Actividad UF1-1.

Reporte de la actividad UF1-1 para Programación de Servicios y Procesos.

Lanzamiento de procesos. Triángulo numérico

Partíamos del siguiente programa de Java, sobre el que se nos pide crear un segundo programa que lance el proceso hasta tres veces de forma simultánea imprimiendo en un documento,además de la secuencia de números descrita, la fecha de inicializacióny la fecha de finalización de ejecución.

```
public class Triangulo
{
    public static void main(String[] args)
    {
        if (args.length == 0)
        {
            System.out.println("Se requiere un argumento");
            return;
        }
        int filas = Integer.parseInt(args[0]);
        for (int i=filas; i>=1; i--)
        {
            for (int n=1; n<=i; n++)
            {
                System.out.print(n);
            }
            System.out.println();
        }
    }
}
```

Para ajustar la clase "Triángulo" a los requisitos de la prueba, únicamente ha sido necesario importar los módulos de LocalDateTimee instanciarlos, para a continuación almacenar en una variable las fechasde inicio y la fecha de finalización del programa.

Para ello, es suficiente con llamar al método now() de la clase LocalDateTime, como podemos ver en el código a continuación:

```
LocalDateTime fechaDeInicio = LocalDateTime.now();

// Resto del código

LocalDateTime fechaFinalizacion = LocalDateTime.now();
```

Programa "Lanzador de Procesos"

La parte principal del ejercicio, y sobre la que se ha desarrollado el ejercicio, consistía precisamente en el código del programa que debería lanzar la clase Triángulo.

Para dicho programa, se ha realizado lo siguiente: -Importación de las clases correspondientes (ProcessBuilder es nativo, así que no es necesario). -Instanciación de del ProcessBuilder. -Obtención del directorio actual para poder lanzar el programa. -Creación de un ArrayList para guardar los procesos y, posteriormente, cerrarlos al mismo tiempo.

Adicionalmente, el ejercicio nos pide lanzar 3 procesos simultáneos, y que cada uno diese como parámetro adicional a la clase Triángulo los número 5, 7 y 9.

Para realizar esto, existían varias opciones.

1. Crear un array que contenga los parámetros que deseemos introducir.
2. Crear un for que comience en 5 y que tenga un "step" de +2 hasta llegar a 9.

** ¿Por qué la opción del array **

Simplemente por una cuestión de reutilizar el código. Puede que para este ejercicio estemos utilizando solamente 3 números y tengan un "step" fijo (de +2).

¿Pero y si para el próximo ejercicio nos pidiesen que los parámetros de la función fuese una serie de 20, 500 o 6000 números aleatorios? De ser este caso, iterar en el for como lo planteábos sería imposible, así que como alternativa, dejamos los parámetros dentro de Array y optamos por que se itere sobre ellos mismos.

El resultado es el siguiente:

```
    for (int i = 0; i<numeros.length; i++){
        // ProcessBuilder se inicializa para ejecutar un proceso
        ProcessBuilder pb = new ProcessBuilder("java", "-cp", classpath,
"Triangulo", numeros[i]);

        //El output irá a la carpeta de "resource"
        pb.redirectOutput(new
File("./src/main/java/resource/salida"+numeros[i]+".txt"));
        pb.redirectError(new File("./src/main/java/resource/errores.txt"));

        // Inicializar el proceso (pero aún no lo terminamos)
        Process process = pb.start();
        procesos.add(process);
        System.out.println("Lanzando proceso nº " + i);
    }
```

Inicializamos el bucle, a cada iteración del bucle se ejecuta el programa Triangulo con un parámetro diferente -guardado dentro del Array numeros[]- y se redirige el output a un fichero que, en caso de no existir, gracias a los métodos redirectOutput o redirectError.

Estos métodos tienen la virtud de que en caso de que no exista el archivo, lo crea. Y si existe, lo sobreescribe. El simple hecho de conocer este detalle nos está ahorrando el tener que escribir líneas de código adicionales para verificar si el archivo existe y manejar la excepción en caso de que no exista.

Cierre de procesos

Tras acabar el primer bucle *for*, podemos comenzar a finalizar los procesos.

```
for (Process proceso: procesos){
    proceso.waitFor();
    System.out.println("Proceso terminado");
}
```

Como los procesos están guardados en un *ArrayList*, será suficiente con llamarlos secuencialmente para cerrarlos.

Manejo de errores

Los dos bucles descritos anteriormente quedarán encerrados dentro de un try-catch para manejar las excepciones correctamente.

Concretamente, se están contemplando dos excpeciones: -IOException (que se refiere a las excepciones del ProcessBuilder en caso de un error de entrada-salida). -InterruptedException (en caso de que algo salga mal durante el método waitFor y se interrumpa este hilo durante la ejecución).

```
try{
    // RESTO DEL CÓDIGO
 } catch (IOException e) {
    throw new RuntimeException(e);

} catch (InterruptedException e) {
    throw new RuntimeException(e);
}
```