Chapitre 2 :

Le langage UML pour la modélisation statique

1. Diagrammes de classes
2. Diagrammes d'objets

Phases de modélisation

- Expression des besoins : s'accorder sur ce qui doit être fait dans le système
- Analyse : Comprendre les besoins et les décrire dans le système (quoi)
- Conception : s'accorder sur la manière dont le système doit être construit (comment)
- Implémentation : Codage du résultat de la conception
- Test : Le système est il conforme au cahier des charges?

Diagramme de classes

- Le diagramme de classes représente la structure statique d'un système
  - les classes du système
  - les associations entre les classes
  - les attributs et les opérations qui caractérisent chaque classe
- Le diagramme de classes est le principal diagramme
  - il est construit à partir des objets du monde réel (univers du discours)
  - il est le canevas pour les autres diagrammes

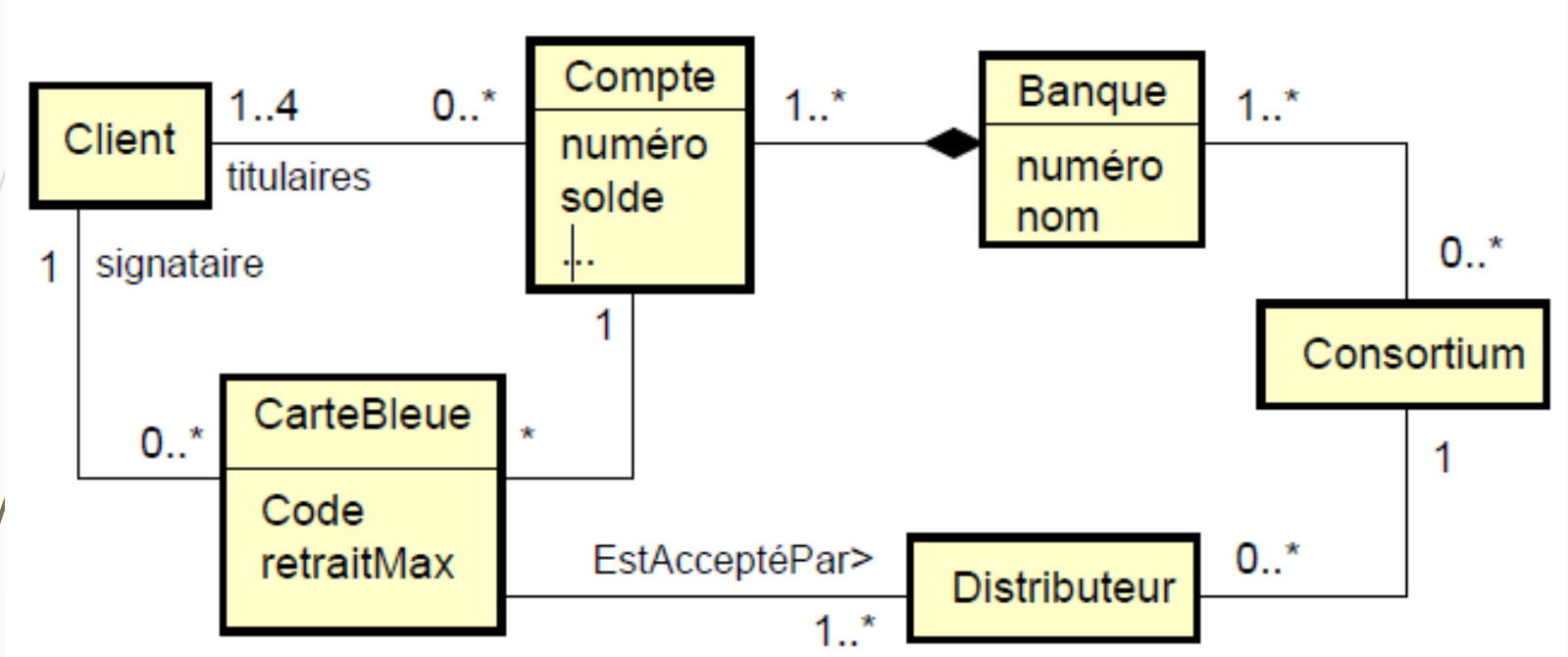
Diagramme de classes

- Le plus connu, le plus utilisé
  - Est le point central dans un développement orienté objet
- Il est utilisé dans les deux étapes de la cycle de vie d'un logiciel : analyse et conception

Diagramme de classes

- La construction du diagramme de classes commence par identifier les concepts ou les entités utilisés dans le domaine de l'application.
  - Les modéliser ensuite en tant que classes
  - Exemple de classes dans les systèmes bancaires : Banque, Clients, Employés, Comptes, cartes bleues, distributeurs, opérations bancaires, …
- Pour chaque concept, identifier les propriétés qui le caractérisent.
- Ensuite, identifier toutes les relations ou associations pertinentes entre les différents concepts.

Diagramme de classes

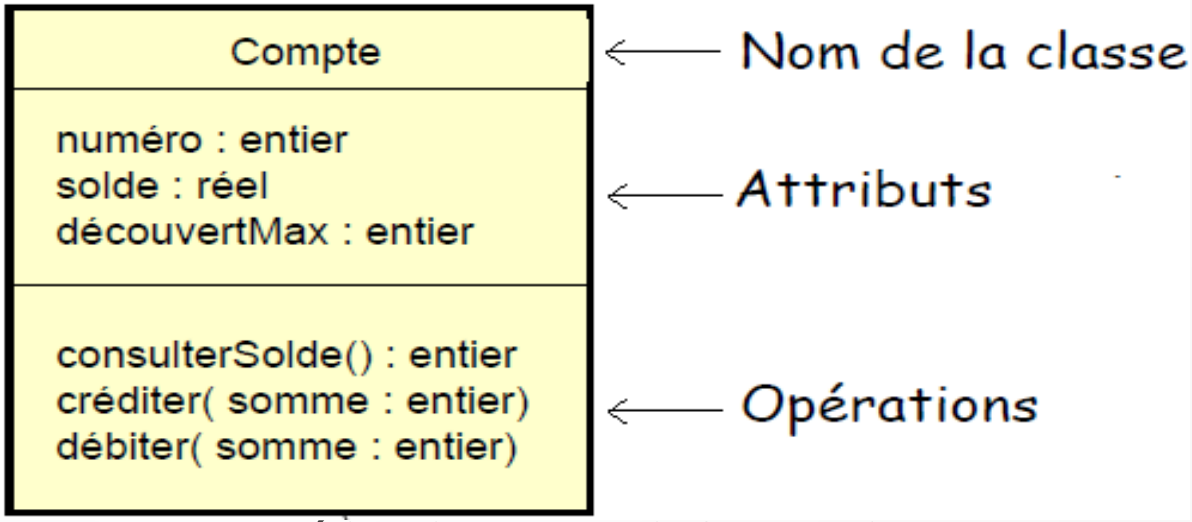


Classes

- Une classe est la description d'un ensemble d'objets ayant une structure commune (les même attributs) et un comportement commun (les même méthodes).

- Une classe est composée d'un nom, d'attributs et d'opérations.

- En UML, une classe est représentée par un rectangle avec 3 compartiments.



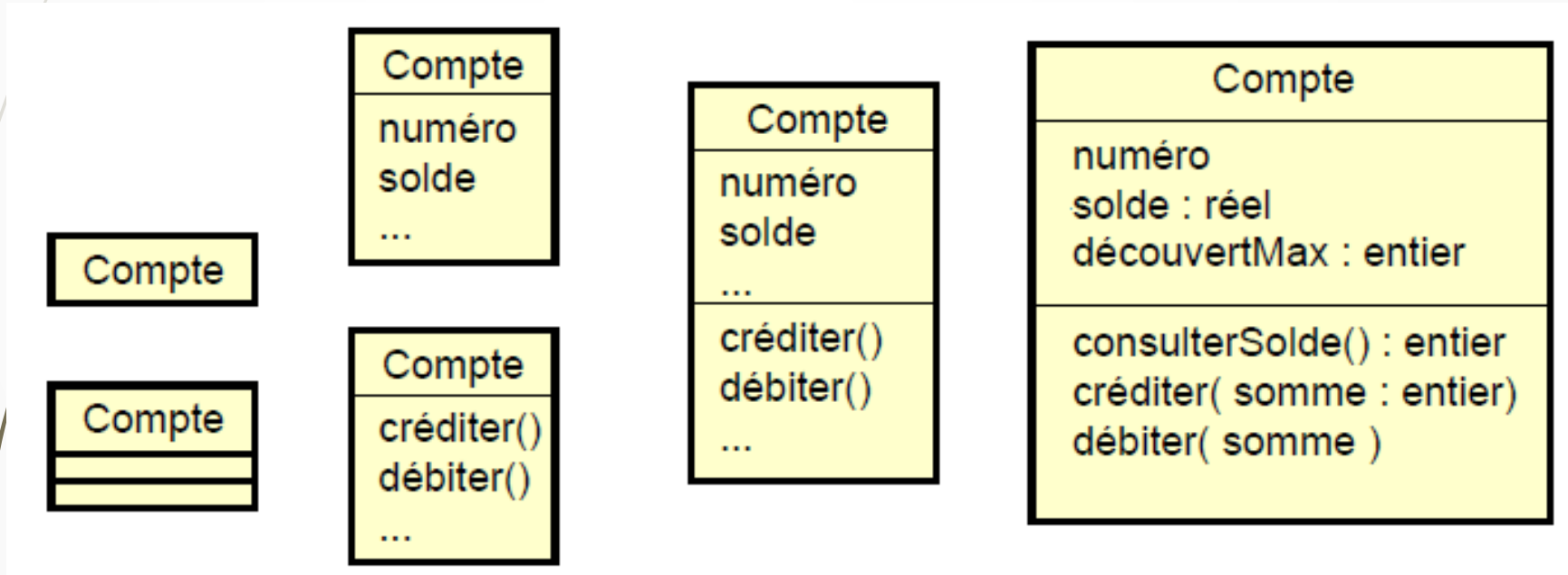
- Selon l'avancement de la modélisation, ces informations ne sont pas forcement toutes connues.

Niveaux de détails

- Plusieurs niveaux de détails sont possibles : Analyse et conception
- En analyse, dans le diagramme de classes on peut commencer par des classes non complètes et ensuite on les enrichit au fur et à mesure.

- Exemples :
  - Niveau de détail d'analyse
    - simplifié
      - Uniquement le nom de la classe
    - intermédiaire
      - Nom de la classe
      - Nom des attributs, ou des opérations
    - Complet
      - Nom de la classe
      - Noms des attributs
      - Noms des méthodes

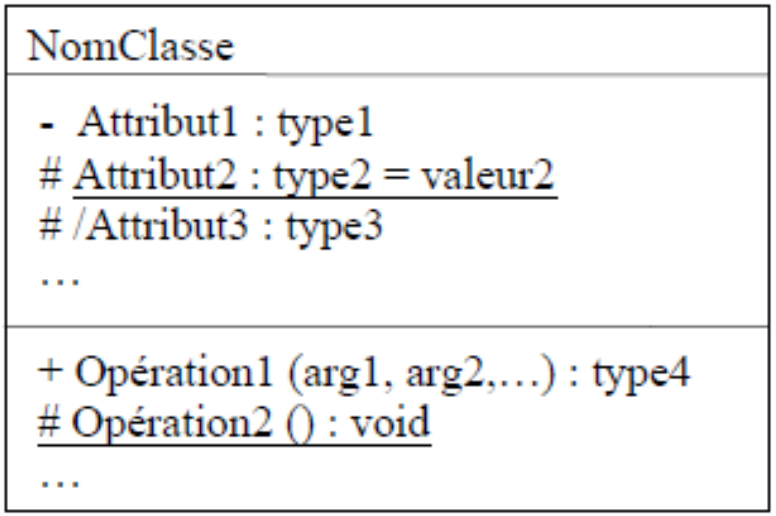
Notations simplifiées pour les classes



Niveaux de détails

- En conception, le diagramme de classes est représenté à un niveau de détail plus important, s'approchant ainsi de la structure d'un code orienté objet.

- Niveau de détail de conception
  - Attributs :
    - Type
    - Valeurs par défaut
    - Degré de visibilité
  - Opérations :
    - Signature
    - Degré de visibilité

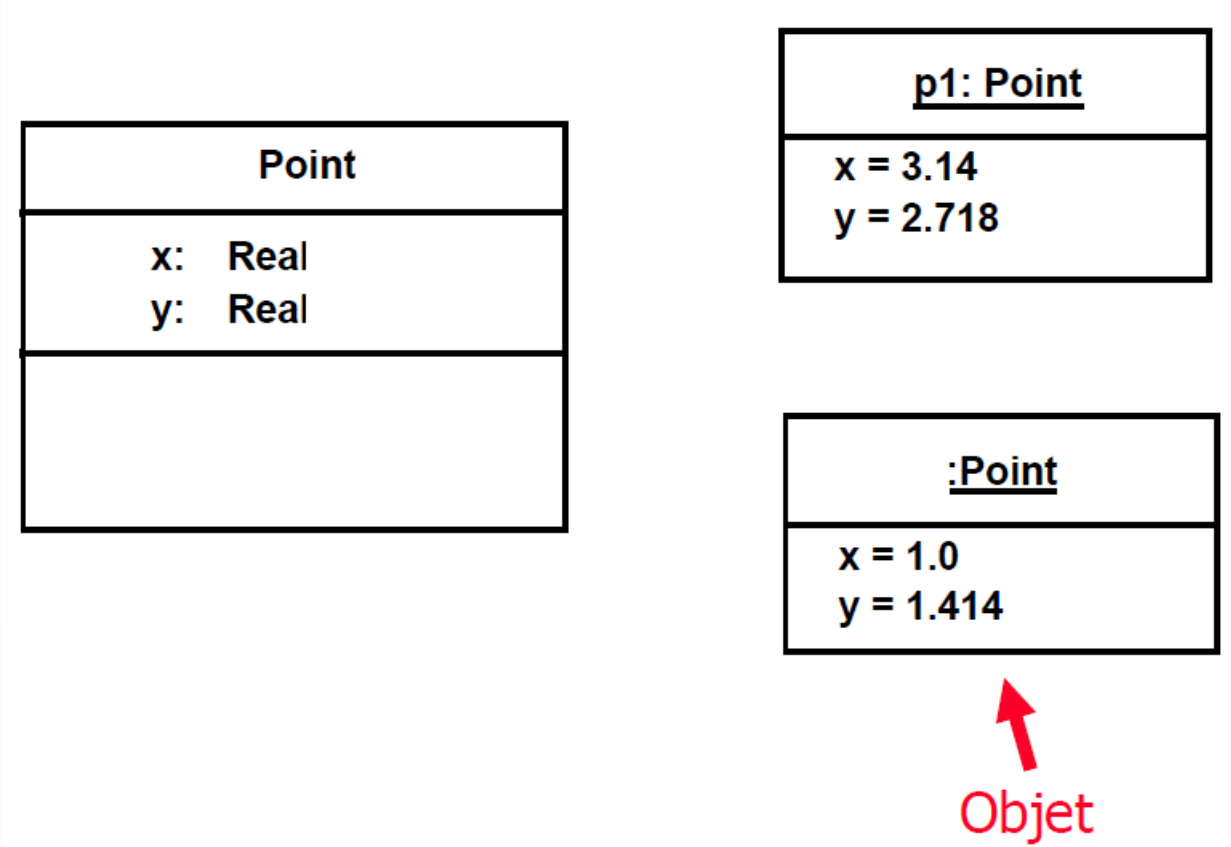


Propriétés : attributs et opérations

- Les attributs et les opérations sont les propriétés d'une classe.
- Un attribut décrit une donnée de la classe.
  - Les types des attributs et leurs initialisations ainsi que les visibilités peuvent être précisés dans le modèle.
  - Les attributs prennent des valeurs lorsque la classe est instanciée.
- Une opération est un service offert par la classe. Un traitement que les objets correspondant peuvent effectuer.

Objets (instances de classe)

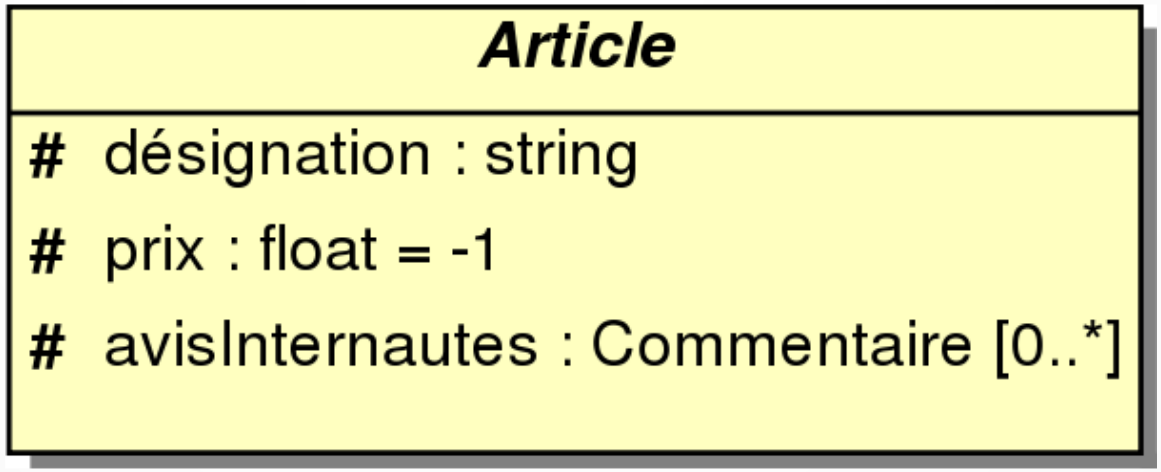
- Un objet est une instance d'une classe



Compartiment des attributs

- Un attribut peut être initialisé et sa visibilité est définie lors de sa déclaration.
- Syntaxe de la déclaration d'un attribut :

  visibilité **nomAttribut**: nomClasse [multi]= valeurInitiale

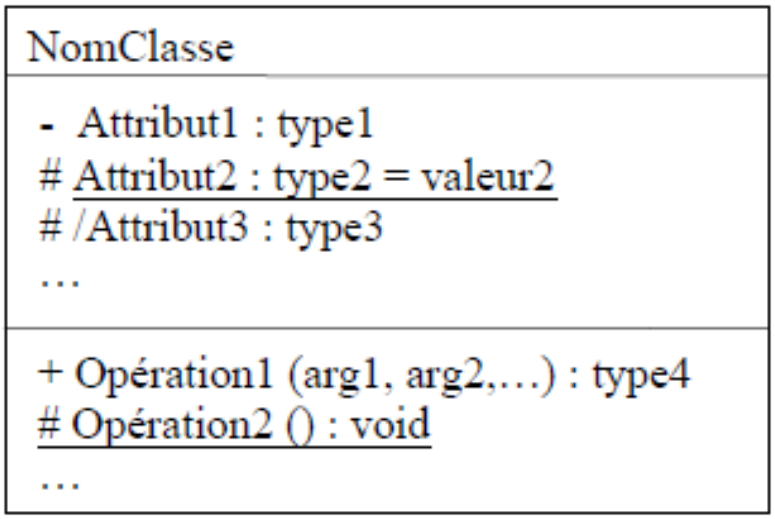


Visibilité

Visibilité = degré de protection

- + : publique (accessible à toutes les classes de l'application)
- # : protégé (accessibles uniquement aux sous-classes)
- ~ : paquetage (accessible uniquement aux classes du même paquetage)
- - : privé (inaccessible à tout objet hors de la classe)

  - Utile lors de la conception et de l'implémentation, pas avant !
  - N'a pas de sens dans un modèle conceptuel (abstrait)

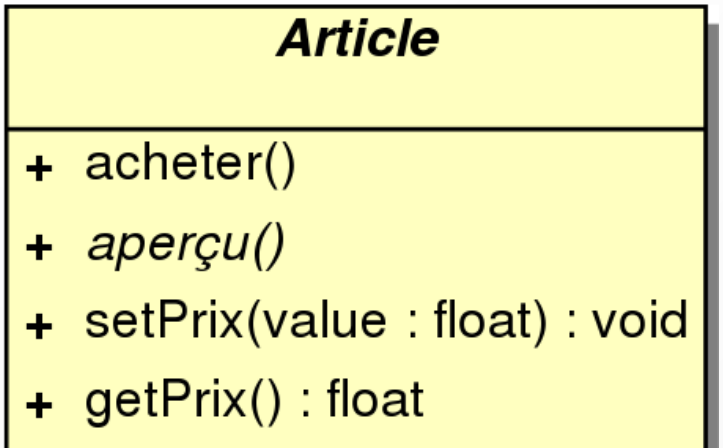


Compartiment des opérations

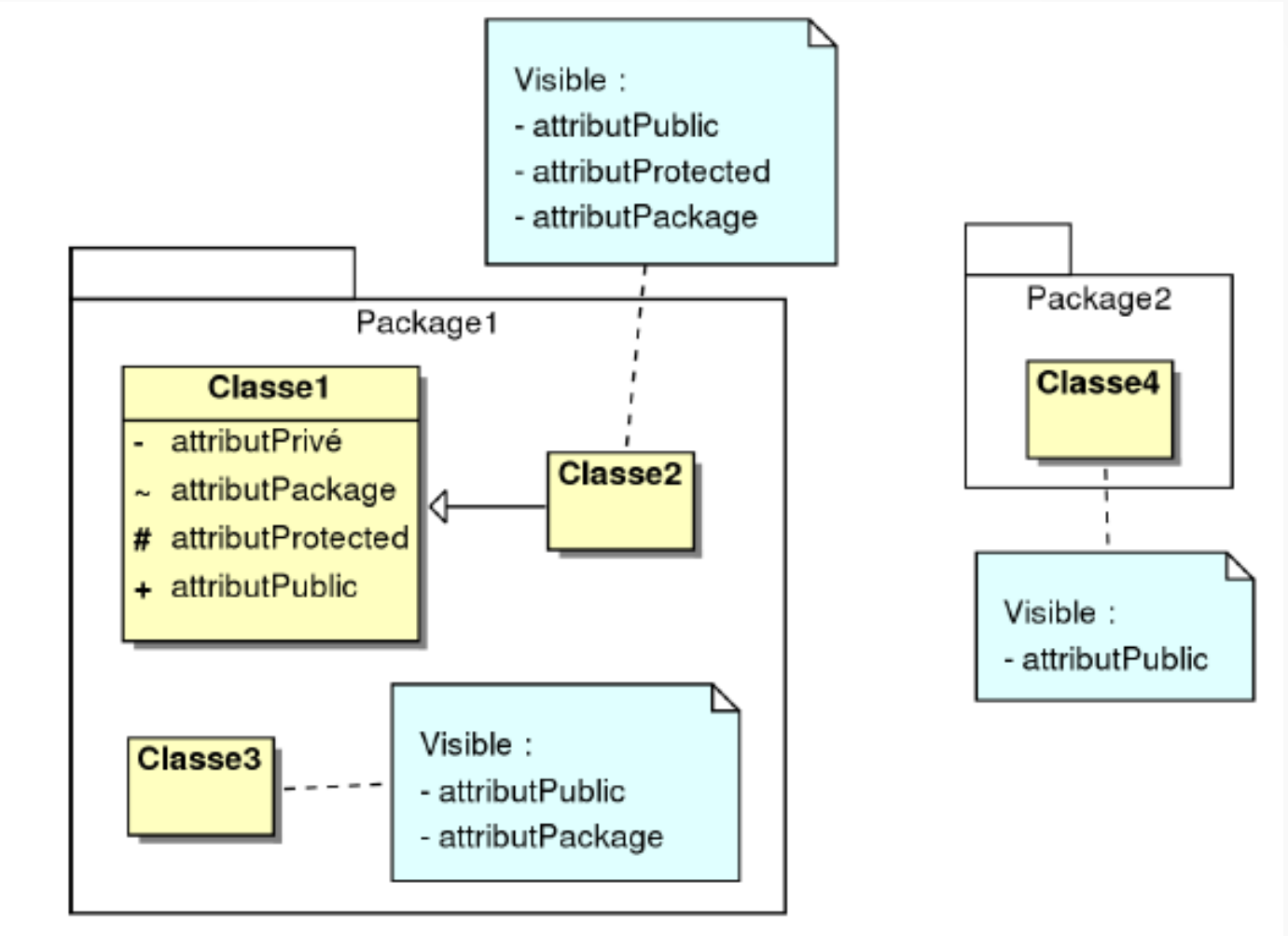
- Une opération est définie par son nom ainsi que par les types de ses paramètres et le type de sa valeur de retour.
- La syntaxe de la déclaration d'une opération est la suivante :

  Visibilité **nomOperation** (parametres): TypeRetour
- La syntaxe de la liste des paramètres est la suivante :

  nomParam1 : nomType1, ... ,nomParamN : nomTypeN



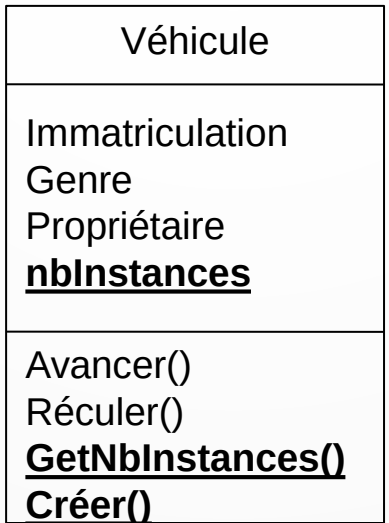
Exemple d'encapsulation



- Les packages : Permettent de structurer le modèle de l'application en le divisant en plusieurs packages contenant des classes, …
- Ils sont particulièrement utiles dès que le modèle comporte de nombreuses classes et que celles-ci peuvent être triées selon plusieurs aspects structurants.

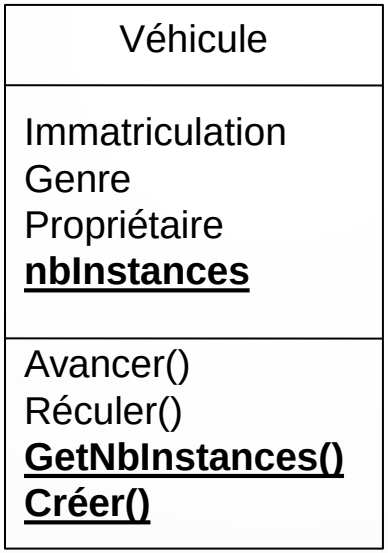
Attributs de classe

- Par défaut, les valeurs des attributs définis dans une classe diffèrent d'un objet à un autre.
  - Chaque objet a ses propres valeurs
- Parfois, il est nécessaire de définir un attribut de classe qui garde une valeur unique et partagée par toutes les instances.
- Graphiquement, un attribut de classe est souligné



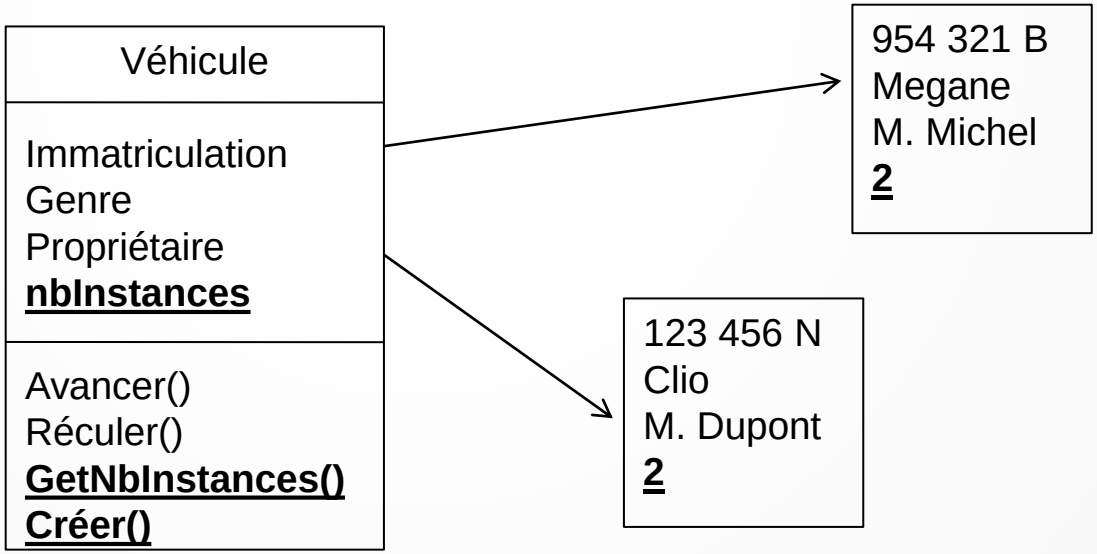
Opérations de classe

- Semblable aux attributs de classe
- Une opération de classe est une propriété de la classe, et non de ses instances.
- Elle n'a pas accès aux attributs des objets de la classe.



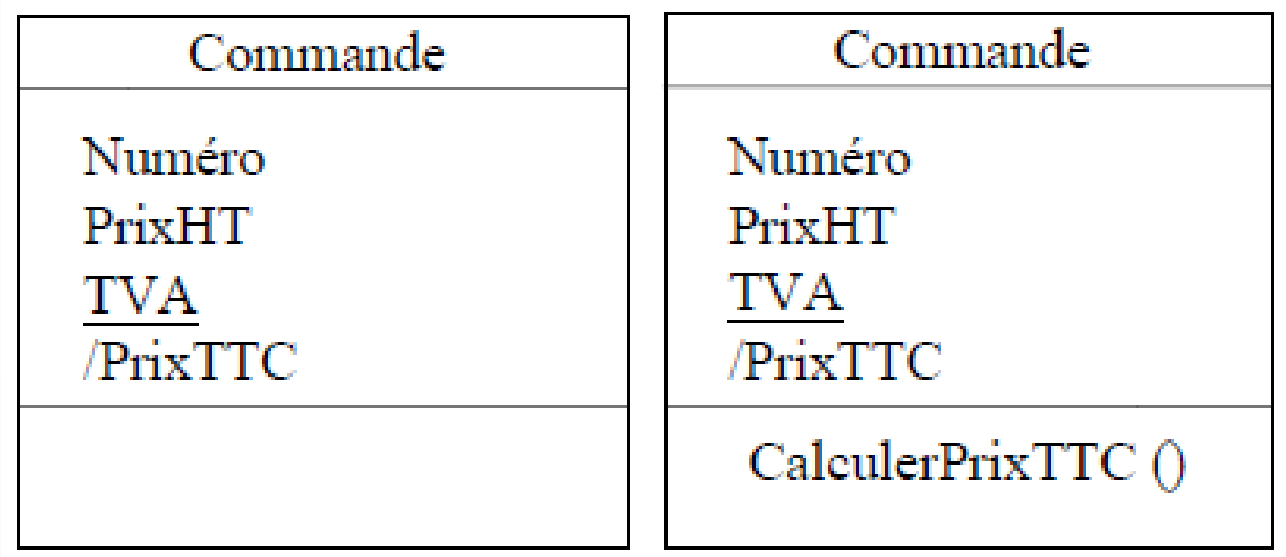
Variable et méthode de classe

- Les instances de la classe *Véhicule* partagent la même valeur de la variable de classe
- Pour les autres attributs, chaque objet a ses propres valeurs.



Attributs dérivés

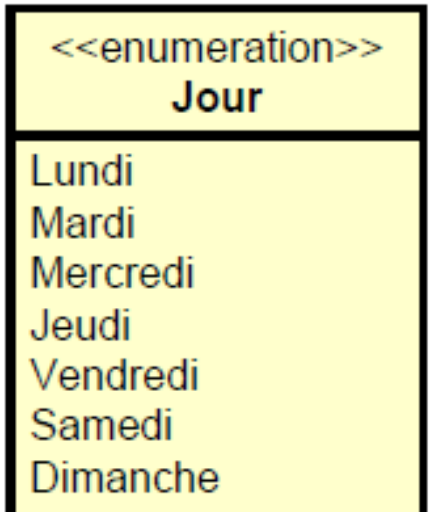
- Notation : /nomAttribut
- Propriété redondante, dérivée d'autres propriétés déjà allouées
- En conception, un attribut dérivé peut donner lieu à une opération qui encapsulera le calcul effectué

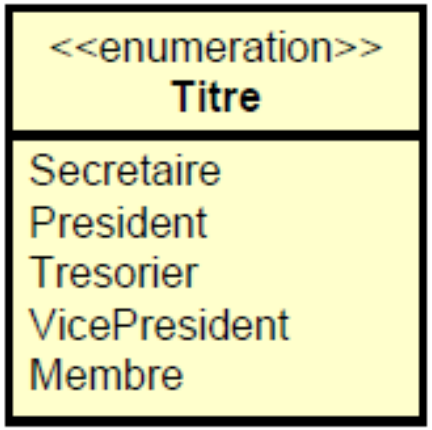


Classes particulières : Enumération

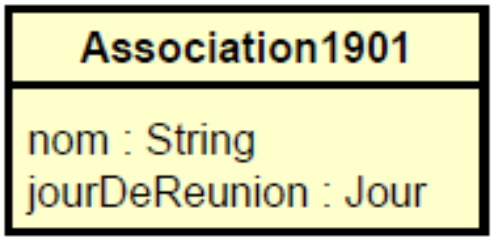
- Une énumération est un nouveau type de données
- Une énumération est un ensemble fini et constante de valeurs
- Les valeurs sont appelées litéraux
- C'est une classe définie en ajoutant le stéréotype « enumeration » avant le nom de la classe

- Définition





- Utilisation

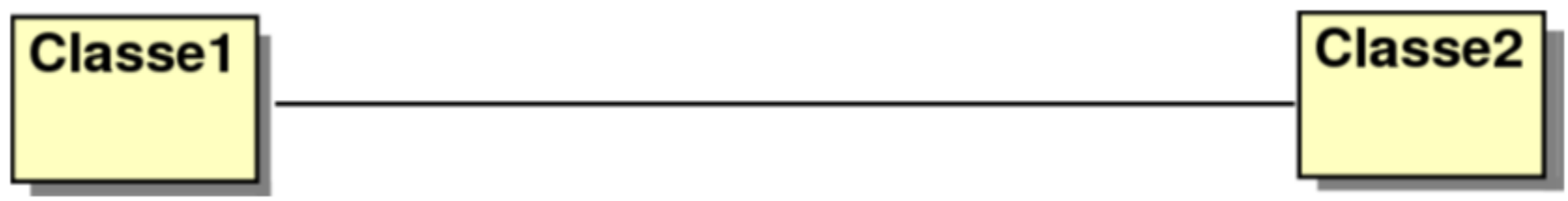


Les relations entre classes

- L'association
- L'agrégation
- La composition
- La généralisation

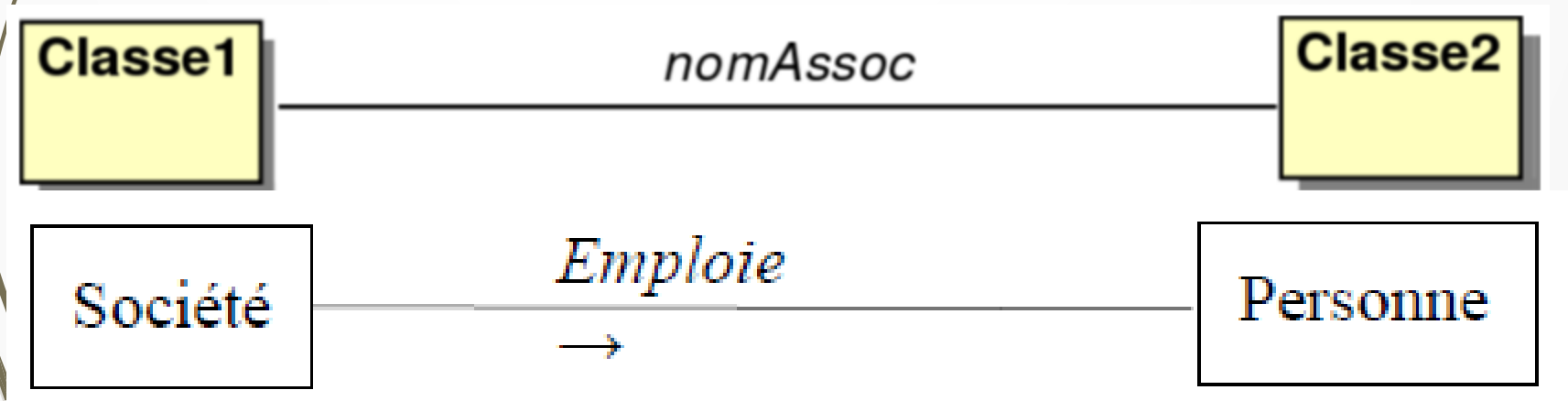
Association

- Une association est une relation sémantique et structurelle entre deux (ou plusieurs) classes.
- Elle est représentée par une ligne continue entre classes.
- La plupart des associations sont binaires : elles connectent 2 classes.
- Entre deux classes on peut avoir plusieurs associations
- Notation :



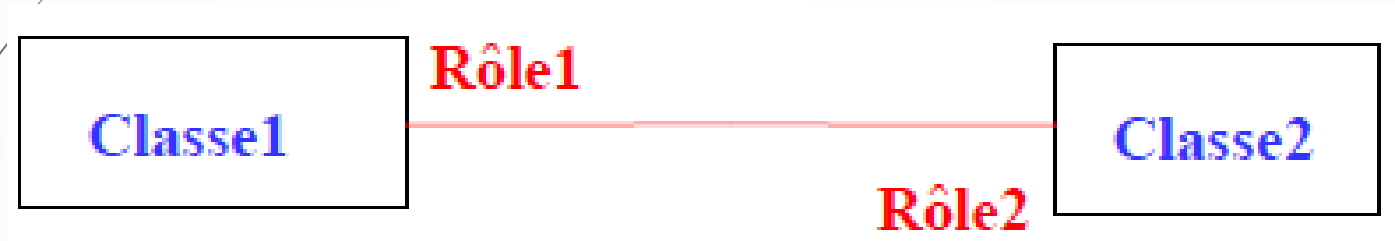
Nommage des associations

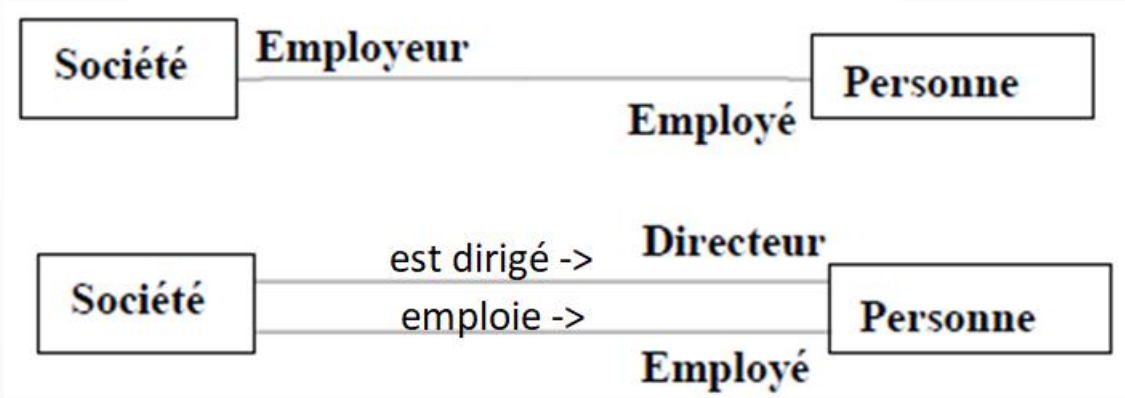
- Une association est désignée par un nom ou par deux rôles
- Une association peut se lire dans les 2 sens (bidirectionnelle), en fonction des besoins
- Indication possible du sens de lecture de l'association
- Usage : forme verbale, active ou passive
- Surtout utilisé dans les modèles d'analyse (en conception, on utilise plutôt le nommage des rôles)



Rôles dans les associations

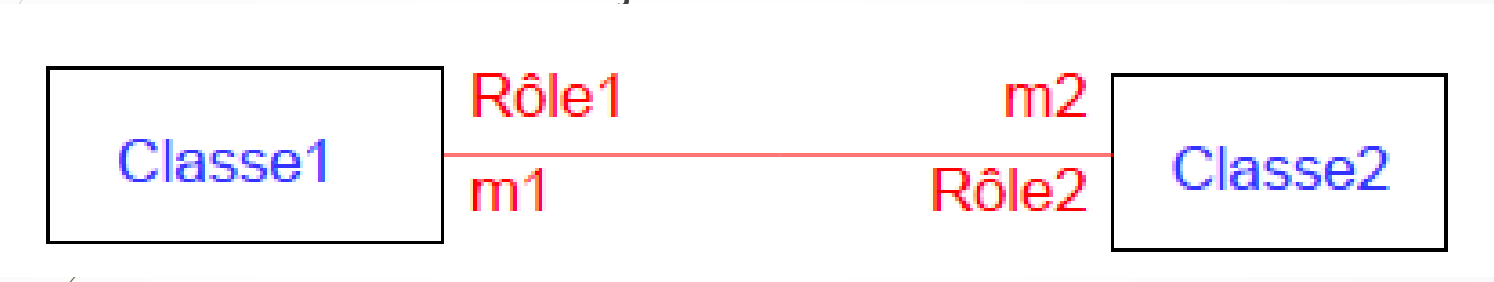
- Le rôle décrit comment une classe voit une autre classe à travers une association
- Une association a par essence 2 rôles, un de chaque extrémité de l'association





Multiplicité ou cardinalité

- La multiplicité : précise le nombre d'objets d'une classe pouvant être liés à un objet de l'autre classe

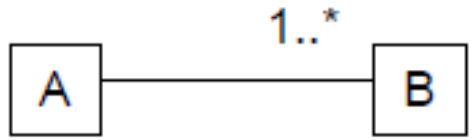


- Un objet de classe1 est lié à m2 objets de classe2.
- Un objet de classe2 est lié à m1 objets de classe1.

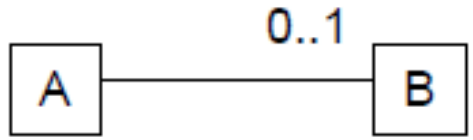
Valeur :	signification :
1	Un et un seul
0..1	Zéro ou un
M .. N	De M à N (entiers naturels)
* (ou 0 .. *)	De zéro à plusieurs
N	Exactement N
1 .. *	D'un à plusieurs

Multiplicité

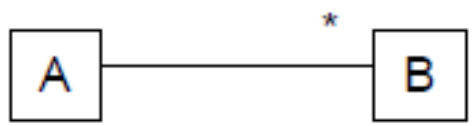
Une instance de la classe A est toujours associée avec une instance de la classe B



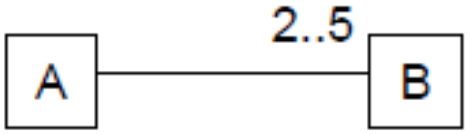
Une instance de la classe A est toujours associée avec une ou plusieurs instances de la classe B



Une instance de la classe A est associée avec zéro ou une instance de la classe B



Une instance de la classe A est associée avec zéro, une ou plusieurs instances de la classe B

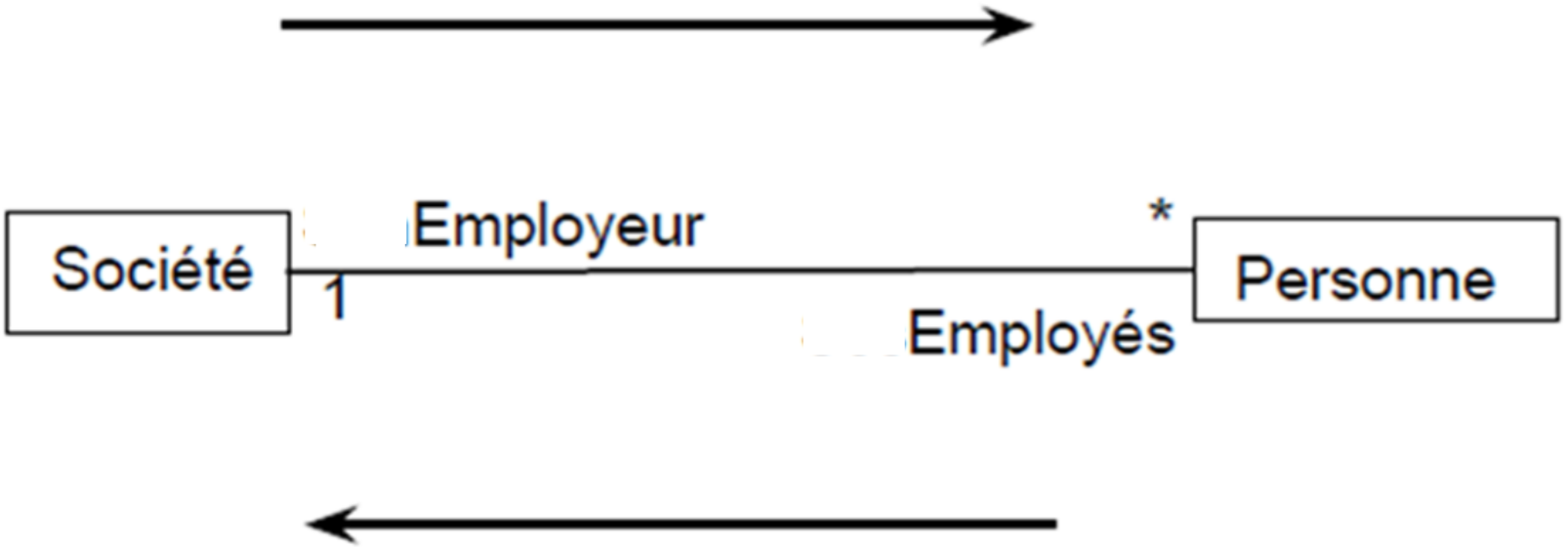


Une instance de la classe A est associée avec entre deux et cinq instances de la classe B

Lecture d'une association

Societe

« Un employeur emploie plusieurs personnes »

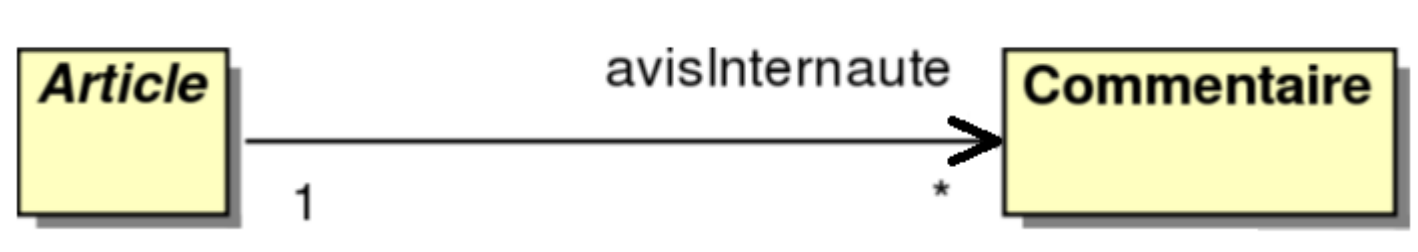


personne

« Un employé est employé par un seul employeur »

Navigabilité d'une association

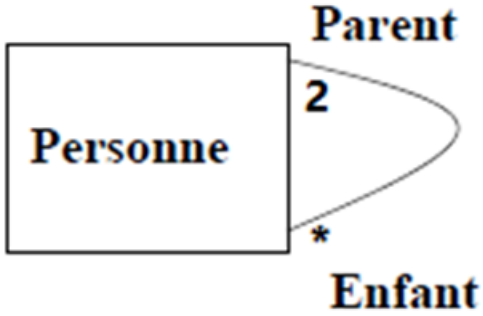
- La navigabilité permet de spécifier dans quel(s) sens il est possible de traverser l'association à l'exécution.
- On restreint la navigabilité d'une association à un seul sens à l'aide d'une flèche.
- Exemple : Connaissant un article on peut connaître les commentaires, mais pas l'inverse.



- On peut aussi représenter les associations navigables dans un seul sens par des attributs.
- **Exemple** : En ajoutant à la classe Article un attribut « avisInternaute » qui est une collection de classe « Commentaire » à la place de l'association (avisInternaute : Commentaire [0..*].

Associations reflexives

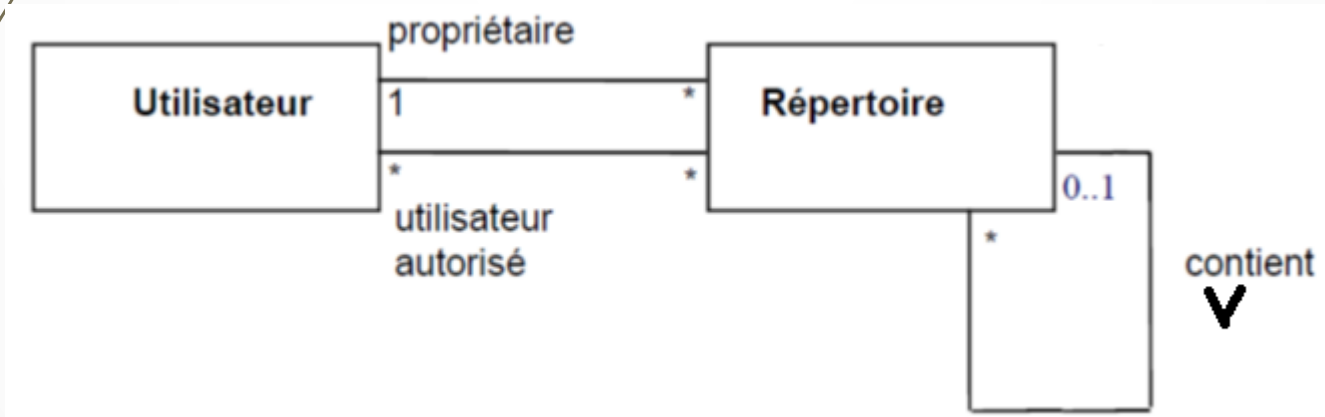
- L'association la plus utilisée est l'association binaire (reliant deux classes).
- Parfois, les deux extrémités de l'association pointent vers la même classe. Dans ce cas, l'association est dite « réflexive ».



est parent de (ou est un enfant de)

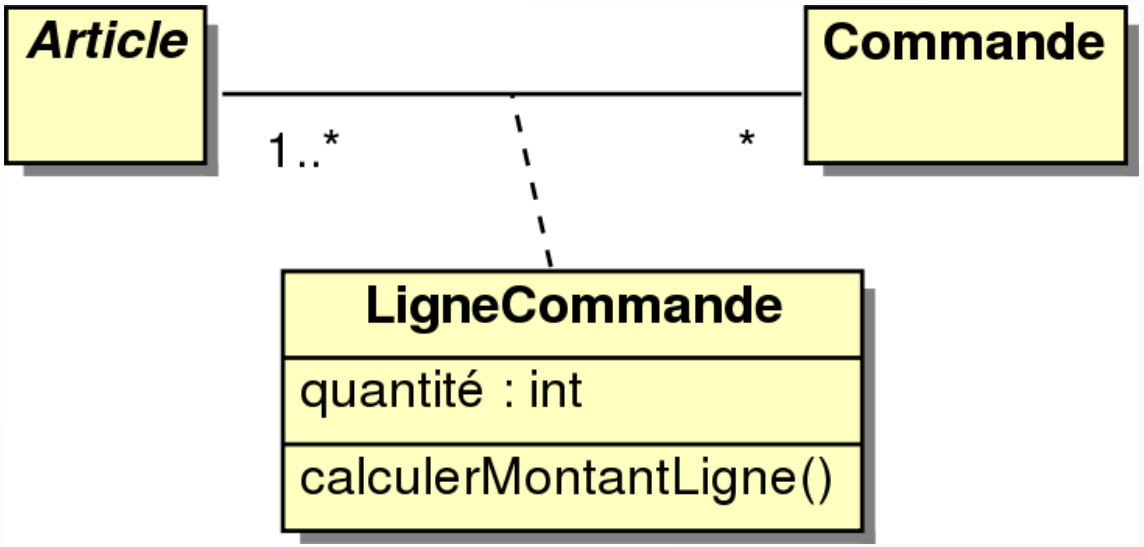
Rôles dans les associations

- Un répertoire peut contenir plusieurs répertoires (contient) et peut être éventuellement contenu dans un autre répertoire (est contenu dans)
- Chaque répertoire a un seul utilisateur qui est son propriétaire (propriétaire) et plusieurs utilisateurs sont autorisés à accéder au répertoire (utilisateur autorisé)



Classe-association

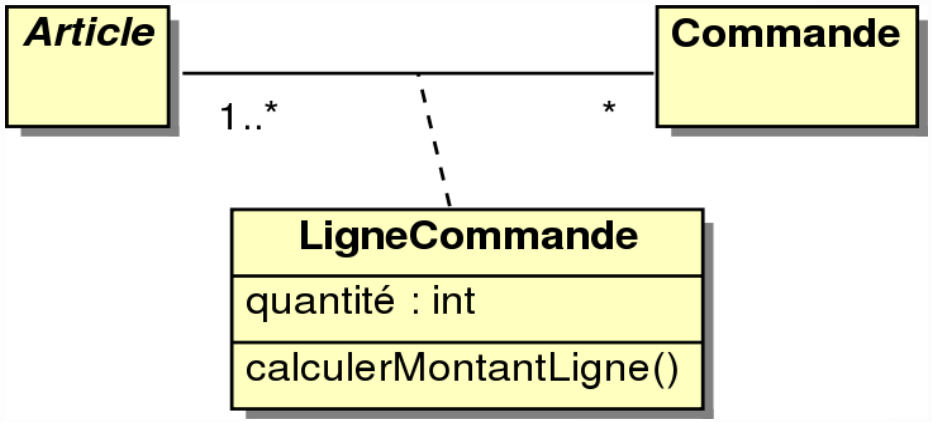
- Une association peut avoir ses propres attributs, qui ne sont disponibles dans aucune des classes qu'elle lie.
- Comme, dans le modèle objet, seules les classes peuvent avoir des attributs, cette association devient alors une classe appelée « classe-association » ou « classe-associative ».



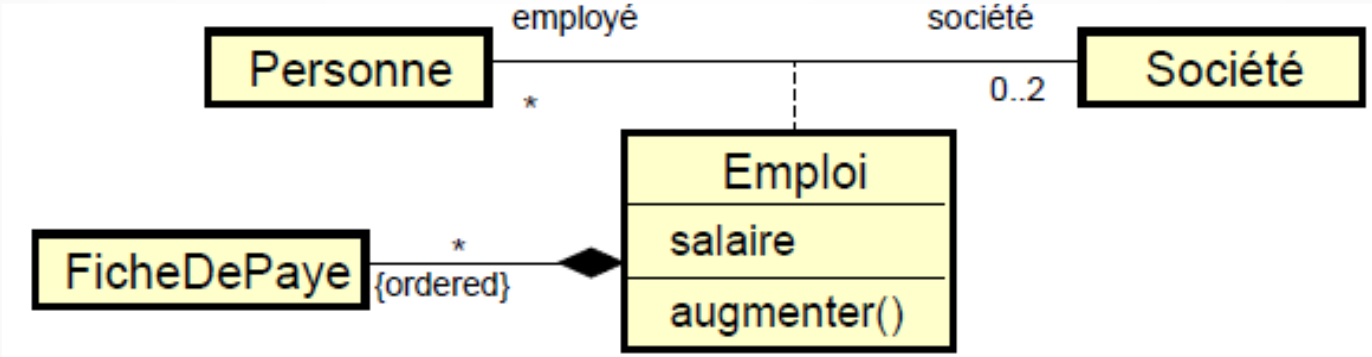
- Chaque instance de l'association (un lien entre un objet Article et un objet commande), a ses valeurs pour les attributs de la classe associative

Classe-association

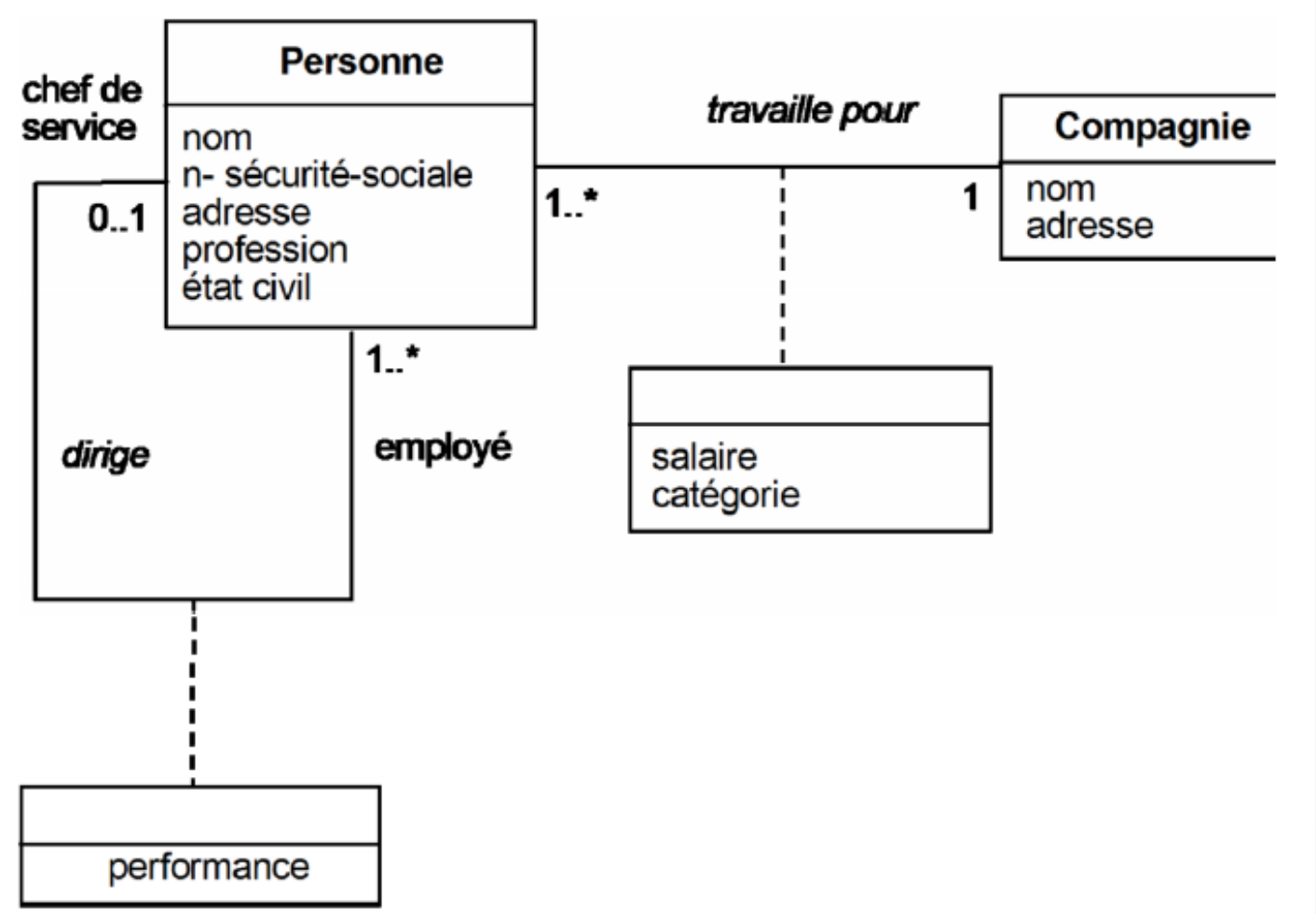
- Les classes associatives sont des associations mais aussi des classes.



- Elles ont donc les mêmes propriétés et peuvent par exemple être liées par des associations.

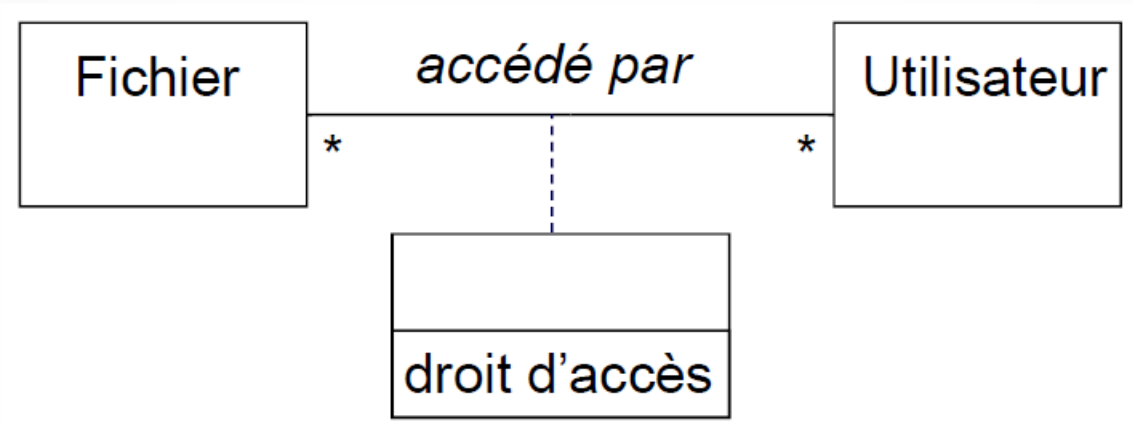


Classe-association



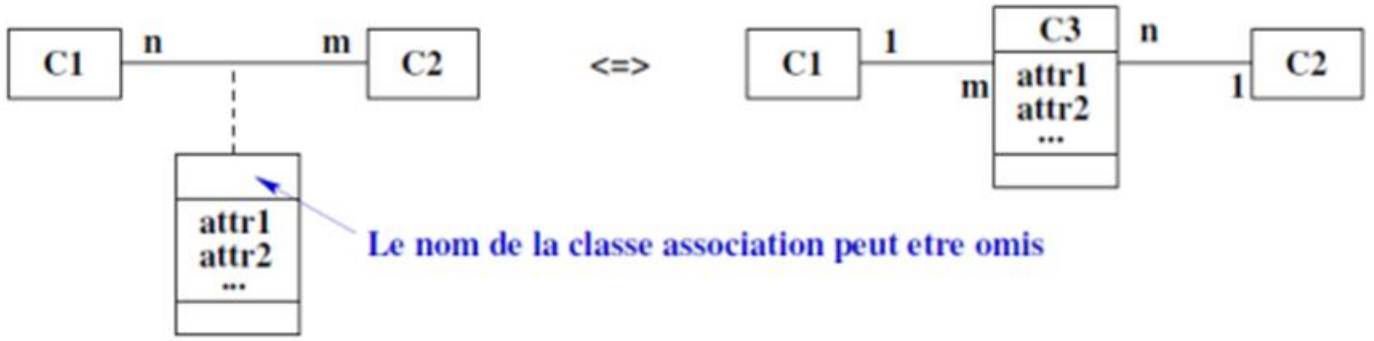
Classe-association

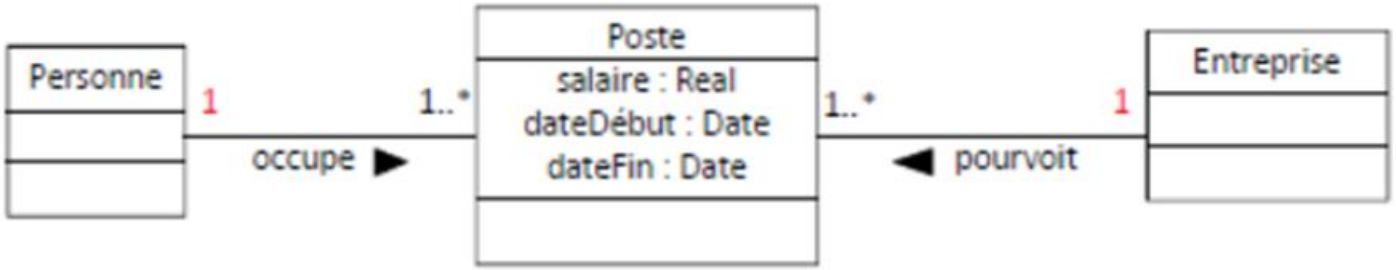
- Un attribut de lien est une propriété spécifique d'un lien dans une association
- Un attribut de lien contient des informations qui ne peuvent pas être réparties entre les objets en relation
  - Si on mets droit d'accès dans la classe fichier : on définit les droits d'accès sur un fichier pour tous les utilisateurs
  - Si on mets droit d'accès dans la classe Utilisateur : on définit le droit d'accès d'un utilisateur sur tous les fichiers



Classe-association

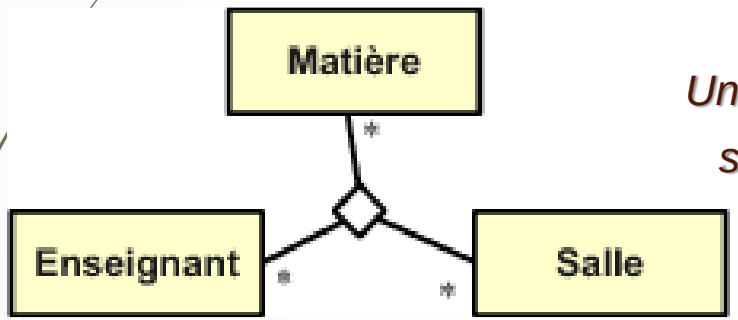
Une classe-association peut être remplacée par une classe intermédiaire qui sert de pivot (on sépare la relation initiale : attention au changement de multiplicité par rapport à la version avec classe-association)





Associations n-aires

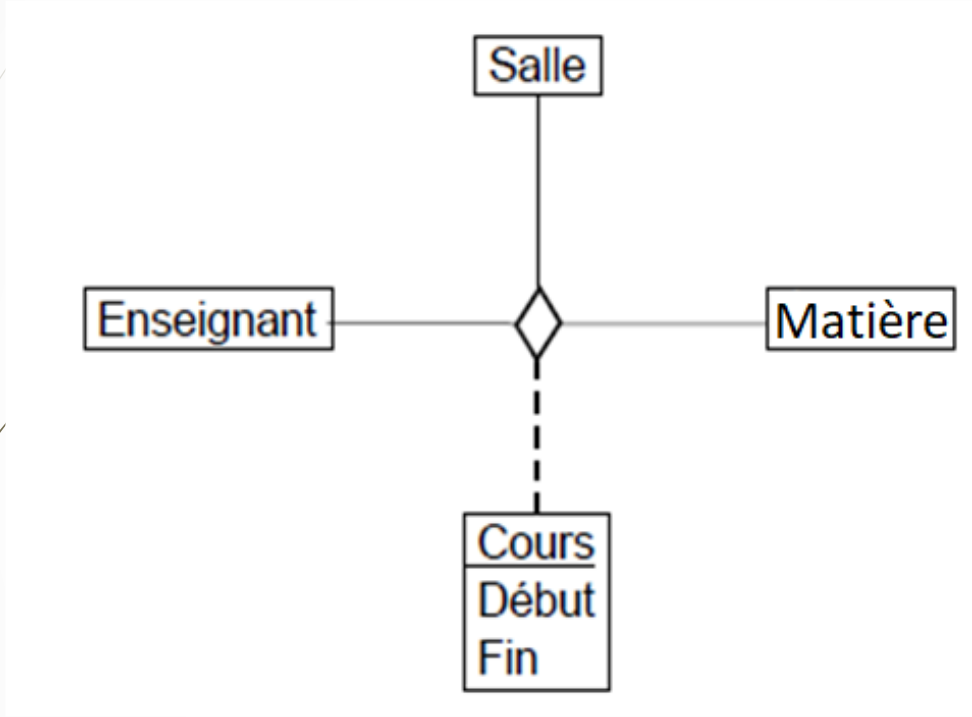
- Une association n-aire lie plus de deux classes.
- Notation avec un losange central pouvant éventuellement accueillir une classe-association.



Un enseignant enseigne une matière dans une salle (un lien entre un enseignant, une matière et une salle)

- Les associations n-aires sont peu fréquentes et concernent surtout les cas où les multplicités sont toutes « * ». Dans la plupart des cas, on utilisera plus avantageusement des classes-association ou plusieurs relations binaires.

Associations ternaires



Associations ternaires

Souvent lorsque relation de plusieurs à plusieurs : réduction des associations n-aires

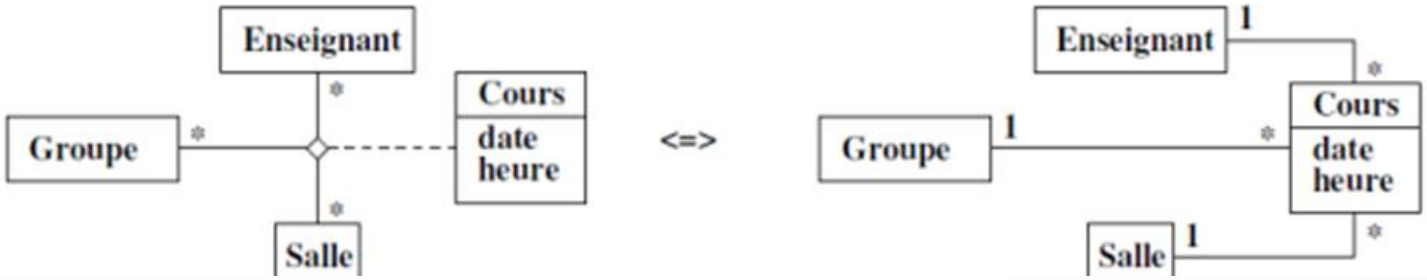
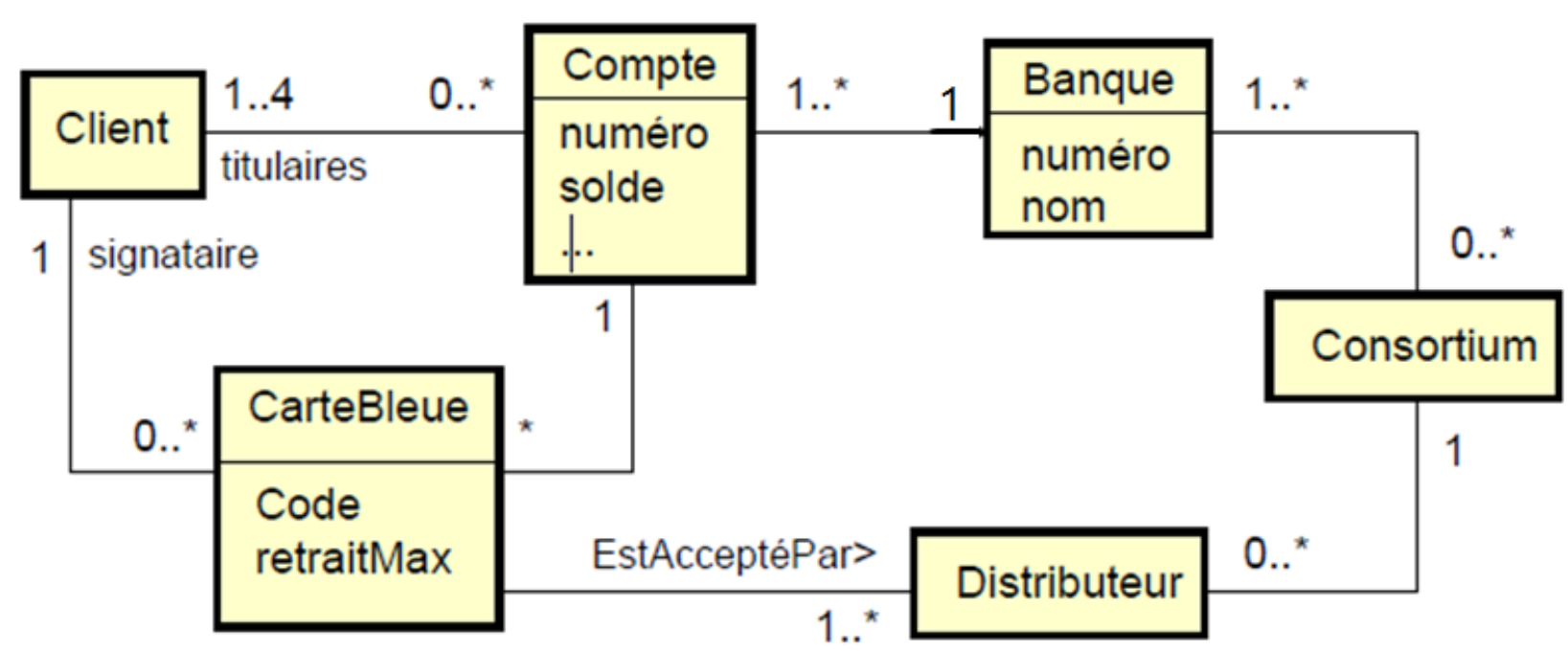


Diagramme de classe

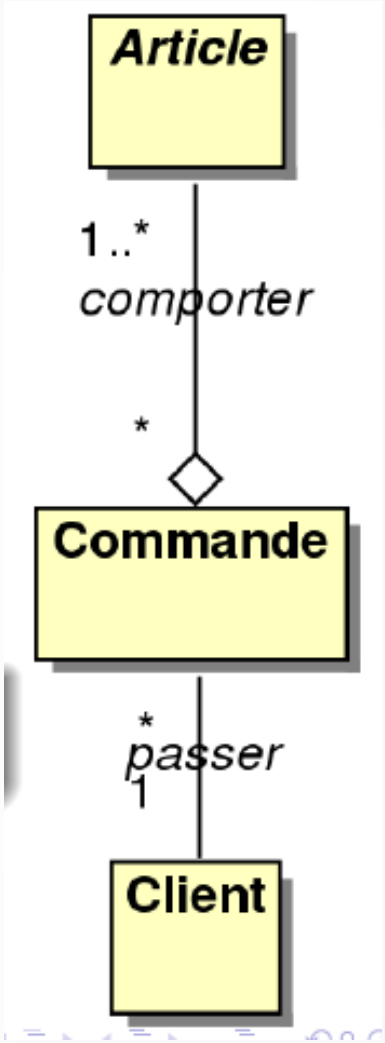


Les relations entre classes

- L'association
- L'agrégation
- La composition
- La généralisation

Association de type agrégation

- Une **agrégation** est un cas particulier d'association non-symétrique exprimant une relation de contenance.
- Une agrégation dénote une relation d'un ensemble (tout) à ses parties.
  - L'ensemble est l'agrégat (commande) et la partie est l'agrégé (article).
- Elle n'a pas besoin d'être nommée, implicitement elle signifie « contient » ou « comporte ».
- On représente l'agrégation par l'ajout d'un losange vide du côté de l'agrégat.

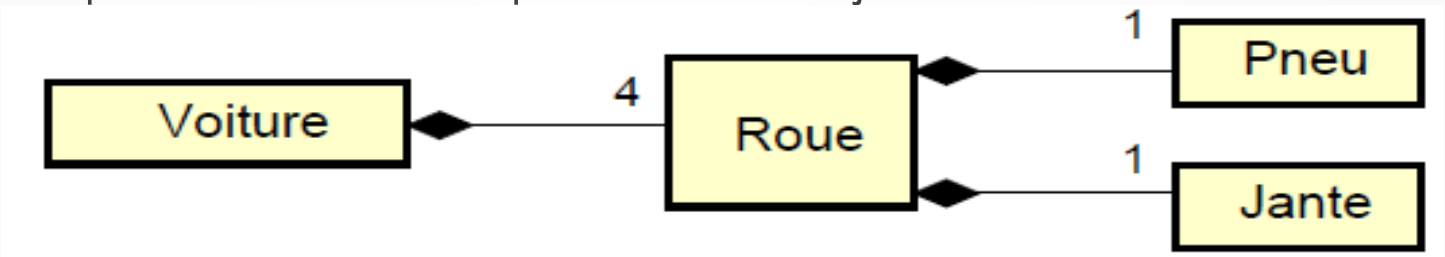


Les relations entre classes

- L'association
- L'agrégation
- La composition
- La généralisation

Association de type composition

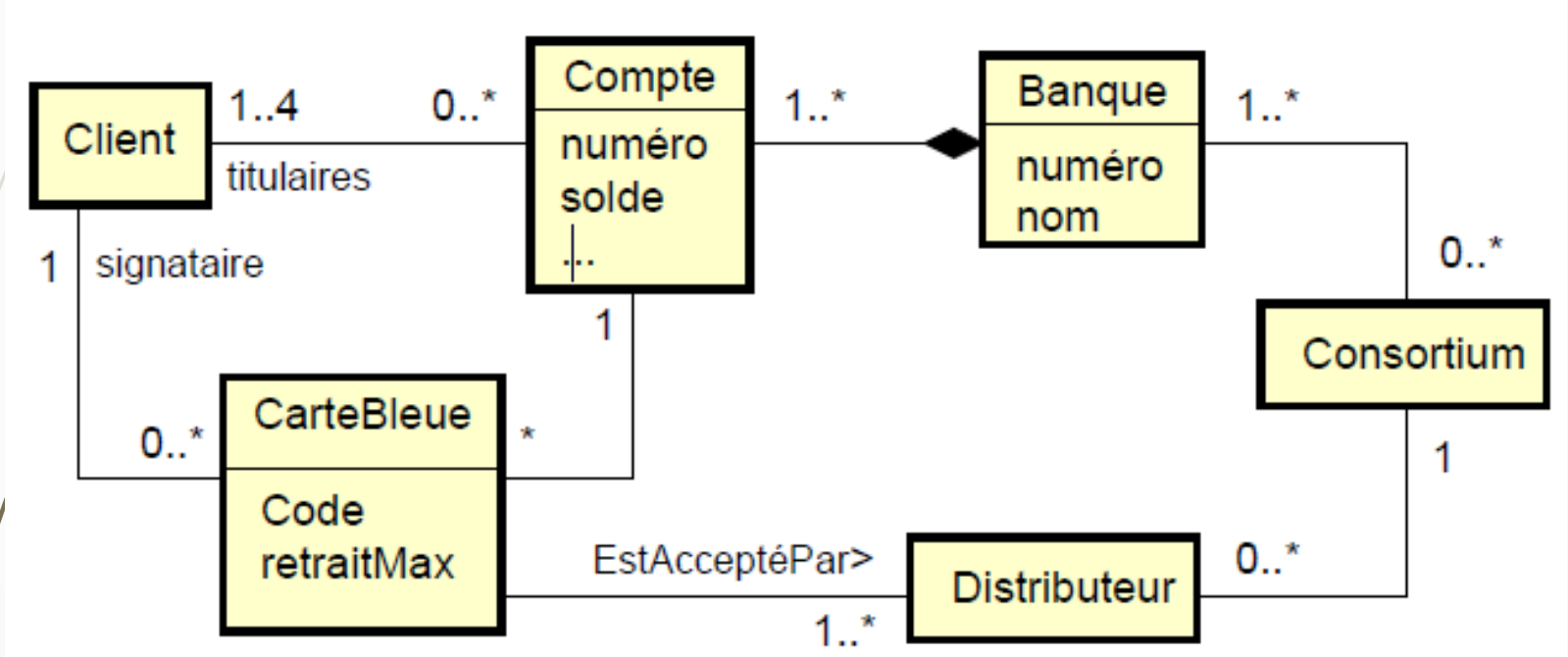
- La **composition** est aussi dite « agrégation forte ».
- La relation de composition décrit une contenance structurelle entre instances.
- Une agrégation dénote une relation d'un ensemble (composite) à ses parties (composants).
- La destruction de l'objet composite impliquent la destruction de ses composants.
- Une instance de la partie n'appartient jamais à plus d'une instance de l'élément composite.
- On utilise un losange plein, du coté composite, pour représenter une composition.
- La multiplicité coté composite est toujours 1.



Composition et agrégation

- Dès lors que l'on a une de relation contenance, on a une relation d'agrégation ou de composition.
- Pour décider de mettre une composition plutôt qu'une agrégation, on doit se poser la question suivante :
  - Est-ce que la destruction de l'objet composite implique nécessairement la destruction des objets composants ?
- Si on répond par l'affirmative à cette question, on doit utiliser une composition.

Diagramme de classes

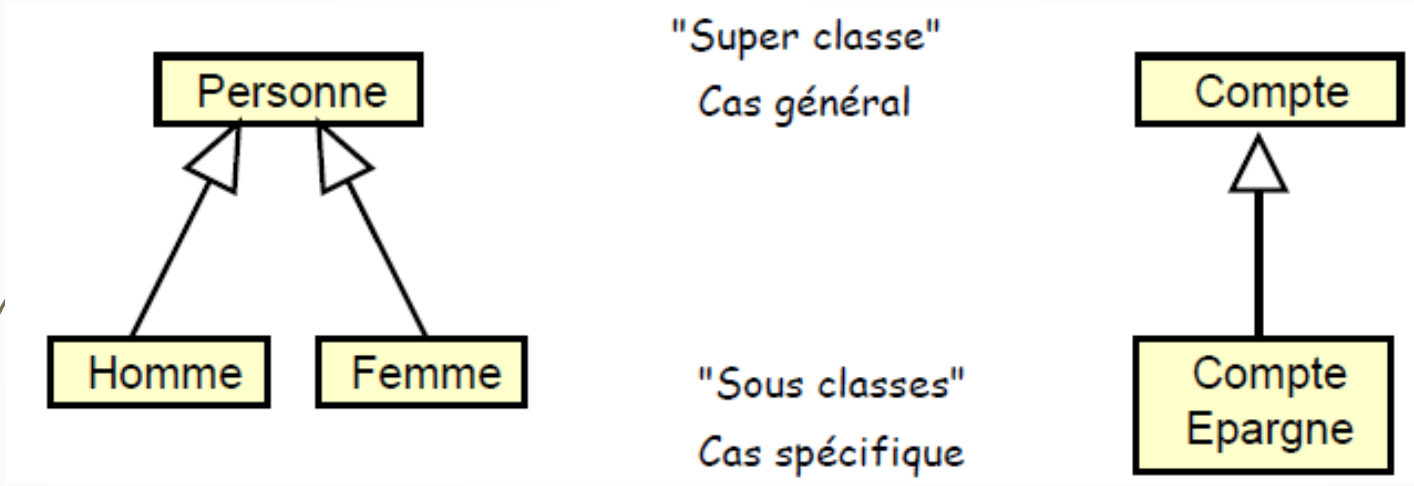


Les relations entre classes

- L'association
- L'agrégation
- La composition
- La généralisation

Généralisation / Spécialisation

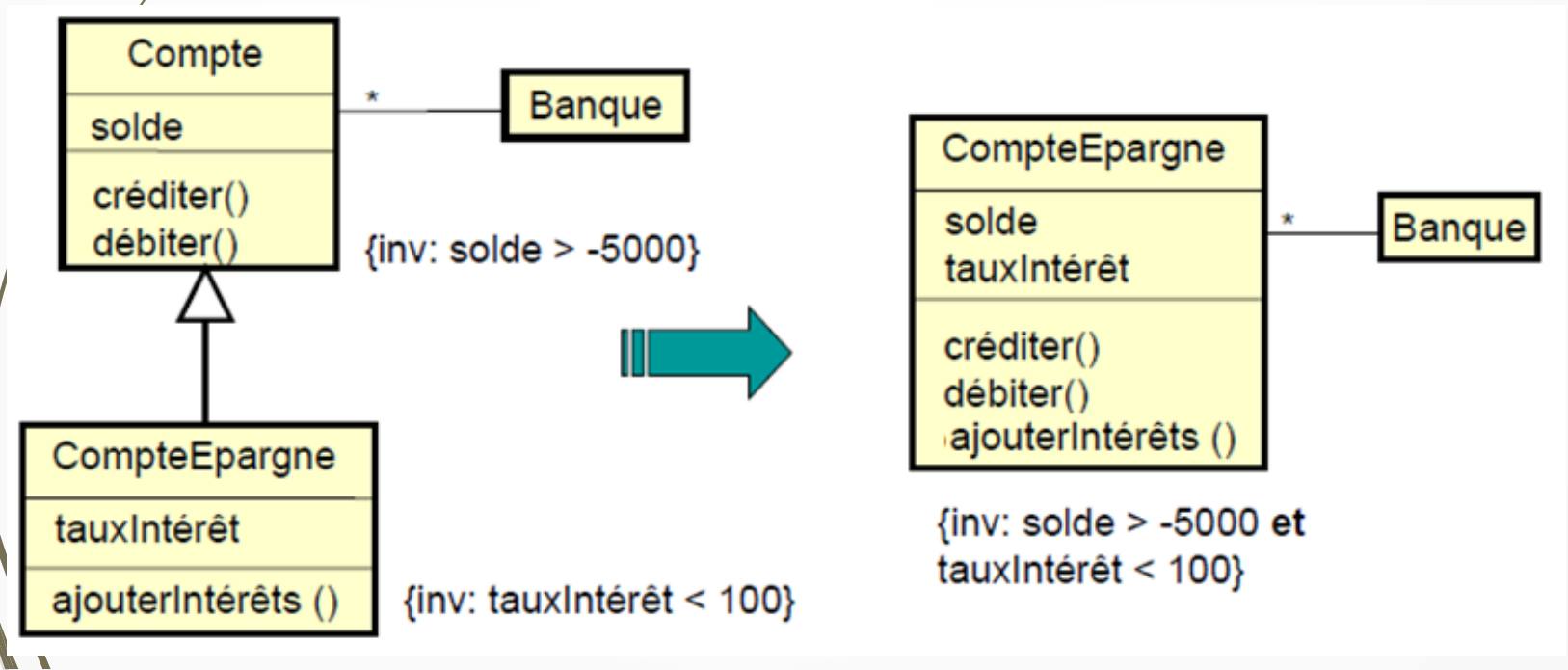
- Une classe peut être la généralisation d'une ou plusieurs autres classes.
- Ces classes sont alors des spécialisations de cette classe.



- Relation : « est une sorte de »
- En UML on parle souvent de généralisation alors que dans les langage orientée objet on parle d'héritage.

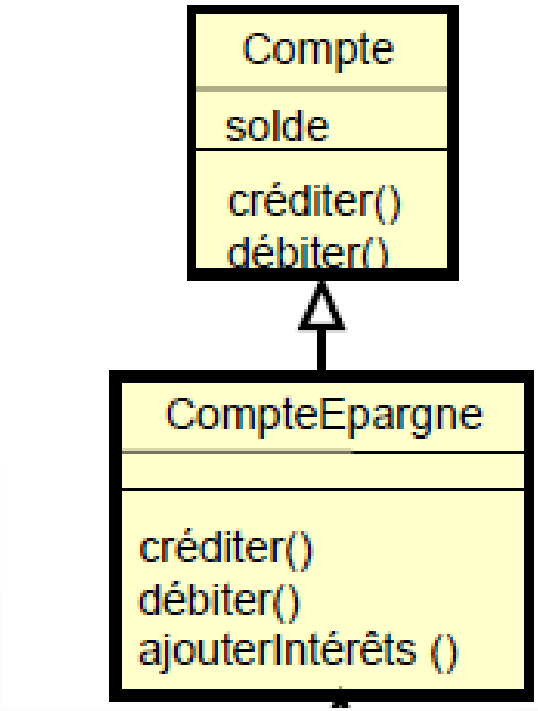
Relation d'héritage

- Les sous-classes « héritent » des propriétés des super-classes (attributs, méthodes, associations, contraintes)
- La classe enfant possède toutes les propriétés de ses classes parents (attributs et opérations)
  - peut définir de nouveaux attributs, de nouvelles opérations et de nouvelles associations qui lui sont propres



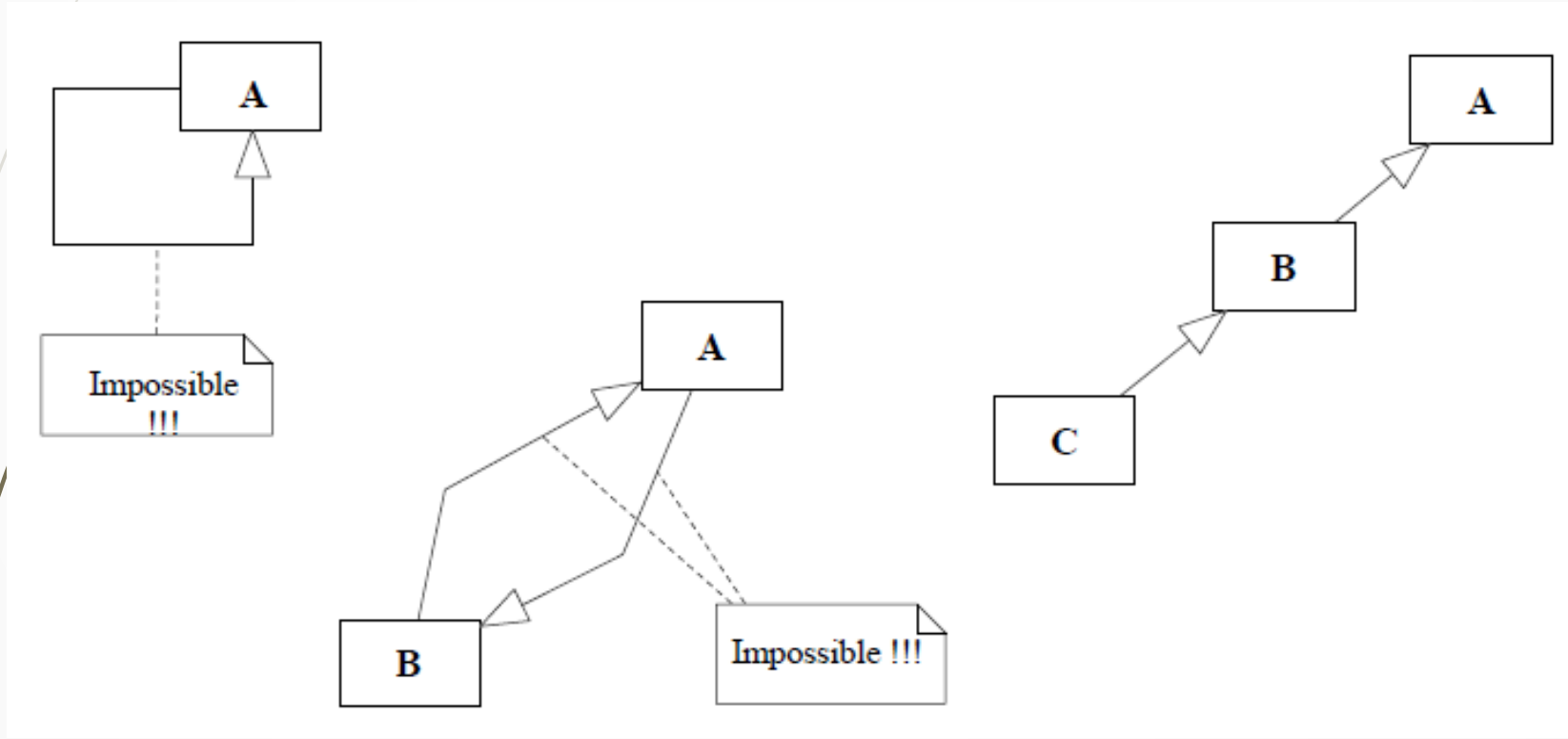
Relation d'héritage et redéfinitions

- Une opération peut être "redéfinie" dans les sous-classes
- Plusieurs implantations pour une même méthode
- Permet d'associer des méthodes spécifiques à chaque sous-classe pour réaliser une même opération



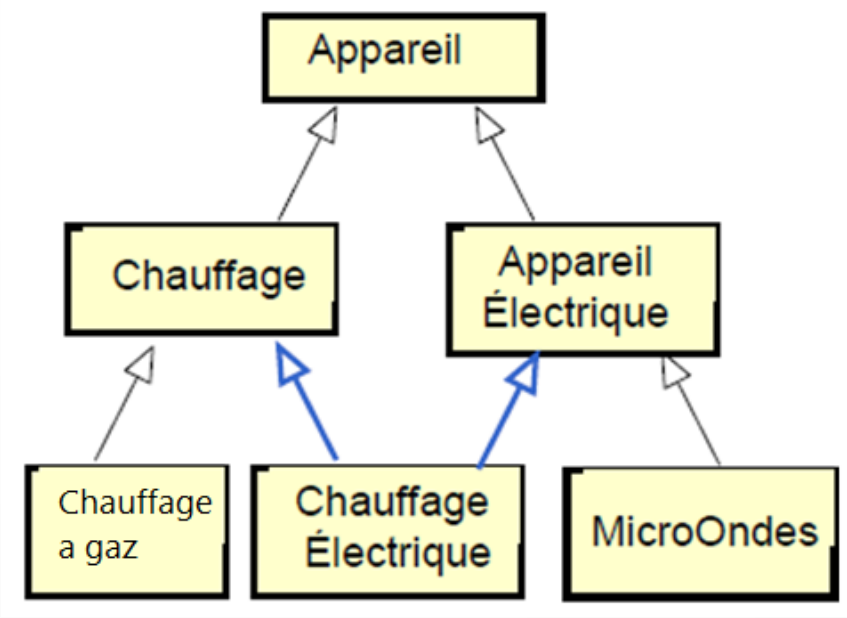
Propriétés de la généralisation

- Non-réflexive, non-symétrique, transitive



Héritage multiple

- Une classe peut avoir plusieurs classes parents. Autrement dit, une classe peut hériter de plusieurs super-classes. On parle alors d'héritage multiple.
- Le langage C++ est un des langages objet permettant son implantation effective.
- Interdit dans certains langages de programmation (Java et C#)

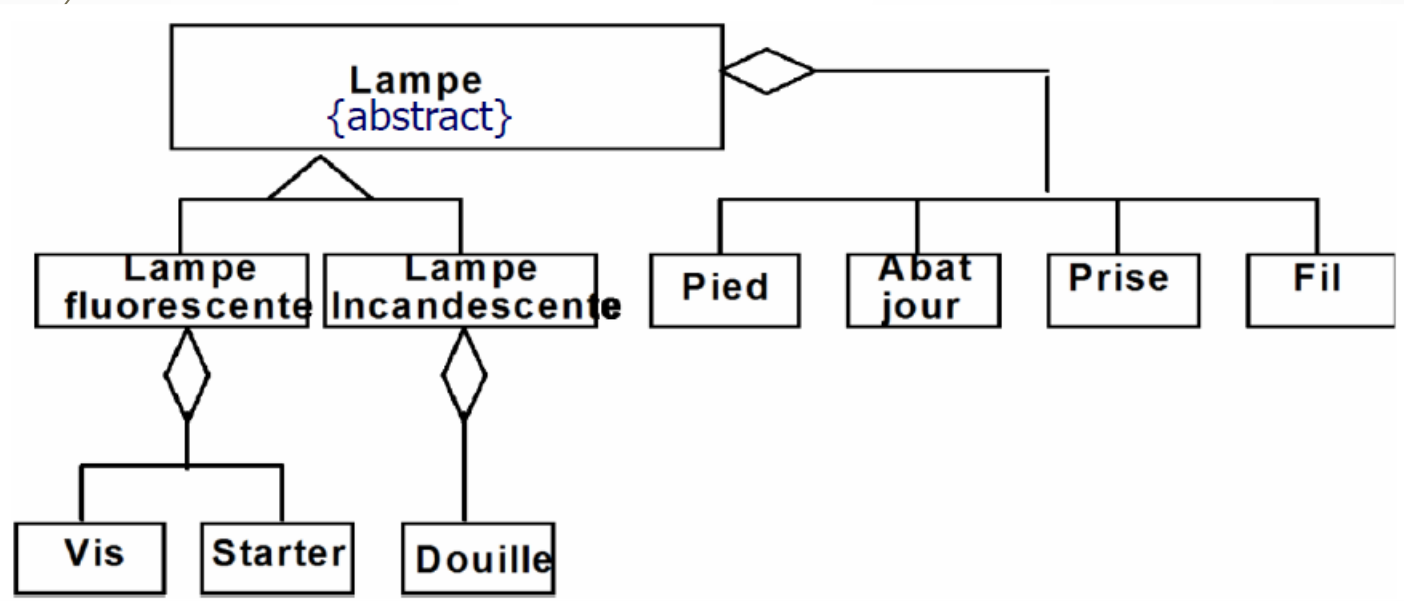


Agrégation et généralisation

- L'agrégation et la généralisation sont des concepts différents :
  - l'agrégation relie des instances
  - la généralisation relie deux classes
- L'agrégation : deux objets distincts sont impliqués si l'un d'entre eux est une partie du second
- La généralisation : la super-classe et la sous-classe définissent les propriétés d'un même objet (avec la généralisation un objet est simultanément une instance de la super-classe et de la sous-classe)
- L'agrégation est une relation "une-partie-de" (relation "et") ; la généralisation est une relation "une-sorte-de" ou "est-un " (relation "ou")

Agrégation et généralisation

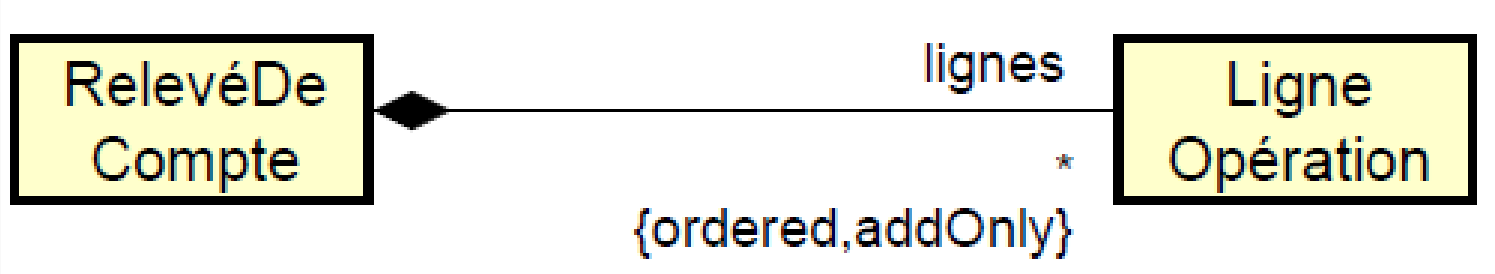
- Une lampe est composé d'un pied, d'un abat-jour, d'une prise et d'un fil
- Une lampe est une lampe incandescente ou une lampe fluorescente



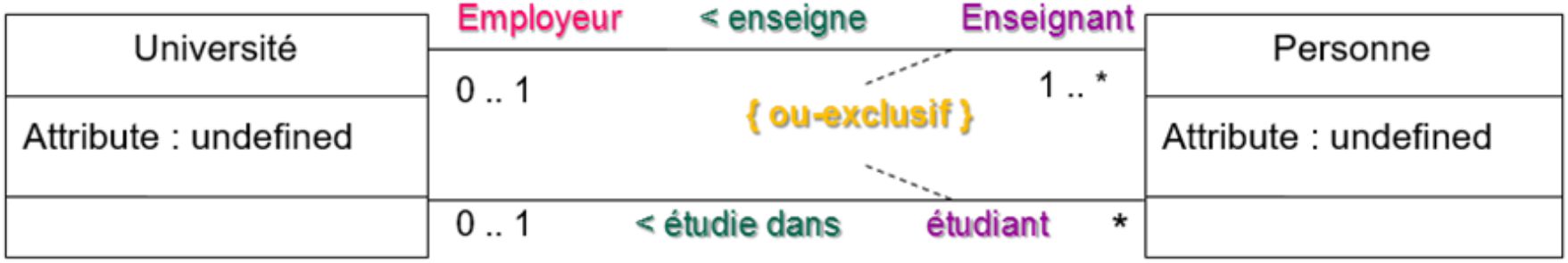
Contraintes sur les associations

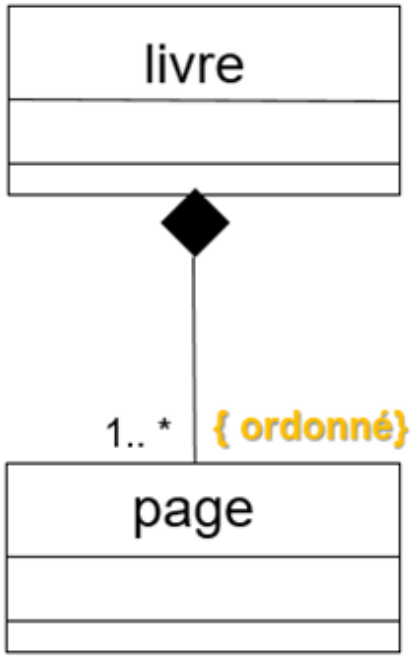
Par exemple :

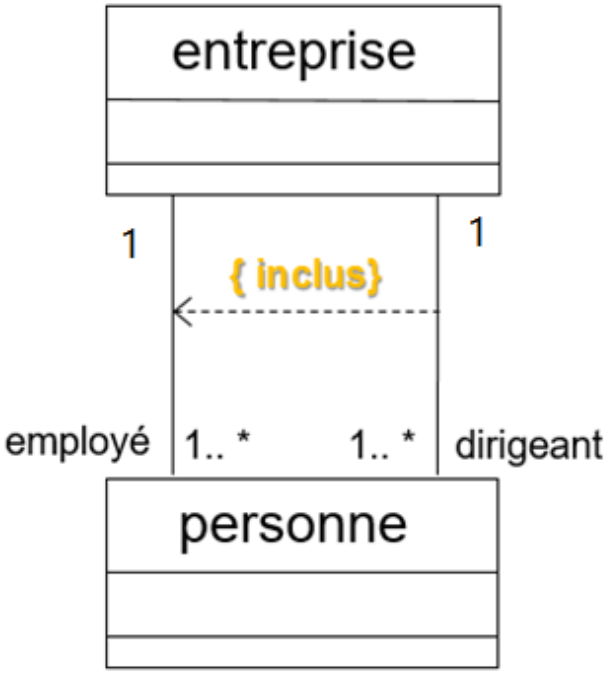
- { ordered } : les éléments de la collection sont ordonnés
- { nonUnique } : répétitions possibles (UML2.0)
- { frozen } : fixé lors de la création de l'objet, ne peut pas changer
- { addOnly } : impossible de supprimer un élément
- {x-or} : précise que, pour un objet donné, une seule association parmi un groupe d'associations est valide
- {subset} : précise que, pour un objet donné, une association est incluse dans une autre association



Contraintes sur les associations

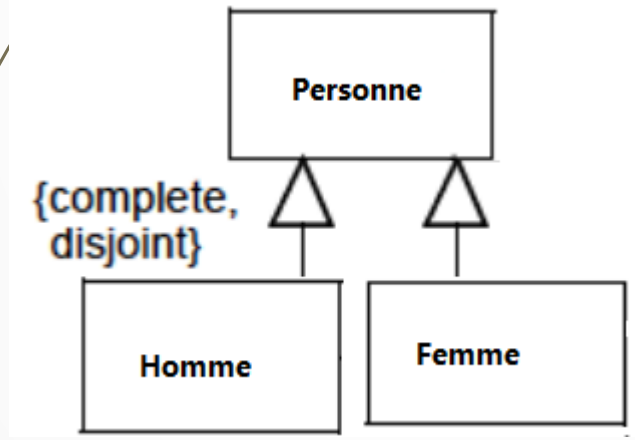


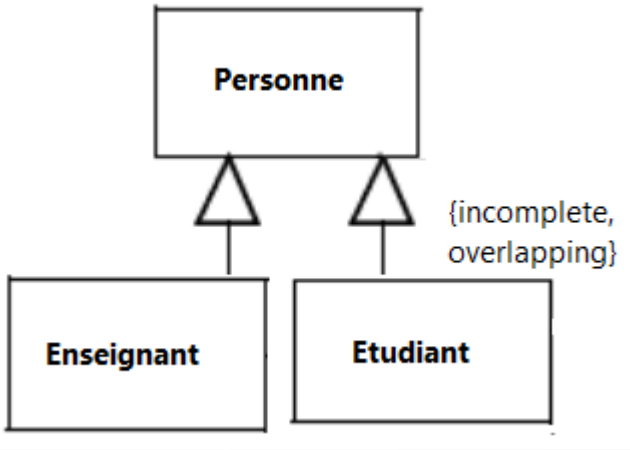




Contraintes sur les relations de généralisation

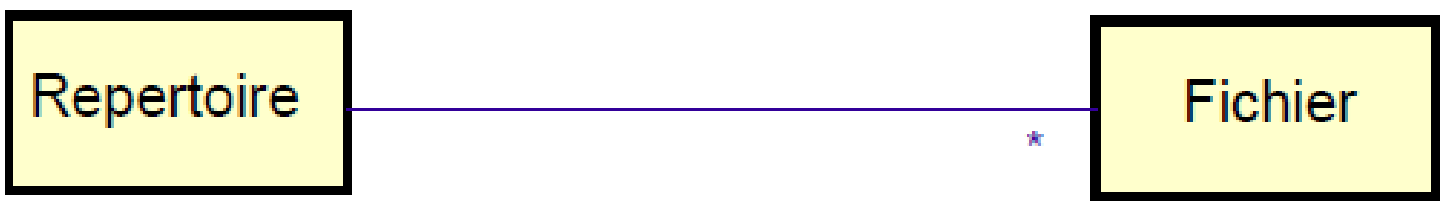
- {disjoint} ou {overlapping}
- {complete} ou {incomplete}
  - {disjoint, complete} ⇒ Partition
- {leaf} ou {root}



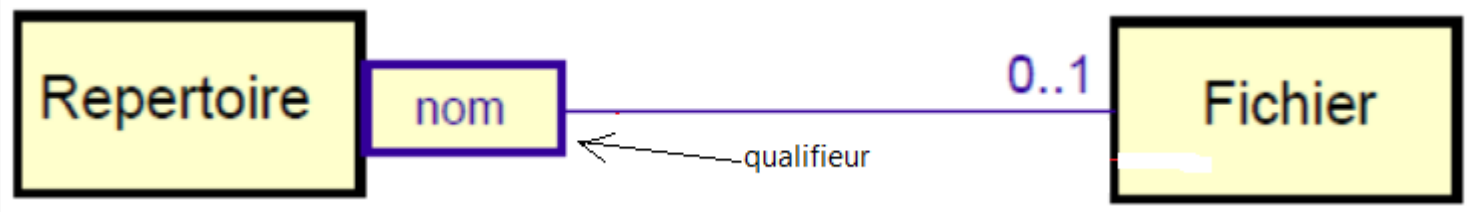


Associations qualifiées

- Un qualifieur est un attribut (ou un ensemble d'attributs) dont la valeur sert à déterminer l'ensemble des instances associées à une instance via une association.
- Un qualifieur sélectionne un sous-ensemble d'objets
- Il permet de réduire la multiplicité sur un extrémité de l'association
- Exemple : En absence de qualifieur :

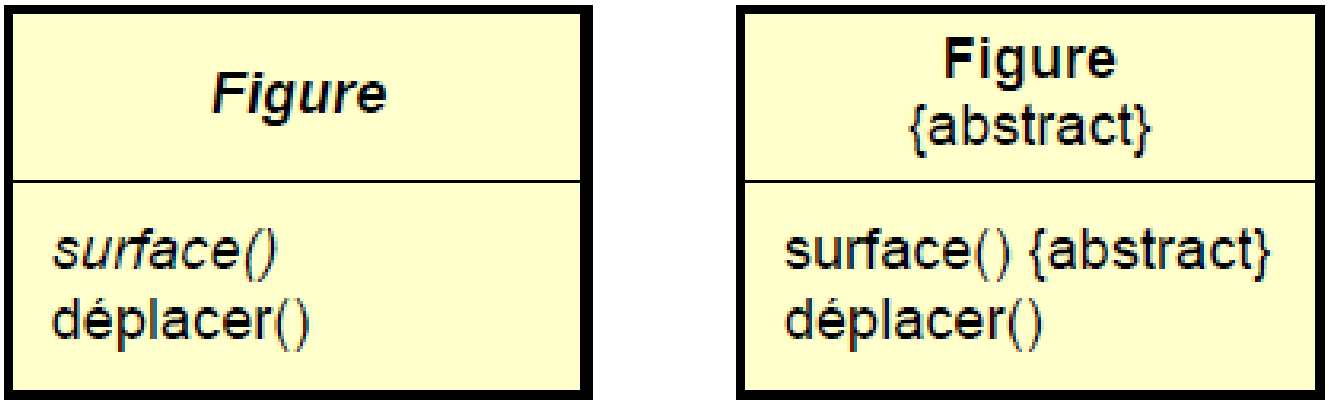


- Pour un répertoire, à un nom donné on associe qu'un seul fichier (ou 0 s'il existe aucun fichier de ce nom dans ce répertoire)."

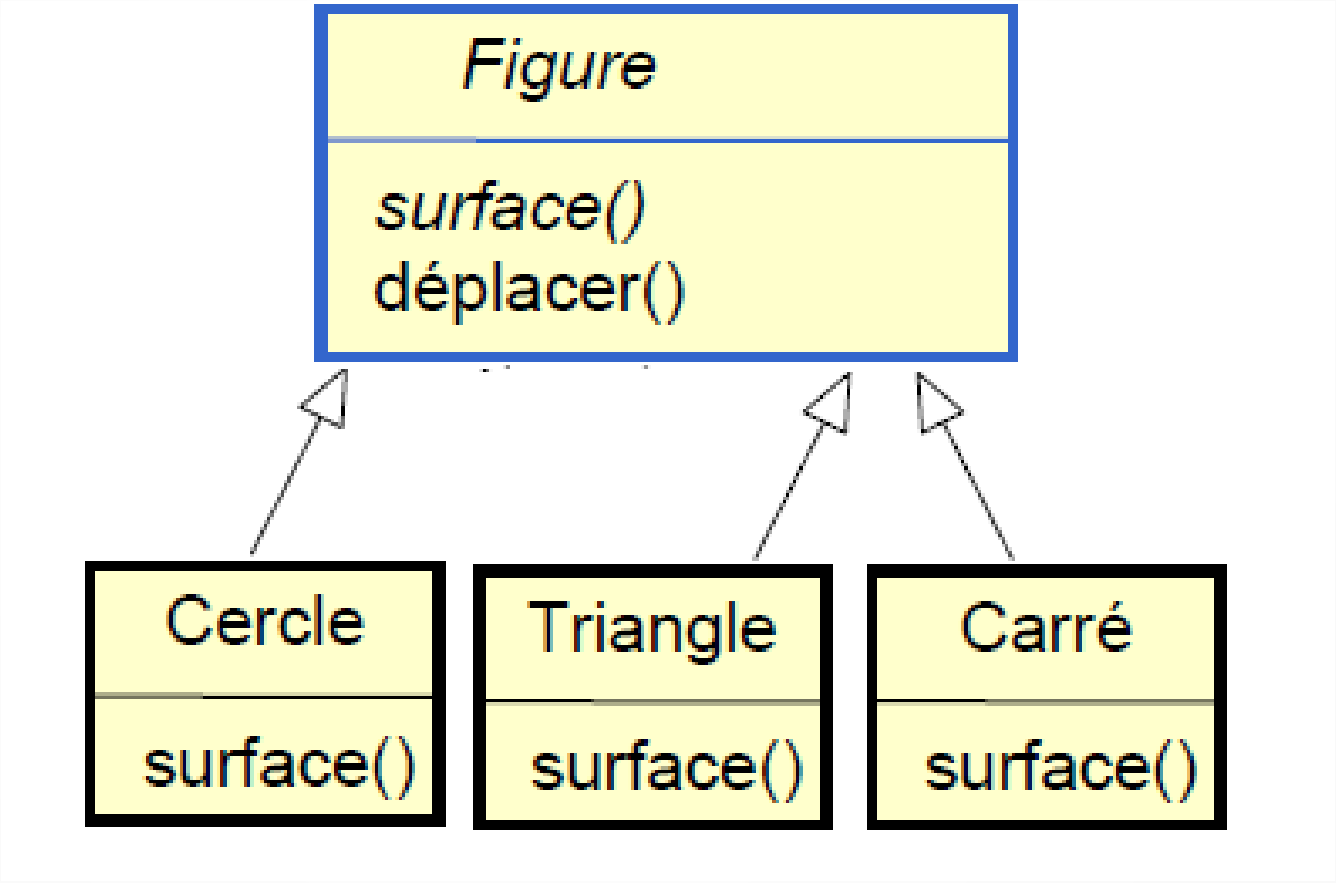


Classes et méthodes abstraites

- Une classe abstraite
  - ne peut pas être instanciée
  - peut contenir des méthodes abstraites
- Une méthode abstraite
  - Méthode non implémentée
  - est dans une classe abstraite
  - doit être définie dans une sous classe



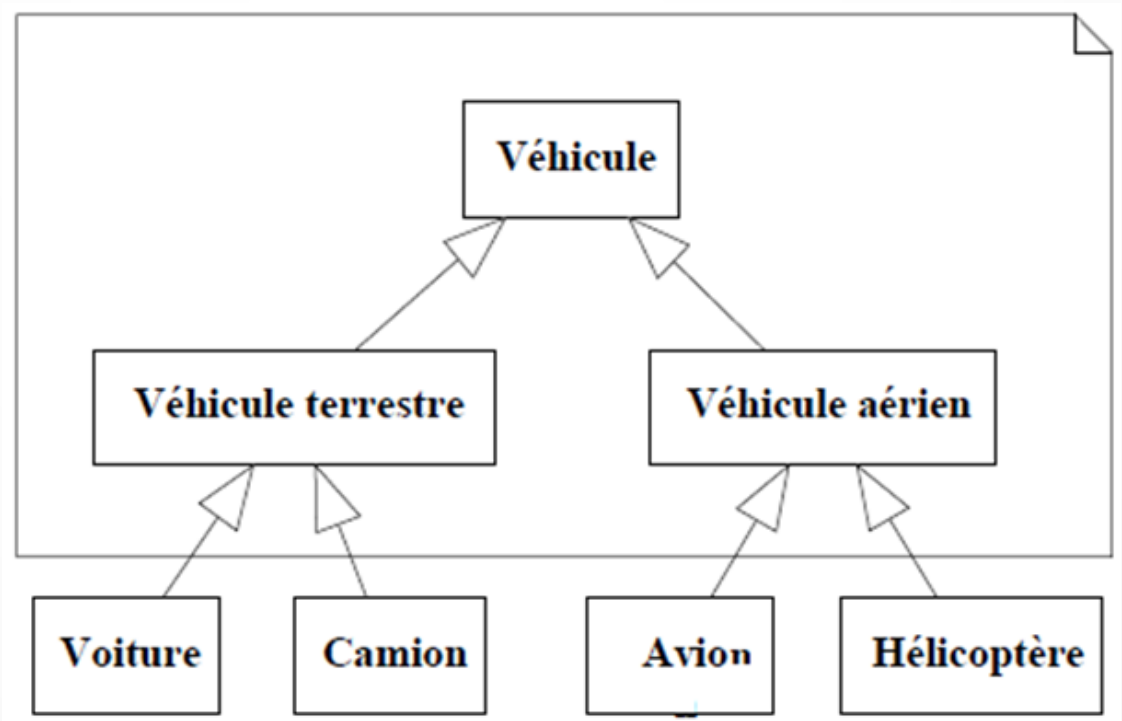
Classes et méthodes abstraites



Généralisation

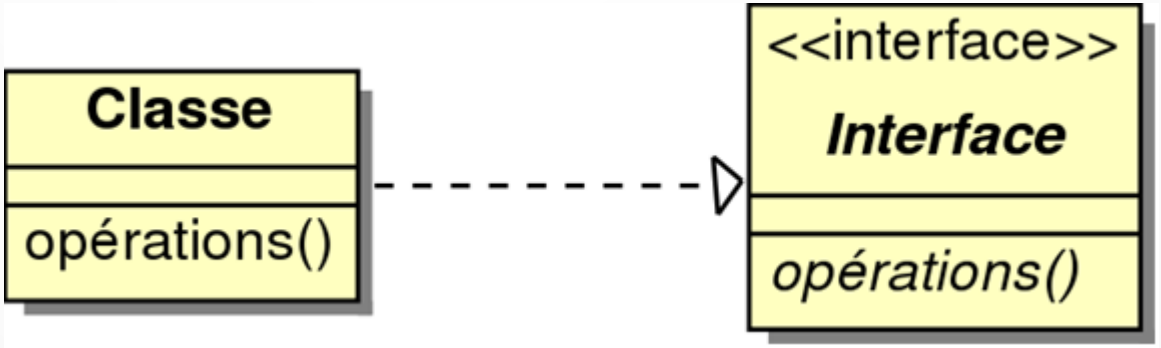
- Factoriser les éléments communs, et les mettre dans une super-classe
  - attributs, opérations et contraintes

- Généralement, cette super-classe est abstraite



Interface

- Une interface regroupe un ensemble d'opérations abstraites, sans implémentation, assurant un service cohérent.
- Les interfaces sont implémentées par des classes. Les classes implémentant une interface doivent implémenter toutes les opérations décrites dans l'interface
  - L'interface est une sorte de contrat
- Une interface est définie comme une classe, avec les mêmes compartiments. On ajoute le **stéréotype interface** avant le nom de l'interface.
- On utilise une relation de type **réalisation** entre une interface et une classe qui l'implémente.



Difference Interface – classe abstraite

Pourquoi des interfaces ?

- Utilisation similaire aux classes abstraites
  - une classe abstraite peut avoir des attributs.
  - une classe abstraite peut contenir des méthodes non abstraites
- En Java : une classe ne peut hériter de plus d'une classe, mais elle peut réaliser plusieurs interfaces (simulation de l'héritage multiple)

Construction - diagramme de classes

1. Trouver les classes du domaine étudié ;
   - Souvent, concepts et substantifs du domaine.
2. Trouver les associations entre classes;
   - Souvent, verbes mettant en relation plusieurs classes
3. Trouver les attributs des classes ;
   - Souvent, substantifs correspondant à un niveau de granularité plus fin que les classes. Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs.
4. Organiser et simplifier le modèle en utilisant l'héritage et les types d'association ;
5. Itérer et raffiner le modèle.

Conclusion- diagramme de classes

Très nombreuses utilisations, à différents niveaux :

- Pendant la capture des besoins : **Modèle du domaine**
  - Classes = Objets du domaine
  - Définition des attributs uniquement, aucune opération n'est introduite
- Pendant la conception/implémentation : **Modèle de conception**
  - Classes = Objets logiciels
  - On ajoute les opérations
  - On ajoute d'autres classes qui sont necéssaires à l'implémentation

Chapitre 2 :

1. Diagrammes de classes
2. Diagrammes d'objets

Diagrammes d'objets

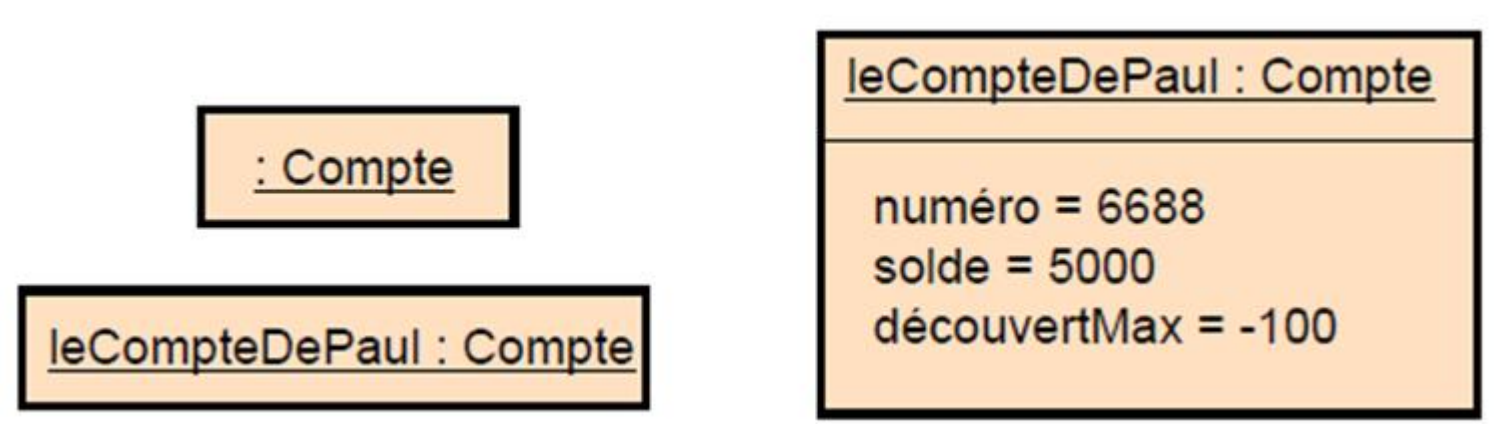
- Un diagramme de classes
  - Définit la structure interne statique du système
  - définit l'ensemble de tous les états possibles
  - les contraintes doivent toujours être vérifiées

- Un diagramme d'objets
  - décrit un état possible à un instant t, un cas particulier
  - A un instant t, il contient les objets et les liens qui les relient
  - doit être conforme au modèle de classes

- Les diagrammes d'objets peuvent être utilisés pour
  - expliquer un diagramme de classe (donner un exemple)
  - valider un diagramme de classe (le "tester")

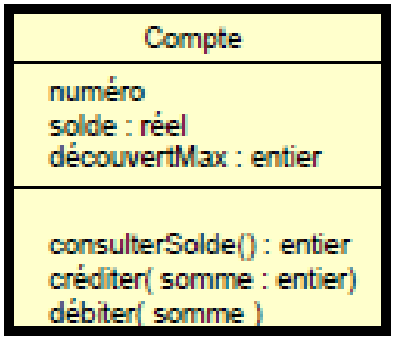
Représentation des objets

- Comme les classes, on utilise des cadres compartimentés.
- En revanche, les noms des objets sont soulignés et on peut rajouter son identifiant devant le nom de sa classe.
- Plusieurs notations possibles :



Classe vs. Objets

- La structure d'une classe est constante



- Des **objets** peuvent être ajoutés ou détruits pendant l'exécution.
- Les valeurs des attributs des objets peuvent changer d'un objet à un autre

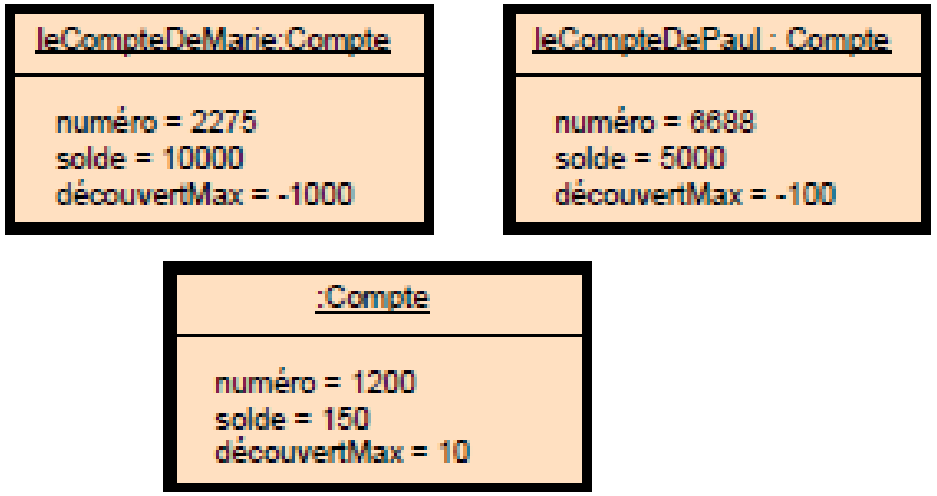


Diagramme d'objets

- Le diagramme d'objets contient des objets qui existent à un instant t avec les liens qui les relient
- Le diagramme de classes contraint la structure et les liens entre les objets.
- Diagramme cohérent avec le diagramme de classes :

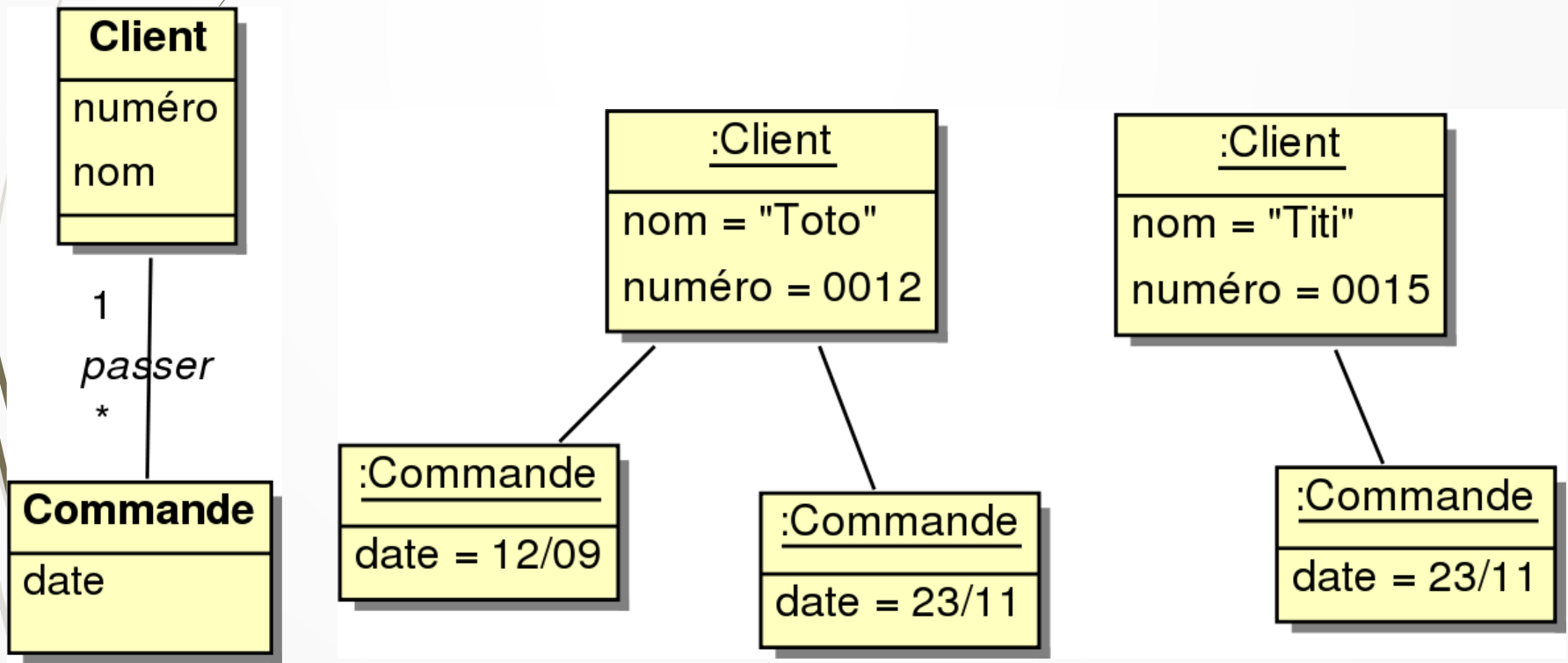
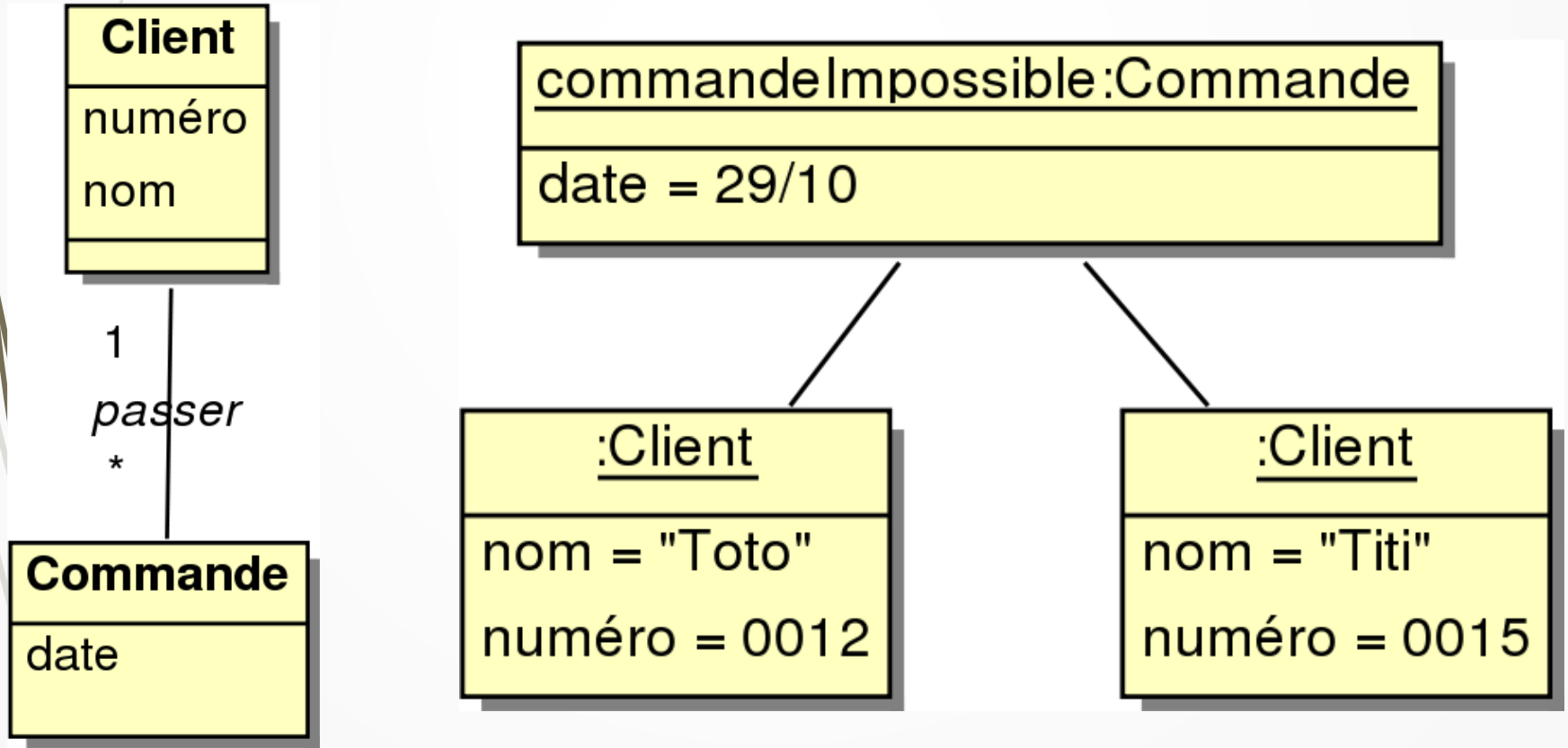


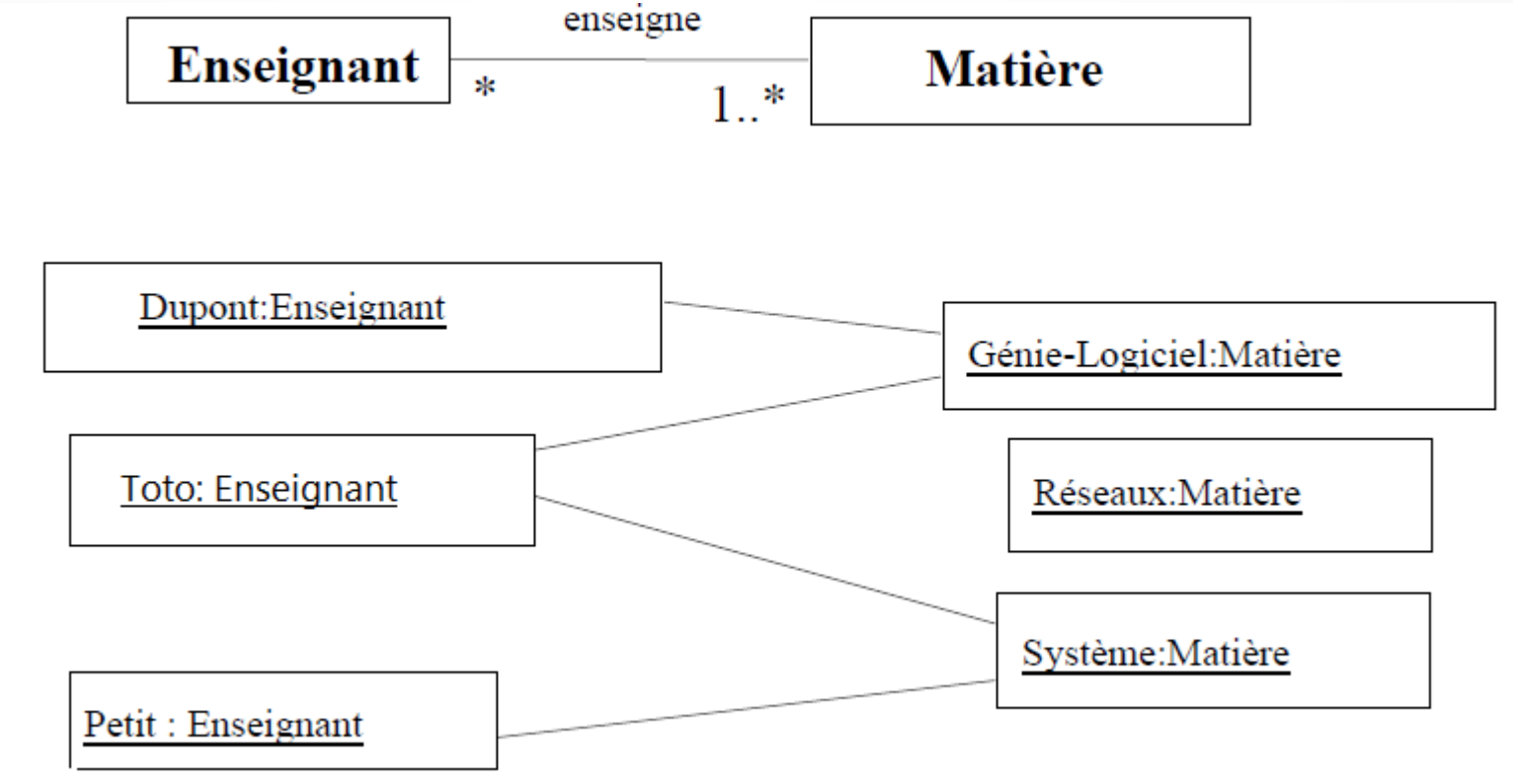
Diagramme d'objets

- Exemple de **diagramme incohérent** avec le diagramme de classes :

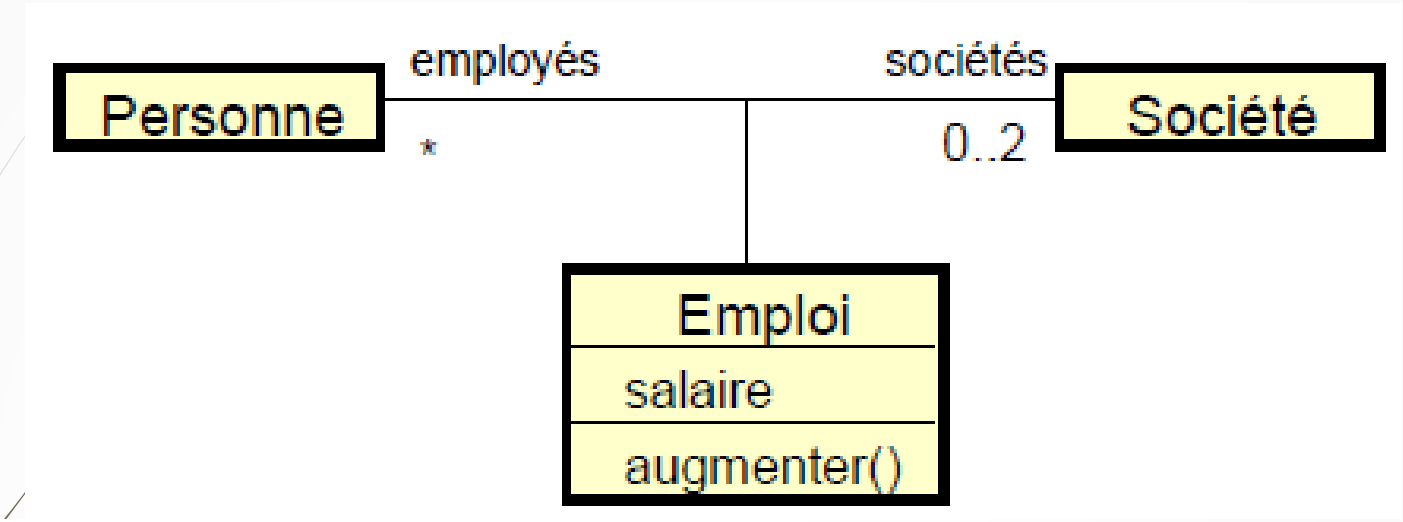


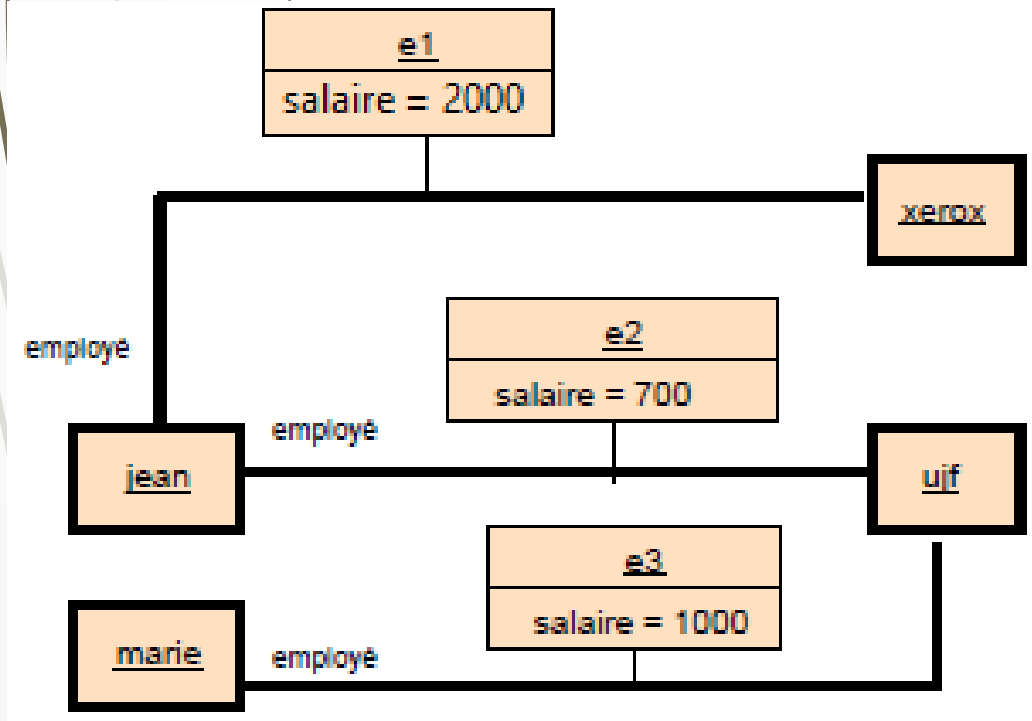
Liens et association

- Un lien est une instance d'une association.
- Un lien se représente comme une association mais s'il a un nom, il est souligné.

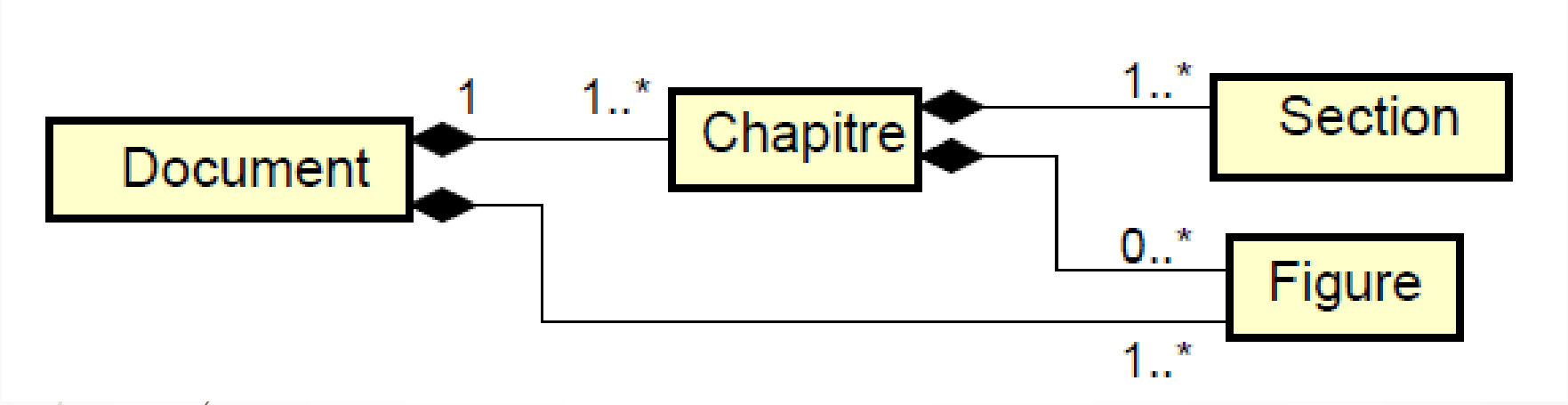


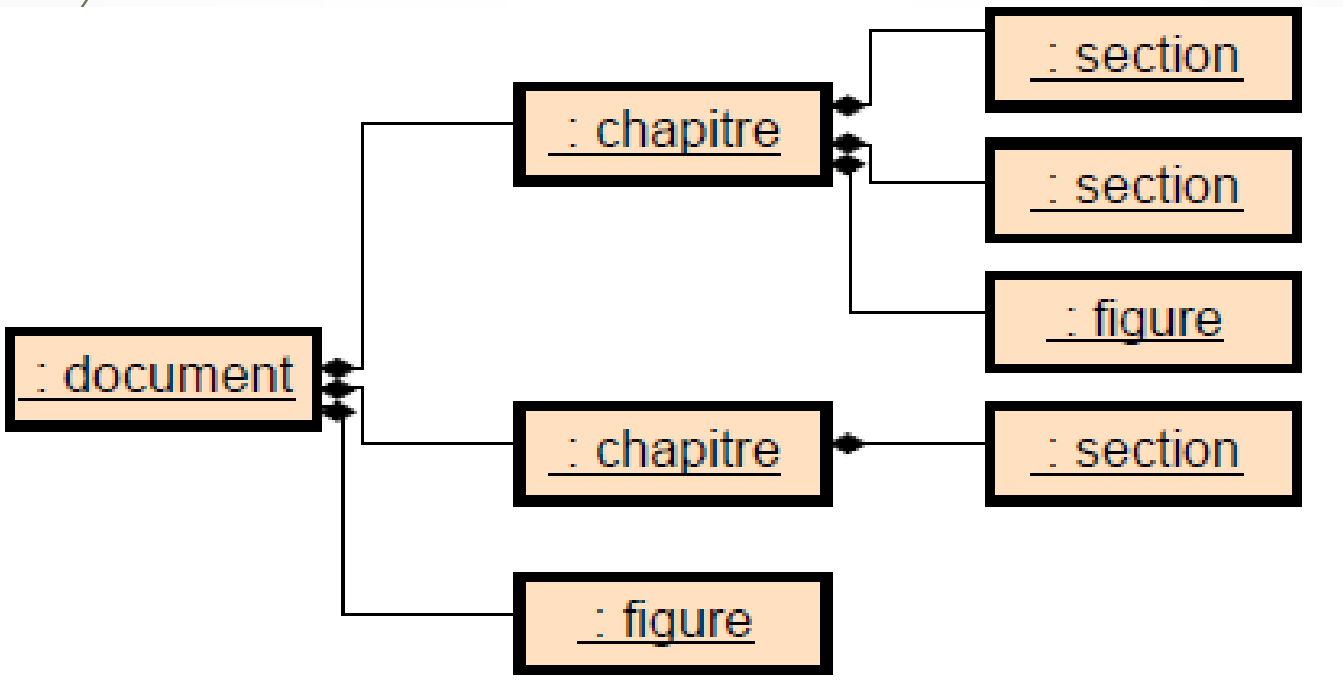
Diagrammes de classes et d'objets: exemple





Diagrammes de classes et d'objets: exemple





Diagrammes de classes et d'objets: exemple