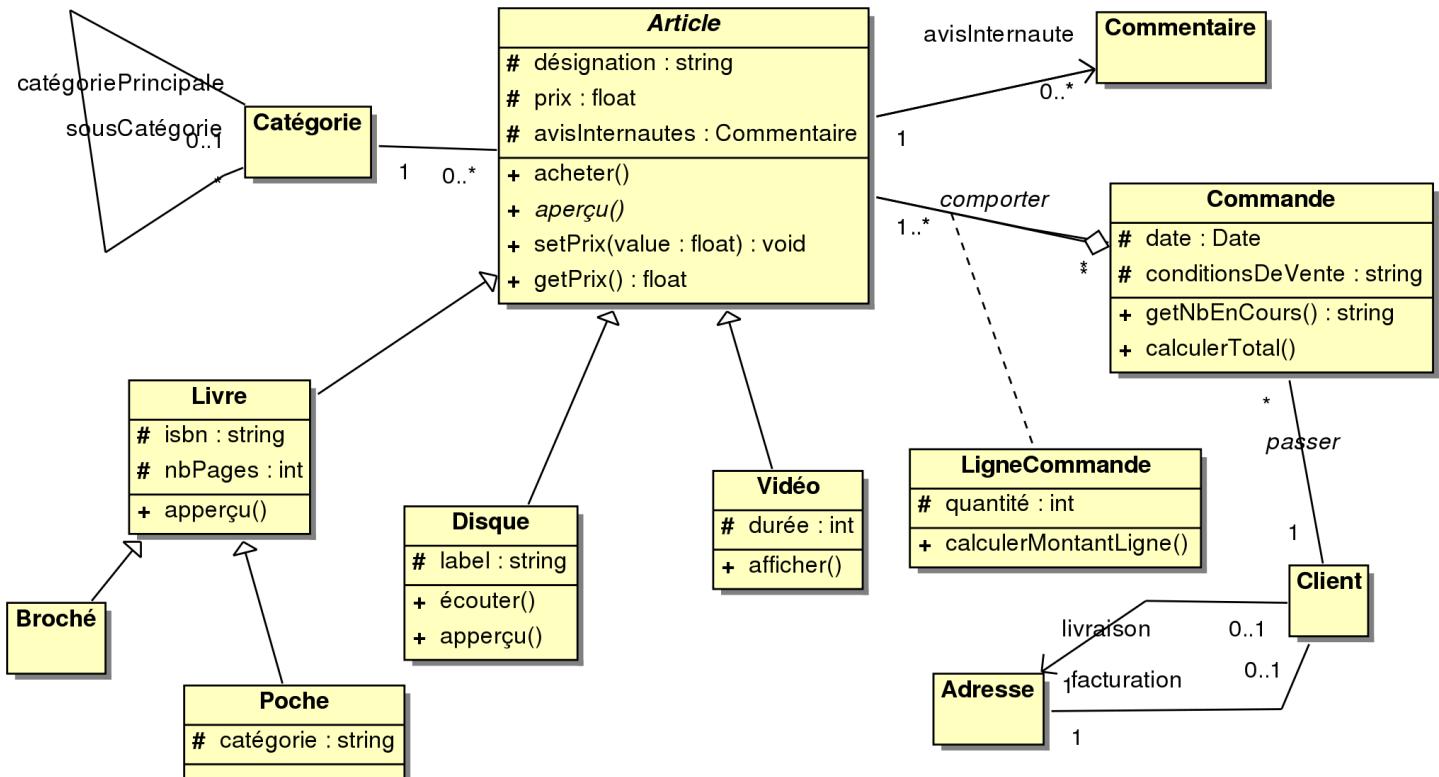


# Objectif

- Les **diagrammes de cas d'utilisation** modélisent à QUOI sert le système.
- Le système est composé d'objets qui interagissent entre eux et avec les acteurs pour réaliser ces cas d'utilisation.
- Les **diagrammes de classes** permettent de spécifier la structure et les liens entre les objets dont le système est composé.

# Exemple de diagramme de classes



# Concepts et instances

- Une **instance** est la concrétisation d'un **concept abstrait**.
  - Concept : Stylo
  - Instance : le stylo que vous utilisez à ce moment précis est une instance du concept stylo : il a sa propre forme, sa propre couleur, son propre niveau d'usure, etc.
- Un **objet** est une instance d'une **classe**
  - Classe : Vidéo
  - Objets : Pink Floyd (Live à Pompey), 2001 Odyssée de l'Espace etc.

Une classe décrit un *type* d'objets concrets.

- Une classe spécifie la manière dont tous les objets de même type seront décrits (désignation, label, auteur, etc).
- Un **lien** est une instance d'**association**.
  - Association : Concept « avis d'internaute » qui lie commentaire et article
  - Lien : instance [Jean avec son avis négatif], [Paul avec son avis positif]

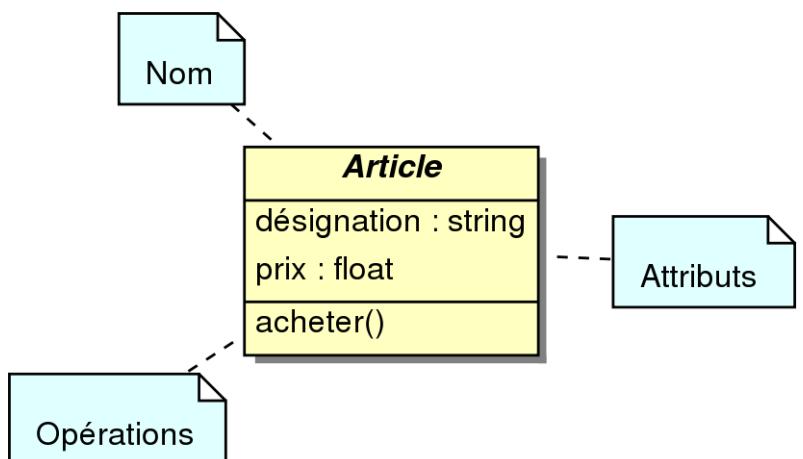
# Classes et objets

- Une **classe** est la description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun. Elle spécifie l'ensemble des caractéristiques qui composent des objets de même type.

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

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

- D'autres compartiments peuvent être ajoutés : responsabilités, exceptions, etc.



## Propriétés : attributs et opérations

- Les attributs et les opérations sont les **propriétés** d'une classe. Leur nom commence par une minuscule.
  - Un **attribut** décrit une donnée de la classe.
    - Les types des attributs et leurs initialisations ainsi que les modificateurs d'accès peuvent être précisés dans le modèle
    - Les attributs prennent des valeurs lorsque la classe est instanciée : ils sont en quelque sorte des « variables » attachées aux objets.
  - Une **opération** est un service offert par la classe (un traitement que les objets correspondant peuvent effectuer).

## Compartiment des attributs

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

```
modifAcces nomAtt:nomClasse[multi]=valeurInit
```

<i>Article</i>
# désignation : string
# prix : float = -1
# avisInternautes : Commentaire [0..*]

## Compartiment des opérations

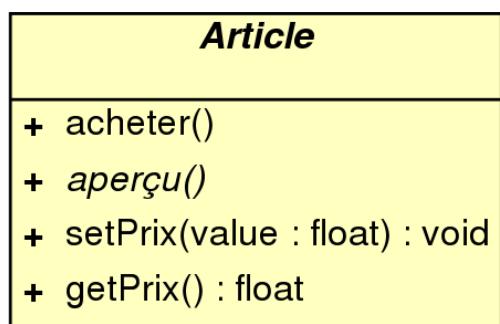
Une opération est définie par son nom ainsi que par les types de ses paramètres et le type de sa valeur de retour.

- La **syntaxe de la déclaration d'une opération** est la suivante :

```
modifAcces nomOperation(parametres):ClasseRetour
```

- La **syntaxe de la liste des paramètres** est la suivante :

```
nomClasse1 nomParam1, ..., nomClasseN nomParamN
```

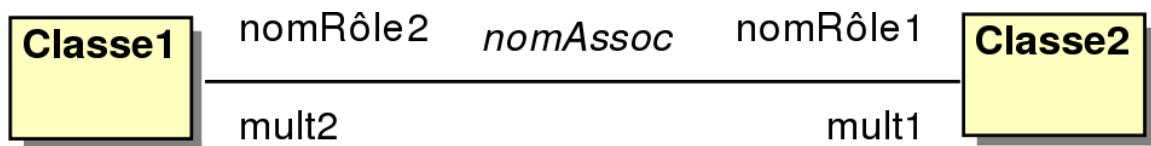


## Relations entre classes

- Une **relation d'héritage** est une relation de généralisation/specialisation permettant l'abstraction.
- Une **dépendance** est une relation unidirectionnelle exprimant une dépendance sémantique entre les éléments du modèle (flèche ouverte pointillée).
- Une **association** représente une relation sémantique entre les objets d'une classe.
- Une **relation d'agrégation** décrit une relation de contenance ou de composition.

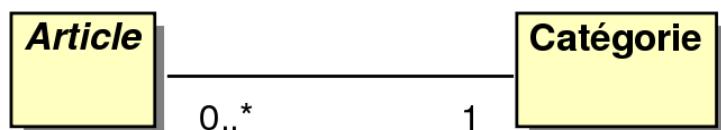
# Association

- Une **association** est une relation structurelle entre objets.
  - Une association est souvent utilisée pour représenter les liens possibles entre objets de classes données.
  - Elle est représentée par un trait entre classes.



## Multiplicités des associations

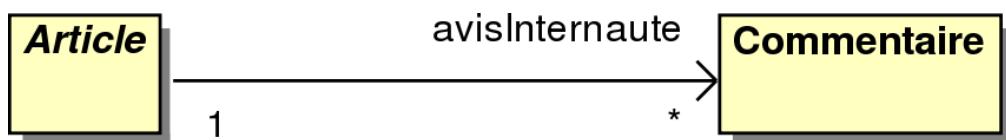
- La notion de **multiplicité** permet le contrôle du nombre d'objets intervenant dans chaque instance d'une association.
  - Exemple :** un article n'appartient qu'à une seule catégorie (1) ; une catégorie concerne plus de 0 articles, sans maximum (\*).



- La syntaxe est MultMin..MultMax.
  - « \* » à la place de MultMax signifie « plusieurs » sans préciser de nombre.
  - « n..n » se note aussi « n », et « 0..\* » se note « \* ».

## Navigabilité d'une association

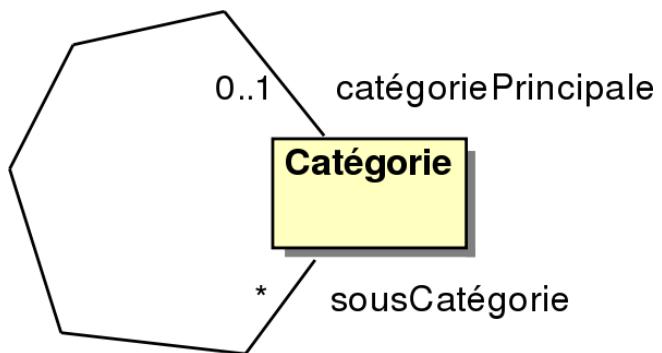
- La **navigabilité** permet de spécifier dans quel(s) sens il est possible de traverser l'association à l'exécution.
- On restreint la navigabilité d'une association à un seul sens à l'aide d'une flèche.



- **Exemple :** Connaissant un article on connaît les commentaires, mais pas l'inverse.
- On peut aussi représenter les associations navigables dans un seul sens par des attributs.
  - **Exemple :** En ajoutant un attribut « avisInternaute » de classe « Commentaire » à *la place de l'association*.

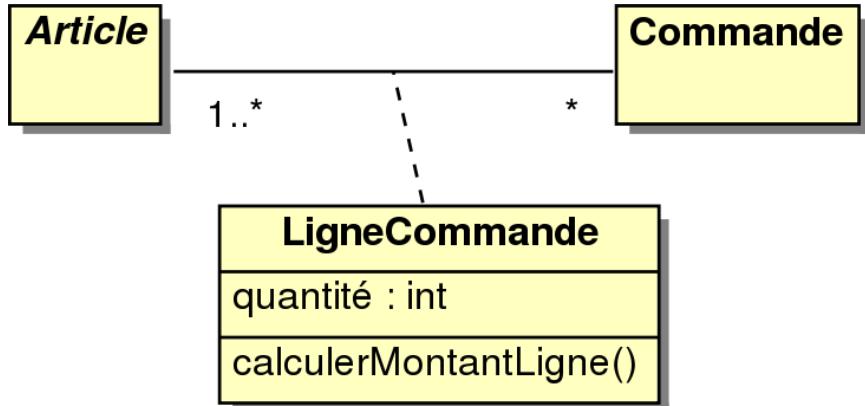
## Associations réflexives

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



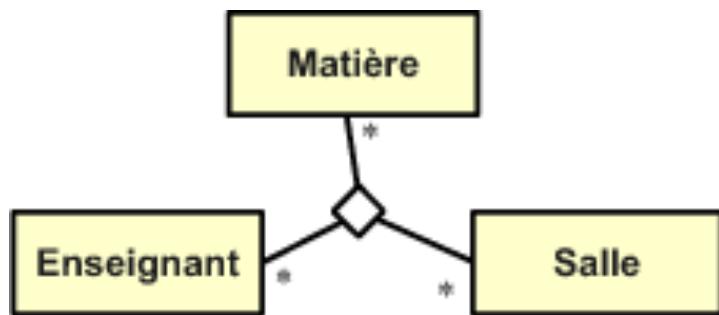
## Classe-association

- Une association peut être raffinée et 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** ».



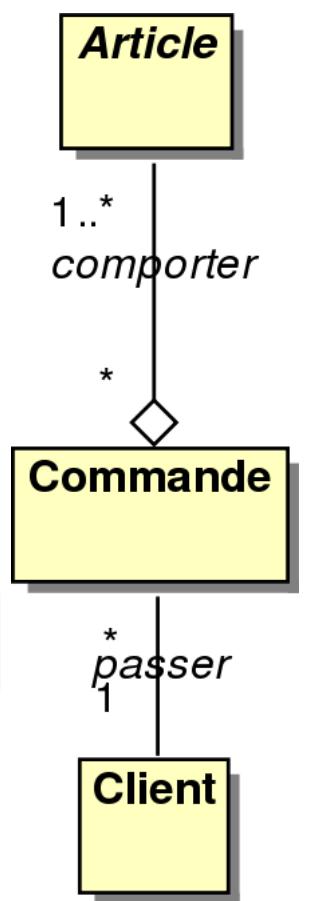
## Associations n-aires

- Une **association n-aire** lie plus de deux classes.
  - Notation avec un losange central pouvant éventuellement accueillir une classe-association.
  - La multiplicité de chaque classe s'applique à une instance du losange.



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

## Association de type agrégation

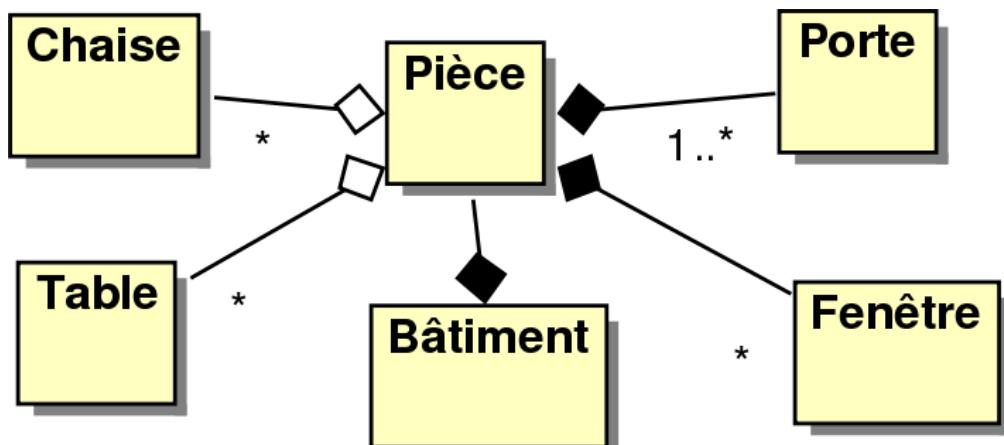


- Une **agrégation** est une forme particulière d'association. Elle représente la relation d'**inclusion** d'un élément dans un ensemble.
- On représente l'agrégation par l'ajout d'un losange vide du côté de l'agrégat.

Une agrégation dénote une relation d'un ensemble à ses parties. L'ensemble est l'**agrégat** et la partie l'**agrégé**.

## Association de type composition

- La relation de **composition** décrit une **contenance structurelle** entre instances. On utilise un losange plein.
- La **destruction** et la **copie** de l'objet composite (l'ensemble) impliquent respectivement la destruction ou la copie de ses composants (les parties).
- Une instance de la partie n'appartient jamais à plus d'une instance de l'élément composite.



## Composition et agrégation

- Dès lors que l'on a une relation du tout à sa partie, on a une relation d'agrégation ou de composition.

La composition est aussi dite « agrégation forte ».

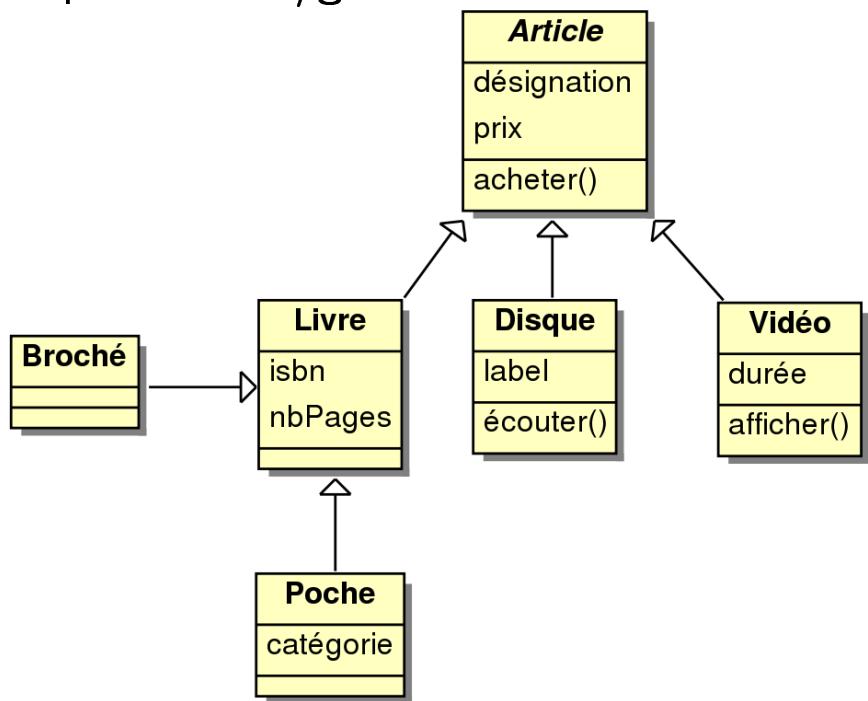
- Pour décider de mettre une composition plutôt qu'une agrégation, on doit se poser les questions suivantes :
  - Est-ce que la destruction de l'objet composite (du tout) implique nécessairement la destruction des objets composants (les parties) ? C'est le cas si les composants n'ont pas d'autonomie vis-à-vis des composites.
  - Lorsque l'on copie le composite, doit-on aussi copier les composants, ou est-ce qu'on peut les « réutiliser », auquel cas un composant peut faire partie de plusieurs composites ?

Si on répond par l'affirmative à ces deux questions, on doit utiliser une composition.

# Relation d'héritage

- L'héritage une relation de spécialisation/généralisation.

- Les éléments spécialisés héritent de la structure et du comportement des éléments plus généraux (attributs et opérations)
- **Exemple :** Par héritage d'Article, un livre a d'office un prix, une désignation et une opération acheter(), sans qu'il soit nécessaire de le préciser



## Implantation de l'héritage en Java

```
class Article {  
    ...  
    void acheter() {  
        ...  
    }  
}  
class Livre  
    extends Article {  
    ...  
}
```

### Attention

Les « extends » Java n'a rien à voir avec le « extend » UML vu pour les cas d'utilisation

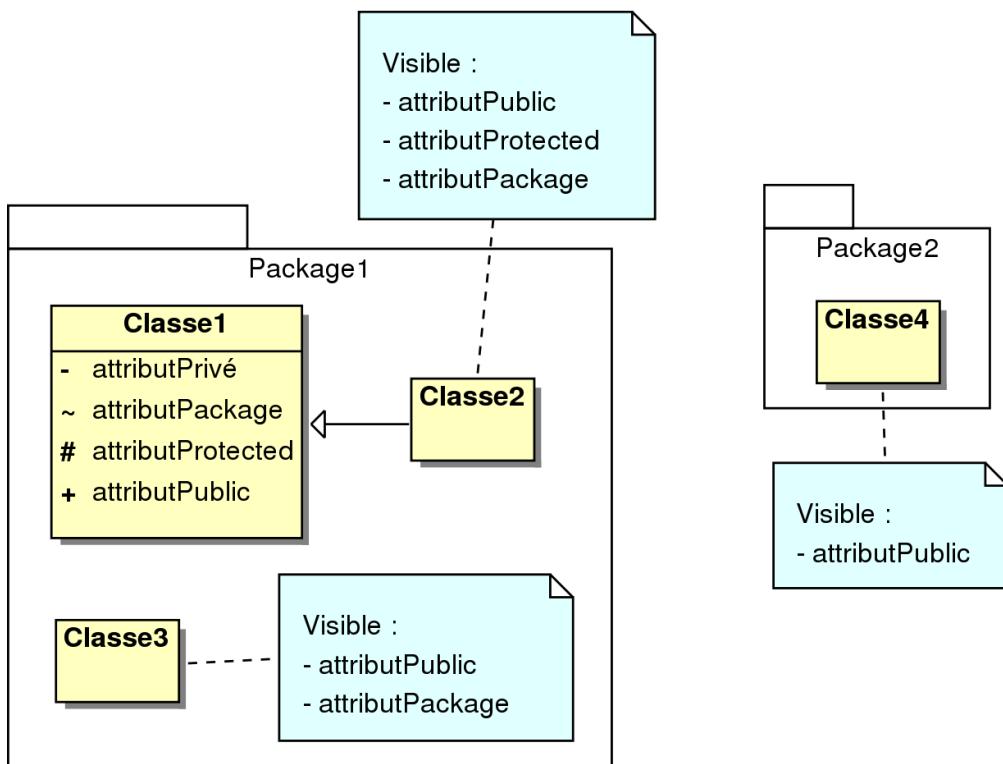
## Encapsulation

- L'**encapsulation** est un principe de conception consistant à protéger le cœur d'un système des accès intempestifs venant de l'extérieur.
- En UML, utilisation de **modificateurs d'accès** sur les attributs ou les classes :
  - **Public** ou « + » : propriété ou classe visible partout
  - **Protected** ou « # ». propriété ou classe visible dans la classe et par tous ses descendants.
  - **Private** ou « - » : propriété ou classe visible uniquement dans la classe
  - **Package**, ou « ~ » : propriété ou classe visible uniquement dans le paquetage
- **Il n'y a pas de visibilité « par défaut ».**

## Package (paquetage)

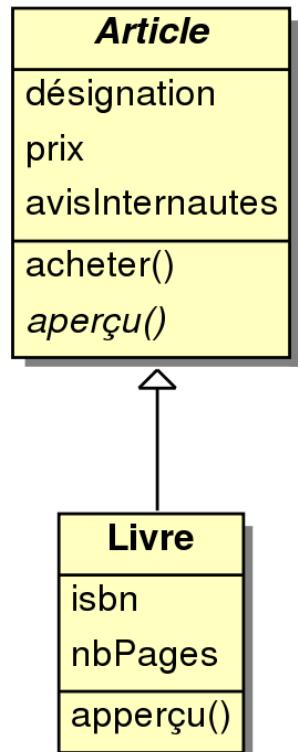
Les packages contiennent des éléments de modèle de haut niveau, comme des classes, des diagrammes de cas d'utilisation ou d'autres packages. On organise les éléments modélisés en packages et sous-packages.

## Exemple d'encapsulation



Les modificateurs d'accès sont également applicables aux opérations.

# Relation d'héritage et propriétés



- La classe enfant possède toutes les propriétés de ses classes parents (attributs et opérations)
  - La classe **enfant** est la classe spécialisée (ici **Livre**)
  - La classe **parent** est la classe générale (ici **Article**)
- Toutefois, elle n'a pas accès aux propriétés privées.

## Terminologie de l'héritage

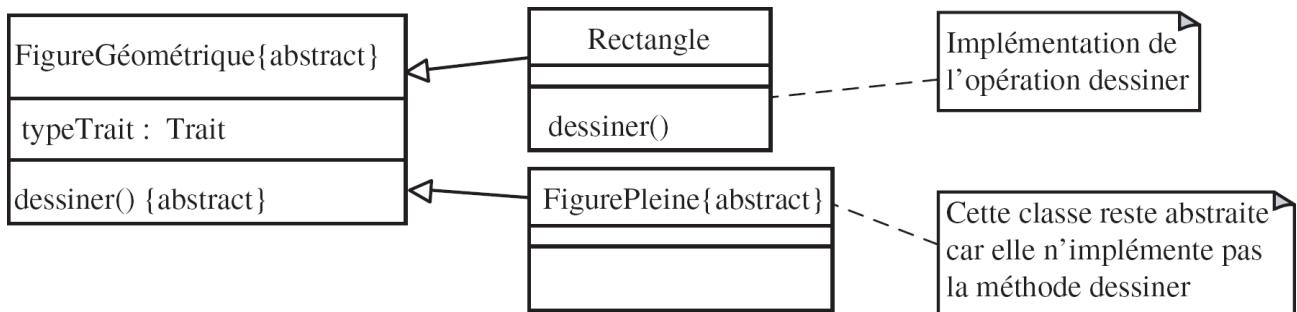
- Une classe enfant peut **redéfinir** (même signature) une ou plusieurs méthodes de la classe parent.
  - Sauf indications contraires, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes.
  - La **surcharge d'opérations** (même nom, mais signatures des opérations différentes) est possible dans toutes les classes.
- 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.
  - Par 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).

## Classes abstraites

- Une méthode est dite **abstraite** lorsqu'on connaît son entête mais pas la manière dont elle peut être réalisée.

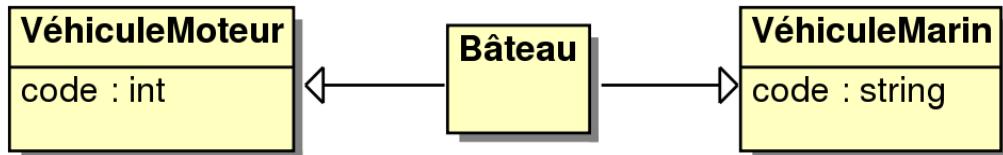
Il appartient aux classes enfant de définir les méthodes abstraites.

- Une classe est dite **abstraite** lorsqu'elle définit au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.



# Héritage multiple

- Une classe peut avoir plusieurs classes parents. On parle alors d'**héritage multiple**.
  - Le langage C++ est un des langages objet permettant son implantation effective.
  - Java ne le permet pas.



# A vous de jouer (1)

## Gestion de la cité U (diagramme de classes)

**Une Cité U est constituée d'un ensemble de bâtiments. Un bâtiment comporte un certain nombre de chambres. La cité peut employer du personnel et est dirigé par l'un des employés. Chaque chambre de la cité se loue à un prix donné (suivant ses prestations).**

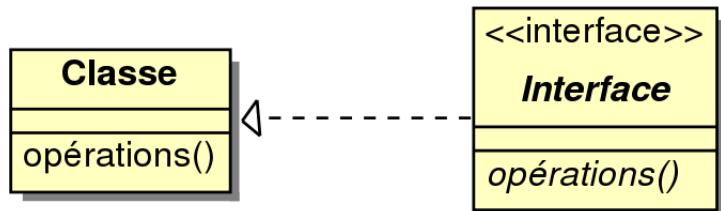
L'accès aux salles de bain est compris dans le prix de la location d'une chambre. Certaines chambres comportent une salle de bain, mais pas toutes. Les hôtes de chambres sans salle de bain peuvent utiliser une salle de bain sur le palier. Ces dernières peuvent être utilisées par plusieurs hôtes.

Une personne peut louer une et une seule chambre et une chambre peut être loué par une ou deux personnes.

Les pièces de la Cité U qui ne sont ni des chambres, ni des salles de bain (hall d'accueil, cuisine...) ne font pas partie de l'étude (hors sujet).

# Interface

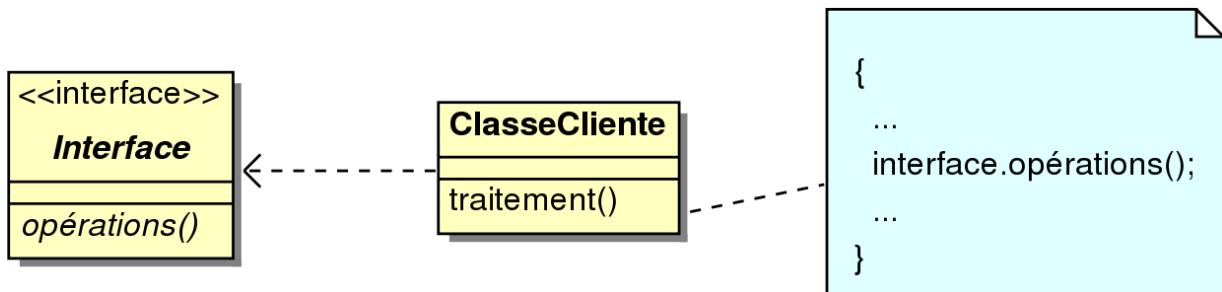
- Le rôle d'une **interface** est de regrouper un ensemble d'opérations assurant un service cohérent offert par un classeur et une classe en particulier.
- Une interface est définie comme une classe, avec les mêmes compartiments. On ajoute le stéréotype « **interface** » avant le nom de l'interface.
- On utilise une relation de type **réalisation** entre une interface et une classe qui l'implémente.



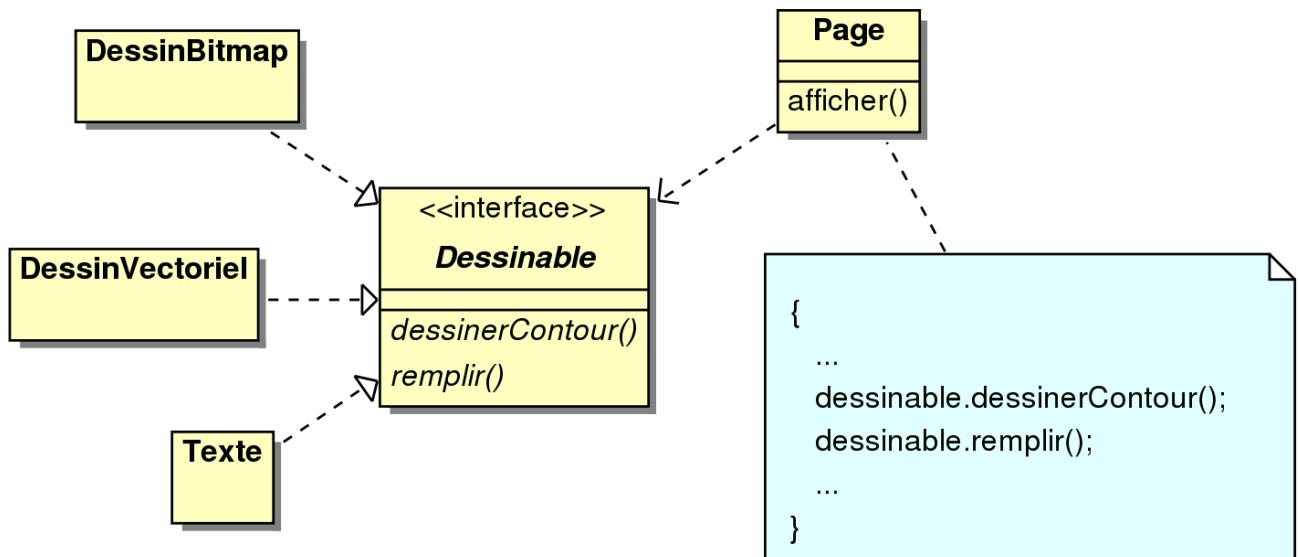
- Les classes implémentant une interface doivent implémenter toutes les opérations décrites dans l'interface

## Classe cliente d'une interface

- Quand une classe dépend d'une interface (**interface requise**) pour réaliser ses opérations, elle est dite « **classe cliente de l'interface** »
- On utilise une relation de dépendance entre la classe cliente et l'interface requise. Toute classe implémentant l'interface pourra être utilisée.



## Exemple d'interface



## Eléments dérivés

- Les **attributs dérivés** peuvent être calculés à partir d'autres attributs et des formules de calcul.
  - Les attributs dérivés sont symbolisés par l'ajout d'un « / » devant leur nom.
  - Lors de la conception, un attribut dérivé peut être utilisé comme marqueur jusqu'à ce que vous puissiez déterminer les règles à lui appliquer.
- Une **association dérivée** est conditionnée ou peut être déduite à partir d'autres autres associations. On utilise également le symbole « / ».

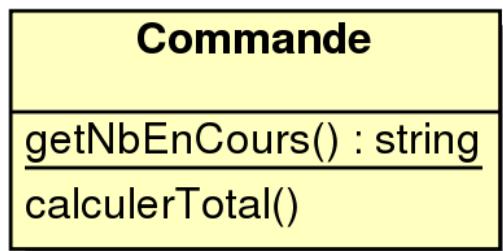
# Attributs de classe

- 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.
- Graphiquement, un attribut de classe est souligné



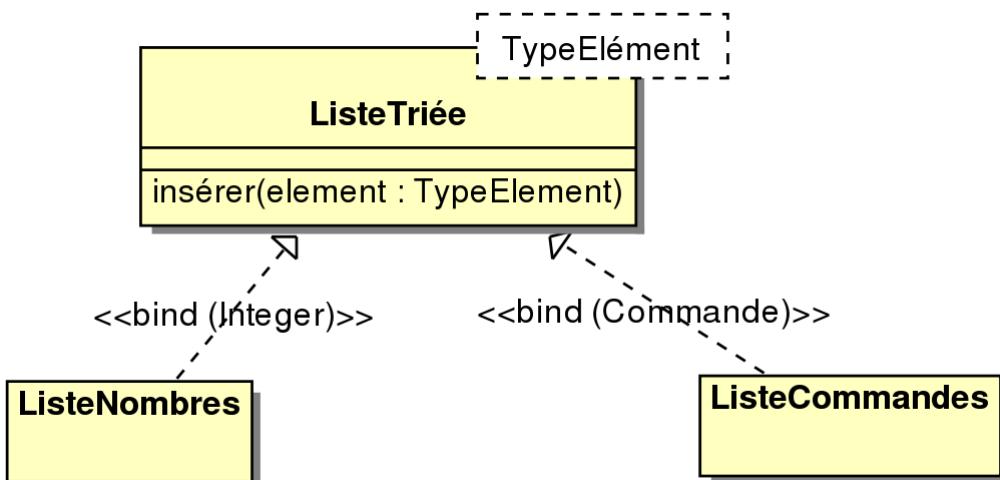
# Opérations de classe

- 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.



# Classe paramétrée

- Pour **définir une classe générique et paramétrable** en fonction de valeurs et/ou de types :
  - Définition d'une classe paramétrée par des éléments spécifiés dans un rectangle en pointillés ;
  - Utilisation d'une dépendance stéréotypée « bind » pour définir des classes en fonction de la classe paramétrée.



- Java5 : **généricité**
- C++ : **templates**

# Plan

## 1 Introduction à la Modélisation Orientée Objet

## 2 Modélisation objet élémentaire avec UML

- Diagrammes de cas d'utilisation
- Diagrammes de classes
- **Diagrammes d'objets**
- Diagrammes de séquences

## 3 UML et méthodologie

## 4 Modélisation avancée avec UML

## 5 Bonnes pratiques de la modélisation objet

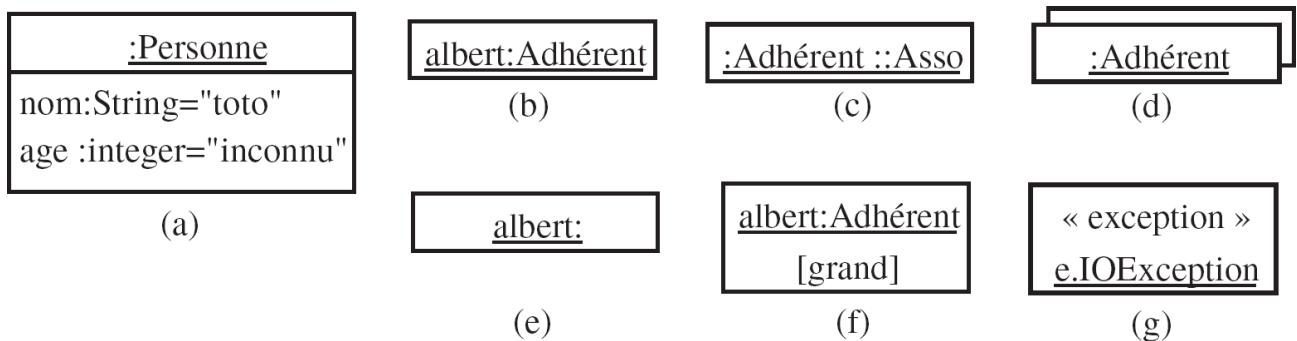
# Objectif

- Le **diagramme d'objets** représente les objets d'un système à un instant donné. Il permet de :
  - Illustrer le modèle de classes (en montrant un exemple qui explique le modèle) ;
  - Préciser certains aspects du système (en mettant en évidence des détails imperceptibles dans le diagramme de classes) ;
  - Exprimer une exception (en modélisant des cas particuliers, des connaissances non généralisables...).

Le diagramme de classes modélise des *règles* et  
le diagramme d'objets modélise des *faits*.

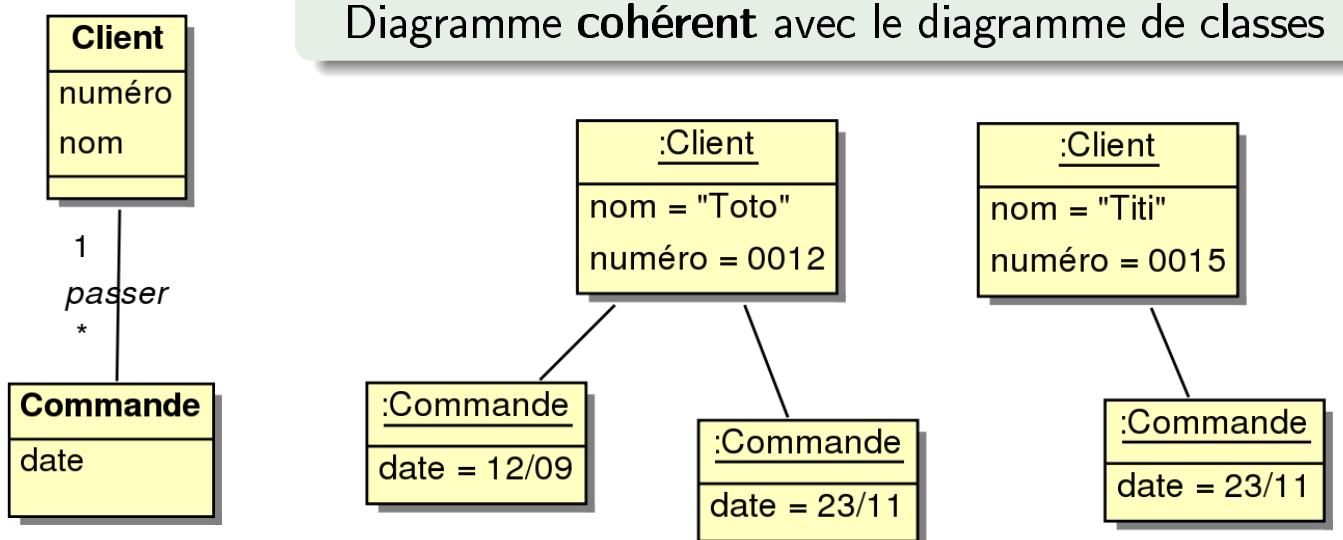
# Représentation des objets

- 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.
- Les valeurs (a) ou l'état (f) d'un objet peuvent être spécifiées.
- Les instances peuvent être **anonymes** (a,c,d), **nommées** (b,f), **orphelines** (e), **multiples** (d) ou **stéréotypées** (g).



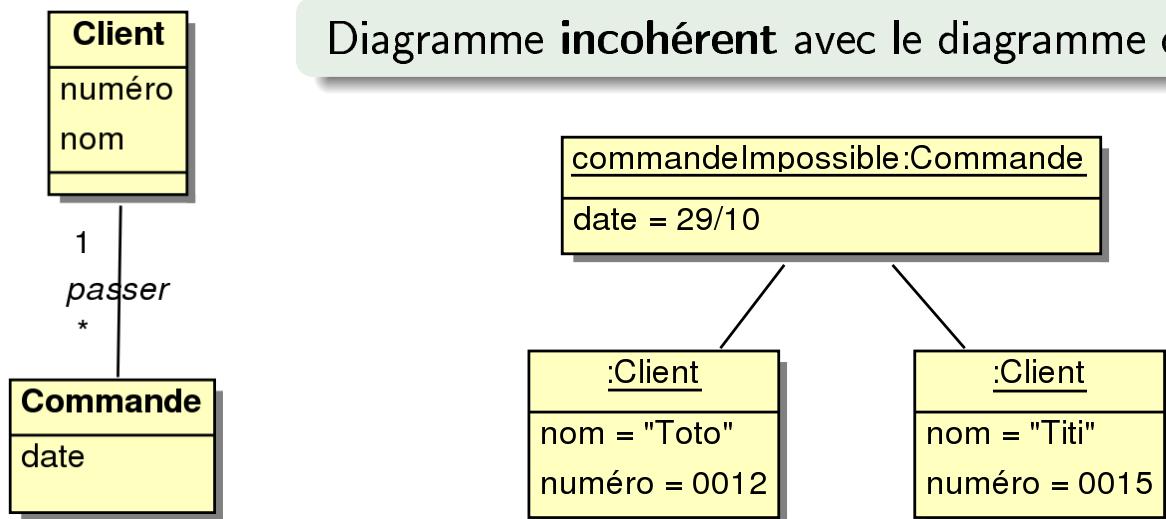
## Diagramme de classes et diagramme d'objets

- Le diagramme de classes **constraint** la structure et les liens entre les objets.



## Diagramme de classes et diagramme d'objets

- Le diagramme de classes **contraint** la structure et les liens entre les objets.



## Liens

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

### Attention

Naturellement, on ne représente pas les multiplicités qui n'ont aucun sens au niveau des objets.

## Relation de dépendance d'instanciation

- La relation de **dépendance d'instanciation** (stéréotypée) décrit la relation entre un classeur et ses instances.
- Elle relie, en particulier, les associations aux liens et les classes aux objets.

