Processes programming exercises

1) In this first activity you should practice the use of process management commands, both in Windows and Linux. You will see them in the notes of unit 1. When you have located them you must achieve the following in both OS:

   a) Open any application (notepad or a command prompt)
   b) Show active processes by means of a command. Write down the name and ID of the recently opened application.
   c) Now, use the corresponding command to kill that process, both by name and by PID. You should see the app crash.
   d) Finally, reopen the application and change the execution priority. Does it change any value in the output of the command that shows the processes after that change?

2) Run this code for launching any program from your computer. Before compiling, fill the path to the executable file:

```
public class ProcessLauncher {
        public void execute(String path){

                ProcessBuilder pb;
                try {
                        pb = new ProcessBuilder(path);
                        pb.start();
                } catch (Exception e) {
                        e.printStackTrace();
                }

        }

        public static void main(String[] args) {
                String path = "write full path using \\";
                ProcessLauncher lp = new ProcessLauncher();
                lp.execute(path);
                System.out.println("Terminated.");
        }
}
```

   Try starting Notepad, Windows Calculator, and any other programs you want. Remember to enter, as indicated above, the full path to the executable and use the double slash \\. You should also test the application in a Linux environment (Ubuntu, for example). To do this, mount and install Ubuntu in a virtual machine, and do the same. Now, change the paths you have used before for any Ubuntu executable application.

   When you have achieved all of the above, now you should try it but changing the “ProcessBuilder” class to “Runtime” (you will have to investigate it because there are more changes to make, such as the use of the “start” method….)

3) Create a program that launches a process and, using the **`isAlive()`** method, checks to see if it is still running. The program should check every 3 seconds to see if the process is running, until it is no longer running, and then it should terminate. After each check, a message must be sent informing the status. To pause for a specific duration, **`Thread.sleep(int time_ms)`** can be used.

4) Create a program that shows the default execution directory of a process, using **`directory()`** of **`ProcessBuilder`**, and that executes the same command (for example, ls or dir) in various directories, assigned with **`directory(File directory)`** of **`ProcessBuilder`**. After this, it will show the process execution environment with ProcessBuilder's **`Map<String,String> environment`**. You have to iterate over the **`Map`** to show each entry.

5) Create a program that asks for file names until a 0 is entered (exit); for each one it will show the number of lines, words and characters obtained by calling the wc (Linux) or find (Windows) command. For each file provided, the standard input of a process created for those commands must be directed from the indicated file. The standard and error output of the created process must be redirected to those of the parent process, that is, the developed program, using the **`redirectOutput`** and **`redirectError`** methods.

6) The following example program executes a command that is supplied as a command-line argument. The output of process “p” is obtained with **`p.getInputStream()`**. A **`BufferedReader`** is built on the **`InputStream`** obtained, with the objective of obtaining the output as text and line by line. (At this point it is important that you remember what you studied about file management in the “Programming” module of the 1st course). Test it. After the code, an exercise is proposed for which the example code will be useful:

```java
import java.util.Arrays;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class LaunchProcessCaptureOutput {

  public static void main(String[] args) {

    if (args.length <= 0) {
      System.out.println("Type the correct command...");
      System.exit(1);
    }

    ProcessBuilder pb = new ProcessBuilder(args);
    try {
      Process p = pb.start();
      try (InputStream is = p.getInputStream();
              InputStreamReader isr = new InputStreamReader(is);
              BufferedReader br = new BufferedReader(isr)) {
              int codRet = p.waitFor();
              System.out.println("Execution of " + Arrays.toString(args)
                + " returns " + codRet
                + " " + (codRet == 0 ? "(execution OK)" : "(ERROR!!!)")
        );
        System.out.println("Output of process");
        System.out.println("------------------");
        String linea = null;
        while ((linea = br.readLine()) != null) {
          System.out.println(linea);
        }
        System.out.println("------------------");
      }
    } catch (IOException e) {
      System.err.println("Error during process execution.");
      e.printStackTrace();
      System.exit(2);
    } catch (InterruptedException ex) {
      System.err.println("Process interrupted...");
      System.exit(3);
    }
  }
}
```

Once tested, it is proposed to do the following:

Create a Linux program to which the path of a directory is passed as an argument. The path entered must be a folder, and if it does not exist, an error message must be displayed, and also if it exists but corresponds to a file and not a directory. Remember that you will have to use the File class. If it exists, the result of executing the “ls -lF” command on that directory should be shown, but the lines should be numbered starting with 1. The program will obtain a stream associated with the standard output of the process and then read line by line of the. To run the process you can use both **ProcessBuilder** and Runtime.