Ejercicios - gestión de procesos en Java

1) En esta primera actividad debes practicar el uso de comandos de gestión de procesos, tanto en Windows como en Linux. Los verás en los apuntes de la unidad 1. Cuando los tengas localizados, debes conseguir en ambos SO lo siguiente usando Java:
    a. Abre una aplicación cualquiera (un editor de texto, por ejemplo).
    b. Muestra los procesos activos mediante un comando. Toma nota del nombre y PID del proceso de la aplicación abierta.
    c. Ahora, usa el comando correspondiente para "matar" el proceso, tanto por el nombre como por el PID. Deberías ver cómo la aplicación se cierra abruptamente.
    d. Finalmente, vuelve a abrir una aplicación y cambia la prioridad de ejecución, ¿cambia algún valor en el resultado del comando que muestra los procesos tras ese cambio?

    NOTAS:

    1. Como el propósito del ejercicio es probar los comandos del sistema operativo, no debe usarse la clase `Process`.
    2. Los procesos se pueden matar por el nombre y por el PID ¿qué opción es mejor aunque en este caso resulte en más trabajo?

    CONSEJO:

    Probar los comandos en el SO antes de realizar el código.

2) Crea un programa que lance un proceso y, empleando el método `isAlive()`, compruebe si se sigue ejecutando. El programa debe comprobar cada 3 segundos si el proceso está en ejecución, hasta que ya no lo esté, y entonces debe terminar. Tras cada comprobación debe lanzar un mensaje informando del estado. Para hacer una pausa de duración determinada puede usarse `Thread.sleep(int tiempo_ms)`.

3) El siguiente programa de ejemplo ejecuta un comando que se proporciona como argumento de línea de comandos. La salida del proceso "p" se obtiene con `p.getInputStream()`. Sobre el `InputStream` obtenido se construye un `BufferedReader`, con el objetivo de obtener la salida como texto y línea a línea. Pruébalo. Tras el código se propone un ejercicio para el cual te será útil el código de ejemplo:

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

public class LanzaProcesoCapturaSalida {
```

```
  public static void main(String[] args) {
    if (args.length <= 0) {
      System.out.println("Introduzca el comando...");
      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("La ejecución de " + Arrays.toString(args)
               + " devuelve " + codRet
               + " " + (codRet == 0 ? "(ejecución OK)" : "(ERROR!!!)")
             );
             System.out.println("Salida del proceso");
             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 durante ejecución del proceso");
      e.printStackTrace();
      System.exit(2);
    } catch (InterruptedException ex) {
      System.err.println("Proceso interrumpido...");
      System.exit(3);
    }
  }
}
```

Una vez probado, se propone hacer lo siguiente:

Crea un programa en Linux al que se le pase como argumento la ruta o "path" de un directorio. El path introducido debe ser de una carpeta, y, si no existe o si corresponde a un fichero, debe mostrarse un mensaje de error.

Si existe, debe mostrarse el resultado de ejecutar el comando "`ls -lF`" sobre ese directorio, pero las líneas deben ir numeradas empezando por 1. El programa obtendrá un *I/O stream* asociado a la salida estándar del proceso y, después, leer línea a línea de él. Para ejecutar el proceso puede usar tanto `ProcessBuilder` como `Runtime`.

4) Crea un programa que muestre cuál es el directorio por defecto de ejecución de un proceso, usando `directory()` de `ProcessBuilder`, así como el entorno de ejecución con `environment()` que devuelve un `Map<String, String>`. Muéstralo en el formato *clave*= *valor*.

Después ejecuta un mismo comando (por ejemplo, `ls` o `dir`) en diversos directorios (2 o 3), asignados con `directory(File directorio)` de `ProcessBuilder`, mostrando la salida de cada ejecución.

Después de esto, puedes mostrar para cada proceso el entorno de cada proceso lanzado ¿son distintos?

5) Crea un programa que pida nombres de archivos hasta introducir un 0 (salir); para cada uno mostrará el número de líneas mediante una llamada al comando `wc` (Linux) o `find` (Windows). Para cada archivo proporcionado debe dirigirse la entrada estándar de un proceso creado para dichos comandos desde el fichero indicado, con `redirectInput`. La salida estándar y de error del proceso creado deben redirigirse hacia las del proceso padre, es decir, el programa desarrollado, mediante los métodos `redirectOutput` y `redirectError`. Estos métodos funcionan con descriptores de archivo.

   NOTAS:
   1. El comando equivalente a `wc` en Windows es `find /c /v ""`. Nótese que a diferencia de `wc`, solamente cuenta líneas del archivo.
   2. La clase `ProcessBuilder` proporciona un descriptor de archivo especial para heredar la entrada o salida estándar correspondiente de la clase padre: `ProcessBuilder.Redirect.INHERIT`
   3. Es frecuente usar el método `ProcessBuilder.inheritIO()` que redirecciona las entradas y salidas estándar del padre al hijo. En otras palabras, para un `ProcessBuilder pb`, hacer esto:

      ```
      pb.inheritIO();
      ```

      Equivale a:

      ```
      pb.redirectInput(ProcessBuilder.Redirect.INHERIT);
      pb.redirectOutput(ProcessBuilder.Redirect.INHERIT);
      pb.redirectError(ProcessBuilder.Redirect.INHERIT);
      ```

      ¿Cómo podría usarse este método en este ejercicio?
   4. Intenta hacer el programa de manera que sea multisistema, usando el método `System.getProperty("os.name")`

   Como información adicional, en C un proceso por defecto hereda la entrada y salida estándar y de error (stdin, stdout y stderr) del proceso padre. Entonces, en este ejercicio estaríamos emulando el funcionamiento de los procesos en C.