

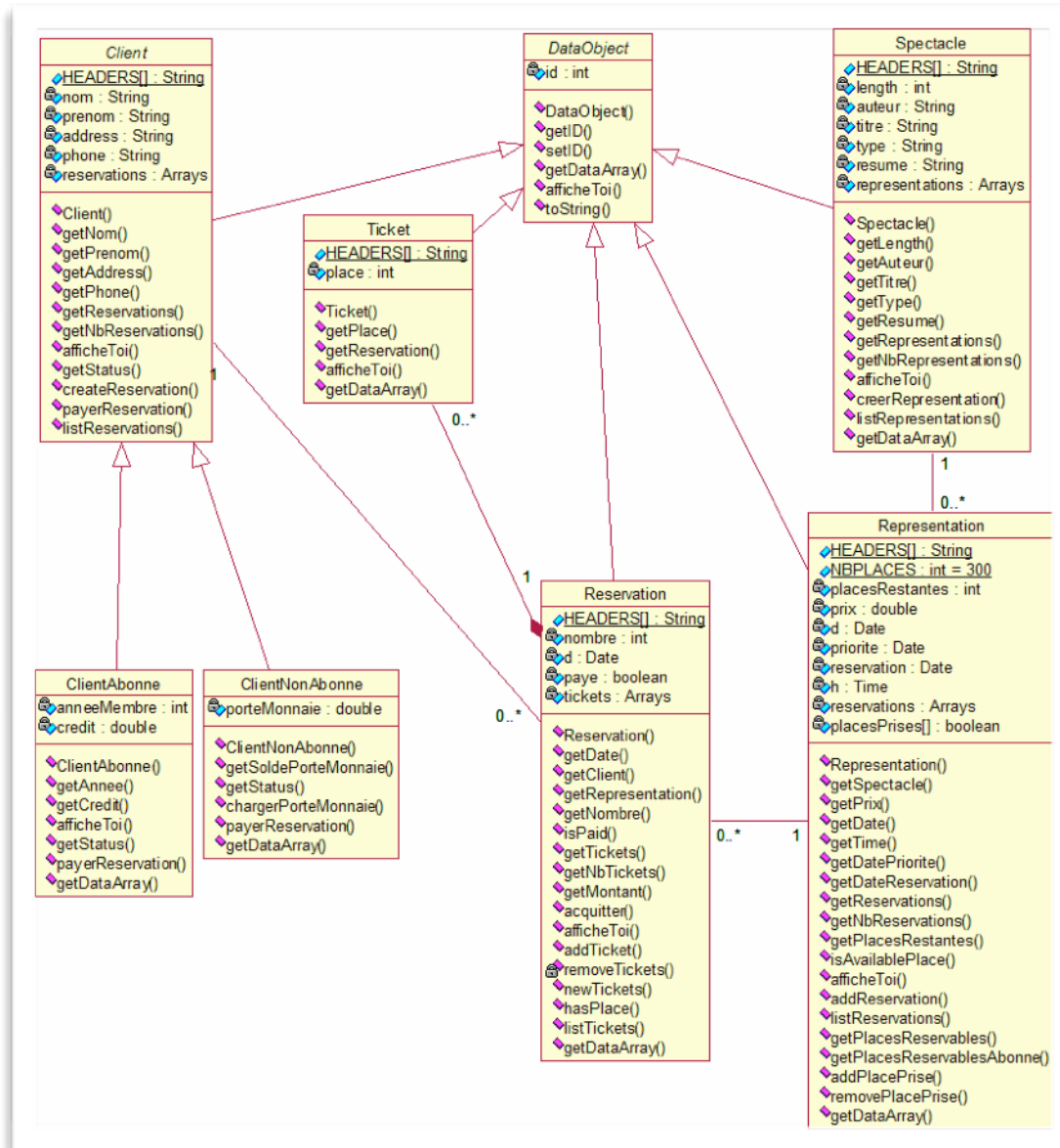
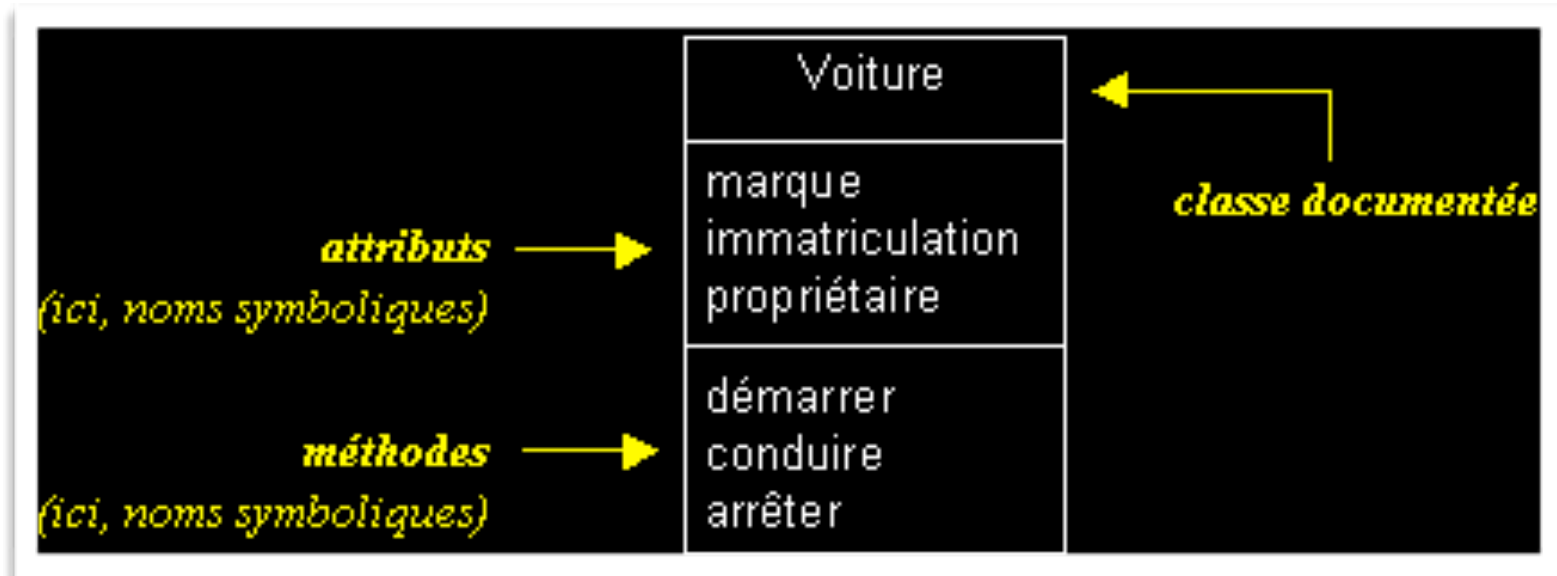


Diagramme de classes

v 2.0 - 2023



Les classes

Diagramme de classes

- ⌞ Le + important de la MOO (Modélisation Orientée Objet)
- ⌞ Le diagramme de cas d'utilisation --> système (acteurs)
- ⌞ Le diagramme de classes --> la structure interne

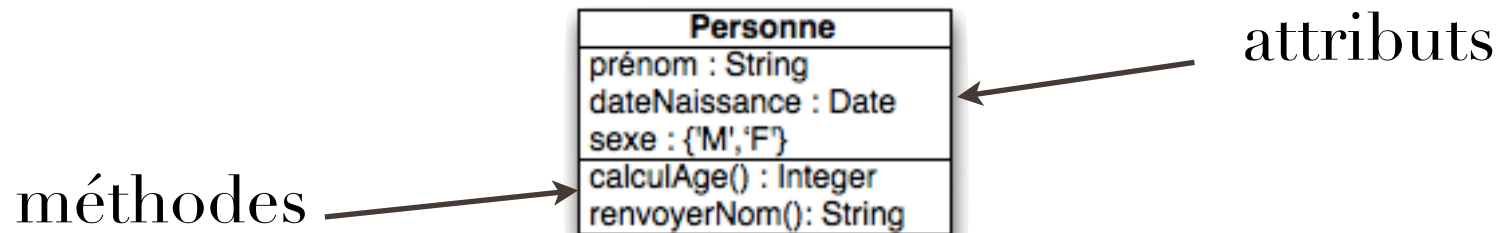
De l'objet à la classe

- ~ Au commencement était l'objet !
- ~ Afin de limiter la complexité on fait des choix
- ~ Un objet est donc une abstraction d'un concept
Ex: Un élève -> nom, prénom, adresse, etc.
- ~ On dit que nom, prénom sont des attributs de l'objet
- ~ Pour un objet on donne des valeurs à ces attributs
Ex : nom=PICARD Prénom=Alain ...

De l'objet à la classe

- ~ Certains objets ont des attributs identiques
Ex : Chaque élève d'une classe à un nom, prénom, etc
- ~ L'idée est de rassembler ces attributs ça sera le rôle des classes
- ~ On peut avoir également besoin de faire des actions sur les objets : -> méthodes
- ~ Une classe est une description de l'objet ayant une sémantique, des attributs, des méthodes, des relations en commun.

Un exemple de classe



➡ Le nom de la classe doit être significatif

➡ Liste des attributs avec le type

Remarque : Dans l'approche Objet tout est objet (sauf exception)
y compris les types des attributs : classe String, classe Date, classe Integer

➡ Les méthodes avec paramètre si besoin



Implémentation Java

Personne
prénom : String
dateNaissance : Date
sexe : {'M','F'}
calculAge() : Integer
renvoyerNom() : String

```
1 import java.util.Calendar;
2
3 public class Personne {
4     // Attributs
5     String prenom;
6     Calendar dateNaissance;
7     Sexe sexe;
8
9     // Constructeur
10    public Personne(String prenom, Calendar datenaissance, Sexe sexe) {
11        this.prenom = prenom;
12        this.dateNaissance = datenaissance;
13        this.sexe = sexe;
14    }
15
16    // Méthodes
17    public int calculAge() {
18        Calendar maintenant = Calendar.getInstance();
19        return maintenant.get(Calendar.YEAR) - this.dateNaissance.get(Calendar.YEAR);
20    }
21
22    public String renvoyerPrenom() {
23        return this.prenom;
24    }
25 }
```



Implémentation Java

Personne
prénom : String
dateNaissance : Date
sexe : {'M','F'}
calculAge() : Integer
renvoyerNom() : String

new()

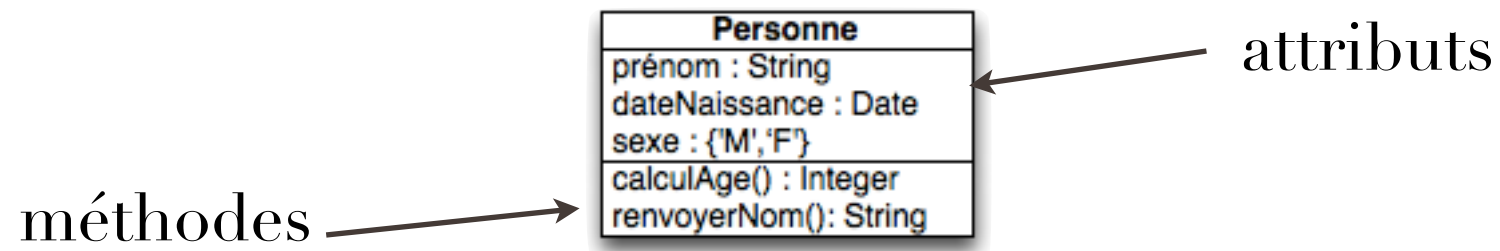
Objet

```
TestPersonne.java ✕
1 import java.util.Calendar;
2 import java.util.Date;
3 public class TestPersonne {
4
5     public static void main(String[] args) {
6
7         Calendar d = Calendar.getInstance();
8         // Création de la Date
9         d.set(1970, 4, 12);
10
11         Personne PerA = new Personne("Jean", d, Sexe.M);
12         System.out.println("L'age de "+PerA.renvoyerPrenom()+ " est de "+PerA.calculAge());
13     }
14
15 }
```

Problems Javadoc Declaration Console Error Log Git Staging Git Repositories History Call Hierarchy

<terminated> TestPersonne [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_40.jdk/Contents/Home/bin/java (29 nov. 2014 18:32:34)
L'age de Jean est de 44

Un exemple d'objet



➡ Un objet est une instance d'une classe

➡ Ex : Jean, 12/04/1970, M

La méthode `calculAge()` donne 52 ans
la méthode `renvoyerPrenom()` retourne Jean

Faire exercice 1

Correction exercice 1

Case
couleur : (B, N)
rangee : 1..8
colonne : a..h

On a ici des propriétés mais pas de méthodes (pour l'instant)

Correction exercice 1

Avion
Immatriculation
Longueur
Poids
Vitesse
Altitude
Vitesse_max
Altitude_max
Autonomie
décoller()
atterrir ()
sortirtrain ()
acceler ()
ralentir ()
monter()
descendre()

Classe avec propriétés et méthodes

Encapsulation

- ~ **L'encapsulation** -> idée de **protéger** le coeur du système

Un objet ne doit pas être «modifié» sans passer par ses méthodes.

Exemple : Pour accélérer sur une voiture vous passez par la pédale d'accélération et non via le moteur.

L'avantage est que les méthodes peuvent protéger, borner, contrôler les demandes.

- ~ UML définit **4 niveaux d'encapsulation** pour une propriété

Les 4 niveaux d'encapsulation

La représentation de la visibilité d'une propriété est la suivant

- + public (visible de partout)
- # protected (visible de la classe ou des classes qui héritent)
- - private (que la classe)
- Si pas de signe -> Visible du paquetage (rassemblement de classe)

Exemple

Personne
-prénom : String
-dateNaissance : Date
-sexe : {'M','F'}
+calculAge() : Integer
+renvoyerNom(): String

Ici toutes les propriétés sont en *private*
(*c'est souvent le cas*)

Et les méthodes en *public*
(*Elles pourront être appelé de partout !*)

Attention ce n'est pas toujours le cas :
On peut avoir des méthodes *private* !



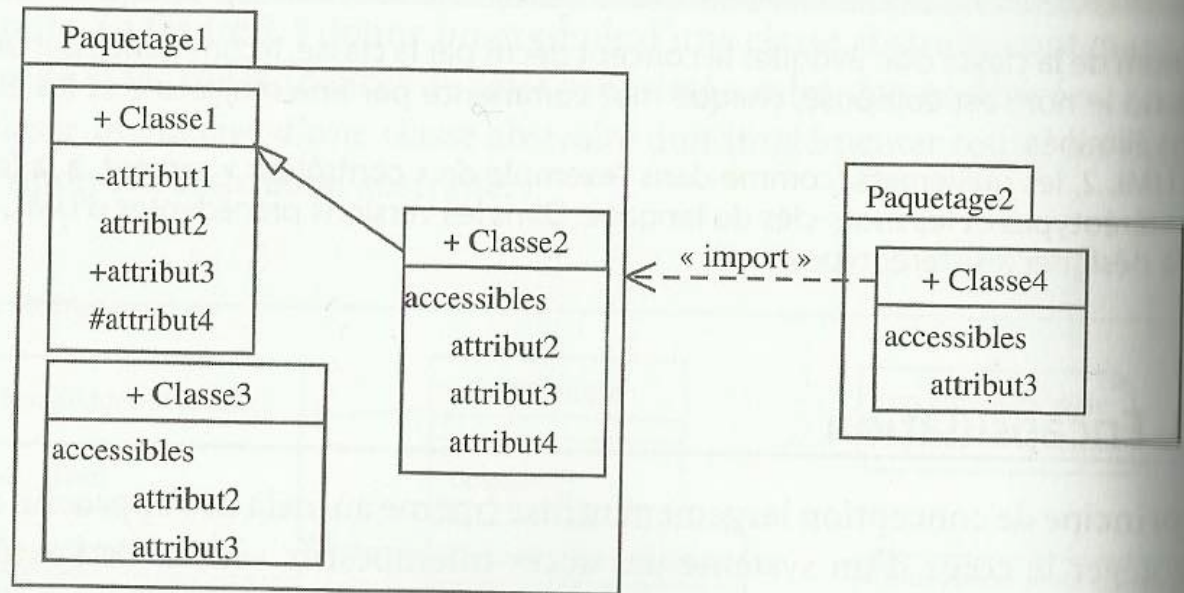
Implémentation Java

Personne
-prénom : String
-dateNaissance : Date
-sexe : {'M','F'}
+calculAge() : Integer
+renvoyerNom() : String

```
1 import java.util.Calendar;
2
3 public class Personne {
4     // Attributs
5     private String prenom;
6     private Calendar dateNaissance;
7     private Sexe sexe;
8
9     // Constructeur
10    public Personne(String prenom, Calendar datenaissance, Sexe sexe) {
11        this.prenom = prenom;
12        this.dateNaissance = datenaissance;
13        this.sexe = sexe;
14    }
15
16    // Méthodes
17    public int calculAge() {
18        Calendar maintenant = Calendar.getInstance();
19        return maintenant.get(Calendar.YEAR) - this.dateNaissance.get(Calendar.YEAR);
20    }
21
22    public String renvoyerPrenom() {
23        return this.prenom;
24    }
25 }
```

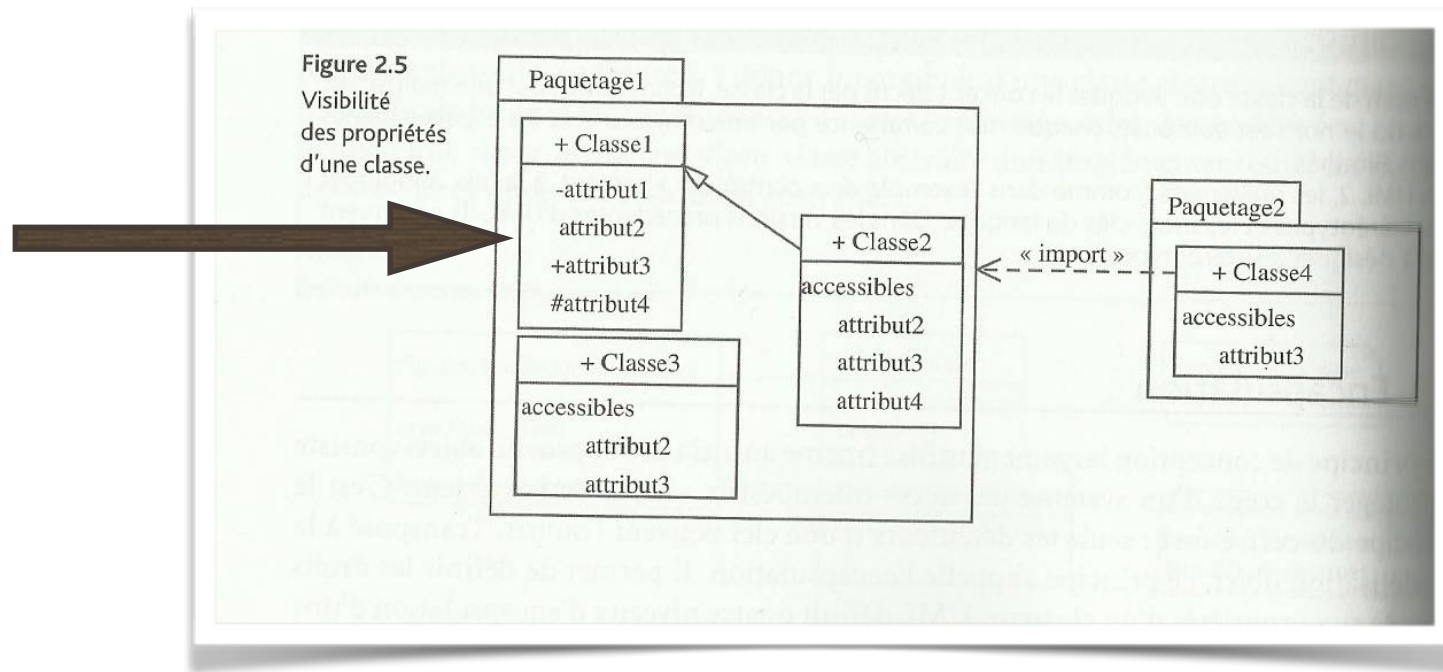
Exercice

Figure 2.5
Visibilité
des propriétés
d'une classe.



Quel est la visibilité de l'attribut1 ?

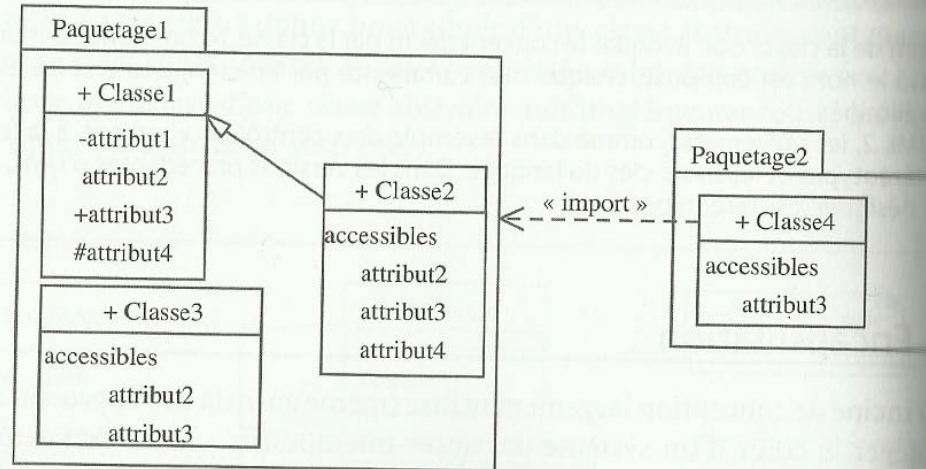
Exercice



L'attribut1 n'est accessible que dans la Classe1

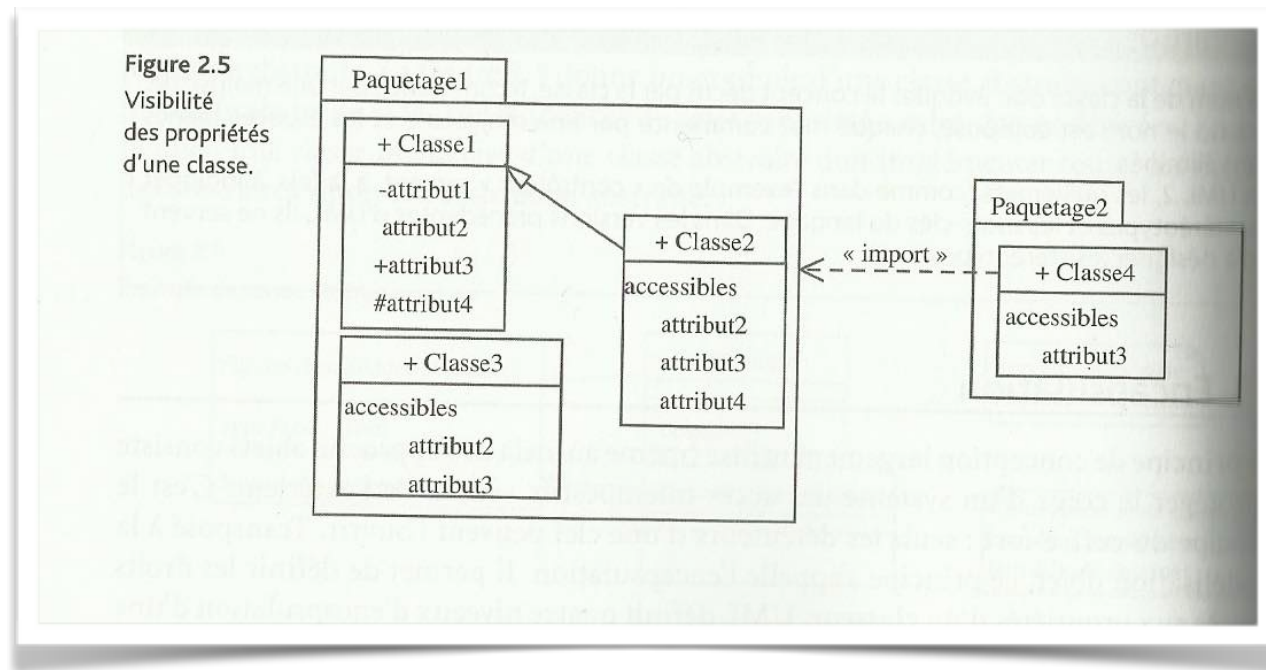
Exercice

Figure 2.5
Visibilité
des propriétés
d'une classe.



Quel est la visibilité de l'attribut2 ?

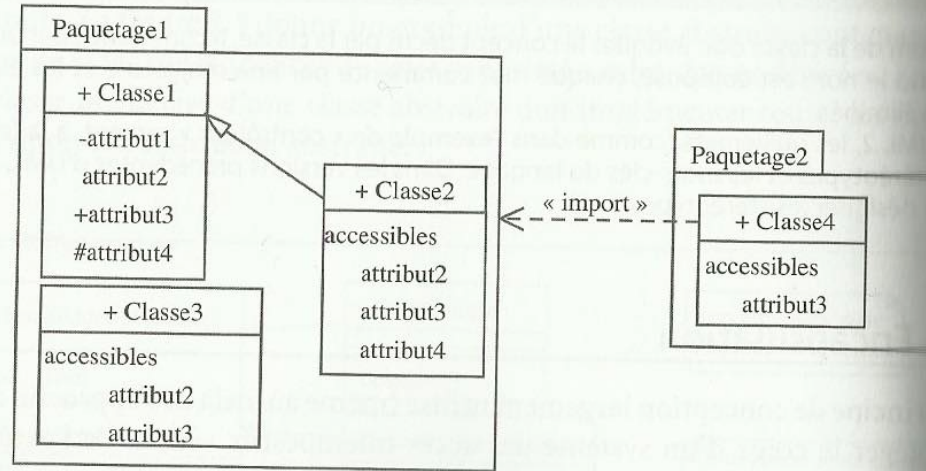
Exercice



L'attribut2 est « public » dans la paquetage. Donc visible à partir des Classe1, Classe2, Classe3

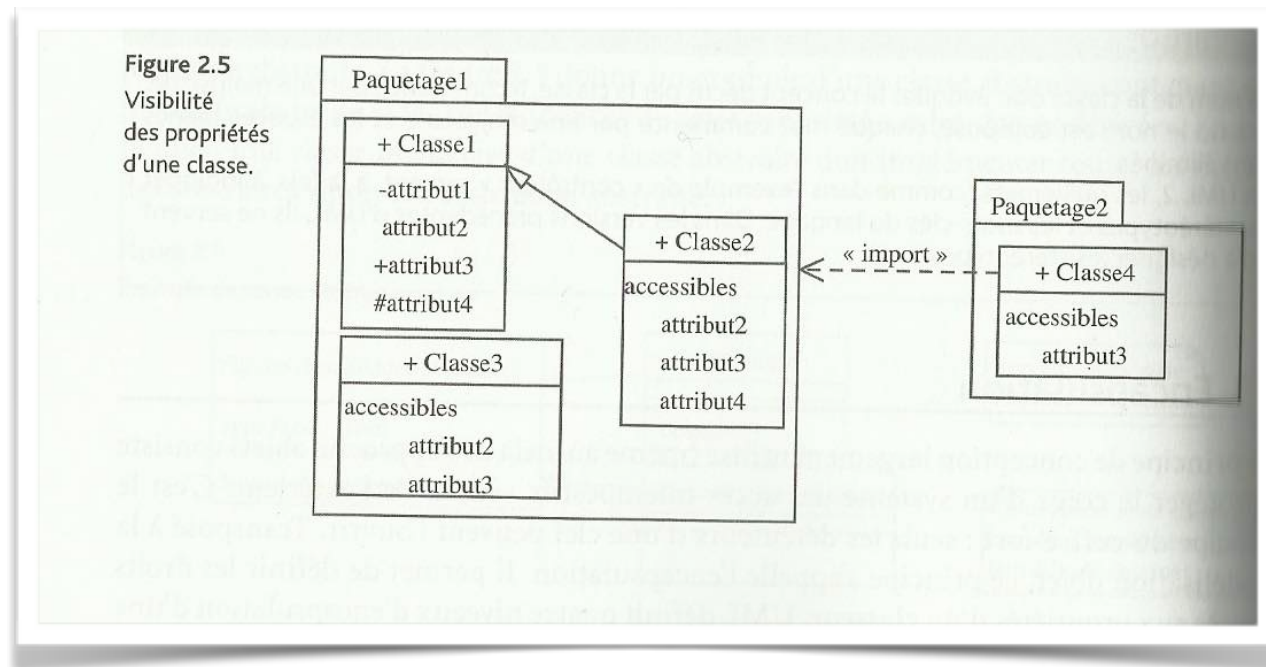
Exercice

Figure 2.5
Visibilité
des propriétés
d'une classe.



Quel est la visibilité de l'attribut3 ?

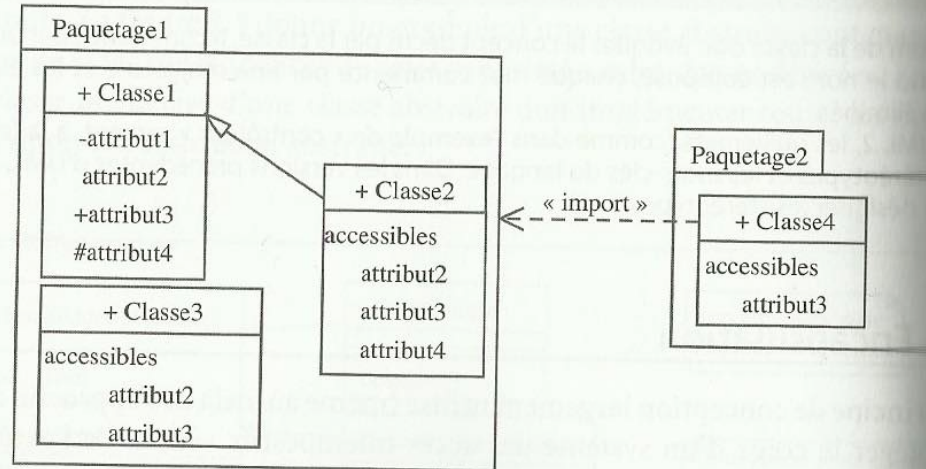
Exercice



L'attribut3 est visible depuis: Classe1,
Classe2, Classe3, Classe4

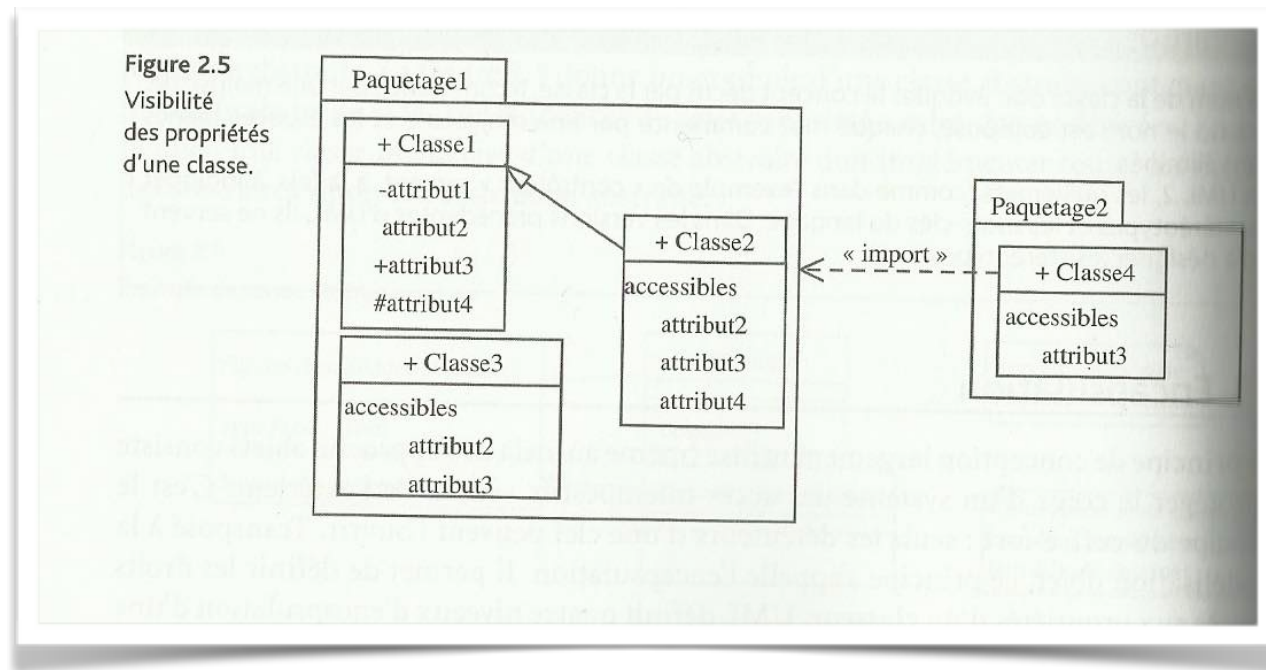
Exercice

Figure 2.5
Visibilité
des propriétés
d'une classe.

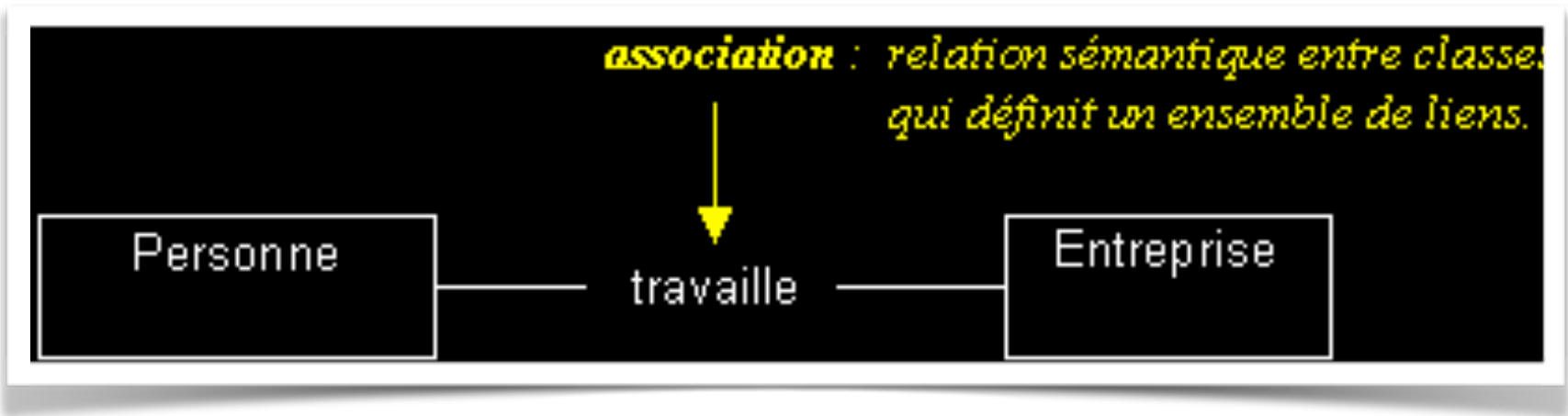


Quel est la visibilité de l'attribut4 ?

Exercice



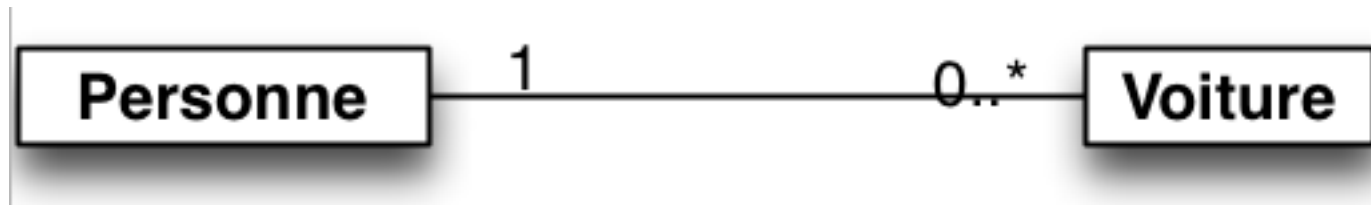
L'attribut4 n'est visible que dans les classes
Classe1 et Classe2



Relation entre les classes

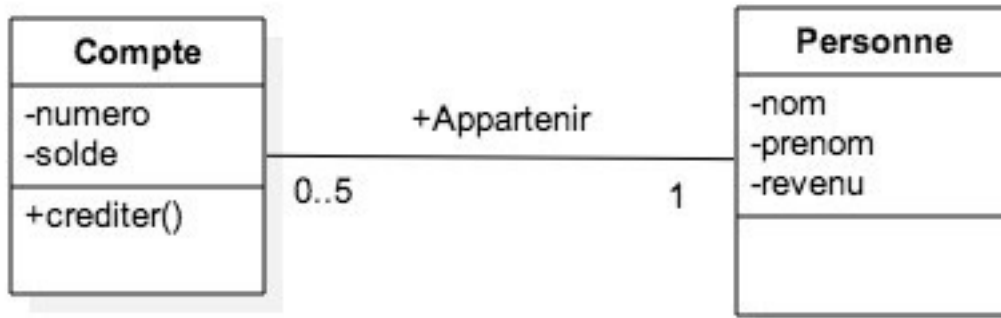
Association

- ~ Une association représente une relation sémantique durable entre deux classes
- ~ Exemple : Une personne peut posséder des voitures



Implémentation Java

Les associations entre classes sont tout simplement représentées par des références. Les classes associées possèdent en attribut une ou plusieurs références vers l'autre classe



```
Compte.java
Personne.java

1 class Compte
2 {
3     private int numero;
4     private float solde;
5
6     private Personne proprietaire; //l'attribut proprietaire est une référence
7
8     public Compte(int num, Personne propr) //constructeur
9     {
10         numero = num;
11         proprietaire = propr;
12         solde = 0;
13     }
14
15     public void crediter( )
16     { ... }
17     ... //autres méthodes
18 }
```

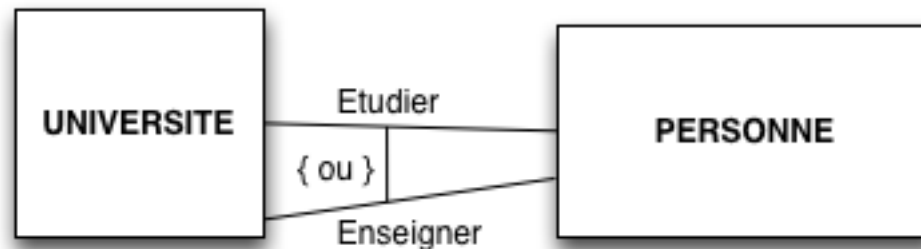
```
Compte.java
Personne.java

1 class Personne
2 {
3     private String nom;
4     private String prenom;
5     private float revenu;
6
7     private Compte[] tabcompte = new Compte[5];
8
9     public Personne(String n, String p, float r)
10    {
11        nom = n;
12        prenom = p;
13        revenu = r;
14    }
15
16    //autres méthodes
17 }
```

Attention : le tableau est construit, mais pas les comptes à l'intérieur !

Contraintes entre associations

- ~ Les personnes qui sont associées à l'université sont des étudiants ou des professeurs



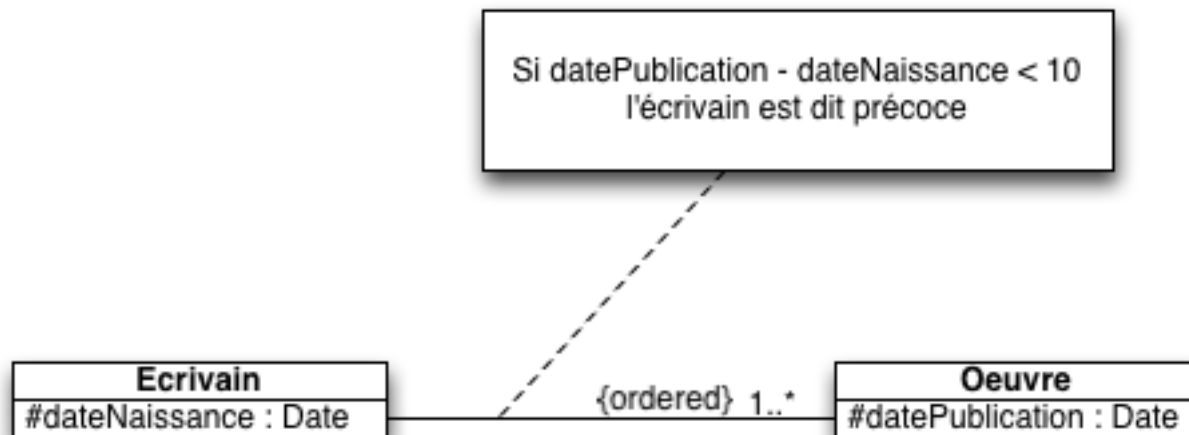
On met juste le nom des classes n'ayant pas d'autres informations

La contrainte est simplement indiqué avec le ou (ou exclusif)

La spécialisation de Personne avec Etudiant et Professeur est une autre solution voir plus loin

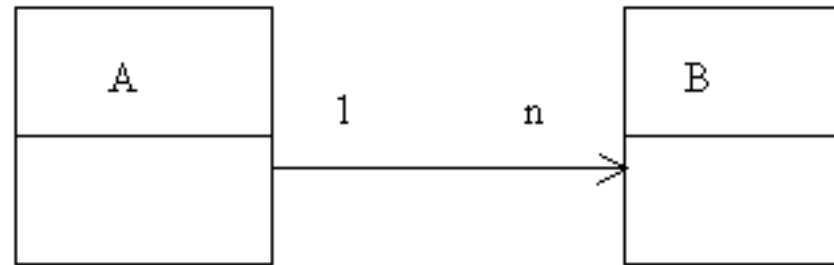
Contraintes entre associations

- Un écrivain possède au moins une oeuvre. Ses oeuvres sont ordonnées selon l'année de publication. Si la première publication est faite avant l'âge de dix ans, l'écrivain est dit «précoce».



Navigation restreinte

On peut assimiler une association comme une relation permettant un parcours de l'information. On "passe" d'une classe à l'autre en parcourant les associations; ainsi par défaut les associations sont "navigables" dans les deux sens. Il peut être pertinent au moment de l'analyse de réduire la navigabilité à un seul sens.



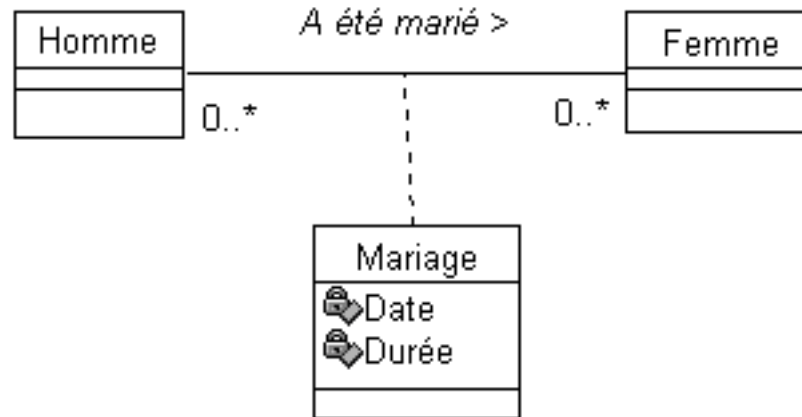
```
public class A
{
    private B[ ] tabB = new B[ n ] ;
}
public class B
{
    // pas de référence à un objet de la classe A
}
```

Association porteuse

L'association entre deux classes peut elle même posséder des attributs. Par exemple:

" Un homme peut avoir été marié plusieurs fois, et réciproquement, les dates et durées des mariages nous importent ".

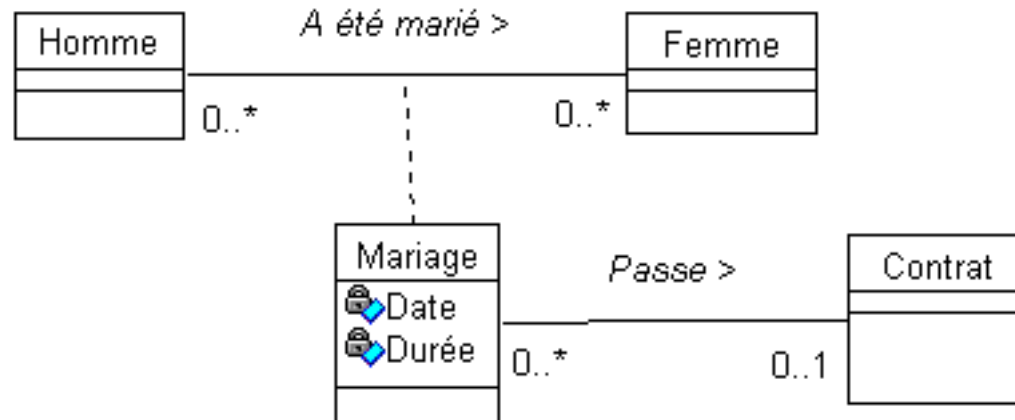
Les attributs "date" et "durée" ne concernent pas seulement chaque instance des deux classes mais des couples d'instances. Dans ce cas l'association va jouer à la fois le rôle d'association et le rôle de classe, on parle alors de classe-association.



Remarque : La classe-association ne porte pas de valeurs de multiplicité. Chaque instance de la classe-association **Mariage** est reliée à un couple d'instances des classes **Homme** et **Femme**.

Une classe association peut participer à d'autres relations.

Imaginons que le mariage soit l'objet d'un contrat de mariage type.

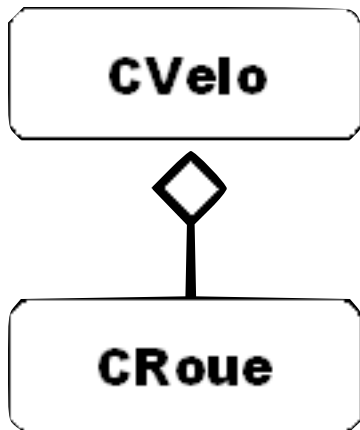




- ~ Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance
- ~ Elle signifie «contient», «est composé de» sans pour autant les lier physiquement.
- ~ En programmation cela peut se traduire par des pointeurs ou des références.



Implémentation Java



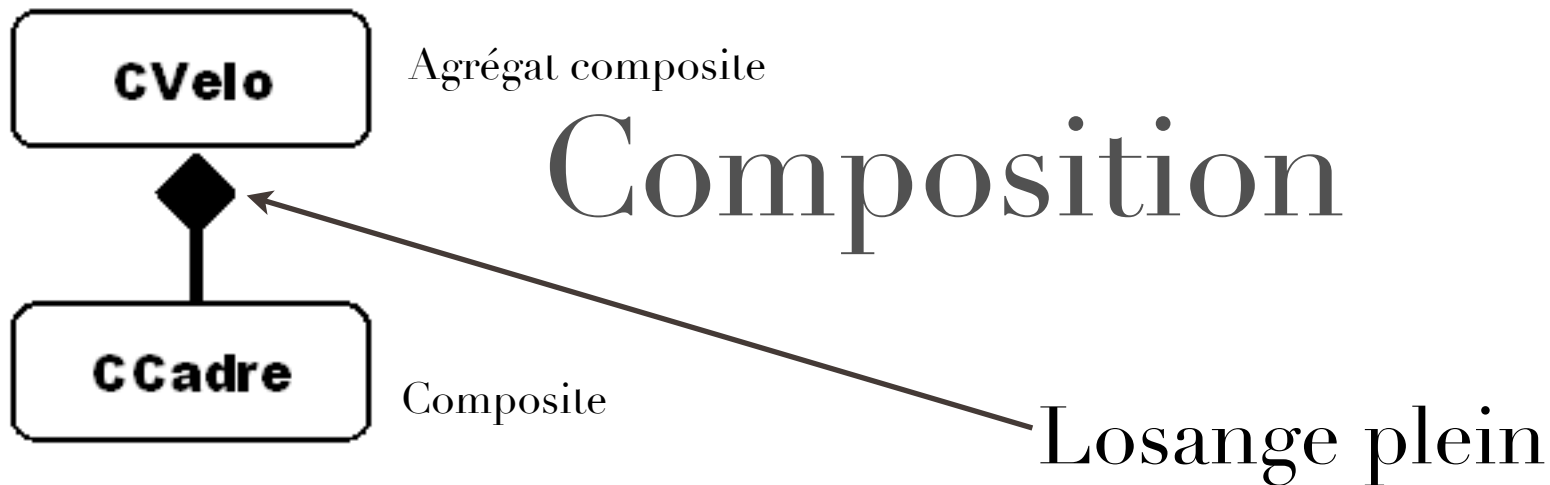
```
public class CVelo {
    private String couleur;
    private float prix;
    protected CRoue r1,r2;

    public CVelo(String c, float p) {
        this.couleur = c;
        this.prix = p;
    }

    // Agrégation
    public float getDiametre(CRoue r) {
        return r.getDiametre();
    }

    public float getPrix() {
        return prix;
    }

    public void setPrix(float prix) {
        this.prix = prix;
    }
}
```

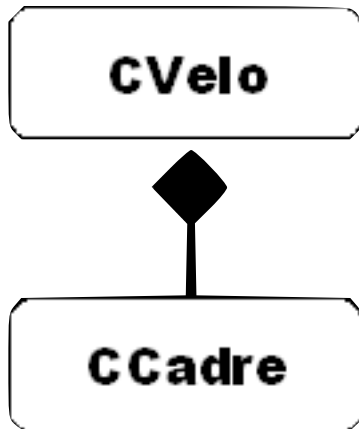


- ~ Une composition est une **agrégation** plus forte
- ~ Un élément **ne peut appartenir qu'à un seul agrégat composite** (agrégation non partagée)
- ~ La destruction de l'agrégat composite entraîne la destruction de tous ses éléments (le composite est responsable du cycle de vie des parties).
- ~ En terme de programmation cela peut se traduire par l'inclusion d'une classe dans une autre.
Ex : la destruction du vélo entraine la destruction du cadre

Faire exercice 2



Implémentation Java



```
public class CVelo {
    private String couleur;
    private float prix;
    protected CRoue r1,r2;
    protected CCadre c;

    public CVelo(String c, float p) {
        this.couleur = c;
        this.prix = p;
        // Composition
        this.c = new CCadre("Carbone", 7);
    }

    // Agrégation
    public float getDiametre(CRoue r) {
        return r.getDiametre();
    }

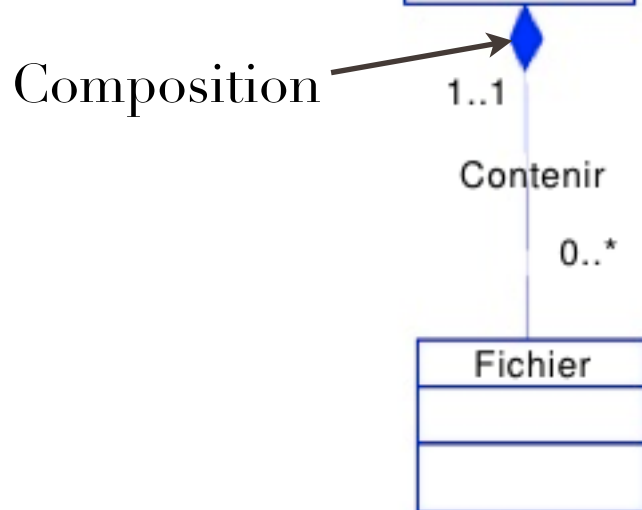
    public float getPrix() {
        return prix;
    }

    public void setPrix(float prix) {
        this.prix = prix;
    }
}
```

Correction Exercice 2

Un répertoire contient des fichiers : il s'agit au moins d'une agrégation

Est-ce une composition ?



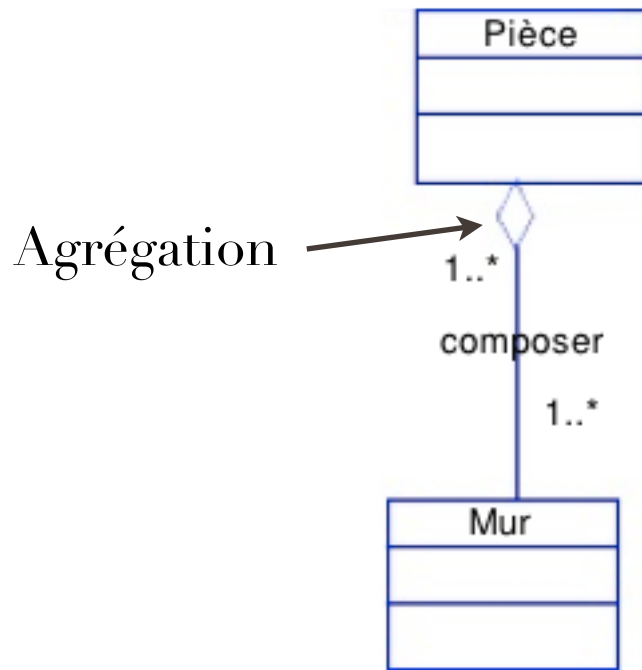
Deux critères:

1. **La multiplicité ne doit pas être supérieur à 1**
C'est le cas un fichier appartient à un seul répertoire
2. **Le cycle de vie des parties doit dépendre du composite**

C'est encore le cas la destruction du répertoire entraine la destruction des fichiers inclus

Donc c'est une composition !

Correction Exercice 2



«Une pièce contient des murs»

Est-ce une composition ?

Deux critères:

1. **La multiplicité ne doit pas être supérieur à 1**
Faux. Un mur peut appartenir à deux pièces.

C'est donc une agrégation !

Classe abstraite

- ~ C'est une classe qui va servir de modèle.
Mais qui n'existe pas en tant que tel.
Ex : *Un mammifère n'existe pas, un chat oui !*
Mammifère est une abstraction.
- ~ Une classe abstraite ne peut donc être instanciée :
C'est à dire qu'on ne peut pas créer d'objet à partir de cette classe.
- ~ Son nom est indiqué en *italique* !

Méthode abstraite

- ~ C'est une méthode dont on connaît l'entête (cad les paramètres) mais pas le corps (le code qui va effectuer le travail)
- ~ On ne sait pas encore comment on va faire
- ~ Mais on sait qu'on va devoir le faire.
Cela va obliger les classes qui vont hériter à définir cette méthode.
- ~ Ex : dessiner() pour une figure géométrique
- ~ Important : une classe qui a une méthode abstraite devient automatique abstraite

Héritage

- ~ C'est l'un des intérêts de la POO
- ~ Sans héritage les classes abstraites ainsi que les méthodes abstraites n'auraient aucun intérêt !
- ~ Cela permet de réutiliser du code, de factoriser
- ~ On parle de Généralisation-Spécialisation



Implémentation Java

```
2 public abstract class Figure {
3     private String couleur;
4
5     public Figure(String Couleur)
6     {
7         this.couleur = Couleur;
8     }
9
10    /**
11     * Renvoie la surface de cette figure
12     */
13    public abstract float getSurface();
14
```

```
15    /**
16     * Renvoie la couleur et la surface de cette figure
17     */
18    public String toString()
19    {
20        return "Figure " + this.couleur + " (" + getSurface() + " m)";
21    }
22 }
```

```
2 public class Rectangle extends Figure {
3
4     private float longueur;
5     private float largeur;
6
7     public Rectangle(String Couleur, float longueur, float largeur) {
8         super(Couleur);
9         // TODO Auto-generated constructor stub
10        this.largeur = largeur;
11        this.longueur = longueur;
12    }
13
14    @Override
15    public float getSurface() {
16        return this.largeur * this.longueur;
17    }
18
19 }
```



Implémentation Java

```
1 public class Cercle extends Figure {  
2     private float rayon;  
3  
4     public Cercle(String Couleur, float rayon) {  
5         super(Couleur);  
6         // TODO Auto-generated constructor stub  
7         this.rayon = rayon;  
8     }  
9  
10    @Override  
11    public float getSurface() {  
12        // TODO Auto-generated method stub  
13        return (float) Math.PI * this.rayon * this.rayon;  
14    }  
15 }
```



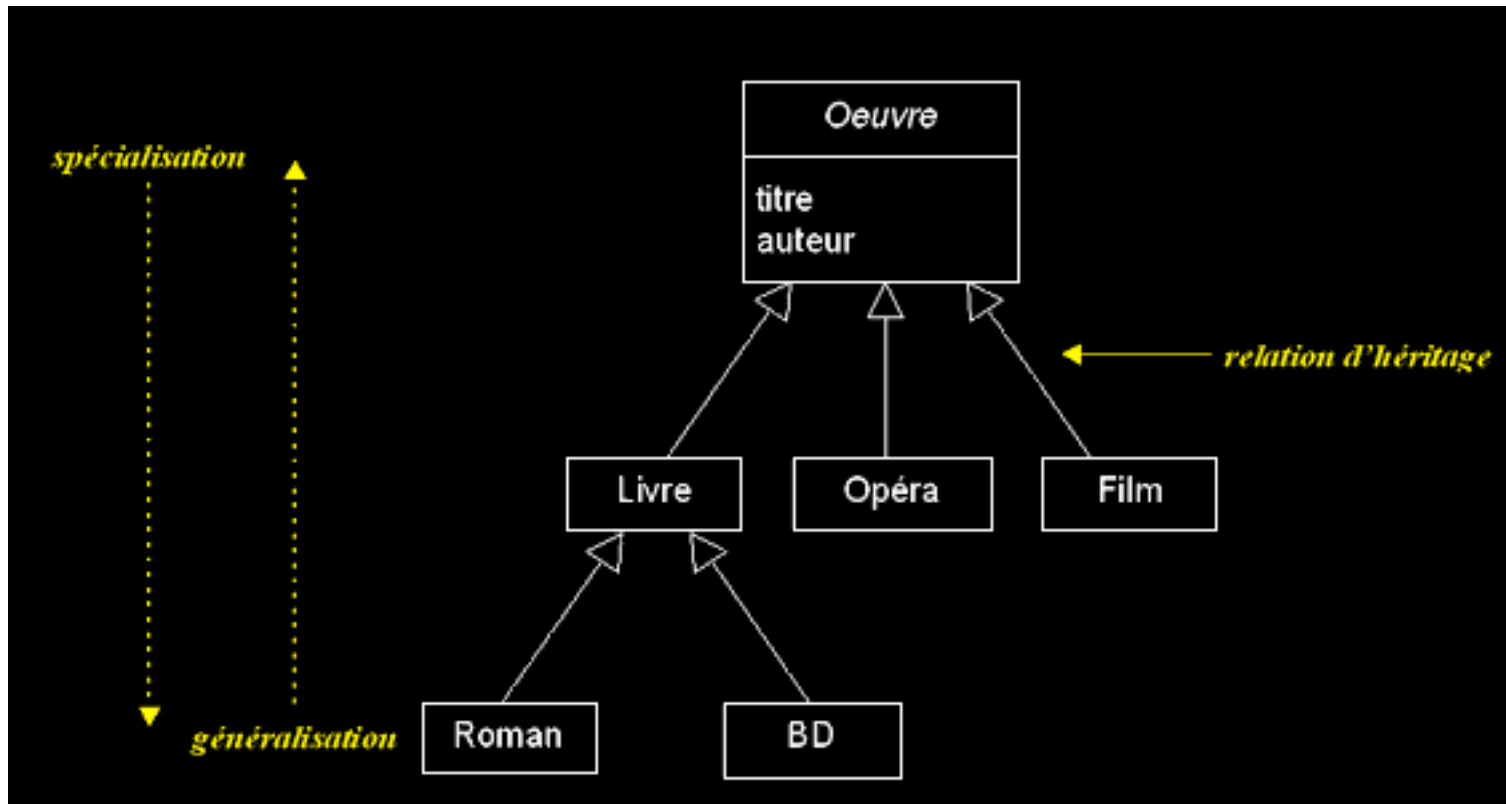
Implémentation Java

```
1
2 public class ComparaisonFigures {
3
4     public static void main(String[] args) {
5         Rectangle rectangleJaune = new Rectangle("jaune", 20, 10);
6         Rectangle rectangleRouge = new Rectangle("rouge", 2, 1.5f);
7         Cercle cercleBleu = new Cercle("bleu", 5);
8
9         System.out.println(cercleBleu.toString());
10        System.out.println(cercleBleu.getSurface());
11    }
12
```

Problems @ Javadoc Declaration Console Error Log Git Staging Git Repos

<terminated> ComparaisonFigures [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_40.jdk/Contents/Home
Figure bleu (78.53982 m)
78.53982

Héritage

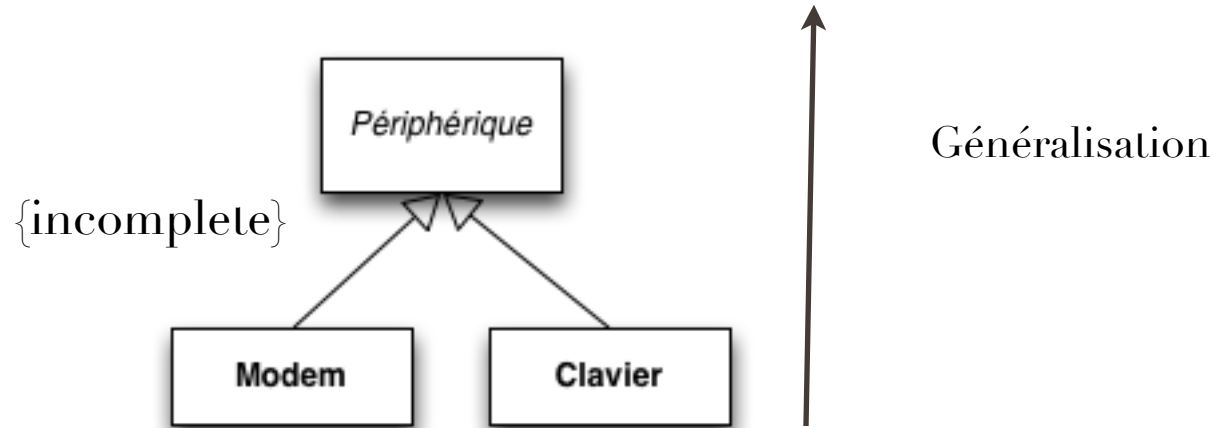


Oeuvre est en italique afin d'indiquer que c'est une classe abstraite

Faire exercice 3

Correction Exercice 3

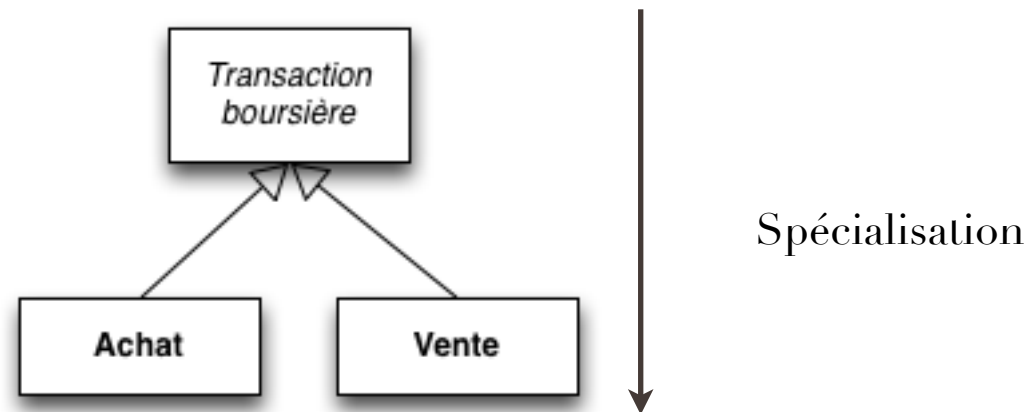
Les modems et les claviers sont des périphériques
d'entrée/sortie



La super classe Périphérique est *abstraite*. Elle ne s'instancie pas directement mais toujours par l'intermédiaire d'une des sous classes. L'arbre de généralisation est incomplet : il y a de nombreux autres périphériques d'entrée/sortie : écran, les souris, etc.

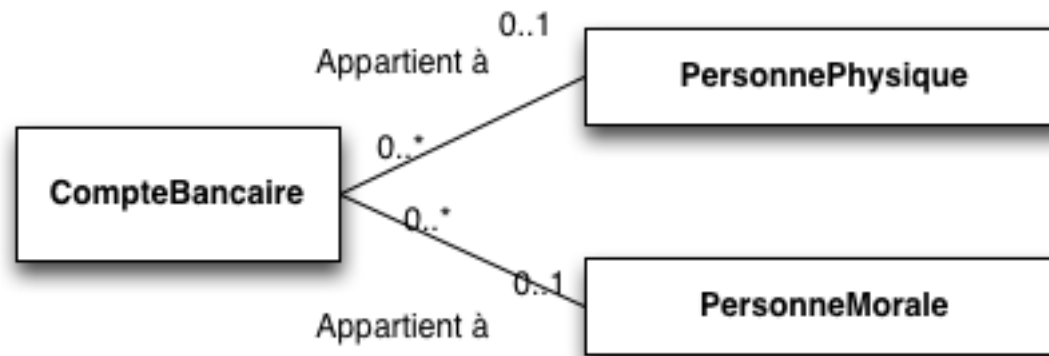
Correction Exercice 3

Une transaction boursière est un
achat ou une vente



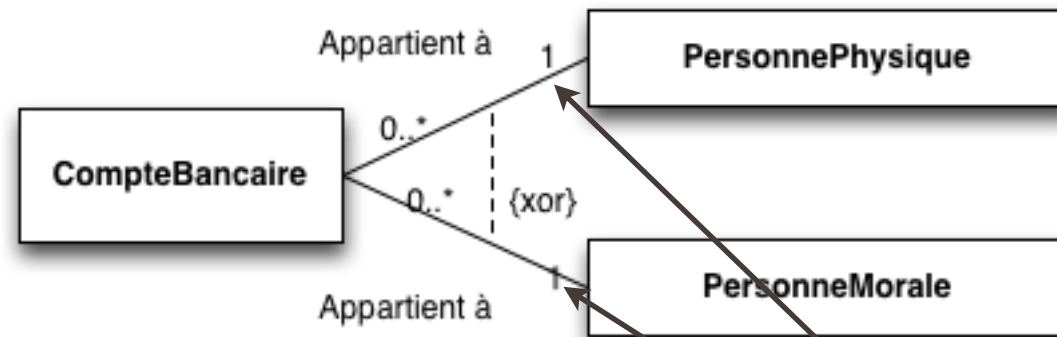
La super classe Transaction boursière est *abstraite*.
Elle ne s'instancie pas directement mais toujours par l'intermédiaire
d'une des sous classes

Correction Exercice 3



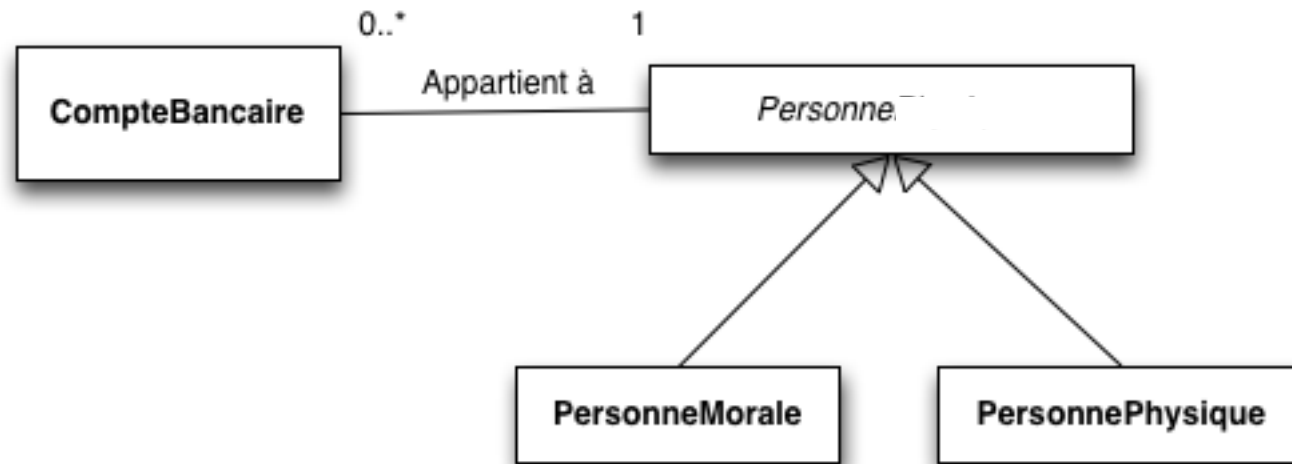
Cette solution ne rend pas
compte du côté exclusif !

Correction Exercice 3



Première bonne solution.
Regardez la multiplicité devient 1

Correction exercice 3



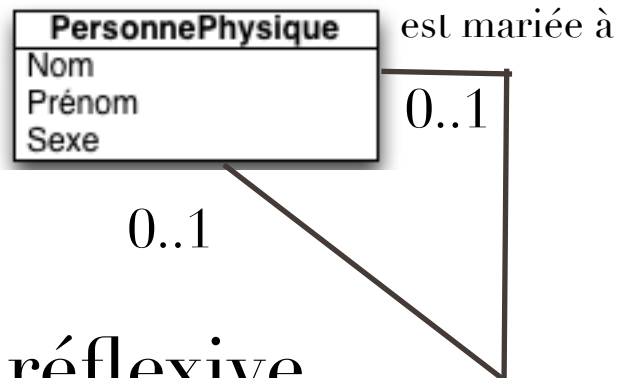
Seconde bonne solution.

Correction exercice 4

Deux personnes peuvent être mariées.

D'après le droit français :

Une personne n'est pas tenue d'être mariée
mais ne peut pas être mariée avec plusieurs personnes à la fois !



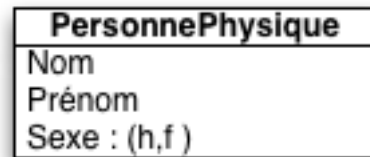
Association réflexive

Correction exercice 4

Deux personnes peuvent être mariées.

De sexe opposé

Contrainte



est mariée à

0..1

0..1

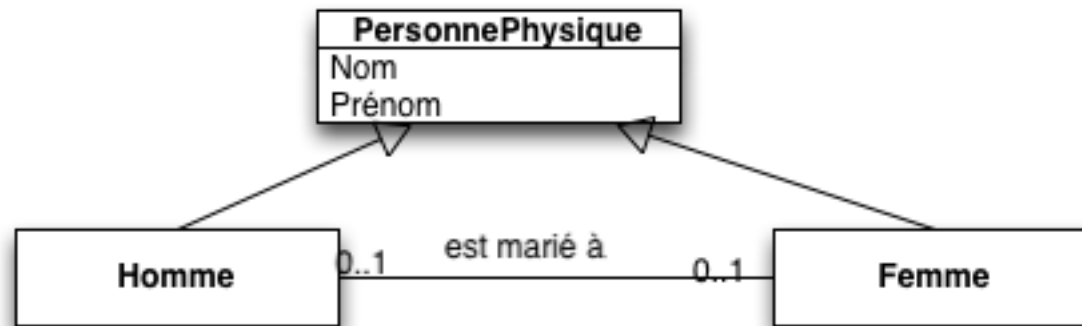
{deux personnes de même sexe
ne peuvent pas être mariées}

Correction exercice 4

Deux personnes peuvent être mariées.

De sexe opposé

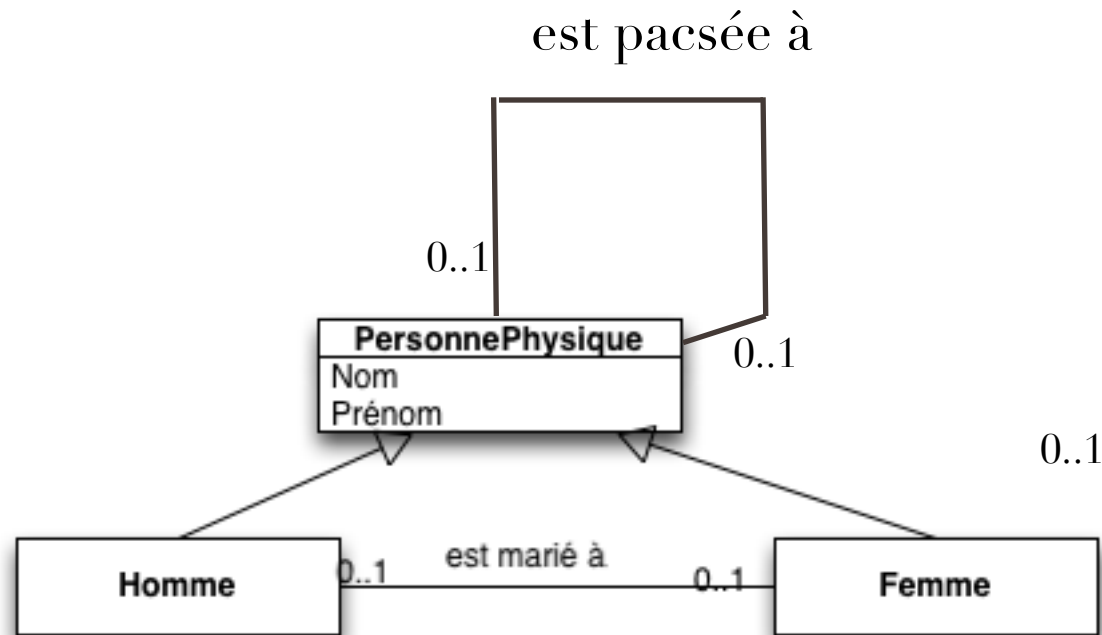
5°



Correction exercice 4

On peut être Pacsé

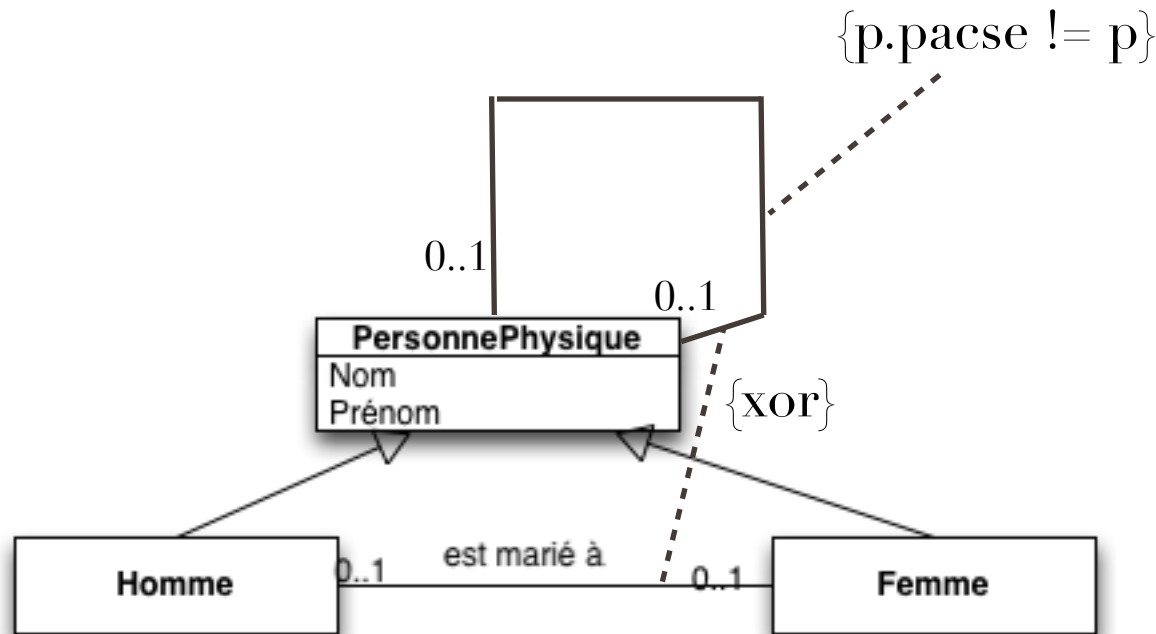
6°



Correction exercice 4

On ne peut pas être pacsé avec soi-même
et être marié et pacsé

7°

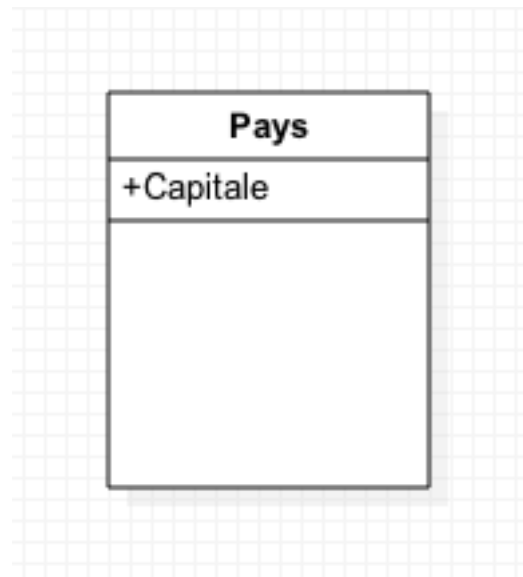


Subjectivité

- ~ La modélisation est hautement subjective !
- ~ On doit faire le choix entre simplicité et évolutivité
- ~ Exemple : Faire exercice 4

Exercice 4

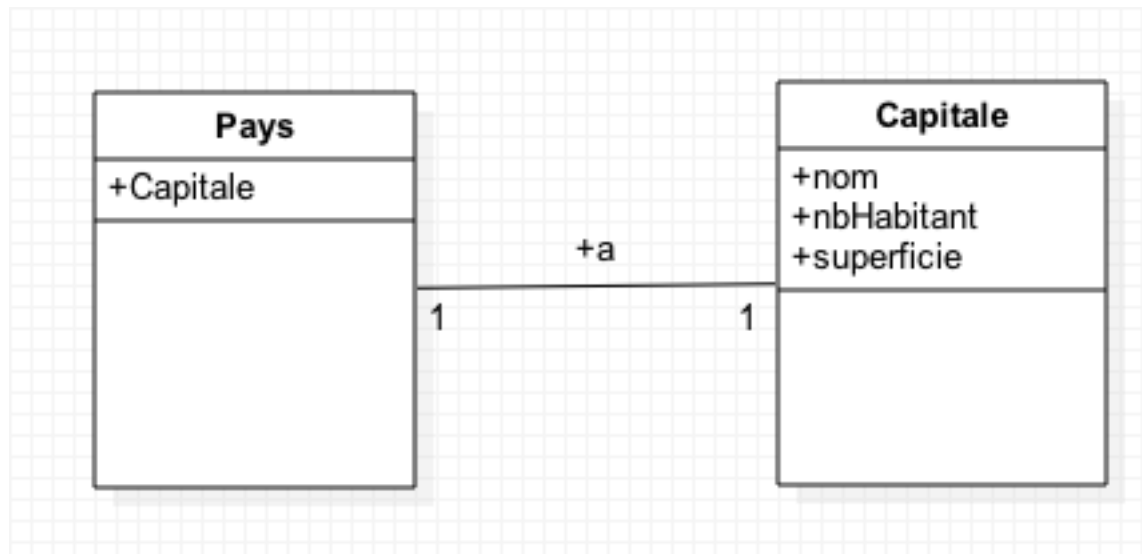
~ Un pays a une capitale



Prb si l'on veut ajouter des propriétés
au concept de capitale : nbr habitant, superficie, etc

Exercice 4

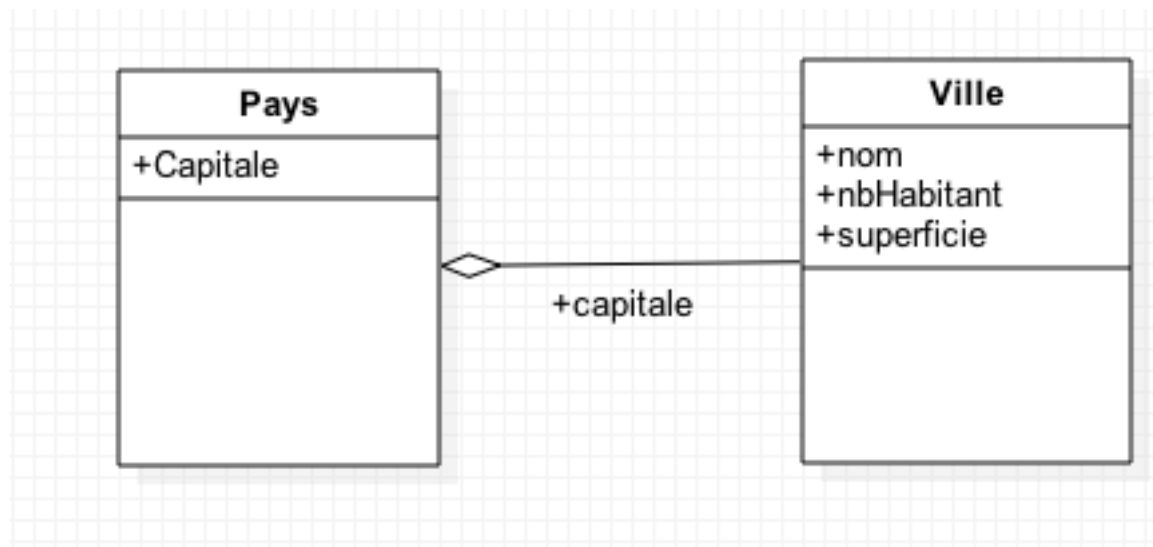
~ Un pays a une capitale



Est-ce une agrégation ? voire une composition ?
La destruction d'un pays entraine la destruction de la capitale ?

Exercice 4

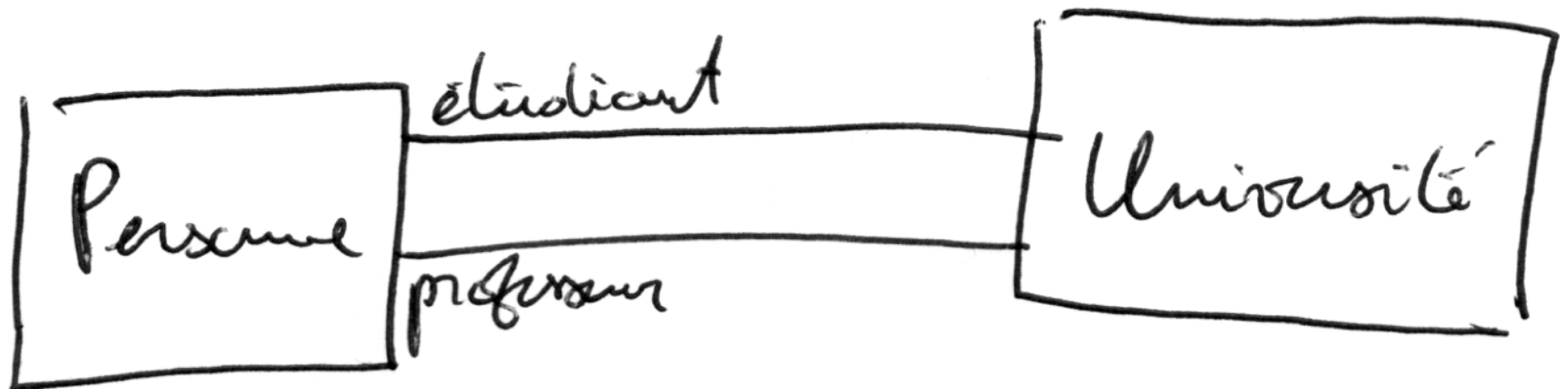
~ Un pays a une capitale



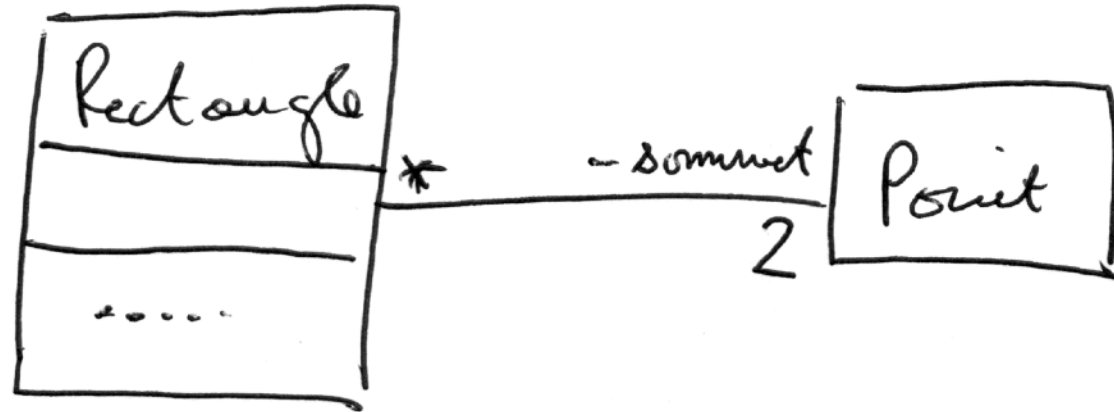
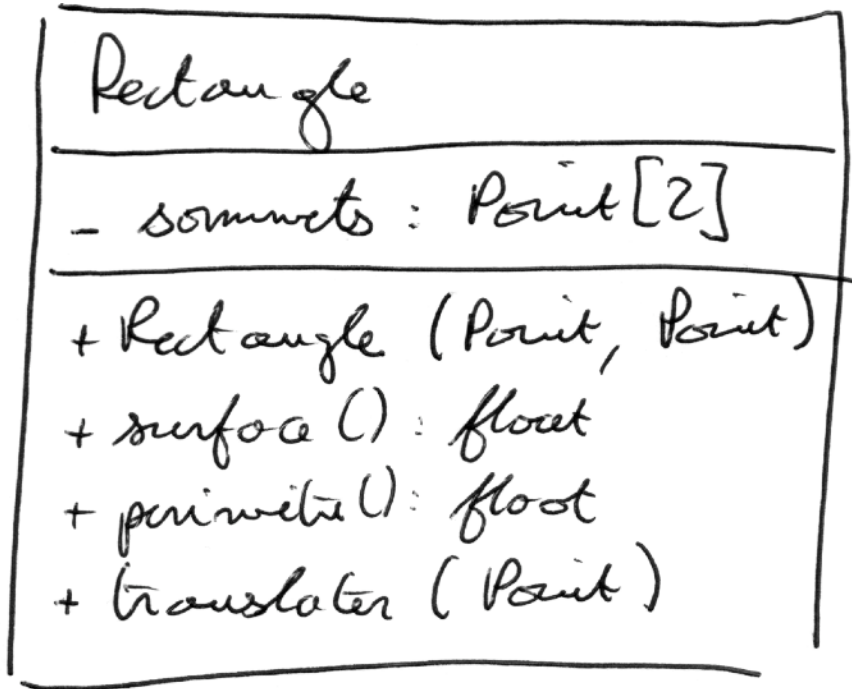
Exercise 5



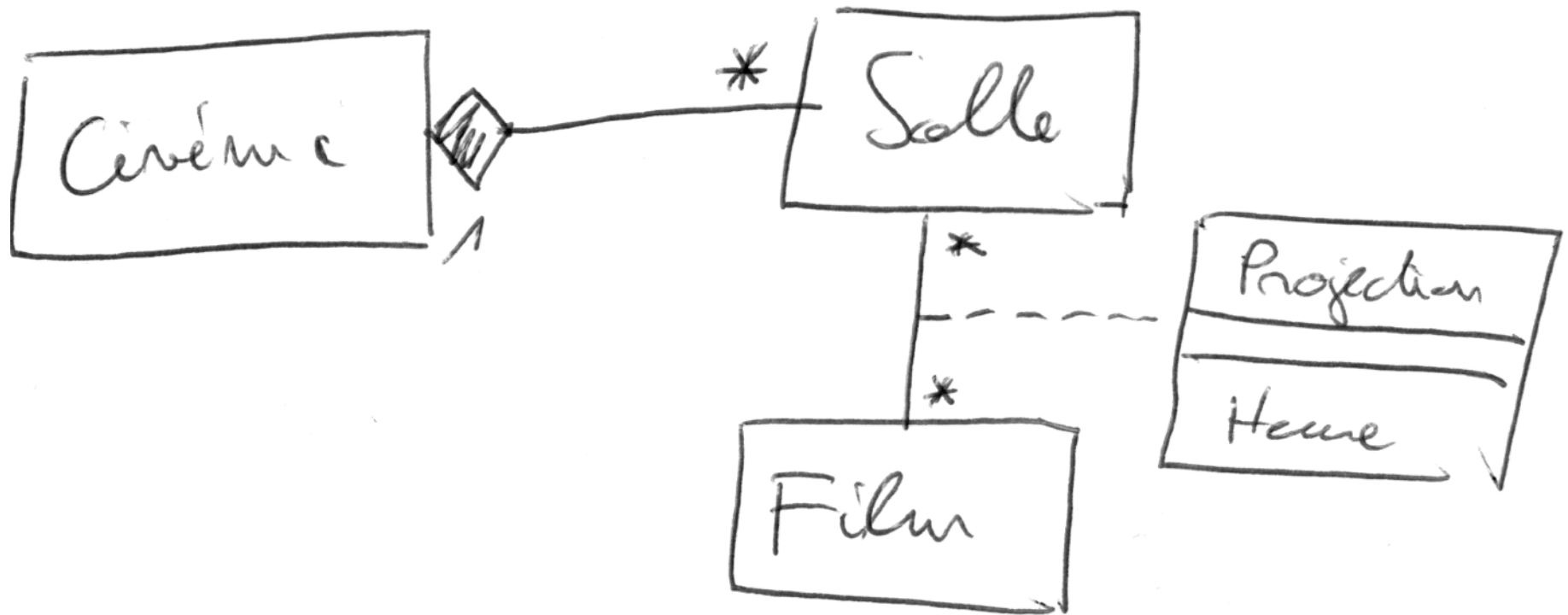
Exercice 5



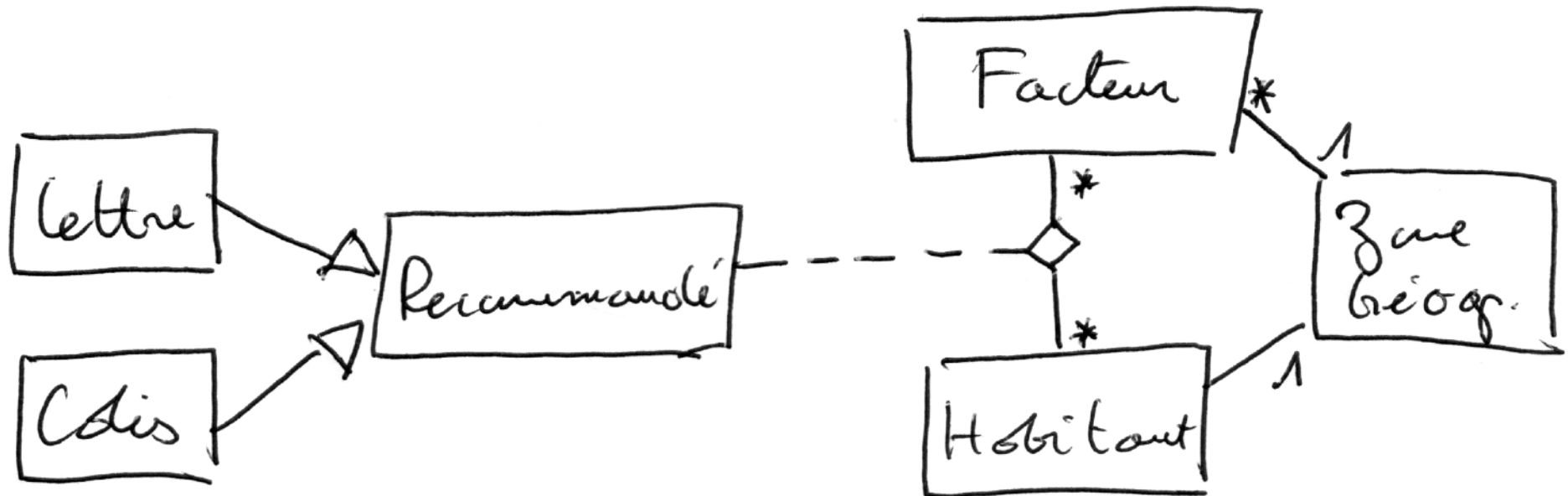
Exercise 5



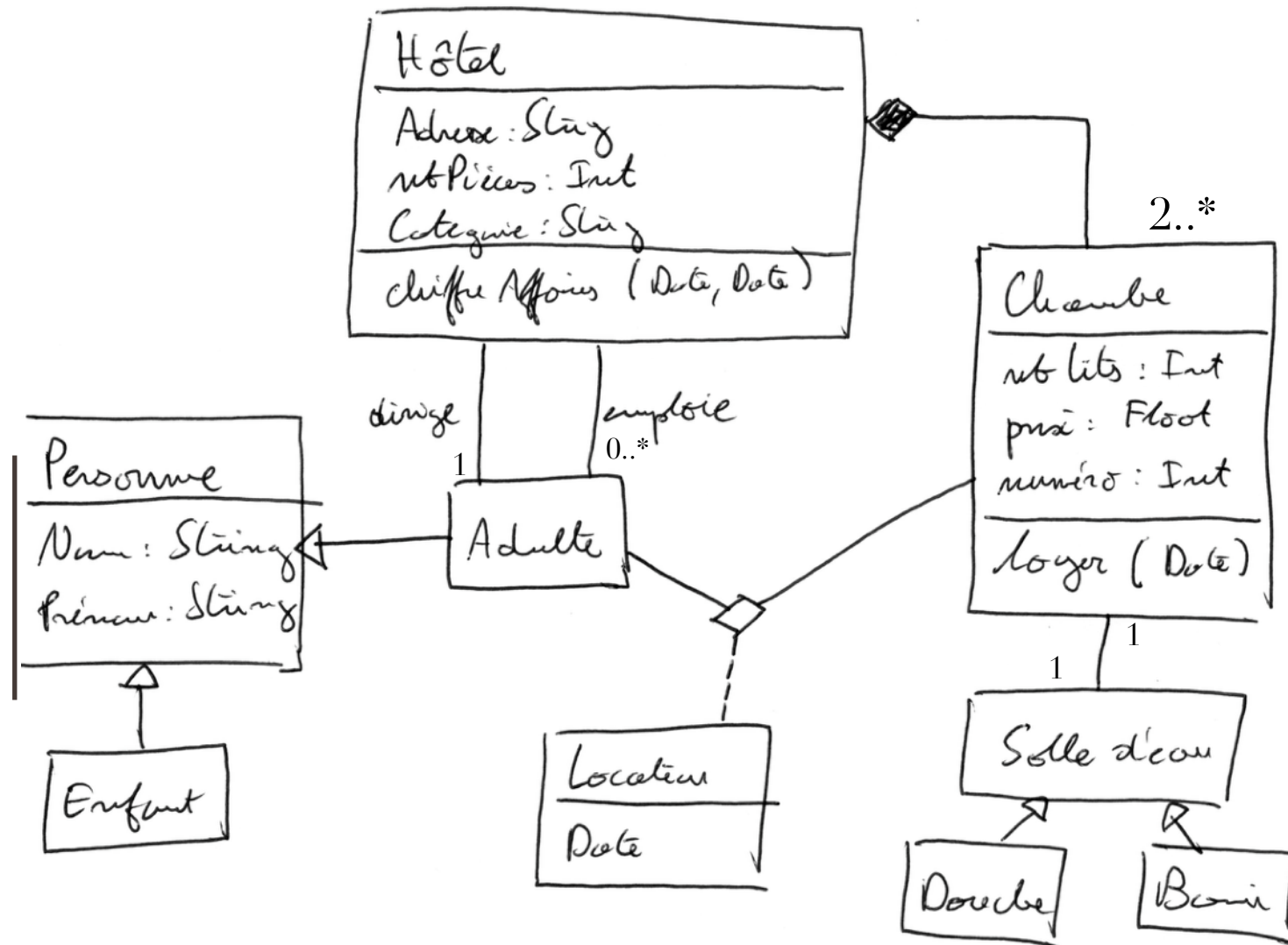
Exercise 5



Exercice 5



Exercice 6



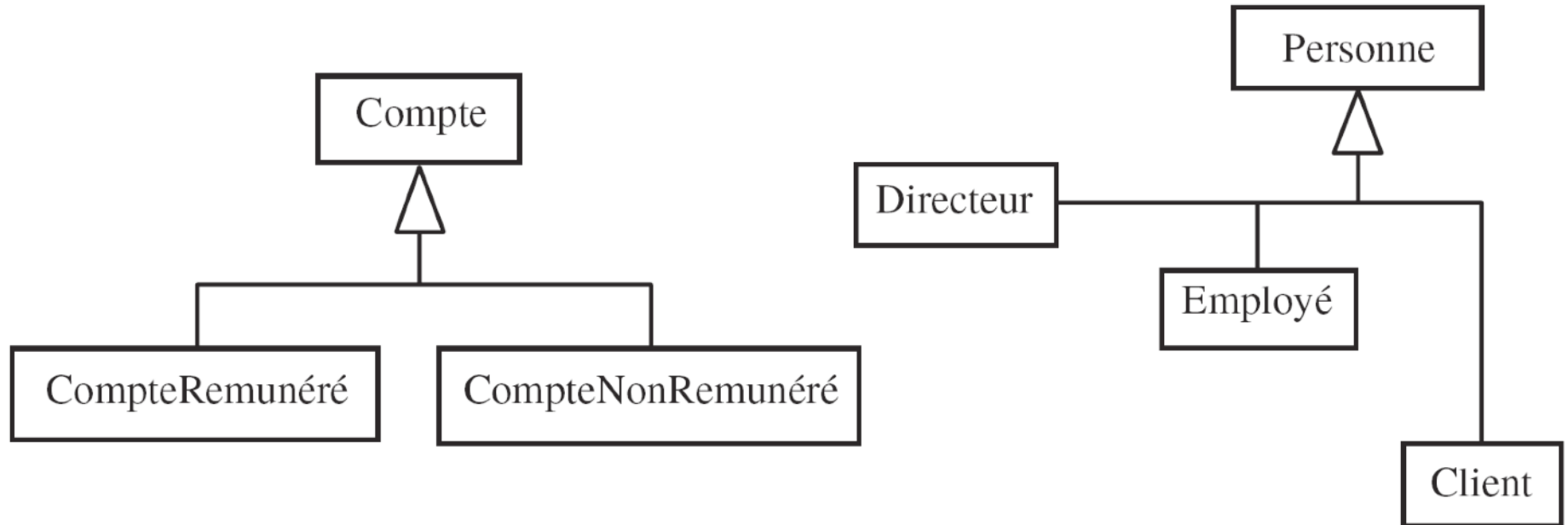
Exercice 7

Trouver les classes avec toutes les propriétés et les méthodes

Banque -nomDirecteur : String -capital : int -adresseSiege : String +getNomDirecteur() : String +setNomDirecteur(String n) +getCapital():int +setCapital(int capital) +getAdresseSiege():String +setAdresseSiege(String s) Banque(String Adresse)	Directeur -nom : String -prenom : String -revenu : float +getNom() : String +setNom (String n) +getPrenom ():String +setPrenom(String p) +getRevenu():float +setRevenu(float s)	Employé -nom : String -prenom : String -dateEmbauche : Date +getNom: String +setNom(String n) +getPrenom():String +setPrenom(String s) +getDate(): Date +setDate(Date s) mutation(Agence g): boolean	Client -nom : String -prenom : String -adresse : String -conseiller : Employer -agence : Agence -comptes : [1..N] Compte +getNom: String +setNom(String n) +getPrenom():String +setPrenom(String s) +getDate(): Date +setDate(Dates) mutation(Agence g):boolean
CompteNonRémunéré -solde : float -numero : int ...	Agence -nomAgence :String -adresseAgence : String +getNomAgence() : String +setNomAgence(String n)	CompteRémunéré -solde : float -numero : int -taux : float ... verserInteret() : void	

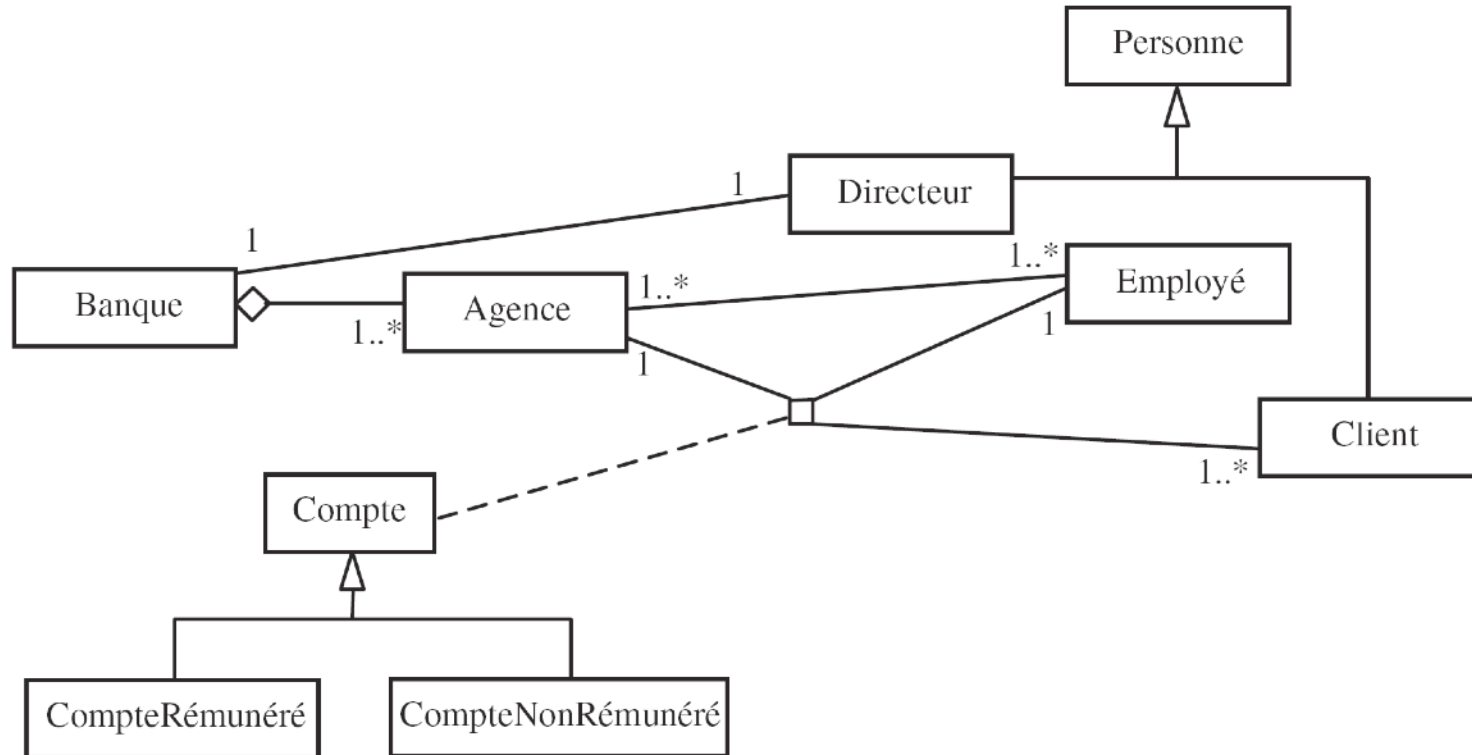
Exercice 7

Utilisez la généralisation pour factoriser au mieux la description des propriétés



Exercice 7

Donner le diagramme de classes



Bilan

Diagramme de classe

- ~ permet de représenter les différentes classes de l'application avec les attributs et méthodes
- ~ Sert aussi pour la BDD
- ~ préciser les relations entre elles
- ~ indiquer la visibilité des propriétés
(+, -, #)

Diagramme de classe

- ~ Les méthodes sont présentées avec leur signature : nom, paramètres, type de retour (peut importe le langage

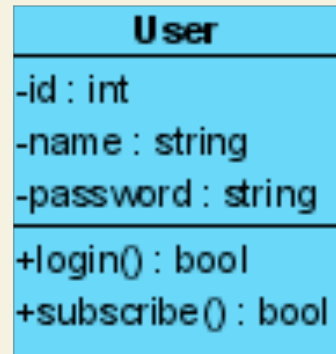


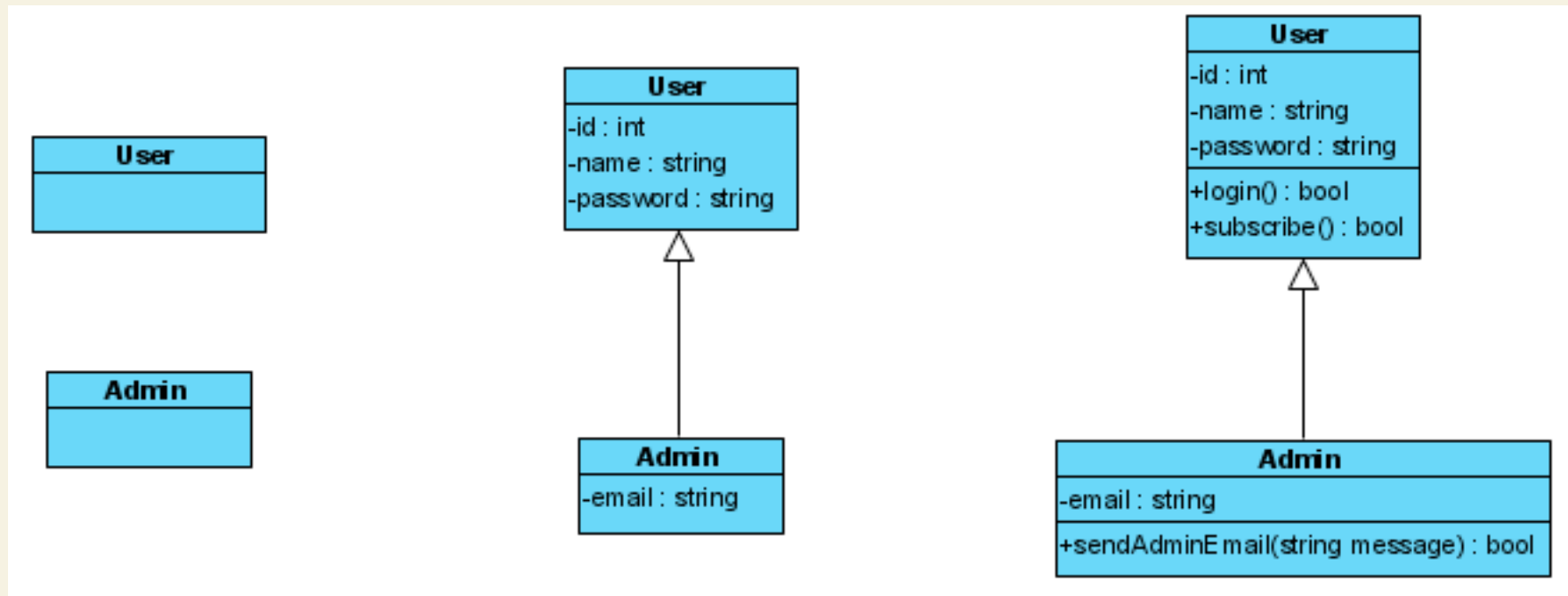
Diagramme évolutif

- ~ Il y a les spécifications fonctionnelles (ex : on veut ajouter des mots clefs aux photos)
- ~ Il y a des spécifications techniques (ex : je vais ajouter une propriété tableau qui va contenir les mots clefs de l'image et les méthodes nécessaires)
- ~ Dans un vie d'un projet ce n'est pas linéaire, cad qui n'y a pas des spécifications fonctionnelles puis des spécifications techniques. On a plus une boucle où le fonctionnel influence le technique.

Diagramme évolutif

- ~ le diagramme de classes va donc se construire sur plusieurs itérations.
- ~ It1: juste repérer les classes
It2: trouver les propriétés
It3: trouver les méthodes
it4: ajouter des propriétés
it5: ajouter des méthodes
etc.

Diagramme évolutif



Un exemple

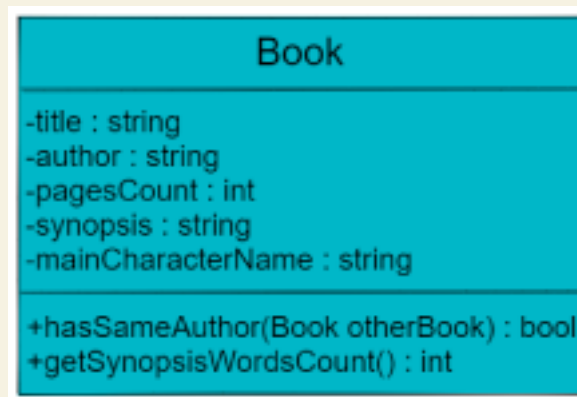
Une bibliothèque souhaiterait avoir un outil informatique pour recenser ses livres et pouvoir les rechercher selon divers critères. Les informations importantes d'un livre sont le titre, l'auteur, le nombre de pages, le synopsis et le nom du personnage principal.

En plus de la recherche, la bibliothèque souhaiterait également que l'outil lui permette de comparer un livre à un autre pour savoir s'ils sont du même auteur ou non. À des fins de statistiques, elle souhaiterait aussi pouvoir compter le nombre de mots d'un synopsis.

Un exemple

Une bibliothèque souhaiterait avoir un outil informatique pour recenser ses livres et pouvoir les rechercher selon divers critères. Les informations importantes d'un livre sont le titre, l'auteur, le nombre de pages, le synopsis et le nom du personnage principal.

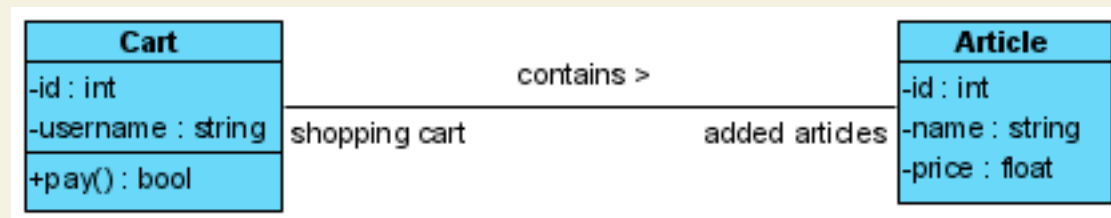
En plus de la recherche, la bibliothèque souhaiterait également que l'outil lui permette de comparer un livre à un autre pour savoir s'ils sont du même auteur ou non. À des fins de statistiques, elle souhaiterait aussi pouvoir compter le nombre de mots d'un synopsis.



Les relations entre les classes

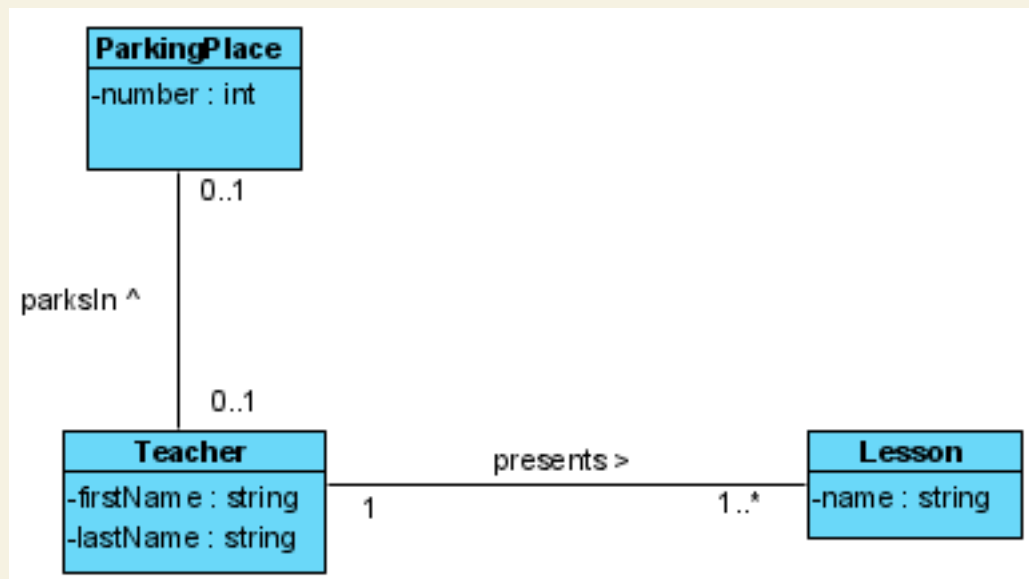
Association

Les associations sont représentées par un **trait plein** et peuvent posséder un nom qui permet de définir la sémantique de l'association. Une **flèche** peut parfois indiquer le sens de lecture de l'association.



Multiplicité

Les multiplicités permettent de définir la quantité d'éléments de l'association.



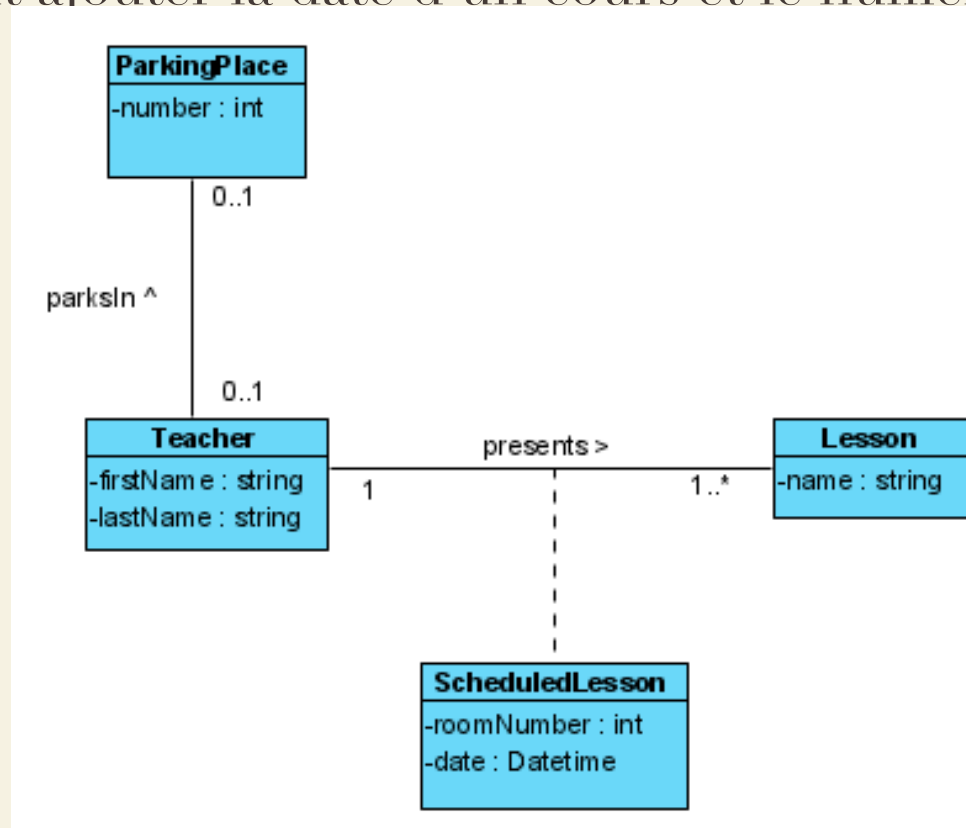
Multiplicité dans le code

- ✱ Chaque association devient une propriété
- ✱ Si on a 1 en multiplicité alors elle est non nullable. la classe Lesson aura une propriété Teacher non NULL
- ✱ Si on a 0..1 c'est nullable
- ✱ 1..* : on aura un tableau non vide
- ✱ 0..* : un tableau qui peut être vide

Ajouter des informations à une association

Lorsqu'on doit ajouter des informations à une association on a recours à une classe d'association. Elle n'a pas de multiplicité car il y aura autant d'instance que d'association.

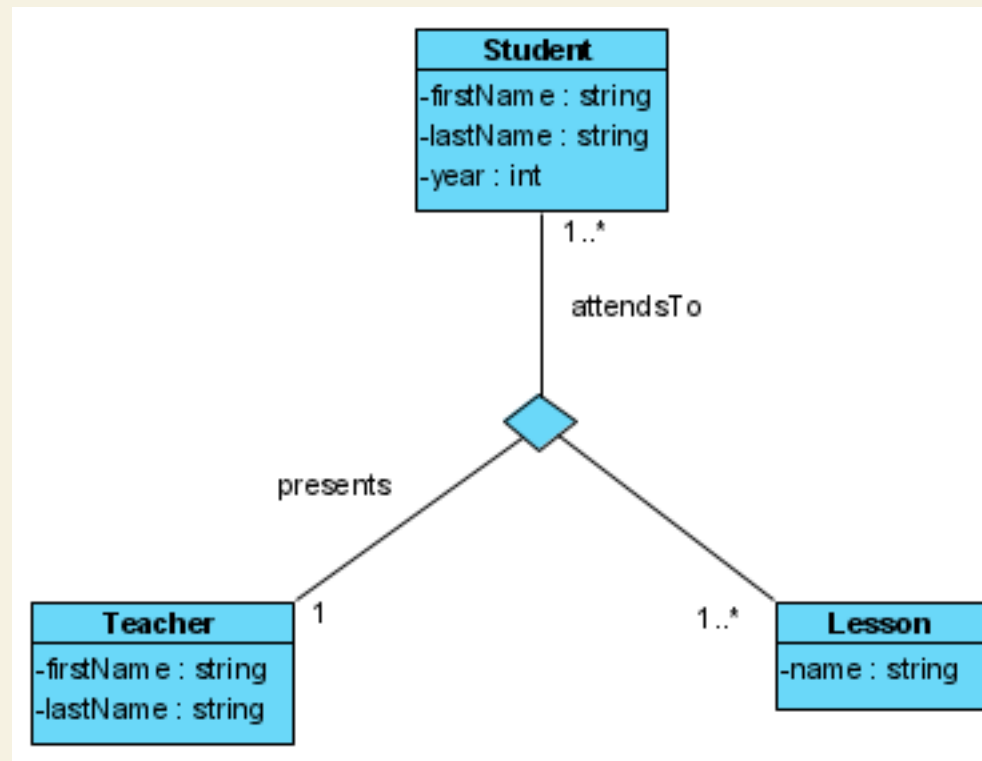
Si on veut ajouter la date d'un cours et le numéro de salle.



Association n-aire

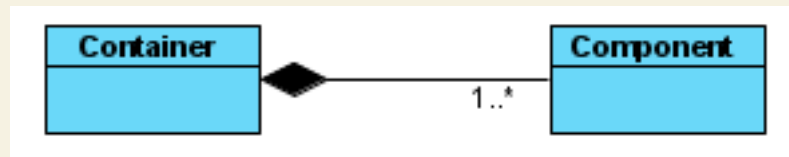
Quand une association doit relier plus de 2 classes on a recours au associations n-aire.

Cette association multiple est représentée par un losange



Type d'association

Il y a l'association simple mais aussi la composition. La **composition** est une association particulière qui permet d'indiquer une dépendance forte entre deux objets et de lier leurs durées de vie



Type d'association

Il y a l'association simple mais aussi l'agrégation. L'agrégation permet de définir une dépendance faible entre deux classes : si une classe A agrège une classe B, alors la classe A a besoin de la classe B pour fonctionner, mais chaque instance possède sa propre durée de vie.

