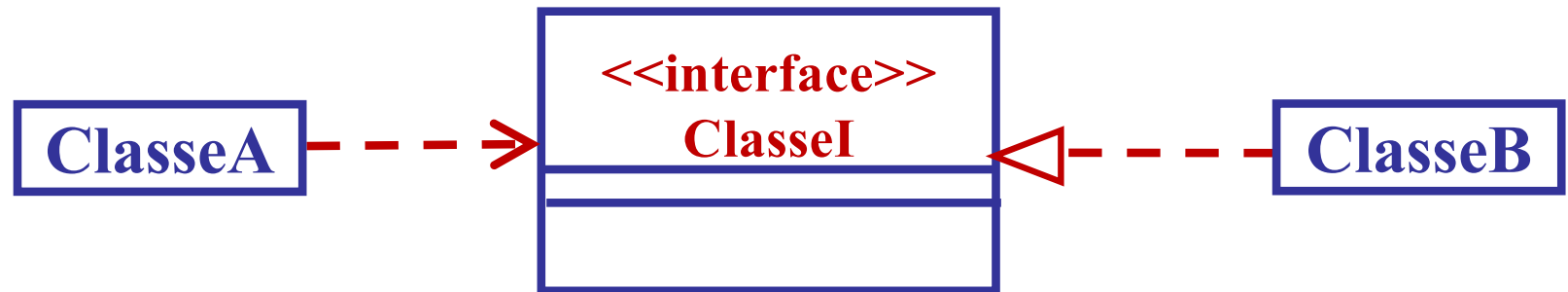


# Interfaces



## Relation de dépendance (utilisation)

- Une classe A requiert une interface I si elle a besoin (ou utilise) d'une instance d'une classe qui implémente cette interface pour travailler.



Autre représentation (*lollipops*)



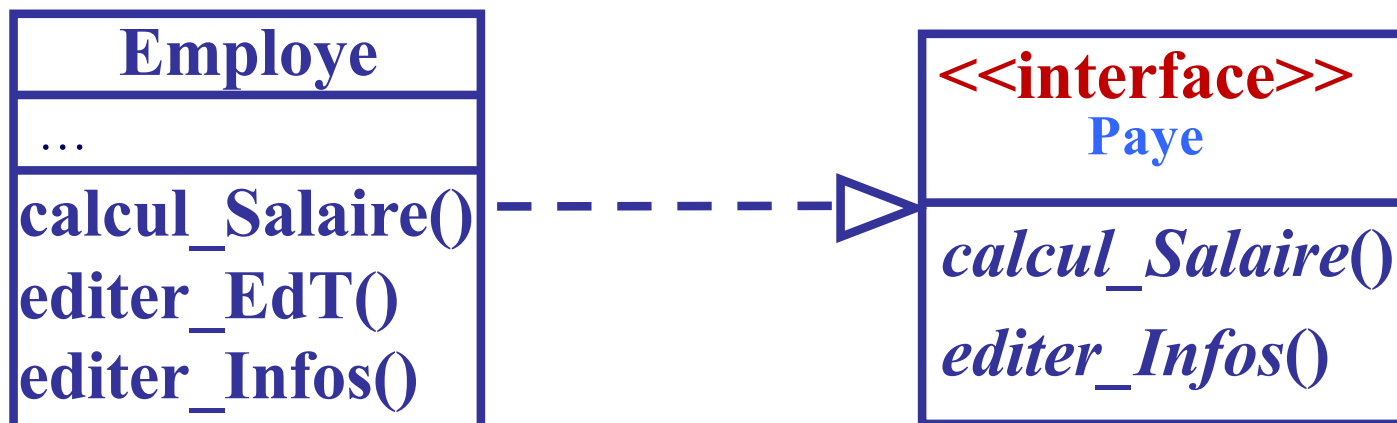
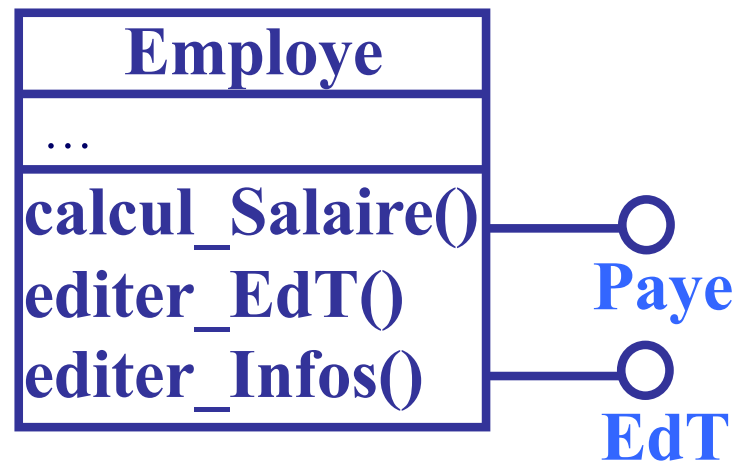




# Interfaces

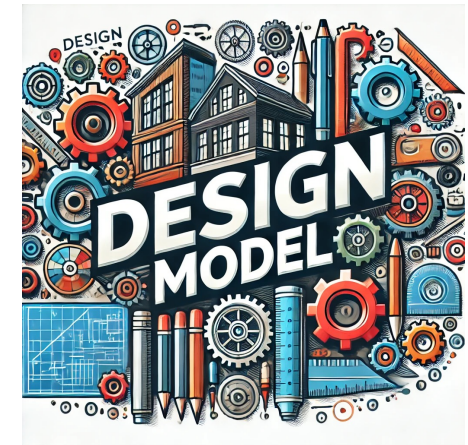


## Exemple



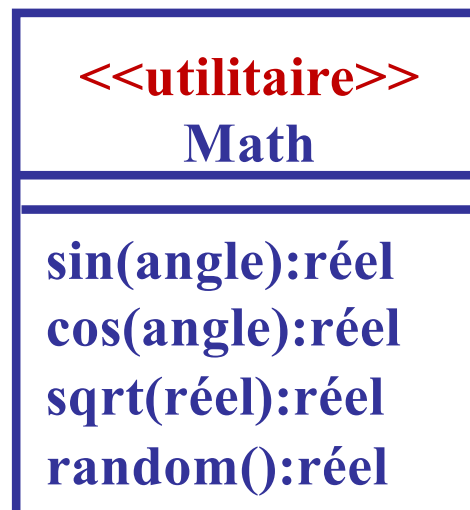


# Autres Concepts (Conception)



- Classes utilitaires
  - Regroupement de variables et opérations de classes
  - Une classe utilitaire ne peut pas être instanciée

## Exemple

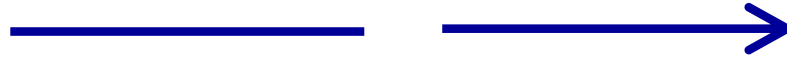




# Formalisme du Diagramme de classes

## Relations entre classes (résumé)

**Association**



**Agrégation**



**Composition**



**Généralisation (héritage)**



**Implémentation (entre une  
classe et une interface)**



**Dépendance (utilisation)**





# CH2 – MODELE DU DOMAINE

## 2.1 – Présentation

## 2.2 – Formalisme des Diagrammes de classe

## 2.3 – Diagrammes d'objets

- Définitions et objectifs
- Représentation d'un objet
- Instances de relations
- Objets composites



## 2.4 – Démarche de construction



# Diagrammes d'Objets

## Objectifs

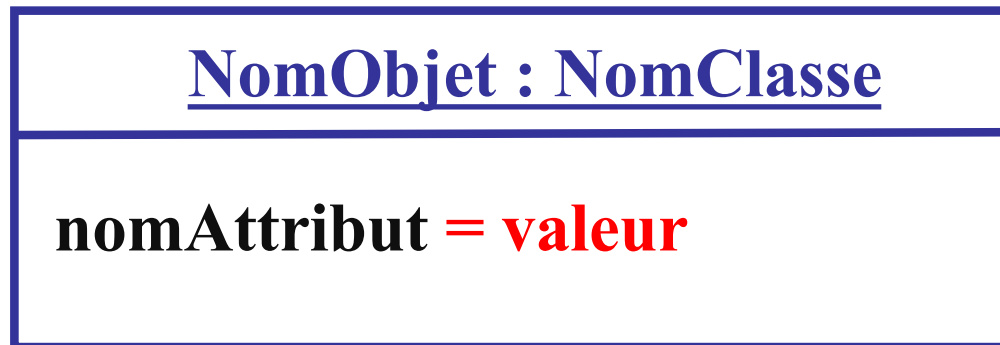
Les diagrammes d'objets décrivent un ensemble particulier d'instances.

- Vérification de l'adéquation d'un diagramme de classes à différents cas possibles
- Explication de **cas particuliers**
- Raisonnement à partir d'**exemples**
- Compréhension de structures de données complexes



# Diagramme d'objets

- Un diagramme d'objet ne contient que deux types d'éléments:
  - des **objets** (instances de classe)
    - Identité (Nom d'objet et/ou Nom de classe)
    - Valeurs d'attribut (Attributs « intéressants » uniquement)



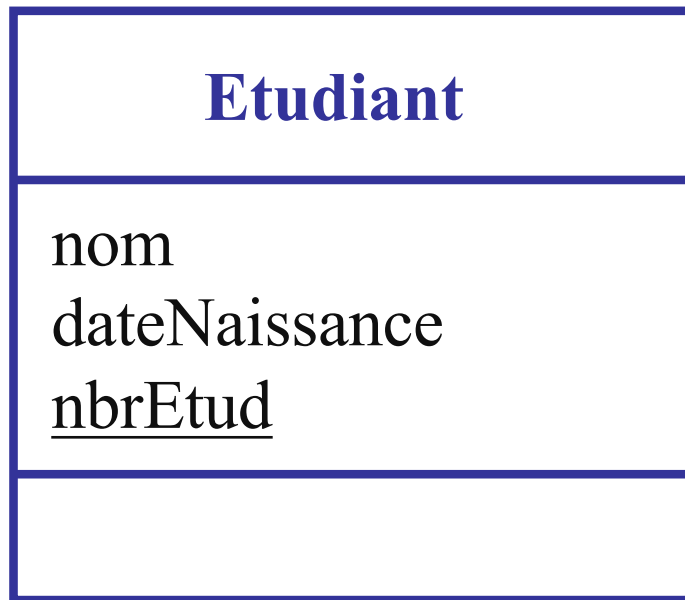
- des **liens** (instances de relation d'association, agrégation ou composition)



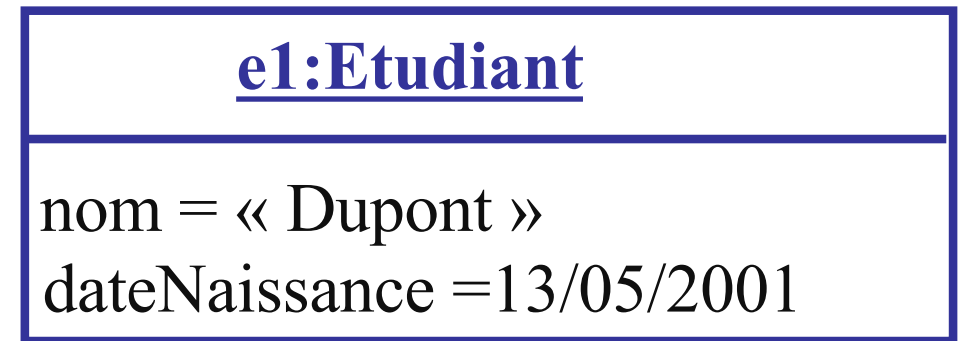


# Diagramme d'objets

## Exemple de classe



## Exemples d'objets



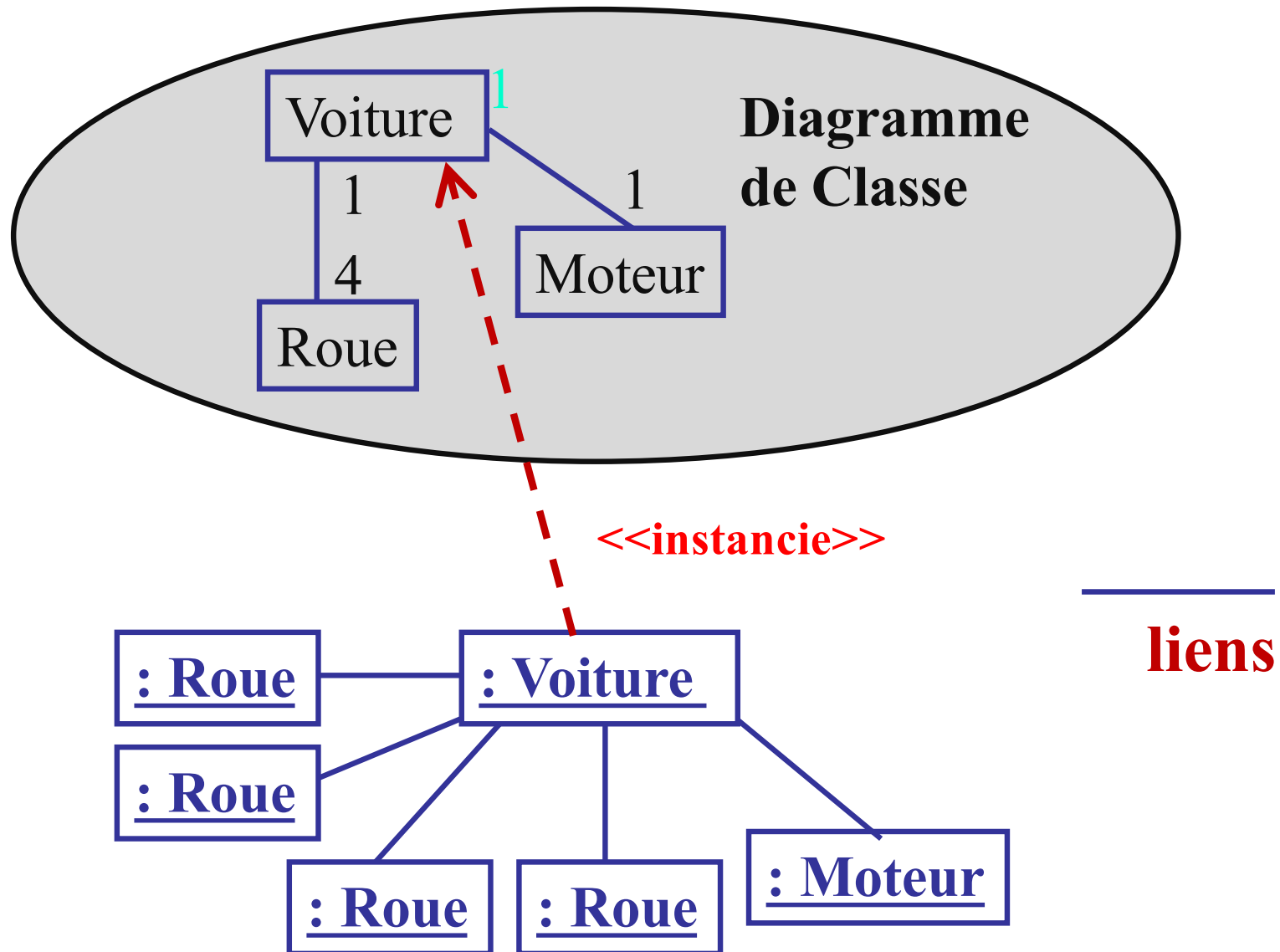
Instance e1 d'une  
classe quelconque



Instance anonyme de la classe Etudiant



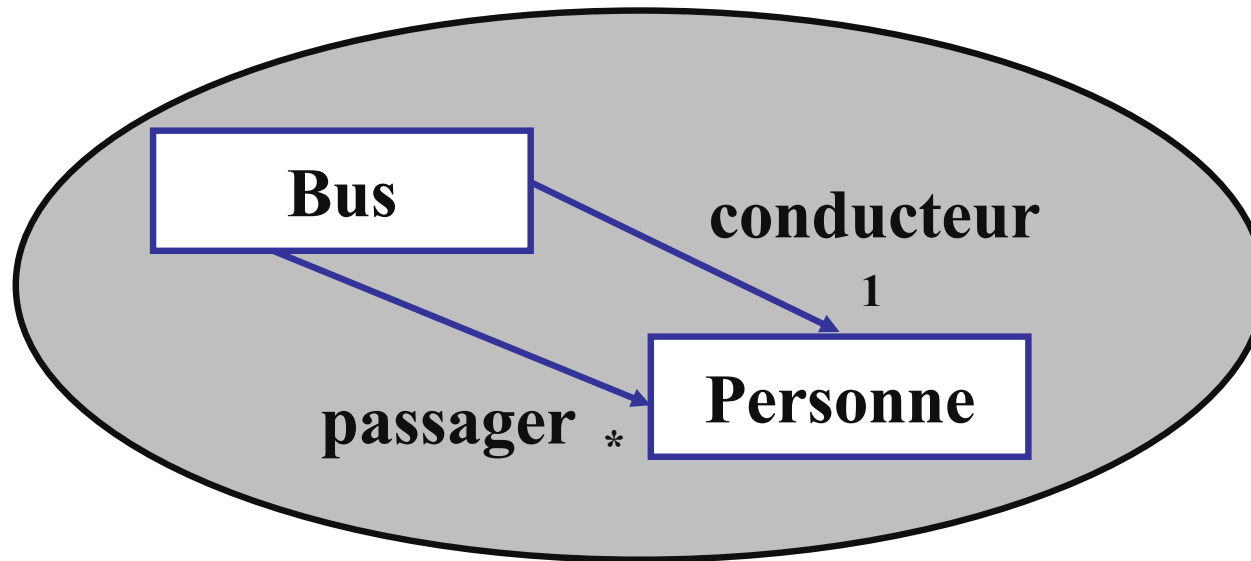
# Instances de Relations



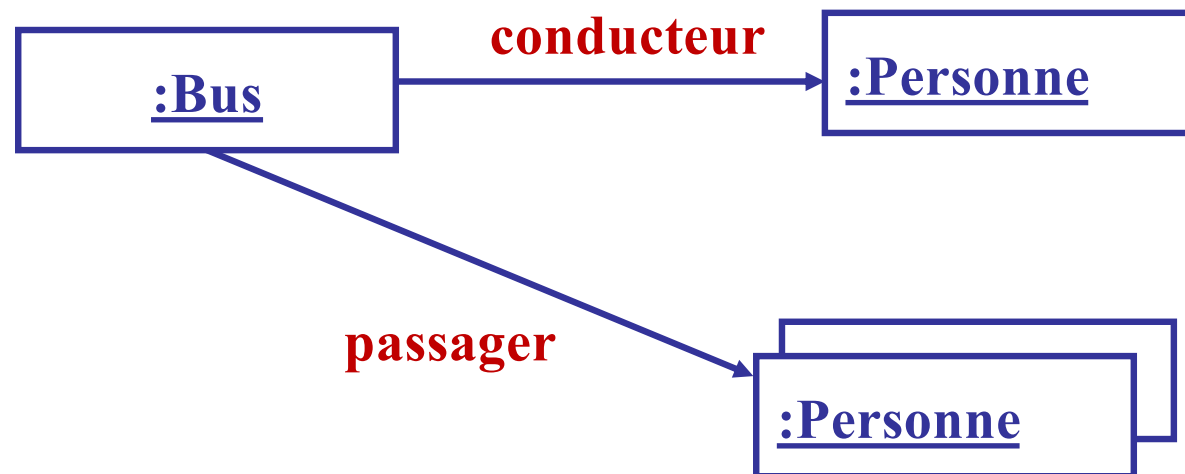




# Instances de Relations



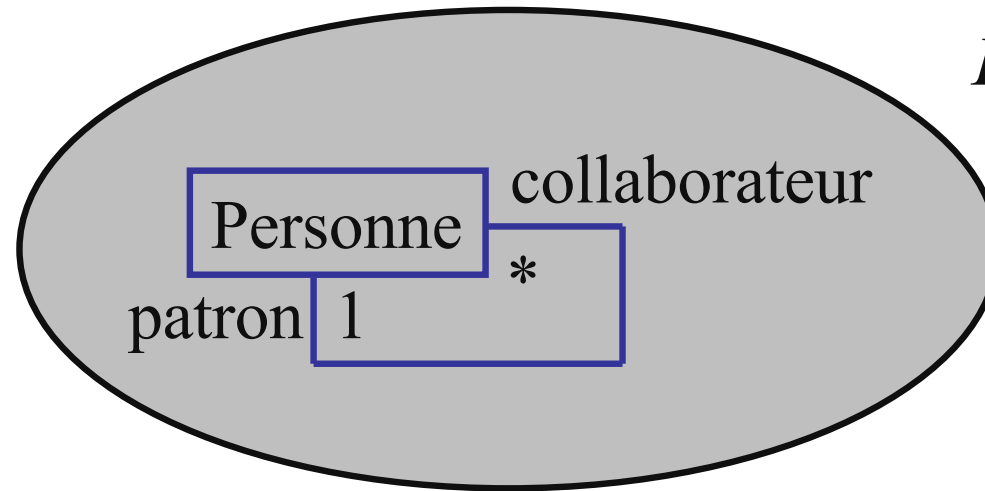
*Diagramme  
de Classe*



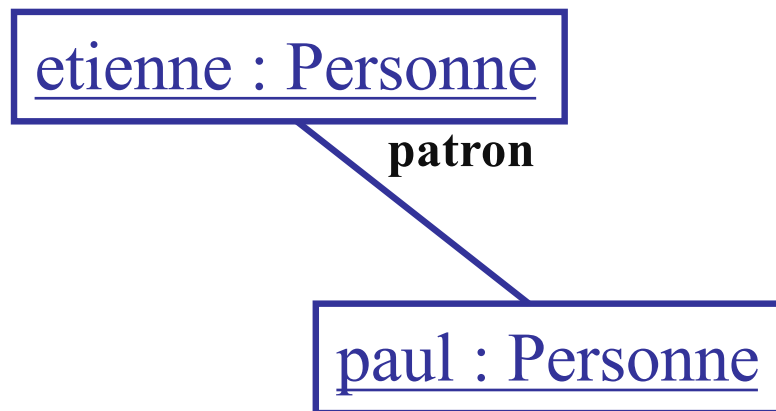


# Instances de Relations

*Diagramme  
de Classe*



Ce diagramme est-il  
valide?  
Qui est le patron  
d'Etienne?

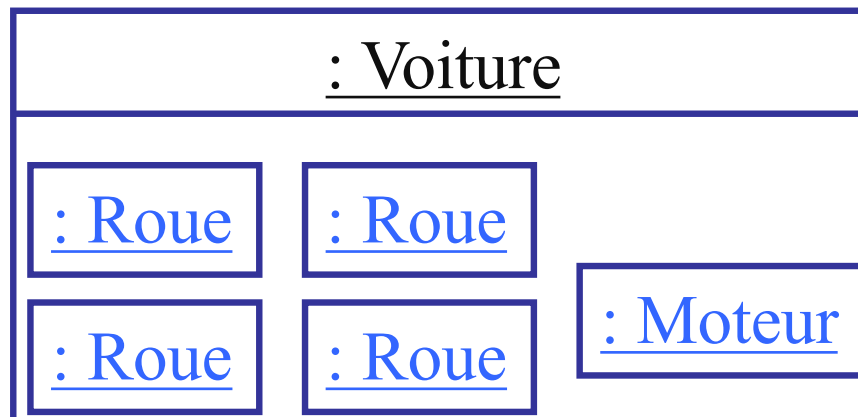
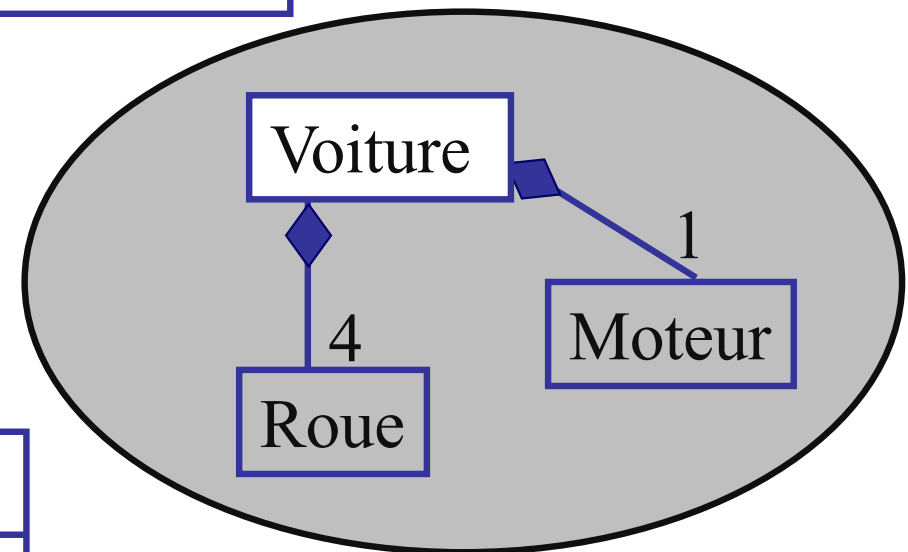
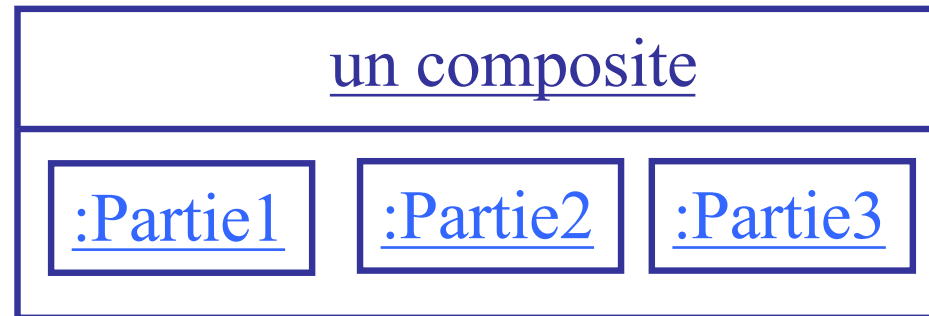


Jacques est son propre patron

Etienne est le patron de Paul



# Objets composites





# CH2 – MODELE DU DOMAINE

2.1 – Présentation

2.2 – Formalisme des Diagrammes de classe

2.3 – Diagrammes d'objets

→ 2.4 – Démarche de construction





## 2.4- Démarche de Construction

### Diagramme de classes du Domaine

1. Identifier les classes

*Concepts du domaine*

2. Identifier les relations

➤ Associations

➤ Agrégations et compositions

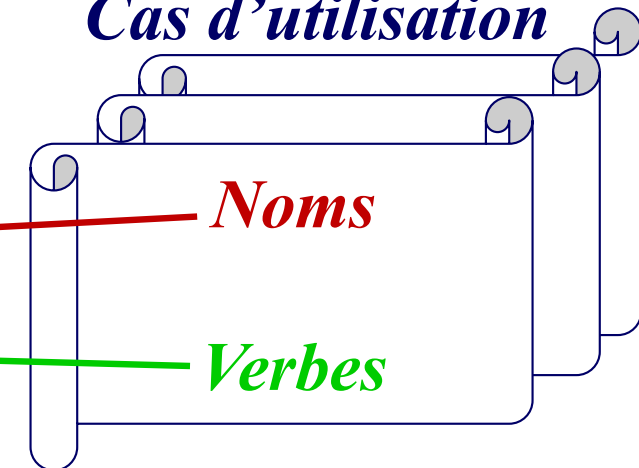
3. Placer les multiplicités

4. Placer les attributs (*noms*)

5. Identifier les relations de généralisation

6. Structurer en packages

*Cas d'utilisation*

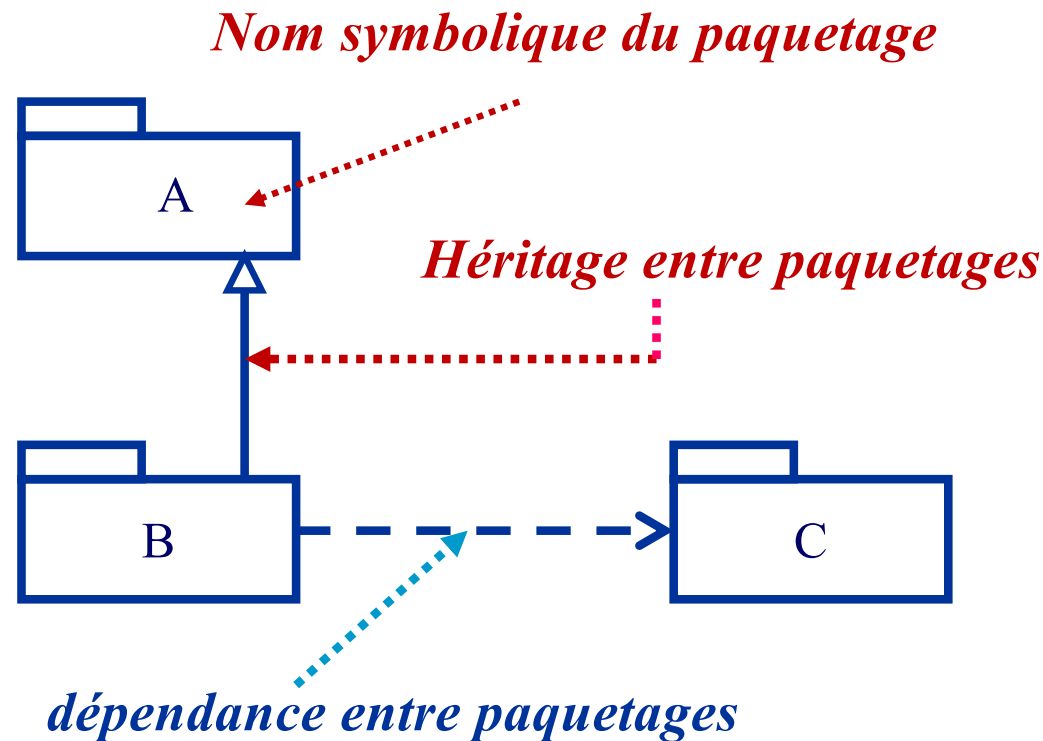


**Diagrammes  
d'objets**



# Démarche de Construction

## Structuration des classes en packages



*Il existe (au moins) un élément du paquetage source qui spécialise (au moins) un élément du paquetage destination*

*Il existe au moins un élément du paquetage source qui utilise les services d'au moins un élément du paquetage destination*



# Démarche de Construction

## Structuration des classes en packages

### Relations entre paquetages

Un paquetage ne peut pas voir le contenu d'un autre paquetage

- ➔ Utiliser la dépendance « accès » si nécessaire  
*Permet la référence des éléments du paquetage fournisseur par les éléments du paquetage client*
- ➔ Dépendance « importation » pour inclure l'élément dans un paquetage



# Démarche de Construction

