

# Modélisation – Partie II

---

Stéphanie CHOLLET

# Diagramme de classes et d'objets

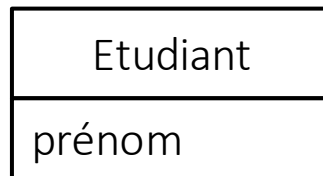
---

# Diagrammes de classes et d'objets – 1/4

## ► Diagramme de classes :

- Description du **cas général** au niveau modèle

Modèle :



- Permet de représenter les **aspects statiques et structurels** du système

## ► Diagramme d'objets :

- Description des **exemples** au niveau instance

Instances :

Pierre



Paul



e1:Etudiant

prénom=« Pierre »

e2:Etudiant

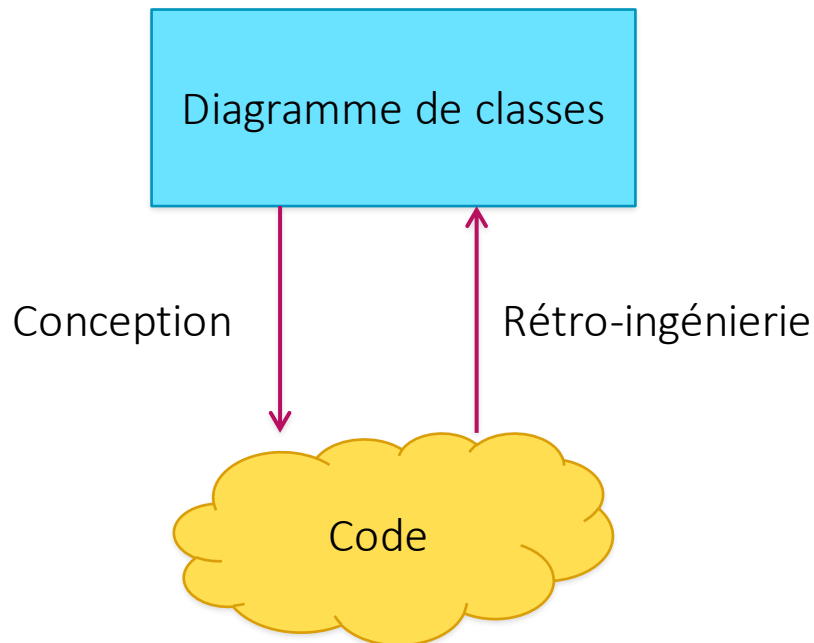
prénom=« Paul »

- Permet de représenter une **image** d'un système à un **instant donné**

# Diagrammes de classes et d'objets – 2/4

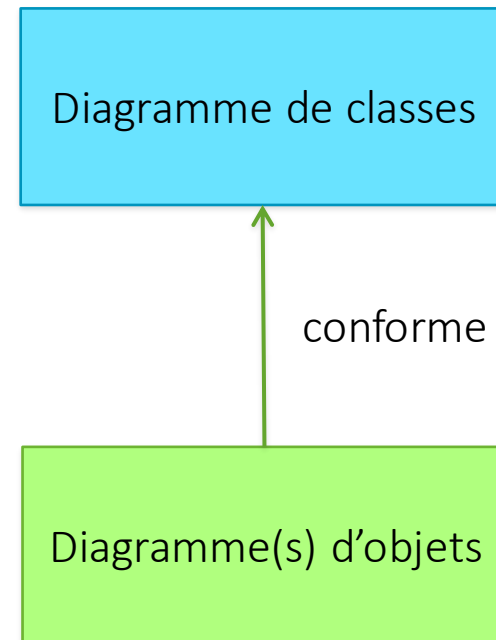
## ► Diagramme de classes :

- Utilisé pour la conception
- Utilisé pour la rétro-ingénierie
- Utilisé pour structurer le développement



## ► Diagramme d'objets :

- Utilisé pour vérifier un diagramme de classes



# Diagrammes de classes et d'objets – 3/4

Diagramme de classes

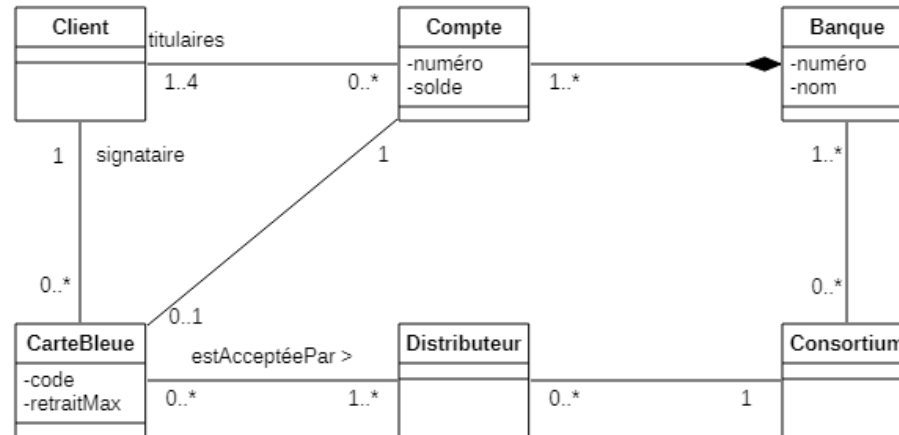
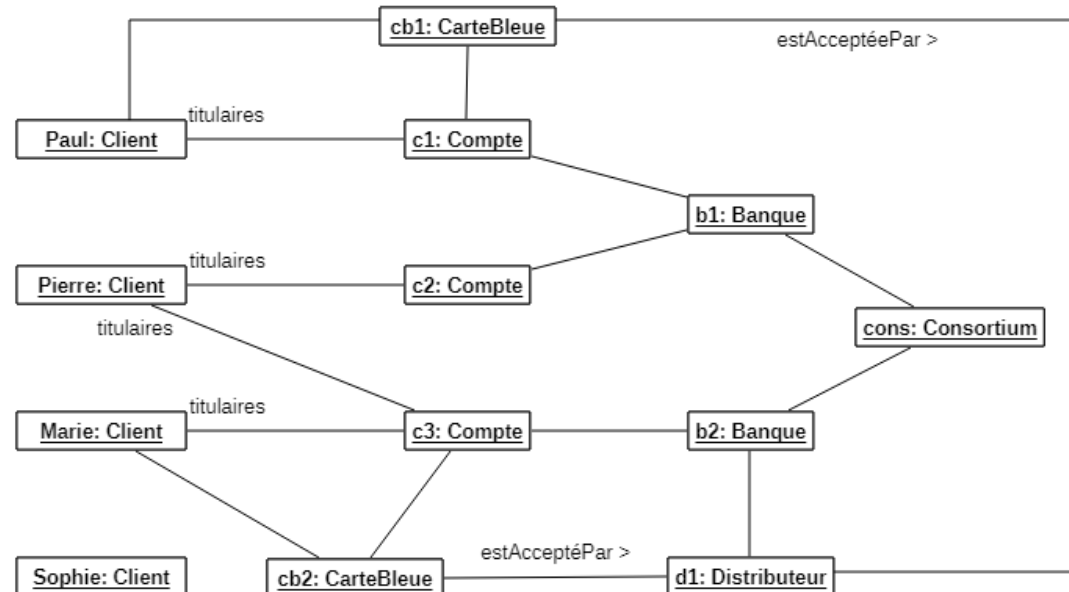


Diagramme d'objets



# Diagrammes de classes et d'objets – 3/4

---

- ▶ Éléments d'un diagramme de classes :
  - ▶ Des classes
  - ▶ Des associations
- ▶ Éléments d'un diagramme d'objets :
  - ▶ Des objets
  - ▶ Des liens
- ▶ Un diagramme de classes peut être :
  - ▶ Surchargé de contraintes OCL (*Object Constraint Language*)
- ▶ Un diagramme de classes ~~peut~~ doit être vérifié avec au moins un diagramme d'objets

# Diagramme de classes

---

Les classes

# Une classe

- Une classe décrit un ensemble d'objets qui partagent une structure et une sémantique communes

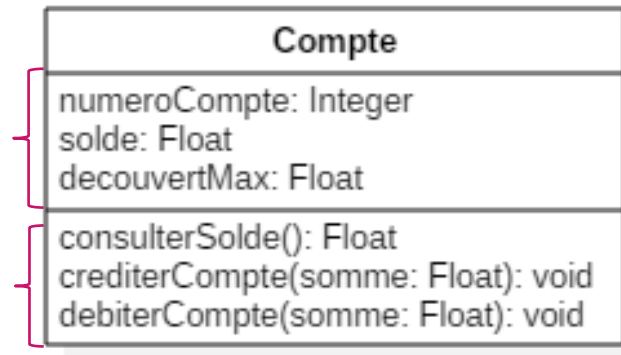
*Nom de la classe*

Attributs :

*nom: type*

Opérations :

*nom(Param.): typeRetour*



*Contraintes*

{inv: solde > decouvertMax}

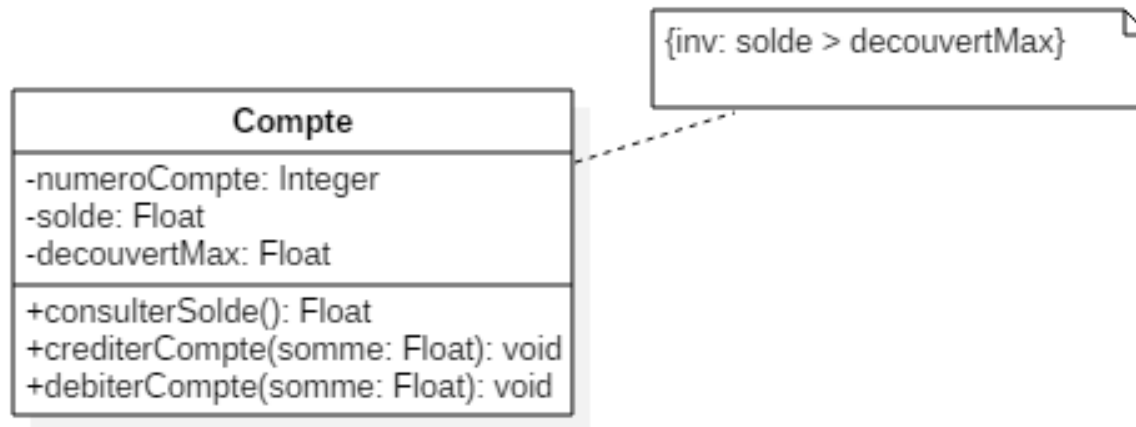
- Convention :

- Le nom de la classe commence par une lettre majuscule et doit être au singulier
- Le nom des attributs et des opérations doivent commencer par une minuscule



# Sémantique d'une classe

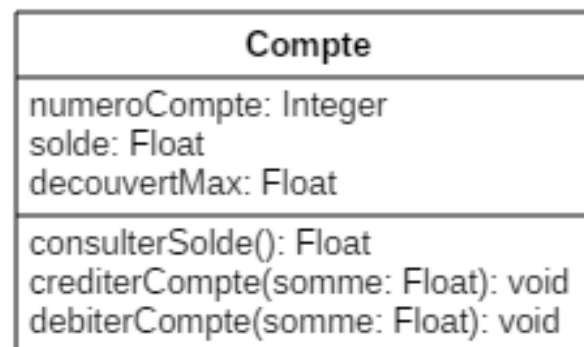
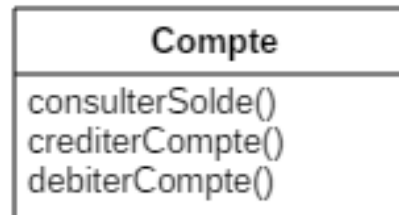
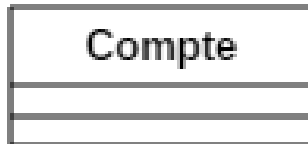
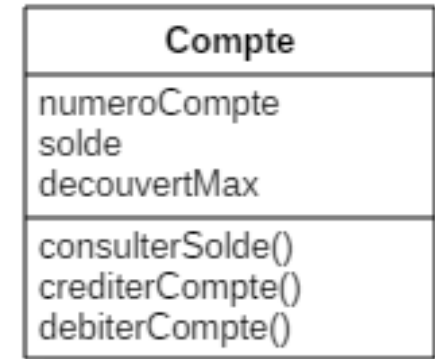
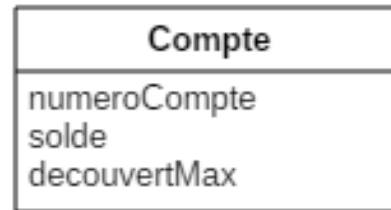
---



- ▶ Le concept de **Compte** est pertinent
  - ▶ Le numéro d'un compte est un Integer
  - ▶ Le solde d'un compte est un Float
  - ▶ Le découvert max d'un compte est un Float
- ▶ Pour un compte donné, il est possible de :
  - ▶ Consulter le solde, créditer une somme, débiter une somme
- ▶ Un compte doit toujours avoir un solde supérieur au découvert max

# Des notations possibles

---



# Concepts avancés pour les classes

---

A utiliser à bon escient !

Lorsque nécessaire et uniquement lorsque nécessaire !

## ► S'adapter :

- Au niveau d'abstraction (Conception vs. Implantation)
- Au domaine d'application
- Aux outils utilisés
- Aux savants et ignorants
- Ingénierie vs. rétro-ingénierie

# Concepts avancés pour les attributs et les opérations

---

## ► Concepts avancés :

- ▶ Visibilités, portée et dérivation
- ▶ Propriétés :
  - ▶ Attributs : {frozen}, {addonly}, {ordered}, {nonunique}
  - ▶ Opérations : {abstract}, {isQuery}, {concurrency = valeur}, {isLeaf}, {isRoot}
- ▶ Enumérations et types de données

## ► Déclarations des attributs :

- ▶ `[visibilité[/]nom[:type][card ordre][= valeurInitiale][{props}]`

## ► Déclaration des opérations :

- ▶ `[visibilité[/] nom [(params)][:type][{props}]`
- ▶ `Params := [in|out|inout] nom [:type][=default][{props}]`

# Visibilité UML

---

Caractère	Rôle	Mot-clé	Description
+	Accès public	public	Accessible par toutes les classes
#	Accès protégé	protected	Accessible uniquement par la classe et ses classes filles
~	Accès package	package	Accessible uniquement par les classes du même package
-	Accès privé	private	Accessible uniquement depuis la classe elle-même



Les visibilités UML ne se correspondent pas exactement aux visibilités de Java mais correspondent à celles de C++ !

# Exemples de déclarations d'attributs

Attribut dérivé

Non modifiable

Seul l'ajout est possible

```
age
+ age
/age
- solde : Integer = 0
# age : Integer[0..1]
# numsecu : Integer{frozen}
# motsClés : String[*]{addOnly}
nbPersonne : Integer
```

{age = dateCourante -  
dateNaissance}

*Conséquences  
d'implantation Java :*

← final

← static



Adapter le niveau de détail au niveau d'abstraction

# Exemples de déclarations d'opérations

---

```
/getAge()  
+ getAge() : Integer  
- updateAge(in date : Date) : Boolean  
# getName() : String[0..1]  
+ getAge() : Integer {isQuery}  
#getAge(): Integer  
#getAge(): Integer {abstract}  
+ addProject() : {concurrency = sequential}  
+ addProject() : {concurrency = concurrent}  
main(in args : String[*]){ordered}}
```

*Conséquences  
d'implantation Java :*

Méthode abstraite



Adapter le niveau de détail au niveau d'abstraction

# Exemples d'énumérations et de types de données

<<enumeration>> Jour
Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche

<<enumeration>> Fonction
Secetaire President Tresorier VicePresident Membre

<<datatype>> Point
x : Integer y : Integer

<<datatype>> Date
compare() jour() mois() annee()

- ▶ Utilisables comme type d'attributs
- ▶ Valeurs (par identité)
- ▶ Exemple d'utilisation :

Association1901
nom : String joursDeReunion : Jour[*] dateDeCreation : Date



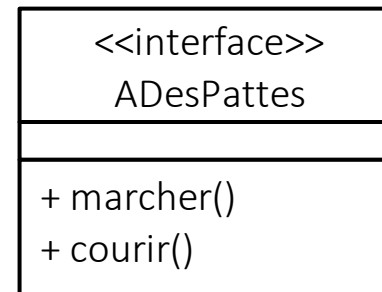
# Les classes abstraites et les interfaces

---

## ► Classes abstraites :



## ► Interfaces :



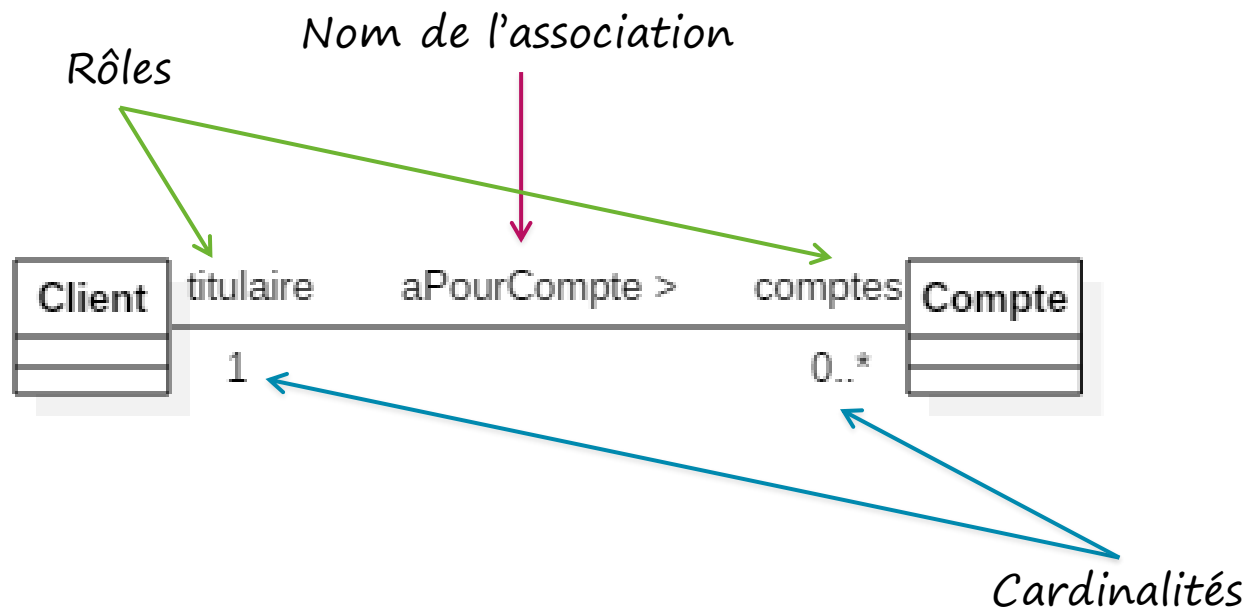
# Diagramme de classes

---

Les associations

# Une association

- ▶ Une association décrit un groupe de liens qui partagent une structure et une sémantique communes

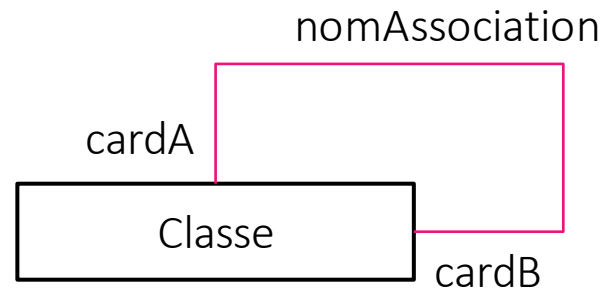


- ▶ Une association relie deux ou plus classes
- ▶ Une association a un nom et/ou des rôles
- ▶ Une association a des cardinalités à chaque extrémité

# Une association particulière

---

- Association **réflexive**:



# Les cardinalités

---

## ► Les cardinalités d'une association :

- Précise combien d'objets peuvent être liés à un autre objet
- Sont définies avec une cardinalité minimale et une cardinalité maximale au format  $C_{\min}..C_{\max}$



« Un client a **0 ou plusieurs** comptes. »

« Un compte a toujours **un et un seul** titulaire. »

# Concepts avancés pour les associations

---

A utiliser à bon escient !

Lorsque nécessaire et uniquement lorsque nécessaire !

► S'adapter :

- ▶ Au niveau d'abstraction (Conception vs. Implantation)
- ▶ Au domaine d'application
- ▶ Aux outils utilisés
- ▶ Aux savants et ignorants
- ▶ Ingénierie vs. rétro-ingénierie

# Concepts avancés pour les associations

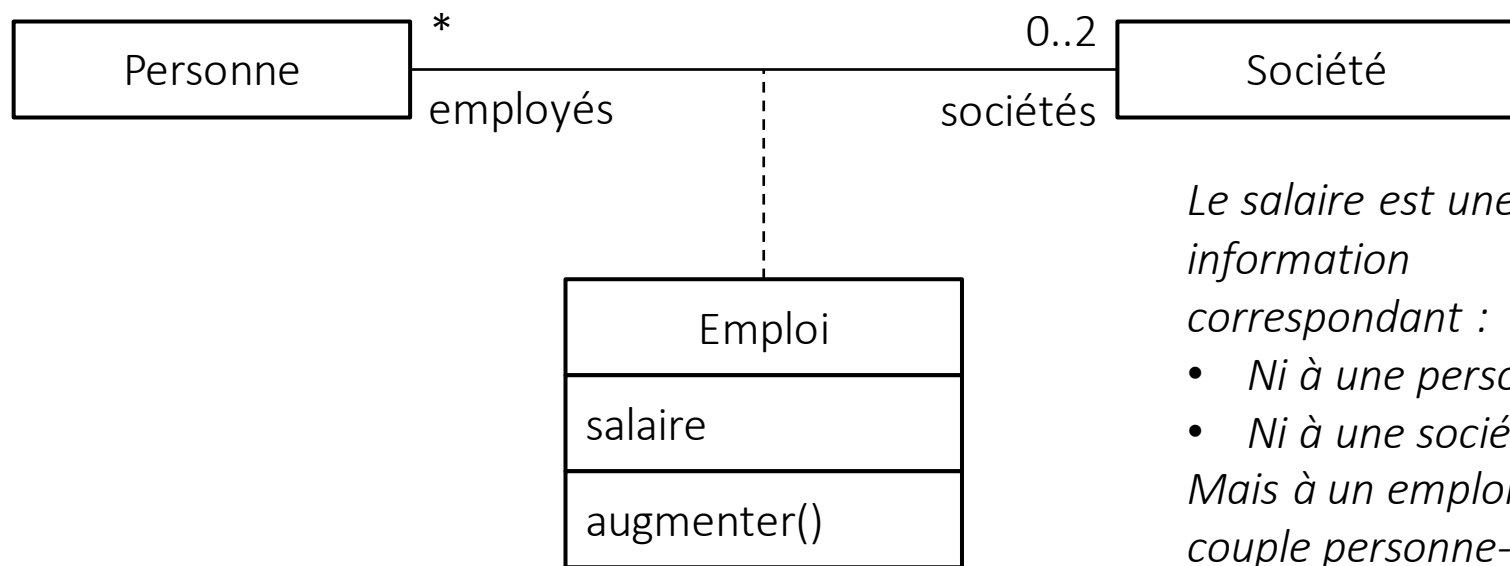
---

- ▶ Classe associative
- ▶ Généralisation
- ▶ Composition et ~~agrégation~~ (Depuis UML 2.0 en 2005)
- ▶ Navigation : associations unidirectionnelle et bidirectionnelle
- ▶ Contraintes : {frozen}, {addonly}, {ordered}, {nonunique}
- ▶ Association qualifiée

# Classe associative – 1/2

## ► Objectif :

- Associer des attributs et/ou des opérations à une association



*Le salaire est une information correspondant :*

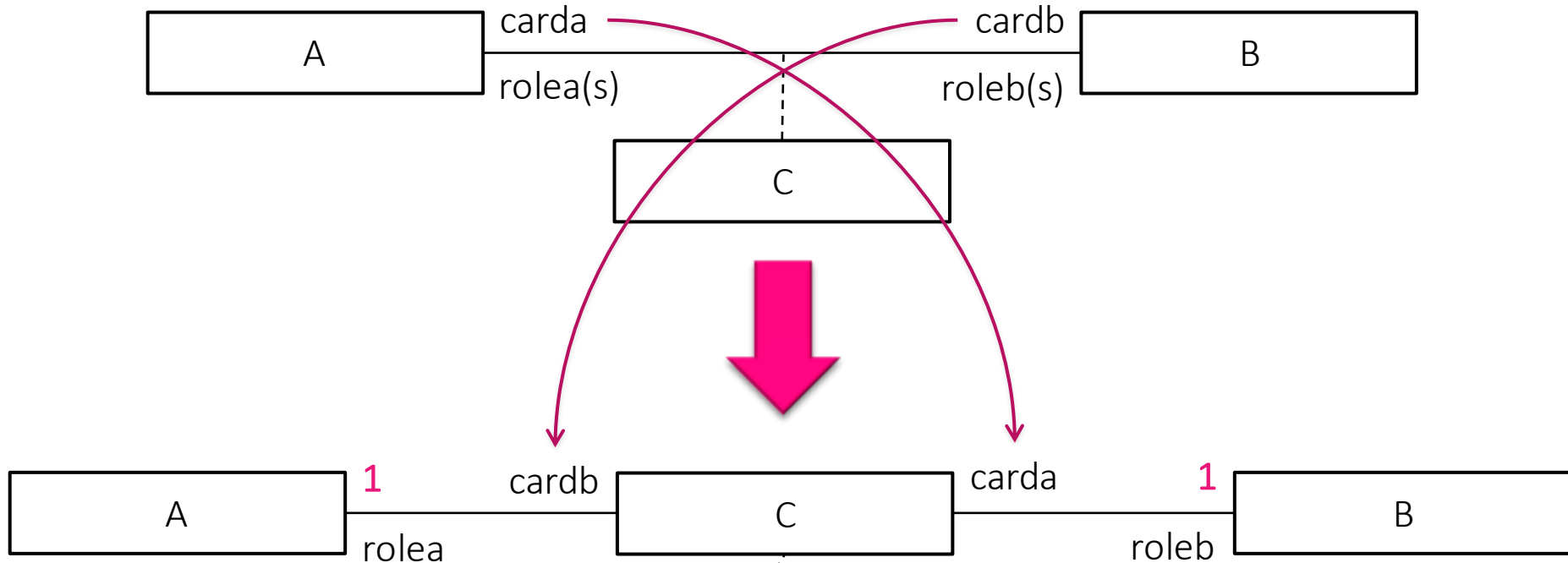
- *Ni à une personne*
  - *Ni à une société*
- Mais à un emploi (un couple personne-société)*

## ► Le nom de la classe correspond au nom de l'association



## Classe associative – 2/2

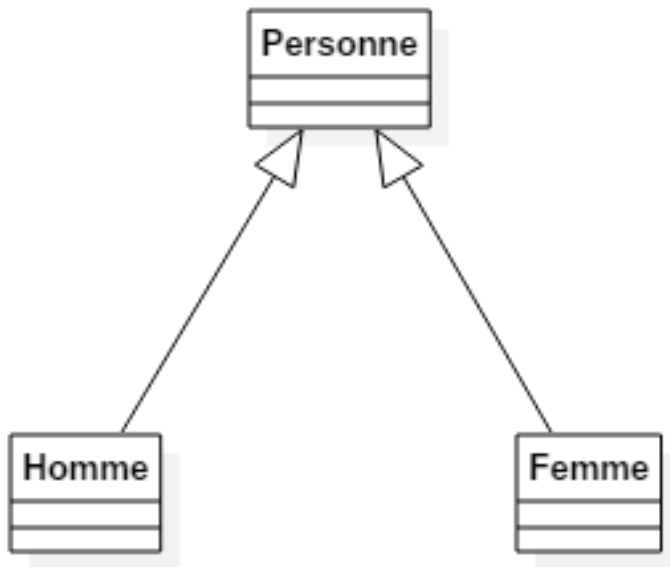
### ► Transformation d'une classe associative



Il ne peut y avoir qu'un objet C  
entre un objet A et un objet B donné

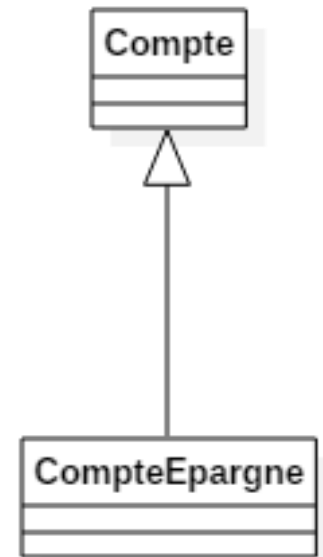
# Généralisation/Spécialisation – 1/2

- Une classe peut être la **généralisation** d'une ou plusieurs classes, ces classes sont alors des **spécialisations**



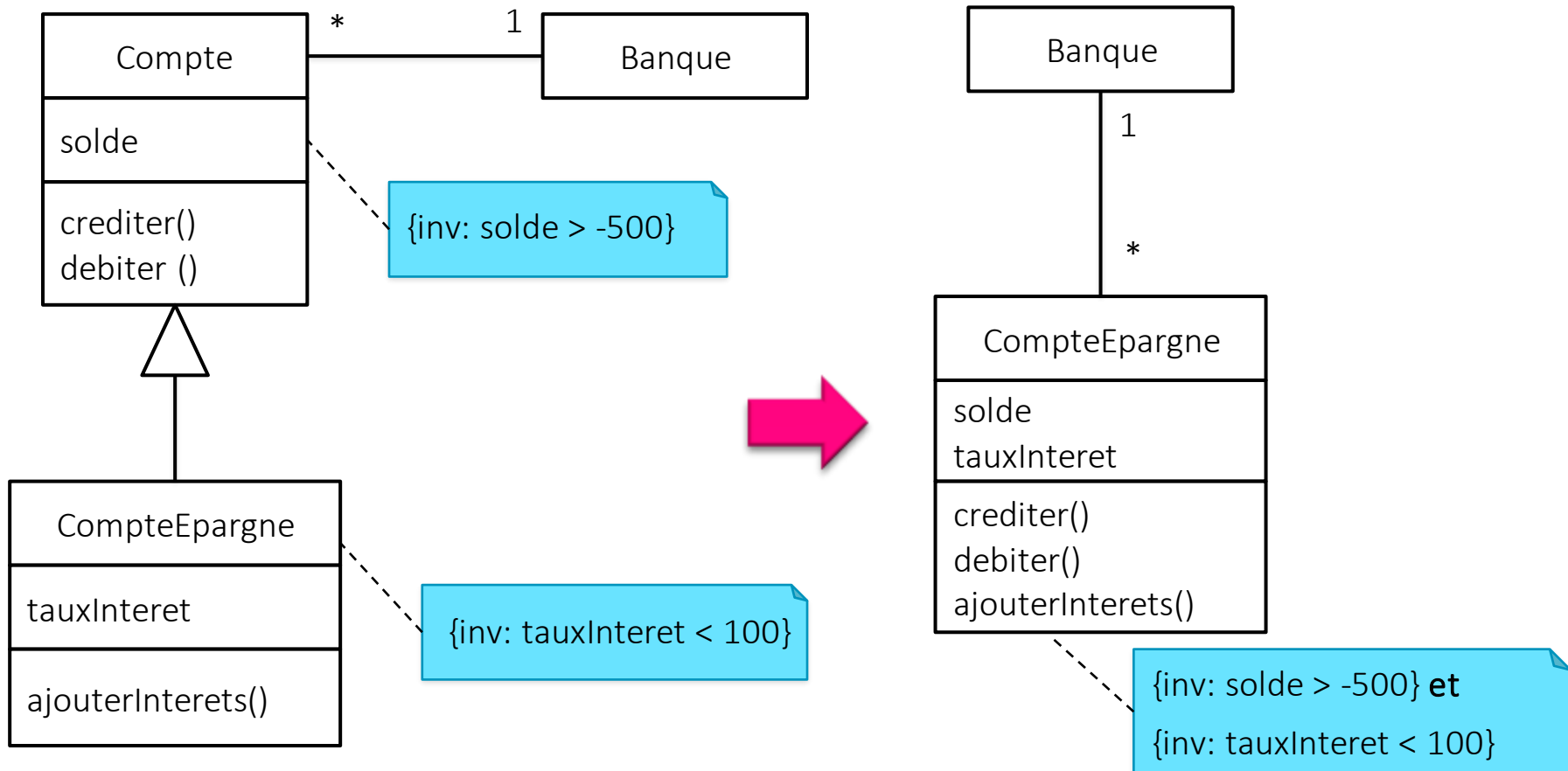
« Super classe »  
Cas général

« Sous-classe »  
Cas spécifique



# Généralisation/Spécialisation – 2/2

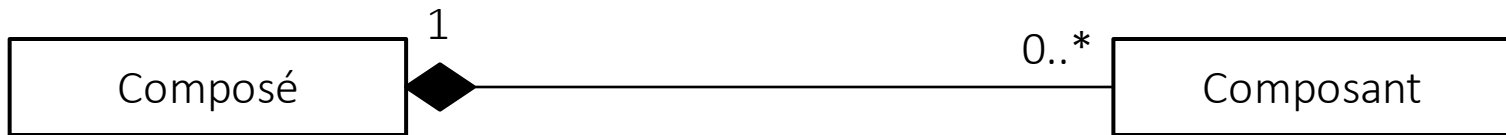
- Les sous-classes **héritent** des propriétés des super classes (attributs, méthodes, associations et contraintes)



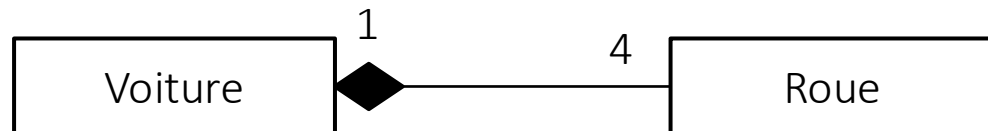
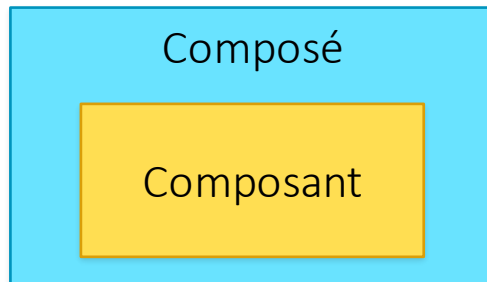
# Composition – 1/2

## ► Objectif :

- Représenter un couplage fort entre le composé (ensemble) et les composants (éléments)
- Lie les cycles de vie du composé avec les composants



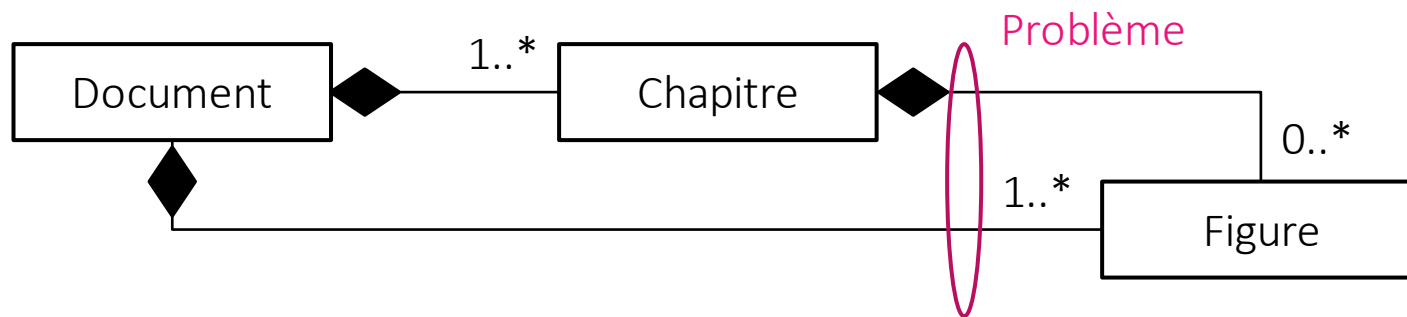
## ► Illustration et exemple :



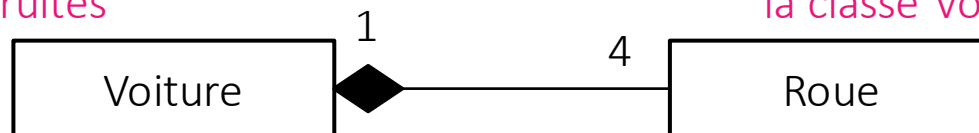
# Composition – 2/2

## ► Contraintes liées à la composition :

- Un objet composant ne peut être que dans un seul objet composé
- Un objet composant n'existe pas sans son objet composé, sa création se fait via son composé
- Si un objet composé est détruit, ses composants aussi



A la destruction de la voiture,  
les roues sont détruites



Roue créée dans  
la classe Voiture

# Navigation – 1/2

---



A utiliser pour les spécifications et l'implémentation

Association bidirectionnelle		
Association unidirectionnelle		
		

# Navigation – 2/2



```
public class A {
    private B rb;

    public void addB(B b) {
        setRb(b);
    }
}
```

```
public class B {
}
```



```
public class A {
    private B rb;

    public void addB(B b) {
        if(getRb() != null) {
            getRb().setRa(null);
        }
        if(b.getRa() != null) {
            b.getRa().setRb(null);
        }
        setRb(b);
        getRb().setRa(this);
    }
}
```

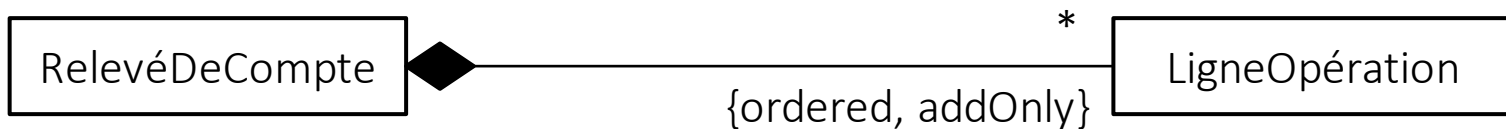
```
public class B {
    private A ra;

    public void addA(A a) {
        if (getRa() != null) {
            getRa().setRb(null);
        }
        if (a.getRb() != null) {
            a.getRb().setRa(null);
        }
        setRa(a);
        getRa().setRb(this);
    }
}
```

# Contraintes sur les associations

---

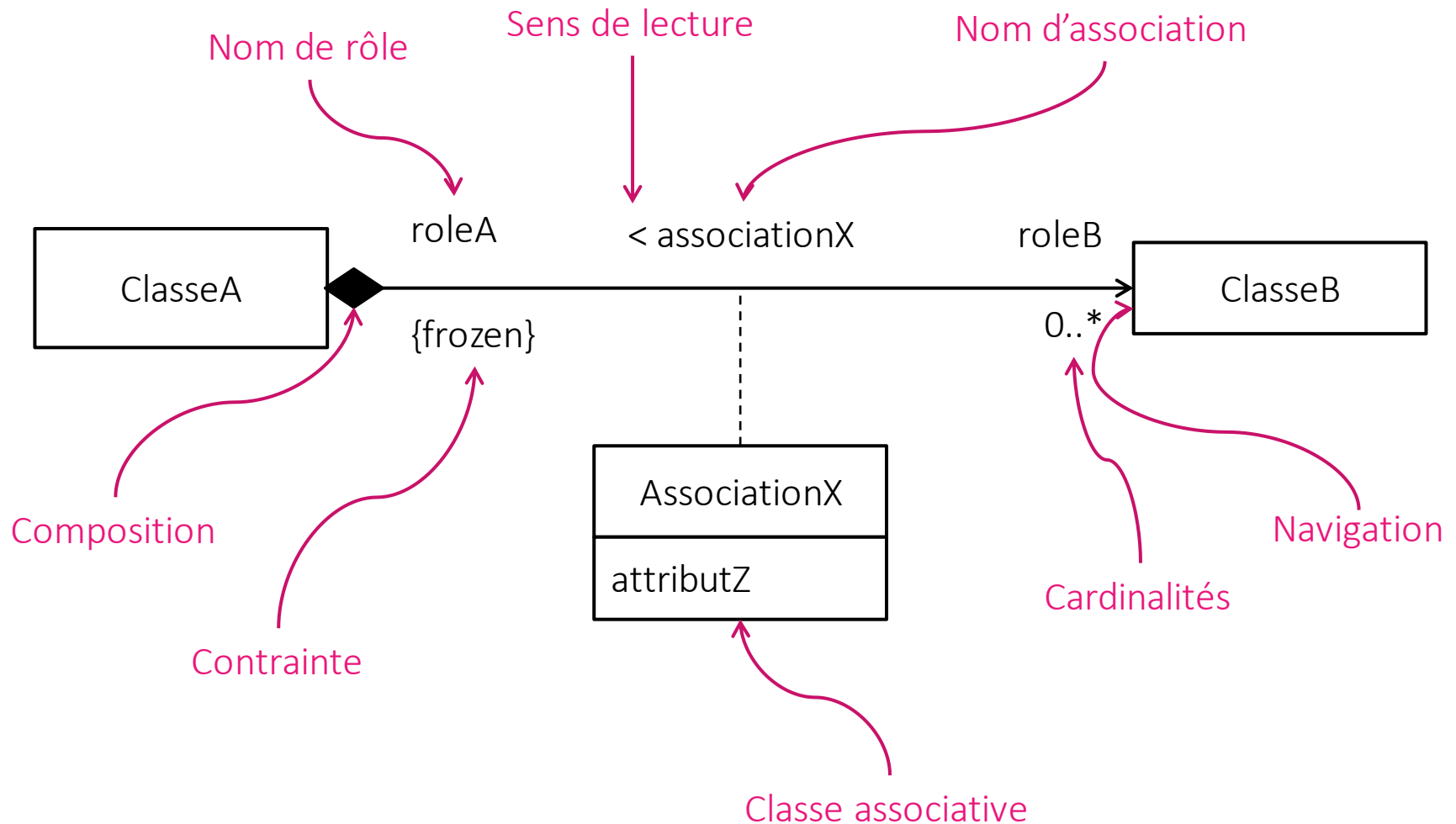
- ▶ **{ordered}** : les éléments de la collection sont ordonnés
- ▶ **{nonUnique}** : répétitions possibles
- ▶ **{frozen}** : fixé lors de la création de l'objet, ne peut pas changer
- ▶ **{addOnly}** : impossible de supprimer un élément



Influence les structures de données de l'implantation



# Synthèse sur les associations



# Diagramme d'objets

---

Des objets et des liens

# Les objets

---

- ▶ Les objets peuvent être ajoutés ou détruits pendant l'exécution
- ▶ La valeur des attributs peut changer
- ▶ Représentations graphiques :

nomDeLObjet

nomDeLObjet:Classe

:Classe

# Exemples d'objets

---

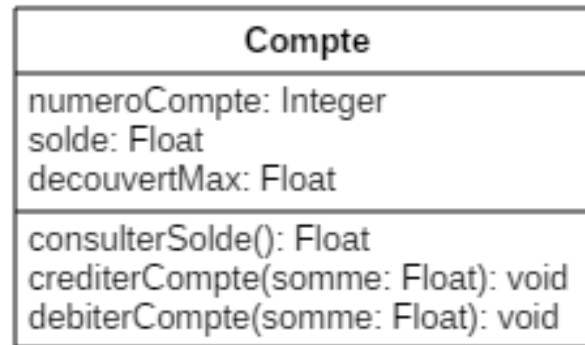
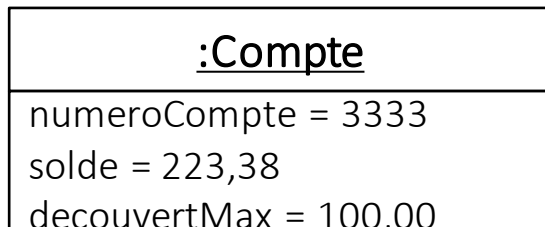
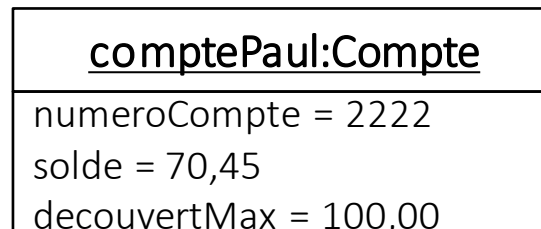
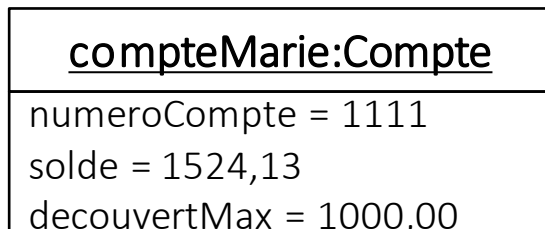


Diagramme de classes

Diagramme d'objets



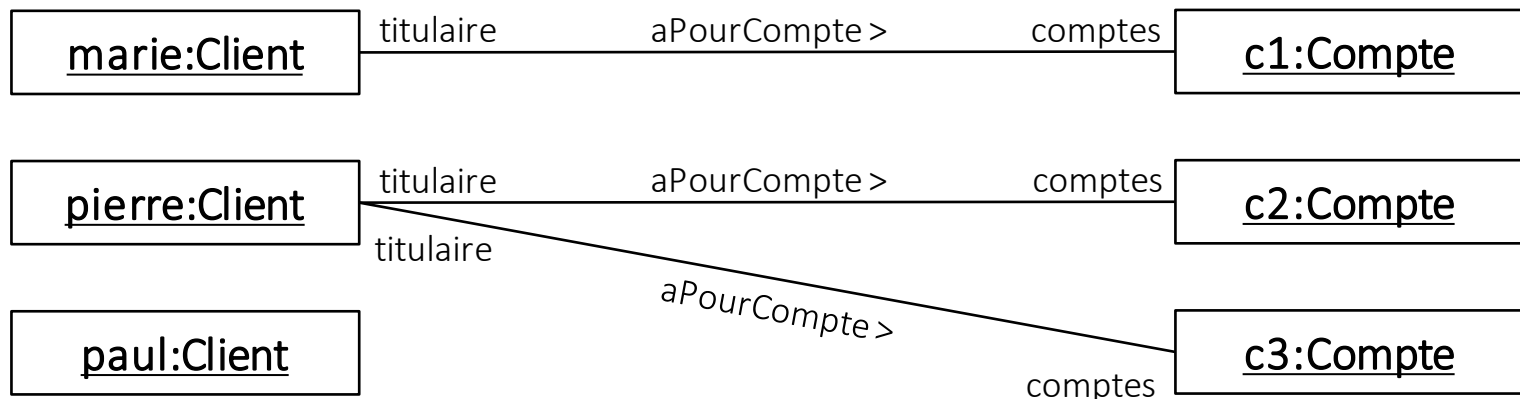
# Les liens

- ▶ Représentent les instances des associations entre les classes des objets considérés



Diagramme de classes

Diagramme d'objets



# Contrainte sur les liens – 1/2

- ▶ Au maximum un lien d'un type donné entre deux objets donnés
- ▶ Exemple :

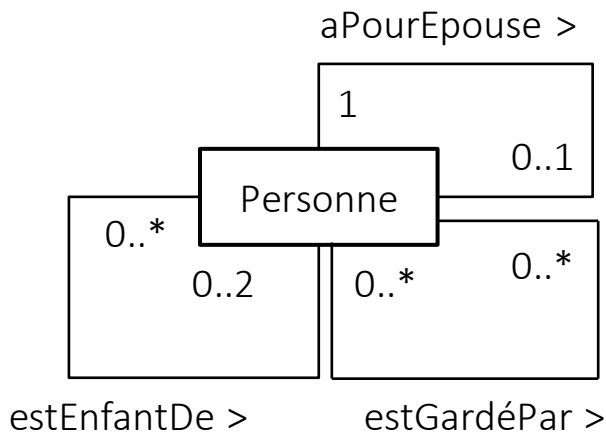
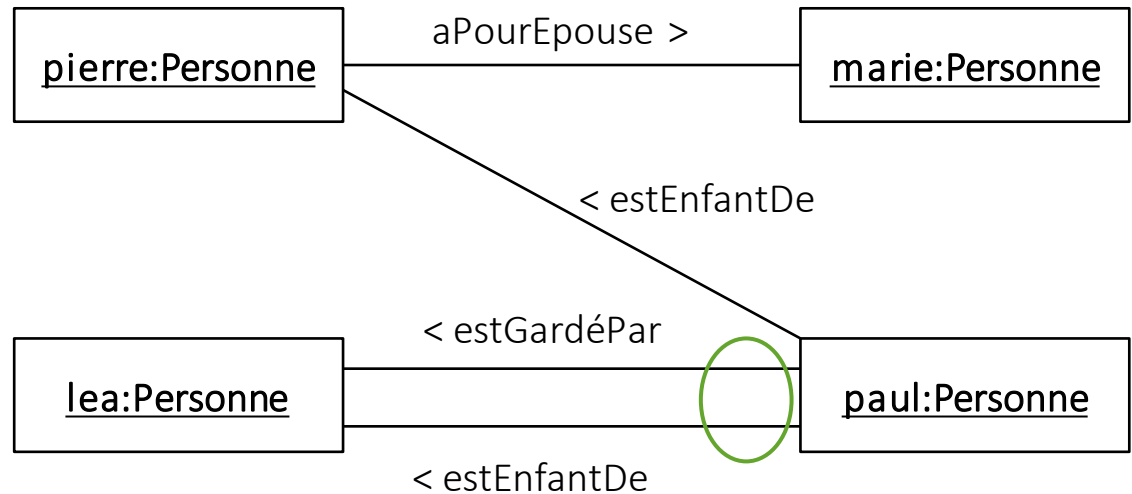


Diagramme de classes

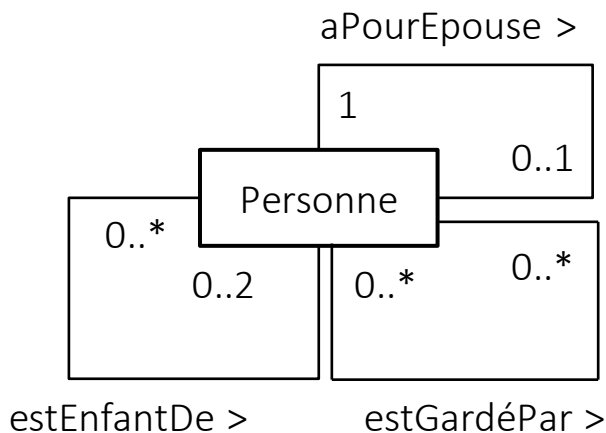


OK

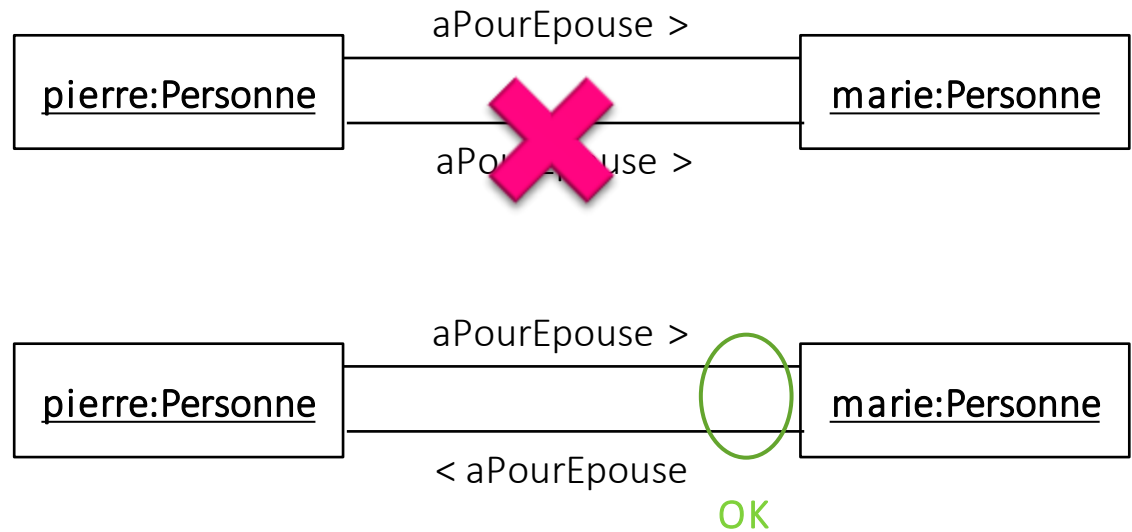
Diagramme d'objets

# Contrainte sur les liens – 1/2

- ▶ Au maximum un lien d'un type donné entre deux objets donnés
- ▶ Contre-exemple :




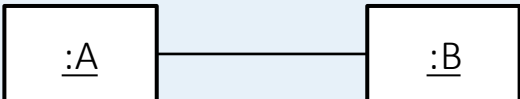

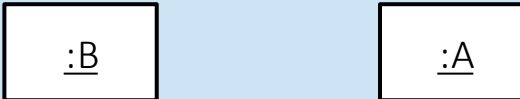
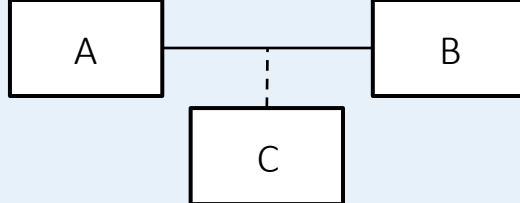
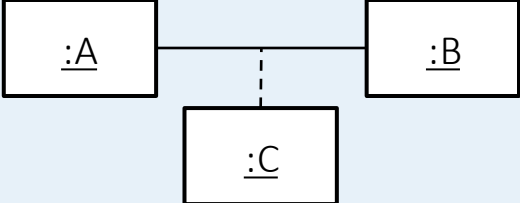


*Diagramme de classes*



*Diagramme d'objets*

# Cas particuliers

	Diagramme de classes	Diagrammes d'objets
Navigation	 <pre>classDiagram     A --&gt; B</pre>	 <pre>graph LR     A[":A"] --- B[":B"]</pre>
Composition	 <pre>classDiagram     A *-- B : cardB</pre>	 <pre>graph LR     A[":A"] --- B[":B"]</pre>
Généralisation	 <pre>classDiagram     B -- &gt; A</pre>	<p>Ou éventuellement</p>  <pre>graph LR     B[":B"]     A[":A"]</pre>
Classe associative	 <pre>classDiagram     A --- B     C -.- A     C -.- B</pre>	 <pre>graph LR     A[":A"] --- B[":B"]     C[":C"] -.- A     C -.- B</pre>



# Conformité entre le diagramme de classes et le système modélisé

- Est-ce que ce diagramme de classes est correct ?

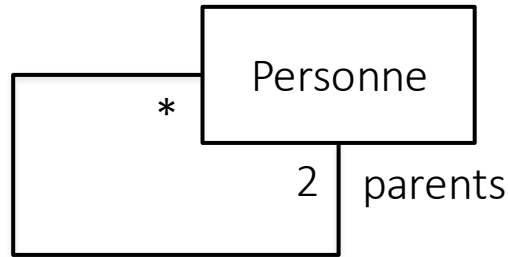


Diagramme  
de classes

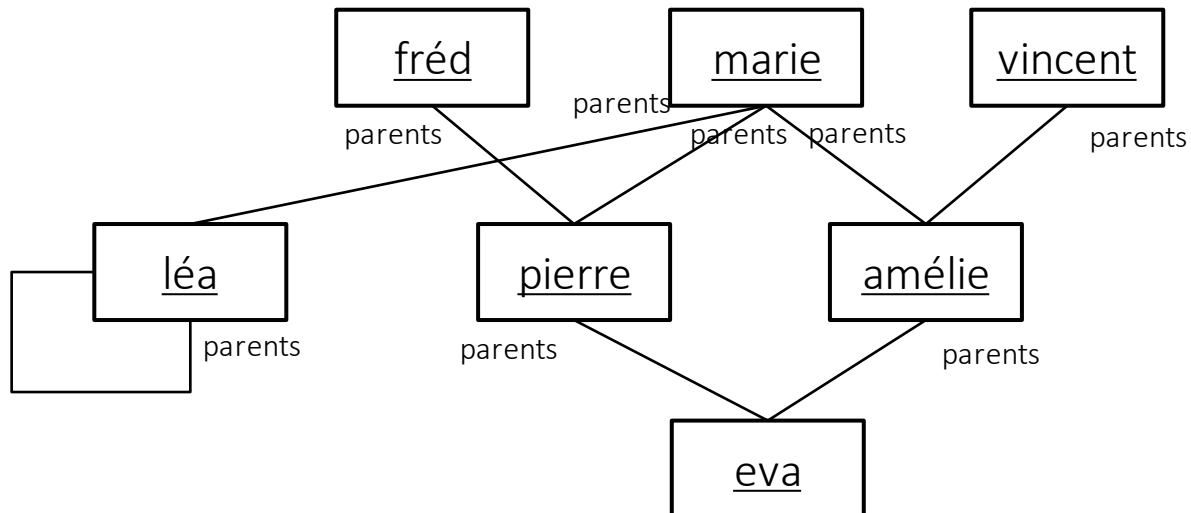


Diagramme  
d'objets

# Conformité entre le diagramme de classes et le système modélisé

- Est-ce que ce diagramme de classes est correct ?

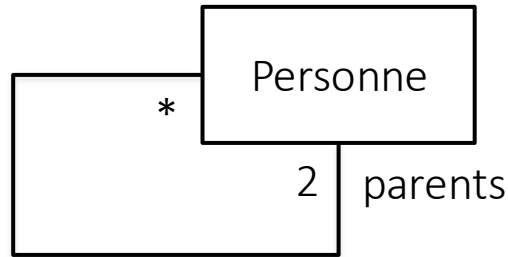
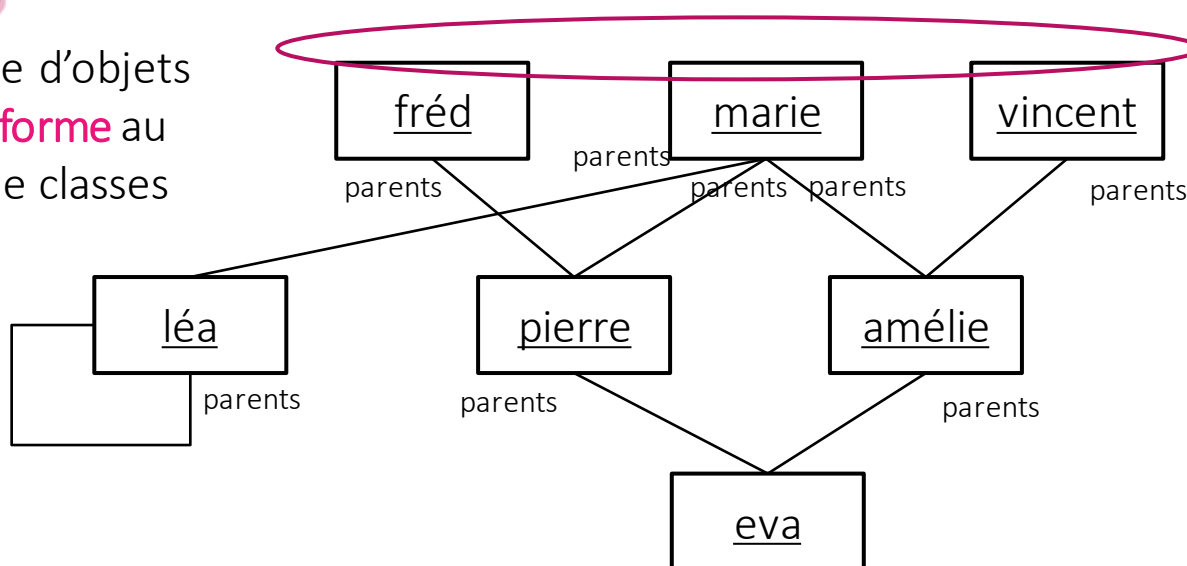


Diagramme  
de classes

Diagramme  
d'objets

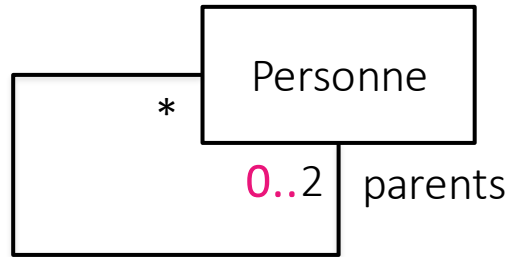


Ce diagramme d'objets  
**n'est pas conforme** au  
diagramme de classes



# Conformité entre le diagramme de classes et le système modélisé

- Est-ce que ce diagramme de classes est correct ?



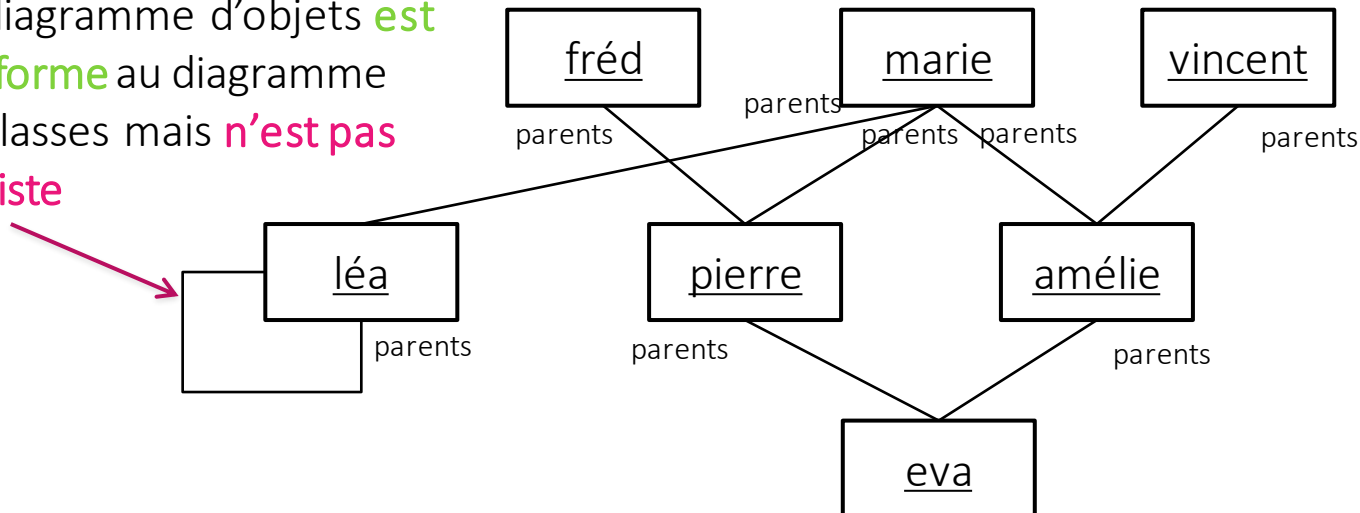
Ce diagramme de classes  
**n'est pas un bon modèle**  
du système modélisé

Diagramme  
de classes

Diagramme  
d'objets



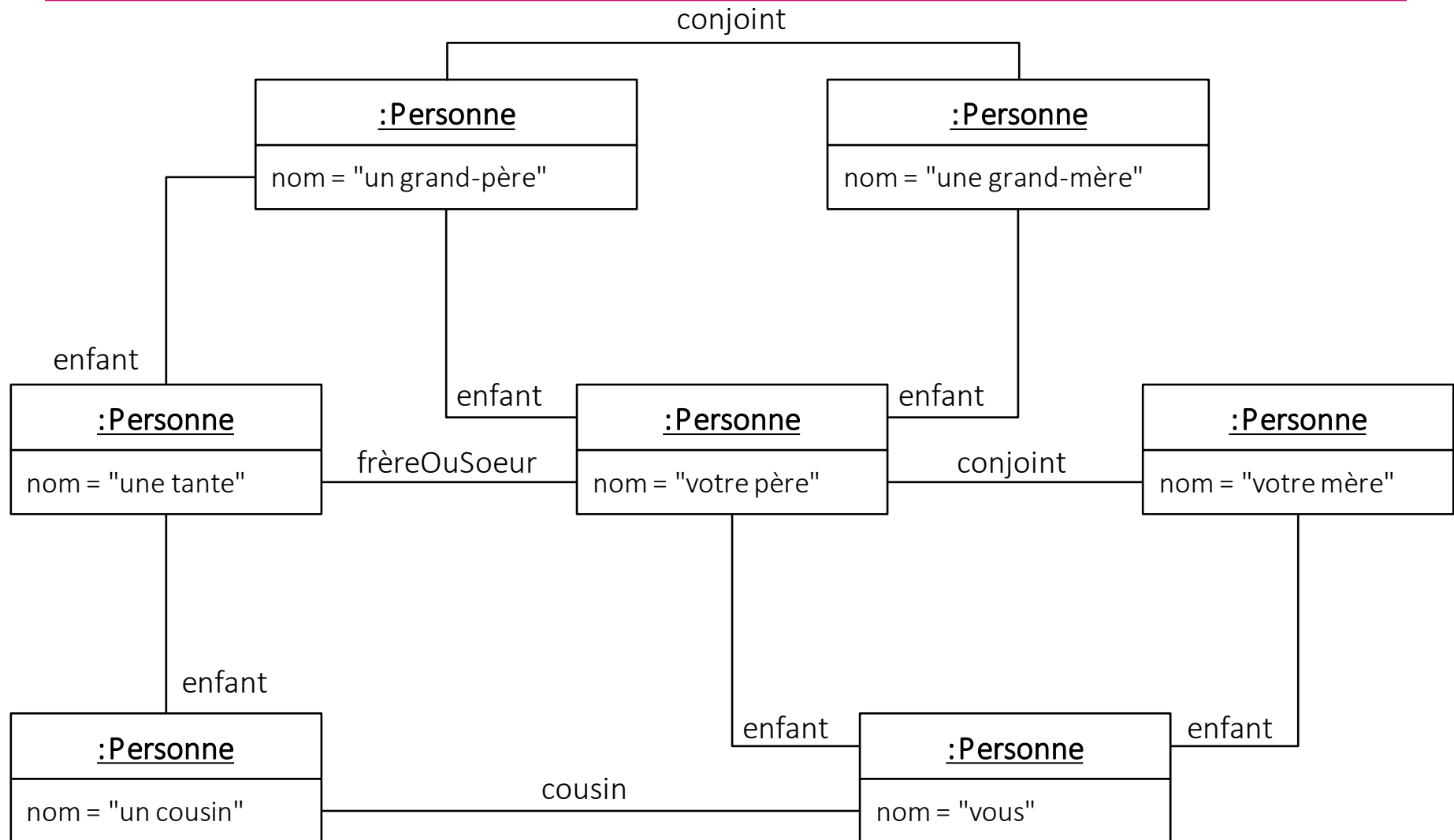
Ce diagramme d'objets **est conforme** au diagramme de classes mais **n'est pas réaliste**



Un peu de gymnastique

---

# Exercice 1 : Donnez le diagramme de classes qui permet de construire ce diagramme d'objets.



## Exercice 2 : Est-ce possible ?

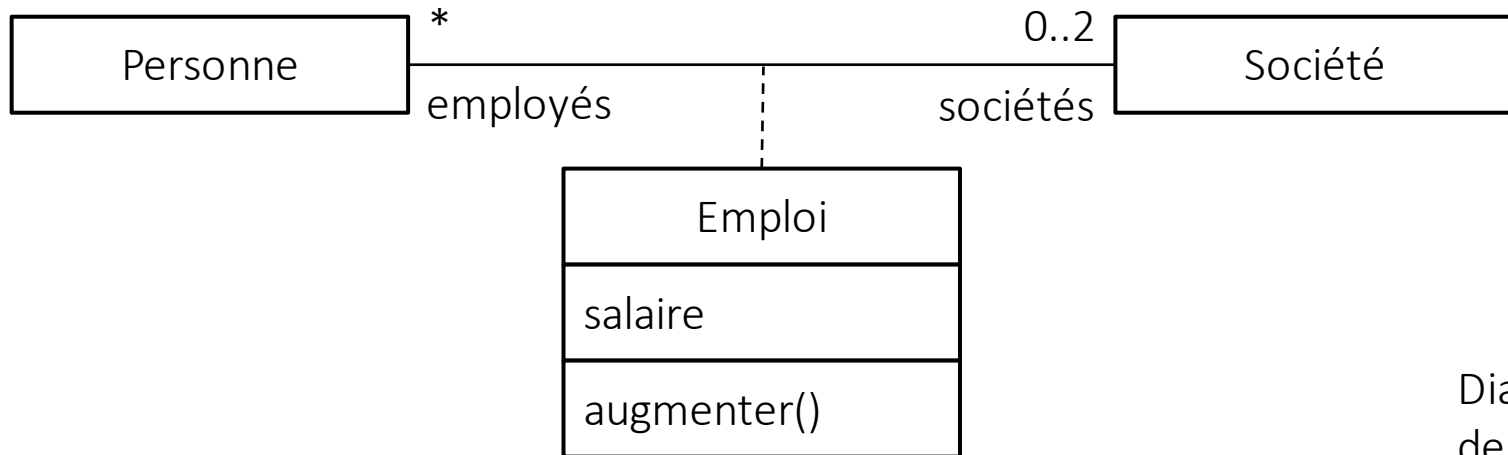


Diagramme  
de classes

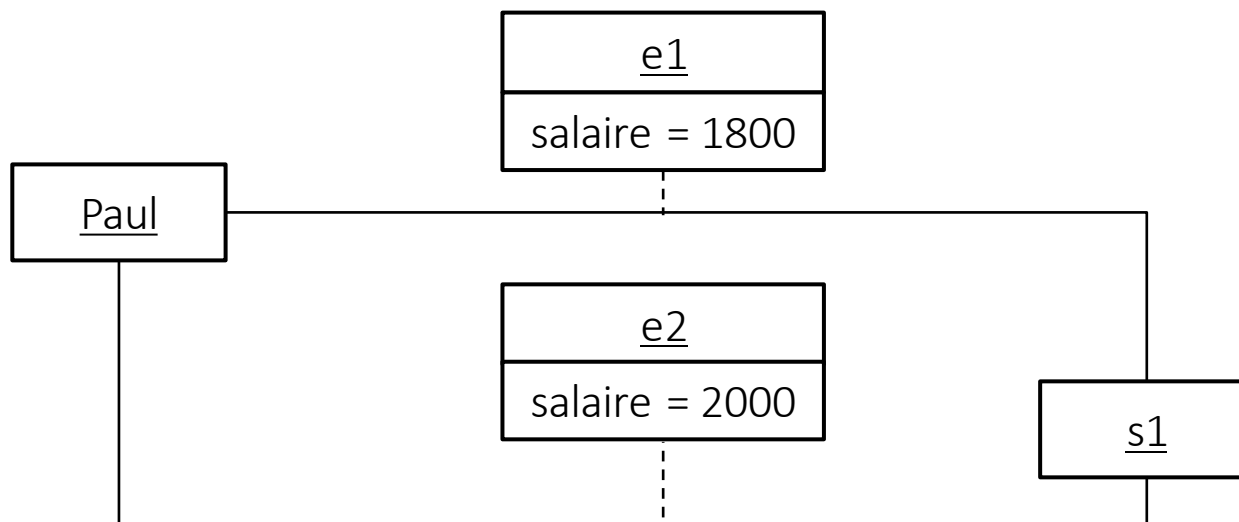
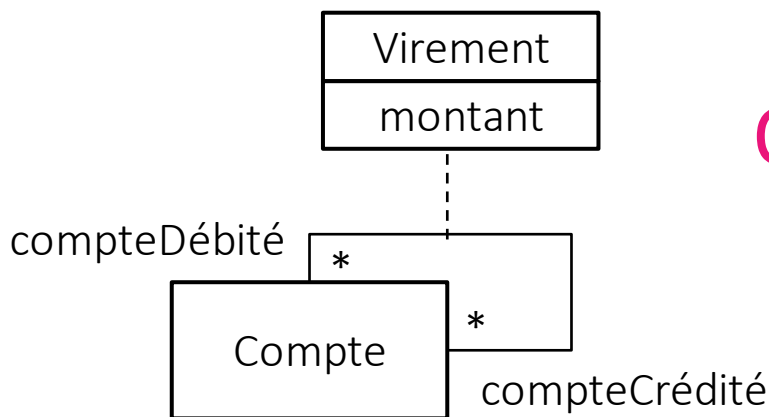


Diagramme  
d'objets

## Exercice 3 : Quelle est la bonne modélisation ?



Ou ?

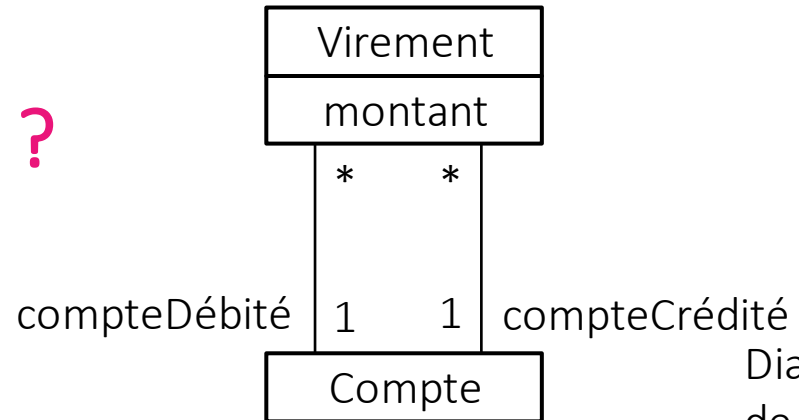
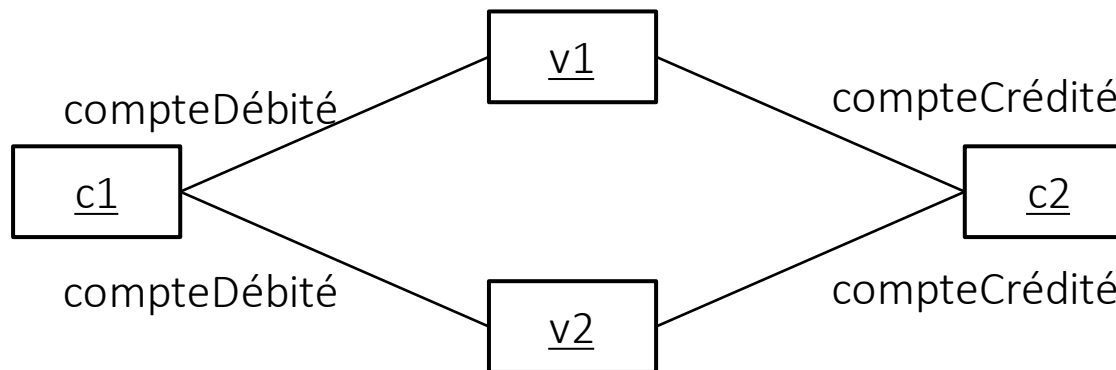


Diagramme de classes

Peut-on avoir plusieurs virements entre deux comptes ?

Diagramme d'objets



## Exercice 4 : Quelle est la bonne modélisation ?

Ou ?

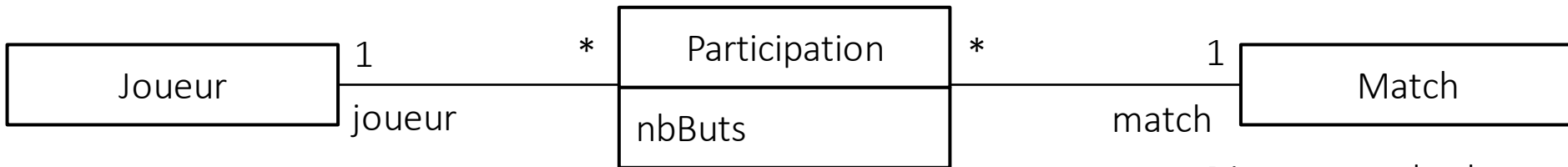
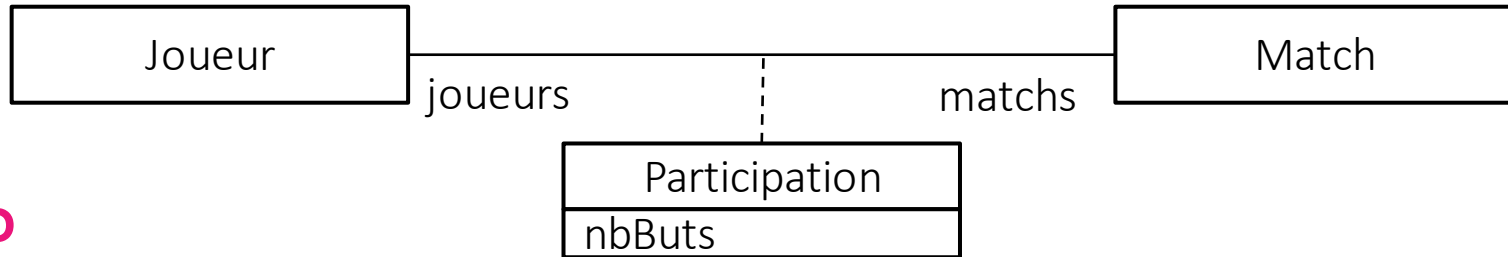
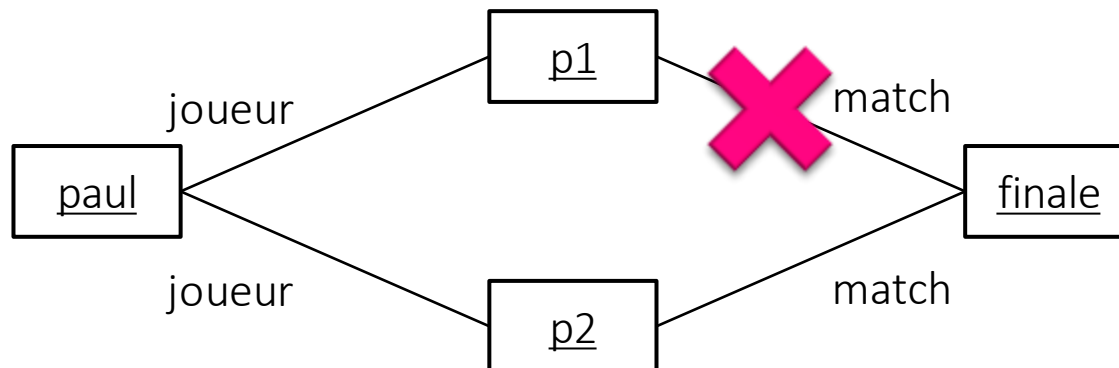


Diagramme de classes

Un joueur peut-il avoir plusieurs participations à un match donné ?

Diagramme d'objets



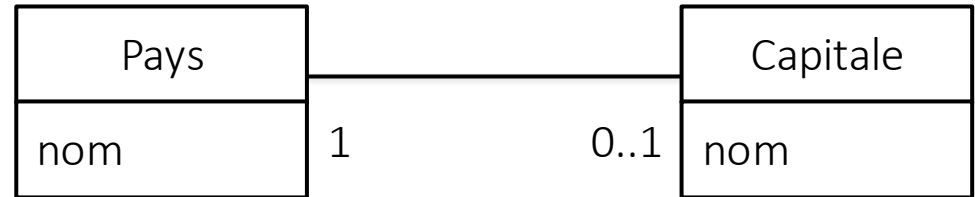


## Exercice 5 : Modélisez « un pays a une capitale »

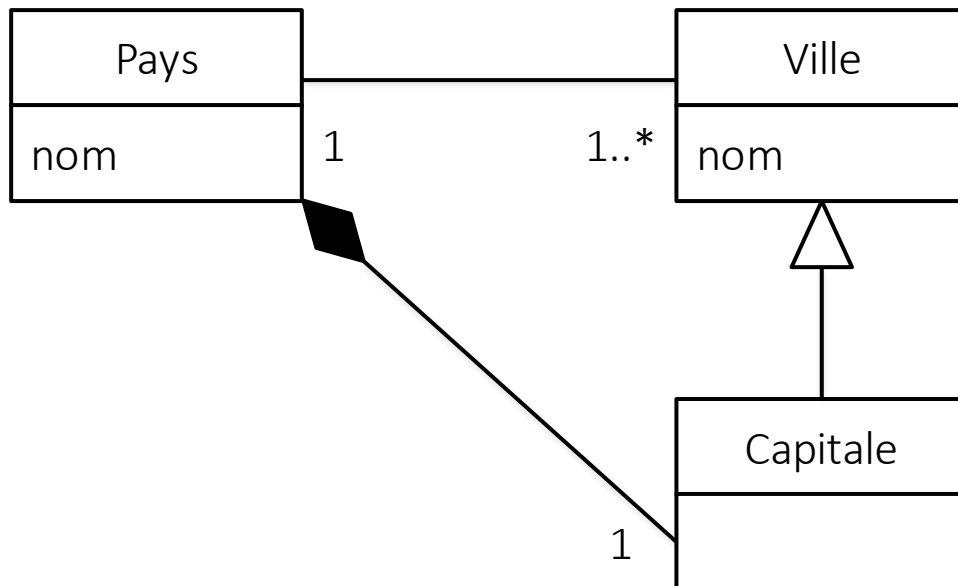
1



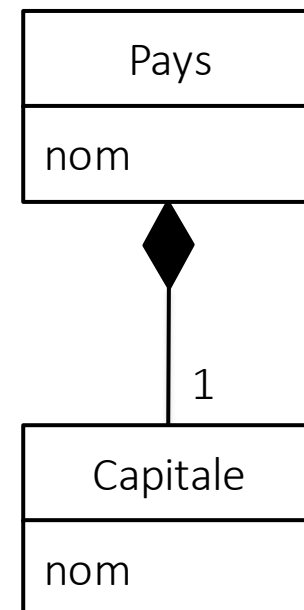
2



3



4



# Synthèse

---

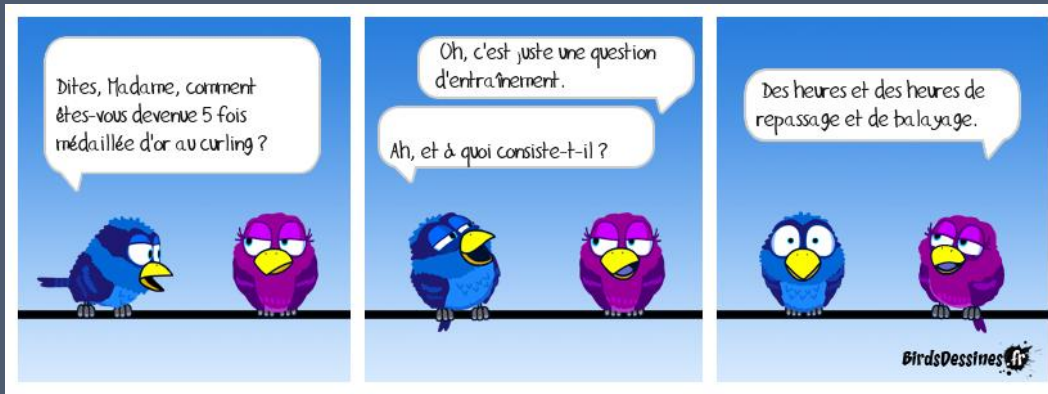
# Synthèse

---

- ▶ Modéliser :
  - ▶ Pour une meilleure compréhension du monde réel
  - ▶ Pour permettre une conception progressive
  - ▶ Pour faciliter la visualisation du système
- ▶ Besoin d'un langage de modélisation pour modéliser :



- ▶ De nombreux diagrammes à utiliser à bon escient en fonction de la phase d'avancement d'un projet et avec le bon niveau de détail !



Comment bien modéliser ?

---

En modélisant !

# A vous de jouer !

---

1. Un philosophe qui mange, se sert d'une fourchette. (*Problème des philosophes OS*)
2. Un fichier est un fichier ordinaire ou un répertoire.
3. Un fichier contient des enregistrements.
4. Un polygone est constitué d'un ensemble ordonné de points.
5. Un objet graphique est une zone de texte, un objet géométrique ou un groupe.
6. Une personne utilise un langage de programmation pour un projet.
7. Modems et claviers sont des périphériques d'entrée/sortie.
8. Les classes peuvent avoir plusieurs attributs.
9. Les personnes qui sont associées à l'université sont des étudiants ou des professeurs.
10. Le facteur distribue le courrier (des lettres et des colis) aux habitants de sa zone d'affectation.
11. Les étudiants assistent à des cours et les professeurs donnent des cours dans une université.
12. Les patients prennent rendez-vous chez le médecin pour un jour et une heure donnés.
13. Une personne réserve un siège pour un concert.
14. Chaque arc d'un graphe orienté est connecté dans un ordre spécifique à exactement deux sommets. Plusieurs arcs peuvent être connectés entre une paire de sommets donnée.