Les classes `Arrays`, `ArrayList` et `Iterator`

La bibliothèque `java.util`

La bibliothèque `java.util` est un paquetage.

Un paquetage est une bibliothèque de classes contenant de nombreuses classes :

- La classe `Arrays`
- La classe `ArrayList`
- La classe `Iterator`

Pour utiliser une classe de ce paquetage, on doit utiliser la directive **import** et une combinaison du nom du paquetage et du nom de la classe que l'on souhaite utiliser.

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

Méthodes `static`

Il est possible en java de définir une méthode qui ne requiert pas de créer préalablement un objet pour l'appeler : les **méthodes de classes**.

Une méthode de classe est définie en ajoutant le mot-clé `static` à sa déclaration. Une méthode de classe peut être `private` ou `public`.

Remarque

Une méthode de classe peut être surchargée.

Méthodes **static**

Une méthode de classe existe dés sa déclaration. Pour appeler une méthode de classe dans une autre cmasse, on préfixe le nom de la méthode du nom de la classe (aucune référence n'est nécessaire).

```
class Person {
  private String name;
  public Person(String name) {this.name = name;}
  public String getName() {return name;}

  public static void print(Person p) {
    System.out.println("My␣name␣is␣" + p.getName()
    ;}
 } // fin Person

class Program {
  public static void main(String[] args) {
    Person.print(new Person("John␣Doe"));
} // fin Program
```

Autoboxing

Le langage JAVA™ associe à chaque type primitif une classe dite enveloppe :

- `Integer` pour le type **`int`** ,
- `Double` pour le type **`double`** ,
- `Boolean` pour le type **`boolean`** ,
- etc

L'autoboxing est la conversion automatique par le compilateur d'une variable de type primitif en un objet de la classe enveloppe associée :

```
Integer i = 1;
Double x = 0.2;
Boolean b = true;
```

Unboxing

L'unboxing est le mécanisme inverse de l'autoboxing, à savoir la converion automatique d'un objet d'une classe enveloppe en une variable du type primitif associé.

```
int j = i; // j = 3;
double y = x; // y = 0.2;
boolean l = b; // l = true;
```

Dans une expression arithmétique, l'unboxing est utilisé pour convertir automatiquement chaque d'une classe enveloppe en une variable du type primitifs associé.

La classe `Arrays`

La classe `Arrays` ne fournit que des méthodes de classes dont :

- `toString` qui retourne une représentation d'un tableau par une chaîne de caractères (en appelant la méthode `toString` des objets référencés).
- `equals` qui comparent deux tableaux (en appelant la méthode `equals` des objets référencés).
- `copyOf` qui crée une copie des références d'un tableau.

Remarque

La classe fournit plusieurs autres méthodes.

La classe `ArrayList`

La classe `ArrayList` modélise un tableau redimensionnable (un tableau redimensionné automatiquement).

A l'instar d'un tableau, une arraylist ne peut contenir que des références d'un même type. Le type des références d'une arraylist est indiquée entre les symboles < et > : `ArrayList<T>`.

Une `ArrayList<T>` contiendra des références de type `T` comme un tableau `T[]`.

Il n'est pas possible de définir une arraylist dont les éléments sont de type primitif; vous devez utiliser les classes enveloppes :

```
ArrayList<Integer> arraylist;
// INTERDIT : ArrayList<int> arraylist;
```

La classe `ArrayList`

La méthode `size()` retourne le nombre de références contenues dans un arraylist.

Contrairement à un tableau `T[]`, une `ArrayList<T>` est vide à sa création.

```
ArrayList<String> arraylist;
arraylist = new ArrayList<String>();
// affiche 0
System.out.println(arraylist.size());
```

Ajout d'un élément

La méthode `add` ajoute un élément en fin d'arraylist.

```
ArrayList<String> arraylist;
arraylist.add("java");
arraylist.add("C++");
arraylist.add("C#");
// affiche 3
System.out.println(arraylist.size());
```

La méthode `toString()`

La méthode `toString` retourne une représentation de l'arraylist par une chaîne de caractères.

```
// affiche [java,C++,C#]
System.out.println(arraylist.toString());
```

La méthode appelle la méthode `toString` de chaque objet

La méthode `equals`

La méthode `equals` comparent deux arraylist et retourne **`true`** si les deux arraylist sont identiques.

> La méthode utilise la méthode `equals` des objets de l'arraylist.

```
ArrayList<String> arraylist1, arraylist2;
arraylist1.add("java"); arraylist2.add("java");
arraylist1.add("C++"); arraylist2.add("C++");
arraylist1.add("C#"); arraylist2.add("C#");
// affiche true
System.out.println(arraylist1.equals(arraylist2
));
```

Deux arraylist sont égales:

- si elles ont la même taille
- et si la liste de leurs éléments sont identiques.

Accéder / Modifier / Retirer un élément

Il est possible d'accéder à un élément, de modifier ou retirer un élement :

- La méthode `get(int i)` retourne le i-^{e} élément de la liste.
- La méthode `set(int i, T o)` remplace le i-^{e} élément de la liste par la référence `o`.
- La méthode `remove(int i)` retire le i-^{e} élément de la liste.

> **Attention :** Si l'indice i n'existe pas, ces méthodes provoquent une erreur à l'exécution.

Recopie d'une arraylist

Pour recopier une arraylist, on doit instancier une nouvelle arraylist en donnant en argument du constructeur l'arraylist à recopier :

```
Arraylist<String> copie;

copie = new ArrayList<String>(arraylist);
```

Convertir une arraylist en tableau

La méthode `toArray` permet de convertir une arraylist en tableau. La méthode est surchargée. La version sans argument converit une arraylist en tableau d' `Object` :

```
Object[] tableau;

tableau = arraylist.toArray();
```

Convertir une arraylist en tableau

Une autre version de la méthode `toArray` permet de retourner un tableau d'un type choisi :

```
String[] tableau;

tableau = arraylist.toArray(new String[
arraylist.size()]);
```

Convertir une arraylist en tableau

Attention : Si le tableau donné en argument est assez grand pour contenir l'arraylist, il est rempli avec les références contenues dans l'arraylist et la méthode retourne la référence de ce tableau.

Si le tableau donné en argument n'est pas assez grand, un autre tableau du même type est créé et rempli avec les références de l'arraylist. La méthode retourne la référence de ce tableau et non celle du tableau donné en argument.

Si le tableau donné en argument est trop grand, le tableau est rempli avec les références de l'arraylist et complété avec des références `null`.

Convertir un tableau en arraylist

Pour convertir un tableau en arraylist, on doit utiliser la méthode `asList` de la classe `Arrays` :

```
String[] languages = {"java","C++","C#"};
ArrayList<String> arraylist;
arraylist = new ArrayList<String>(Arrays.asList
(languages));
```

Parcourir une arraylist

Comme pour les tableaux, on peut utiliser la version « for-each » de la boucle **for** :

```
for(String language : arraylist)
{
  System.out.prinln(language);
}
```

La classe `Iterator`

Pour parcourir une arraylist, on peut aussi utiliser un itérateur : un itérateur est un objet modélisant un curseur et qui permet de parcourir une arraylist.

La classe `Iterator` modélise un itérateur. Pour créer un itérateur associé à une arraylist, on utilise la méthode `iterator()` de l'arraylist.

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

Iterator<String> iter = arraylist.iterator();
```

La classe `Iterator`

Le parcours de l'arraylist à travers un itérateur se fait en utilisant les méthodes `next()` et `hasNext()` :

- La méthode `next()` retourne l'élément pointé par le curseur et positionne le curseur sur l'élément suivant.
- La méthode `hasNext()` retourne `true` s'il reste des éléments à parcourir et `false` sinon.

```
while (iter.hasNext()) {
  String s = iter.next();
  // instructions
}
```

La classe `Iterator`

Il est possible de supprimer un élément lors du parcours d'une arraylist à travers un itérateur :

```
while (iter.hasNext()) {
   String s = iter.next();
   if (s.startsWith("lic"))
   {
      iter.remove();
   }
}
```

> **Attention :** La méthode `remove()` provoque une erreur à l'exécution si elle est utilisée avant la méthode `next()`.