Langage *Java*™
Correction du contrôle de connaissances

Thomas LEDUC

24 janvier 2001

1 Commande `sort`

1.1 Tri lexicographique simple

```
1  import java.io.*;
2  import java.util.Arrays;
3  import java.util.Vector;
4
5  public class Sort
6  {
7      private Object[] tableauDeLignes = null;
8
9      Sort(String entrée) throws IOException
10     {
11         Vector vecteurDeLignes = new Vector();
12         BufferedReader fluxEntrée;
13         String tmp;
14
15         if (entrée == null)
16             fluxEntrée = new BufferedReader(new InputStreamReader(System.in));
17         else
18             fluxEntrée = new BufferedReader(new FileReader(entrée));
19
20         while ((tmp = fluxEntrée.readLine()) != null)
21             vecteurDeLignes.add(tmp);
22
23         fluxEntrée.close();
24         tableauDeLignes = vecteurDeLignes.toArray();
25     }
26
27     public void afficher()
28     {
29         if (tableauDeLignes != null)
30             for (int i=0; i<tableauDeLignes.length; i++)
31                 System.out.println(tableauDeLignes[i]);
32     }
33
34     public void trier()
35     {
36         // ce qui suit (l'appel de la méthode sort() sur le tableau
37         // "tableauDeLignes") est possible parce qu'il s'agit d'un tableau de
38         // chaînes de caractères et parce que la classe String implémente
39         // l'interface java.lang.Comparable !
40         if (tableauDeLignes != null)
41             Arrays.sort(tableauDeLignes);
42     }
43
44     public static void main(String[] arguments) throws IOException
45     {
46         Sort fluxÀTrier =
47             new Sort((arguments.length == 0) ? null : arguments[0]);
48         fluxÀTrier.trier();
49         fluxÀTrier.afficher();
50     }
```

```
51  }
```

1.2 Tri lexicographique simple avec suppression des “caractères blancs”

On se contente, dans ce cas présent, d’enrichir la méthode `trier` de la classe `Sort` qui précède, en invoquant la méthode statique *sort* de la classe *java.util.Arrays* avec un second argument de type *java.util.Comparator*. Il faut donc adjoindre à cette nouvelle classe, une “relation d’ordre” que l’on explicite par le biais d’une classe annexe `ComparateurILB` qui implémente l’interface *java.util.Comparator*.

```
1   import java.io.*;
2   import java.util.Arrays;
3   import java.util.Comparator;
4   import java.util.Vector;
5   /**
6    * sort -b  ignore leading blanks in sort fields or keys
7    */
8   public class SortILB
9   {
10      private Object[] tableauDeLignes = null;
11
12      SortILB(String entrée) throws IOException
13      {
14          Vector vecteurDeLignes = new Vector();
15          BufferedReader fluxEntrée;
16          String tmp;
17
18          if (entrée == null)
19              fluxEntrée = new BufferedReader(new InputStreamReader(System.in));
20          else
21              fluxEntrée = new BufferedReader(new FileReader(entrée));
22
23          while ((tmp = fluxEntrée.readLine()) != null)
24              vecteurDeLignes.add(tmp);
25
26          fluxEntrée.close();
27          tableauDeLignes = vecteurDeLignes.toArray();
28      }
29
30      public void afficher()
31      {
32          if (tableauDeLignes != null)
33              for (int i=0; i<tableauDeLignes.length; i++)
34                  System.out.println(tableauDeLignes[i]);
35      }
36
37      public void trier()
38      {
39          if (tableauDeLignes != null)
40              Arrays.sort(tableauDeLignes,new ComparateurILB());
41      }
42
43      public static void main(String[] arguments) throws IOException
44      {
45          SortILB fluxÀTrier =
46              new SortILB((arguments.length == 0) ? null : arguments[0]);
47          fluxÀTrier.trier();
48          fluxÀTrier.afficher();
49      }
50  }
51
52  class ComparateurILB implements Comparator
53  {
54      public int compare(Object objet1, Object objet2)
55      {
56          return ((String) objet1).trim().compareTo(((String) objet2).trim());
57      }
58      public boolean equals(Object objet)
59      {
60          return this.toString().trim().equals(((String) objet).trim());
61      }
```

```
62  }
```

1.3 Tri numérique simple

Comme dans le cas de la classe `SortILB`, on se contente ici aussi de mettre au point une “relation d’ordre” adaptée sous forme d’un comparateur qui implémente l’interface *java.util.Comparator*. Ce comparateur commence par comparer deux lignes d’un point de vue numérique (en tentant une conversion des chaînes de caractères sous forme de flottants) puis, sur réception d’une *java.lang.NumberFormatException*, effectue une comparaison lexicographique.

```
1   import java.io.*;
2   import java.util.Arrays;
3   import java.util.Comparator;
4   import java.util.Vector;
5   /**
6    * sort -n  compare according to string numerical value, imply -b
7    * sort -b  ignore leading blanks in sort fields or keys
8    */
9   public class SortNum
10  {
11      private Object[] tableauDeLignes = null;
12
13      SortNum(String entrée) throws IOException
14      {
15          Vector vecteurDeLignes = new Vector();
16          BufferedReader fluxEntrée;
17          String tmp;
18
19          if (entrée == null)
20              fluxEntrée = new BufferedReader(new InputStreamReader(System.in));
21          else
22              fluxEntrée = new BufferedReader(new FileReader(entrée));
23
24          while ((tmp = fluxEntrée.readLine()) != null)
25              vecteurDeLignes.add(tmp);
26
27          fluxEntrée.close();
28          tableauDeLignes = vecteurDeLignes.toArray();
29      }
30
31      public void afficher()
32      {
33          if (tableauDeLignes != null)
34              for (int i=0; i<tableauDeLignes.length; i++)
35                  System.out.println(tableauDeLignes[i]);
36      }
37
38      public void trier()
39      {
40          if (tableauDeLignes != null)
41              Arrays.sort(tableauDeLignes,new ComparateurNum());
42      }
43
44      public static void main(String[] arguments) throws IOException
45      {
46          SortNum fluxÀTrier =
47              new SortNum((arguments.length == 0) ? null : arguments[0]);
48          fluxÀTrier.trier();
49          fluxÀTrier.afficher();
50      }
51  }
52
53  class ComparateurNum implements Comparator
54  {
55      public int compare(Object objet1, Object objet2)
56      {
57          try {
58              double tmp1 = Double.parseDouble(objet1.toString().trim());
59              double tmp2 = Double.parseDouble(objet2.toString().trim());
```

```
60                  return ( int ) ( tmp1−tmp2);
61              }
62              catch ( NumberFormatException uneException){
63                  return (( String ) objet1 ). trim (). compareTo(((String ) objet2 ). trim ());
64              }
65          }
66          public boolean equals(Object objet )
67          {
68              try {
69                  double tmp1 = Double.parseDouble( this . toString (). trim ());
70                  double tmp2 = Double.parseDouble( objet . toString (). trim ());
71                  return ( tmp1 == tmp2);
72              }
73              catch ( NumberFormatException uneException){
74                  return this . toString (). trim (). equals ((( String ) objet ). trim ());
75              }
76          }
77      }
```

2 Gestion des arbres binaires de recherche

2.1 Gestion des arbres binaires

```
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      {
16          this . valeur = valeur ;
17      }
18
19      public void insérer ( int valeur )
20      {
21          if ( valeur <= this . valeur ) {
22              if ( filsGauche == null)
23                  filsGauche = new ABR(valeur);
24              else filsGauche . insérer ( valeur );
25          }
26          else {
27              if ( filsDroit == null)
28                  filsDroit = new ABR(valeur);
29              else filsDroit . insérer ( valeur );
30          }
31      }
32
33      private void calculProfondeur ( int p)
34      {
35          if ( filsGauche != null ) filsGauche . calculProfondeur (p+1);
36          profondeur = Math.max(p,profondeur);
37          if ( filsDroit != null ) filsDroit . calculProfondeur (p+1);
38      }
39
40      public void affichageGraphique (Graphics g)
41      {
42          profondeur = 0;
43          calculProfondeur (0);
44          x = (1« profondeur ) − 1;
45          y = 0;
46          affichageGraphiqueAux(g);
47      }
48
```

```
49      private void affichageGraphiqueAux(Graphics g)
50      {
51          int delta = 1«( profondeur – y – 1);
52
53          g.drawString( Integer . toString ( valeur ),
54                        x ∗ U + dx , y ∗ U + dy);
55
56          if ( filsGauche != null ) {
57              filsGauche .x = x – delta ;
58              filsGauche .y = y + 1;
59              g.drawLine(x ∗ U + dx , y ∗ U + dy,
60                         filsGauche .x ∗ U + dx , filsGauche .y ∗ U + dy);
61              filsGauche .affichageGraphiqueAux(g);
62          }
63
64          if ( filsDroit != null ) {
65              filsDroit .x = x + delta ;
66              filsDroit .y = y + 1;
67              g.drawLine(x ∗ U + dx , y ∗ U + dy,
68                         filsDroit .x ∗ U + dx , filsDroit .y ∗ U + dy);
69              filsDroit .affichageGraphiqueAux(g);
70          }
71      }
72
73      public ABR rotationÀGauche()
74      {
75          if ( filsGauche != null ) {
76              ABR racine = filsGauche ;
77              filsGauche = racine . filsDroit ;
78              racine . filsDroit = this ;
79              return racine ;
80          }
81          else return this ;
82      }
83
84      public ABR rotationÀDroite()
85      {
86          if ( filsDroit != null ) {
87              ABR racine = filsDroit ;
88              filsDroit = racine . filsGauche ;
89              racine . filsGauche = this ;
90              return racine ;
91          }
92          else return this ;
93      }
94  }
```

2.2 Gestion des événements liés à la souris : remplacement d'un délégué par un adaptateur

```
1   import javax.swing.∗;
2   import java.awt.∗;
3   import java.awt.event.∗;
4
5   class ABRMain2
6   {
7       ABR arbre;
8
9       class AdaptateurSurArdoiseGraphique extends MouseAdapter
10      {
11          ArdoiseGraphique ardoise ;
12
13          AdaptateurSurArdoiseGraphique(ArdoiseGraphique ardoise )
14          {
15              this . ardoise = ardoise ;
16          }
17
18          public void mouseClicked(MouseEvent événement)
19          {
20              int drapeau = événement.getModifiers ();
21              if (( drapeau & MouseEvent.BUTTON1_MASK) != 0)
22                  arbre = arbre .rotationÀGauche ();
23              else if (( drapeau & MouseEvent.BUTTON2_MASK) != 0)
```

```
24                      arbre = arbre.rotationÀDroite();
25                  else if ((drapeau & MouseEvent.BUTTON3_MASK) != 0)
26                      System.out.println("Profondeur de l'arbre : " + arbre.profondeur);
27                  ardoise.repaint();
28              }
29          }
30
31          class ArdoiseGraphique extends JPanel
32          {
33              ArdoiseGraphique()
34              {
35                  setPreferredSize(new Dimension(600,200));
36                  addMouseListener(new AdaptateurSurArdoiseGraphique(this));
37              }
38
39              public void paintComponent(Graphics g) {
40                  super.paintComponent(g);
41                  arbre.affichageGraphique(g);
42              }
43          }
44
45          ABRMain2(ABR arbre)
46          {
47              this.arbre = arbre;
48              JFrame laFenêtre = new JFrame("Arbre binaire de recherche");
49              laFenêtre.setContentPane(new ArdoiseGraphique());
50              laFenêtre.pack();
51              laFenêtre.setVisible(true);
52          }
53
54          public static void main(String[] arguments)
55          {
56              ABR arbre = new ABR(8);
57              int[] tab = new int[] {4,12,2,6,10,14,15,13,11,9,7,5,3,1};
58              for (int i=0; i<tab.length; i++)
59                  arbre.insérer(tab[i]);
60
61              new ABRMain2(arbre);
62          }
63      }
```