Langage *Java*™
Contrôle de connaissances

Thomas LEDUC

24 janvier 2001

1 Commande `sort`

Cet exercice a pour but de vous permettre de manipuler un flux de caractères en lecture (sous forme *bufferisée*), deux types de collections indexées et les exceptions en *Java*™. Il s'agit ici de reproduire le fonctionnement (très simplifié) de la commande `sort` du monde Unix. Celle-ci permet de trier les lignes d'un fichier dont le nom est passé en argument de la ligne de commande ou, à défaut, l'entrée standard, si aucun nom de fichier n'est spécifié.

Remarque importante : il ne s'agit absolument pas ici d'implémenter un quelconque algorithme de tri. Vous devez **impérativement** utiliser la méthode statique prédéfinie *sort* de la classe standard *java.util.Arrays* sous l'une des deux formes suivantes :

```
1  static void sort(Object[] unTableauQuelconque);
2  static void sort(Object[] unTableauQuelconque,Comparator laRelationDOrdre);
```

1.1 Tri lexicographique simple

Dans cette première partie, il s'agit d'écrire le corps du constructeur et des trois méthodes de la classe `Sort` dont les signatures sont données ci-après :

```
1  import java.io.*;
2  import java.util.Arrays;
3  import java.util.Vector;
4
5  public class Sort {
6      private Object[] tableauDeLignes = null;
7
8      Sort(String entrée) throws IOException { /* [...] */ }
9      public void afficher() { /* [...] */ }
10     public void trier() { /* [...] */ }
11     public static void main(String[] arguments) throws IOException { /* [...] */ }
12 }
```

Première remarque : le constructeur de la classe `Sort` se charge, en fonction de la valeur de la chaîne `entrée`, de lire le fichier correspondant ou l'entrée standard (si celle-ci est *null*) et de placer chaque ligne lue dans un tableau dynamique d'objets. Une fois la lecture terminée, on utilise la méthode *toArray* pour enrichir *tableauDeLignes*. Vous n'avez pas à traiter les *java.io.IOException*, il suffit de les "relayer" à la machine virtuelle.

Seconde remarque : la classe *java.lang.String* implémente l'interface *java.lang.Comparable*. Il est écrit dans la documentation de référence de cette interface que : "*Lists (and arrays) of objects that implement this interface can be sorted automatically by Collections.sort (and Arrays.sort).*"

1.2 Tri lexicographique simple avec suppression des "caractères blancs"

La commande `sort` (version GNU) offre une option `-b` qui permet d'ignorer les blancs en début de ligne pour la fonction de tri. Le but de cette partie est d'écrire une classe `SortILB` inspirée de la classe précédente (`Sort`) qui offre une fonctionnalité relativement proche.

Pour ce faire, il ne s'agit pas de réécrire la fonction de tri mais d'implémenter une nouvelle relation d'ordre, c'est-à-dire d'implémenter l'interface *java.util.Comparator* dont le descriptif (tiré de la documentation du *Java*™ 2 Platform Std. Ed. v1.3) vous est donné en annexe. En ce qui concerne la comparaison plus précisément, vous utiliserez la méthode *trim* de la classe *java.lang.String*.

Remarque importante : partant du principe que cette nouvelle classe est presque identique à celui de la classe précédente, il vous suffit ici de réécrire le corps de la méthode `trier` (une instruction, en l'occurence) et d'écrire le corps de la classe `ComparateurILB` (c'est le nom de la "relation d'ordre" passée à la méthode de tri).

1.3 Tri numérique simple

Comme dans le cas de la question 1.2, il s'agit ici de produire une nouvelle relation d'ordre. Celle-ci doit commencer par chercher à comparer chaque ligne (dans son ensemble) sous forme d'un flottant en double précision, puis, en cas de levée d'une *NumberFormatException*, sous forme lexicographique classique.

Remarque importante : il suffit, ici encore, de réécrire le corps de la méthode `trier` et d'écrire le corps de la classe `ComparateurNum` (c'est le nom de la "relation d'ordre" passée à la méthode de tri).

2 Gestion des arbres binaires de recherche

Le but de cet exercice est d'implémenter un arbre binaire de recherche sur les entiers, de le représenter graphiquement à l'écran et de gérer les événements tels que le clic souris pour le "déséquilibrer".

2.1 Gestion des arbres binaires

Dans le "squelette" de la classe `ABR` qui suit, il vous est demandé d'écrire le corps du constructeur et des méthodes `insérer` (insertion d'un élément dans l'arbre en respectant la relation d'ordre) et `calculProfondeur`. Vous expliquerez le rôle des variables de classe `U`, `dx` et `dy` et commenterez les lignes 22, 29, 31 et 32 du code ci-après.

```
1  import java.awt.Graphics;
2  import java.awt.Dimension;
3
4  class ABR
```

```
5   {
6       final static int U = 30;
7       final static int dx = 30;
8       final static int dy = 30;
9
10      static int profondeur = 0;
11      int valeur , x = 0 , y = 0;
12      ABR filsGauche = null , filsDroit = null;
13
14      ABR(int valeur ) { /* [...] */ }
15      public void insérer (int valeur ) { /* [...] */ }
16      private void calculProfondeur (int p ) { /* [...] */ }
17
18      public void affichageGraphique (Graphics g)
19      {
20          profondeur = 0;
21          calculProfondeur (0);
22          x = (1« profondeur ) - 1;
23          y = 0;
24          affichageGraphiqueAux(g);
25      }
26
27      private void affichageGraphiqueAux(Graphics g)
28      {
29          int delta = 1«( profondeur - y - 1);
30
31          g.drawString( Integer . toString ( valeur ),
32                        x * U + dx , y * U + dy);
33
34          if ( filsGauche != null ) {
35              filsGauche .x = x - delta ;
36              filsGauche .y = y + 1;
37              g.drawLine(x * U + dx , y * U + dy,
38                         filsGauche .x * U + dx , filsGauche .y * U + dy);
39              filsGauche .affichageGraphiqueAux(g);
40          }
41
42          if ( filsDroit != null ) {
43              filsDroit .x = x + delta ;
44              filsDroit .y = y + 1;
45              g.drawLine(x * U + dx , y * U + dy,
46                         filsDroit .x * U + dx , filsDroit .y * U + dy);
47              filsDroit .affichageGraphiqueAux(g);
48          }
49      }
50  }
```

2.2 Équilibrage d'arbre

Dans cette partie, il vous est demandé d'écrire les deux méthodes de la classe `ABR` dont les signatures suivent :

```
1       public ABR rotationÀGauche();
2       public ABR rotationÀDroite()
```

Celles-ci permettent, partant d'un état initial "équilibré" tel que celui de la figure 1, d'obtenir, par invocation de la méthode `rotationÀGauche`, l'arbre de la figure 2, ou, d'obtenir, par invocation de la méthode `rotationÀDroite`, l'arbre de la figure 3.

2.3 Gestion des événements liés à la souris

Pour implémenter cette structure d'arbre et la représenter graphiquement à l'écran, nous avons écrit la classe `ABRMain1`. Cette classe, dont nous vous livrons le code ci-après, se charge d'instancier un conteneur principal *JFrame* et de lui associer un composant-conteneur intermédiaire sous forme d'une instance de la classe interne `DéléguéSurArdoiseGraphique`, dans laquelle on procède à l'affichage graphique de l'arbre

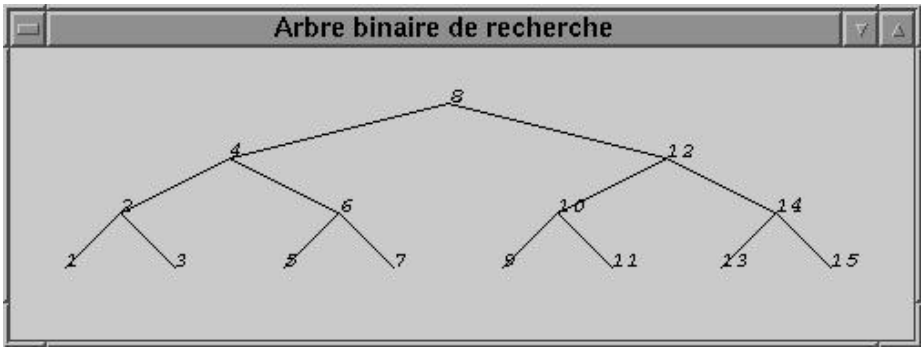

FIG. 1 – L'arbre équilibré

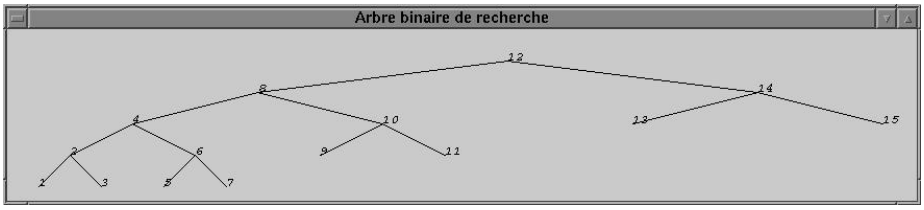

FIG. 2 – L'arbre après invocation de la méthode `rotationÀGauche`

binaire par invocation de la méthode `affichageGraphique` prédéfinie. La classe interne a aussi un rôle de délégué à la gestion des "clics souris", puisque sur réception d'un clic du bouton de gauche, elle invoque la méthode `rotationÀGauche` sur l'arbre courant, sur réception d'un clic du bouton du milieu, elle invoque la méthode `rotationÀDroite` sur l'arbre courant et, sur réception d'un clic du bouton de droite, elle affiche la profondeur de l'arbre courant.

```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;
4
5   class ABRMain1
6   {
7       ABR arbre;
8
9       class DéléguéSurArdoiseGraphique extends JPanel implements MouseListener
10      {
11          DéléguéSurArdoiseGraphique()
12          {
13              setPreferredSize (new Dimension(600,200));
14              addMouseListener(this );
15          }
16
17          public void paintComponent(Graphics g) {
18              super.paintComponent(g);
19              arbre.affichageGraphique(g);
20          }
21
22          public void mouseClicked(MouseEvent événement)
23          {
24              int drapeau = événement.getModifiers ();
25              if (( drapeau & MouseEvent.BUTTON1_MASK) != 0)
26                  arbre = arbre.rotationÀGauche();
27              else if (( drapeau & MouseEvent.BUTTON2_MASK) != 0)
28                  arbre = arbre.rotationÀDroite ();
```

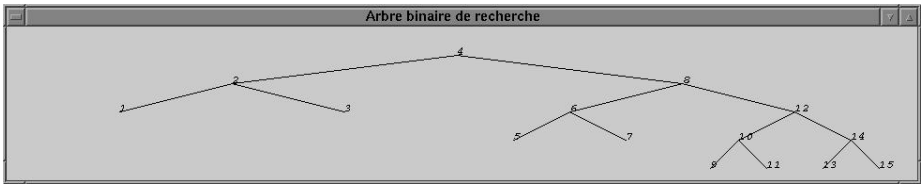

FIG. 3 – L'arbre après invocation de la méthode `rotationÀDroite`

```
29                  else if (( drapeau & MouseEvent.BUTTON3_MASK) != 0)
30                      System.out. println ("Profondeur de l'arbre : " + arbre.profondeur);
31                  repaint ();
32              }
33
34              public void mouseEntered(MouseEvent événement) {}
35              public void mouseExited(MouseEvent événement) {}
36              public void mousePressed(MouseEvent événement) {}
37              public void mouseReleased(MouseEvent événement) {}
38          }
39
40          ABRMain1(ABR arbre)
41          {
42              this.arbre = arbre;
43              JFrame laFenêtre = new JFrame("Arbre binaire de recherche");
44              laFenêtre.setContentPane(new DéléguéSurArdoiseGraphique());
45              laFenêtre.pack();
46              laFenêtre. setVisible (true);
47          }
48
49          public static void main(String [] arguments)
50          {
51              ABR arbre = new ABR(8);
52              int [] tab = new int [] {4,12,2,6,10,14,15,13,11,9,7,5,3,1};
53              for ( int i =0 ; i<tab.length ; i++)
54                  arbre. insérer (tab[i ]);
55
56              new ABRMain1(arbre);
57          }
58      }
```

Nous vous demandons, dans cette question, d'écrire une classe `ABRMain2` ayant exactement les mêmes fonctionnalités que la classe `ABRMain1`. Cette nouvelle classe doit cependant être à base non pas d'un "délégué à la gestion des événements de la souris", mais plutôt d'un "adaptateur chargé de prendre en charge cette gestion événementielle". Attention, l'héritage multiple de classes n'existe pas en *Java*™...