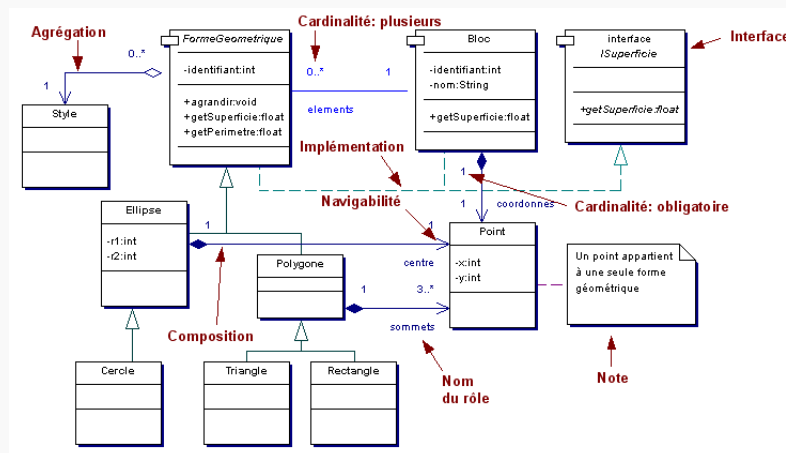


R2-01b

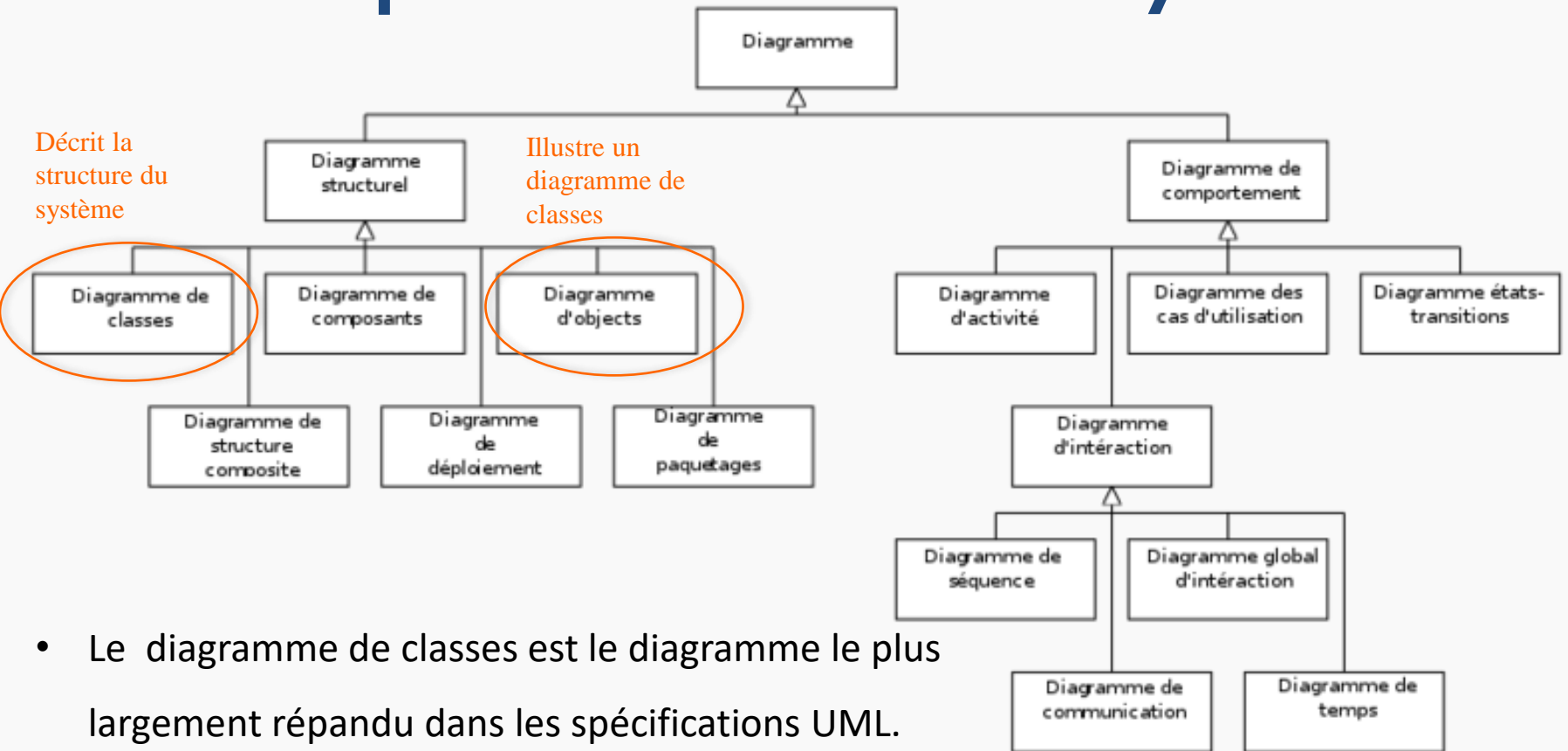
Bases de la conception orientée objet

Partie 4

UML/Diagramme de classes et d'objets



Aspects structurels du système

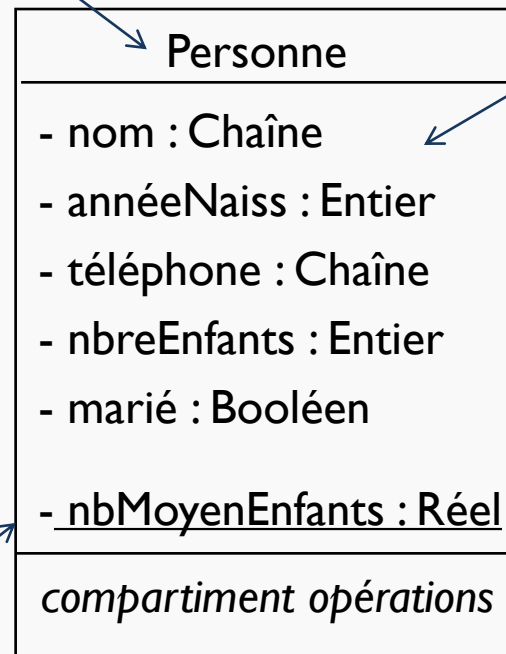


- Le diagramme de classes est le diagramme le plus largement répandu dans les spécifications UML.
- Il décrit la structure du système sous forme de classes (attributs + opérations) et de relations entre classes (associations et spécialisation).

Classe – Compartiment d'attributs

nom de la classe

(commence par une majuscule)



attribut de classe (souligné)

valeur attachée à la classe (*static* en Java)

attribut d'instance

nom (commence par une minuscule)

visibilité

+ public
protégé
- privé

type de la valeur (**type de base**)

jamais une classe

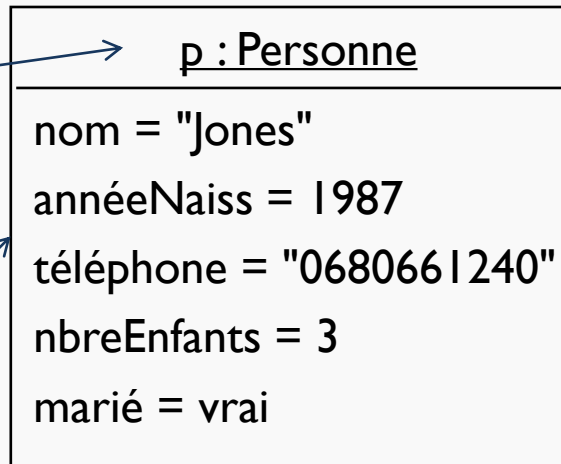
Entier
Réel
Chaîne
Car
Booléen
Date

Objet

- Un **objet** est **instance** d'une classe. Il admet une valeur pour chaque attribut de sa classe conforme au type, ces valeurs définissent l'**état** de l'objet.

**identificateur de l'objet
et nom de la classe**
(soulignés)

**nom des attributs
et valeurs**



En Java : Personne (String no, int anNais, String numTel, int nbEnf, boolean situ)
constructeur de la classe Personne

Personne p = new Personne ("Jones" , 1997, "0680661240", 3, true);

p est l'identificateur de l'objet créé.

Rappel OO : Identité d'objet

▪ L'identificateur :

- permet de faire référence à un objet.
- permet de distinguer des objets ayant mêmes valeurs d'attributs.

```
public class GroupeNatation {  
    private int niveau ;  
    private String jour ;  
    private int heureDebut ;  
    private int heureFin;  
    public void modifierHoraire (int hd, int hf) {  
        heureDebut = hd ;  
        heureFin = hf ;  
    }  
}
```

```
g1 = new GroupeNatation (1, "lundi ", 17, 18);  
g2 = new GroupeNatation (1, "lundi ", 17, 18);
```

Ces 2 groupes de natation **différents**
de même niveau ont leur séance au même moment.

Les objets g1 et g2 sont des " jumeaux "

En Java

g1 == g2 retourne faux
g1.equals(g2) retourne vrai

Chaque objet a une existence propre.
g1.modifierHoraire(19,20)

Multiplicité d'un attribut, Valeur par défaut

Multiplicité d'un attribut [Mmin .. Mmax] : nombre de valeurs qu'un attribut peut prendre au minimum et au maximum.

Personne
- nom : Chaîne - annéeNaiss : Entier - téléphone [0..2] : Chaîne - nbreEnfants : Entier - marié : Booléen = vrai - <u>nbMoyenEnfants : Réel</u>
<i>compartiment opérations</i>

[1.. 1]	par défaut , obligatoirement une seule valeur
[0 .. 1]	aucune valeur ou une seule valeur
[1 .. *]	au moins une valeur ou plusieurs valeurs
[0 .. *]	aucune valeur ou plusieurs valeurs
[n .. m]	au moins n valeurs et au plus m valeurs

La **valeur par défaut** est affectée à l'attribut à la création des instances de la classe à moins qu'une autre valeur ne soit spécifiée.

En Java : constructeurs de la classe Personne

Personne (String no, int anNais, String numTel, int nbEnf, boolean situ)

Personne (String no, int anNais, String numTel, int nbEnf)

Contraintes usuelles sur un attribut

- Contrainte d'**unicité** de la valeur d'un attribut : attribut identifiant (ou clé)

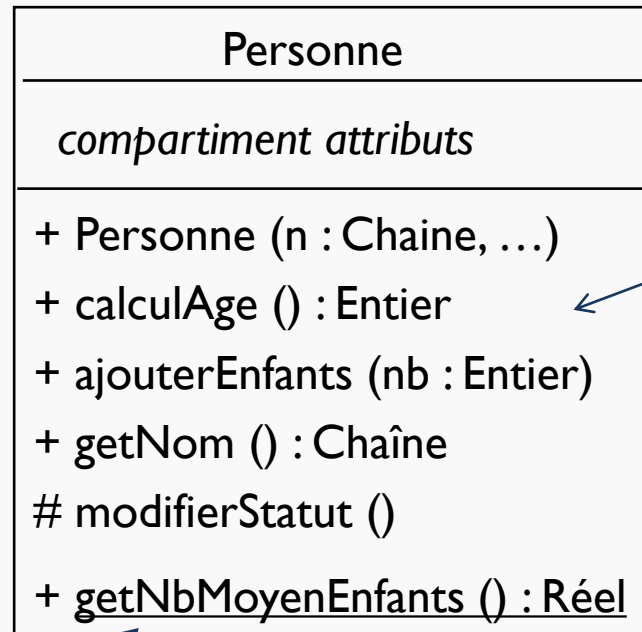
numEtudiant : Chaîne {**unique**} (unique implique gelé)

- Contrainte de **non modifiabilité** de la valeur d'un attribut

prix : Réel {**gelé**} (**frozen**, par défaut non modifiable)

final en Java

Classe – Compartiment opérations



opération de classe (souligné)

permet de manipuler les attributs de classe (*static* en Java)

Personne.getNbMoyenEnfants();

opération (méthode)

nom (*commence par une minuscule*)

visibilité

+ public
protégé
- privé

[paramètres formels typés]

[type du résultat]

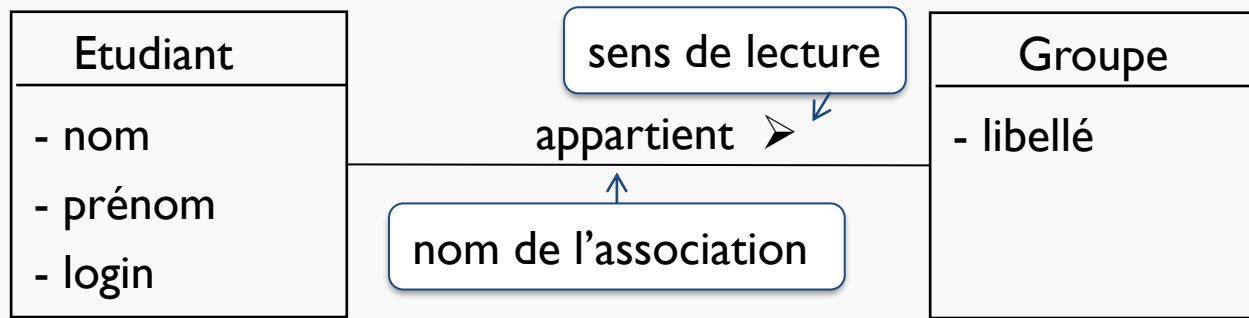
type des paramètres et résultat :
type de base ou classe

Dans le compartiment opérations →

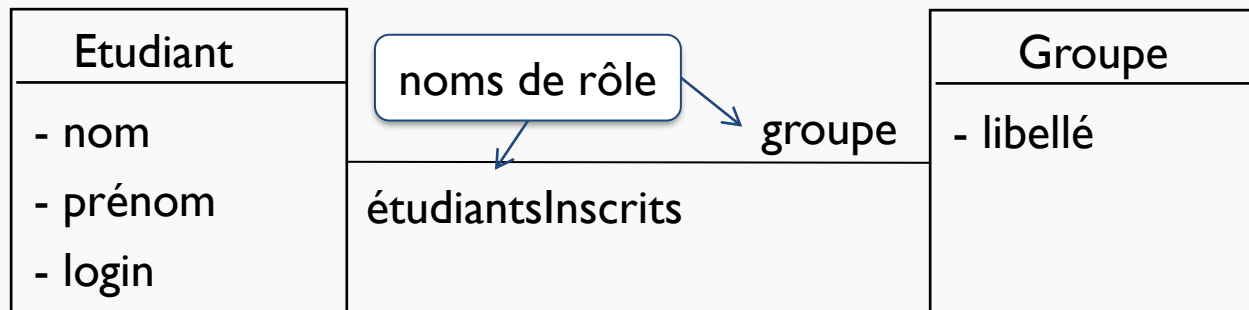
signature des opérations = nom [paramètres formels typés] [type du résultat]

Association, rôle

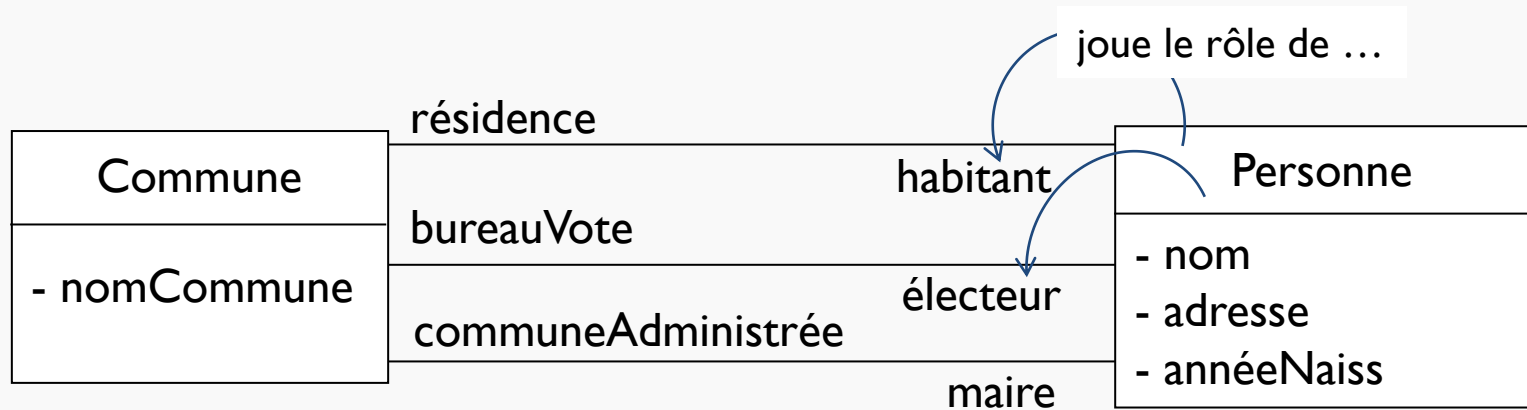
- Une **association** est une relation durable entre des classes (binaire ou n-aire).



- Une association possède 2 **rôles** inverses l'un de l'autre.
- Chaque classe joue un rôle différent dans l'association. Un rôle indique comment une classe source voit sa classe destination.
- Le nom du rôle est noté du **côté de la classe jouant ce rôle** dans l'association.



Importance du nommage des rôles



Conseils :

Choisir soigneusement le nom de chacun des rôles d'une association.

Préférer les noms de rôles au nom de l'association.

Association / Attribut ?

Employé
- nomEmployé - grade - salaire

<u>e1 :Employé</u>
nomEmployé = "Dupont" grade = cadreA salaire = 40250 40600

<u>e2 :Employé</u>
nomEmployé = "Durand" grade = cadreA salaire = 40250

On augmente le salaire de l'employé e1 mais pas celui de e2.

Employé	employés	grade	Grade
- nomEmployé	*	1	- nomGrade - salaire

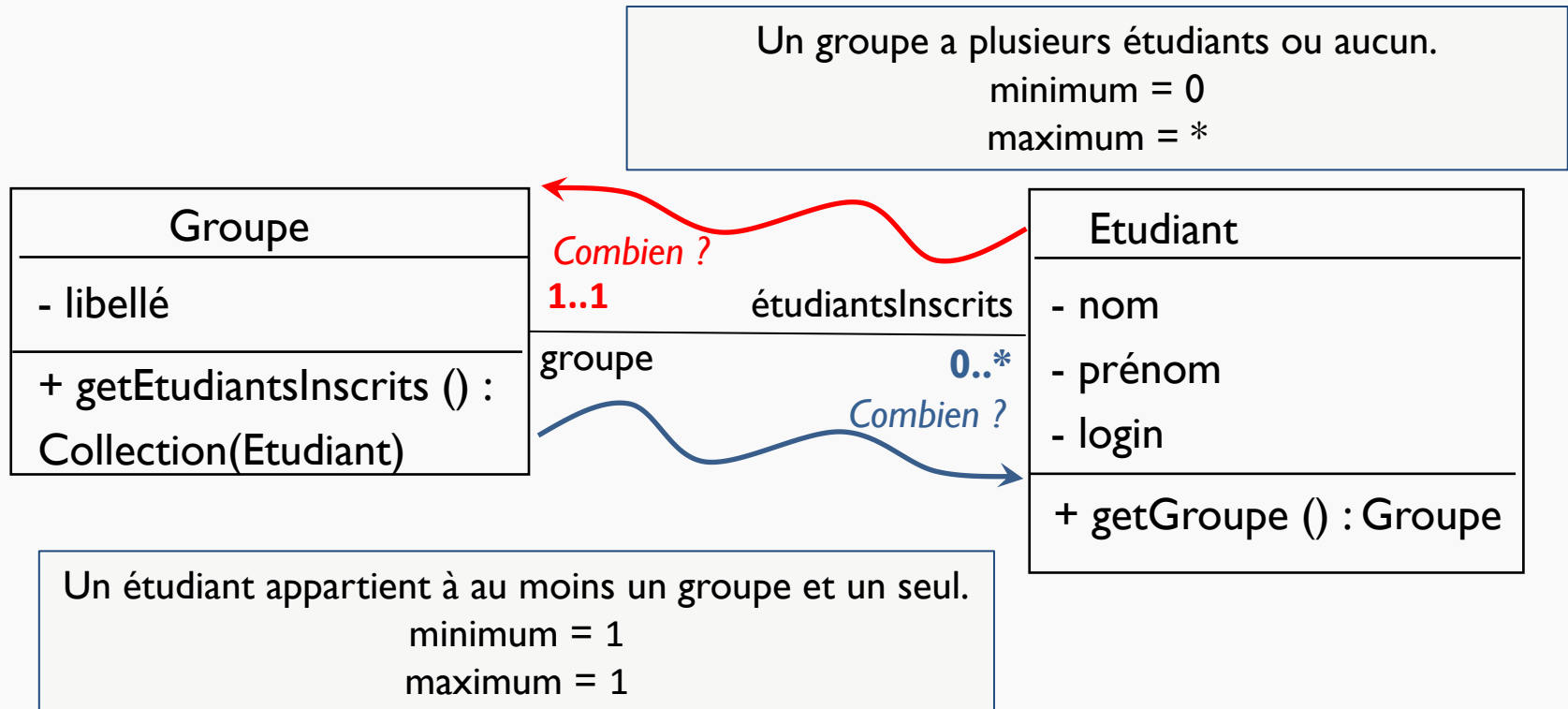
On augmente le salaire associé au grade g1.

Les employés e1 et e2, qui partagent le même grade, ont la même augmentation de salaire.

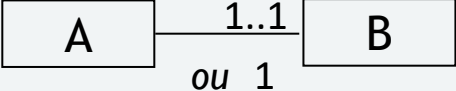
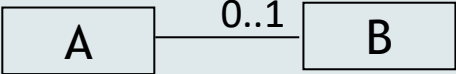
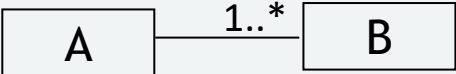
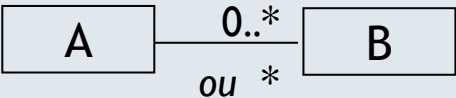

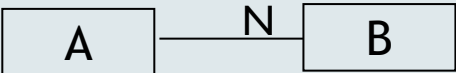
<u>e1 :Employé</u>	employés	grade	<u>g1 : Grade</u>
nomEmployé = "Dupont"			nomGrade = cadreA salaire = 40250 40600
<u>e2 :Employé</u>	employés	grade	
nomEmployé = "Durand"			

Multiplicité d'un rôle d'association

- La **multiplicité** d'un rôle précise combien d'objets de la classe destination peuvent être liés à un objet de la classe source, au **minimum** et au **maximum**. Elle est portée par le rôle.
- La multiplicité exprime une contrainte.



Notation de la multiplicité

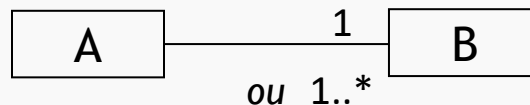
	<p>un et un seul objet de la classe B est lié à un objet de la classe A</p>
	<p>aucun objet ou un seul objet de la classe B peut être lié à un objet de la classe A</p>
	<p>un seul objet ou plusieurs objets de la classe B peuvent être liés à un objet de la classe A</p>
	<p>aucun objet ou plusieurs objets de la classe B peuvent être liés à un objet de la classe A</p>
	<p>de M à N objets de la classe B peuvent être liés à un objet de la classe A</p>
	<p>exactement N objets de la classe B sont liés à un objet de la classe A</p>

Vocabulaire

multiplicités min .. max

- min = 1 le rôle est **total**.
Il existe **obligatoirement** un objet destination lié.
- min = 0 le rôle est **partiel**.
Il n'y pas toujours un objet destination lié.
- max = 1 le rôle est **monovalué**.
- max > 1 le rôle est **multivalué**.

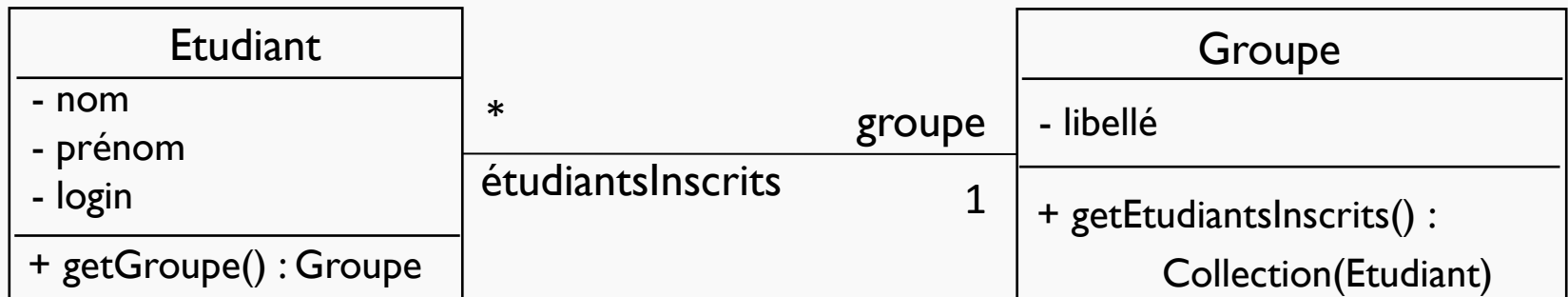
- **Cas d'un rôle total**



Lors de la création d'un objet instance de la classe A, on doit obligatoirement le lier à un objet de la classe B.

Navigation le long d'une association

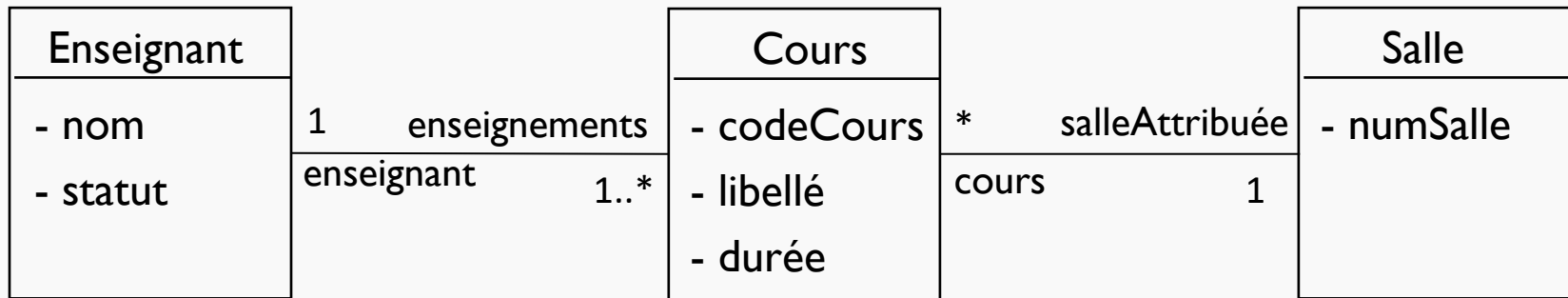
- Un **rôle d'association** est un moyen d'accès :
Un objet de la classe cible peut être atteint depuis un objet de la classe source en **navigant** le long de l'association.



- Depuis un objet de la classe Groupe, ses étudiants peuvent être atteints en navigant le long de l'association (via le rôle étudiantsInscrits). Un objet de Groupe peut invoquer des opérations (publiques) détenues par ses étudiants.
- Depuis un objet de la classe Etudiant, son groupe peut être atteint en navigant le long de l'association (via le rôle groupe).
- Par défaut, une association est navigable dans les 2 sens.

Navigation dans un diagramme

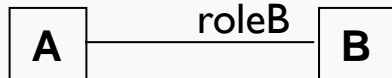
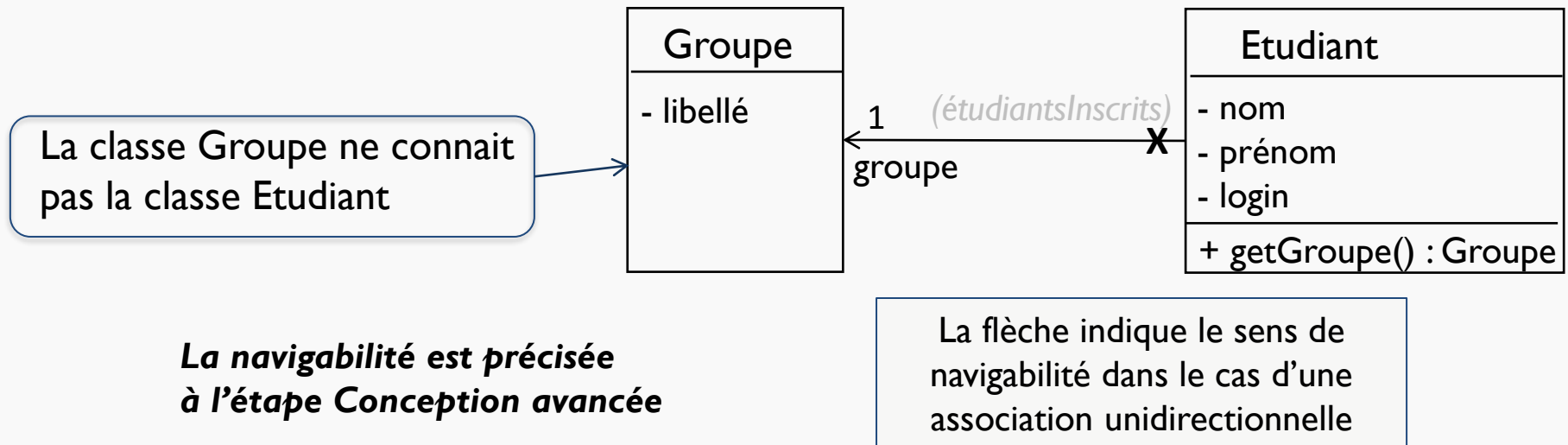
- La **navigation** dans un diagramme de classes permet l'accès de manière **transitive** à tous les objets liés à l'objet de la classe source.



- Depuis un objet de la classe Enseignant, ses cours peuvent être atteints ainsi que les salles où ils ont lieu en navigant successivement le long des 2 associations.

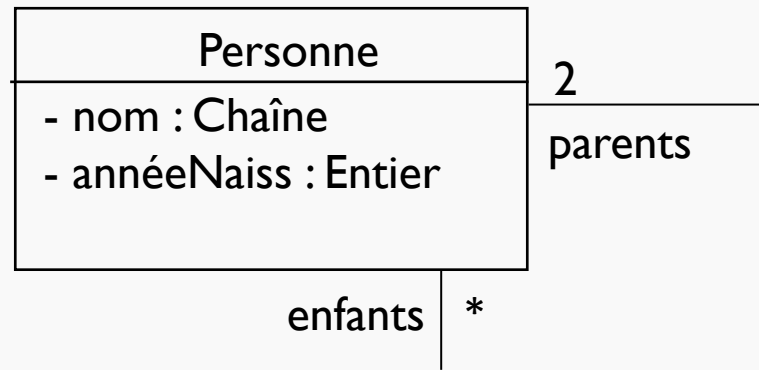
Navigabilité d'un rôle

- **Par défaut**, une association est **bidirectionnelle**. Elle est navigable dans les 2 sens.
- **Navigabilité d'un rôle d'association** : exprime l'obligation pour tout objet source d'identifier le ou les objets cibles.
- Une association **unidirectionnelle** n'est navigable que dans un seul sens.



Un objet de la classe A ne peut envoyer un message à un objet de la classe B **que si roleB est navigable** (sauf si un autre chemin de navigation permet de le connaître).

Association réflexive



- Dans cette association, une personne joue de rôle de parent et l'autre le rôle de d'enfant.

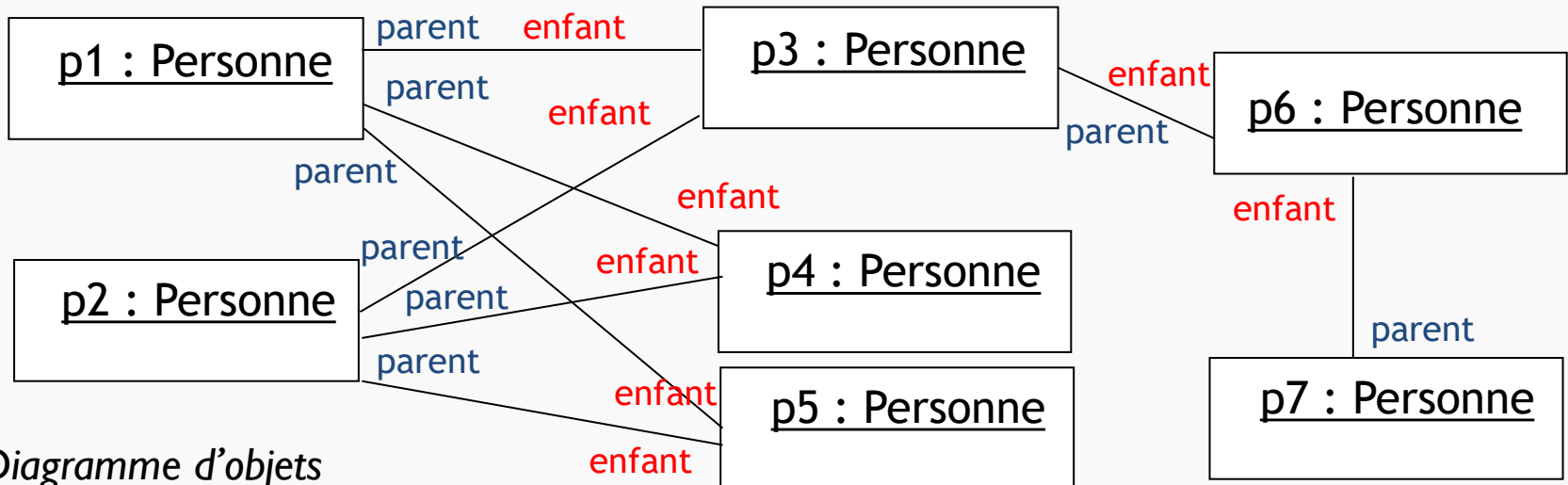


Diagramme d'objets

Implémentation des rôles en Java



```
public class Groupe {
    private String libelle ;
    private ArrayList<Etudiant> etudiants ;
}
```

```
public class Etudiant {
    private String nom ;
    private String prenom ;
    private Groupe groupe ;
}
```

- Un **rôle navigable** d'association est traduit par un **attribut** en Java de **même nom**.
- La classe de définition de l'attribut est :
 - rôle monovalué → la classe destination du rôle
 - rôle multivalué → une classe Collection paramétrée par la classe destination du rôle

Implémentation des rôles en Java



```
public class Groupe {  
    private String libelle ;  
}
```

```
public class Etudiant {  
    private String nom ;  
    private String prenom ;  
    private Groupe groupe ;  
}
```

- Seul un **rôle navigable** d'association est traduit par un **attribut** en Java.
- Un objet de Groupe ne connaît pas les étudiants qui en font partie.

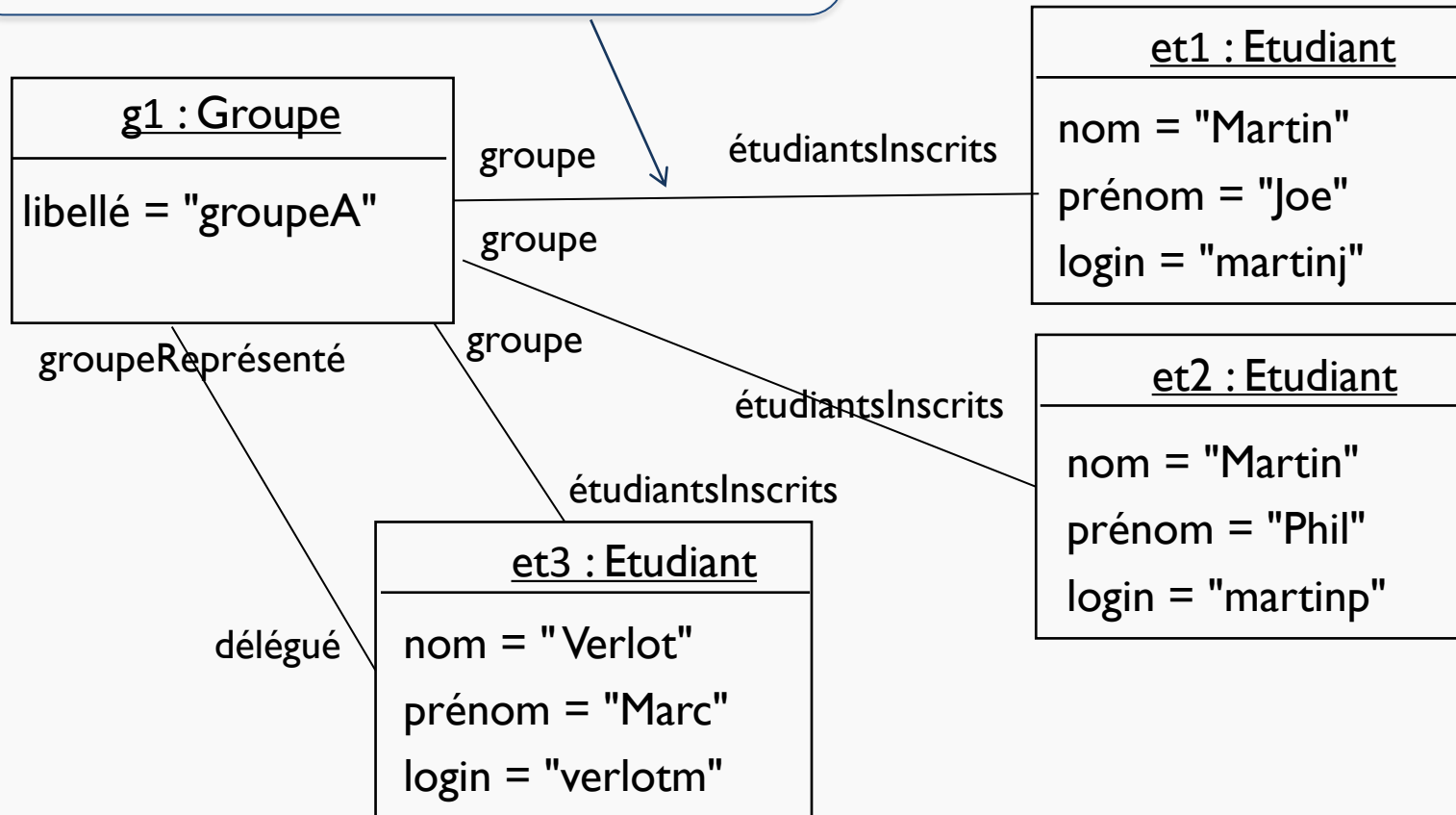
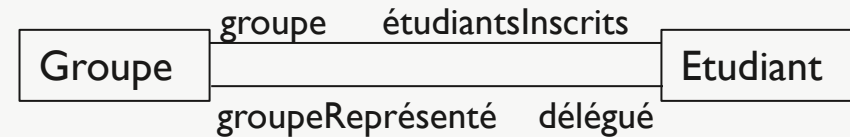
Diagramme d'objets

- Un **diagramme d'objets** représente des objets et leurs liens pour donner une vue figée de l'état du système à un instant donné.
- Il peut être utilisé pour :
 - illustrer le diagramme de classes,
 - préciser certains aspects du système,
 - montrer un cas particulier
 - invalider (ou montrer une incohérence) dans un diagramme de classes

Diagramme d'objets

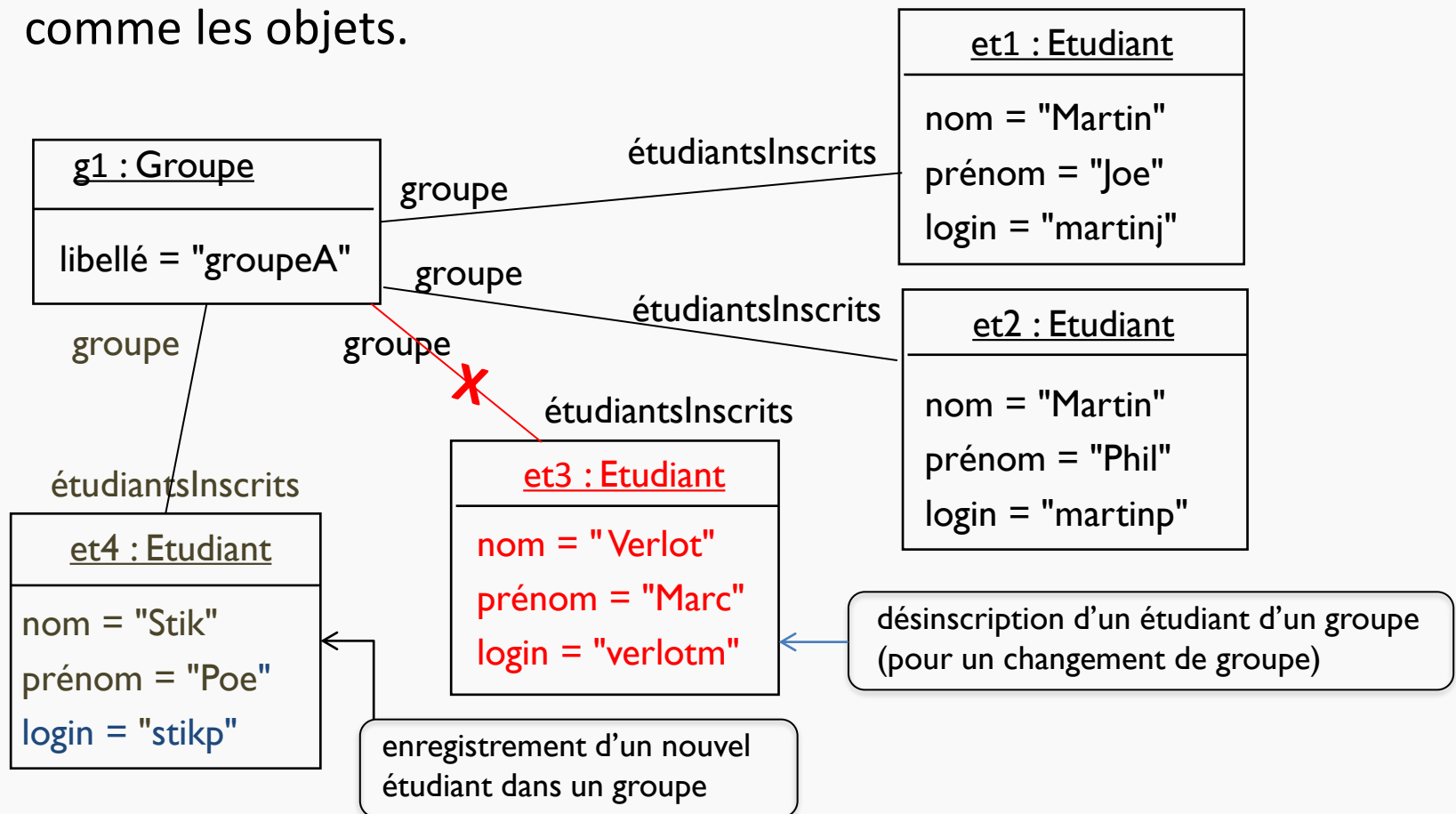
lien

connexion entre objets
chaque objet joue un rôle différent dans le lien



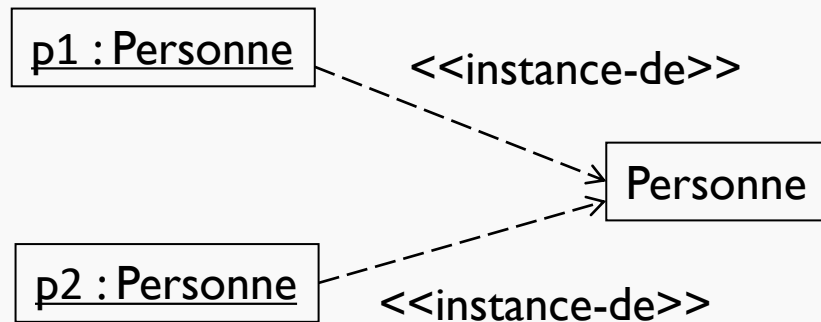
Lien entre objets

- Un **lien** lie deux **objets**.
- Une **association** lie deux **classes**
- Des **liens** peuvent être ajoutés ou détruits pendant l'exécution, tout comme les objets.

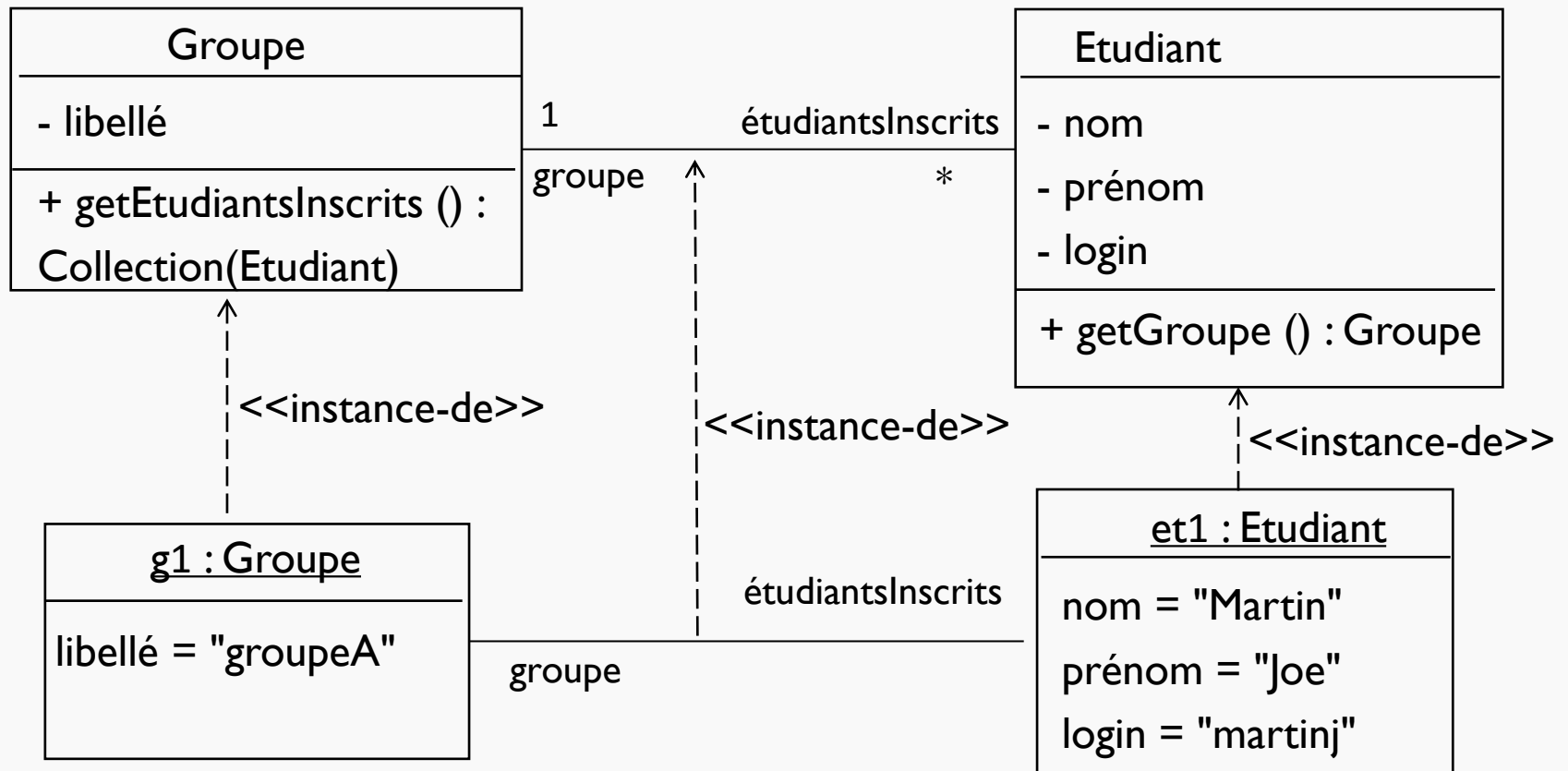


Relation d'instanciation

- Un objet est relié à sa classe par la **relation d'instanciation**.
- Cette relation permet de connaître les opérations applicables à l'objet détenues par la classe et la valeur des attributs de classe.

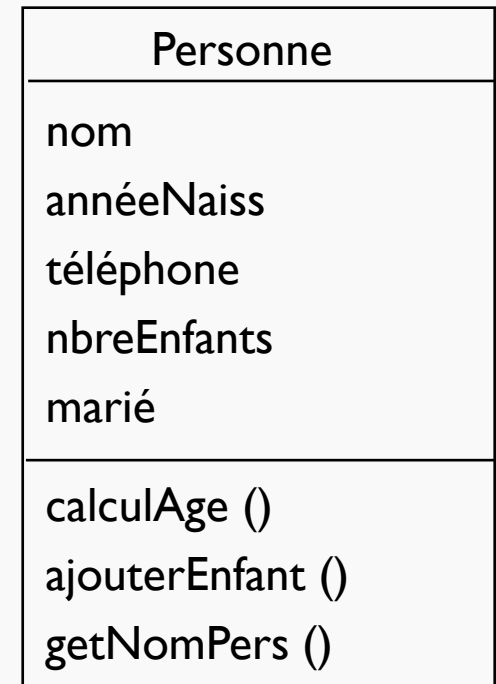
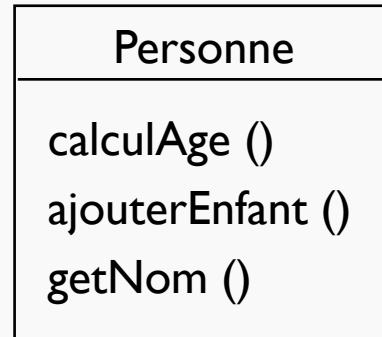
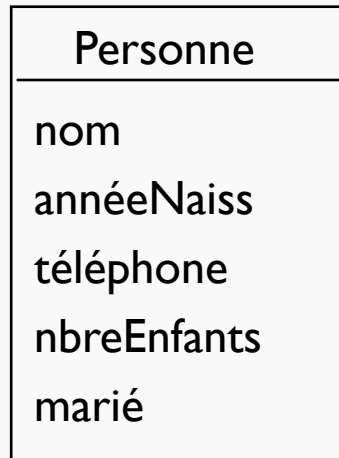


Relation d'instanciation



Notations simplifiées

■ Classe



■ Objet



Objet anonyme

- Le niveau de détail de la notation est adapté au niveau d'abstraction correspondant à une étape dans le processus de développement du logiciel.

Vocabulaire

- **objet / classe**

Un **objet** est un élément (exemplaire, occurrence) d'une et une seule classe.

On dit qu'un objet est **instance** d'une classe.

Une **classe** décrit la structure d'un ensemble d'objets de même nature (même structure et même comportement).

- **lien / association**

Un **lien** est instance d'une association.

Une **association** décrit un ensemble de liens de même sémantique.

- **diagramme d'objets / diagramme de classes**

Un **diagramme d'objets** décrit un état possible à un instant t , un cas particulier.

Il doit être conforme au diagramme de classes.

Un **diagramme de classes** définit l'ensemble de tous les états possibles.

Les contraintes doivent toujours être vérifiées.

Les diagrammes d'objets peuvent être utilisés pour illustrer un diagramme de classes ou le valider.

Contraintes

Un diagramme de classe ne permet pas d'exprimer tout ce qui est défini dans la spécification textuelle.

=> Nécessité d'ajouter des contraintes.

Contrainte

- exprime des conditions ou des restrictions sur le diagramme de classe.
- porte sur un ou plusieurs éléments (classe, association) du modèle et doit être **vraie** à tout instant.
- invariant qui doit être respecté par tous les objets auxquels il est associé

Comment exprimer des contraintes ?

- en langage formel : par exemple OCL (Object Constraint Language)
- en langue naturelle, mais la compréhension peut-être ambiguë et ne permet pas d'exprimer clairement des contraintes complexes.

Contraintes sur les attributs

- Contrainte associée au **type** d'un attribut : restriction sur le type

- Contrainte sur une chaîne :

numSécu : Chaîne {**taille**(numSécu) = 13}

- Contrainte sur un numérique :

annéeNaiss : Entier { $0 < \text{annéeNaiss}$ }

- Contrainte d'**unicité** de la valeur d'un attribut : attribut identifiant (ou clé)

numEtudiant : Chaîne {**unique**} (unique implique gelé)

- Contrainte de **non modifiabilité** de la valeur d'un attribut

prix : Réel {**gelé**} (par défaut non modifiable)

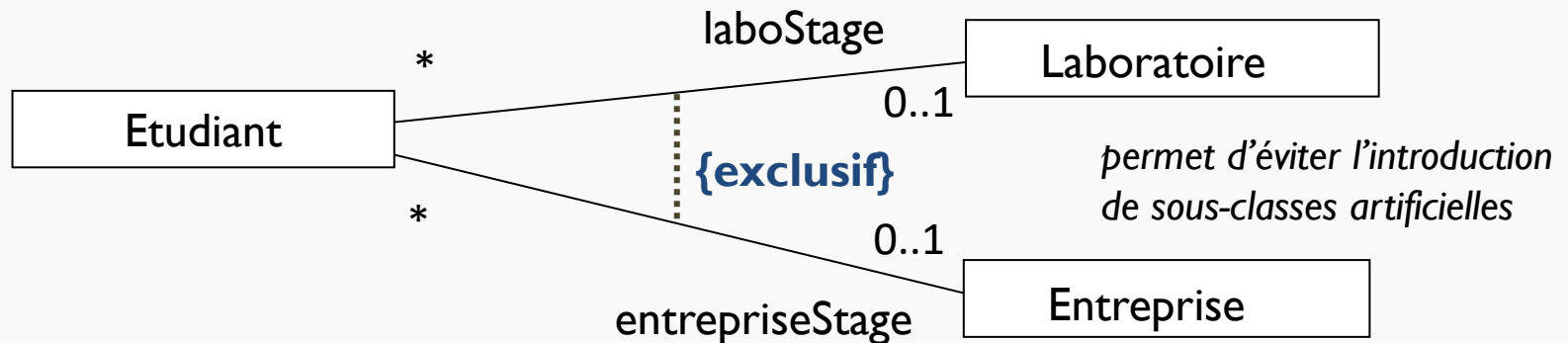
En Java, *final*

Contraintes sur les associations

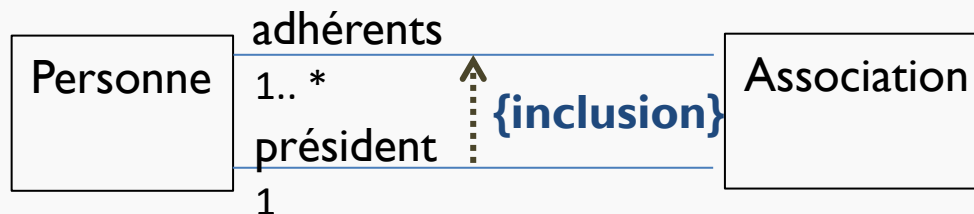
L'ensemble des villes desservies par une ligne ferroviaire est ordonné.



Un étudiant doit faire un stage soit en laboratoire soit en entreprise.



Le président d'une association est un adhérent de l'association.



Traduction en Java d'un rôle ordonné

```
public class LigneFerrovière {  
    private String nomLigne ;  
    private ArrayList<Ville> villesDesservies ;  
    public Ville villeDepart() {  
        return this.getVillesDesservies().get(0);  
    }  
    public Ville villeArrive() {  
        return this.getVillesDesservies().get(this.getVillesDesservies ().size() - 1);  
    }  
    private ArrayList<Ville> getVillesDesservies () {  
        return villesDesservies;  
    }  
}
```

```
public Ville {  
    private String nomVille ;  
    private ...;  
}
```

Diagramme de classes

Suite

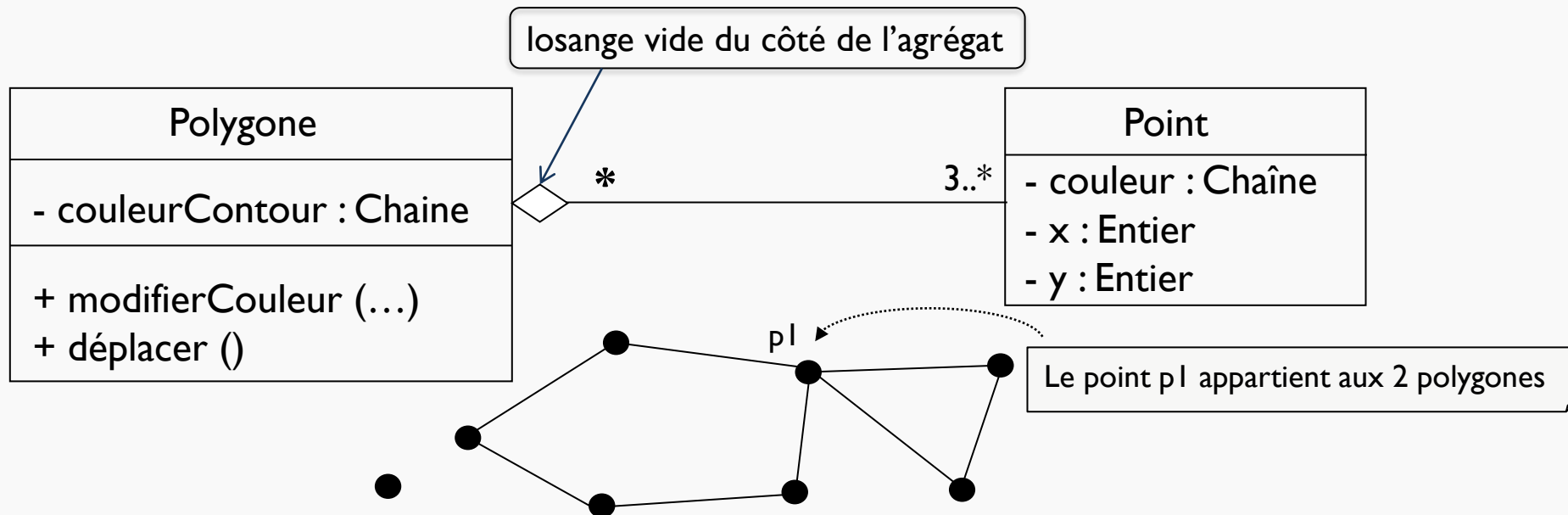
Agrégation et composition

Classe Association

Spécialisation, héritage, polymorphisme

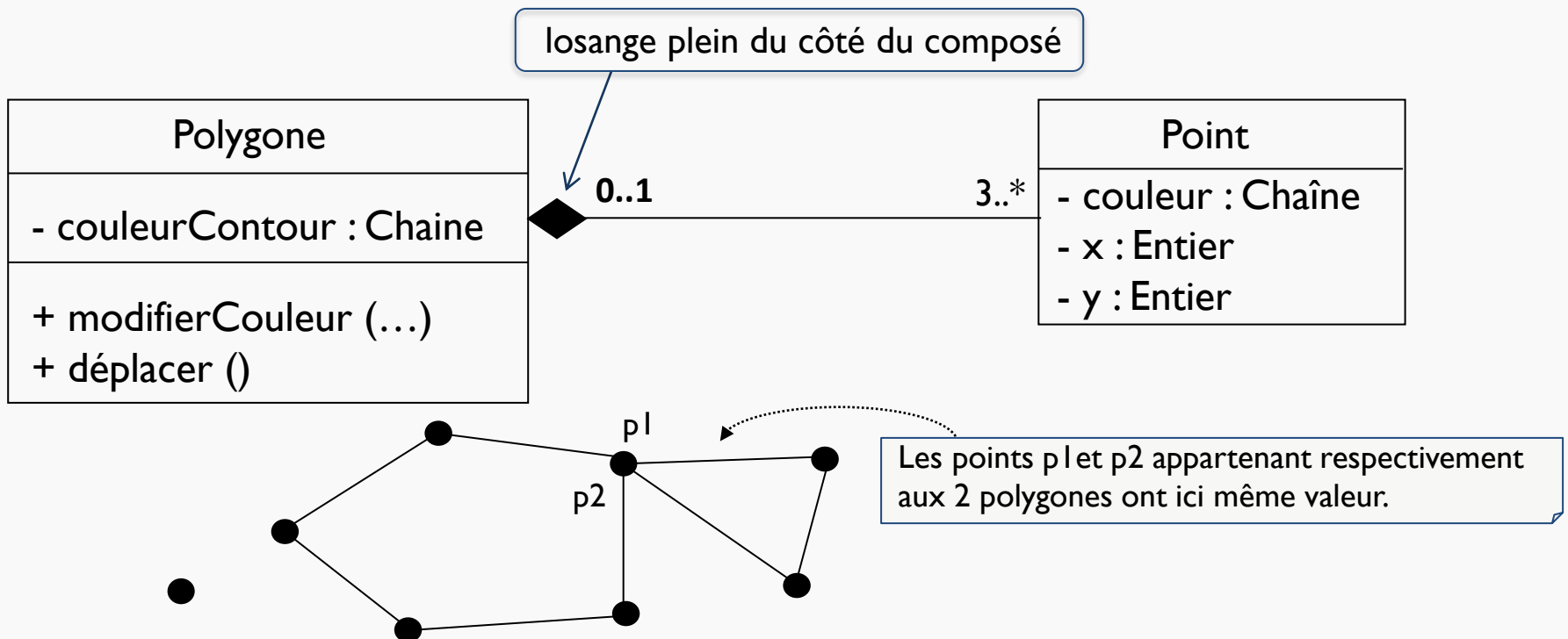
Association particulière : agrégation

- Forme particulière d'association non symétrique
- Une classe constitue un "tout" (l'agrégat) composé de "parties".
- Un objet "partie" peut être partagé (pas d'exclusivité)
- Reconnaître une agrégation :
 - des valeurs d'attributs sont-elles propagées de l'agrégat vers les parties ?
 - des opérations appliquées à l'agrégat sont-elles automatiquement appliquées aux parties ?
- S'implémente comme une association.



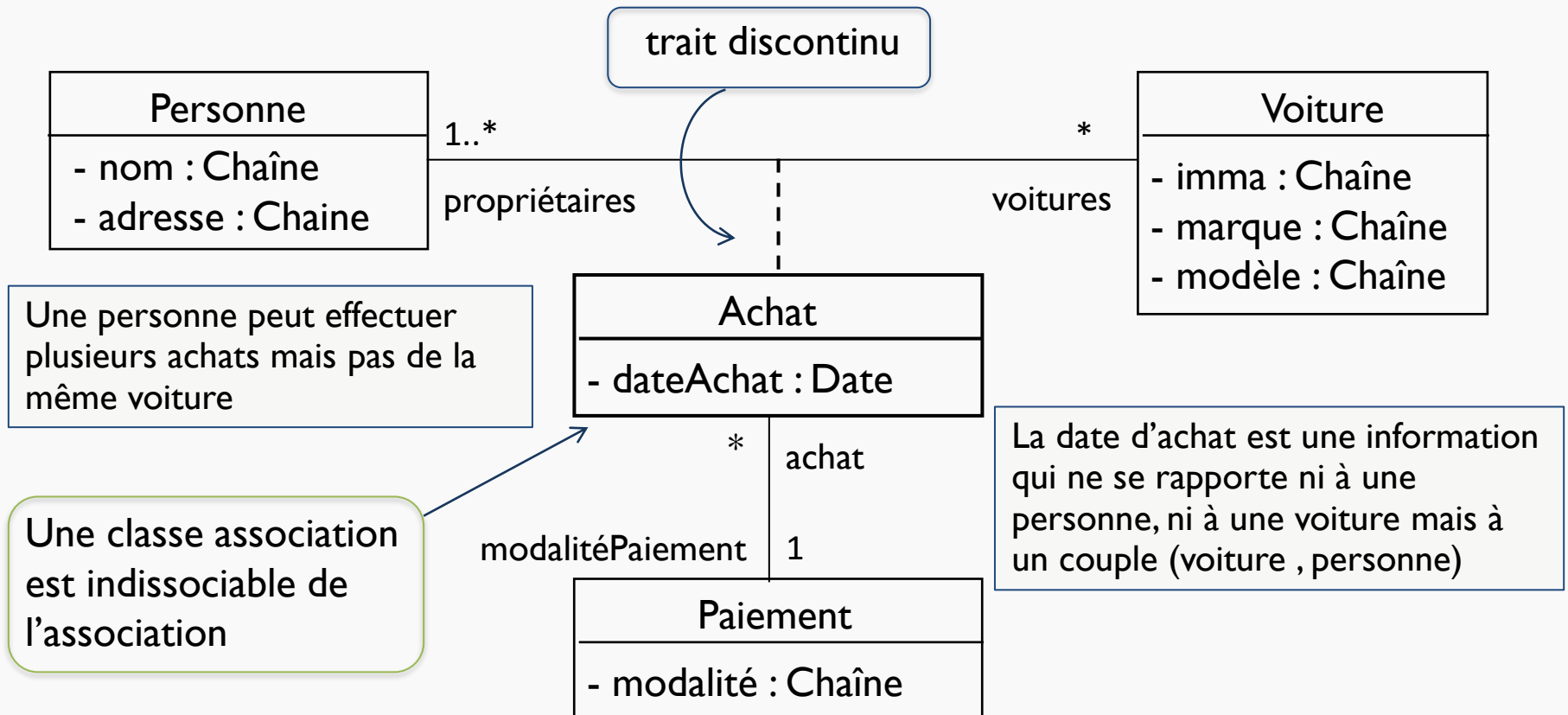
Association particulière : composition

- Forme particulière d'agrégation avec une dépendance forte.
- Un objet composant n'appartient qu'à un seul composé (pas de partage).
 - le composant est détruit lors de la destruction du composé.
 - le composant n'intervient pas dans d'autres associations de composition ou d'agrégation.
- S'implémente comme une association.



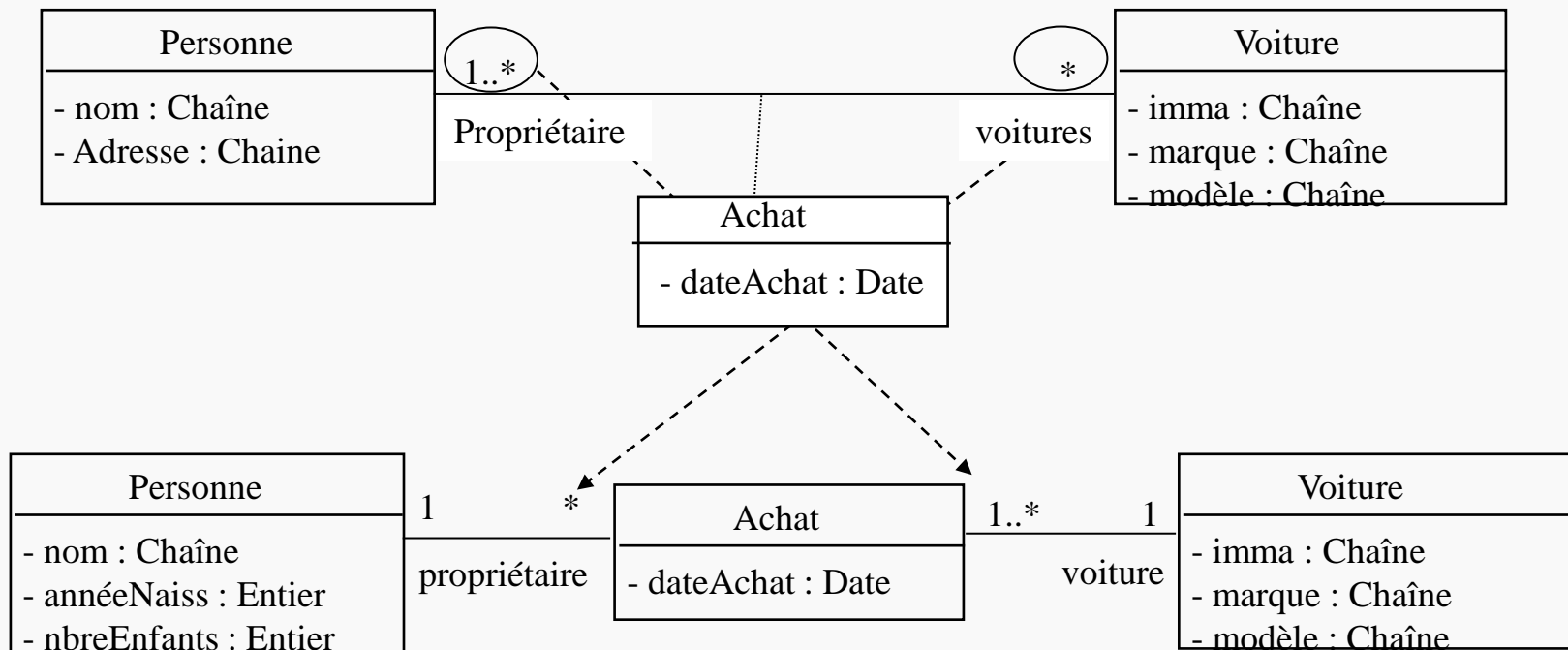
Classe association

- Une association peut posséder des attributs et des opérations.
- Elle est alors qualifiée à l'aide d'une **classe-association**.
- Une classe-association est une classe à part entière et peut donc participer à d'autres associations. Cette classe est indissociable de l'association.

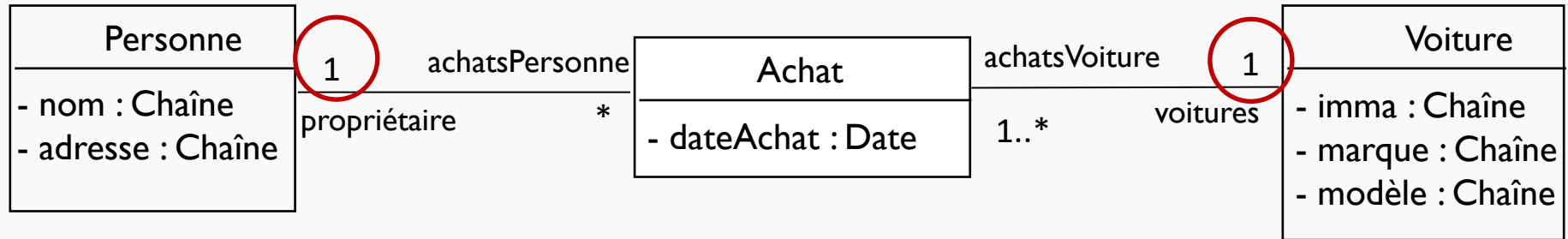


Transformation d'une classe associative

- Schéma mental permettant de traduire une association avec classe associative en 2 associations et donc **4 attributs en java**



Transformation d'une classe association



- Mais attention ce modèle n'est pas équivalent. Il est moins « fort ».
- Il faudra donc coder cette contrainte lors de la réalisation.

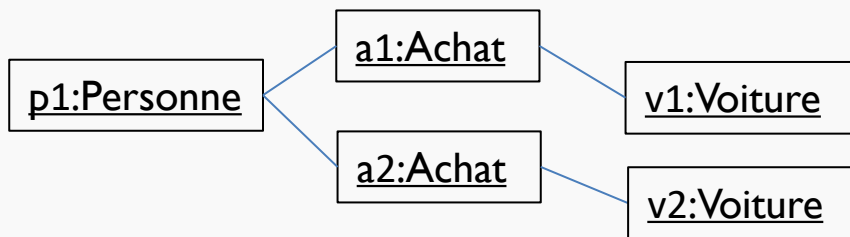


Diagramme d'objets avec une classe association

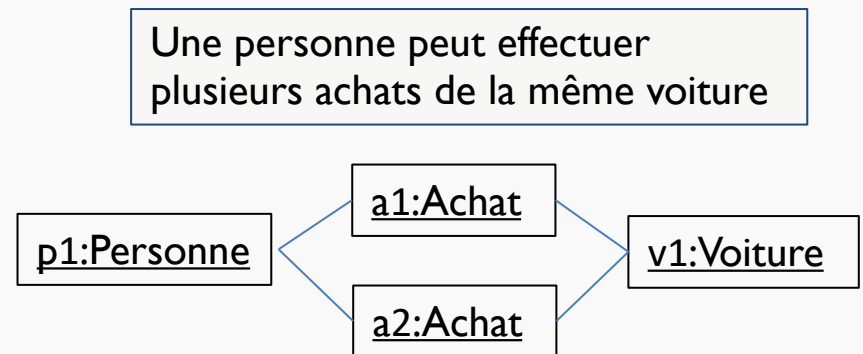
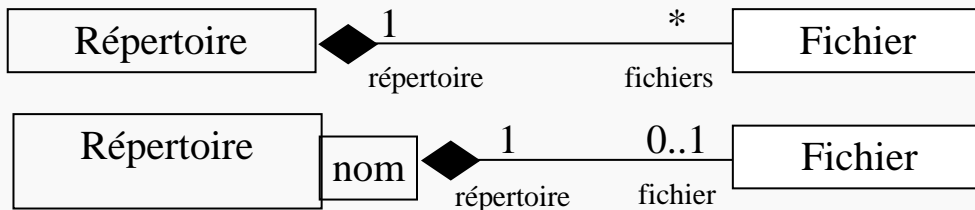


Diagramme d'objets sans classe association

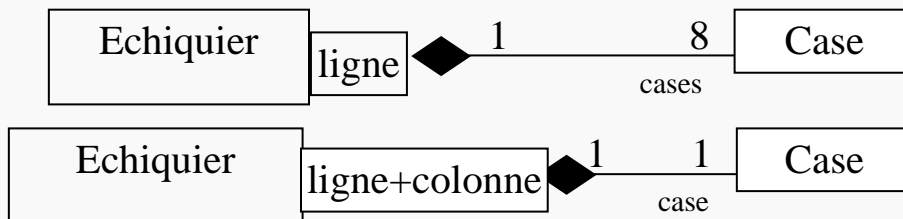
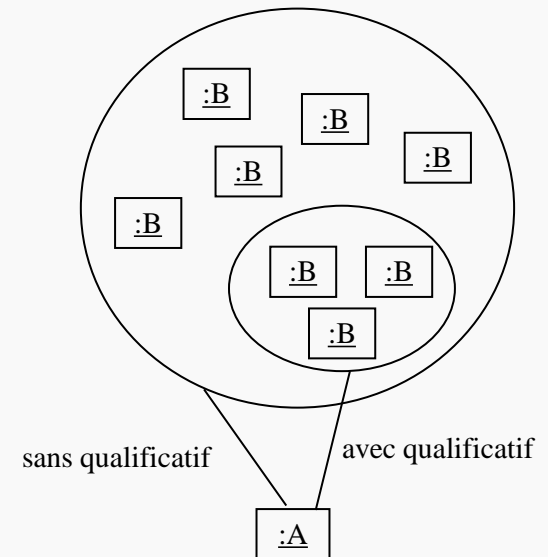
Association qualifiée

- Un qualificatif est un attribut (ou un ensemble d'attributs) dont la valeur sert à partitionner l'ensemble des objets associés à un objet à travers une association.
- Correspond à la notion d'index.



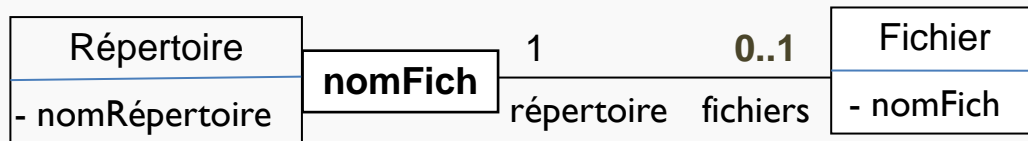
Cas classique : réduction de la multiplicité à 1

Un nom de fichier est unique dans un répertoire



Traduction en Java d'une association qualifiée

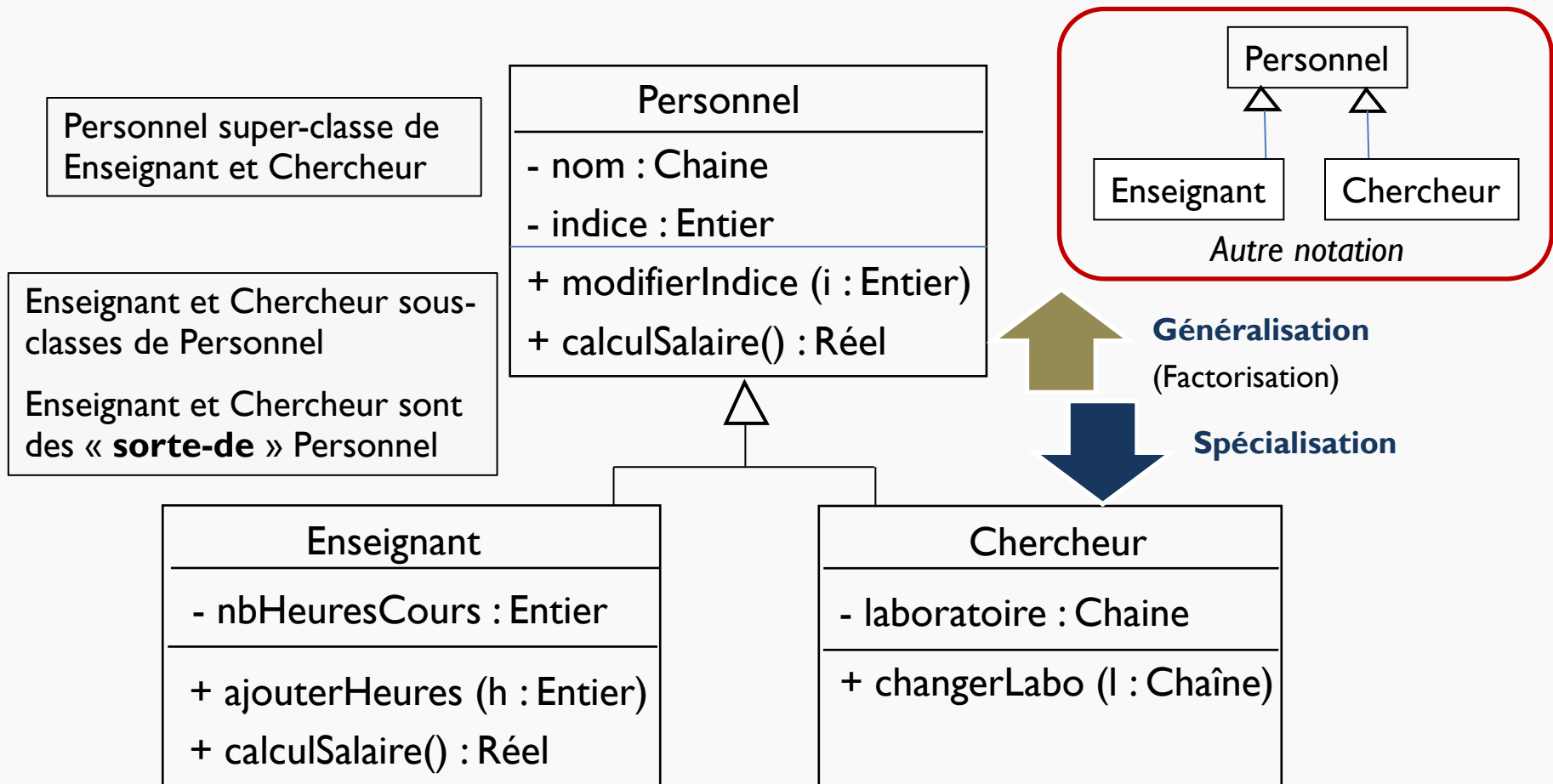
dans le cas où la multiplicité est réduite à 1



```
public class Répertoire {  
    private String nomRepertoire ;  
    private HashMap<String, Fichier> fichiers ;  
    public Fichier getFichier(String nomFich) {  
        return getFichiers().get(nomFisch);  
    }
```

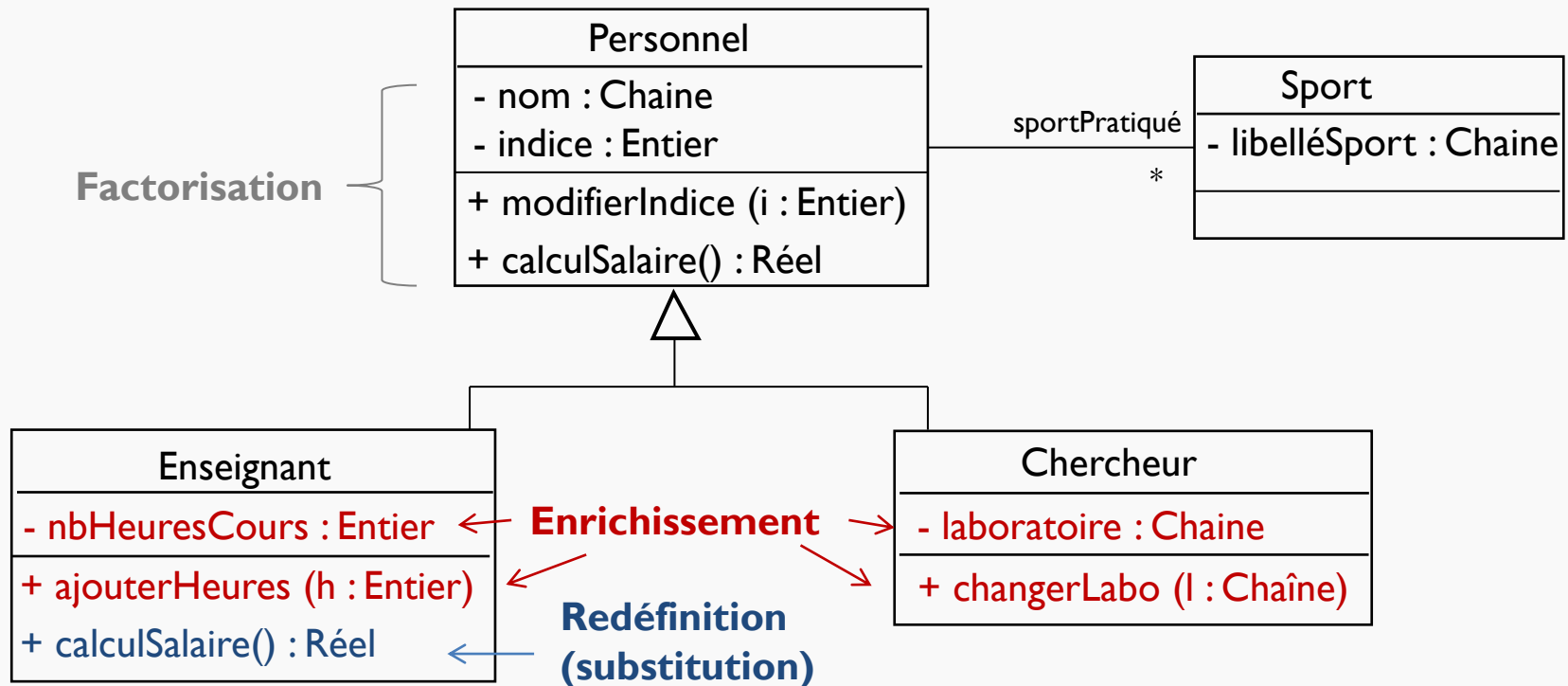
Hiérarchie de classes

- **Généralisation** : factorisation des attributs et opérations communs à un ensemble de classes dans une classe plus générale.
- **Spécialisation** : particularisation d'une classe dans une classe plus spécifique.



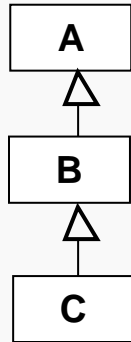
Mécanisme d'héritage

- L'**héritage** est le mécanisme qui permet d'implémenter la relation de généralisation/spécialisation.
 - **Propagation** des attributs, des opérations et des associations d'une classe vers ses sous-classes
 - **Héritage** dans une sous-classe de **tous** les attributs et de **toutes** les opérations de sa super-classe (héritage **complet**)



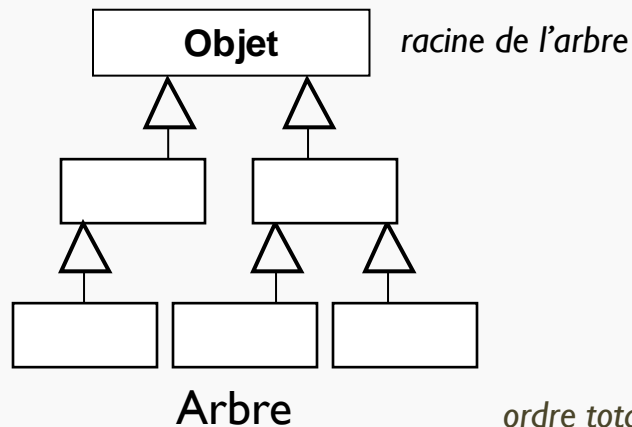
La relation de spécialisation

- **Relation d'ordre entre classes** : transitive, non réflexive, non symétrique

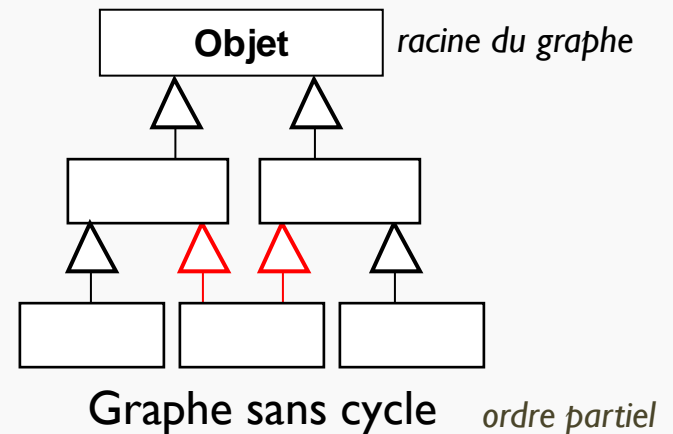


$C \text{ spécialise } B, B \text{ spécialise } A \Rightarrow C \text{ spécialise } A$

spécialisation **simple**



spécialisation **multiple**



Classe abstraite

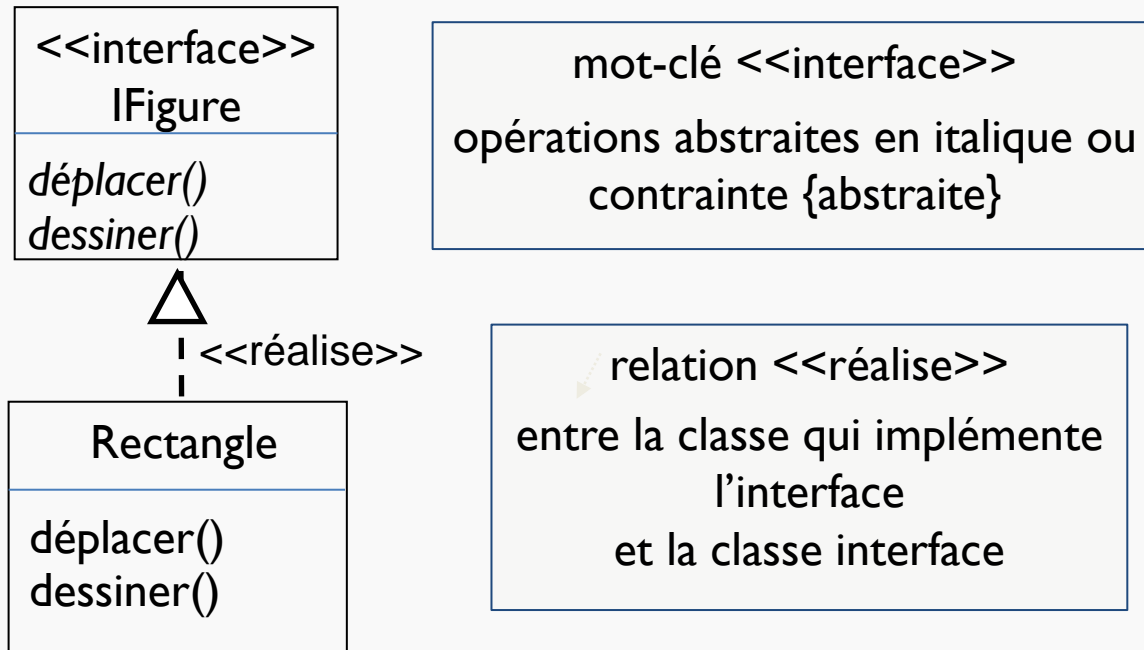
<i>Personne</i>	nom de la classe en italique ou contrainte {abstraite}	Personne {abstraite}
attributs		attributs
+ calculAge() : Entier + <i>modifierIndice(...)</i>	opération abstraite en italique ou contrainte {abstraite}	+ calculAge() : Entier + modifierIndice(...) {abstraite}

- Une classe **abstraite** est une classe qui n'est pas directement instanciable.
- Elle permet de factoriser les propriétés de ses sous-classes en regroupant attributs et opérations communs.
- Une opération abstraite est sans implémentation. Elle doit être redéfinie dans les sous-classes concrètes.

En Java

```
public abstract class A { ...
```

Classe interface



- Une classe **interface** contient uniquement des méthodes abstraites (liste de **services**) qu'une classe concrète doit implémenter si elle veut disposer de cette interface.

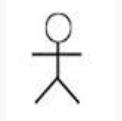
En Java

```
public interface IFigure { ...  
public class Rectangle implements IFigure { ...
```

Stéréotypes

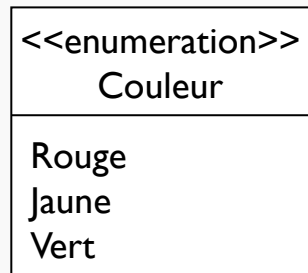
- Un stéréotype permet d'apposer une sémantique particulière aux éléments de modélisation.
- Quelques stéréotypes de classe prédéfinis :

<<acteur>> : classe modélisant un ensemble de rôles joués par un acteur



<<interface>> : classe contenant uniquement les opérations publiques abstraites qu'une classe concrète doit implémenter si elle veut disposer de cette interface

<<énumération>> : classe définissant en extension l'ensemble des valeurs d'un type

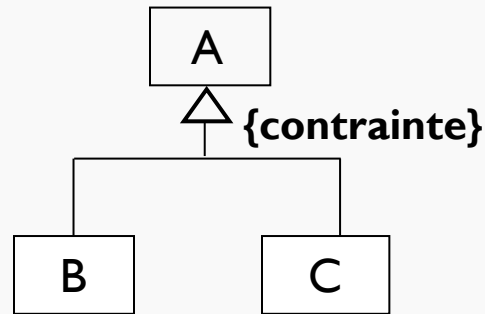


<<instance-de>> : relation de dépendance d'instanciation

<<réalise>> : relation entre la classe qui implémente l'interface et la classe interface

Contraintes sur la spécialisation

- Des **contraintes** peuvent être appliquées sur la relation de spécialisation.



Avec notation (en « ratelier »)

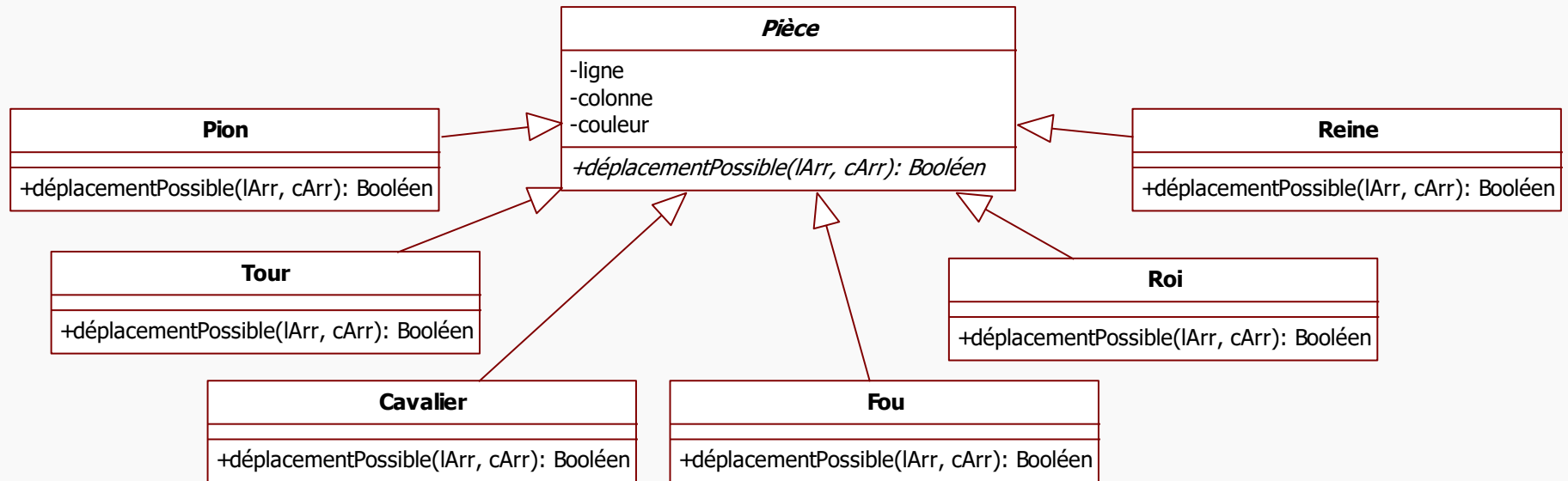
{complète} :

conséquence : la classification de A est terminée

{incomplète} : *par défaut*

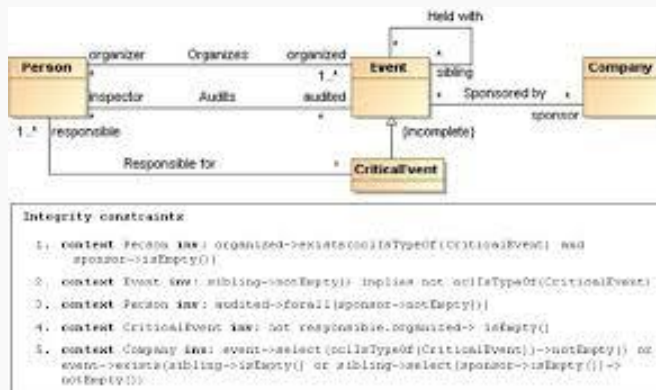
conséquence : la classification de A est extensible

Exemple du jeu d'échec



Object Constraint Language

Un langage de contrainte
Complète les diagrammes UML



modèle graphique insuffisant
pour une spécification
complète et non ambiguë

OCL permet d'exprimer des
contraintes supplémentaires

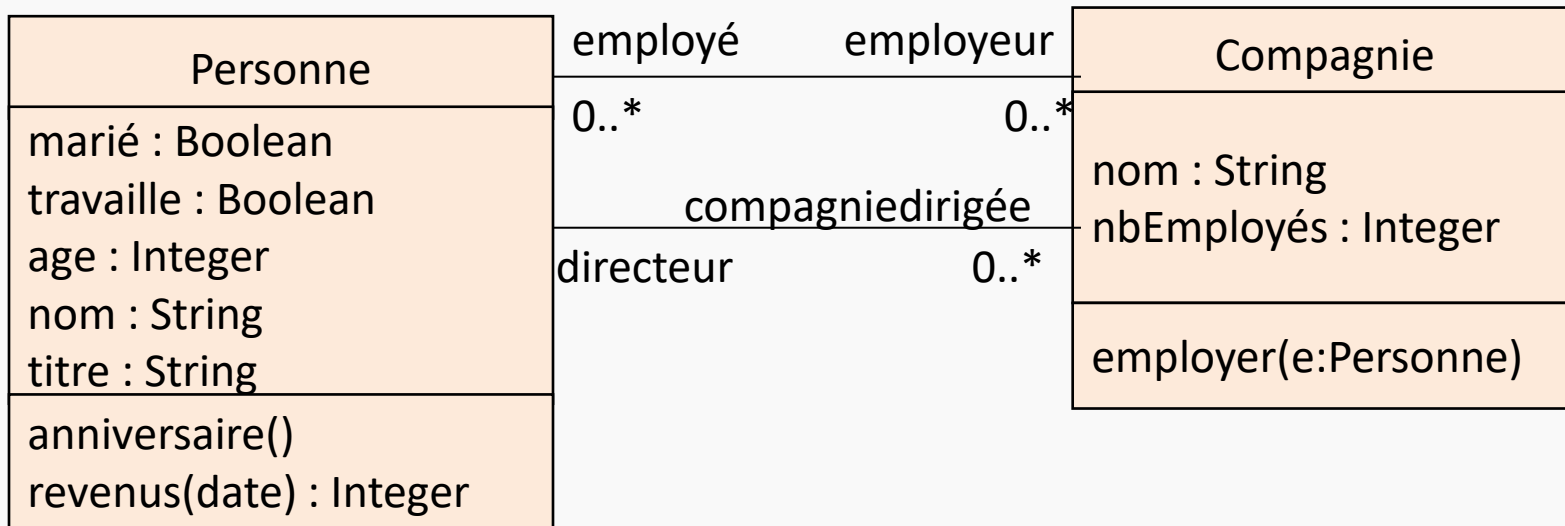
Example OCL

```
context Account::deposit(Real : amount)  
pre: amount > 0  
post: balance = balance@pre + amount
```


Object Constraint Language

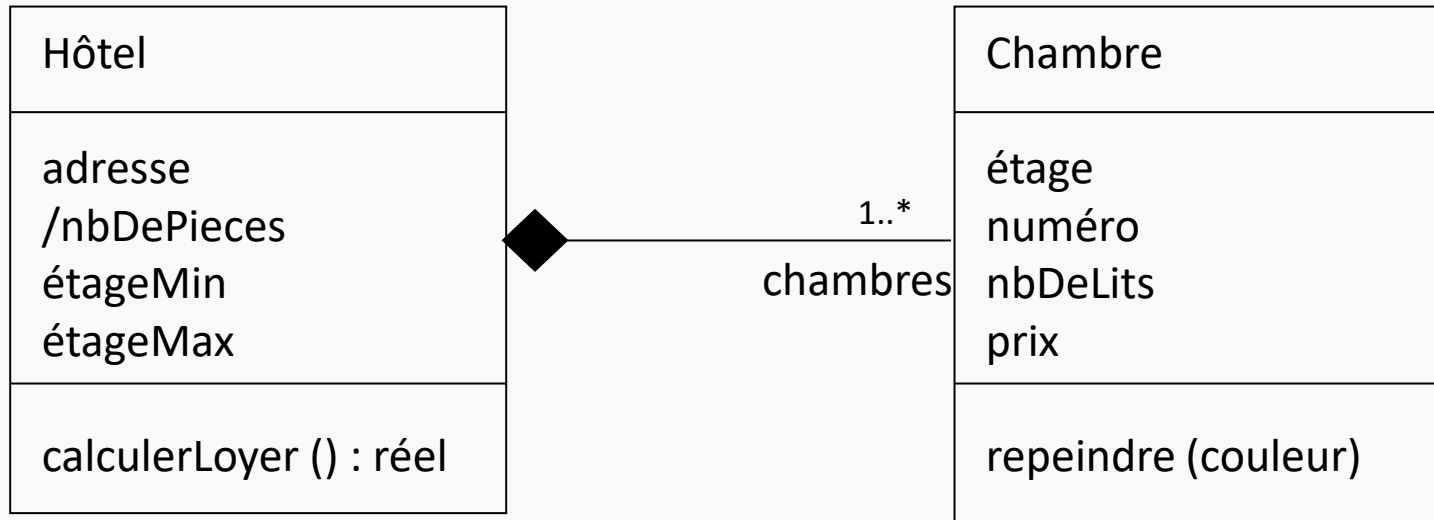
- OCL
 - Un langage formel pour exprimer des contraintes attachées aux diagrammes
 - volontairement simple et possédant une grammaire élémentaire
 - un juste milieu entre langage naturel et langage mathématique
- Permet de décrire des contraintes
 - Invariants sur les classes
 - Pré et post conditions sur les opérations
 - Gardes sur les transitions
 - Définitions des états

Invariants de classes



- Les personnes mariées ont un âge ≥ 18
context Personne **inv** :
self.marié implies self.âge ≥ 18
- Une compagnie a au plus 50 employés
context Compagnie **inv** :
self.employé \rightarrow size ≤ 50

Invariants de classes

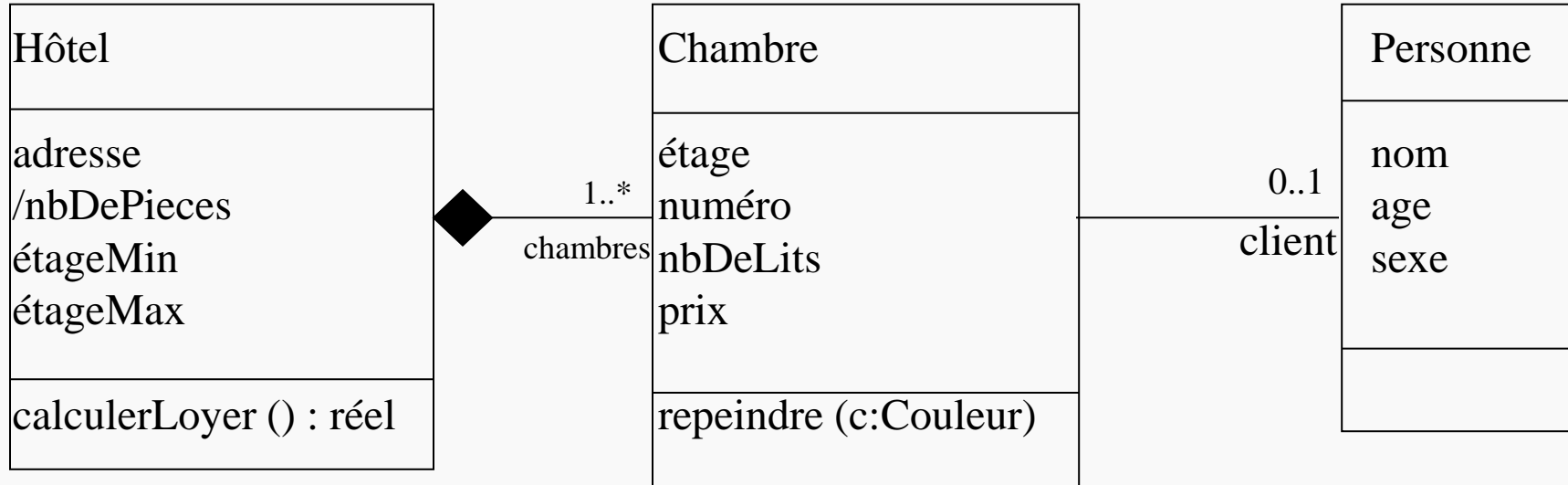


context Chambre **inv** :
self.étage <> 13

context Hotel **inv** :
self.chambres → forall (c | c.étage <= **self.étageMax** and c.étage >= **self.étageMin**)

Inspiré de <http://uml.free.fr/>

Pré/post conditions d'opérations



context Chambre::repeindre (c: Couleur)
pre : self.client → isEmpty
post : self.prix = self.prix@pre + 20

context Hôtel::calculerLoyer() : Real
post : result = self.chambres → select (ch | ch.client → notEmpty).prix → sum

les chambres occupées

leurs prix