|  |  |  |
| --- | --- | --- |
| accceso a datos | | TAREA 1 UD1 |
|  | | |
|  | | |
|  |  | |
| alumno cesur 25/26  Alejandro Muñoz de la Sierra | PROFESOR  Patricio Santiago Fernández Florez | |

introduccion

En este caso práctico, el objetivo era bastante claro: desarrollar una pequeña aplicación en Java que pudiera leer un archivo de texto, línea por línea, para luego mostrar el contenido en pantalla y calcular la longitud de cada línea, así como el número total de líneas. La idea era, sobre todo, practicar con la lectura de archivos y los flujos de entrada, incluyendo FileReader y BufferedReader, así como el manejo de excepciones. Por supuesto, el reto era aplicar buenas prácticas relacionadas con el cierre de recursos y la detección de errores.

El proceso que seguimos fue iterativo. Empezamos con el código base inicial, que era como el punto de partida del enunciado, y analizamos cómo funcionaba. Luego, poco a poco, fuimos mejorando la robustez mediante el uso de try-with-resources, realizando validaciones previas de la existencia del fichero y, por supuesto, tratando los errores adecuadamente. De hecho, nos topamos con un problema real de ejecución relacionado con los permisos de Windows y la caché de archivos. La solución llegó tras varias pruebas.

# 01

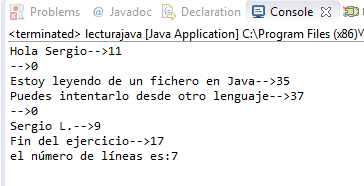
Objetivos del ejercicio

El objetivo principal era crear una clase en Java capaz de, digamos, leer un archivo de texto y mostrarlo por pantalla.

Además, queríamos mostrar, junto a cada línea, su contenido y el número de caracteres que contenía, algo así: línea --> nCaracteres.

También era importante contabilizar y mostrar el número total de líneas procesadas.

Por último, no podíamos olvidarnos de gestionar posibles excepciones, como archivo no encontrado, errores de E/S, permisos, etc.

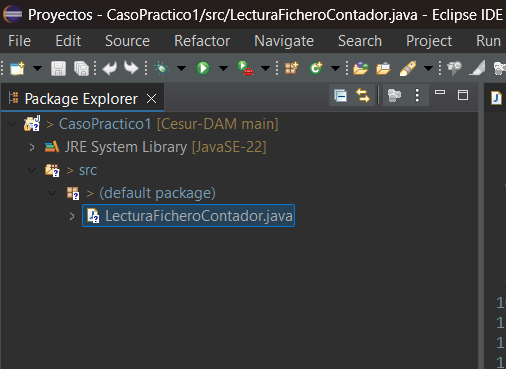


# 02

Entorno y consideraciones prácticas

El desarrollo lo realizamos en Eclipse, ejecutando el programa directamente desde el entorno de desarrollo. Usamos Java SE en una versión que fuera compatible con try-with-resources (Java 7 o superior).

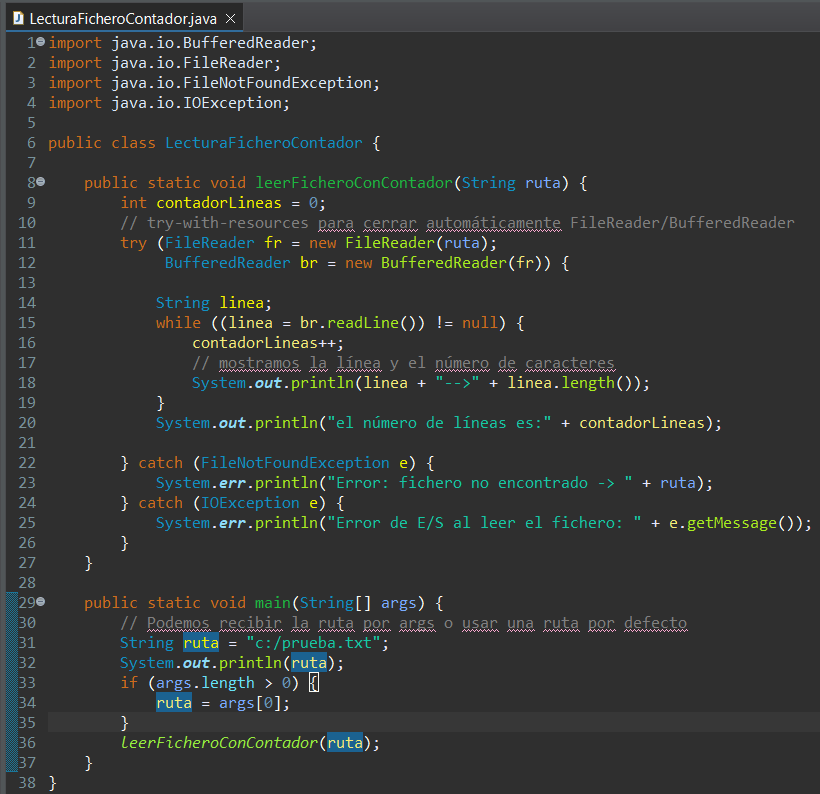
Durante las pruebas, detectamos ciertos comportamientos extraños relacionados con permisos del sistema operativo, especialmente al trabajar directamente en el directorio raíz C:\. Por este motivo, lo más recomendable es usar siempre rutas dentro del perfil del usuario, por ejemplo: C:\Users\Alex\Documentos\JavaPruebas\. Esto ayuda a evitar errores de lectura o escritura que pueden ser causados por restricciones del sistema o por herramientas de sincronización como OneDrive.



# 03

Código implementado (versión final)

La versión final del programa cumple todos los requisitos iniciales: lee un fichero de texto línea a línea, muestra cada línea junto con su longitud, y al final, indica el número total de líneas que se han procesado. Además, incluye comprobaciones previas, manejo de excepciones y mensajes informativos que, en mi opinión, facilitan bastante la depuración.



# 04

Explicación línea por línea (principales fragmentos)

Al principio, importamos las clases necesarias para el funcionamiento del programa:

**import java.io.BufferedReader;**

**import java.io.FileReader;**

**import java.io.IOException;**

Estas librerías permiten la lectura de texto y el control de errores de entrada/salida.

La clase principal, **LecturaFicheroContador**, encapsula toda la lógica del programa. Dentro de ella, el método **leerFicheroConContador**(String ruta) es el encargado de realizar la lectura.

Antes de abrir el fichero, creamos un **objeto File** con la ruta recibida. Esto nos permite comprobar si el archivo existe (**exists**()) y si tiene permisos de lectura (**canRead**()), lo que nos ayuda a evitar errores más adelante.

La lectura se realiza mediante un bloque try-with-resources:

**try (FileReader fr = new FileReader(archivo);**

**BufferedReader br = new BufferedReader(fr)) {**

Esta estructura es crucial, garantiza que los recursos se cierren automáticamente, incluso si ocurre una excepción. **FileReader** convierte los bytes en caracteres, mientras que **BufferedReader** optimiza el proceso y nos permite usar el método **readLine**().

Dentro del bucle **while** **((linea = br.readLine()) != null)**, vamos mostrando cada línea junto con su longitud, incrementando un contador que registra el número total de líneas. Al finalizar, mostramos un resumen con el total leído.

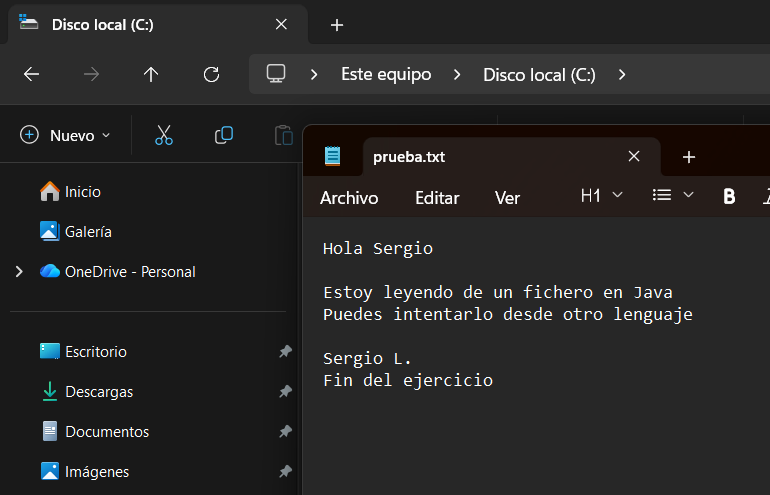
Finalmente, gestionamos posibles errores con **catch** (**IOException** e), mostrando un mensaje claro para el usuario.

# 06

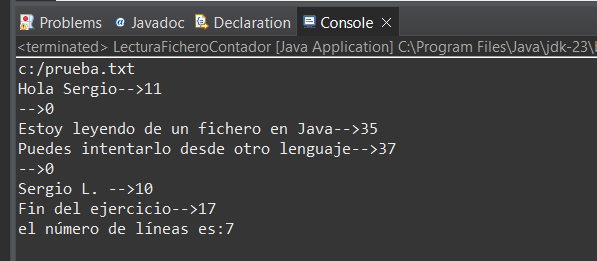
Pruebas y resultados

6.1 Fichero de prueba

Para las pruebas, usamos un archivo llamado prueba.txt. Se accede al programa a través de una ruta local, bastante accesible, la verdad.



En cuanto al resultado, lo que buscábamos era que el programa mostrara cada línea, indicando su longitud en caracteres y, al final, el número total de líneas que había leído. Hemos agregado un **print a la ruta,** para asegurarnos que el programa apunta al archivo correcto, esta línea no la cuenta para el contador de líneas del archivo a leer.



Durante las pruebas, nos topamos con algo bastante peculiar. Al modificar el archivo c:\alex\prueba2.txt, ¡el programa seguía leyendo la versión antigua! Un verdadero dolor de cabeza.

Parece que el problema radicaba en restricciones de permisos en la unidad C: y en el comportamiento algo extraño del editor, que no sobrescribía el archivo como debería. Para solucionarlo, movimos los archivos a una carpeta donde sí tuviéramos permisos de escritura. Además, usamos los métodos **exists**() y **canRead**() para asegurarnos de tener acceso, y nos cercioramos de cerrar correctamente todos los procesos de Eclipse antes de volver a ejecutar el programa. Funcionó.

# 05

conclusiones

Al final, conseguimos una aplicación que funciona bien y está limpia, cumpliendo con todos los objetivos: lee un archivo, muestra cada línea con su longitud y cuenta el total de líneas.

Durante todo el proceso, aplicamos buenas prácticas de programación, sobre todo en el manejo de recursos y excepciones. Además, resolvimos un problema real relacionado con permisos y la sincronización de archivos, lo que nos permitió experimentar con un escenario muy similar a lo que puede ocurrir en proyectos reales. De hecho, me recuerda a aquel proyecto que tuve...

En resumen, esta práctica nos ayudó a reforzar nuestros conocimientos sobre flujos de datos, rutas de archivos, gestión de errores y la importancia del entorno de ejecución. Una experiencia muy valiosa para seguir mejorando en la creación de programas robustos y seguros. Sin duda, muy útil.

# 06

referencias

<https://docs.oracle.com/javase/tutorial/essential/io/>

<https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/io/File.html>

<https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/io/BufferedReader.html>

<https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/io/FileReader.html>

<https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html>

<https://www.baeldung.com/reading-file-in-java>

<https://www.w3schools.com/java/java_files_read.asp>

<https://stackoverflow.com/questions/4716503/reading-a-plain-text-file-in-java>