ArrayList et Itérateur

La classe java.util.Vector est une classe héritée de Java 1. Elle n'est conservée dans l'API actuelle que pour des raisons de compatiblité ascendante et elle ne devrait pas être utilisée dans les nouveaux programmes.

Dans tous les cas, il est préférable d'utiliser un ArrayList.

java.util.ArrayList :

Un ArrayList est un tableau qui se redimensionne automatiquement. Il accepte tout type d'objets, null y compris. Chaque instance d'ArrayList a une capacité, qui définit le nombre d'éléments qu'on peut y stocker.

Au fur et à mesure qu'on ajoute des éléments et qu'on dépasse la capacité, la taille augmente en conséquence.

3

Un ArrayList fournit un accès aux éléments par leur indice très performant et est optimisé pour des opérations d'ajout/suppression d'éléments en fin de liste.

Complexité : Les opérations size, isEmpty, get, set, iterator sont exécutées en temps constant.

Les opérations d'ajout/suppression sont exécutées en temps constant amorti (les ajouts/suppressions en fin de liste sont plus rapides).

4

Exemple pour ArrayList

Les opérations principales sur un ArrayList sont :

add(Object o) :	ajoute l'objet o à la fin du ArrayList
clear() :	vide le ArrayList
get(int index) :	renvoie l'Object à l'index spécifié. Renvoie une exception si vous dépassez le tableau (IndexOutOfBoundsException)
size() :	renvoie la taille du ArrayList

```
// on crée un ArrayList de taille 10
List monArrayList = new ArrayList(10) ;

// on ajoute 30 entiers
for(int i=0;i<30;i++) {
        monArrayList.add(new Integer(i));
}
```

On remarque que le ArrayList , au moment d'ajouter 10, atteint sa capacité maximale, il se redimensionne pour en accepter plus

Tri et classe Comparable

Comment trier une List (ArrayList, Vector, ...) ou un tableau

La classe java.util.Collections apporte la solution :

```
List ma_liste = new ArrayList();
Collections.sort(ma_liste);
```

ma_liste est ainsi triée par ordre croissant. Pour trier dans l'ordre décroissant, il suffit de faire ceci :

```
Collections.sort(ma_liste, Collections.reverseOrder());
```

Attention cependant, cet exemple fonctionne que si les éléments dans la liste sont de type Comparable, comme int, String, Date, ... (voir l'interface java.lang.Comparable).

Si ce n'est pas le cas (les éléments sont des objets que vous avez défini), il faut que votre classe implémente l'interface java.lang.Comparable, donc en fait il faut définir la méthode int compareTo(Object o).

Interface Comparable

Une seule méthode est contenue dans l'interface Comparable. Il s'agît de la méthode compareTo() ayant pour fonction de comparer un objet passé en argument à l'instance de classe courante.

```
public interface Comparable {
   public int compareTo(Object o);
}
```

La méthode compareTo() suite à la comparaison de l'objet passé en argument par rapport à l'objet courant, retourne une valeur entière qui indique l'ordre de ces objets.

- Un entier négatif signifie que le premier argument est inférieur au second.
- Une valeur égale à zéro signifie que le premier argument est égal au second.
- Un entier positif signifie que le premier argument est supérieur au second.

Exemple

```
class Voiture {
   String marque = "";
   int nbCheveau = 0;

   public Voiture(String s, int i) {
      marque = s;
      nbCheveau = i;
   }

   public int getNbCheveau()  { return
nbCheveau;  }

   public String toString()  { return marque + "\t" +
nbCheveau;   }
}
```

On veut pouvoir trier une liste de Voiture suivant leur nombre de chevaux. Voilà alors les modifications à apporter à la classe :

```
class Voiture implements java.lang.Comparable {
   // méthode à implémenter
   //autre doit être de type Voiture !

public int compareTo(Object autre) {
 int nombre2 = ((Voiture) autre).getNbCheveau();
 int nombre1 = this.getNbCheveau();
 if (nombre1 > nombre2)  return 1;
 else if(nombre1 == nombre2) return 0;
 else return -1;
   }
}
```

Maintenant, ont peut trier facilement une liste de voitures :

```
Collections.sort(liste_voitures);
```

Le code est identique pour un tableau. Il faut seulement utiliser la classe Arrays.

```
Arrays.sort(unTableauDeVoitures);
```

Consulter le site suivant pour la classe Arrays :
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Arrays.html

Interface Iterator

Une des méthodes de l'interface Collection, à savoir

Iterator iterator()

vise à construire un itérateur pour une collection.

Iterator est une interface dont ListIterator est une sous-interface adaptée aux listes.

Au travers d'une implantation de cette interface, il doit être possible de parcourir les objets constituant une collection.

Les trois opérations réalisées au cours d'un tel parcours sont :

une initialisation,
un test de fin de parcours,
un passage à l'élément suivant.

Méthodes

boolean hasNext() : elle permet de tester l'existence d'un élément après le curseur ;

Object next() : elle renvoie l'élément courant et déplace le curseur après cet élément ;

void remove() : elle supprime le dernier élément renvoyé par next (c'est-à-dire l'élément avant le curseur). Cette opération est optionnelle et peut se résumer à la levée de l'exception UnsupportedOperationException

Les différentes collections possèdent par ailleurs une méthode Iterator iterator() dont la fonction est d'initialiser un itérateur.

La sous-interface ListIterator permet :

- le parcours d'une liste dans les deux sens (cela a un sens car une liste est une collection ordonnée) ;
- la modification de la liste pendant le parcours ;
- d'obtenir une position courante (située entre l'élément qui serait retournée par previous et celui retourné par next). Dans une liste de longueur n, les index valides sont 0, 1, ..., n.

Les méthodes ajoutées dans cette interface sont les suivantes :

- boolean hasPrevious() qui teste s'il existe un élément avant le curseur ;
- Object previous() qui renvoie, s'il existe, l'élément avant le curseur ;
- void add(Object o) qui permet l'insertion d'un élément avant le curseur (c'est-à-dire entre les éléments qui seraient retournés par appel de Object previous() et Object next()) ;
- int nextIndex() qui renvoie l'index de l'élément avant le curseur ;
- int previous qui renvoie l'index de l'élément après le curseur ;
- void set(Object o) qui remplace le dernier élément retourné par previous ou next (si la dernière opération a modifié la liste, c'est-à-dire si la dernière opération a été add ou remove, l'exception IllegalStateException est levée).

Exemple

```
import java.util.*;
class TestListes {
   public static void main(String[] arg) {
    List liste1 = new ArrayList();
    for(int i=0; i<10; i++) {
       liste1.add(new Integer(i));
}
    Iterator i1 = liste1.iterator();
    while(i1.hasNext())
       System.out.print(i1.next() + " ");
   }
}
```

Exemple

```
/*
Contenu de al: C A E B D F
Après modification: F+ D+ B+ E+ A+ C+

 */

import java.util.ArrayList;
import java.util.Iterator;
import java.util.ListIterator;

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

    al.add("C");
    al.add("A");
    al.add("E");
    al.add("B");
    al.add("D");
    al.add("F");
```

17

```
System.out.print("Contenu de al: ");
    Iterator<String> itr = al.iterator();
    while (itr.hasNext()) {
      String element = itr.next();
      System.out.print(element + " ");
    }

    System.out.println();

    ListIterator<String> litr = al.listIterator();
    while (litr.hasNext()) {
      String element = litr.next();
      litr.set(element + "+");
    }

    // Now, display the list backwards.
    System.out.print("Après modification:  ");
    while (litr.hasPrevious()) {
      String element = litr.previous();
      System.out.print(element + " ");
    }
  }
}
```

18

for each

```
/*

    Original : 1 2 3 4 5
    Somme: 15
     */

    import java.util.ArrayList;

    public class MainClass {
      public static void main(String args[]) {
        ArrayList<Integer> vals = new ArrayList<Integer>();

        vals.add(1);
        vals.add(2);
        vals.add(3);
        vals.add(4);
        vals.add(5);
```

19

```
System.out.print("Original: ");
      for (int v : vals)
        System.out.print(v + " ");
      System.out.println();

      int sum = 0;
      for (int v : vals)
        sum += v;

      System.out.println("Somme: " + sum);
    }
  }
```