PRG – (E.T.S. de Ingeniería Informática) – Academic year 2017-2018

Lab practice 4 – Files and exception handling

Third session

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Contents

1 Context and previous work

This lab practice is about units 3 and 4:

Unit 3: *Elements of Object Oriented Programming: inheritance and exception handling*

Unit 4: *Input/Output: Files and Streams*

The main goal to be reached is that students become familiar with exception handling and management of input/output streams/files. In particular:

- Throw, ignore, and catch exceptions, both locally and remotely.
- Read from (or write to) text files.
- Handle exceptions related to input/output.

For it, during the three sessions of this lab practice, we are going to develop a small application to process data loaded from text files and saving the results into another file.

2 Problem statement

In this second part of the lab practice 4, its third session, students have to use the classes `CorrectReading` and `SortedRegister` from first part (first and second sessions) for developing a small application, `SortFiles`, to open all the data files located in a directory, then it will add all counts of accidents in an internal data structure for writing the results later. Results must be stored following a chronological order.

Figure 1 shows a folder (i.e. directory) created as a subfolder of the folder corresponding to project `prg`. Some data files have been copied into such subfolder.

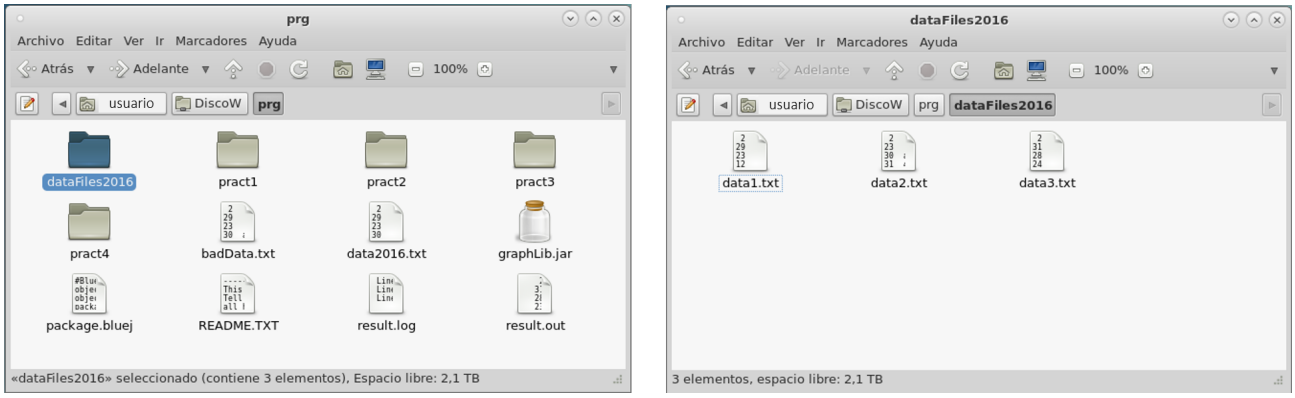

Figure 1: Directory `dataFiles2016` containing the data files `data1.txt`, `data2.txt` and `data3.txt`.

The application creates a folder specific to save the files generated during the data processing. These are both the file with the results `result.out` and the log file `result.log` with error messages. Figure 2 shows an example of such folder.

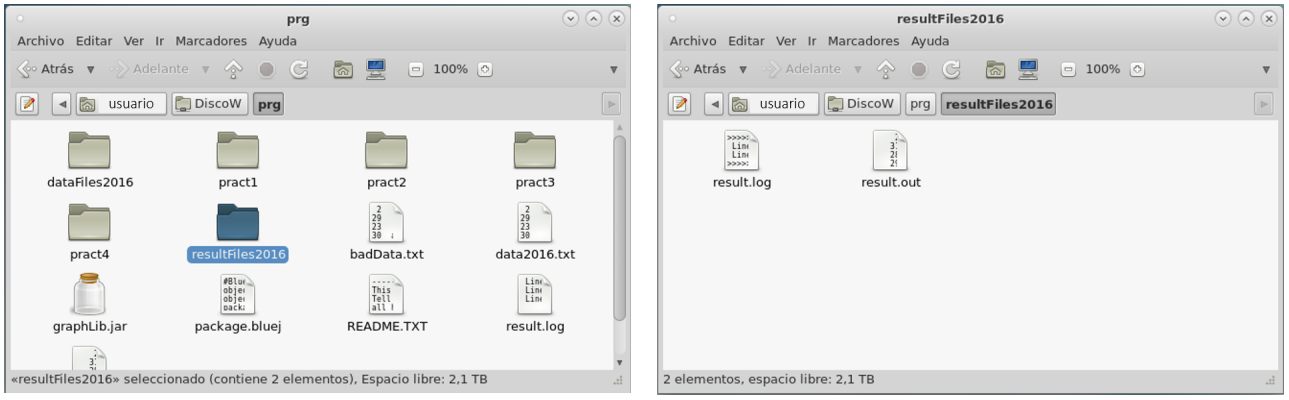

Figure 2: Directory `resultFiles2016` containing the file with the results `result.out` and the file with the errors `result.log`.

The processing task carried out by `SortFiles` will be completed just in the case all the files can be accessed with no problems, i.e. all the input files exist and can be opened, and the user have enough permissions to created all the output files.

3 How to use class `File`, an example

Activity 1: downloading of files in `pract4`

- Download from `Recursos/Laboratorio/Lab practice 4` from `PoliformaT` de `PRG`, the file with the code `SortFiles.java`, and add it into the package `pract4`.
- Create the directory `dataFiles2016` in the directory `prg`. Download from the same location the files `data1.txt`, `data2.txt` and `data3.txt`, and save them in `dataFiles2016`.

Activity 2: testing the class `SortFiles`

The class `SortFiles` downloaded in the previous activity assumes that, given a year `AAAA` for which we have data, the directory `dataFilesAAAA` already exists and contains the data files.

For processing such data files, the class has two methods:

- Method `main` is in charge of finding the directory where data files are located and creating the directory where save the results. For it, this method ask the user for the year `AAAA` and the location for the data directory, `dirParent`. Then, it creates an object of the class `File` (a file descriptor) for `dirParent/dataFilesAAAA`, and thanks to such object checks if the directory exists in the file system.

  If it exists, then, the array with the file descriptors for all the files in such directory is obtained. With such array `main` processes all the files and saves the results in the directory `dirParent/resultFilesAAAA` by invoking the method `reportedSortFiles` described next. Otherwise a message informing the user the directory doesn't exist is shown.

- Method `reportedSortFiles` creates the object of the class `SortedRegister`, where all data loaded from all files will be aggregated. With this object results will be written into the files for results and errors. This method has as parameters the list with the file descriptors (objects of the class `File`, the year corresponding to the data, and the file descriptor corresponding to the folder where to save the results.

  The body of the method is empty, it must be completed by students following the activities proposed in the next section. As it is now, the method is invoked from `main` and and do nothing but creating the folder for the results, that will remain empty.

The method `main` itself is an example of how to use objects of the class `File` for describing paths in the file system and for do modifications. In this activity you have to check the application using the directory created in the previous activity. To do it the following values must be entered by using the keyboard:

- Year 2016.
- The location for directory `dataFiles2016`.

  As it is inside the project `prg` where the class `SortFiles` is being executed, the relative path referencing the current working directory must be used. It is just `.`, the period, as Figure 3 shows.

As a result of running the preliminary test, it should be created the empty directory `resultFiles2016` inside `prg`. If it existed previously, then it is not created.

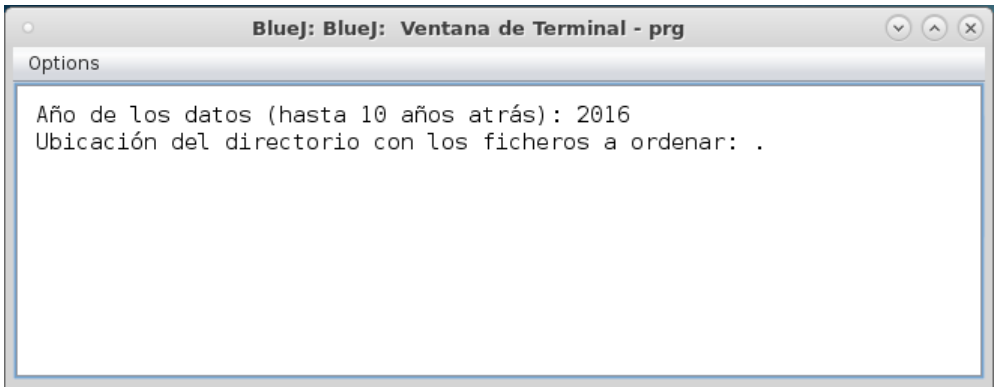

Figure 3: Example of the execution of `SortFiles`.

4 Exceptions accessing files

According to the description in the previous Section, class `SortFiles` has a method for reading the data of the problem, and generates the results. The profile of such method is:

```
public static void reportedSortFiles(File[] listF, int year, File place)
```

Figure 4 shows the results obtained after processing files `data1.txt`, `data2.txt` and `data3.txt` as they were downloaded from `PoliformaT` and stored in `dataFiles2016`.

Realise that this method receives as parameters the file descriptors in an array, the year, and the place where creating the output files for results and errors. It opens for reading all the file in the array `listF`. If output file existed then this method overrides its contents, otherwise it creates the output files.

This method is the responsible for locally solving all the checked exceptions this operations imply.

Activity 3: implementation of method `reportedSortFiles`

Exception `FileNotFoundException` is a checked exception and is thrown when programs try to open files. It must be explicitly ignored in the profile of the method or caught inside it. In this case, this exception must be caught inside the method. The action when an exception of the class `FileNotFoundException` is thrown should be to inform the user of what file generated access problems and aborted the procedure.

For achieving this, the following declaration of variables must be at the beginning of the method:

```
Scanner in = null;
PrintWriter out = null, error = null;
File f = null;
```

and the following steps must be done in the `try` block.

1. Create an object of the class `SortedRegister` for `year` referenced by the variable `c`. This object will be used for classifying all the loaded data.

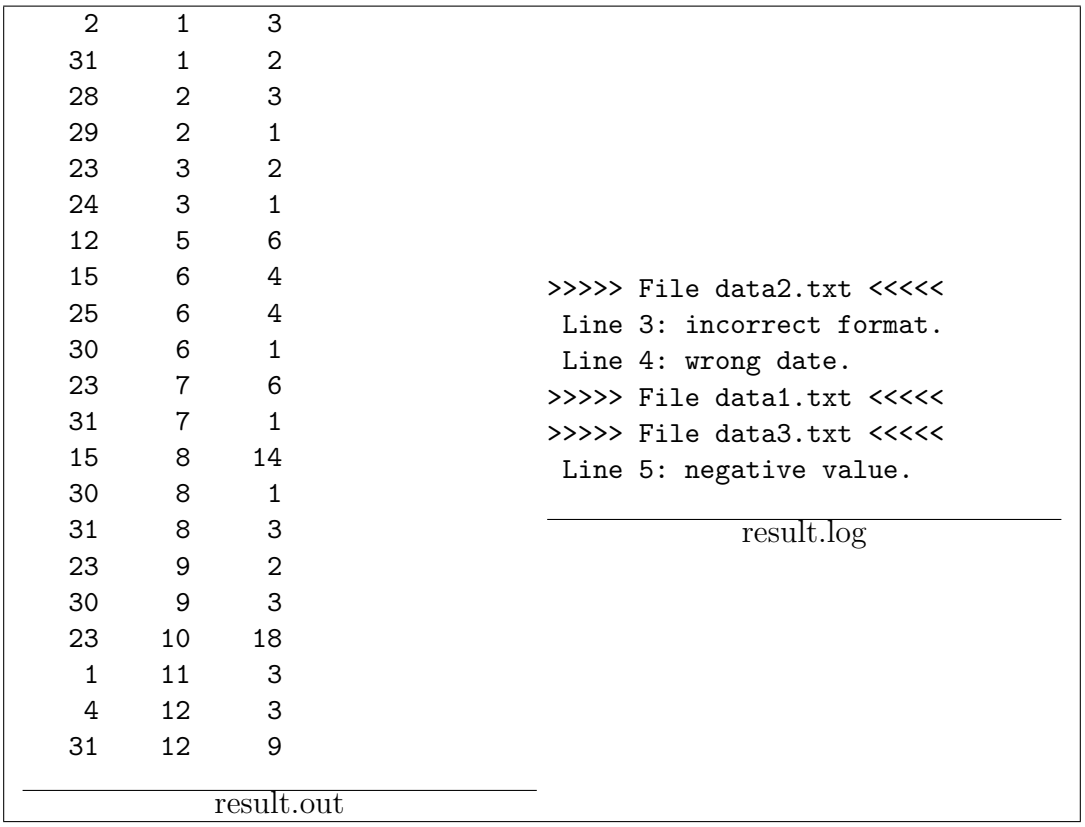

Figure 4: Results of `SortFiles` with files stored in `dataFiles2016`.

2. From the file descriptor `f = new File(place + "/result.log")`, open the file `error` in write mode.

3. Then, open for reading all the files in `listF`, and aggregate all data in `c`, and register all the errors in `error`:

```
for (int i = 0; i < listF.length; i++) {
    f = listF[i];
    in = new Scanner(f);
    error.println(">>>>> File " + listF[i].getName() + " <<<<<");
    c.add(in, error);
    in.close();
}
```

   Realise that all files in the list are closed as soon as all data contained in them are loaded.

4. Given the file descriptor `f = new File(place + "/result.out")`, open in write mode the file `out`, and write in it all the classified information stored in `c`. Class all the files opened in write mode before the method finishes.

Exceptions of the class `FileNotFoundException` can be thrown, so a catch block for it must be implemented. As variable `f` will contain the file descriptor that triggered the exception, it can be used for informing the user by showing on screen the following message: `"Process not completed: Error opening the file" + f`, and then the process must be aborted.

It should be highlighted that in case of any exception, all the files still opened must be closed, but `error`, so in a `finally` block it must be checked if `error` could be opened without any exception, i.e. it is different of `null`, then it must be closed here.

Activity 4: testing method `reportedSortFiles`
Once completed the method, the following tests must be carried out:

- Run `main` by entering all the data shown in Figure 3, and check the obtained result files match with the ones in figure 4.
- For testing an execution where an exception of the class `FileNotFoundException` is thrown, you must change the permissions of file `result.out` created during the previous execution of the class `SortFiles`. For doing this, you must change the properties of the file by means of the contextual menu you can show with the right button of the mouse when the mouse pointer is on the icon corresponding to the file. Permissions must be read only, as shown by Figure 5.

  Run again the class `SortFiles` and check the following error message in shown:

  ```
  Incomplete process: error opening ./resultFiles2016/result.out
  ```

- Run again the execution after removing the write permission to file `result.log` for checking that when `error` cannot be opened, this situation is properly managed in the `finally` block, in such case `error` must not be closed.

5 Evaluation

This lab practice belongs to the second block of lab practices of PRG, it will evaluated during the second midterm exam. The weight of this block is 60% in relation to the final lab grade. The final lab grade is the 20% of the final grade of PRG.

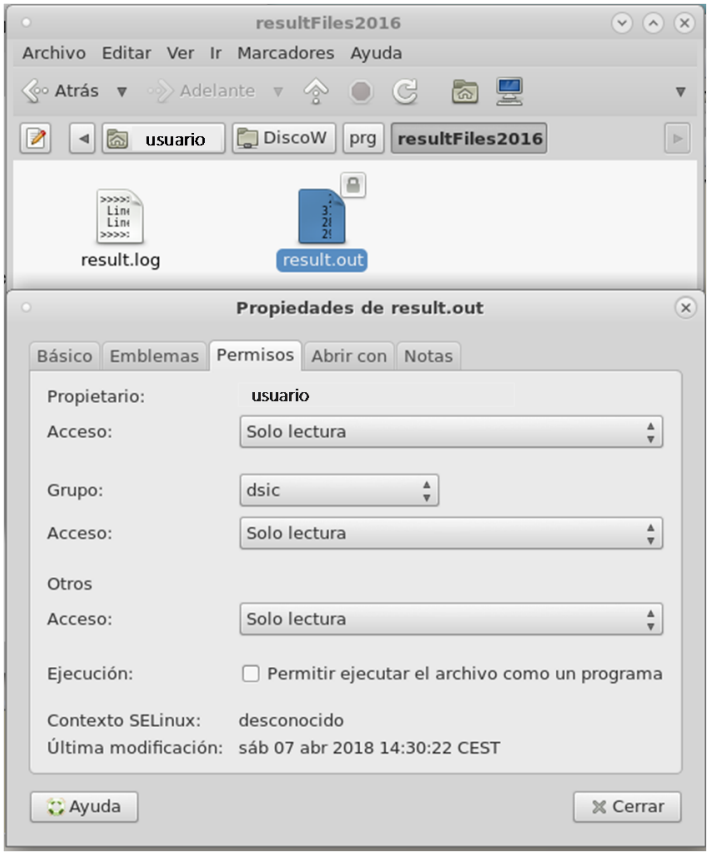

Figure 5: Modification of access permissions of `result.out`.