THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

COMPSCI 105 SS C 2007
Principles of Computer Science

Assignment 3 – Dealing with Trees of Files
Due 8th of February 4pm

Introduction

Files are stored in all current Operating Systems using a tree of directories (or folders). In this assignment you have to read all files and directories from a given position, show the file tree and play around with the file names.

Before you begin

Remember from tutorials 3 and 4, the Java `File` class provides the methods you need to get information about the files and directories on disk. Directories are just files with their contents being references to more files. If the `File` object is a directory the `list()` method returns a `String` array of all files in that directory; the `listFiles()` method returns a `File` array of all files in that directory. The following program prints out the names of all files (including directories) stored in the current directory (the directory the user was in when the program was run).

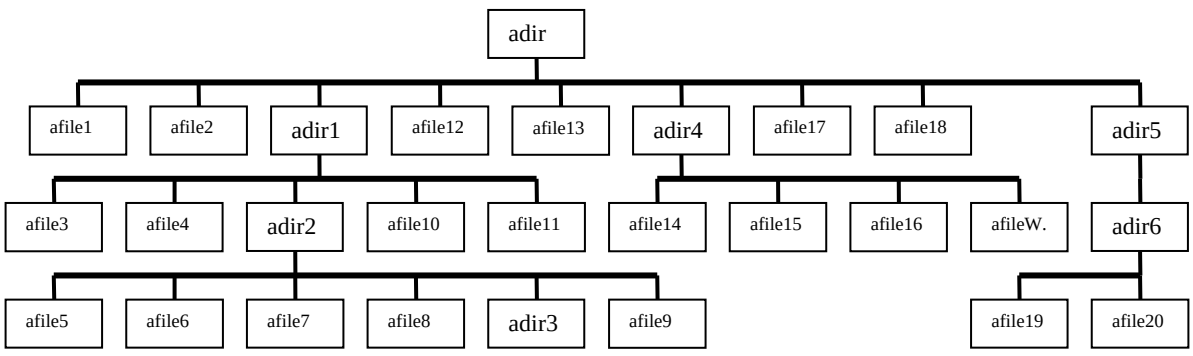
```
import java.io.File;

public class FileExample {
	public static void main(String[] args) {
		File directory = new File("."); // . is the name of the current directory
		String[] listOfFileNames = directory.list();
		for (String name : listOfFileNames) {
			System.out.println(name);
		}
	}
}
```

You can tell if a `File` object is a directory (rather than an ordinary file) with the `isDirectory()` method.
The `getName()` method returns the file name (not counting any directory names in the path).
The `getParent()` method returns the name of the directory the `File` is in.
The `getPath()` method returns the path to the File (useful in Step 4).
There are several useful constructors for the File class: `new File(String name)`, `new File(File directory, String name)`, `new File(String directory, String name)`.
For further information see the `File` API.

Step 1

Get all Java files from the assignment web site and put them in the same directory.
Run the `A3Setup` program. This creates a number of files and directories, represented schematically below:



Complete the `A31.java` program so that when it is run and the user enters `adir` at the prompt the result is:

```
adir1
adir2
adir3
...     Remaining files omitted
afile12
afile13
afile17
afile18
afile2
```

This is the complete list of all 27 file and directory names stored under the `adir` directory.

Step 2

Complete the `A32.java` program so that when it is run and the user enters `adir` at the prompt it prints the length of the longest file or directory name at each level of the directory tree. In the diagram on the previous page each level of the tree is represented by a row. For the `adir` directory the answer must be displayed as the list:

```
[7, 18, 7, 0]
```

This means that the longest filename in the first level is 7 (`afile12` and others), the longest at the second level is 18 (`afileWithALongName` this is abbreviated to `afileW.` in the diagram), the longest at the third level is 7 (`afile19` and `afile20`). The fourth level has no files and hence has a longest filename of length 0. The reason the fourth level exists is because of directory `adir3` which does not hold any files. If there was no directory in the third level the output would have been:

```
[7, 18, 7]
```

Step 3

Complete the `A33.java` program which displays the tree of files and directories descending from a directory as in the example below. If run with `adir` entered at the prompt it should produce exactly the same output as in the example.

Each level (column) of the tree has a width determined by the longest filename at that level plus 4 (for the arrow and space `" -> "`). Be careful to get the lines of `"|"` and `"-"` exactly the same. After each directory there is a line without a file name, just one or more `"|"`s.

```
adir -> adir1 ---> adir2 --------------> adir3
        |          |                     afile5
        |          |                     afile6
        |          |                     afile7
        |          |                     afile8
        |          |                     afile9
        |          |
        |          afile10
        |          afile11
        |          afile3
        |          afile4
        |
        adir4 ---> afile14
        |          afile15
        |          afile16
        |          afileWithALongName
        |
        adir5 ---> adir6 --------------> afile19
        |                                 afile20
        |
        |
        afile1
        afile12
        afile13
        afile17
        afile18
        afile2
```

Step 4

Complete the `A34.java` program which collects all files and directories descending from a directory and stores them as `Files` in a Binary Search Tree. Insert the `Files` based on their names (from `getName()`). You may assume that all names are unique.
The program then prints the names out from the Binary Search Tree in-order, pre-order and post-order. The result if run with `adir` entered at the prompt should be similar to:

```
in-order
========
adir1 adir2 adir3 adir4 adir5 adir6 afile1 afile10 afile11 afile12 afile13 afile14
afile15 afile16 afile17 afile18 afile19 afile2 afile20 afile3 afile4 afile5 afile6
afile7 afile8 afile9 afileWithALongName

pre-order
=========
adir1 adir2 adir3 afile5 afile10 adir4 adir5 adir6 afile1 afile11 afile3 afile14 afile12
afile13 afile15 afile16 afile19 afile17 afile18 afile20 afile2 afile4 afile6 afile7
afile8 afile9 afileWithALongName

post-order
==========
afile1 adir6 adir5 adir4 afile13 afile12 afile18 afile17 afile2 afile20 afile19 afile16
afile15 afile14 afile4 afile3 afile11 afile10 afileWithALongName afile9 afile8 afile7
afile6 afile5 adir3 adir2 adir1
```

The in-order output should be exactly as shown (but all on one line). The pre- and post-order outputs may be different depending on the order of insertion into the tree, but if you insert in the order produced by the `listFiles()` method it should be exactly as shown.
Then the program repeatedly asks for the name of a file to show the path to that file. If the file (or directory) exists it prints out the path as below:

```
find path to: afileWithALongName
adir\adir4\afileWithALongName
```

If the file does not exist the program prints `unknown` and terminates.

```
find path to: anotherFileWithALongName
unknown
```

Mark allocation (total 25)

The programs must calculate the output from the specified directories; hard-coding the answers will receive no marks ☺.

Style
Every file has the students upi in the comment at the top. (1 mark)
Good use of identifiers. (1 mark)
Good use of comments. (1 mark)
Good use of methods. (1 mark)
Good use of classes. (1 mark)

a31.java
Works as specified on the `adir` directory. (1 mark)
Works correctly on another directory. (1 mark)

a32.java
Works as specified on the `adir` directory. (2 marks)
Works correctly on another directory. (2 marks)

a33.java
Works as specified on the `adir` directory. (3 marks)
Works correctly on another directory. (3 marks)

a34.java
Works as specified on the `adir` directory. (4 marks)
Works correctly on another directory. (4 marks)

Submitting the assignment

Make sure your name and upi is included in every file you submit.
Use the assignment drop box to submit your `a31.java`, `a32.java`, `a33.java` and `a34.java` and any other files required to enable your programs to run. You do not need to submit the `A3Setup.java` file or the `Keyboard.java` file.
Any work you submit must be your work and your work alone – see the Departmental policy on cheating http://www.cs.auckland.ac.nz/CheatingPolicy.html.