# Langage *Java*™

## Contrôle de connaissances - DESS et IST3/SETI

22 février 2002

Thomas LEDUC

## I. Classes *File*, *Hashtable* et *JTree* - commande *tree*

Cet exercice a pour but de vous permettre de manipuler les classes *java.io.File*, *java.util.Hashtable* et *javax.swing.JTree*. Il s'agit ici de reproduire, de manière très simplifiée (mais **graphique**), le comportement de la commande `tree`[1] du monde Unix.

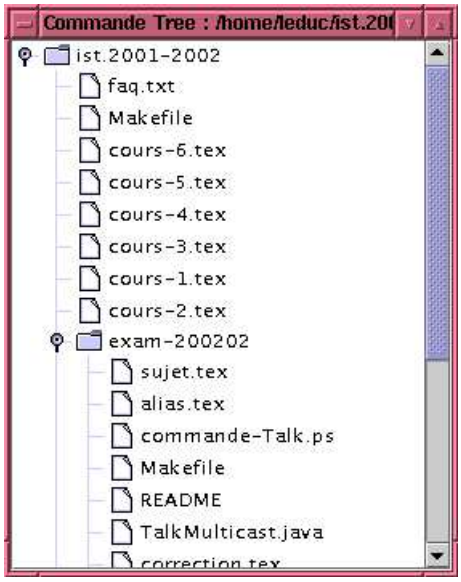


FIG. 1 – Rendu visuel de la commande `java Tree /home/leduc/ist.2001-2002`

► **Remarque importante :** il ne s'agit absolument pas ici d'implémenter un quelconque algorithme de représentation graphique d'arbre mais d'exploiter les potentialités de la classe *JTree* (qui dérive de la classe *javax.swing.JComponent*) et de son constructeur dont la signature est : `JTree(Hashtable value)`.

Ce constructeur permet d'instancier un *JTree* dont le contenu est celui de la table de hachage (qui peut être un hachage de hachages récursifs si nécessaire) passée en unique argument.

[1] Extrait du manuel : "Tree is a **recursive** directory listing program that produces a depth indented listing of files [...]. With no arguments, tree lists the files in the current directory. When directory arguments are given, tree lists all the files and/or directories found in the given directories each in turn."

► **Quelques informations** concernant deux méthodes que vous pourrez être amenés à utiliser :

```
public Object put(Object key,Object value)
-------------------------------------------
Method of class java.util.Hashtable. Maps the specified key to the specified
value in this hashtable. Neither the key nor the value can be null. The value
can be retrieved by calling the get method with a key that is equal to the
original key.

Specified by:
    put in interface Map
Overrides:
    put in class Dictionary
Parameters:
    key - the hashtable key.
    value - the value.
Returns:
    the previous value of the specified key in this hashtable, or null if it
    did not have one.
Throws:
    NullPointerException - if the key or value is null.

public File[] listFiles()
-------------------------
Method of class java.io.File. Returns an array of abstract pathnames denoting
the files in the directory denoted by this abstract pathname.

If this abstract pathname does not denote a directory, then this method
returns null. Otherwise an array of File objects is returned, one for each
file or directory in the directory. Pathnames denoting the directory itself
and the directory's parent directory are not included in the result. Each
resulting abstract pathname is constructed from this abstract pathname using
the File(File, String) constructor. Therefore if this pathname is absolute
then each resulting pathname is absolute; if this pathname is relative then
each resulting pathname will be relative to the same directory.

There is no guarantee that the name strings in the resulting array will appear
in any specific order; they are not, in particular, guaranteed to appear in
alphabetical order.

Returns:
    An array of abstract pathnames denoting the files and directories in the
    directory denoted by this abstract pathname. The array will be empty if
    the directory is empty. Returns null if this abstract pathname does not
   denote a directory, or if an I/O error occurs.
Throws:
    SecurityException - If a security manager exists and its
    SecurityManager.checkRead(java.io.FileDescriptor) method denies read access to
    the directory
```

## II. Classes *InetAddress*, *DatagramPacket* et *MulticastSocket*, gestion événementielle - un "*talk*" *multicast*

Cet exercice a pour but de vous permettre de manipuler les classes relatives au "*multicast*" du paquetage *java.net*, il reprend un exemple trouvé sur les pages W3 de Irène Charon (Professeur à l'ENST Paris). Il s'agit d'émuler une pseudo-commande *talk*, graphique, en mode *multicast*, avec un champ de saisie (champ texte éditable) et un champ de réception (champ texte non-éditable).


FIG. 2 – Rendu visuel de la commande `java TalkMulticast` lancée sur deux machines distinctes

► **Rapide mode d'emploi de l'API *multicast* du langage *Java*™ :**

– joindre un groupe :

```
1   int portXcast;
2   String hostXcast;          /* adresse IP comprise entre 224.0.0.1 et 239.255.255.255 */
3   InetAddress groupeXcast = InetAddress.getByName(hostXcast);
4   MulticastSocket socketMulticast = new MulticastSocket(portXcast);
5   socketMulticast.joinGroup(groupeXcast);
```

– émission de datagramme :

```
1   byte[] message = "Hello World!";
2   DatagramPacket hi = new DatagramPacket(message, message.length, groupeXcast, portXcast);
3   socketMulticast.send(hi);
```

– réception de datagramme :

```
1   byte[] message = new byte[512];
2   DatagramPacket recv = new DatagramPacket(message, message.length);
3   socketMulticast.receive(recv);
```

► **D'un point de vue pratique,** il vous est demandé d'écrire le corps des contructeurs et méthodes des classes `TalkMulticast` et `EmetteurEtRecepteur` dont les signatures sont données ci-après :

```
1   import java.net.*;
2   import java.awt.*;
3   import java.awt.event.*;
4   import javax.swing.*;
5
6   public class TalkMulticast extends JFrame
7   {
8       TalkMulticast () { /* à compléter... */ }
9       public static void main(String[] arguments) { /* à compléter... */ }
10  }
11
12  class EmetteurEtRecepteur extends JPanel implements Runnable,ActionListener
13  {
14      private final static int portXcast = 8084;
15      private final static int tailleMessage = 60;
16      private InetAddress groupeXcast = null;
17      private MulticastSocket laSocket = null;
18
19      private JTextField saisieEnEmission = new JTextField(tailleMessage);
20      private JTextField texteEnRéception = new JTextField(tailleMessage);
21
22      EmetteurEtRecepteur () { /* à compléter... n'oubliez pas d'activer le thread courant, après avoir enrichi l'interface graphique et rallié le
23        groupe multicast ! */ }
24
25      public void run () { /* à compléter... il s'agit ici de boucler indéfiniment sur une phase de réception de datagramme, datagramme dont on place
26        le contenu - après réception - dans le champ "texteEnRéception" à l'aide de la méthode d'instance setText(String) de la classe JTextField */ }
27
28      public void actionPerformed(ActionEvent événement) { /* à compléter... il s'agit ici "d'empaqueter" le contenu du champ de saisie (récupéré par
```

```
29          la méthode getText() de la classe JTextField) dans un datagramme et de l'envoyer au groupe multicast */ }
30    }
```