Associations UML et leur implémentation en Java

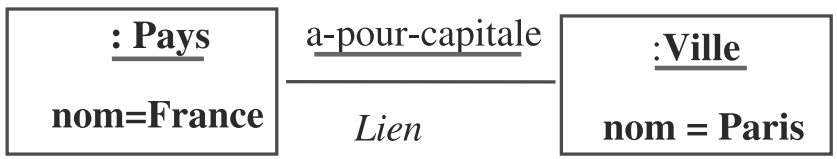
LIRMM / Université de Montpellier

15 février 2021

Sommaire

Définition

- une association est une relation entre 2 ou plusieurs classes qui décrit les connexions structurelles entre leurs instances
- une classe est un ensemble d'objets, une association est un ensemble de tuples
- $a\text{-}pour\text{-}capitale \subseteq Pays \times Ville$
- un lien relie des instances : $(france, paris) \in a\text{-}pour\text{-}capitale$



Représentation des associations

Association binaire = une ligne entre 2 classes éventuellement annotée par :

- le nom de l'association (orientation de lecture par le triangle noir),
- le nom des rôles aux extrémités de l'association, un rôle décrit la fonction de l'objet dans l'association
- la multiplicité des extrémités,
- la navigabilité (flèche au bout de la ligne).

Nom d'association

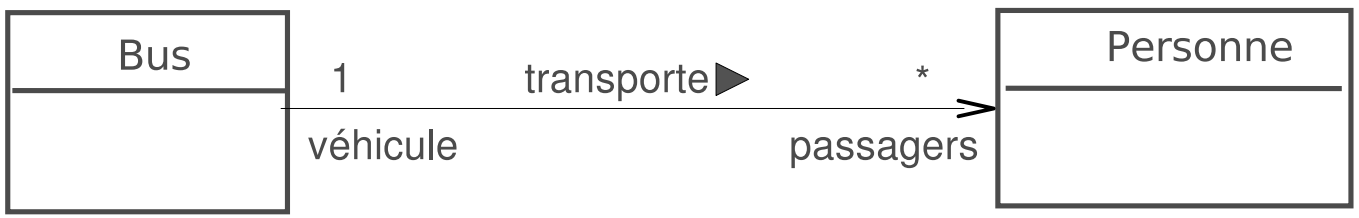
- le nom de l'association est suivi par un triangle noir
- l'association s'appelle `transporte`
- on lit "un bus transporte des personnes" et non l'inverse
- $transporte \subseteq Bus \times Personne$

Nom de rôle

- le nom de rôle décrit la fonction d'un objet dans une association
- le bus joue le rôle de `vehicule` dans l'association
- la personne joue le rôle de `passager` dans l'association
- dans une autre association, la personne pourrait jouer un autre rôle : conducteur, contrôleur, etc.
- noter la position du nom de rôle, contre la classe qu'il décrit : `passagers` contre `Personne`, `vehicule` contre `Bus`

Multiplicité

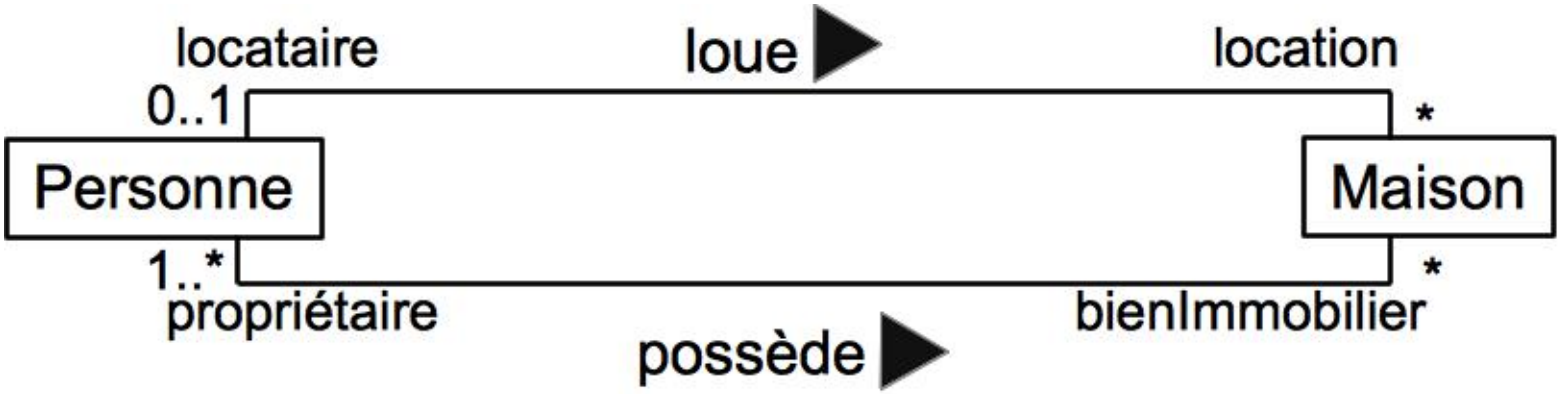
- Indique le nombre d'instances d'une extrémité qui peuvent être reliées à une instance de l'autre extrémité
- Technique : on fixe une instance d'un côté pour décider, en considérant les rôles
- Etant donné 1 véhicule, combien de passagers transporte-t-il ? **plusieurs**, d'où *****,
- Etant donné 1 passager, par combien de véhicules peut-il être transporté (à un moment donné) ? **un seul**, d'où **1**

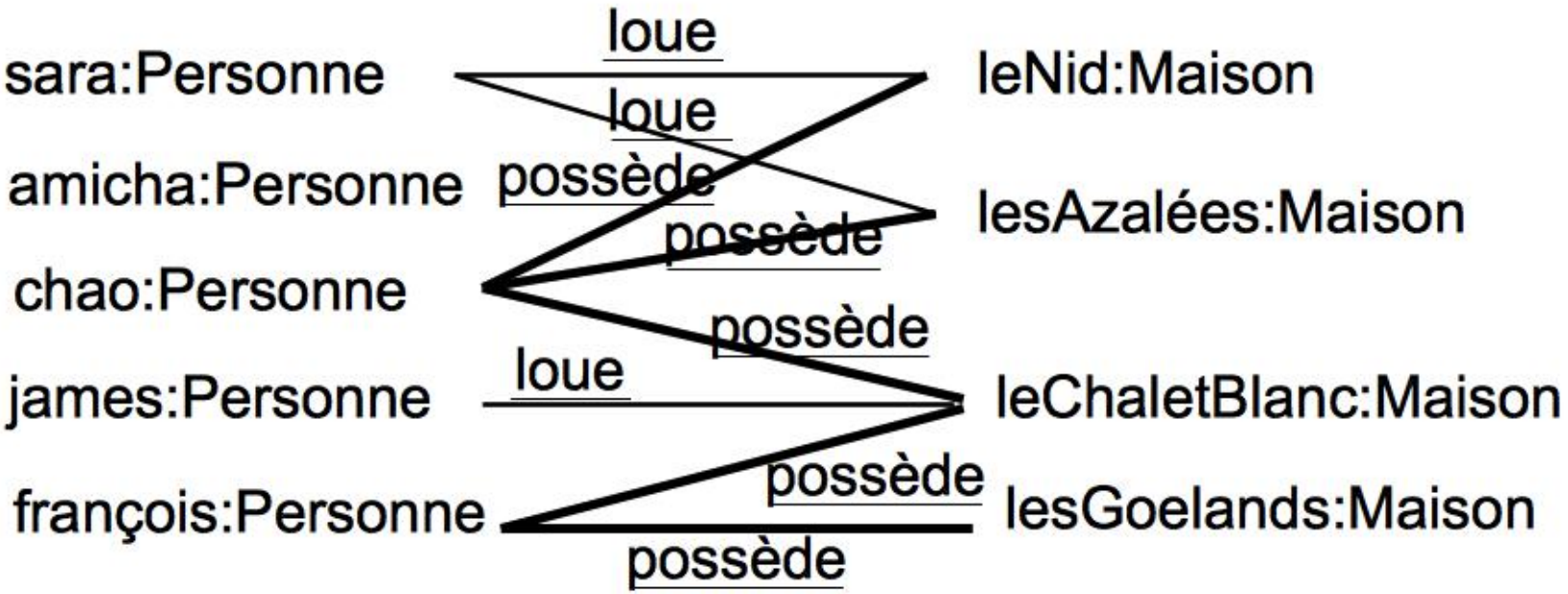


Navigabilité

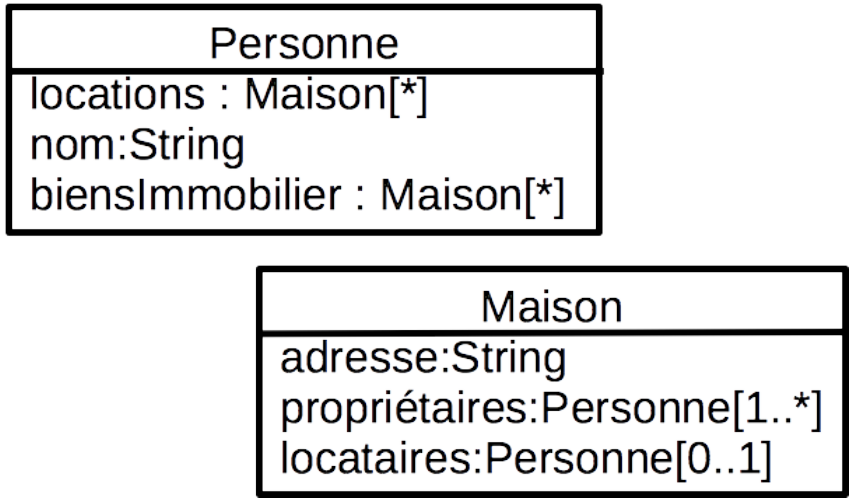
- Indique si un objet à une extrémité "connaît" les objets auxquels il est rattaché
- Pour prendre une décision au moment de la traduction en attributs dans le langage de programmation (qui n'a en général pas de structure pour traduire directement les associations)
- La représentation (classe) du bus contient la liste des passagers
- La représentation (classe) de la personne ne contient pas le véhicule dans lequel elle se déplace
- c'est toujours un choix à discuter

Autre exemple d'associations et de liens





Alternative avec des attributs



On y reviendra pour la traduction ... mais quel attribut est l'opposé de quel attribut ?

Associations et attributs

Intérêt des associations

- Lisibilité : on voit bien mieux les liens entre classes avec une association qu'avec des attributs ;
- Liens dépendants : avec une association, on définit 2 liens dépendants : les 2 extrémités de l'association.
- Associations complexes : on pourra définir des associations complexes, par exemple pour attacher des attributs aux liens.

Convention utilisée dans la suite de ce module

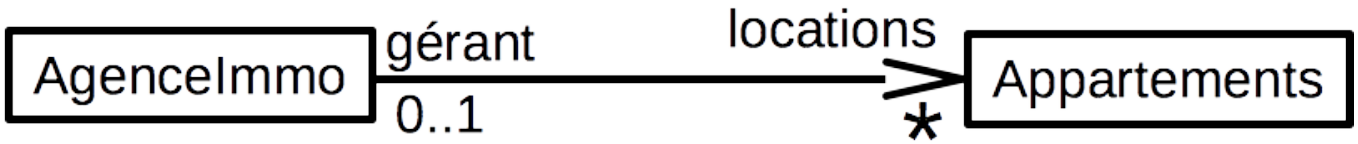
- Convention : les attributs seront uniquement de type simple (entiers, flottants, booléens, ...) ou des chaînes de caractères (String)
- Pas d'attribut de type complexe (classe), on préférera dans ce cas une association.

Sommaire

Comment traduire les associations en Java ?

- Par une classe, ou par 1 ou 2 attributs.
- On ne traduit par un attribut que les extrémités navigables.
- Si extrémité de cardinalité $\leq 1 \rightarrow$ attribut du type de l'extrémité.
- Si extrémité de cardinalité $> 1 \rightarrow$ attribut de type collection : liste, tableau, ensemble, ...
- Si association bidirectionnelle $\rightarrow$ attention à bien à faire les mises à jour des 2 côtés.

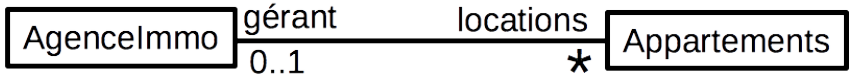
Traduction par 1 attribut (association unidirectionnelle)




```
public class AgenceImmo{
   private ArrayList<Appartement> locations;
   // collection d'appartements (voir plus loin)
}

public class Appartement{
   // rien concernant l'association
}
```

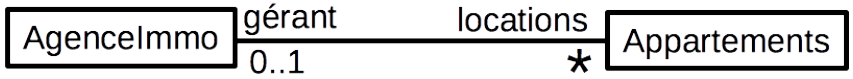
Traduction par 2 attributs (association bidirectionnelle)




```
public class AgenceImmo{
   private ArrayList<Appartement> locations;
   // collection d'appartements
}

public class Appartement{
   private AgenceImmo gerant;
}
```

Traduction par 1 classe (réification)

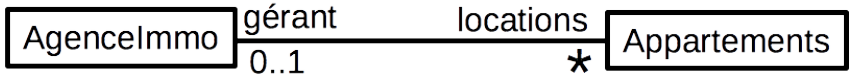



```
public class Location{
   private AgenceImmo gerant;
   private Appartement location;
   private Date dateDebut, dateFin;
}

public class AgenceImmo{
   // rien concernant Location
}

public class Appartement{
   // rien concernant Location
}
```

Traduction par 1 classe et liens inverses (réification)




```
public class Location{
   private AgenceImmo gerant;
   private Appartement location;
   ....
}

public class AgenceImmo{
   private ArrayList<Location> locations;
}

public class Appartement{
   private Location contratEnCours;
}
```

Collections

Groupe de valeurs d'un même type

- Tableaux primitifs (accès par un indice, taille fixe)
- Listes (accès par un indice, extensible)
- Ensembles (sans doublons)
- Dictionnaires associatifs (accès par une clef, extensible)

Le terme de `Collection` dans l'API Java est réservé pour certaines de ces structures.

Tableaux primitifs

```
// Déclaration
 Appartement[] tab;
 int[] tabEntiers;

 // Construction effective (avec une taille donnée)
  tab = new Appartement[10];
  tabEntiers = new int[4];
```

Tableaux primitifs

```
 // Construction avec des valeurs initiales
 tabEntiers = new int[]{11,13,17,19,23,29};

 // Initialisation par une itération
int[] tab = new int[10];
for(int i = 0; i < 10; i++)
      tab[i] = i;

// Création avec des valeurs initiales
int[] tab2={0,1,2,3,4,5,6,7,8,9};
```

Tableaux à plusieurs dimensions

```
int[][] tab;
tab = new int[N][M]; // N lignes, M colonnes

int[][] tab = {{1,2,3},{4,5,6},{7,8,9}};
```

Classes collections

- Classes qui définissent des structures de données regroupant plusieurs objets ; certaines sont des classes abstraites ; certaines sont même des interfaces
- Exemples : Pile (`Stack`), Liste chaînée (`LinkedList`), Liste tabulaire (`ArrayList`), etc.
- Des méthodes de manipulation les rendent plus facile d'usage que les tableaux primitifs

Les collections Java

Les collections sont génériques à plusieurs sens :

- Elles sont paramétrées par le type des éléments stockés (type non primitif).
- Des opérations sont définies pour toutes les collections.

La liste tabulaire (ArrayList)

- Collection qui stocke ses éléments dans un tableau mais qui est extensible

```
ArrayList<MonType> v; // déclaration
v=new ArrayList<MonType>(); // création
```

Déclaration et création en une seule ligne :

```
ArrayList<MonType> v=new ArrayList<>();
```

Méthodes classiques des listes tabulaires

Pour une ArrayList<E> :

- `void add(E obj)`. Ajoute l'objet `obj` à la fin de la liste.
- `boolean contains(E obj)`. Retourne vrai ssi `obj` est dans la liste.
- `E get(int index)`. Retourne l'objet placé en position `index` dans la liste.
- `boolean isEmpty()`. Retourne vrai ssi la liste n'a aucun élément.
- `int size()`. Retourne la taille de la liste.
- `E remove(int index)`. Supprime l'objet en position `index` et le retourne.
- `boolean remove(Object o)`. Supprime la première occurrence de o rencontrée (laisse la liste inchangée si l'objet o n'est rencontré, et retourne alors faux).

L'agence immobilière - version 1

```
 // L'association est unidirectionnelle
// l'appartement ne connaît pas son gérant
public class AgenceImmo{
  private String nom;

  private ArrayList<Appartement> apptGeres // "locations"
                      = new  ArrayList<>();

  public AgenceImmo(){}

  public void ajoutApptGere(Appartement appt){
  if (! apptGeres.contains(appt))
         apptGeres.add(appt);
  }
}
```

L'agence immobilière - version 2

```
// si l'appartement connaît quelle agence le loue
public class Appartement{
   private AgenceImmo gerant;
   ...
}
// on observe des modifications dans les deux objets liés
public class AgenceImmo{
  ...
  public void ajoutApptGere(Appartement appt){
  if (! apptGeres.contains(appt))
         {
          apptGeres.add(appt);
          appt.setGerant(this);
         }
  }
}
```

L'agence immobilière

Un exemple d'itération sur la liste tabulaire

```
public class AgenceImmo{

    public Appartement recherche(String adresse){
            for (int i=0; i < apptGeres.size(); i++)
                {
                    if (apptGeres.get(i).getAdresse()
                                        .equals(adresse))
                        return apptGeres.get(i);
                }
             return null;
    }
}
```

L'agence immobilière

Le même exemple avec une autre forme d'itération

```
public class AgenceImmo{

    public Appartement recherche(String adresse){
            for (Appartement a : apptGeres)
                {
                    if (a.getAdresse().equals(adresse))
                        return a;
                }
             return null;
    }
}
```

à suivre ...

- Associations complexes
- Dictionnaires associatifs