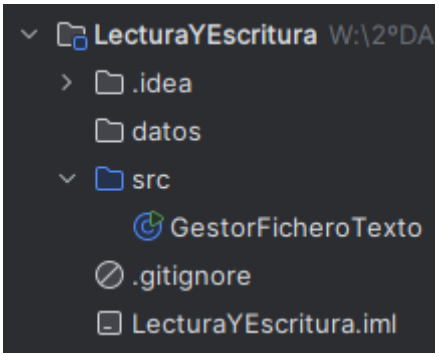
Actividad Práctica - Tema 3: Lectura y Escritura Secuencial

Iris Pérez Aparicio
2º DAM
Campus FP Emprende Humanes

ESTRUCTURA INICIAL



GestorFicheroTexto.java

```java
import java.io.*;

public class GestorFicheroTexto {
    public static void main(String[] args) {
        try {
            // Escritura
            FileWriter fw = new
FileWriter("datos/registro.txt");
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("Registro 1");
            bw.newLine();
            bw.write("Registro 2");
            bw.newLine();
            bw.write("Registro 3");
            bw.newLine();

            bw.flush();
            bw.close();
            System.out.println("Archivo escrito con éxito.");

            // Lectura
           FileReader fr = new FileReader("datos/registro.txt");
           BufferedReader br = new BufferedReader(fr);
            String linea;
```

```
            System.out.println("Contenido del archivo:");
            while ((linea = br.readLine()) != null) {
                System.out.println("> " + linea);
            }
            br.close();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Esta clase implementa los siguientes pasos en el método main:

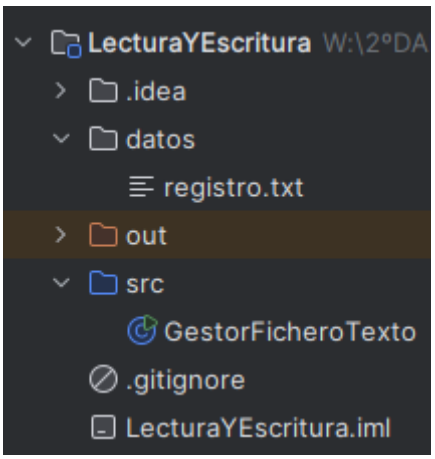
- Escribe 3 líneas de texto en un archivo datos/registro.txt (usando BufferedWriter)
- Lee el contenido de ese archivo y lo muestra por consola (usando BufferedReader)

EJECUCIÓN

```
Archivo escrito con éxito.
Contenido del archivo:
> Registro 1
> Registro 2
> Registro 3

Process finished with exit code 0
```

ESTRUCTURA FINAL



Preguntas de reflexión

1. ¿Qué ocurre si se vuelve a ejecutar el programa sin cambiar el nombre del archivo?

```
Archivo escrito con éxito.
Contenido del archivo:
> Registro 1
> Registro 2
> Registro 3

Process finished with exit code 0
```

El archivo se vuelve a crear, sobrescribiendo el archivo anterior.

2. ¿Cómo podrías añadir texto sin borrar el contenido anterior?

Añadiendo un segundo parámetro (**true**) al constructor de FileWriter, esto hace que el archivo se abra en modo append (agregar).

Modifico el código:

```
FileWriter fw = new FileWriter("datos/registro.txt", true);
```

Pruebo a ejecutar de nuevo el programa:

```
Archivo escrito con éxito.
Contenido del archivo:
> Registro 1
> Registro 2
> Registro 3
> Registro 1
> Registro 2
> Registro 3

Process finished with exit code 0
```

3. ¿Qué diferencias observas si eliminas el `BufferedWriter` y usas solo `FileWriter` ?

- FileWriter escribe directamente en el archivo cada vez que llamas al método write(), mientras que BufferedWriter usa un búfer (memoria intermedia) y escribe en disco solo cuando el búfer se llena o llamas a flush() o close().

- FileWriter no tiene el método newLine() para el salto de línea (se hace con "\n")

- Dato extra: Si escribes mucho texto o muchas líneas, BufferedWriter es más rápido y eficiente. Para textos cortos, FileWriter funciona bien, pero será menos eficiente si haces muchas escrituras pequeñas.

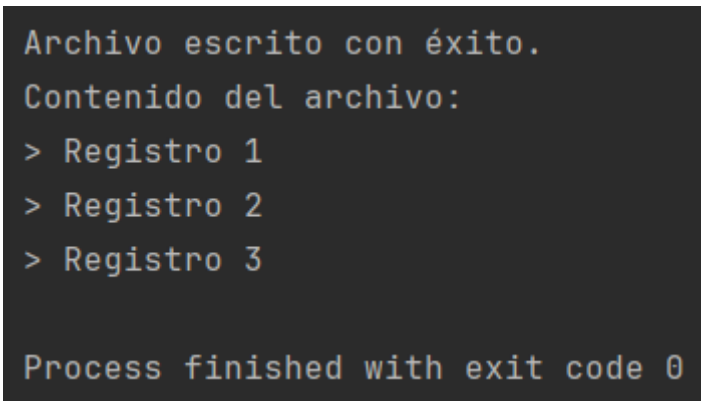
  Código usando solo FileWriter:

```java
import java.io.*;

public class GestorFicheroTextoSoloFW {
    public static void main(String[] args) {
        try {
            // Escritura SOLO CON FILEWRITER (sin
BufferedWriter)
            FileWriter fw = new
FileWriter("datos/registro.txt");
            fw.write("Registro 1\n");
            fw.write("Registro 2\n");
            fw.write("Registro 3\n");
            fw.close();
            System.out.println("Archivo escrito con
éxito.");

            // Lectura
            FileReader fr = new
FileReader("datos/registro.txt");
            BufferedReader br = new BufferedReader(fr);
            String linea;
            System.out.println("Contenido del archivo:");
            while ((linea = br.readLine()) != null) {
                System.out.println("> " + linea);
            }
            br.close();
        } catch (IOException e) {
            System.out.println("Error: " +
e.getMessage());
        }
    }
}
```

EJECUCIÓN
(El resultado es el mismo)



4. ¿Por qué es importante cerrar los buffers después de usarlos?

Cerrar los buffers después de usarlos es muy importante porque permite liberar los recursos que el programa está utilizando, como la memoria y el acceso al archivo. Además, si no se cierran correctamente, puede ocurrir que los datos que hemos escrito no se guarden bien en el archivo, ya que los buffers almacenan temporalmente la información hasta que se cierran. Esto ayuda a evitar errores, archivos corruptos o pérdidas de información, y es una buena práctica de programación para que todo funcione correctamente y el sistema no se quede con recursos ocupados innecesariamente.