Suite … Diagrammes Objets et de Classes

GL 1

2016-2017

Comment créer un DCU?

Cas d'utilisation: chaque comportement du système attendu par l'utilisateur:

1. **Définir le périmètre du système : Paquetage**
2. **Identifier les acteurs (ceux qui utiliseront le futur logiciel)**
3. **Évaluer les besoins de chaque acteur en CU**
4. **Regrouper ces CU dans un diagramme présentant une vue synthétique du paquetage**
5. **Possibilité de documentation des CU complexes (cahier de charge)**

Remarque: Ne pas descendre trop bas et réinventer l'analyse fonctionnelle

Diagramme 2: « objet»

objectif

- **Objectif : Représenter les objets et leurs liens à un instant donné**

Diagramme 2: « objet»
représentation

Moyen : Graphe

- **Nœuds du graphe** = Objets

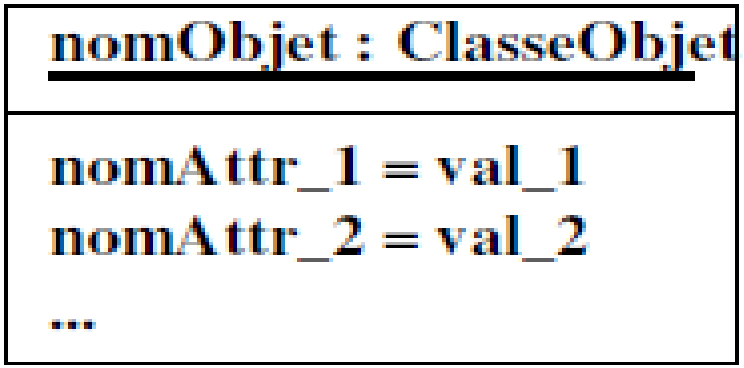
Possibilité de supprimer le nom, la classe et/ou les attributs (objet anonyme, non typé ou d'état inconnu)

- **Arêtes du graphe** = Liens entre objets

- Lien binaire : entre 2 objets. Lien n-aire : entre n objets. Possibilité de nommer les liens et les rôles

Diagramme 2: « objet»

Exemple 1



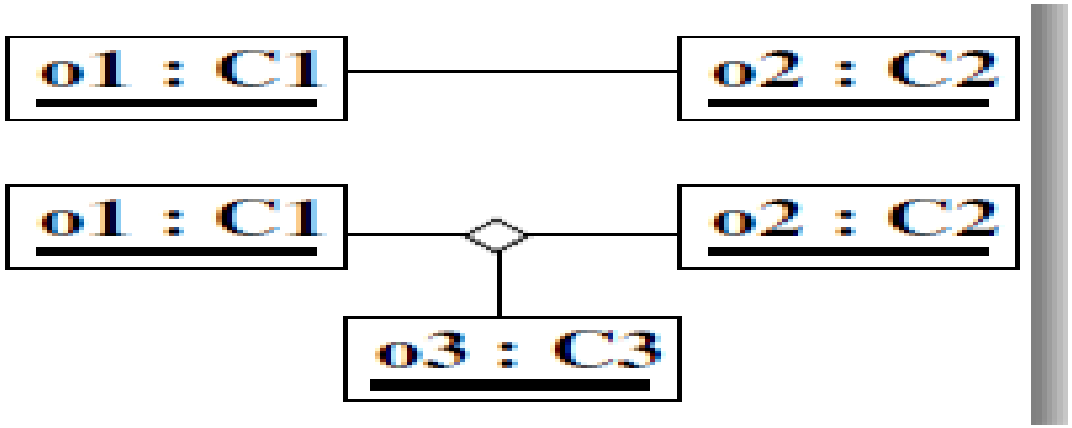


Diagramme 2: « objet»

Exemple 2

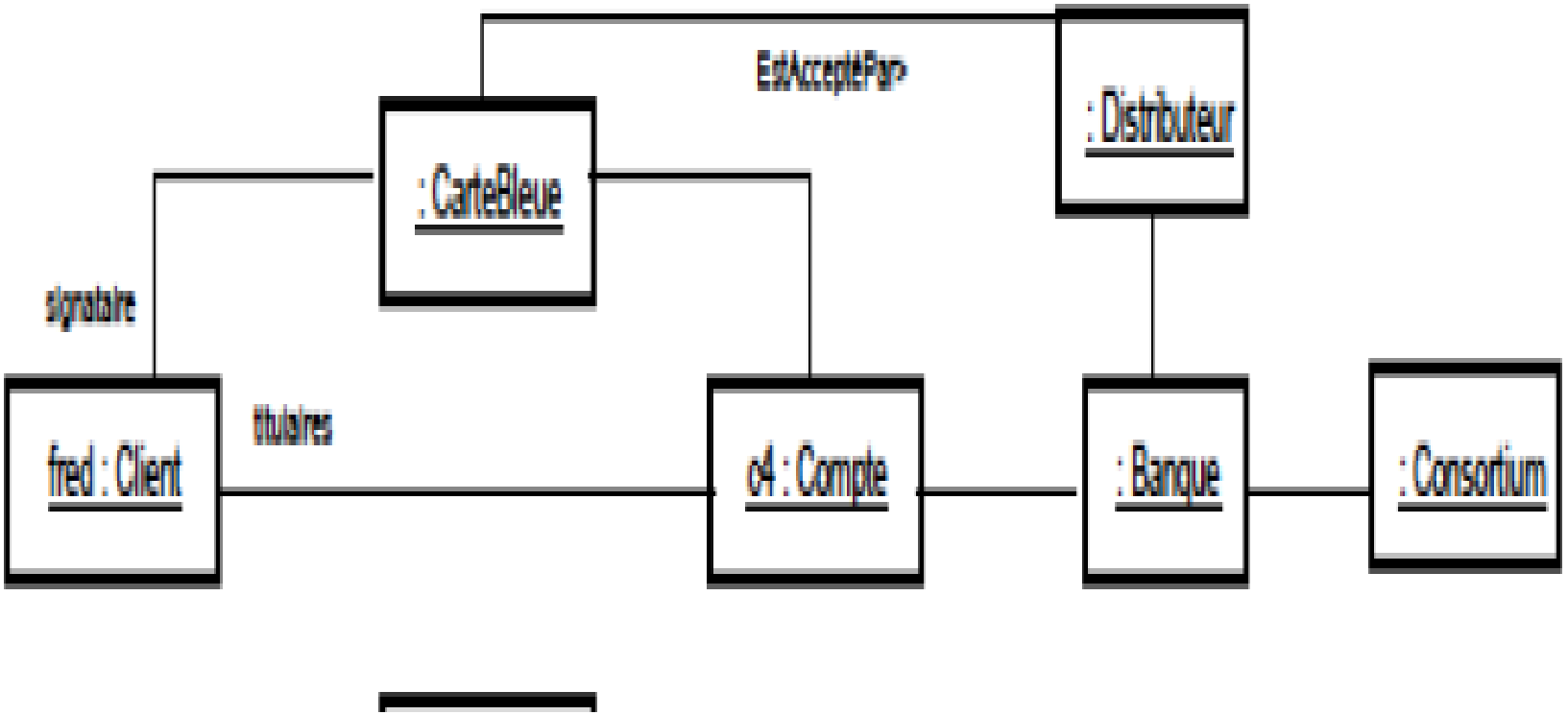


Diagramme 2: « objet»

Exemple 3

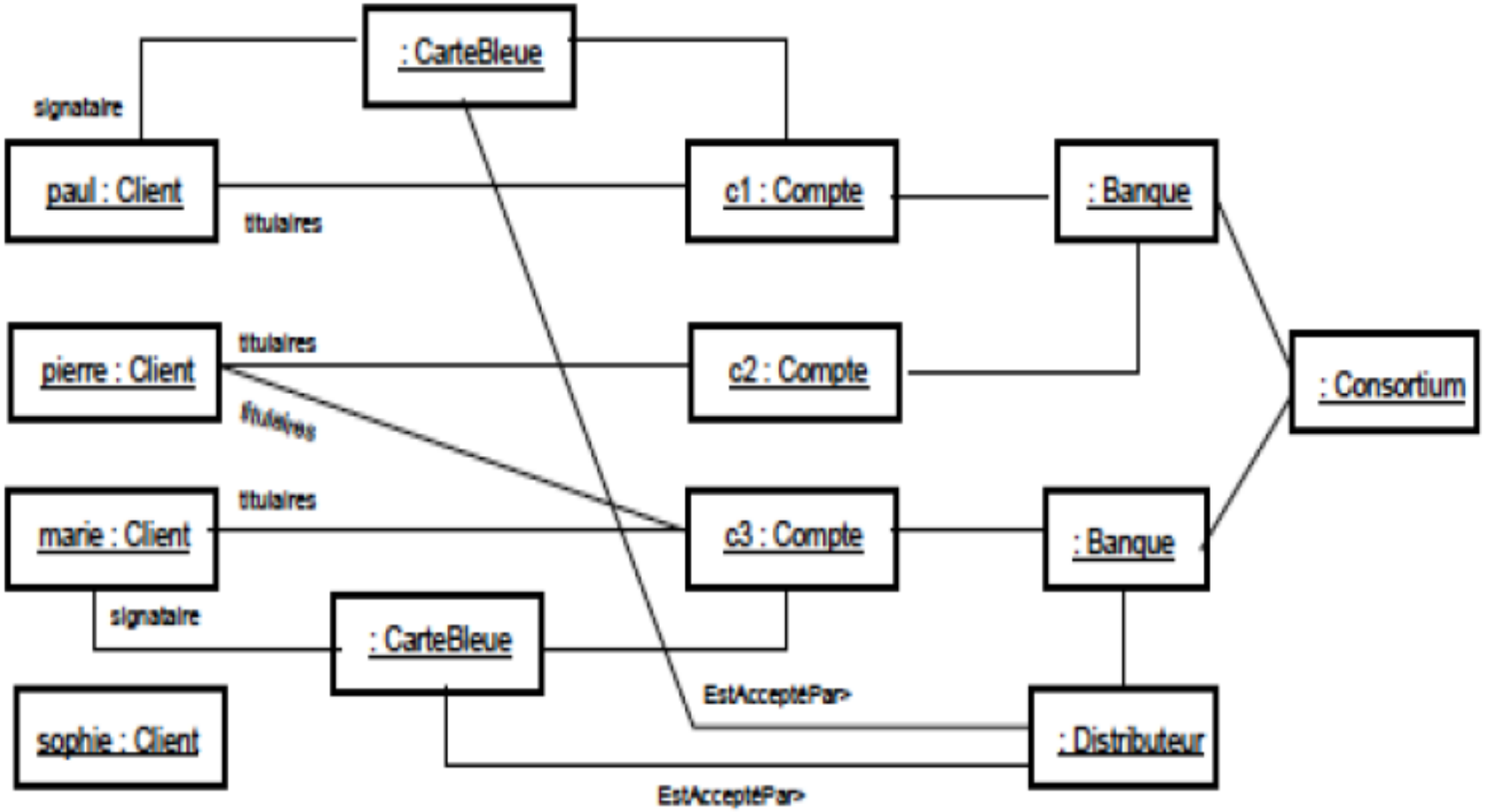


Diagramme 1: « Classes» pourquoi ces diagrammes?

Ce diagramme est une abstraction d'un ensemble de diagrammes d'objets

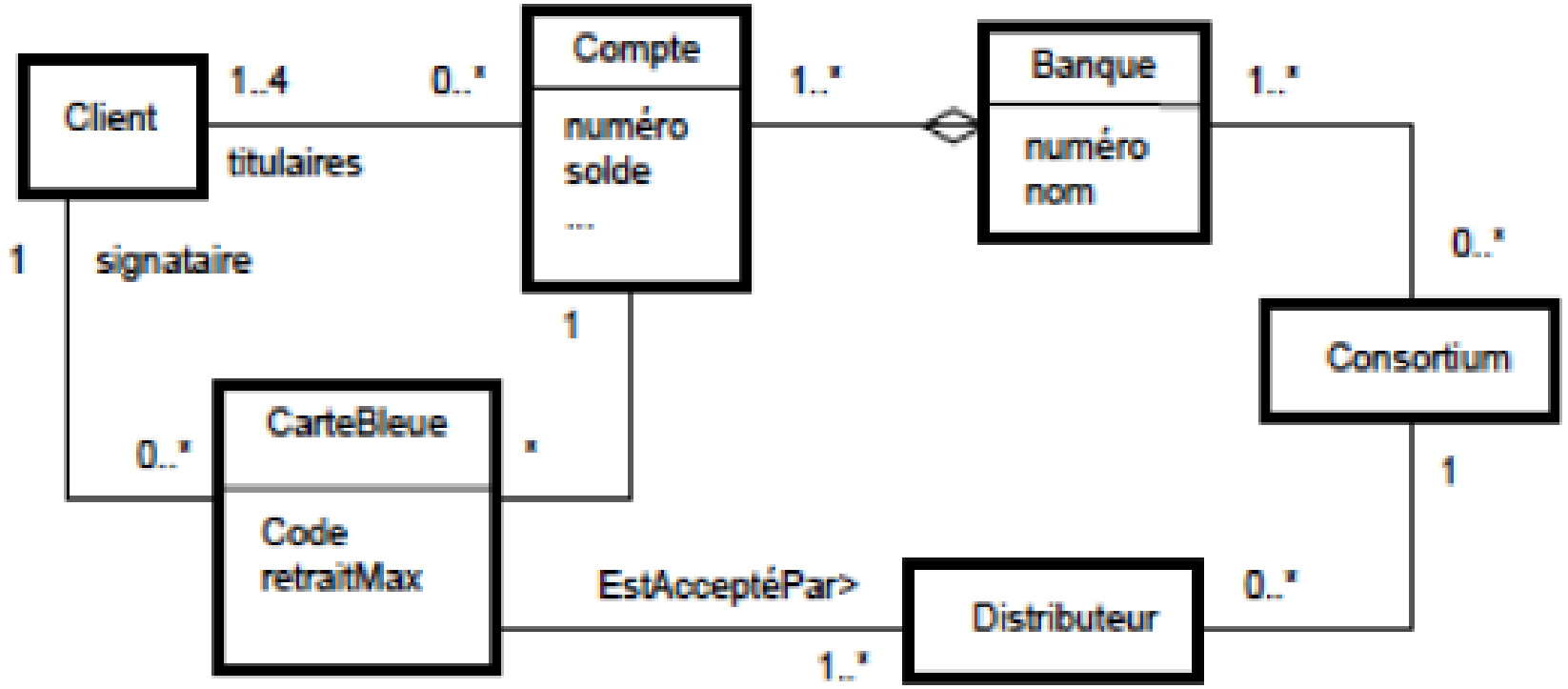


Diagramme 1: « Classes» pourquoi ces diagrammes?

Ce diagramme sert à:

- Montrer les **classes** du systèmes: **nom**, **propriétés**, **méthodes**, **modificateurs**

- Montrer les **relations** ou **associations** entre ces classes: **héritage**, **agrégation**, **composition**, **interaction**

Il peut être:

Documenté ou non (seulement le nom de la classe)

Diagramme 4: « Classes» représenter les classes

avec plus ou moins d'information

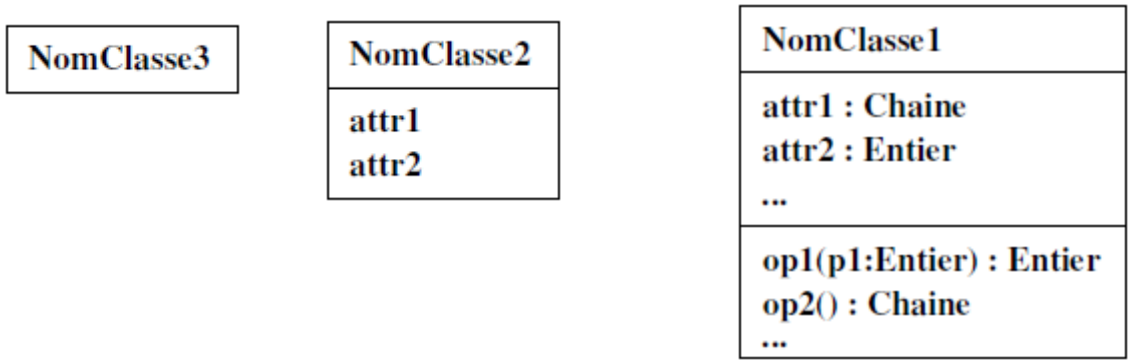


Diagramme 4: « Classes»
les attributs des classes

- Format de description d'un attribut :
  [Vis] Nom [Mult] [":" TypeAtt] ["=" Val] [Prop]

  - **Vis** : + (public), - (privé), # (protégé), (package)
  - **Mult** : "[" nbElt "]" ou "[" Min .. Max "]"
  - **TypeAtt** : type primitif (Entier, Chaîne, ...) ou classe
  - **Val** : valeur initiale à la création de l'objet
  - **Prop** : {gelé}, {variable}, {ajoutUniquement}, ...

- Attributs de classe (statiques) doivent être **soulignés**

- Attributs dérivés précédés de **"/"** : ces attributs peuvent devenir des méthodes par la suite.

Exemples :
- # onOff : Bouton - x : Réel coord[3] : Réel + pi : réel = 3.14 {gelé}
- inscrits[2..8] : Personne /age : Entier

Diagramme 4: « Classes»

les méthodes des classes

Format de description d'une opération :

[Visibilité] **Nom** **["(" Arg ")"]** **[":" Type]**

- **Visibilité** : + (public), - (privé), # (protégé)
- **Arg** : liste des arguments selon le format
  [Dir] NomArgument : TypeArgument
  où Dir = in (par défaut), out, ou inout
- **Type** : type de la valeur retournée (type primitif ou classe)

- Opérations abstraites/virtuelles (non implémentées) en ***italique***
- Opérations de classe (statiques) **soulignées**
- Possibilité d'annoter avec des contraintes stéréotypées
  «precondition» et «postcondition»
  ⇝ Programmation par contrats
- Possibilité de surcharger une opération :
  ⇝ même nom, mais paramètres différents
- Stéréotypes d'opérations : «create» et «destroy»

Diagramme 4: « Classes» Exemple1

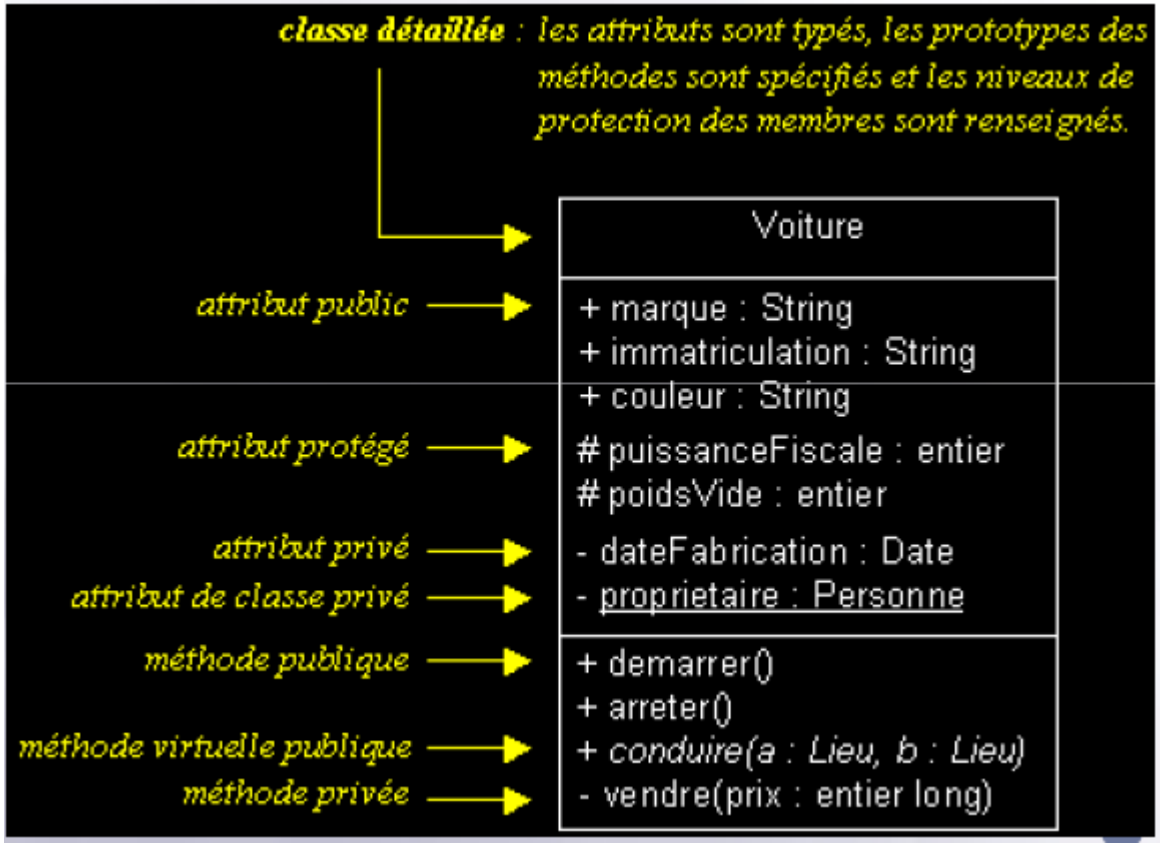
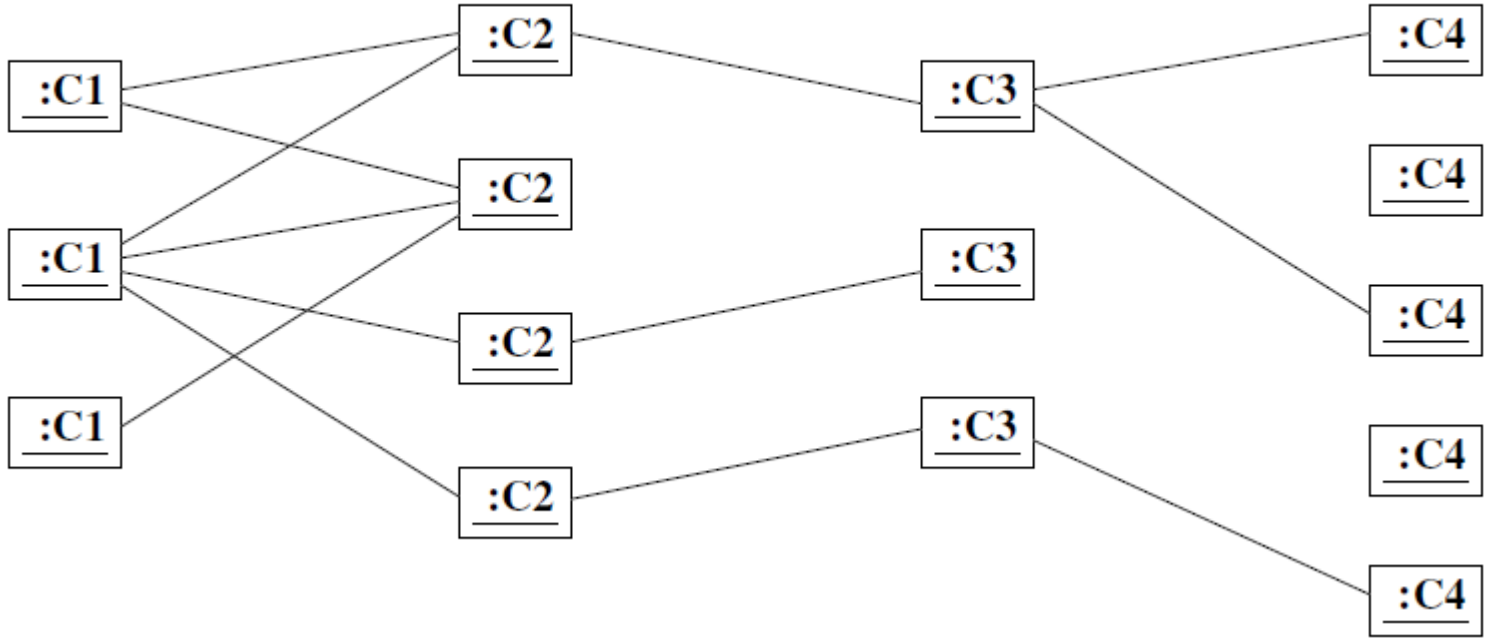


Diagramme 4: « Classes»

les associations entre classes

Liens entre objets :



Associations entre classes d'objets :

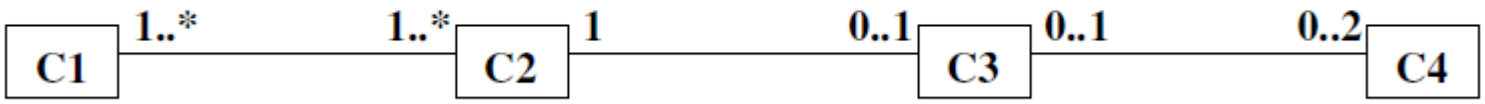
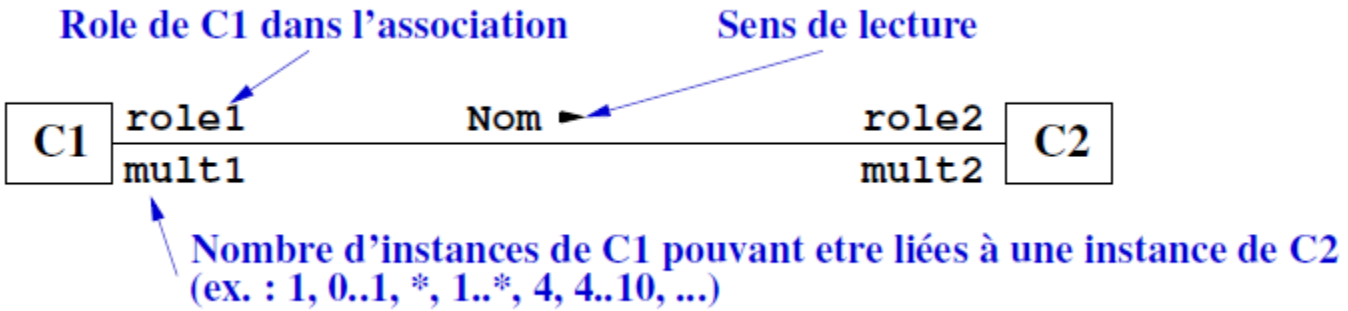
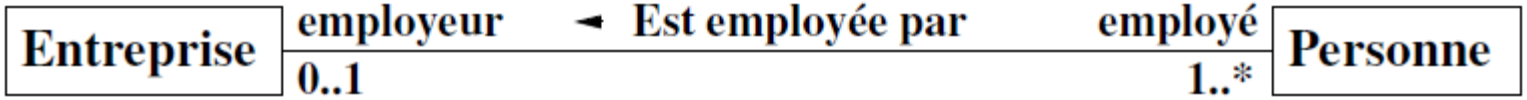


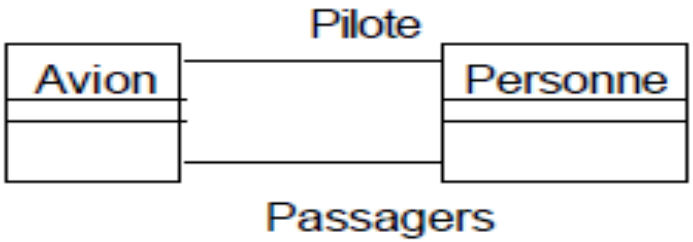
Diagramme 4: « Classes»

les associations entre classes



Exemple :





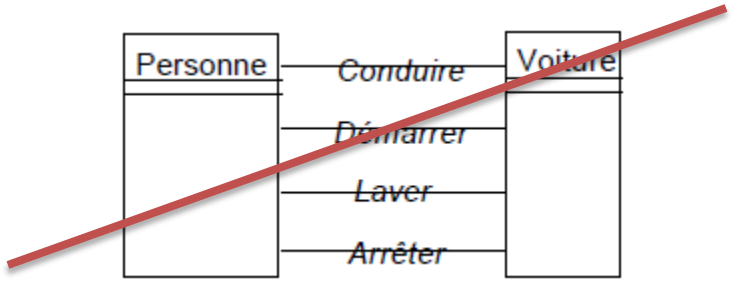
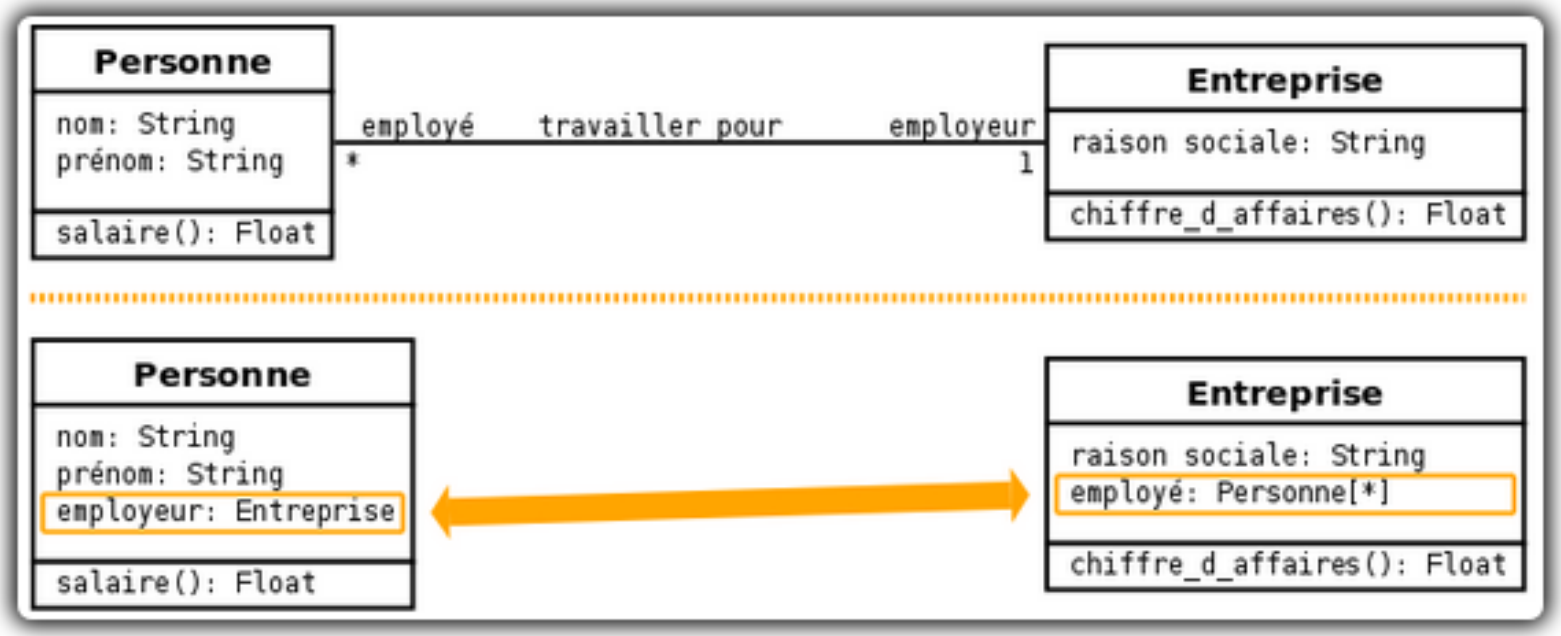


Diagramme 4: « Classes»

les associations entre classes: exemple



Deux façons de modéliser une association.

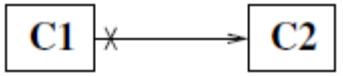
Diagramme 4: « Classes» navigabilité entre classes

Navigabilité d'une association entre C1 et C2 :

- Capacité d'une instance de C1 (resp. C2) à accéder aux instances de C2 (resp. C1)

- **Par défaut :** Navigabilité dans les deux sens. C1 a un attribut de type C2 et C2 a un attribut de type C1

- **Spécification de la navigabilité : Orientation de l'association:** C1 a un attribut du type de C2, mais pas l'inverse



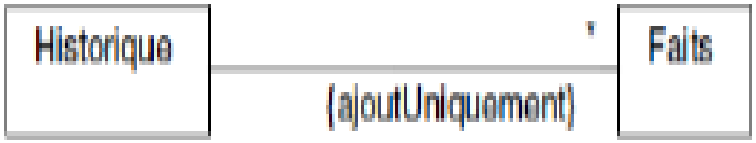
- **Attention :**

Dans un diagramme de classes conceptuelles, toute classe doit être accessible à partir de la classe principale

Diagramme 4: « Classes»

contraintes sur les associations

- Contraintes sur **une extrémité** de l'association: {ordered}, {variable}, {frozen} , {addOnly}



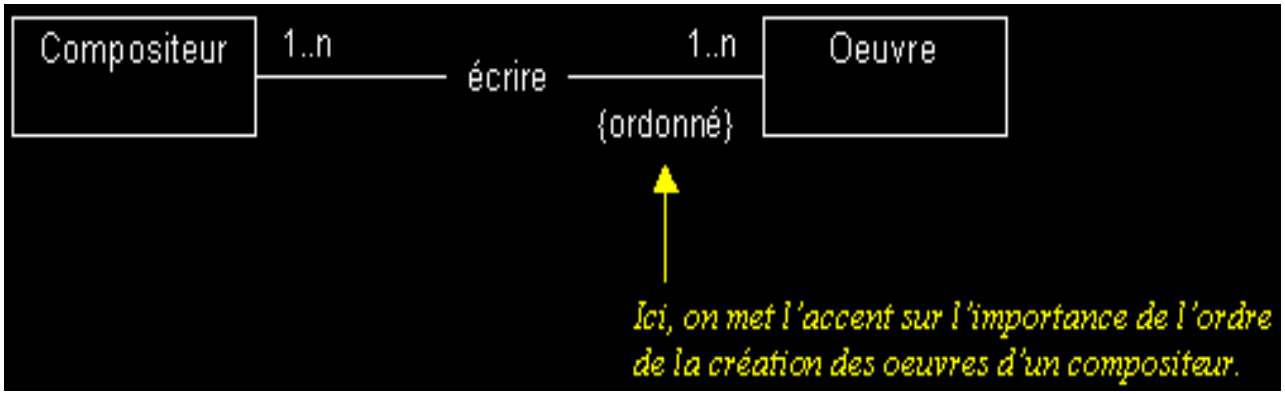


Diagramme 4: « Classes» contraintes sur les associations

- La variabilité (*changeability) indique si des* ***données peuvent être mises à jour*** *(ajoutées, modifiées, supprimées) ou non. Par* défaut, une donnée peut être mise à jour.

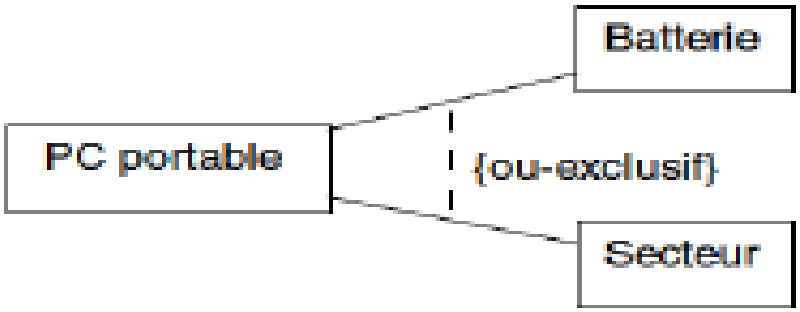
- la contrainte {frozen} indique qu'une donnée, une fois créée, **ne peut plus être mise à jour.**

- La contrainte {addOnly} indique **qu'on ne peut qu'ajouter des données**.

Diagramme 4: « Classes»

contraintes sur les associations (2)

- Contrainte entre deux associations: {subset}, {xor}, ...



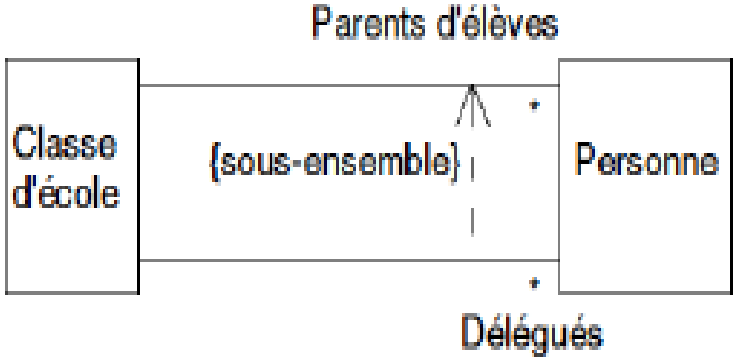
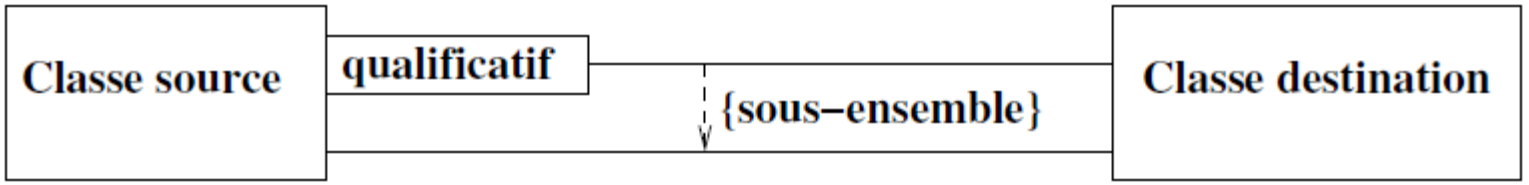


Diagramme 4: « Classes»
contraintes sur les associations (3)

- Il est possible que les cardinalités changent suite à une restriction (qualification)



Exemple:

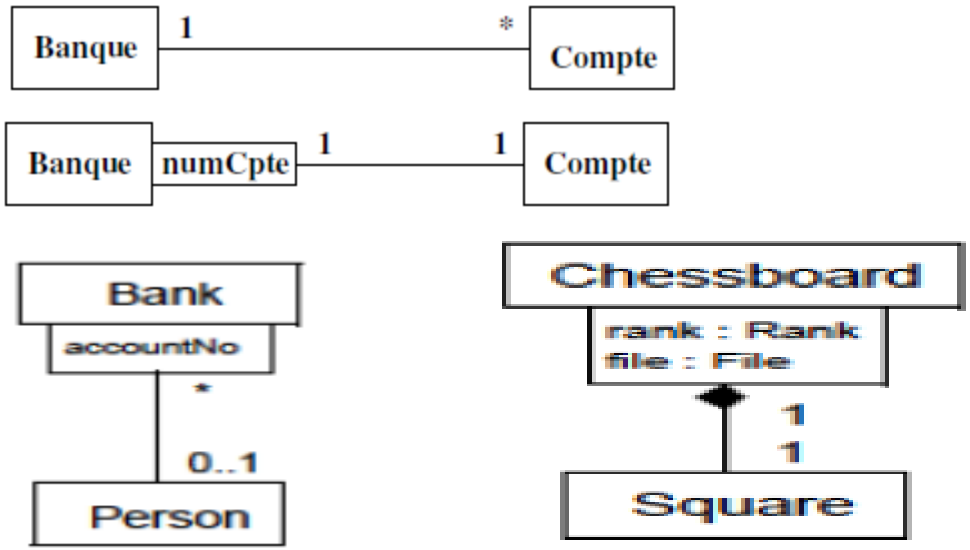
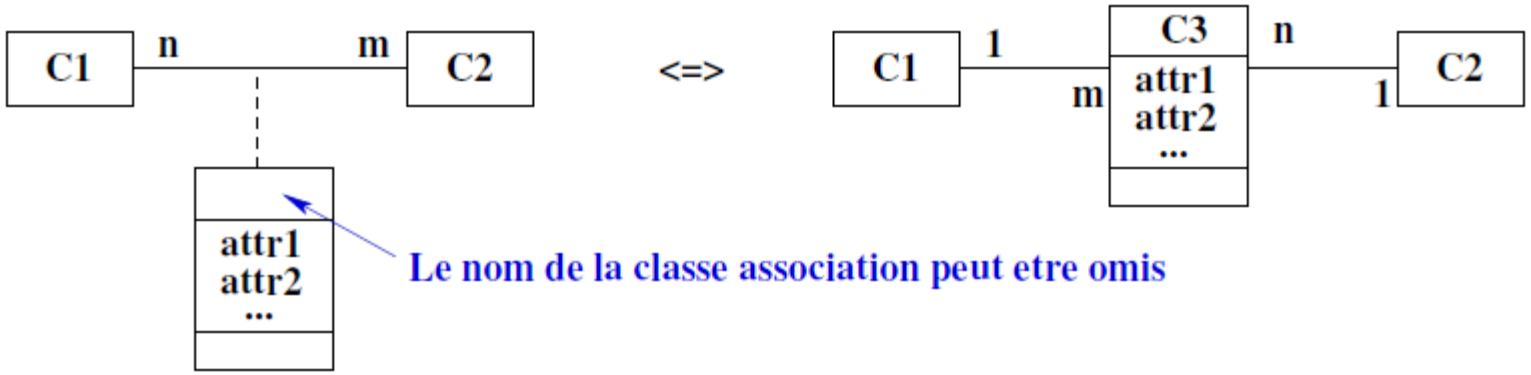


Diagramme 4: « Classes»

Classe Association (1)

- Une association peut devenir une classe



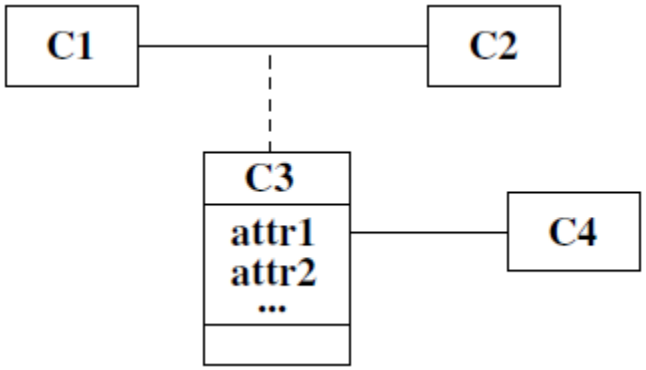


Diagramme 4: « Classes»

Classe Association (2): exemple

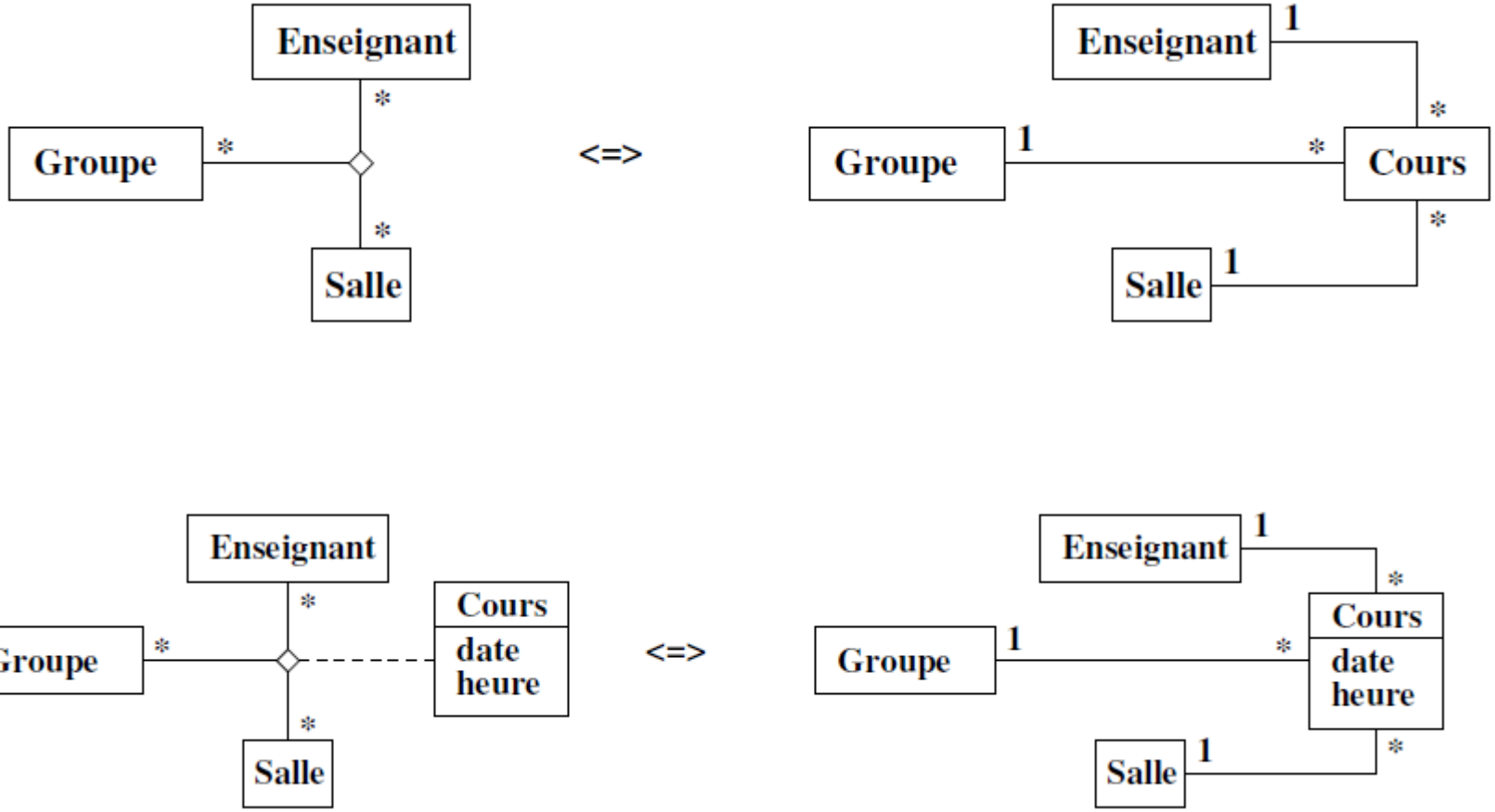


Diagramme 4: « Classes»

Associations particulières: composition & agrégation

- Composition:
- Relation transitive et antisymétrique
- La création (copie, destruction) du composite (container) implique la création (copie, destruction) de ses composants
- Un composant appartient à au plus un composite

- Agrégation:

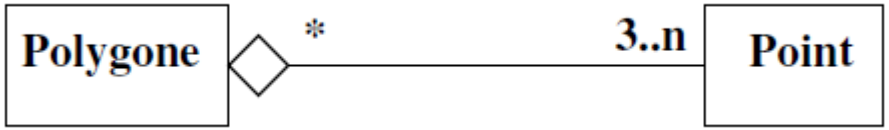


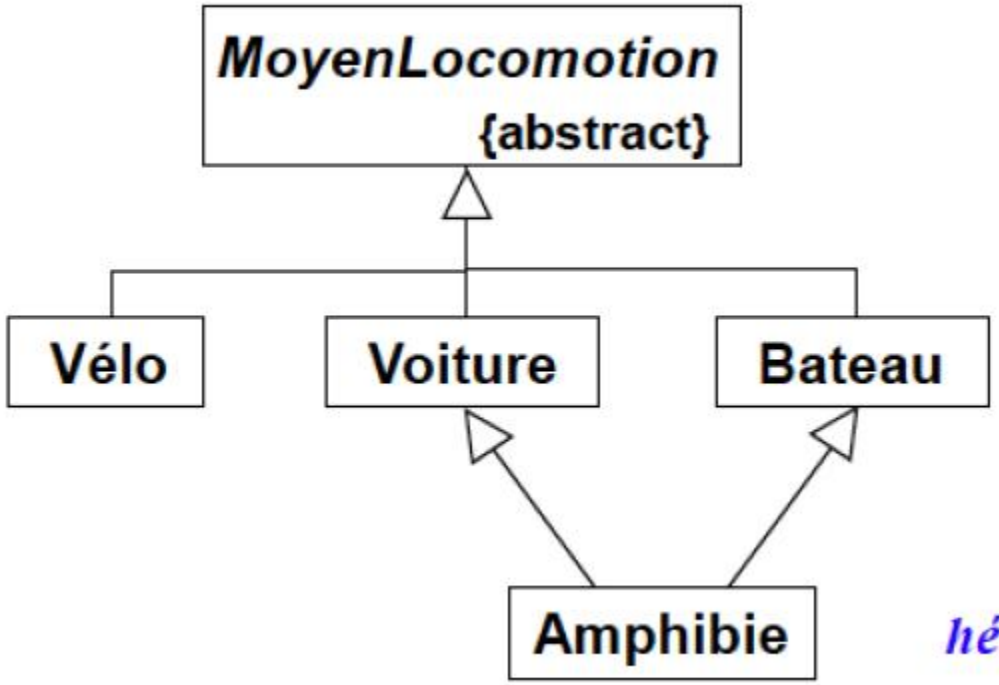
Diagramme 4: « Classes»

Association particulières: héritage

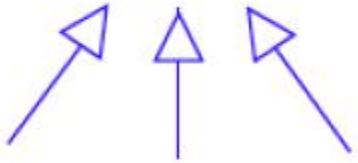
Mécanisme proposé par les langages de programmation Objet: "B hérite de A" signifie que B possède :

1. Toutes les propriétés de A (attributs, op., assoc., contraintes) avec:
   - •Possibilité de redéfinir les opérations de la sous-classe
   - •Polymorphisme
2. Ainsi que des nouvelles propriétés qui lui sont propres

Diagramme 4: « Classes» héritage: exemple



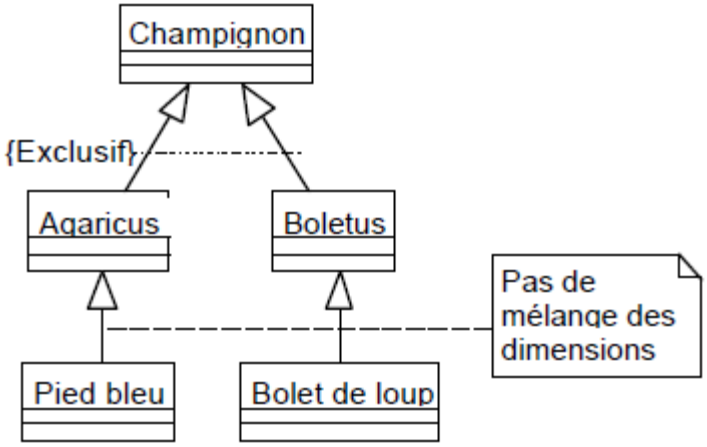
notation équivalente



héritage multiple

Diagramme 4: « Classes»
Héritage: contraintes

- {disjoint} ou {overlapping}: disjoint A descend de B, donc A ne peut avoir comme **mère qu'une seule classe**



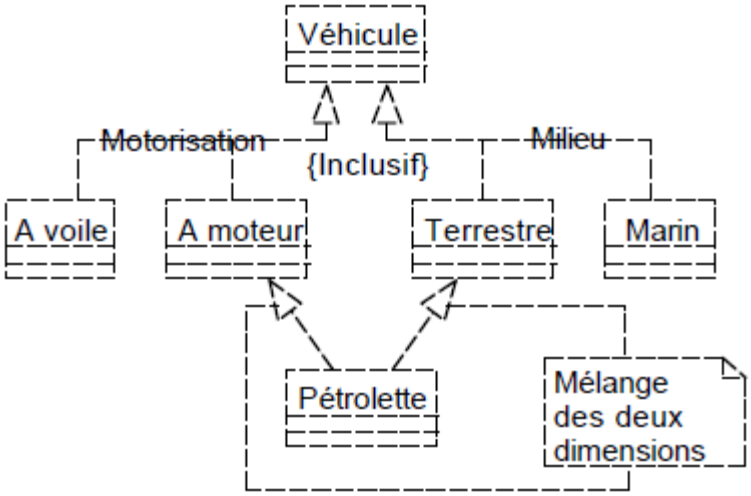
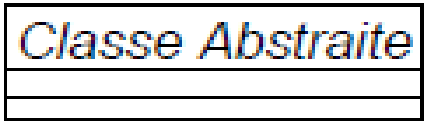


Diagramme 4: « Classes» classes abstraites

- Classe qui ne peut être instanciée,
- Ne servent pas à créer des objets,
- Contient des opérations non définies: abstract (Java) ou virtual (C++)
- Doit être spécialisée en une ou plusieurs classes non abstraites
- Notation : mot clé {abstract} ou nom en italique



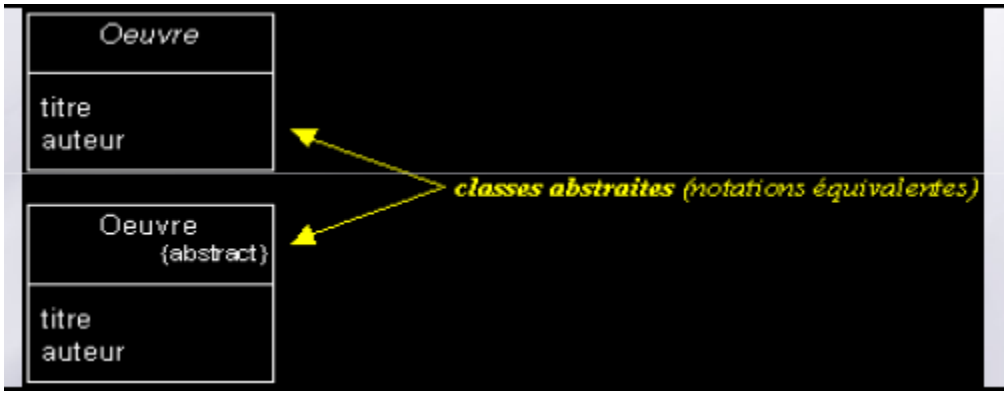


Diagramme 4: « Classes» Interface

Qu'est-ce qu'une interface ?

Classe sans attribut dont toutes les opérations sont abstraites

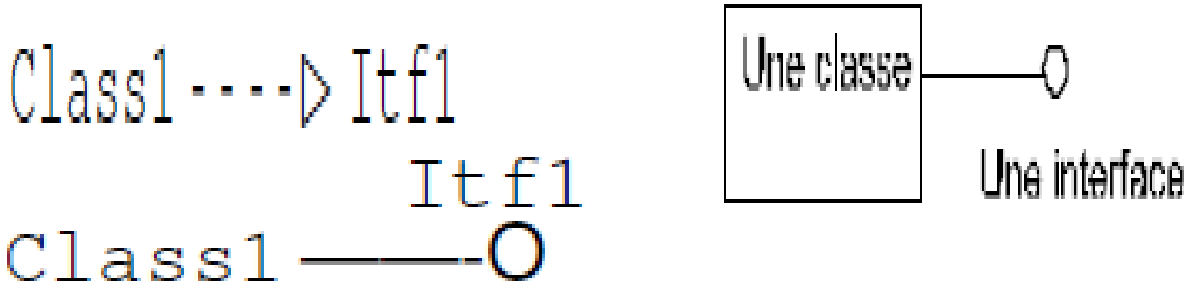
- Ne peut être instanciée,
- Doit être réalisée (implémentée) par des classes non abstraites,
- Peut hériter d'une autre interface

Pourquoi des interfaces ?

- Utilisation similaire aux classes abstraites
- En Java : une classe ne peut hériter que de plus d'une classe, mais elle peut réaliser plusieurs interfaces

Diagramme 4: « Classes» Interface: modélisation (1)

- Implémentation :



- Utilisation:

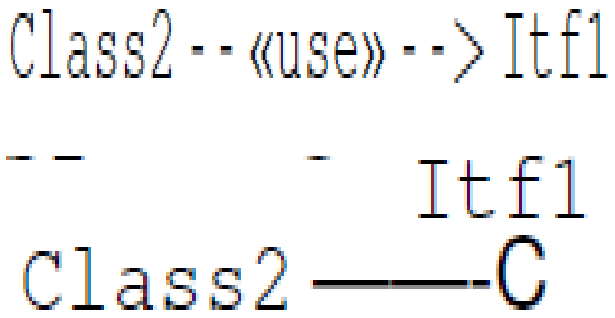


Diagramme 4: « Classes» Interface: modélisation (2)

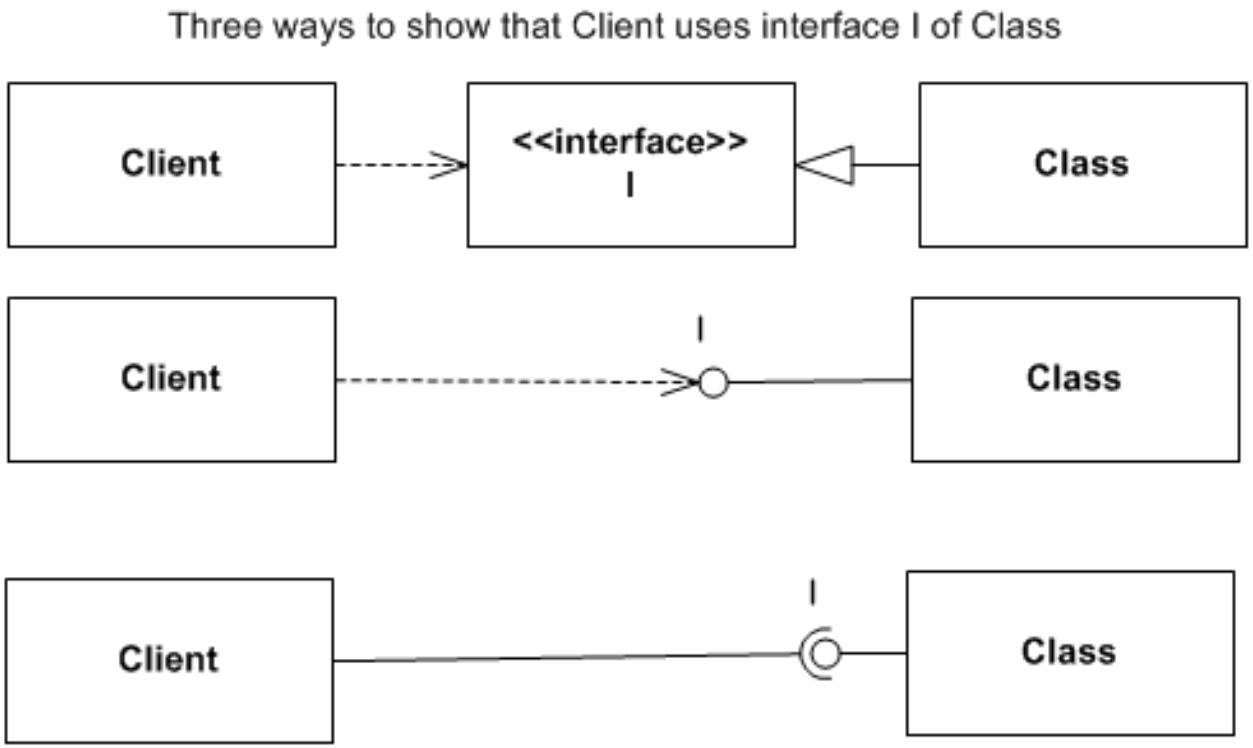
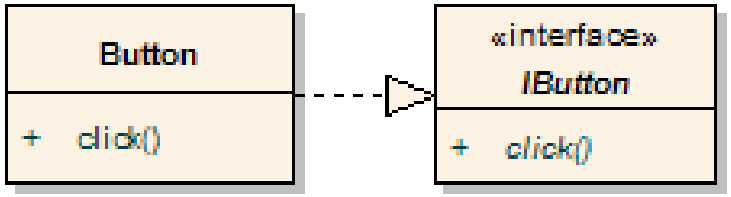


Diagramme 4: « Classes»
Interface: Exemples




```
interface IButton {
        function click();
}

class NiceButton inplements IButton {
        function click() {
                ...
        }
}
```

Diagramme 4: « Classes»
Interface: Exemples

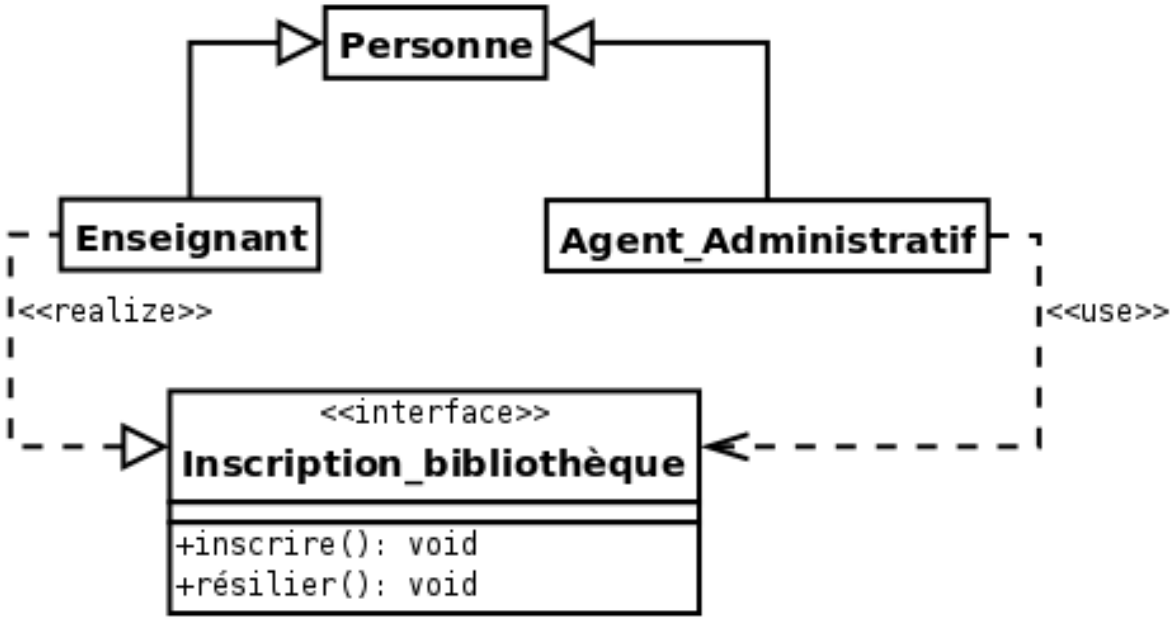


Diagramme 4: « Classes»
Interface: Exemples