Manuel Marcote Codesido 29/09/2025

ACTIVIDAD 1

PROGRAMACIÓN DE SERVICIOS Y PROCESOS:

Se nos pide comentar un código relacionado con la programación de procesos y la redirección de flujos de E/S en Java. En este caso el proyecto se basa en un programa principal (`LanzadorPuntuacionPaso5`, versión definitiva del lanzador) que se desarrolla incrementalmente a lo largo de cuatro manifestaciones diferentes de código, haciendo ver el desarrollo iterativo de un proyecto conforme a escalabilidad, complejidad en la gestión y optimización. Junto al programa principal o padre se lanza un proceso independiente que calcula una puntuación (`CalculadoraPuntuacion`) en función de ciertos parámetros recogidos a lo largo de una partida arcade. El objetivo del proyecto a escala general trata de lanzar diferentes procesos, manejar errores asociados y gestionar la entrada y salida de los flujos de datos concurrentes.

Por otro lado, podemos analizar el proyecto en función de la separación de responsabilidades de este. El programa que maneja la lógica de negocio (`CalculadoraPuntuacion`) permite desarrollar una funcionalidad de nuestro programa de acuerdo con su operatividad real (en este caso un sistema de puntuación basado en el progreso de un jugador en una partida arcade). Por otro lado, la lógica de ejecución u orquestación desarrolla el código de funcionamiento de una aplicación Java y sus reglas.

Este lanzamiento de procesos se ha desarrollado a partir de la clase `ProcessBuilder` utilizando argumentos para definir comandos, configurando un directorio de trabajo asociado al proceso que se lanza y redireccionando sus outputs y errores bien a ficheros bien a consola.

Asimismo, se gestionan los recursos a los que se llama mediante la estructura `try-with-resources`, asegurando así que estos cesan su actividad una vez utilizados. De esta forma se evitan fugas de memoria sin tener que cerrar directamente su funcionamiento.

Asimismo, el conjunto del proyecto está sometido a una validación exhaustiva de los diferentes valores de entrada que se recogen en las distintas partes del programa. Ejemplos de ello son los distintos `if` que vemos tanto en el programa

padre como en el proceso hijo, examinando la existencia y formato del fichero de injerencia de datos (no así con el fichero de logs o salida), como la entrada de datos o argumentos en el proceso lanzado desde el programa padre.

Como último apunte es conveniente resaltar la capacidad de duplicado que tiene el programa, pues utiliza rutas haciendo uso de llamadas al sistema.

Una vez trazadas las líneas generales del proyecto, veamos el código de las diferentes clases que desarrollan el programa:

- `public class CalculadoraPuntuacion {`

En esta clase que representa el proceso hijo del programa padre se establecen una serie de validaciones tales como que los argumentos o inputs que recibe el proceso deben ser no mayores a tres diferentes cantidades. En el caso de incumplir este filtro, lanza un error a la consola de errores que más tarde veremos recoge el padre, además de iniciar un shutdown de la JVM iniciada en su proceso e indicar un código de estado.

Se inicializan las variables enteras y se procede a asignarles el valor de los argumentos recogidos por el lanzador del programa como `String` a `int`, recogiendo su excepción en caso de no resultar posible la conversión de tipo cadena a tipo entero. Como en el caso anterior, se lanza un error a consola y se indica un código de estado del proceso. Por último, se finaliza la actividad de la JVM.

El siguiente `if` permite validar que los enteros recibidos son positivos, asegurando así que no se viola conceptualmente el propósito del proceso. Se recoge el error y se finaliza la JVM, además de asignar un código de estado en la consola.

Por último, se instancia una variable `long` que recoge el cálculo de diseño de la puntuación de partida de nuestro arcade y se lanza por consola.

}

- `public class LanzadorPuntuacionPaso2 {`

A través de un scanner se inicializan tres variables de enteros una a una por consola.

Se declara un objeto de tipo `String` para acceder a propiedades externas al propio programa, dándonos acceso al ejecutable de java.

Se instancia el proceso hijo mediante un constructor `ProcessBuilder`, asignándole argumentos como variables del comando que queremos que ejecute nuestro constructor para poder iniciar el proceso hijo.

Se lanza un nuevo proceso mediante el método `start()` que lanza, además de nuestro comando una serie de configuraciones de variables de entorno al sistema operativo que alberga nuestro programa.

“Se ejecuta el proceso hijo”

Inicializamos una variable con la salida que nos devuelve el código de estado del proceso.

Se crea un `BufferedReader` para acelerar el proceso de lectura, se instancia un objeto `InputStreamReader` que toma como input stream el output del proceso. Inicializa una variable de tipo String y realiza una lectura del flujo de datos mediante el método `readLine()` asociado al `BufferedReader`.

Por último, imprime por consola el estado de código del proceso.

}

`Public class LanzadorPuntuacionPaso3 {`

Al código anterior se le añaden mejoras. En este caso se instancia un objeto de la clase `File` que apunta a un fichero entrada.txt ubicado en la ruta de proyecto.

Para validar la existencia del fichero se establece un `if` que da salida a un mensaje por consola indicando la ruta del directorio en el que no se encuentra el fichero de entrada de datos.

El `Scanner` que se utiliza en este caso está diseñado para generar logs de errores previos a su output en dirección al proceso. Hay que tener en cuenta que el scanner únicamente se centra en que el número de entradas sea justo 3 (kills, coins y tiempo vivo). Como vamos a usar un formato de entrada de datos tipo texto plano sin saltos de línea y sin separadores se usa el método `next()`.

}

Por último, en las siguientes iteraciones de la clase Lanzador se procede a redirigir el output del proceso hacia dos ficheros, inicializados sin validar su existencia, mediante los métodos de PB `redirectError()` y `redirectOutput()`.