La classe *ArrayList* permet de mettre en oeuvre un tableau dont la taille peut varier dynamiquement.

La classe *ArrayList* implémente l'interface *List* qui elle même étend *Collection*. De plus, elle est une extension de la classe abstraite *AbstractList*.

Les éléments d'un objet *ArrayList* peuvent être de n'importe quel type référence. En outre, la valeur *null* est également tolérée dans cette liste.

La plupart des opérations (*size()*, *isEmpty()*, *get()*, *set()*, *iterator* et *listIterator()*) s'effectuent en temps constant (*O(1)*). Par contre, l'ajout d'éléments demande un temps de traitement proportionnel au nombre d'éléments à ajouter (*O(n)*).

La création et l'initialisation d'un objet *ArrayList* dépend de trois constructeurs. Le premier permet d'instancier un objet vide avec une capacité initiale égale à 10. Un autre accepte comme argument une valeur destinée à définir cette capacité initiale. Le dernier initialise la liste avec les éléments d'une collection qui lui aura été spécifiée.

```
// par défaut : capacité initiale = 10
ArrayList liste = new ArrayList();
// initialisé avec une capacité initiale
ArrayList liste = new ArrayList(100);
// initialisé avec un collection
Collection collection = new LinkedList();
collection.add(objet);
// ...
ArrayList liste = new ArrayList(collection);
```

L'ajout d'éléments dans une collection *ArrayList* se réalise de différentes manières. Il est possible d'ajouter un objet ou tous les éléments d'une collection existante. Egalement, des méthodes offrent l'opportunité d'insérer des objets à partir d'un index précis.

```
// ajout d'un objet
liste.add(objet);
// ajout des éléments d'une collection
liste.addAll(collection);
// insertion d'un objet à l'index spécifié
liste.add(10, objet);
// insertion d'éléments d'une collection à l'index spécifié
liste.addAll(10, collection);
```

L'accès aux éléments d'une collection *ArrayList* s'effectue de la même manière que dans un tableau, c'est-à-dire par l'intermédiaire de leur index. Néanmoins, l'objet retourné sera du type *Object*, si bien qu'il faudra penser à le convertir dans la plupart des cas d'utilisation.

```
Object obj = liste.get(index);
```

Les méthodes *contains()* et *containsAll()* fournissent un moyen de vérification de présence d'un ou plusieurs éléments au sein d'une collection *ArrayList*.

```
// vérifie si la liste contient l'élément spécifié
boolean resultat = liste.contains(objet);
// vérifie si la liste contient tous
//les éléments de la collection spécifiée
boolean resultat = liste.containsAll(collection);
```

Les méthodes *indexOf()* et *lastIndexOf()* permettent de rechercher l'élément fourni, respectivement à partir du début et de la fin de la liste.

```
// recherche la première occurence de l'objet spécifié
Object element = liste.indexOf(obj);
// recherche la dernière occurence de l'objet spécifié
Object element = liste.lastIndexOf(obj);
```

Les méthodes *subList()* et *toArray()* sont capables d'extraire une partie ou la totalité des éléments d'une liste, dans respectivement un objet *List* et un tableau.

```
// retourne une liste contenant les éléments
au sein de l'intervalle spécifié
List sousListe = liste.subList(5, 15);
// retourne tous les éléments de la liste dans un tableau
Object[] tableau = liste.toArray();
// retourne tous les éléments de la liste dans un tableau
//et affecte le type d'exécution du tableau spécifié au tableau retourné
String[] tabType;
Object[] tableau = liste.toArray(tabType);
```

La suppression d'un ou plusieurs éléments d'une liste s'accomplit par le truchement de plusieurs méthodes, en l'occurrence *remove()*, *removeRange()*, *retainAll()* et *clear()*.

```
// supprime l'objet à l'index spécifié
liste.remove(10);
// supprime les objets compris dans l'intervalle spécifié
liste.removeRange(5, 15);
// supprime tous les éléments ne faisant
//pas partie de la collection spécifiée
liste.retainAll(colection);
// supprime tous les éléments de la liste
liste.clear();
```

La taille de la liste est accessible à partir de la méthode *size()*.

```
int taille = liste.size();
```

La capacité peut être augmentée par la méthode *ensureCapacity()* et réadapter par rapport à la taille courante avec *trimToSize()*.

```
// augmente la capacité de la liste
liste.ensureCapacity(50);
// réajuste la capacité à la taille de la liste
int taille = liste.size(); // taille = 33
liste.trimToSize();  capacité = 33
```

La méthode *set()* remplace un élément de la liste par un objet spécifié et retourne l'élément remplacé.

```
Object eltRemplace = liste.set(10, nouvelObjet);
```

Le parcours des éléments d'un objet *ArrayList* peuvent se faire en utilisant une boucle, ou en appliqant un itérateur sur les éléments de la liste.

```
for(int i = 0; i < liste.size(); i++){
    Object element = liste.get(i);
}
// itérateur
Iterator iterateur = liste.iterator();
while(iterateur.hasNext()){
    Object element = iterateur.next();
}
// itérateur de liste
ListIterator iterateur = liste.listIterator();
while(iterateur.hasNext()){
    Object element = iterateur.next();
}
```

L'objet *ListIterator* possède plusieurs méthodes supplémentaires efficaces dans la gestion des listes, par rapport à une implémentation de l'interface *Iterator* surtout utilisée dans le cadre des ensembles (*Set*).

Ainsi, il est possible de parcourir une liste dans les deux directions possibles.

```
// parcours du bas vers le haut
ListIterator iterateur = liste.listIterator();
while(iterateur.hasPrevious()){
   Object element = iterateur.previous();
}
// parcours du haut vers le bas
ListIterator iterateur = liste.listIterator();
while(iterateur.hasNext()){
   Object element = iterateur.next();
}
```

D'autre part, les méthodes permettent de retourner l'index suivant (*nextIndex()*) ou précédent (*previousIndex()*), de supprimer (*remove()*), d'ajouter (*add()*) et de remplacer (*set()*) des éléments.

Exemple [voir]

```
import java.util.ArrayList;
import java.util.Iterator;

public class programme {
  public static void main(String[] args) {
    ArrayList tableau = new ArrayList(500);

    for(int i = 1; i <= 100; i++)
      tableau.add(new Integer(i));
    tableau.trimToSize();

    System.out.println("Taille du tableau : " + tableau.size());

    if(tableau.contains(new Integer(50)))
      System.out.println("L'objet Integer ayant une "
            + "valeur égale à 50 a été trouvé");

    for(int i = 0; i < tableau.size(); i++)
      System.out.println(i + " : " + tableau.get(i).getClass()
            + " " + tableau.get(i));

    for(int i = 0; i < tableau.size(); i++)
      if(((Integer)tableau.get(i)).intValue() % 2 != 0)
        tableau.set(i, null);

    Iterator iterateur = tableau.iterator();
    while(iterateur.hasNext())
      System.out.println(iterateur.next());

    System.out.println("Le premier null se trouve à l'index "
              + tableau.indexOf(null));
    System.out.println("Le dernier null se trouve à l'index "
              + tableau.lastIndexOf(null));

    Integer[] type = new Integer[10];
    Object[] sauvegarde = tableau.toArray(type);
    System.out.println(sauvegarde.getClass());

    if(!tableau.isEmpty())
      tableau.clear();

    System.out.println("Taille du tableau : " + tableau.size());
  }
}
```