Chapitre 8

Les concepts de base des collections

L'objectif de ce cours est de découvrir et utiliser les concepts de base de la collection ArrayList

1. Introduction

Comme nous avons pu le voir précédemment, il est très important, en informatique, de stocker des informations de même nature et de manipuler, rechercher, recréer de tels structures d'informations.

On a souvent besoin de ordonner des éléments, rechercher, ajouter ou supprimer un éléments, insérer un nouvel élément, …

Pour cela, nous avons utilisé les tableaux qui ont une contrainte forte : connaître à priori le nombre d'élément à gérer et donc dimensionner le tableau à sa création. Avec la contrainte de gérer également un entier permettant de savoir à tout instant le nombre d'éléments utiles.

Pour pallier à cette contrainte, une **collection** est une abstraction qui généralise la notion de tableau en cachant l'implémentation de la collection.

En JAVA, il existe de nombreuses collections qui sont des classes d'objet. Chacune des classes a des propriétés différentes et des propriétés communes.

A ce stade du discours, il nous faut donc parler des différentes structures de données qui existent dans le monde de l'informatique :

- les listes
- les files
- les piles
- les ensembles
- les tables de hashing
- les arbres

Il est également difficile de faire abstraction de l'implémentation d'une collection quand il s'agit de rendre performant un programme informatique.

Car chaque type de collection fait un choix d'implémentation qui a ses avantages et ses désavantages.

1.1. Documentation et API de Java

1.1.1. Où trouver la documentation ?

Sur le site officiel de java.

Sur mon site dans la page des exercices (version doc java 1.5)

1.1.2. Structure de la documentation

La documentation est structuré en packages puis en classes.

On peut accéder à une classe par son nom ou y accéder par son appartenance à un package.

Un package peut contenir d'autres packages.

La difficulté est de savoir quelle classe cherchée.

Souvent le plus simple est de rechercher sur internet un besoin. Dans la plupart des cas on trouve facilement.

Les packages les plus utilisés et qui correspondent à nos besoins dans le cadre de la formation sont :

java.lang	classes de base du langage Java
java.util	classes utilitaires
java.io	classes des entrées sorties (écran, clavier, fichier, répertoire)

java.awt	classes de base pour faire les IHM
javax.swing	classes de base pour faire les IHM (plus haut niveau que awt)

1.1.3. L'utilisation des API

On accède aux API java : **docs jdk 1.5\api\index.html**

La documentation d'un package est structuré en :
- les interfaces
- les classes
  - les champs accessibles
  - les constructeurs
  - les méthodes (static et non static)
- les énumérations
- les exceptions
- les erreurs
- les notes

Les classes importantes :
java.lang.Integer
java.lang.Double
java.lang.Character
java.lang.Math
java.lang.String
java.lang.StringBuffer
java.lang.System
java.lang.Thread

java.util.ArrayList
java.util.Date mais plutôt java.util.Calendar
java.util.Hashtable
java.util.Properties
java.util.StringTokenizer
java.util.Vector

java.io.**BufferedReader**
java.io.DataInputStream
java.io.DataOutputStream
java.io.File
java.io.FileDescriptor
java.io.FileInputStream
java.io.FileOutputStream
java.io.InputStream
java.io.**InputStreamReader**
java.io.OutputStream

2. Les concepts des structures de données

2.1. La conception et l'utilisation des collections

- notion d'accès à une donnée
- recherche d'une donnée
- types de collections : liste (ou tableau), dictionnaire, ensemble, arbre (liste récursive)
- relation entre les collections

- copie ou partage des données
- clefs d'indexation des données entre les collections
- comparaison entre tableau et dictionnaire ➔ description graphique et UML des collections ➔ notion de classe d'association (lien entre deux collections avec des caractéristiques spécifiques : montrer les différentes implémentations)

2.2. Les listes

Une liste d'élément est une structure de données qui permet de ranger des éléments de même nature dans un ordre donné. On dit que les éléments sont rangés. Ils ont donc un **rang** (ou indice dans le vocabulaire "tableau")).

On accède à un élément en fonction de son rang.

On peut notamment :
- ajouter un élément à la fin de la liste
- insérer un élément à un rang donné. Les éléments qui suivent le rang d'insertion se décalent d'un rang.
- supprimer un élément à un rang donné. Les éléments qui suivent le rang d'insertion se décalent d'un rang.
- connaître le nombre d'élément

Une liste peut contenir plusieurs fois le même élément.

En fonction de l'implémentation, le rang est ou non un accès direct à l'élément :
- le tableau est un exemple d'accès direct
- le chaînage des éléments est un exemple d'un accès séquentiel

<donner en séance des exemples d'implémentation avec tableau et chaînage, …>

2.3. Les files

Une file est une liste dans laquelle on ne peut que ajouter en tète ou à la fin de la liste.
On supprime soit l'élément entête ou en fin.

La file est donc comme une liste mais avec une restriction sur les méthodes associées.

<faire un schéma en séance de représentation d'une file

2.4. Les piles

Une pile est une liste dans laquelle on ne peut qu'ajouter ou supprimer en fin de la liste.
La pile est donc comme une liste mais avec une restriction sur les méthodes associées. Nous voyons donc qu'une pile est un sous-ensemble fonctionnel d'une liste. Nous verrons que l'héritage objet est une réponse à l'implémentation avec une telle relation fonctionnelle. On dit que Pile hérite de Liste. Pour les ensembles c'est pareil.
Remarque : L'héritage est aussi une réponse à un sur-ensemble fonctionnel.

2.5. Les ensembles

Un ensemble est une collection qui ne peut pas contenir plusieurs fois le même élément. On peut parcourir les éléments d'un ensemble mais la notion de rang n'est pas définie. Cela veut dire que le parcours des éléments d'un ensemble n'est pas le même à tout instant.

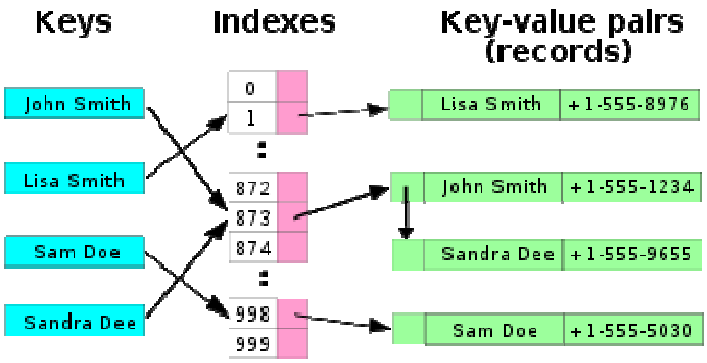
2.6. Les tables de hashing (ou achage)

Une table de hashing est une structure de données qui implémente l'association clef-valeur.

On accède à la valeur par sa clef. La clef est convertit en une valeur de hashing.

Dans une Hashtable la clef est une chaîne de caractère et la valeur n'importe quel objet.

Exemple d'implémentation par chaînage :



2.7. Les arbres

Un arbre est une collection assez à part car elle se définit de manière récursive.
Par définition, un arbre est un couple de deux valeurs :

- un nœud
- une liste d'arbre (qui peut être vide)

On démonter qu'un arbre n-aire se ramène à un arbre binaire.

Etant donné qu'un ArrayList peut contenir d'autres ArrayList alors la classe ArrayList est aussi un "Arbre".

Faire une démonstration en séance avec un schéma.

3. Les bases de la classe ArrayList

3.1. Présentation

La classe **ArrayList** permet de construire des tableaux de taille variable.

Cette classe est dans le package **: java.util**

Ne pas oublier de mettre en début de votre fichier source la ligne :

import java.util.*;

De la même manière qu'un tableau est un tableau d'int, de char, de String (etc.), une ArrayList contient des valeurs d'un type donné. On doit préciser ce type quand on déclare la variable. Pour cela, on fait suivre le nom de la *classe* ArrayList par le type des éléments, entre chevrons < et > :

ArrayList<E> où est le type des éléments de la liste

Exemple de création d'une liste de chaine :

```
ArrayList<String> liste = new ArrayList<String>();
```

3.2. Exemple (voir ExempleArrayList.java)

Il est obligatoire de définir un import du package util qui contient les classes de collection

```
import java.util.*;

public class ExempleArrayList
{
    public static void main(String[] args)
    {
        ArrayList<String> listeChaine;
        listeChaine = new ArrayList<String>();

        listeChaine.add("un");
        String str = new String("deux");
        listeChaine.add(str);
        listeChaine.add("trois");
        listeChaine.add("quatre");

        Terminal.ecrireStringln("--1-- Affichage de listeChaine");
        for(String s:listeChaine)
            Terminal.ecrireStringln(s);

        listeChaine.add(2,"toto");

        //listeChaine.add(20,"fin");            //      Exception:
IndexOutOfBoundsException

        Terminal.ecrireStringln("--2-- Affichage de listeChaine");
        for(String s:listeChaine)
            Terminal.ecrireStringln(s);

        listeChaine.addAll(listeChaine);
        Terminal.ecrireStringln("--3-- Affichage de listeChaine");
        for(int i=0;i<listeChaine.size();i++)
            Terminal.ecrireStringln(listeChaine.get(i));

        String str1 = listeChaine.get(3);
        Terminal.ecrireStringln("--4-- " + str1);


        Terminal.ecrireStringln("--5-- Iterator de listeChaine");
        Iterator<String> iter = listeChaine.iterator();
        while(iter.hasNext())
            Terminal.ecrireStringln(iter.next());


        Terminal.ecrireStringln("--6-- Liste d'objet quelconque");
        Bidule b = new Bidule(100);
        Truc t = new Truc("TRUC");
```

```
          ArrayList liste = new ArrayList();
          // Il n'y a pas de chevrons

          liste.add(b);
          liste.add(t);

          // Voici le 1er exemple d'un traitement générique
          for(Object obj:liste)
              Terminal.ecrireStringln(obj.toString());

          //TOUTE LES CLASSSES HERITENT DE OBJECT et la classe
OBJECT contient une méthode toString qui est surchargée par les
méthodes toString des classes Bidule et Truc.

      }
  }

  class Bidule
  {
      int x;
      public Bidule(int x){this.x=x;}
      public String toString()
      {
          return ""+x;
      }
  }

  class Truc
  {
      String  a;
      public Truc(String a){this.a = new String(a);}
      public String toString()
      {
          return ""+a;
      }
  }



  Exécution :
  java ExempleArrayList
  --1-- Affichage de listeChaine
  un
  deux
  trois
  quatre
  --2-- Affichage de listeChaine
  un
  deux
  toto
  trois
  quatre
  --3-- Affichage de listeChaine
  un
  deux
  toto
  trois
  quatre
  un
  deux
  toto
  trois
```

```
quatre
--4-- trois
--5-- Iterator de listeChaine
un
deux
toto
trois
quatre
un
deux
toto
trois
quatre
--6-- Liste d'objet quelconque
100
TRUC
```

3.3. Les méthodes de la classe

Dans les descriptions de méthodes suivantes E dénote le type de l'élément
Les méthodes de la classe de base sont :

boolean add(E e);
Ajouter un élément en fin de la liste (il n'y a pas de limite)

E get(int index);
Retourne l'élément se trouvant à l'indice *index*

E remove(int index);
Supprime l'élément se trouvant à l'indice *index*

E set(int index, E element);
Modifie la valeur de l'élément se trouvant à l'indice *index*

int size();
Retourne le nombre d'élément de la liste

E[ ] toArray();
Retourne la liste sous la forme d'un tableau.
La taille du tableau est égal au nombre d'élément de la liste

3.3.1. Remarques

Pour les méthodes qui ont en paramètre un indice, si cet indice est en dehors de l'intervalle [0; size()-1] alors la méthode retourne l'exception IndexOutOfBoundsException.

On peut utiliser la boucle for énumérative sur un ArrayList.

3.4. Exercice

3.4.1. Enoncé

1/ Créer la classe **ListeString** qui est une implémentation personnelle d'une collection de genre "liste" contenant des chaînes de caractère.

Le constructeur de la classe permet de créer une liste vide (0 éléments).
La classe implémente les méthodes suivantes :

public void add(String:s)
ajoute un nouveau élément dans la collection à la fin de ceux qui existent déjà

public int size()
retourne le nombre d'élément de la collection

public String get(int indice)
retourne l'élément se trouvant à l'indice donné de la collection (les indices vont de 0 à size() - 1
Si indice est <0 ou >= size() alors retourne null

public boolean set(int indice; String valeur)
modifie l'élément se trouvant à l'indice donné avec la valeur donnée et retourne true
Si indice est <0 ou >= size() alors ne fait rien et retourne false

public int indexOf(String occurrence)
retourne l'indice de la 1ère occurrence d'une chaine à partir de 0
si pas trouvé alors retourne -1

public int indexOf(int debut; String occurrence)
retourne l'indice de la 1ère occurrence d'une chaine à partir de l'indice debut
si pas trouvé ou indice début en dehors des éléments alors retourne -1

public boolean remove(int indice)
supprime l'élément se trouvant à l'indice donné (les éléments suivants ont donc leurs indices décalés et retourne true
Si indice est <0 ou >= size() alors ne fait rien et retourne false

Le choix de l'implémentation de cette classe utilise le tableau de String : String[]

2/ Faire un programme principal qui permet de tester les méthodes de la classe ListeString

3.4.2. Correction

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exercice15_ListeString**

4. La récursivité

4.1. Introduction

Définition :
La récursivité est la propriété d'un langage informatique d'appeler par un traitement, le traitement lui-même, et d'empiler les paramètres d'appel.

Le principe de récursivité "s'oppose" au principe itératif qui utilise des boucles de variables.

Principe :

Lors de l'appel d'un sous-programme, le programme empile au-dessus des données précédentes (données globales et locales du sous-programme appelant), les valeurs des paramètres d'appel puis réalise le sous-programme.

Si ce même sous-programme s'appelle lui-même, il empile à nouveau par-dessus les nouvelles valeurs des paramètres d'appel, et ainsi de suite pour chaque appel.

Puis, au retour de chaque appel, le programme dépile les paramètres concernés.

Le sous-programme doit contenir un test d'arrêt de récursivité.

4.2. Un traitement récursif : calcul de fibonacci

La série dite "de fibonacci" est définie de la manière suivante :
fibo de 0 = 0
fibo de 1 = 1
fibo de n = fibo de n-1 + fibo de n-2

On obtient la série suivante :
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

Voici le code de manière récursive :

```
static int fibo(int n)
{
    int n1,n2,v;
    if (n<=1) return n; // si n=1 alors 1, si n=0 alors 0

    n1 = fibo(n-1);
    n2 = fibo(n-2);
    v = n1 + n2;

    return (v);
}
```

Prenons l'exemple de l'appel :

```
int i,x;
i = 3;
x = fibo ( i );
```

4.3. Un autre exemple: le factoriel

```
static int fact(int n)
{
    if (n==0) return 1;
    else return n * fact(n-1);
}
```

On voit bien la puissance synthétique et descriptive d'une telle méthode pour décrire un traitement.

4.4. Les structures de données récursives

Un type T d'une structure de données est récursive si un de ses attributs est défini sur T.

Cela marche aussi s'il y a un type intermédiaire.

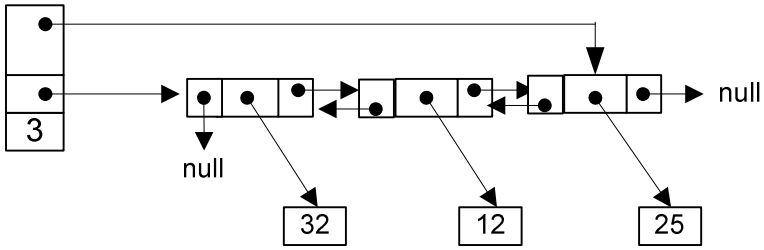
En programmation objet, on parle de classe récursive.

Dans cette définition sont englobées les structures de données définies par le système d'exploitation comme l'arborescence des répertoires qui sous-entend une structuration arborescence
et donc une définition récursive en interne des données du système d'exploitation.

4.4.1. Les listes chainées

Les listes chaînées sont des collections constituées de "cellules" chaînées les unes aux autres par un pointeur. Chaque cellule contient la valeur de la collection.

Un exemple de représentation avec double chainage :



Pour implémenter un tel objet, il nous faut définir une classe récursive :

```
class Cellule
{
    Cellule prec;
    Object  value;
    Cellule suiv;
}
```

Définition de la liste :

```
class Liste
{
    private int nb;
    private Cellule prem;
    private Cellule dern;

    public Liste()
    {
        nb=0;
        prem=null;
        dern=null;
    }

    public void add(Object value)
    {
        Cellule cel = new Cellule();
        cel.value = value;
        nb++;
        if (prem==null)
            {
                prem = cel;
                dern = cel;
            }
```

```
            else
                {
                    cel.prec=dern;
                    cel.suiv=null;
                    dern.suiv=cel;
                    dern=cel;
                }
        }

        public Object get(int rang)
        {
            Cellule p = getCellule(rang);
            if (p==null) return null;
            return(p.value);
        }

        public Cellule insert(Object value,int rang)
        {
            Cellule p = getCellule(rang);
            if (p==null) return null;

            Cellule cel = new Cellule();
            cel.value = value;
            cel.prec = p.prec;
            cel.suiv = p;
            if (p==prem) prem = cel;
            else p.prec.suiv = cel;
            p.prec = cel;
            nb++;
            return cel;

        }

        public void afficher()
        {
            Cellule p = prem;
            while(p!=null)
                {
                    System.out.print(p.value.toString()+" ");
                    p=p.suiv;
                }
            System.out.println();
        }


        private Cellule getCellule(int rang)
        {
            int n=0;
            Cellule p = prem;
            while(p!=null)
                {
                    if(n==rang) return p;
                    n++;
                    p=p.suiv;
                }
            return(null);
        }
```

Programme de test :

```
public class ListeLink
```

```
  {
      public static void main(String[] args)
      {
          Liste l = new Liste();
          l.add(new Integer(32));
          l.add(new Integer(12));
          l.add(new Integer(25));
          l.afficher();
          l.insert(new Integer(100),0);
          l.insert(new Integer(200),2);
          l.insert(new Integer(300),4);
          l.afficher();
          System.out.println(l.get(2));
          System.out.println(l.get(10));
      }
  }
  }
```

4.4.2. Les arbres binaires

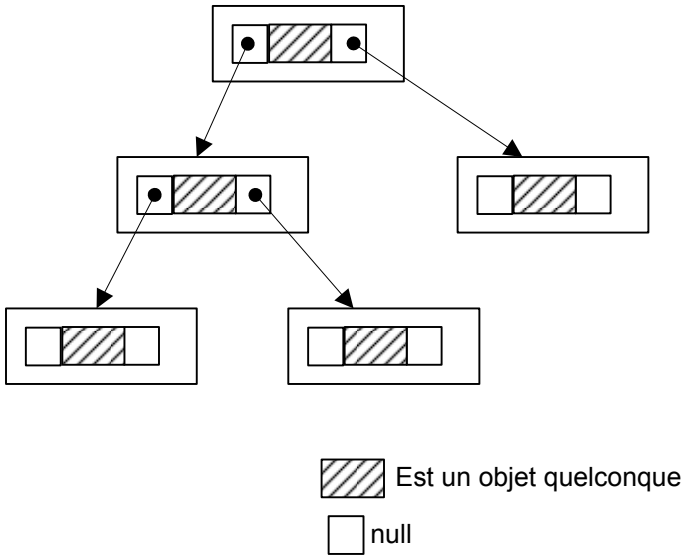
L'arbre est un concept mathématique qui est un cas particulier de la théorie des graphes.

Il ne s'agit pas ici de faire un cours sur les graphes mais de voir comment il est possible d'implémenter un arbre pour ensuite l'utiliser.

Un arbre est un nœud qui est caractérisé par 3 informations :

- la valeur du nœud qui est un objet quelconque. Cela permet d'y mettre n'importe quelle information qui peut être complexe ;
- la référence vers le sous-arbre gauche qui est donc un arbre ;
- la référence vers le sous-arbre droit qui est donc aussi un arbre ;

Les arbres sont donc des structures de données par définition récursives : un arbre est constitué de deux sous-arbres.




```
public class Arbre
{
   private Object value;
   private Arbre gauche;
   private Arbre droit;
}
```

On voit ici que la classe Arbre a deux attributs de type Arbre : la donnée est récursive.

Deuxième implémentation possible :

```
public class Arbre
{
    private ArrayList<Object> noeud;
}
```

Dans cette implémentation, il n'est pas visible que la classe Arbre contient d'autres Arbre.

Il faut comprendre que :

- nœud.get(0) est de type Object et contient la valeur du nœud
- nœud.get(1) est de type Object et contient un Arbre (un Arbre est un Object) : l'arbre gauche.
- nœud.get(2) est de type Object et contient un Arbre : l'arbre Droit.

Le choix de cette implémentation est donc d'utiliser la classe **ArrayList** qui est une donnée récursive et polymorphe car n'importe quel élément peut être à son tour un ArrayList.
La donnée ArrayList est une donnée récursive.

On peut donc créer des arbres (binaires ou n-aires) sans la nécessité de créer une classe.
Mais il est plus "objet" de créer une classe comme on va le voir dans l'exemple.

On peut donc créer des arbres sans la nécessité de créer une classe

Implémentation de la 1ère solution :

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple11_ArbreBinaire**

4.4.3. Les arbres n-aires

Un arbre n-aire est un nœud qui est caractérisé par 2 informations :

- la valeur du nœud qui est un objet quelconque. Cela permet d'y mettre n'importe quelle information qui peut être complexe ;
- une collection (tableau, ArrayList, ...) qui contient la liste des références vers les sous-arbres qui sont donc des arbres ;

```
public class Arbre
{
    private Object value;            // Valeur du noeud
    private ArrayList<Arbre> fils;   // Les sous-arbres
}
```

Voir sur le site http://jacques.laforgue.free.fr l'exemple **Exemple12_Arbre**

4.5. Utilisation des Arbres

Les arbres sont utilisés dans de très nombreux domaines : base de données, intelligence artificielle, réseau, système, scientifiques, ...

Par exemple :

- gestion des arborescences de répertoires et de fichiers;
- optimisation d'accès aux données en séquentielles indexées
- programme d'aide à la décision (jeux, simulation du raisonnement)
- programmation des réseaux sémantiques
- recherche opérationnel (graphe)

Au-delà des arbres, il y a aussi les graphes dont les arbres est un cas particulier.

5. Conclusion

En informatique les structures de données et leurs optimisations appartiennent à un domaine de l'informatique à part entière.

On dit que la base (le socle) d'un bon système d'information est la définition de ses données.

Cela est d'autant plus vrai, que dans une démarche de programmation orientée objet dans laquelle la définition des objets est la première chose à faire, il est important de concevoir les structures de données et leurs dépendances entre elles de telle manière que soit optimal :

- la compréhension de ces données
- le temps d'exécution des traitements d'accès, de mise à jour et de recherche,
- l'évolution de ces données à de nouveaux besoins
- le niveau de complexité ou tout au moins sa maitrise