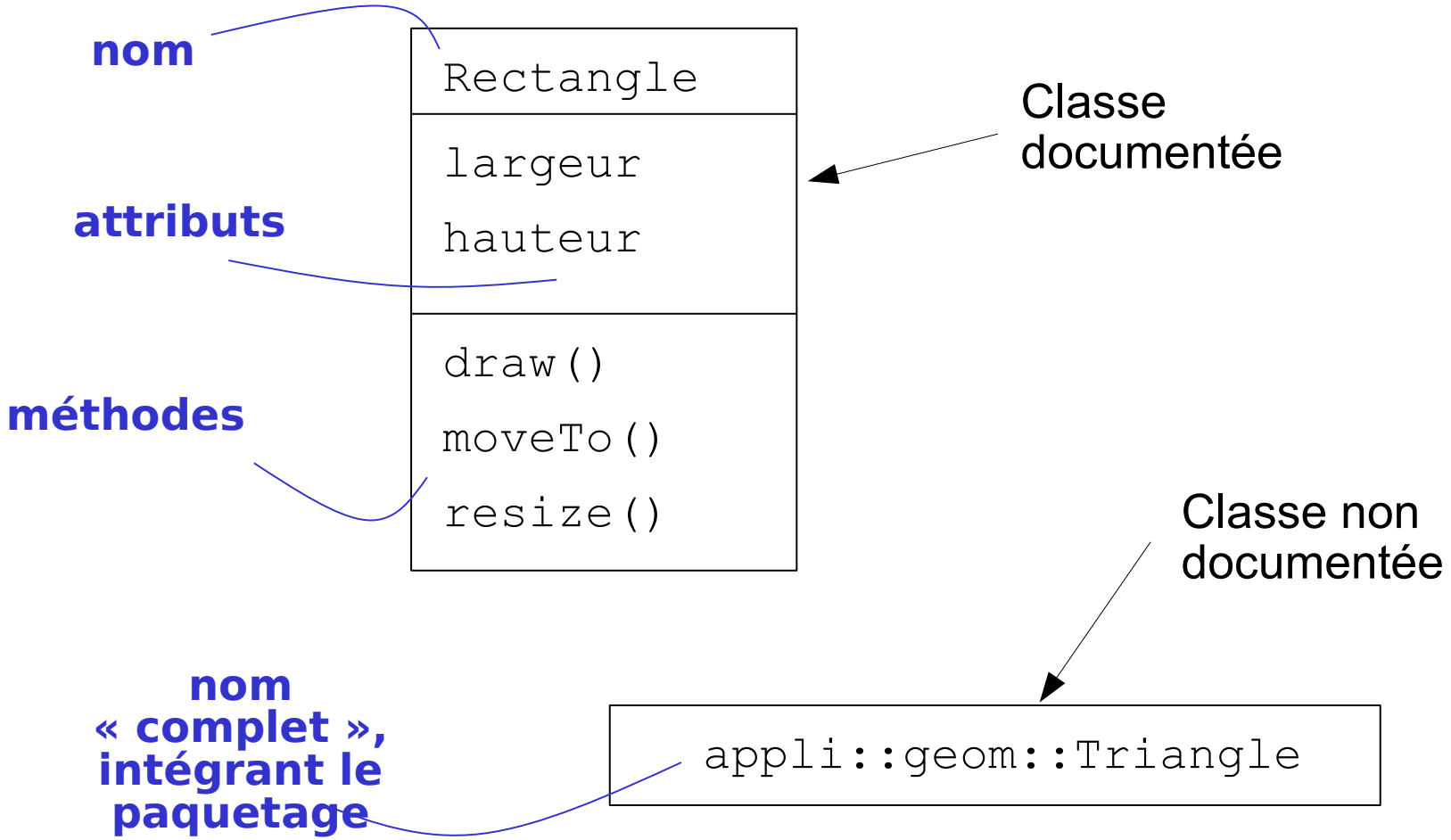
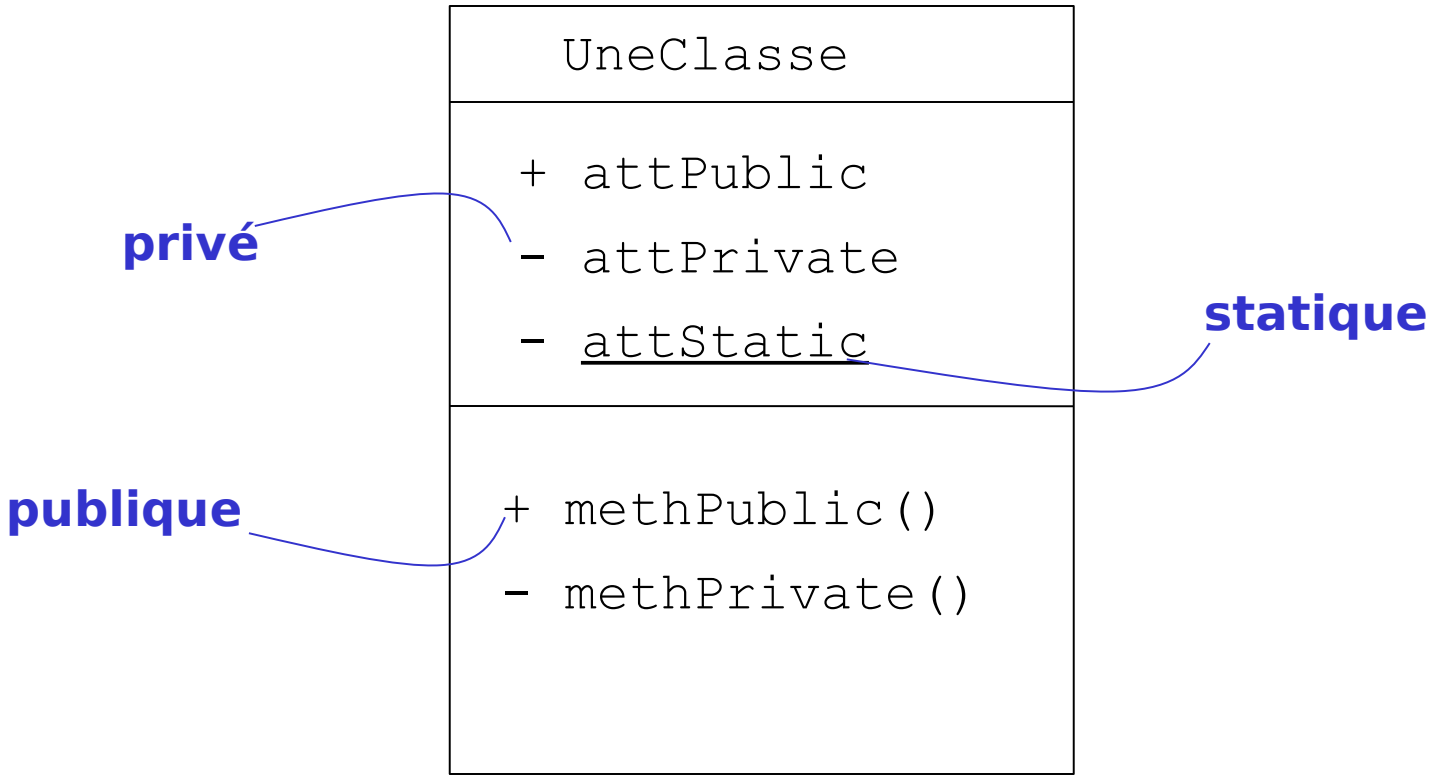
UML : relations

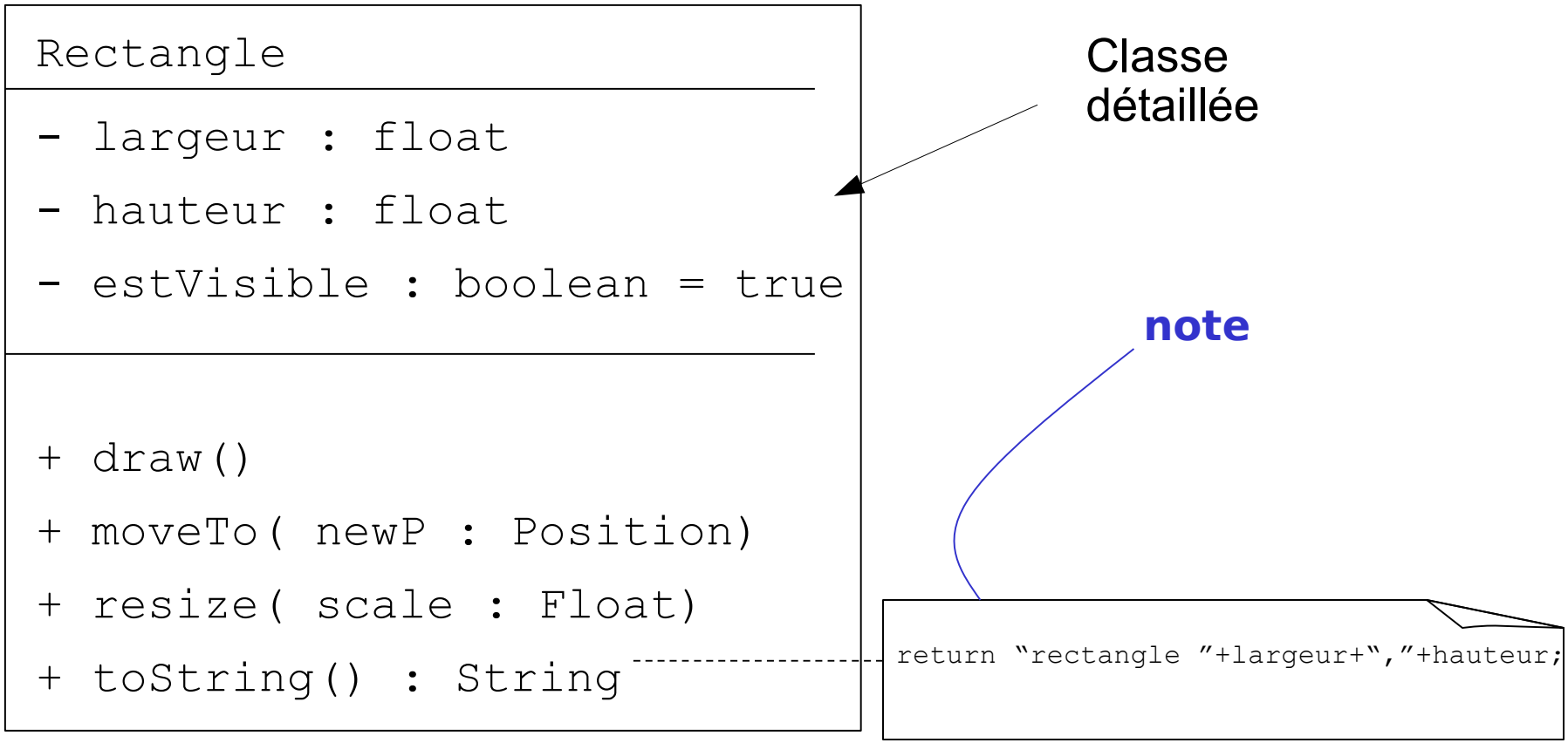
Programmation Orientée Objet

Jean-Christophe Routier
Licence mention Informatique
Université des Sciences et Technologies de Lille

Classes


nom
Rectangle
largeur
hauteur
attributs
draw()
moveTo()
resize()
méthodes
Classe documentée
Classe non documentée
nom « complet », intégrant le paquetage
appli::geom::Triangle


UneClasse
+ attPublic
- attPrivate
- attStatic
+ methPublic()
- methPrivate()
privé
statique
publique


Rectangle
- largeur : float
- hauteur : float
- estVisible : boolean = true
+ draw()
+ moveTo(newP : Position)
+ resize(scale : Float)
+ toString() : String
Classe détaillée
note
return "rectangle "+largeur+","+hauteur;

Relations

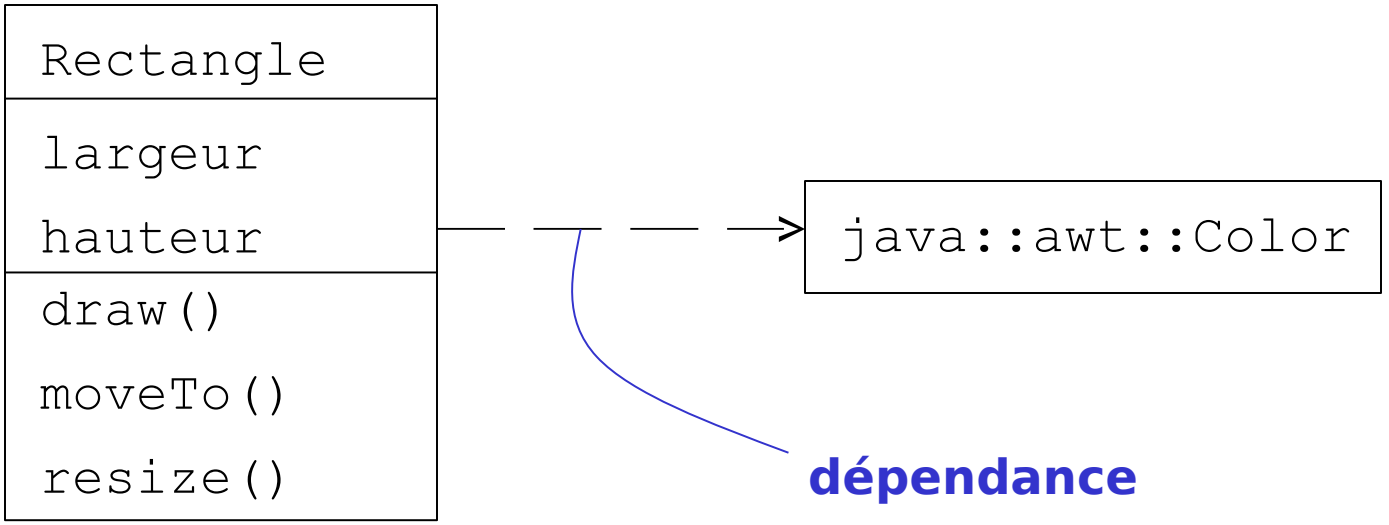
Identifier les classes ne suffit pas, elles coopèrent/ interagissent entre elles, il faut exprimer ces relations (le plus souvent binaires).

- Dépendances
  - relations d'utilisation
- Généralisations
- Associations
  - relations structurelles, connexion sémantique

Dépendance

Exprimer le fait qu'une classe en utilise une autre.

Toute modification sur la classe utilisée peut avoir un impact sur la classe utilisante.



Association

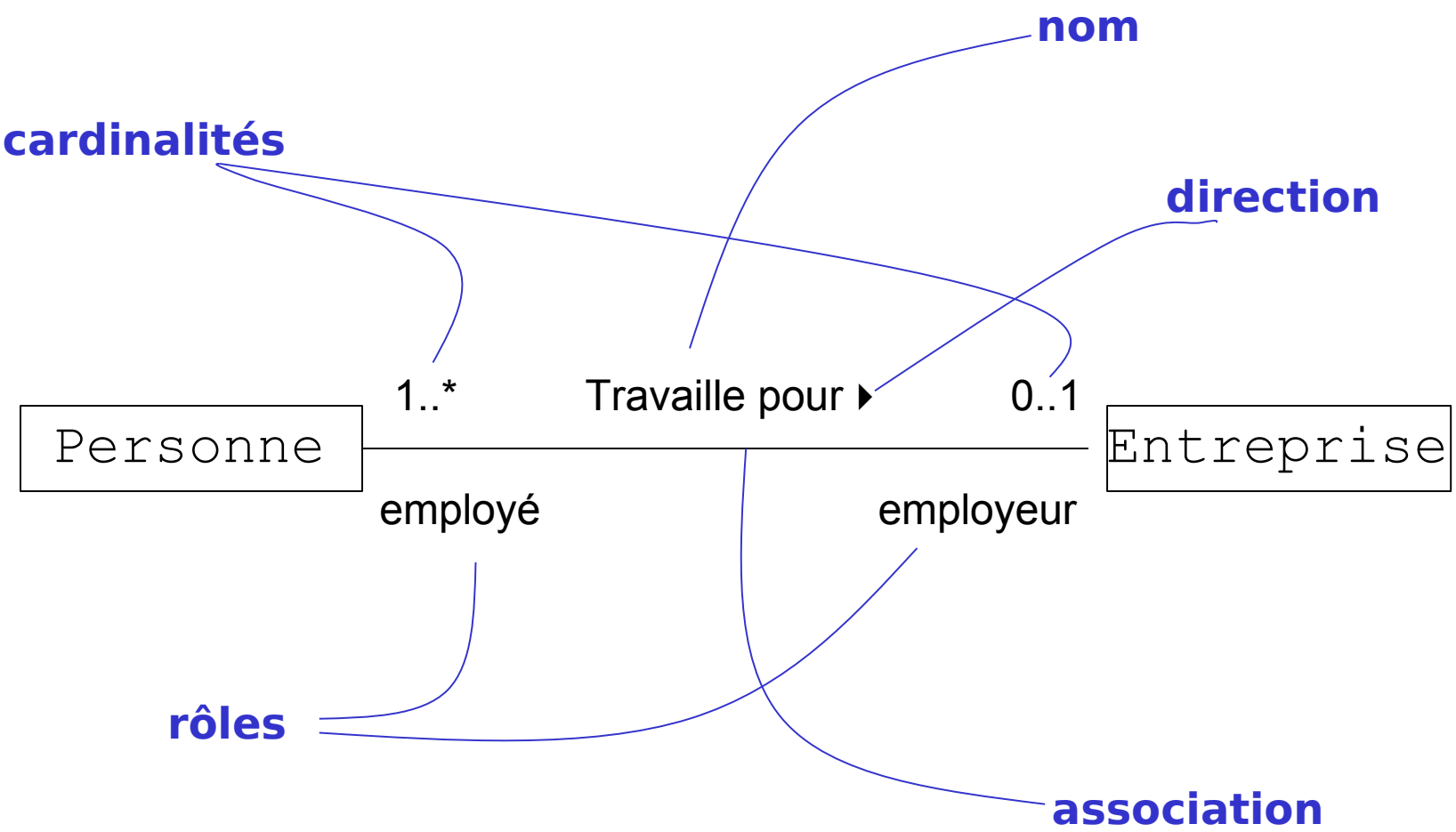
C'est une relation structurelle qui exprime une relation sémantique entre (le plus souvent) deux classes.

Elle est le plus souvent réflexive.

On peut la compléter de 4 informations :

- Nom
- Rôles
- Multiplicité
- Agrégation

Exemple


nom
cardinalités
direction
1..*
Travaille pour
0..1
Personne
Entreprise
employé
employeur
rôles
association

Cardinalités

- Définissent la multiplicité des rôles

- Une cardinalité à une extrémité signifie qu'à l'autre extrémité ce nombre d'éléments doit exister pour tout objet de la classe.

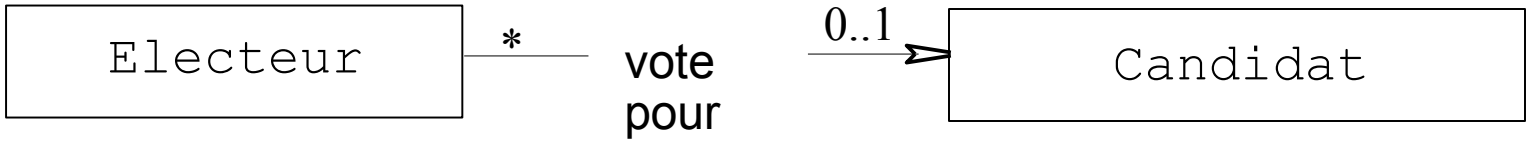
- Expressions possibles :
  - n : exactement *n*
  - n..m : de *n* à *m*
  - * : quelconque (équivalent à « 0..n » ou « 0.. »)
  - n..* : *n* ou plus
  - liste de cardinalités : 1..2,3..5 = 1 à 5 sauf 4

Il peut exister plusieurs relations entre les mêmes classes

Navigabilité restreinte

Rendre unidirectionnelle la relation

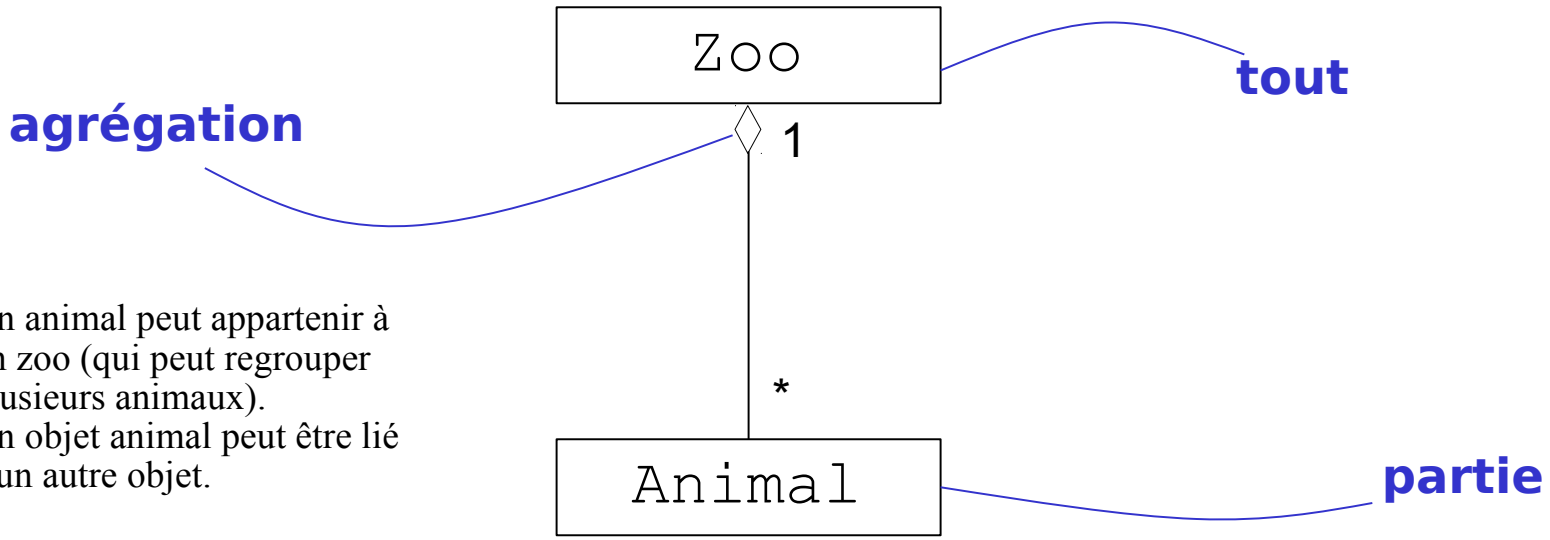
Pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre.



Agrégation/Composition

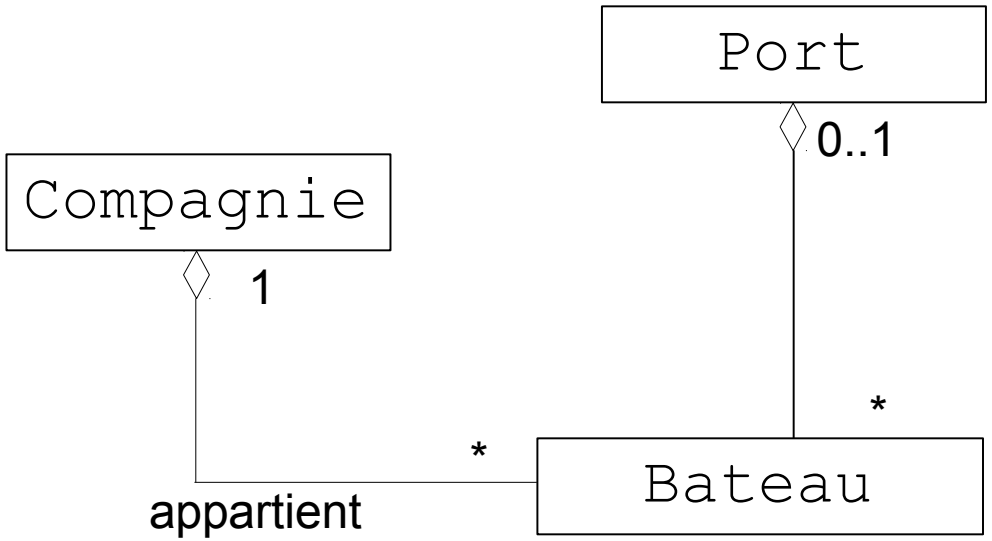
Association « tout/partie », relation de possession « has-a »

Agrégation : le tout est responsable de la gestion de ses parties.
Relation de subordination.

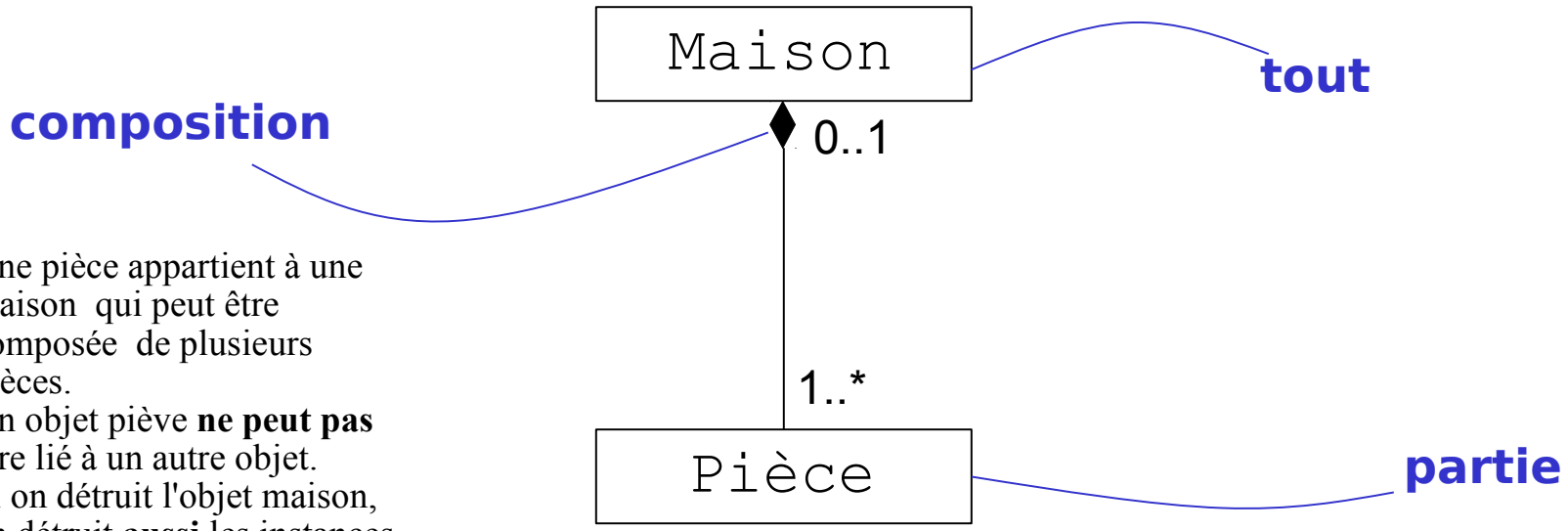


Un animal peut appartenir à un zoo (qui peut regrouper plusieurs animaux).
Un objet animal peut être lié à un autre objet.

la partie est éventuellement partagée

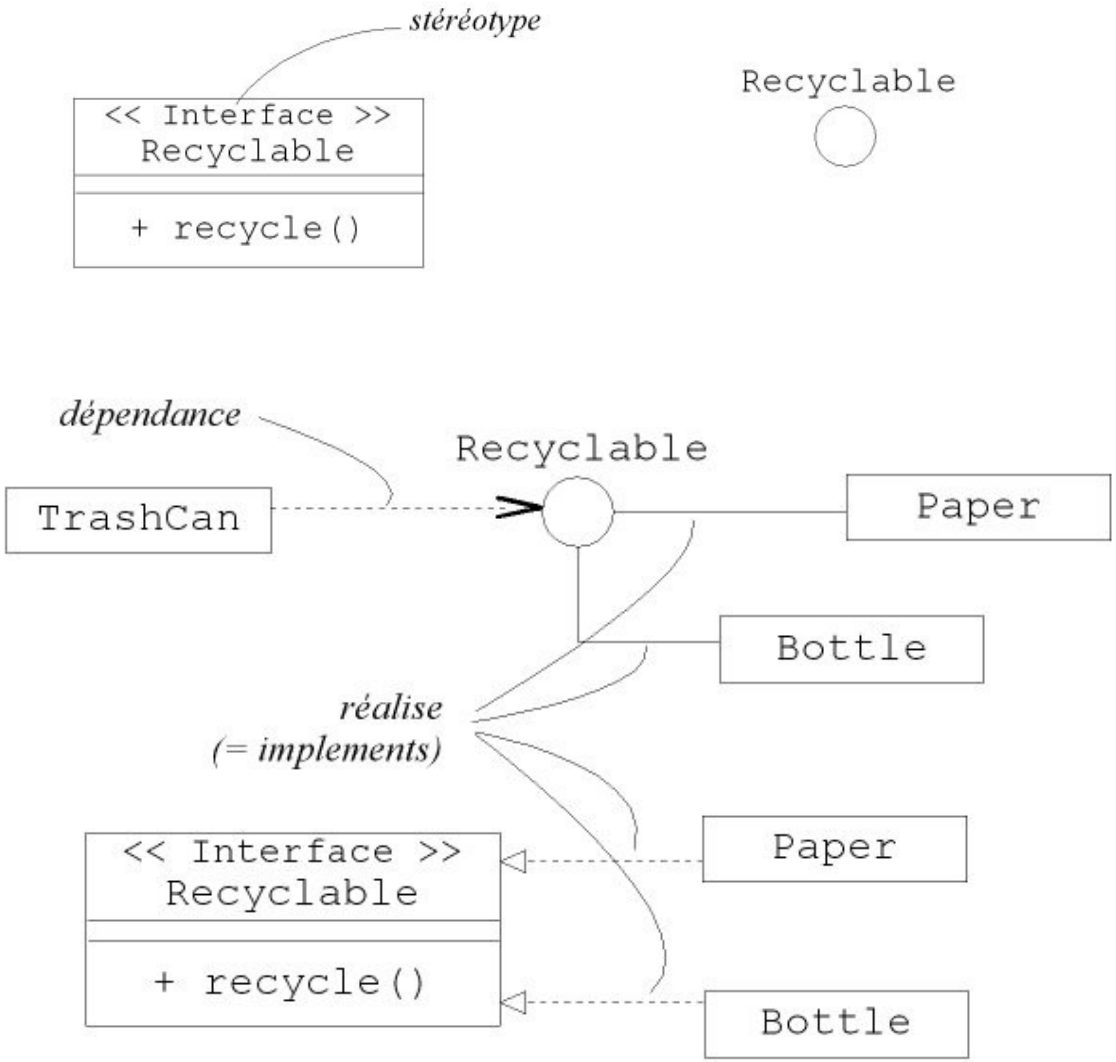


Composition : agrégation **forte,** la partie n'est pas partagée

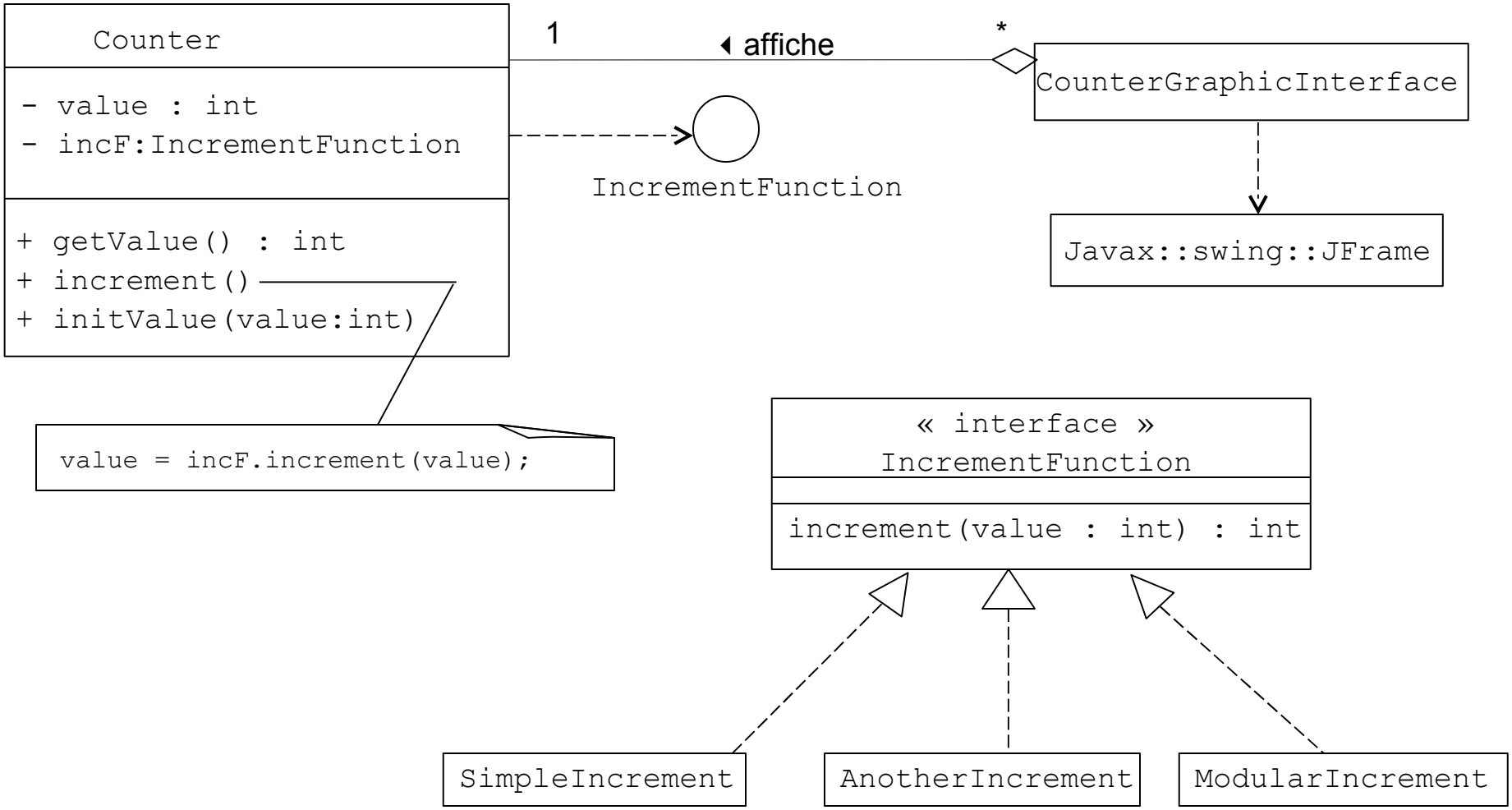


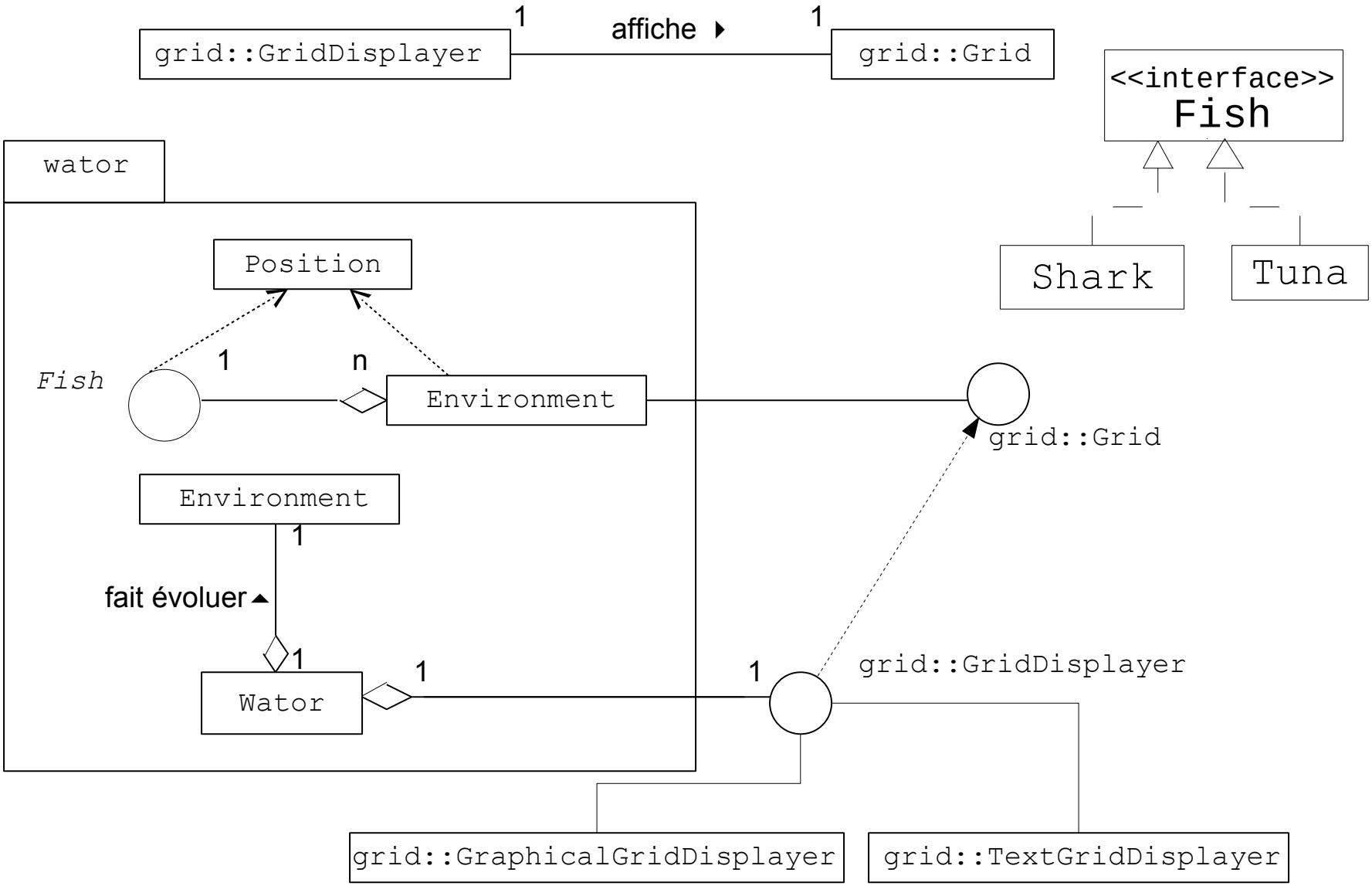
Une pièce appartient à une maison qui peut être composée de plusieurs pièces.
Un objet pièce **ne peut pas** être lié à un autre objet.
Si on détruit l'objet maison, on détruit **aussi** les instances pièces liées.

Interfaces


stéréotype
<< Interface >>
Recyclable
+ recycle()
Recyclable
dépendance
TrashCan
Recyclable
Paper
Bottle
réalise
(= implements)
<< Interface >>
Recyclable
+ recycle()
Paper
Bottle

Exemple : compteur




grid::GridDisplayer
1
affiche
1
grid::Grid
<<interface>>
Fish
Shark
Tuna
wator
Position
Fish
1
n
Environment
grid::Grid
Environment
1
fait évoluer
1
1
Wator
1
grid::GridDisplayer
grid::GraphicalGridDisplayer
grid::TextGridDisplayer