Chapitre 12

Utilisation d'objets : String et ArrayList

Dans ce chapitre, nous allons aborder l'utilisation d'objets de deux classes prédéfinies de Java d'usage très courant. La première, nous l'utilisons depuis longtemps déjà, c'est la classe `String`. En java, les chaînes de caractères sont des objets. Nous allons apprendre dans ce chapitre à mieux les utiliser. La seconde classe s'appelle `ArrayList`. Les objets de cette classes sont équivalent à des tableaux, mais sont plus agréables à utiliser que les tableaux grâce aux méthodes qu'elle fournit.

12.1 Ce que l'on sait déjà de String

Il existe en java un type prédéfini pour les chaînes de caractères avec une syntaxe spéciale. Ce type s'appelle `String` et pour écrire une chaîne, on place les caractères de la chaîne entre guillemets. Il existe pour ce type un opérateur qui s'appelle la *concaténation* et qui s'écrit +. La concaténation permet de créer une chaîne en collant bout à bout deux chaînes existantes. Par exemple `"to"+"to"` crée la chaîne `"toto"`.

Par extension on peut concaténer une chaîne avec une valeur d'un type de base (int, boolean, double ou char). Il y a une conversion de type implicite et le résultat est une chaîne de caractère. Par exemple `"resultat: "+125` crée la chaîne : `"resultat: 125"`.

12.2 Naissance et vie des chaînes de caractères

Le type `String` existe dès l'origine dans le système java, mais ce n'est pas le cas des valeurs de ce type, c'est à dire des chaînes particulières que l'on peut utiliser. Par exemple, la chaîne `"bonjour"` n'existe pas au démarrage du système : si on veut l'utiliser il faut d'abord la créer. Pour cela, il existe quatre moyens :

- en écrivant cette chaîne avec ses guillemets dans le programme.
- au moyen de l'instruction `new`.
- au moyen de l'opérateur + : le résultat d'une concaténation est une nouvelle chaîne qui n'existait pas avant.
- au moyen d'une méthode qui renvoie une nouvelle chaîne.

Il existe plusieurs façons de créer une chaîne avec un `new`. Nous allons utiliser celle qui consiste à spécifier dans un tableau de caractères le contenu de la chaîne à créer.

```
char[] tab = {'b','o','n','j','o','u','r'};
String s = new String(tab);
```

Cela crée la chaîne `"bonjour"`. Si l'on n'a plus besoin du tableau après la création de la chaîne, on n'a pas besoin d'utiliser une variable pour désigner ce tableau, on peut écrire directement la création :

```
String s = new String(new char[]{'b','o','n','j','o','u','r'});
```

Une fois la chaîne créée, on peut l'afficher au moyen de `Terminal.ecrireString` et l'utiliser comme opérande de l'opérateur de concaténation. On peut également appeler des méthodes. Jusqu'à présent, les méthodes que nous avons écrites sont des méthodes des classes, déclarées avec le mot clé `static` et que l'on appelle avec le nom de la classe, un point et le nom de méthode et les paramètres entre parenthèses. Par exemple dans `Terminal.ecrireString(s)`, `Terminal` est le nom de la classe. Avec les chaînes de caractères, nous allons commencer à utiliser des méthodes des objets. Pour les appeler, il faut avant le point non pas un nom de classe, mais un objet, une valeur.

Prenons un premier exemple : la méthode `length()` renvoie la longueur de la chaîne. Elle ne prend pas de paramètre. La longueur est le nombre de caractères de la chaîne. Cette méthode n'a de sens que pour une chaîne donnée ; on l'appelle en mettant une chaîne ou un moyen d'en calculer une, avant le point et le nom de la méthode.

```
String s = "bonjour";
Terminal.ecrireIntln(s.length());
```

L'appel `s.length()` va appeler la méthode `length` sur l'objet contenu dans la variable `s`. Le résultat est 7 puisqu'il y a 7 caractères dans `"bonjour"`.

On appelle souvent les méthodes en mettant un nom de variable avant le point, mais on peut mettre n'importe quelle expression qui calcule un objet du bon type. On peut appeler `length` non seulement sur une variable mais sur d'autres sortes d'expressions calculant un objet `String` :

— une chaîne entre guillement : `"bonjour".length()`
— une chaîne créér par new : `(new String()).length()`
— le résultat d'une concaténation : `(s + "qsd").length()`
— etc.

12.3 Les chaînes ressemblent un peu aux tableaux

Par certains côtés, les chaînes de caractères ressemblent aux tableaux plus qu'aux types de base tels que `int` ou `char`.

Voici les points communs entre tableaux et chaînes :

— il y a deux temps différents : la déclaration et la création d'une valeur.
— il y a possibilité de création explicite d'une nouvelle chaîne au moyen d'une instruction `new`.
— comme pour les tableaux, il y a possibilité de création implicite au moyen d'une syntaxe spéciale. Pour les tableaux, avec les accolades, pour les chaînes, avec les guillemets.
— ce sont des structures regroupant plusieurs valeurs dans un certain ordre.
— plusieurs noms différents peuvent être donnés à une même structure.
— les chaînes et les tableaux ont une longueur et cette longueur est supérieure ou égale à 0. Elle est fixe et invariable dans le temps.
— en revanche, une variable donnée peut contenir successivement des structures de tailles différentes.

Contrairement aux tableaux :

— il n'y a pas une syntaxe spécifique pour accéder directement aux caractères d'une chaîne de caractère.

Les chaînes de caractères sont le premier exemple d'objet que nous voyons en cours. Les types des objets sont comme les tableaux des types *références*, c'est à dire que les variables de ces types contiennent l'adresse des objets ou des tableaux en mémoire.

12.4 Quelques méthodes intéressantes

Voici quelques méthodes intéressantes :

— `charAt(int n)` : cette méthode renvoie le nième caractère de la chaîne, la numérotation commence à 0. Par exemple, si `s` est une `String`, `s.charAt(0)` renvoie le premier caractère (type `char`) de `s`.

— `toCharArray()` permet de transformer une chaîne en un tableau de `char`. Par exemple, si `s` est la `String "bonjour"`, `s.toCharArray()` renvoie un tableau de 7 `char` :

0	1	2	3	4	5	6
'b'	'o'	'n'	'j'	'o'	'u'	'r'

— `compareTo(String s2)` : compare deux chaînes selon l'ordre lexicographique (l'ordre du dictionnaire). Si `s1` et `s2` sont deux `String`, `s1.compareTo(s2)` renvoie un entier. Cet entier est négatif si `s1` est plus petit que `s2`, positif si `s1` est plus grand que `s2`, et 0 si `s1` et `s2` sont égales.

— `s1.toLowerCase()` et `s1.toUpperCase()` renvoient une nouvelle chaîne égale à `s1` mais avec toutes les lettres en minuscule et en majuscule respectivement.

— `trim()` : renvoie une chaîne dans laquelle les espaces en début et en fin de chaîne ont été supprimés. Par exemple `"  truc chose  ".trim()` renvoie la chaîne `"truc chose"`.

— `split(String s)` : découpe la chaîne en plusieurs morceaux en utilisant la chaîne `s` comme séparateur. Le résultat est un tableau de chaînes. Par exemple `"un;deux;trois".split(";")` renvoie le tableau `{"un","deux","trois"}`.

— `indexOf(String s)` : renvoie l'indice de la première occurrence de la chaîne `s` dans la chaîne. Par exemple `"un deux trois".indexOf("deux")` renvoie 3, car la chaîne `"deux"` commence à l'indice 3 de la chaîne `"un deux trois"`.

— `substring(int debut, int fin)` : renvoie la sous-chaîne de la chaîne sur laquelle la méthode est appelée comprise entre les indices `debut` et `fin`. Le caractère d'indice `debut` est inclus dans le résultat, mais pas celui d'indice `fin`. Par exemple `"bonjour".substring(2,4)` renvoie la sous-chaîne `"nj"`. Autrement dit, elle renvoie la sous-chaîne comprenant les caractères d'indice 2 et 3.

```
public class ExChaine2{
   public static void main (String [] arguments){
      String s1 = "bonjour";
      String s2;
      Terminal.ecrireString("Entrez une chaine: ");
      s2 = Terminal.lireString();
      Terminal.ecrireCharln(s1.charAt(0));
      Terminal.ecrireStringln(s1.toUpperCase());
      Terminal.ecrireIntln(s1.compareTo(s2));
      Terminal.ecrireStringln("" + s1.equals(s2));
   }
```

```
}
```

A noter : il n'y a pas de méthode ni d'autre moyen de changer un caractère dans une chaîne : une fois la chaîne créée, on ne peut pas la modifier. Pour obtenir un résultat plus ou moins équivalent à un changement de caractère, il faut créer une nouvelle chaîne en concaténant des portions de la chaîne originale avec le caractère différent.

12.5 Variables et initialisations

Une valeur spéciale, `null`, est utilisable pour dire qu'une variable `String` ne pointe vers aucun objet. Ça n'est pas la même chose que "non initialisé".

Aucune méthode ne peut s'exécuter sur une variable qui contient `null`. Si l'on essaie de l'appeler cela provoque une erreur à la compilation ou à l'exécution.

Par exemple, dans le code suivant, `s` est non initialisée. le programme ne **compilatera** pas.

```
public class VNI{
   public static void main(String[] args){
      String s;
      Terminal.ecrireIntln(s.length());
   }
}
```

Dabs le code suivant, `s` est à `null`. Le programme compilera, mais on aura une erreur à l'**exécution :** une `NullPointerException` due à l'application de la méthode `length()` à une variable qui vaut `null`.

```
public class NPE {
   public static void main(String[] args){
      String s= null;
      Terminal.ecrireIntln(s.length());
   }
}
```

Concrètement, `null` est utilisé pour dire qu'un objet n'est pas présent (imaginez par exemple qu'on représente une personne par un prénom, un second prénom, et un nom de famille, soit trois Strings. Si la personne n'a pas de second prénom, il pourra être initialisé à null).

Lorsque l'on crée un tableau de chaînes, la valeur `null` est la valeur par défaut placée dans toutes les cases. Dans l'état actuel de vos connaissance, c'est le cas où vous risquez le plus de rencontrer la valeur `null`.

Il est possible d'utiliser `null` comme valeur dans une affectation et de la tester dans un test d'égalité ou de différence. En revanche, on ne peut pas appeler de méthodes sur cette valeur. Par exemple `null` n'a pas de longueur, pas de premier caractère, etc.

```
public class VNI{
   public static void main(String[] args){
      String s = "truc";
      if (s!=null){
         Terminal.ecrireIntln(s.length());
      }
```

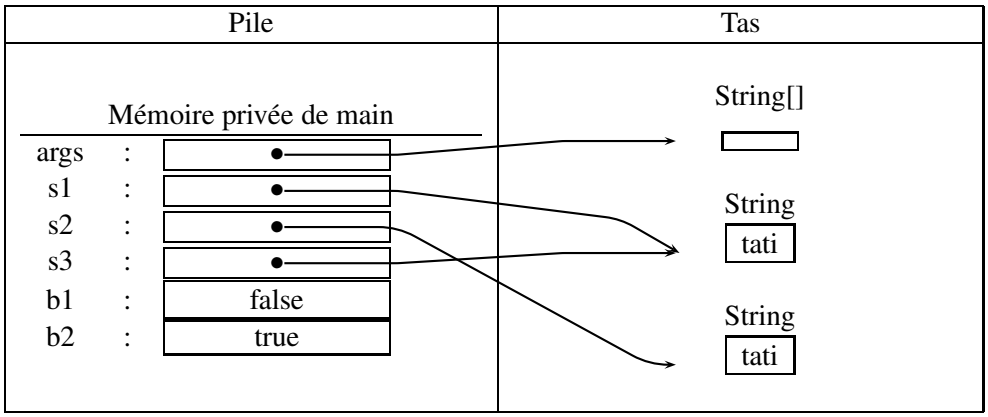
```
      s = null;
      if (s!=null){
         Terminal.ecrireIntln(s.length());
      }
   }
}
```

12.6 Comparaison de chaînes

Comme pour un tableau, les opérateurs == et != ne regardent pas le contenu des chaînes mais juste leur adresse en mémoire. La question posée est : les deux chaînes sont-elles à la même adresse ? Ceci est illustré par le programme suivant.

```
public class EgCh{
   public static void main(String[] args){
      String s1, s2, s3;
      boolean b1, b2;
      s1 = "tati";
      s2 = "ta";
      s2 = s2 + "ti";
      s3 = s1;
      Terminal.ecrireStringln("s1=" + s1 + "␣s2=" + s2 + "␣s3=" + s3);
      b1 = (s1 == s2);
      b2 = (s1 == s3);
      Terminal.ecrireStringln("s1␣==␣s2?␣" + b1);
      Terminal.ecrireStringln("s1␣==␣s3?␣" + b2);
   }
}
```

Si l'on représente par un dession l'état de la mémoire après les affectations des trois chaînes, cela donne le dessin suivant.



Si l'on veut réaliser une comparaison du contenu des chaînes et non plus de leur adresse, il faut utiliser une méthode. Pour une comparaison d'égalité, c'est la méthode `equals` qu'il faut appeler.

Elle renvoie une valeur booléenne. Elle compare deux chaînes : l'une est l'objet sur lequel on appelle la méthode, l'autre est passée en paramètre. `s1.equals(s2)` fait un test d'égalité du contenu entre les deux chaînes `s1` et `s2`. Elles sont considérées comme égales si elles ont la même taille et les mêmes caractères dans le même ordre.

Le programme suivant illustre l'utilisation de `equals`. Notons au passage que l'on peut insérer un caractère guillement dans une chaîne de caractères en le faisant précéder du caractère `\` (voir la variable `s2`).n

```
public class ExChaine{
   public static void main(String[] args){
      String s1 = "Bonjour";
      String s2 = "C'est␣\"bien\"␣";
      String s3;
      String[] ts = {"Paul", "Andre", "Jacques", "Odette"};
      Terminal.ecrireStringln(s2);
      Terminal.ecrireString("Entrez␣une␣chaine:␣");
      s3=Terminal.lireString();
      Terminal.ecrireStringln("s3:␣" + s3);
      s2 = "Bon";
      s3 = s2 + "jour";
      if (s1 != s3){
         Terminal.ecrireStringln("Bizarre:␣s1␣n'est␣pas␣egal␣a␣s3!");
         Terminal.ecrireStringln("s1:␣" + s1 + ":");
         Terminal.ecrireStringln("s3:␣" + s3+ ":");
      }
      if (s1.equals(s3)){
         Terminal.ecrireStringln("s1␣est␣quand␣meme␣egal␣a␣s3!");
      }
      if (!s1.equals(s3)){
         Terminal.ecrireStringln("s1␣n'est␣toujours␣pas␣egal␣a␣s3!");
      }
   }
}
```

La méthode `compareTo` fait une comparaison d'ordre selon le code unicode de chaque caractère. Cela correspond à l'ordre alphabétique uniquement pour les caractères sans accent. Elle renvoie un entier qui est 0 en cas d'égalité, un entier positif si l'objet est plus grand que la paramètre et un entier négatif si c'est le paramètre qui est plus grand.

12.7 Paramètre de la méthode main

Depuis le début de l'année, nous utilisons systématiquement la méthode `main` avec un paramètre de type `String[]`, c'est à dire un tableau de chaînes de caractères. Ce paramètre permet de transférer des informations entre la ligne de commande et le programme java. Prenons un exemple où le programme se contente d'afficher les valeurs passées sur la ligne de commande.

```
public class LigneCommande{
   public static void main(String[] args){
```

```
      for (int i=0; i < args.length; i++){
         Terminal.ecrireStringln(args[i]);
      }
   }
}
```

Voici un exemple d'exécution :

```
> java LigneCommande un deux trois
un
deux
trois
```

La tableau `args` dans cette exécution a trois cases. Sa valeur est

0	1	2
¨un¨	¨deux¨	¨trois¨

Notons que même si l'on passe un nombre en paramètre, celui-ci est contenu dans le tableau sous forme d'une chaîne.

```
> java LigneCommande un 12 56 deux
un
12
56
deux
```

La tableaux args vaut

0	1	2	3
¨un¨	¨12¨	¨56¨	¨trois¨

Si l'on veut transformer une de ces chaînes en un entier, il faut utiliser une fonction de conversion.

12.8 Conversion entre chaînes et autres types

Il est parfois utile de convertir une chaîne de caractère en une valeur d'un autre type. Par exemple, on peut vouloir transformer une chaîne qui ne contient que des chiffres en un nombre entier. Pour réaliser la conversion, il faut utiliser la méthode `Integer.parseInt` et lui donner en paramètre la chaîne à convertir.

```
public class StringInt2{
   public static void main(String[] args){
      int x;
      String s = "12";
      x = Integer.parseInt(s);
      Terminal.ecrireIntln(x);
   }
}
```

Pour convertir une valeur de type double, il faut utiliser la méthode `Double.parseDouble` et pour le type boolean, la méthode `Boolean.parseBoolean`.

Pour convertir dans l'autre sens, un int en chaîne, le plus simple est d'utiliser l'opérateur de concaténation : `""+12` (pour les doubles `""+12.3`, pour les booléens `""+true`). On concatène la chaîne vide avec la valeur à convertir.

12.9 Présentation de la classe ArrayList

Un programme a souvent besoin de pouvoir gérer une suite d'éléments (la liste des produits commandés par un internaute, les cartes dans un paquet, les figures d'un dessin). Dans de nombreux cas, la taille de cette suite d'éléments va varier tout au long du programme. La liste des références des produits commandés par un internaute augmentera à chaque produit commandé, et diminuera quand l'utilisateur décidera de retirer un produit de sa commande.

Une première approche serait d'utiliser un tableau :

```
String [] commandeReferences;
```

C'est possible, mais pas très simple à mettre en ¡uvre : la taille d'un tableau ne peut plus varier une fois qu'il a été créé. Pour commander un nouveau produit, il faudrait donc :

1. créer un nouveau tableau, plus grand ;
2. copier l'ancien tableau dans le nouveau ;
3. ajouter le nouvel élément ;
4. faire pointer la variable d'origine vers le nouveau tableau.

Soit en gros le code suivant :

```
String nouveauProduit= ......... ;
String [] tmp= new String [commandeReferences.length + 1];
for (int i= 0; i < commandeReferences.length; i++) {
  tmp[i]= commandeReferences[i];
}
// ajout du nouveau produit dans la dernière case
tmp[tmp.length - 1]= nouveauProduit;
// la variable qui désigne la commande doit pointer sur le
// nouveau tableau :
commandeReferences= tmp;
```

Il faudrait, de la même manière, écrire le code nécessaire pour supprimer un élément, en rechercher un, etc...

Comme ce type de problème est récurrent en informatique, `java`, comme la plupart des langages de programmation, fournit les classes nécessaires dans ses bibliothèques standard. Nous allons étudier la plus simple, la classe `ArrayList`.

12.10 Contenu de la classe ArrayList

La classe `ArrayList` permet donc de construire des tableaux de taille variable. 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 >). Par exemple :

```
ArrayList<String> maListe;
```

déclare la variable `maListe` comme étant une référence vers une `ArrayList` de `String`s.

Une `ArrayList` est un *objet*, et comme les tableaux, les objets sont créés par l'exécution de l'instruction `new`. Nous verrons plus tard les détails. Pour l'instant, il suffit de comprendre que

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

va créer une `ArrayList` vide.

Pour des raisons sur lesquelles nous reviendront plus tard, les `ArrayList` destinées à contenir des entiers ou des nombres réels se déclarent respectivement comme `ArrayList<Integer>` et `ArrayList<Double>` (notez les majuscules).

Enfin, notez que pour qu'une classe puisse utiliser les `ArrayList`, il faut écrire

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

avant la déclaration de votre classe dans le fichier java :

```
import java.util.ArrayList;
public class MaClasse {
    ...
}
```

12.10.1 Utilisation

Les objets se manipulent essentiellement à travers des *méthodes*. Les méthodes sont des procédures ou des fonctions qui sont appelées ń sur ż l'objet. Par exemple, la méthode `size()`, qui est une fonction, retourne la longueur d'une `ArrayList`. On pourra l'appeler de la manière suivante :

```
int long= maListe.size();
```

En règle générale, pour appeler une méthode *m* sur un objet *o*, on écrit `o.m()`. Donc, dans notre exemple, on appelle la méthode `size` sur l'objet `maListe`.

La liste des méthodes disponible pour un objet donné est fixe, et dépend du type (plus exactement de la *classe*) de l'objet. Notre objet `maListe` est de classe `ArrayList`, il dispose donc des méthodes d'`ArrayList`. La liste des méthodes d'une classe fait partie de la documentation de celle-ci, et explicite *ce qu'il est possible de faire* avec un objet de la classe en question.

Nous allons donc examiner quelques unes des méthodes disponibles sur les `ArrayList`. [1], pour négliger un certain nombre de problèmes qu'il est un peu tôt pour aborder. L'approximation que nous faisons est bien évidemment compatible avec la ń vraie ż spécification.

Le minimum vital

Les méthodes qui suivent permettent d'obtenir la même chose qu'avec un tableau normal, mais en gagnant, en plus, la possibilité d'ajouter une nouvelle case, ou de supprimer une case existante.

Dans le texte qui suit, `Type` correspond au type des éléments de l'`ArrayList`. Pour une `ArrayList` de `String`, par exemple, on remplacera `Type` par `String`

int size() : fonction qui renvoie la longueur d'une `ArrayList`; La fonction booléenne `isEmpty` permet de savoir si une liste est vide.

Type get(int i) renvoie l'entrée de la case numéro `i`. Comme pour les tableaux, les cases des `ArrayList` sont numérotées en commençant à 0. Le type de l'objet retourné est celui précisé lors de la création de l'`ArrayList`. Pour nous ce sera donc `String`, `Double` ou `Integer`. À partir de java 1.5, java ń sait ż convertir un `Integer` ou un `Double` en `int` ou `double`. La fonction suivante permet donc de calculer la somme des éléments d'une `ArrayList` d'entiers :

1. Dans ce cours, nous avons simplifié les en-têtes réels des méthodes de la classe `ArrayList`

```
public static int somme(ArrayList<Integer> liste) {
  int s=0;
  for (int i= 0; i < liste.size(); i++) {
    s= s + liste.get(i);
  }
  return s;
}
```

add(Type element) ajoute un élément à la *fin* de la liste. Pour construire la liste [2,3,5, 7, 11], on écrira donc :

```
ArrayList<Integer> l= new ArrayList<Integer>();
l.add(2); l.add(3); l.add(5);
l.add(7); l.add(11);
```

Notez qu'en toute rigueur, `add` prend un argument du type précisé lors de la création de l'`ArrayList`, c'est-à-dire `Integer` dans notre cas. Il faudrait donc écrire :

```
l.add(new Integer(2));
l.add(new Integer(3));
// etc...
```

Cependant, pour simplifier la vie des programmeurs, `java` 1.5 a introduit un système de conversion automatique entre les types de base `int, double, char, boolean` et les classes correspondantes (`Integer, Double, Character` et `Boolean`.

set(int i, Type element) remplace l'ancienne valeur qui était dans la case i par `element`. Logiquement, `i` doit être inférieure à la `size()` de l'`ArrayList`.

remove(int i) supprime l'élément qui est dans la case `i` ;

remove(Type element) supprime la première occurrence de l'élément de valeur `element` [2] ; si l'élément est présent plusieurs fois, il ne sera enlevé qu'une seule fois. Le contenu des cases est décalé, et la longeur de l'`ArrayList` diminue de 1. Si l'élément n'est pas présent, la liste n'est pas modifiée.

Et un peu plus

La classe `ArrayList` est très riche. Voyons donc quelques méthodes supplémentaires.

boolean contains(Type element) renvoie vrai si la liste contient la valeur `element`.

int indexOf(Type element) renvoie la position de `element` dans la liste, et -1 s'il n'y apparaît pas.

add(int i, Type element) ajoute la valeur `element` à la position `i`. `i` doit être inférieure ou égale à `size()` (pourquoi inférieure ou égale et non pas simplement inférieure stricte comme pour `get` ?). La fin du tableau est décalée (l'ancien élément en position `i` passe en position `i + 1`, etc.)

clear() vide la liste.

addAll(ArrayList<Type> l1) l'appel

2. Il y a une ambiguïté si on appelle `l.remove(5)` sur une `ArrayList<Integer>`. S'agit-il de supprimer la valeur 5, ou la valeur de la case numéro 5 ? Si on passe un int, c'est la première méthode `remove` qui sera appelée (on supprime l'élément qui est dans la case d'indice 5) ; si on appelle `l.remove(new Integer(5))`, c'est la valeur 5 qui sera supprimée.

`l1.addAll(l2)`

ajoute tous les éléments de l2 à la fin de l1 (concatène donc l1 et l2. l1 est modifiée, l2 ne l'est pas.

retainAll l'appel

`l1.retainAll(l2)`

enlève tous les éléments de `l1` qui ne sont pas dans `l2`.

removeAll l'appel

`l1.removeAll(l2)`

enlève tous les éléments de l1 qui sont dans l2. Après l'appel, il ne reste plus d'élément de l2 dans l1.

12.11 Un petit exemple

À titre d'exemple voici un petit programme qui simule une caisse enregistreuse. On peut éditer le ticket de caisse d'un client en y ajoutant des produits ou en les supprimant. Dans l'état actuel de nos connaissances, nous avons représenté le ticket par deux `ArrayList`. La première contient les noms des produits achetés, et la seconde contient leur prix.

```
import java.util.ArrayList;

/**
 *Une simulation de caisse enregistreuse.
 */
public class Caisse {
   public static void main(String args[]) {
      ArrayList<String> nomsArticles= new ArrayList<String>();
      ArrayList<Double> prixArticles= new ArrayList<Double>();
      boolean fin= false;
      while (! fin) {
         Terminal.ecrireStringln("votre choix : ");
         Terminal.ecrireStringln("1: ajouter un article");
         Terminal.ecrireStringln("2: supprimer un article");
         Terminal.ecrireStringln("3: terminer et afficher le total");
         int rep= Terminal.lireInt();
         if (rep == 1) { // ajout d'un produit
            Terminal.ecrireString("nom de l'article :");
            String nom= Terminal.lireString();
            Terminal.ecrireString("prix de l'article :");
            double prix= Terminal.lireDouble();
            nomsArticles.add(nom);
            prixArticles.add(prix);
            afficherTicket(nomsArticles, prixArticles);
         } else if (rep == 2) {
```

```
            // suppression du produit dans la case i.
            afficherTicket(nomsArticles, prixArticles);
            Terminal.ecrireString("numéro de l'article à enlever");
            int i= Terminal.lireInt();
            // on supprime le produit dans les deux ArrayList.
            nomsArticles.remove(i);
            prixArticles.remove(i);
            Terminal.ecrireStringln("Ticket après suppression");
            afficherTicket(nomsArticles, prixArticles);
         } else {
            fin = true;
         }
      }
      Terminal.ecrireStringln("Ticket final");
      afficherTicket(nomsArticles, prixArticles);
      // calcul du prix
      double total= 0;
      for (int i= 0; i < prixArticles.size(); i++) {
         total= total + prixArticles.get(i);
      }
      Terminal.ecrireStringln("Total " + total);
   }

   public static void afficherTicket(ArrayList<String> noms,
                        ArrayList<Double> prixArticles) {
      for (int i= 0; i < noms.size(); i++) {
         Terminal.ecrireStringln(i+ ". "+
                         noms.get(i)+
                         " prix: "+ prixArticles.get(i));
      }
   }
}
```