Listes, énumérations, répétitives

Le cas d'une association de cardinalités 1-n (1 à plusieurs)

Faculté des sciences, Université de Montpellier
Module HAI717I - Programmation

Points étudiés dans ce cours

Notions

- Enumérations (collections fixes de valeurs)
- Association 1-n (1 à plusieurs, 1-* en UML) et navigable dans une direction (unidirectionnelle)
- Listes (collections de valeurs)
- Structure de contrôle : répétitives, itérations

Cas d'étude

Une agence immobilière qui gère des appartements

Une classe Appartement

Description (structure)

- adresse
- superficie
- nombre de pièces
- année de construction
- classification

Classification des appartements

- les classes sont connues à l'avance, de nom et de nombre fixé à l'avance
- T1, T2, T3, F1, F2, F3, studio, duplex, loft, souplex

Une classe Appartement

Description (structure)

- adresse : chaîne de caractères
- superficie : réel
- nombre de pièces : entier
- année de construction : entier
- *classification*

```
public class Appartement {
   private String adresse;
   private double superficie;
   private int nbPieces;
   private int anneeConstruction;
   ....
}
```

Une classe Appartement

Description (structure)

- *adresse*
- *superficie*
- *nombre de pièces*
- *année de construction*
- classification

Classification des appartements

- les classes sont connues à l'avance, de nom et de nombre fixé à l'avance
- T1, T2, T3, F1, F2, F3, studio, duplex, loft, souplex

Enumération (type énuméré)

Classification des appartements

- les classes sont connues à l'avance, de nom et de nombre fixé à l'avance
- T1, T2, T3, F1, F2, F3, studio, duplex, loft, souplex

Type énuméré

- Il se définit par la liste de ses propres valeurs littérales
- Il possède nativement plusieurs opérations pour faciliter son utilisation
- Une variable de ce type prend obligatoirement l'une des valeurs littérales

Enumération (type énuméré)

Type énuméré

- Il se définit par la liste de ses propres valeurs littérales
- Il possède nativement plusieurs opérations pour faciliter son utilisation
- Une variable de ce type prend obligatoirement l'une des valeurs littérales,

```
public enum Classification {
      T1, T2, T3, F1, F2, F3, studio, duplex, loft, souplex
}
```

À Noter. La syntaxe est celle des identificateurs : ex. une valeur ne peut pas commencer par un chiffre, ne peut contenir d'espace

Une classe Appartement

```
public enum Classification {
      T1, T2, T3, F1, F2, F3, studio, duplex, loft, souplex
}

public class Appartement {
   private String adresse = "adresse inconnue";
   private double superficie;
   private int nbPieces;
   private int anneeConstruction;

   private Classification  classif = Classification.T1;
   ...
}
```

Notez la manière de faire référence à une valeur du type énuméré :
`Classification.T1`

Un constructeur pour la classe Appartement

```
public class Appartement {
   private String adresse = "adresse inconnue";
   private double superficie;
   private int nbPieces;
   private int anneeConstruction;

   private Classification  classif = Classification.T1;

   public Appartement(String adresse, double superficie, int nbPieces,
                      int anneeConstruction, Classification classif)
      this.setAdresse(adresse);
      this.setSuperficie(superficie);
      this.setNbPieces(nbPieces);
      this.setAnneeConstruction(anneeConstruction);
      this.setClassif(classif);
   }
   ...
}
```

Les accesseurs pour l'attribut classif

```
public class Appartement {
   private String adresse = "adresse inconnue";
   private double superficie;
   private int nbPieces;
   private int anneeConstruction;

   private Classification  classif Classification.T1;

    public Classification getClassif() {
         return this.classif;
   }

   public void setClassif(Classification classif) {
         this.classif = classif;
   }
}
```

À Noter. Les valeurs de `classif` ne peuvent pas être hors de la liste énoncée dans l'énumération `Classification`, donc pas de contrôle nécessaire dans `setClassif`.

La méthode toString

```
public class Appartement {
   private String adresse = "adresse inconnue";
   private double superficie;
   private int nbPieces;
   private int anneeConstruction;

   private Classification  classif = Classification.T1;

   public String toString() {
         return this.adresse+" "+this.superficie
               +" "+this.nbPieces+" "+this.anneeConstruction
               +" "+this.classif;
   }

}
```

À Noter. La valeur de `classif` se transforme en String sans effort de notre part car les énumérations sont dotées d'une méthode `toString`.

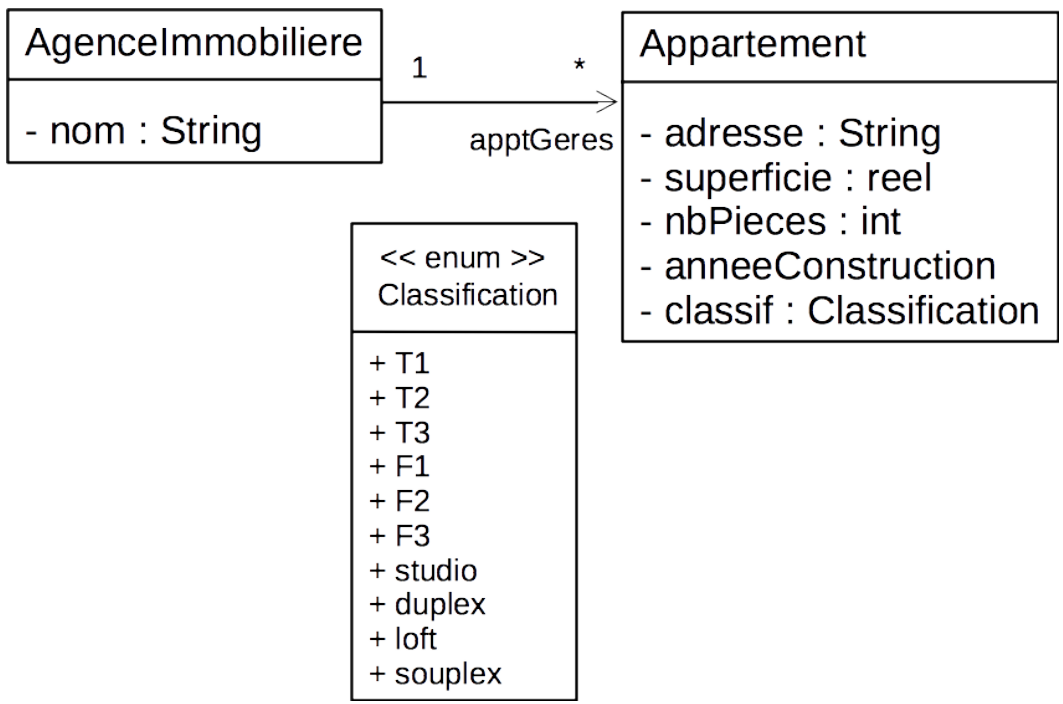
La méthode saisie

```
public class Appartement {
   private String adresse = "adresse inconnue";
   private double superficie;
   private int nbPieces;
   private int anneeConstruction;

   private Classification  classif = Classification.T1;

   public void saisie(Scanner clav) {
      System.out.println("adresse ?");
          this.adresse = clav.next();
      System.out.println("superficie ?");
          this.superficie = clav.nextDouble();
      System.out.println("nombre de pièces ?");
          this.nbPieces = clav.nextInt();
      System.out.println("année de construction ?");
          this.anneeConstruction = clav.nextInt();
      System.out.println("classification ?");
          this.classif = Classification.valueOf(clav.next());
   }
}
```

À Noter. `valueOf` change une chaîne de caractères en valeur du type énuméré (il ne faut pas se tromper lors de la saisie, sinon cela signale une erreur)

Association unidirectionnelle 1-n (1-*)

- Une agence immobilière est connectée à plusieurs appartements qu'elle gère
- Un appartement est géré par une unique agence immobilière mais on ne se préoccupera pas de ce lien inverse
- navigabilité : de AgenceImmobiliere vers Appartement



Liste des appartements

Cas d'étude

Une agence immobilière est connectée à plusieurs appartements qu'elle gère

Cas d'étude

- Une liste peut se représenter en Java par plusieurs classes
- Ici on choisit la classe ArrayList
- Une liste est une succession de valeurs :
  - toutes de même type
  - ordonnées
  - indexées par leur numéro de rang dans la succession
  - extensible
- Dans notre contexte, le type des valeurs peut être une classe
  Si on désire une liste de valeurs d'un type primitif (int, double, ...), on devra utiliser une classe enveloppe (Integer, Double, ...)
- Nous n'étudions pas les tableaux primitifs dans ce cours (ceux-ci ne sont pas extensibles)

Classe AgenceImmobiliere

```
import java.util.ArrayList;

public class AgenceImmobiliere {
    // Attribut représentant le nom : rien de nouveau !
    private String nom = "";

    // Attribut représentant la liste des appartements gérés
    // Vous observez ici la manière dont on DECLARE une liste.
    // On indique le type ArrayList
    // et le type des éléments stockés dans la liste,

    private ArrayList<Appartement> apptGeres;
    ....
```

Syntaxe

```
private ArrayList<Appartement> apptGeres;
```

ArrayList : Type de liste utilisé
Appartement : Type des éléments stockés dans la liste
< > : Parenthèsage du type des éléments
apptGeres : Nom de la variable

Créer la liste : solution 1

Création lors de la déclaration

```
import java.util.ArrayList;

public class AgenceImmobiliere {

    // Attribut représentant le nom : rien de nouveau !
    private String nom = "";

    // Attribut représentant la liste des appartements gérés
    private ArrayList<Appartement> apptGeres = new ArrayList<>();

    public AgenceImmobiliere() {
    }

    public AgenceImmobiliere(String nom) {
        this.nom = nom;
    }
    ...
}
```

Créer la liste : solution 1 ; écriture alternative

Création lors de la déclaration

```
import java.util.ArrayList;

public class AgenceImmobiliere {

    // Attribut représentant le nom : rien de nouveau !
    private String nom = "";

    // Attribut représentant la liste des appartements gérés
    private ArrayList<Appartement> apptGeres = new ArrayList<Appartement>();

    public AgenceImmobiliere() {
    }

    public AgenceImmobiliere(String nom) {
        this.nom = nom;
    }
    ...
}
```

Créer la liste : solution 2

Création dans un ou plusieurs constructeurs

```
import java.util.ArrayList;

public class AgenceImmobiliere {

    // Attribut représentant le nom : rien de nouveau !
    private String nom = "";

    // Attribut représentant la liste des appartements gérés
    private ArrayList<Appartement> apptGeres;

    public AgenceImmobiliere() {
       this.apptGeres = new ArrayList<>();
    }

    public AgenceImmobiliere(String nom) {
        this.nom = nom;
        this.apptGeres = new ArrayList<>();
   }
    ...
}
```

Connaître le nombre d'éléments

Méthode de la classe `ArrayList<T>` int size()
Pour la liste `maListe` maListe.size()

Exemple d'utilisation

```
/* Méthode de AgenceImmobiliere
 * Connaître le nombre d'appartements gérés
 * Illustration de la méthode size
 */

public int nbAppartGeres() {
   return this.apptGeres.size();
}
```

Connaître l'élément de rang i

Méthode de la classe `ArrayList<T>` T get(int i)
Pour la liste `maListe` maListe.get(i)
Les indices vont de 0 à maListe.size()-1

Exemple d'utilisation

```
/*
 * Connaître l'appartement de rang i
 * Illustration de la méthode get
 */

public Appartement apptRang(int i) {
   if (i>=0 && i <this.apptGeres.size())
      return this.apptGeres.get(i);
   else
      return null;
}
```

Appartenance d'un élément

Méthode de la classe `ArrayList<T>` boolean contains(T elem)
Pour la liste `maListe` maListe.contains(elem)

Exemple d'utilisation

```
/*
 * Tester le fait que l'agence gère un certain appartement
 * Illustration de la méthode contains
 */

public boolean gere(Appartement appt) {
   return this.apptGeres.contains(appt);
}
```

Ajout d'un élément

Méthode de la classe `ArrayList` boolean add(T nouvelElement)
Pour la liste `maListe` maListe.add(t)

Exemple d'utilisation

```
/*
 * Ajouter un nouvel appartement
 * s'il y est déjà : afficher un message d'erreur
 * sinon : l'ajouter effectivement
 * Illustration des méthodes contains et add
 */

public void ajoute(Appartement appt) {
    if (this.apptGeres.contains(appt))
        System.out.println("appartement déjà présent");
    else
        this.apptGeres.add(appt);
}
```

Effectuer un traitement sur les éléments

4 formes d'itérations sont disponibles en Java

- for avec un itérateur, pour tout parcourir
- for avec un indice, pour parcourir tout ou partie en connaissant le rang de l'élément
- while avec une condition
- do ... while avec une condition

Effectuer un traitement sur les éléments

for avec un itérateur, pour tout parcourir

L'itérateur est une variable qui prend successivement comme valeur chaque élément de la liste

Exemple d'utilisation, l'itérateur est la variable `appart`

```
public void afficheAdresses() {
   for (Appartement appart : this.apptGeres) {
         System.out.println("adresse :  "+appart.getAdresse());
   }
}
```

`appart` va être successivement le 1er, le 2e, le 3e ... appartement de la liste `apptGeres`
`appart.getAdresse()` correspond à l'application de la méthode `getAdresse()` successivement à chaque appartement

Effectuer un traitement sur les éléments

for avec un indice, pour parcourir jusqu'à un certain rang

L'indice est une variable de type entier qui prend successivement comme valeur des indices en restant entre 0 et `size()-1`

Exemple d'utilisation, l'indice est la variable `i`, qui varie de 0 à `this.nbAppartGeres()-1`.

`this.nbAppartGeres()` est égal à `size()`

```
public void afficheAdresses() {
   for (int i = 0; i < this.nbAppartGeres(); i++) {
      System.out.println("adresse appt. de rang = "+i+" "
         +this.apptGeres.get(i).getAdresse());
   }
}
```

`this.apptGeres.get(i)` va être successivement le 1er, le 2e, le 3e ... appartement de la liste apptGeres

Effectuer un traitement sur les éléments

while avec une condition d'arrêt

Exemple d'utilisation, la condition d'arrêt est que la variable `i`, qui a été initialisée à 0, atteint `this.nbAppartGeres()`.

`this.nbAppartGeres()` est égal à `size()-1`

```
public void afficheAdresses() {
      int i = 0;
      while(i < this.nbAppartGeres()){
            System.out.println("adresse appt. de rang = "+i+" "
                  +this.apptGeres.get(i).getAdresse());
            i++;
      }
}
```

`i < this.nbAppartGeres()` est la condition d'arrêt
`this.apptGeres.get(i)` va être successivement le 1er, le 2e, le 3e ... appartement de la liste apptGeres

Effectuer un traitement sur tous les éléments

do while avec une condition d'arrêt

Exemple d'utilisation, la condition d'arrêt est que la variable `i`, qui a été initialisée à 0, atteint `this.nbAppartGeres()`.

Le do-while effectue au moins une fois le corps de l'itération avant d'effectuer le test "i < this.nbAppartGeres()".
Il faut donc se prémunir du cas où la liste est vide.

```
public void afficheAdresses() {
    if (this.apptGeres.isEmpty())
        System.out.println("pas d'adresse à afficher");
    else {
        int i = 0;
        do {
            System.out.println("adresse appt. de rang = "+i+" "
                    +this.apptGeres.get(i).getAdresse());
            i++;
        }
        while (i < this.nbAppartGeres());
    }
}
```

Synthèse

- Types énumérés `enum`
- Notions à retenir pour les `ArrayList`
  - Déclaration `ArrayList<TypeElement> liste`
  - Création `new ArrayList<TypeElement>();`
  - Ajout `add(TypeElement t)`
  - Test d'appartenance `contains(TypeElement t)`
  - Accès à l'élément de rang i `get(int i)`
  - taille `size()`
- Répétitives `for/itérateur`, `for/indice`, `while`, `do while`