Practicum OO — lente 2015 **Opgave 12**

Goals

This exercise consists of two programs with threads. Use the *6 steps approach to designing concurrent programs* outlined in the lectures and available on BlackBoard. After this exercise you should be able to:

- create threads to execute parts of a computation;
- synchronize threads after termination and combine their result.

1 Find File

The Java class `java.io.File` represents files and directories. We will use this class to locate a file with a given `name` somewhere in a nested directory. The program prints the full path of the file. We assume that there is exactly one file with the given name in the directory. `FileFinderSeq` is a class that executes this search recursively. The constructor sets the initial directory if it exists and throws an exception otherwise. The recursive search by the private method `find` is initiated by a call to the public method `findFile`. This will initiate the search in the give root directory.

```java
public class FileFinder {
    private final File rootDir;

    public FileFinder(String root) throws IOException {
        rootDir = new File(root);
        if (!(rootDir.exists() && rootDir.isDirectory())) {
            throw new IOException(root + " is not a directory.");
        }
    }

    public void findFile(String file) {
        find(rootDir, file);
    }

    private void find (File rootDir, String fileName) {
        File [ ] files = rootDir.listFiles();
        if (files != null) {
            for (File file: files) {
                if (file.getName().equals(fileName)) {
                    System.out.println("Found at: " + file.getAbsolutePath());
                    System.exit(0);
                } else if (file.isDirectory()) {
                    find(file, fileName);
                }
            }
        }
    }
}
```

The class is structured to be suited for a thread based version.
The class `FileFinderTest` initiates such a test for a file with name `FileFinder.java` in on disk `C`.

```java
public class FileFinderTest {

    public static void main(String[ ] args) {
        FileFinderTest fft = new FileFinderTest();
   }

    public FileFinderTest() {
        try {
            String goal = "FileFinder.java";
            String root = "C:\\";
            FileFinder ff = new FileFinder(root);
            ff.findFile(goal);
        } catch (IOException e) {
            System.err.println( e );
        }
    }
}
```

Since there can be many directories and files such a search can take much time. You task is to speed-up the search for a file by creating a new thread searching each directory encountered. In the code above you have to replace the recursive call `find(file, fileName)` by the creation of a thread that tries to find the file `fileName` in the directory `file`. This thread will generate a new thread for every directory encountered.

The number of threads created in your program may be big. Hence, you might want to limit the search by choosing a better starting directory. Moreover, you can give your Java program more executing space to accommodate more threads. By extending the heap space of the Java virtual machine it can contain more threads before it throws an exception indicating that there is no space to create an additional thread. You can set the maximum heap space of your Java virtual machine to 512 MB by the option `−Xmx512m` in the properties of your project.

2 Fibonacci Numbers

In the second program you combine the result of threads to a single result. This implies that you have to wait until the threads you created are completed by `join`.

The Fibonacci n^{th} number is 1 if $n < 2$, otherwise it is the sum of the Fibonacci numbers $n-1$ and $n-2$. The naive recursive formulation of computing Fibonacci numbers results in a huge number of functions calls. Using threads we can execute these calls concurrently. Define a class `ParFib` that computes the n^{th} Fibonacci number. For n larger then 1, a new thread is created to compute the Fibonacci number of $n - 2$ while the current thread computes the Fibonacci number for $n - 1$. When both computations are finished Fibonacci of n is computed as the sum of these results. The class `ParFib` has to implement the `Runnable` interface to be used as computation in a thread.

Also this program can create a huge number of threads. You can limit the number of threads by choosing a smaller initial number, e.g. 18. The Java virtual machine can handle more threads if you give it more heap space.

3 Deadline

Tuesday May 19, 19:00.