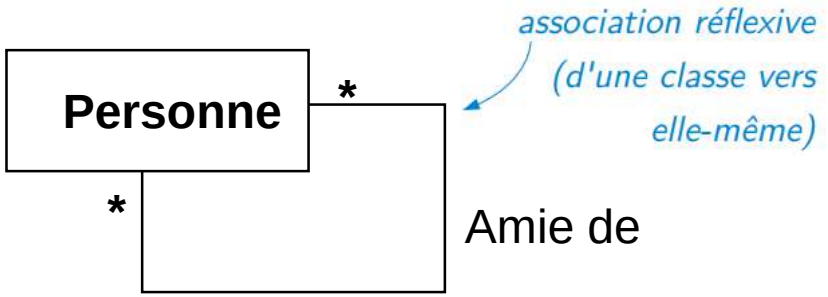
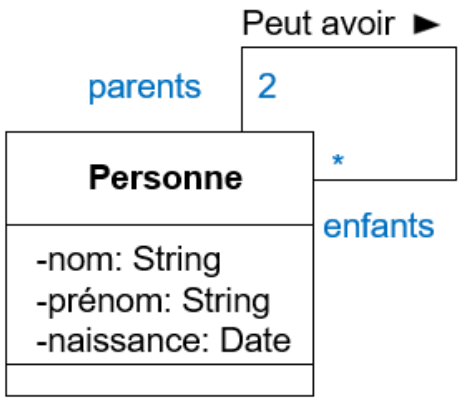
Association réflexive

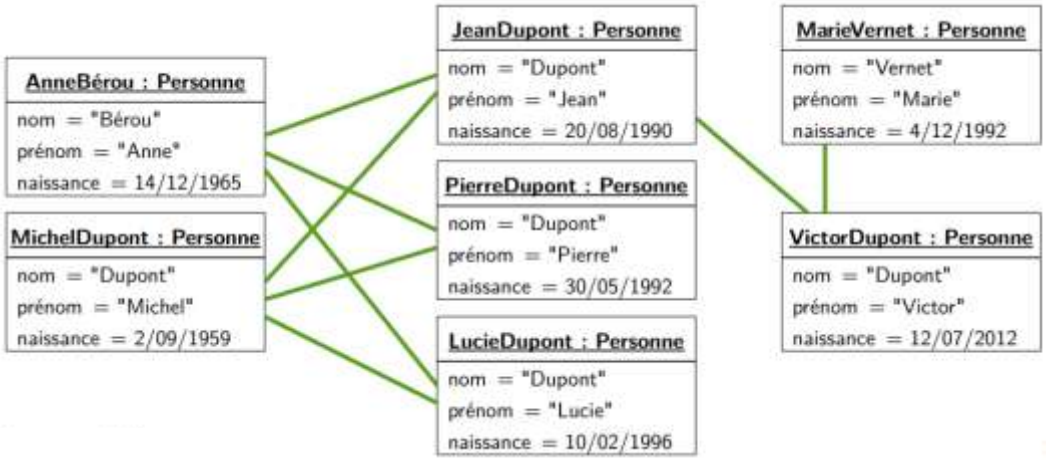
- Le lien existe entre des objets de la même classe
- on considère que l'amitié est réciproque



Association réflexive (2)

- Une personne peut avoir des enfants et des parents





Question

- Est-ce que cette représentation est satisfaisante ?

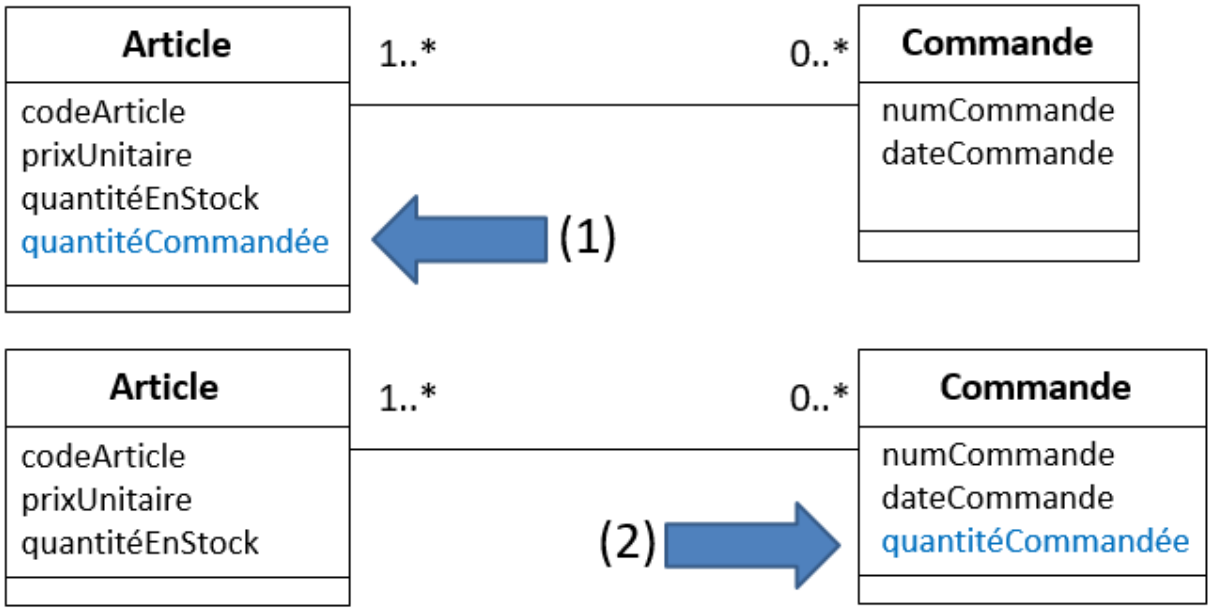
Question (2)

- Est-ce que cette représentation est satisfaisante ?

- Non
  - ✓ Il faut tenir compte de la quantité commandée

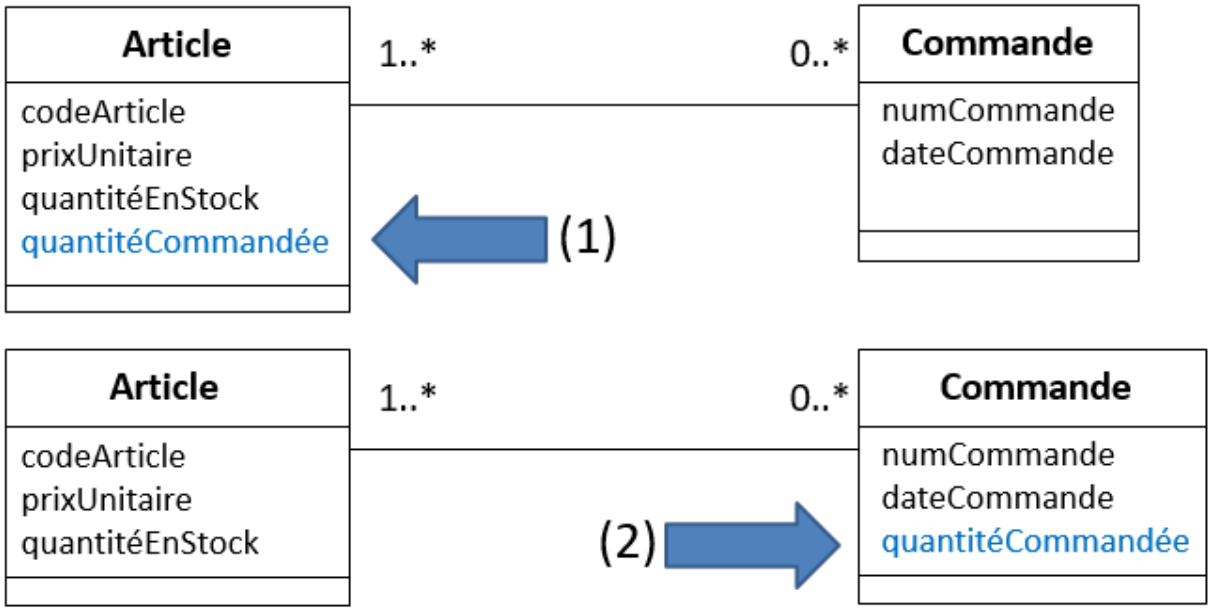
- Comment peut-on représenter la quantité commandée?

Question (3)



- (1) ou (2) ?

Question (4)



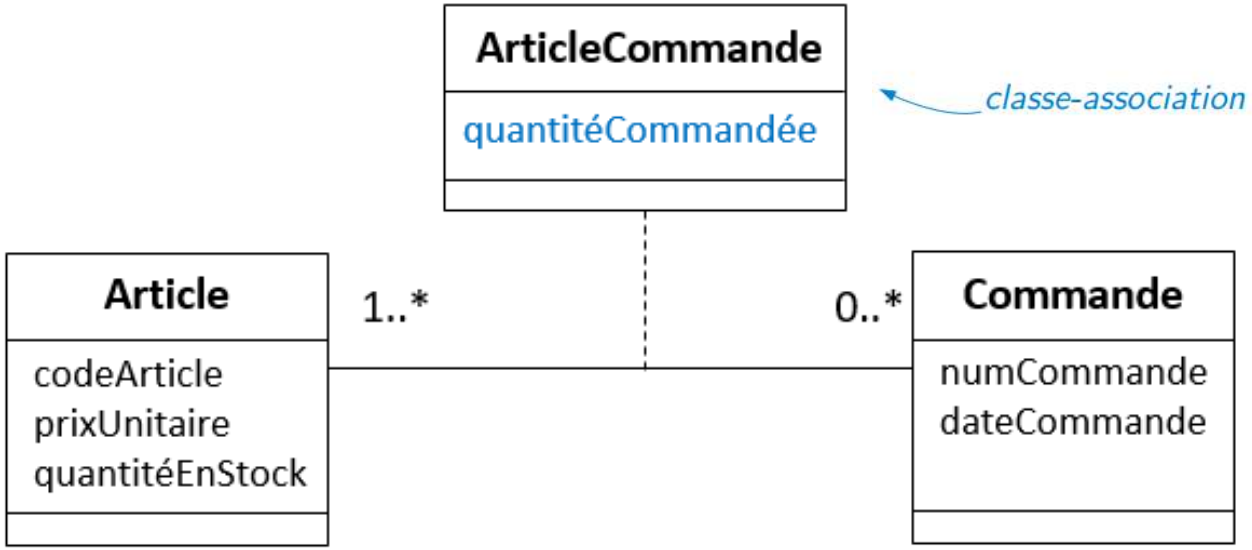
(1) Une seule quantité commandée pour tous ceux qui commandent le même article

(2) Une seule quantité commandée pour tous les articles d'une même commande

Question - Réponse

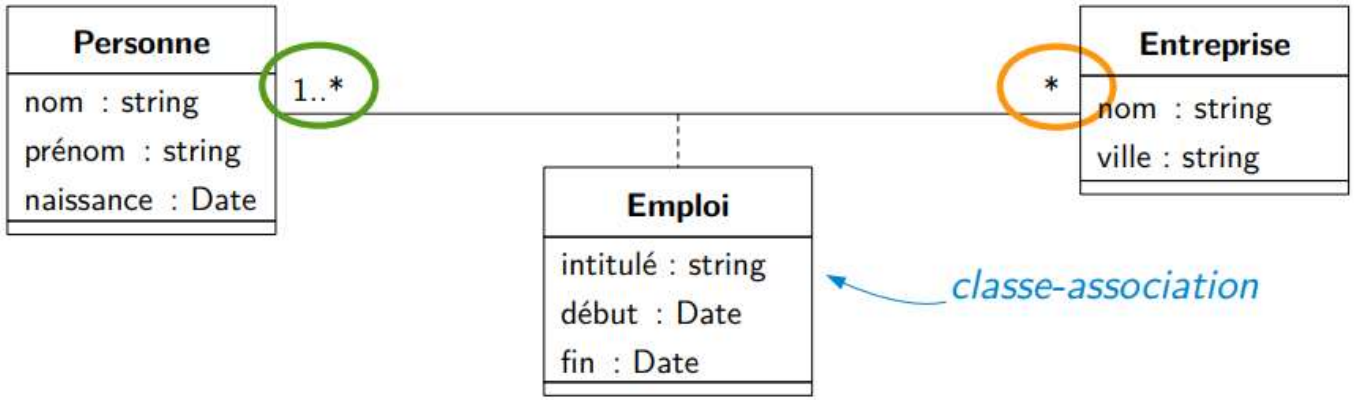
- La quantité commandée ne peut être ni dans Article ni dans Commande

- Solution : Créer une classe d'association



Classe - Association

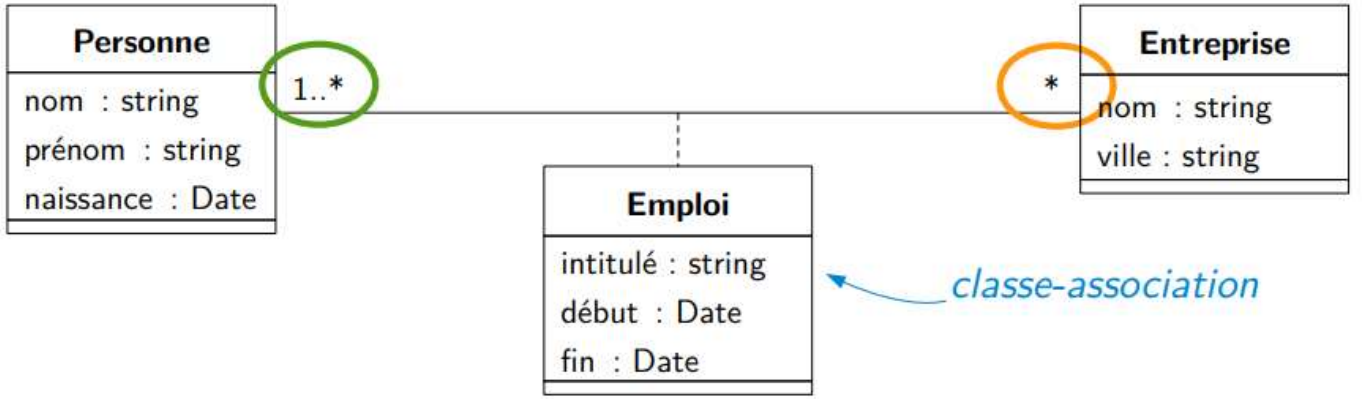
- Nécessaire lorsqu'une association possède ces propres propriétés
- décrire soit des attributs soit des opérations propres à l'association

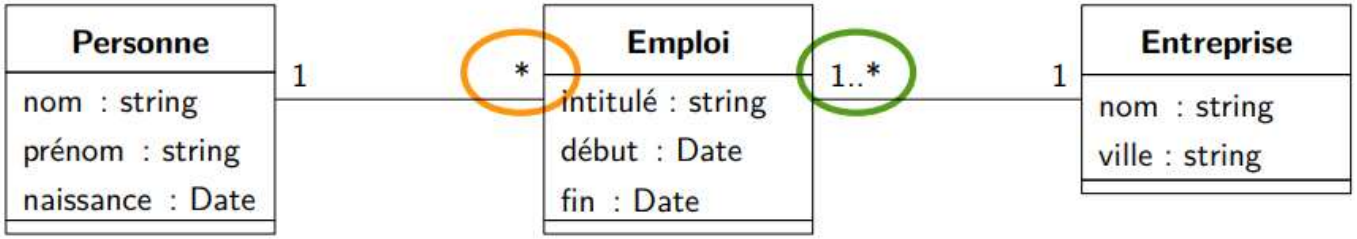


- Instance unique de la classe-association pour chaque lien entre objets

Classe – Association (2)

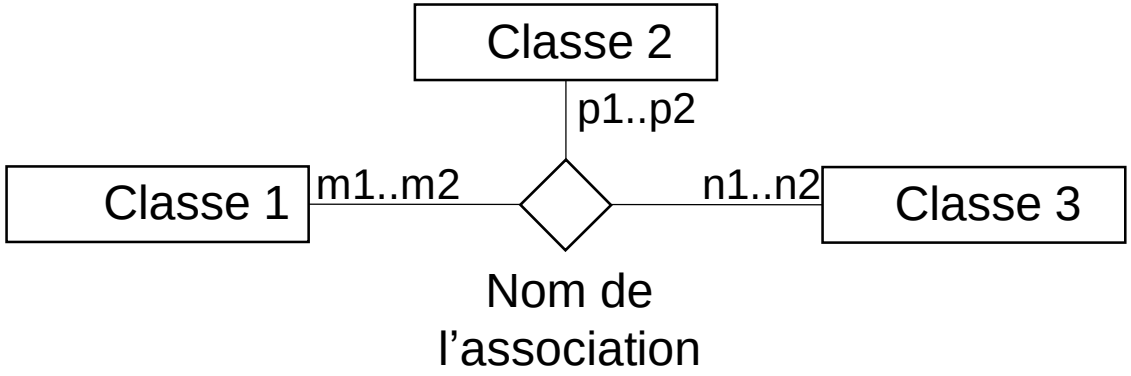
- La classe association peut être remplacée par des associations binaires





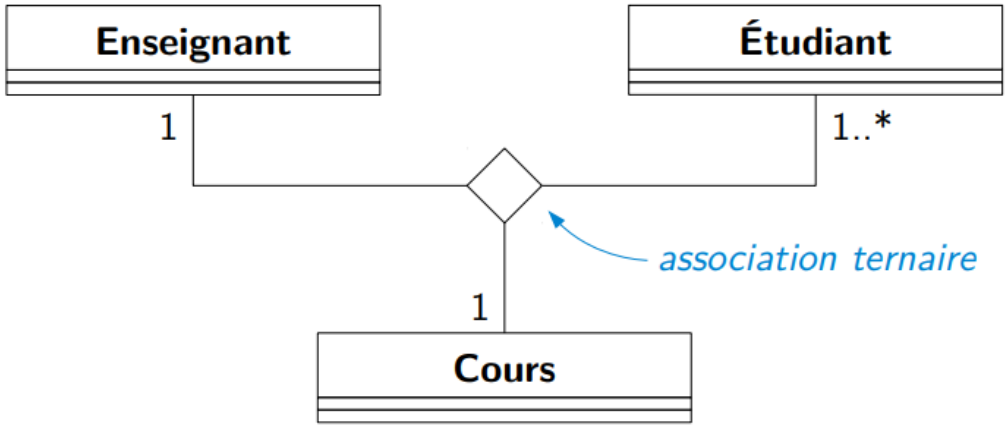
Association n-aire

- Association reliant plus de deux classes
  - ✓ représentée en utilisant un losange



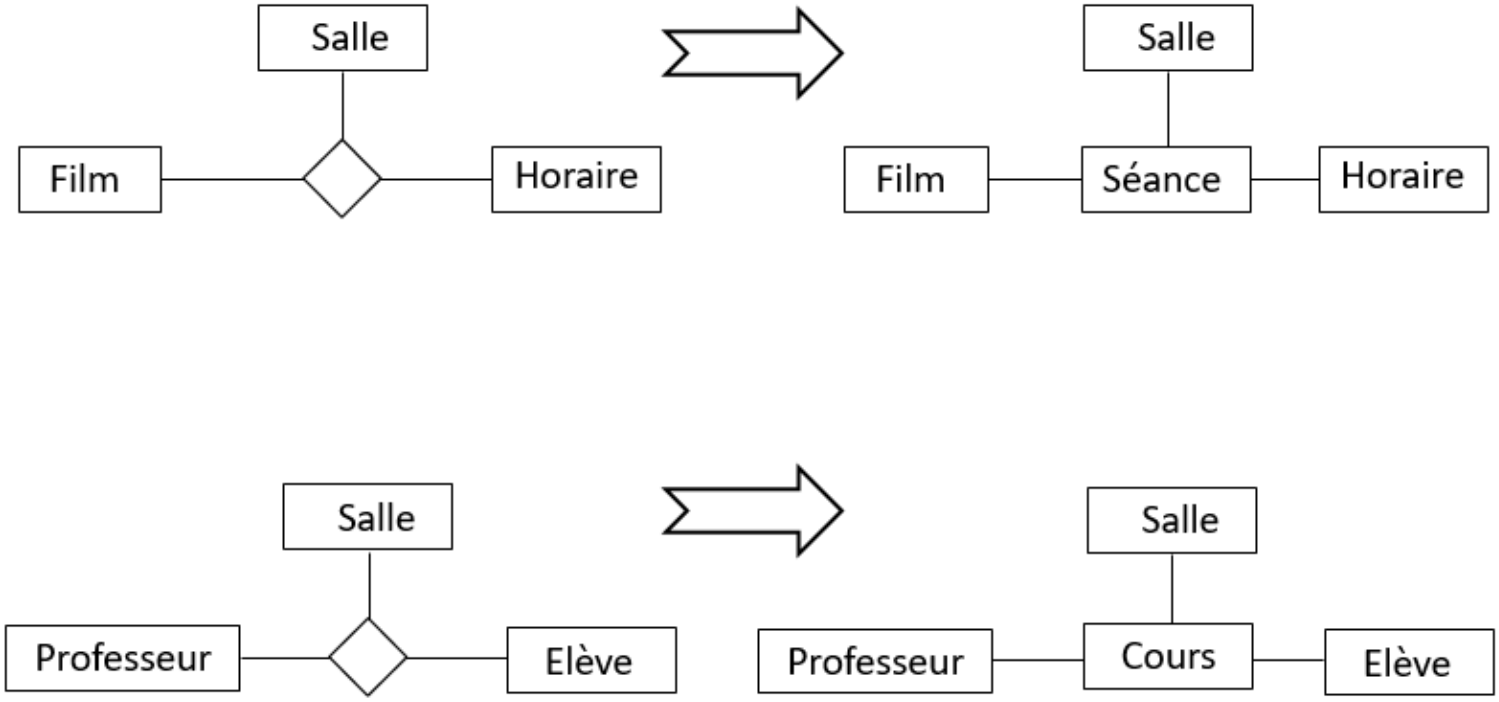
- Les cardinalités se lisent:
  - ✓ Pour un couple instance de classe 1 et instance de classe 2, il y a au minimum *n1* et au maximum *n2* instances de classe 3

Association n-aire (2)



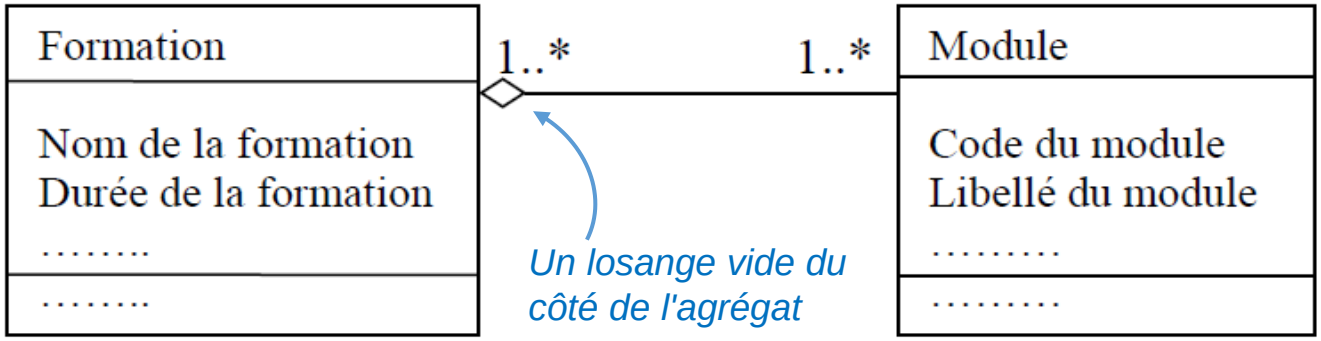
- Très peu utilisée
  - ✓ Imprécise
  - ✓ difficile à interpréter
  - ✓ souvent source d'erreur
- Utilisée dans la plupart du temps pour esquisser la modélisation au début du projet, puis elle est remplacée par des associations binaires afin de lever toute ambiguïté

Association n-aire - Conversion



Agrégation

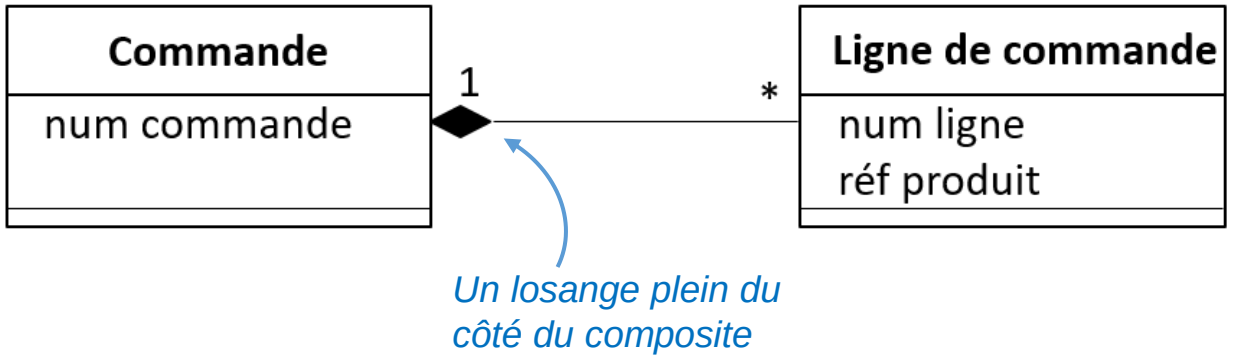
- Décrit une relation d'inclusion entre une partie (agrégé) et un tout (l'agrégat)
  - ✓ Simple regroupement de parties dans un tout



✓ La suppression d'une formation ne conduit pas automatiquement à la suppression des modules

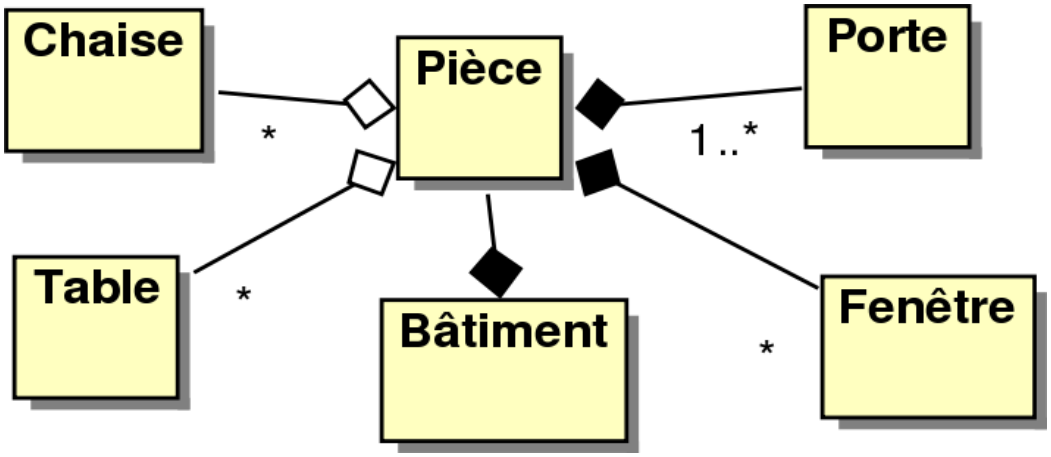
Composition

- Représente une relation composite/composants
  - ✓ Une forme forte d'agrégation
- Les cycles de vie de l'objet composite et ses composants sont liés:
  - ✓ La suppression du composite mène à la suppression de ses composants
- La composition est exclusive :
  - ✓ Un composant ne peut être liée qu'à un seul objet composite



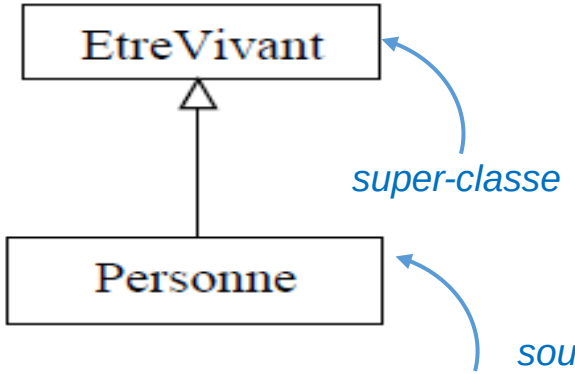
  - ✓ La suppression d'une commande conduira obligatoirement à la suppression de toutes ses lignes

Agrégation - Composition



Héritage

- Un partage hiérarchique de propriétés et de comportements (attributs et opérations)
  - ✓ Construire une classe à partir d'une classe plus haute dans la hiérarchie
  - ✓ Réutiliser le code
  - ✓ Eviter la duplication d'attributs et de méthodes



super-classe

Personne Est_un EtreVivant

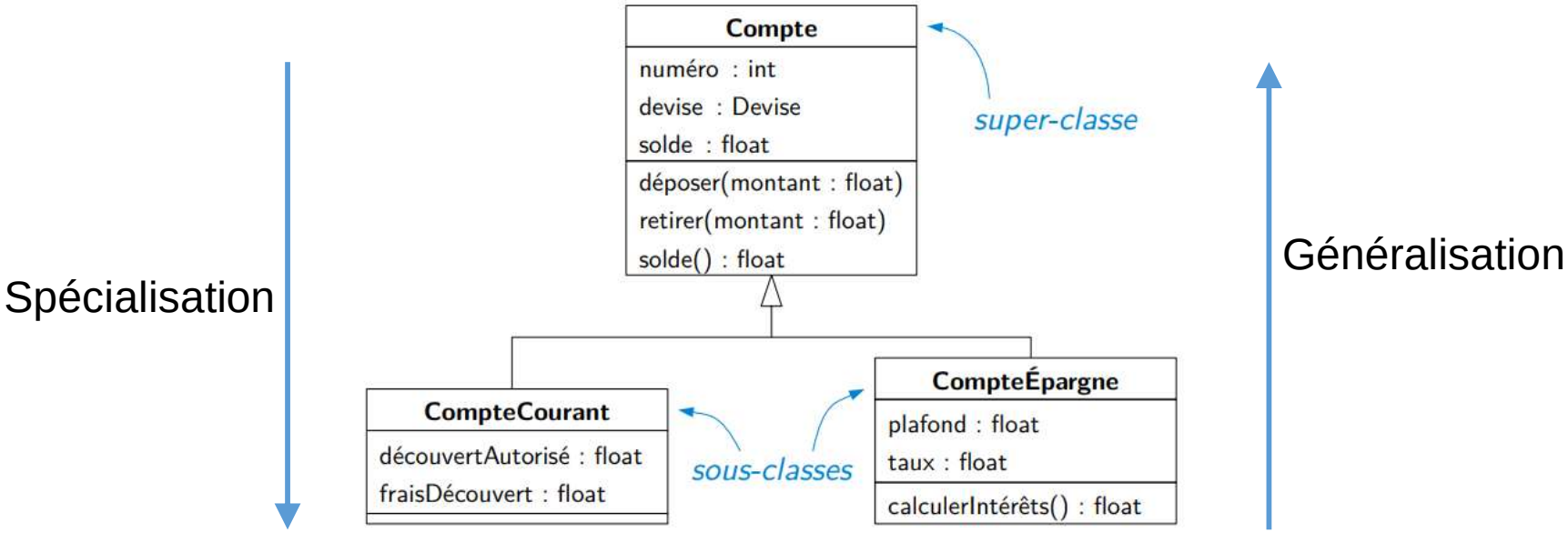
sous-classe

Héritant de la classe EtreVivant

- Mis en œuvre grâce à deux propriétés qui sont : la généralisation et la spécialisation

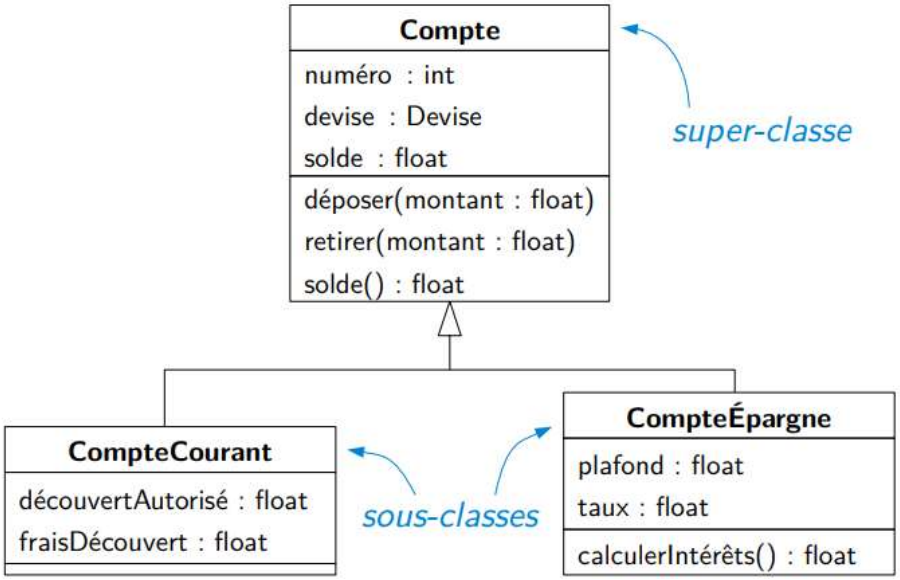
Généralisation et spécialisation

- Spécialisation :
  - ✓ Raffinement d'une classe en une sous-classe
- Généralisation :
  - ✓ Abstraction d'un ensemble de classes en super-classe



Héritage et propriétés/Associations

- La classe enfant possède toutes les propriétés de ses classes parents (attributs et opérations)
- Toutefois, elle n'a pas accès aux propriétés privées



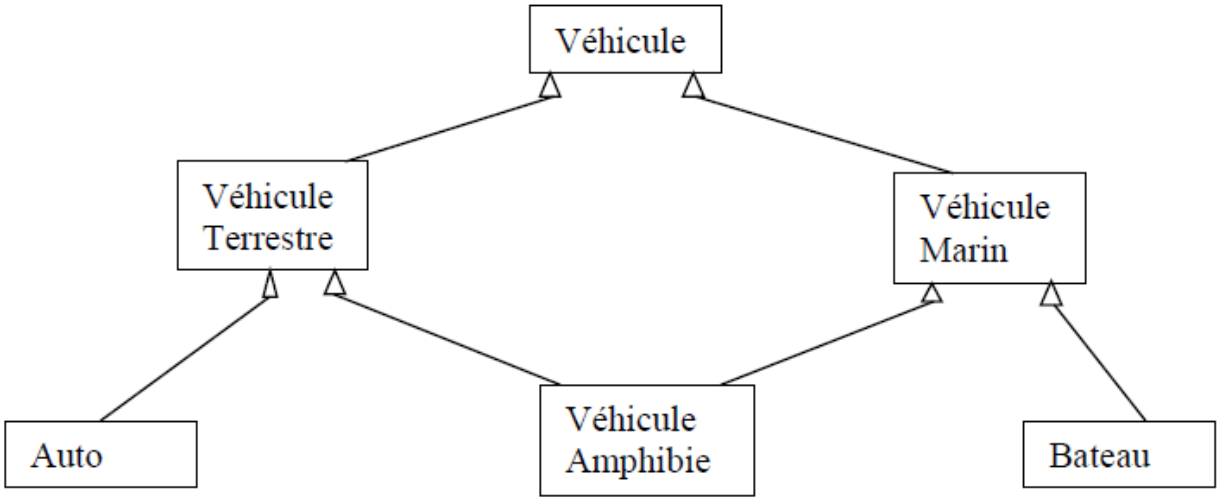
- Toutes les associations de la classe parent s'appliquent, par défaut, aux classes dérivées (classes enfant)

Principe de substitution

- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue
- Exemple:
  - ✓ Toute opération acceptant un objet d'une classe Animal doit accepter tout objet de la classe Chat (l'inverse n'est pas toujours vrai)

Héritage simple et multiple

- Une classe peut avoir plusieurs classes parents



- C++ permet son implantation effective
- Java ne le permet pas

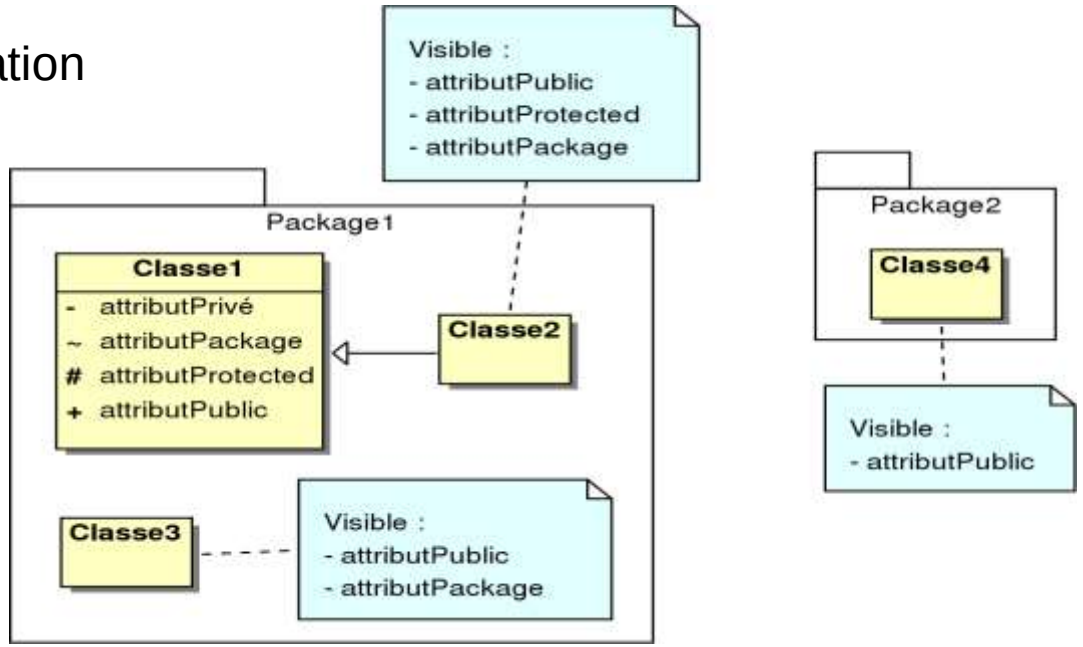
Encapsulation

- Encapsulation
  - ✓ Un principe de conception consistant à protéger le cœur d'un système des accès intempestifs venant de l'extérieur
- Niveaux de visibilité (Rappel)
  - ✓ (+) public : visible par tous les autres objets
  - ✓ (-) private : visible seulement depuis l'intérieur de l'objet
  - ✓ (#) protected : visible par certains objets (les descendants de la classe)
  - ✓ (package ou ~ ou rien) : visibilité à l'intérieur du package

Package (paquetage)

- Contiennent des éléments de modèle de haut niveau, comme des classes, des diagrammes de cas d'utilisation ou d'autres packages
- On organise les éléments modélisés en packages et sous-packages

- Exemple d'encapsulation

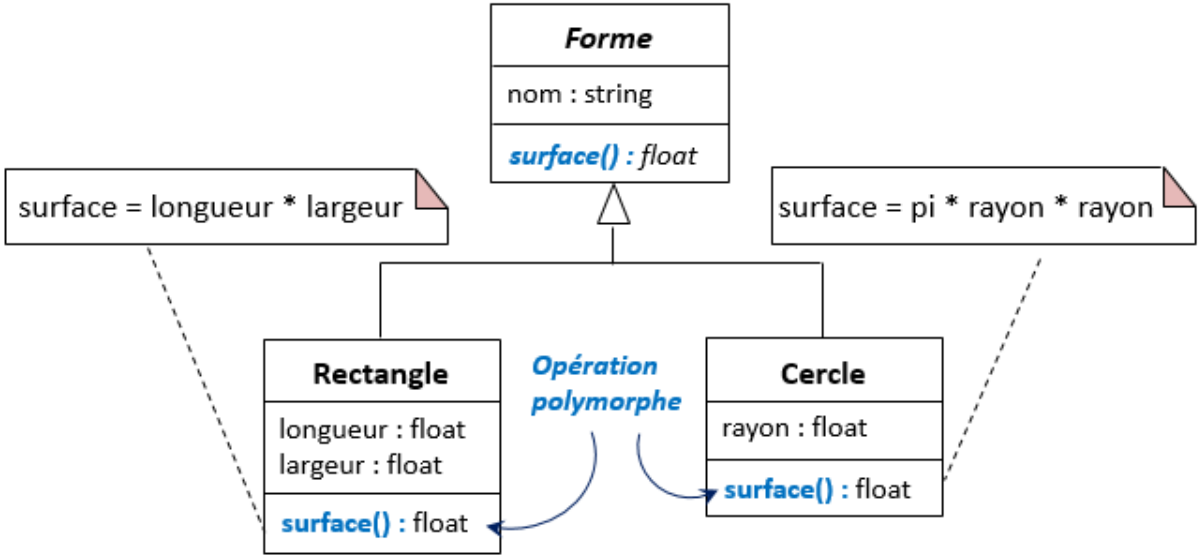


- Les modificateurs d'accès sont également applicables aux opérations

Polymorphisme

- Un mécanisme permettant à des objets de réaliser les opérations d'une interface commune de façon propre
  - ✓ Chaque sous-classe peut modifier localement l'implémentation des ses opérations pour considérer le particularisme de son niveau d'abstraction

- Possibilité de définir plusieurs opérations avec le même nom



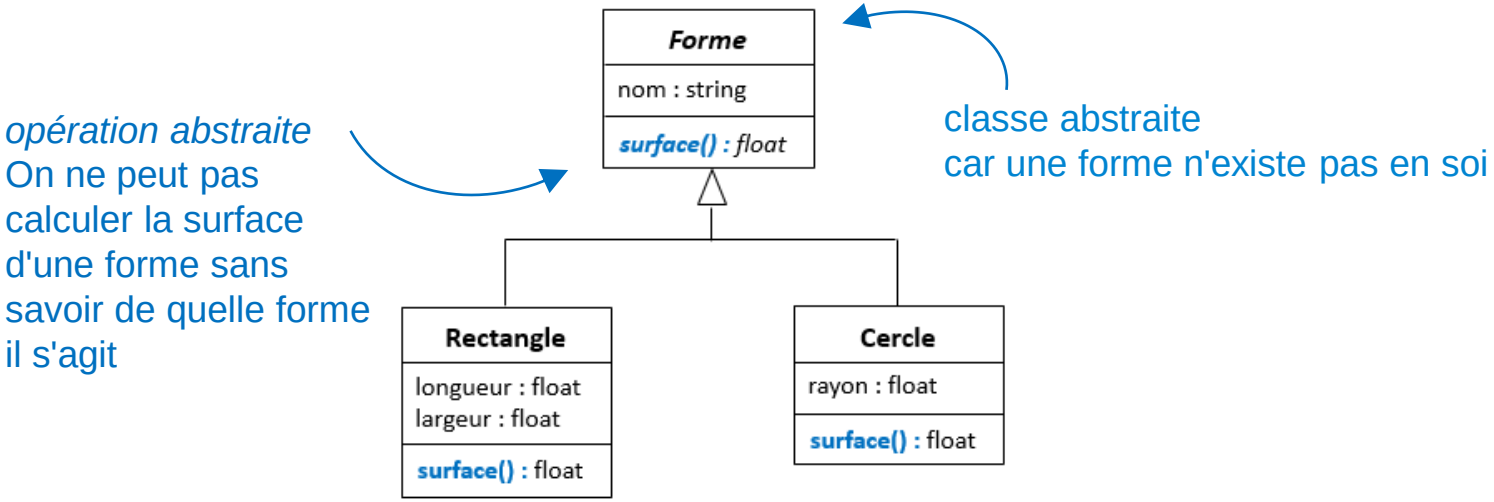
Polymorphisme (2)

- Surcharge (overload)
  - ✓ Dans une classe, plusieurs méthodes portant le même nom et avec des signatures différentes
  - ✓ N’est pas autorisée par certains langages

- Redéfinition
  - ✓ Redéfinir une méthode héritée dans la classe fille avec une signature identique

Classe abstraite

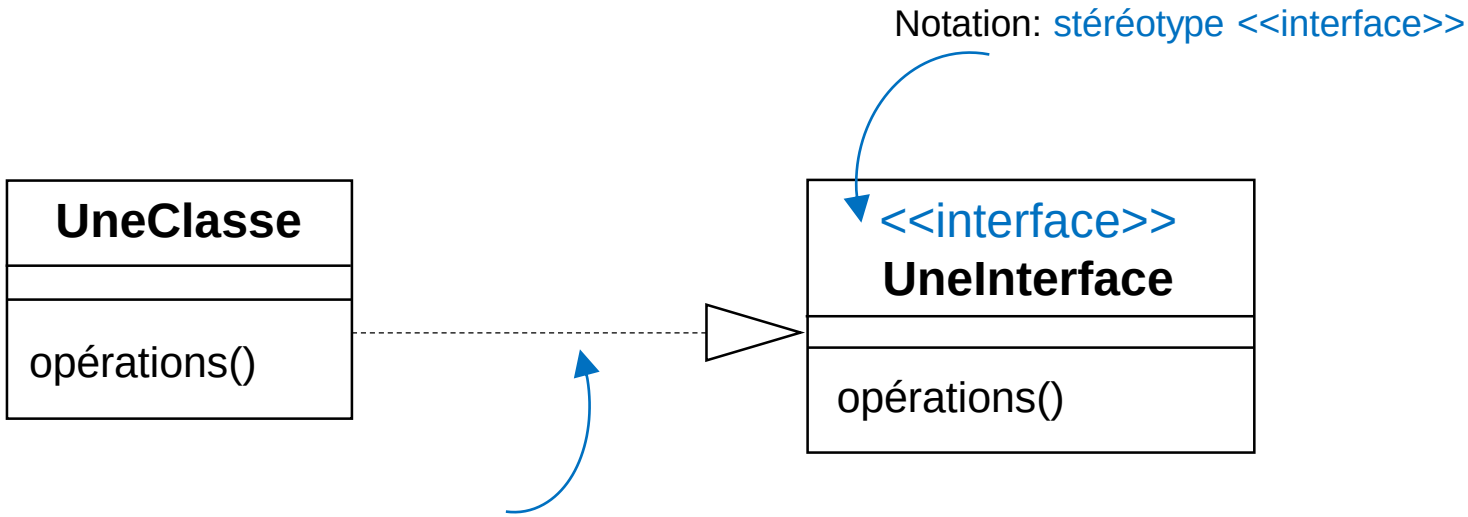
- Une classe qui n'a pas d'instances
  - ✓ Certaines opérations sont abstraites (ne sont pas implémentées)
    - Notées en italique
  - ✓ Nom de la classe en italique (ou stéréotype « abstract »)



- Si une classe contient une méthode abstraite, elle doit être déclarée abstraite
- Si une classe hérite d'une classe abstraite, elle doit implémenter les méthodes abstraites

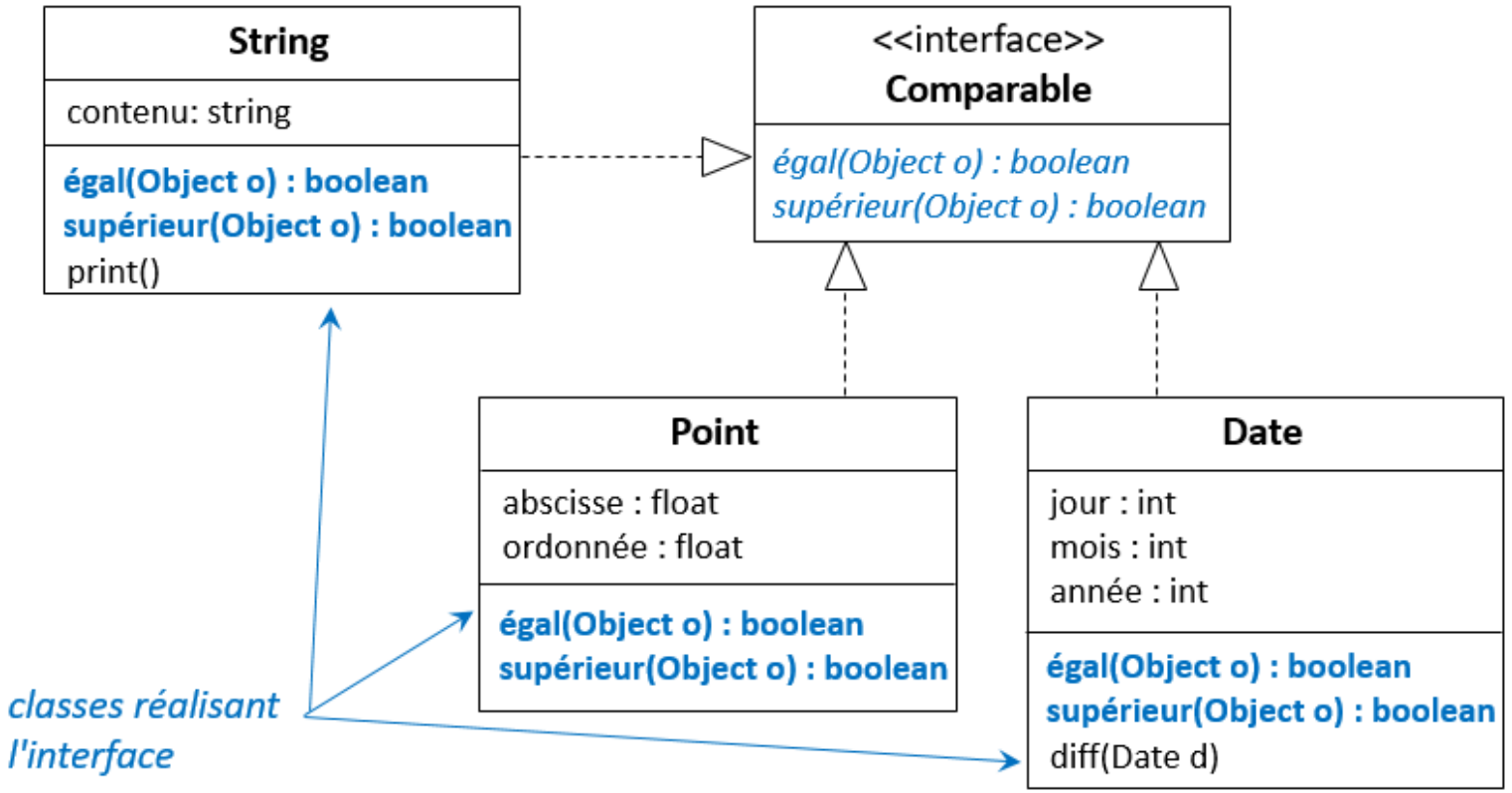
Interface

- Un contrat à respecter par les classes qui réalisent l'interface
  - ✓ Le contrat est constitué d'une liste d'opérations
- Comme une classe abstraite dont toutes les opérations sont abstraites
  - ✓ Mais pas une classe,
  - ✓ Ne peut pas servir à créer des objets



On utilise une relation de type réalisation entre une interface et une classe qui l'implémente

Interface (2)

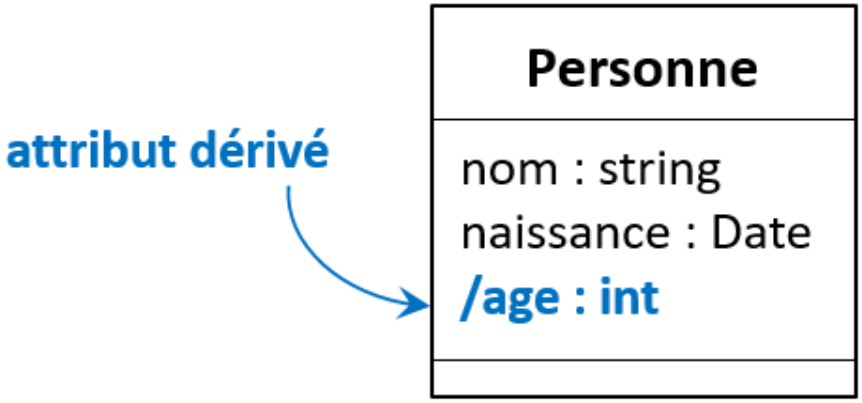

String
contenu: string
égal(Object o) : boolean
supérieur(Object o) : boolean
print()
<<interface>>
Comparable
égal(Object o) : boolean
supérieur(Object o) : boolean
Point
abscisse : float
ordonnée : float
égal(Object o) : boolean
supérieur(Object o) : boolean
Date
jour : int
mois : int
année : int
égal(Object o) : boolean
supérieur(Object o) : boolean
diff(Date d)
classes réalisant
l'interface

Interface (3)

- Une classe qui réalise une interface doit implémenter toutes ses opérations
- Une classe peut réaliser plusieurs interfaces
- Une interface peut hériter d'autres interfaces

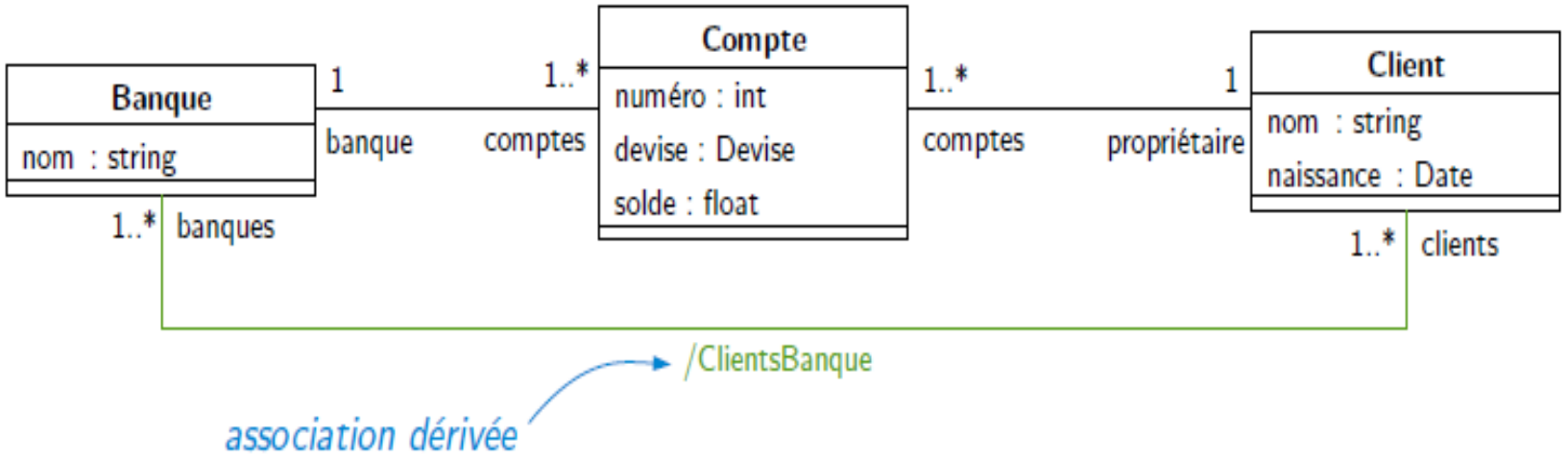
Attribut dérivé

- Peut être calculé à partir d'autres informations du système (d'autres attributs et des formules de calcul)
  - Notation : /attribut
  - Peut nécessiter des informations de plusieurs classes
  - Lors de la conception, il peut être utilisé comme marqueur jusqu'à ce qu'on puisse déterminer les règles à lui appliquer



Association dérivé

- Peut être calculé à partir d'autres informations du système
- Notation : /association



Attribut de classe (static)

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

un attribut de classe est souligné

Opération de classe

- Semblable aux attributs de classe
  - ✓ Une propriété de la classe, et non de ses instances
  - ✓ N'a pas accès aux attributs d'objets de la classe