Actividad 1.1: Configuración entorno y Tratamiento fichero.txt

1. Utilidad de las clases del paquete java.io

- **File:** Representa una ruta (un archivo o un directorio) en el sistema de archivos, no el contenido del archivo en sí. Se utiliza para consultar las propiedades del archivo (tamaño, permisos), comprobar si existe, crearlo, eliminarlo o listar el contenido de directorios. No sirve para leer ni escribir texto.
- **FileReader:** Es una clase orientada a caracteres utilizada para leer el contenido de un archivo de texto. Lee el flujo de datos byte a byte o carácter a carácter directamente del sistema de archivos.
- **BufferedReader:** Es un envoltorio (*wrapper*) para otras clases de lectura (como FileReader). Su función es almacenar en memoria caché (un *buffer*) un bloque grande de caracteres. En lugar de leer carácter por carácter del disco, proporciona métodos como readLine() para leer líneas completas de texto de forma optimizada.
- **FileWriter:** Es la clase análoga a FileReader, pero para escritura. Permite escribir secuencias de caracteres directamente en un archivo de texto.
- **BufferedWriter:** Funciona de manera similar a BufferedReader. Envuelve a un Writer (como FileWriter) y acumula los datos a escribir en un búfer interno. Solo realiza la escritura física en el disco cuando el búfer se llena o cuando se invoca el método flush().
- **PrintWriter:** Es una clase avanzada de escritura que proporciona métodos de formato convenientes (print(), println(), printf()). Permite escribir representaciones en texto de distintos tipos de datos (int, double, objetos) sin tener que convertirlos explícitamente a cadenas de texto primero.

2. FileReader vs. BufferedReader

La diferencia principal radica en **cómo interactúan con el disco duro**. **FileReader accede al disco físico por cada carácter (o pequeño bloque) que lee, mientras que BufferedReader lee un bloque grande de una sola vez y lo guarda en la memoria RAM** (el *buffer*).

¿Cuál es más eficiente y por qué?

BufferedReader es drásticamente más eficiente. Las operaciones de entrada/salida (I/O) en disco son las tareas más lentas que puede realizar un ordenador. Al usar un búfer, BufferedReader **minimiza el número de accesos físicos al disco**. Si necesitas leer 1000 caracteres, FileReader podría consultar el disco 1000 veces, mientras que BufferedReader hará una sola consulta, traerá un bloque a memoria y te servirá los caracteres desde la RAM.

3. FileWriter, BufferedWriter y PrintWriter

Aunque las **tres sirven para escribir texto, operan en distintos niveles de abstracción**:

- **FileWriter**: Es la herramienta de bajo nivel. Escribe caracteres en el archivo, pero sufre el mismo problema de rendimiento que FileReader: **accesos constantes al disco si no se envuelve.**
- **BufferedWriter:** Es la herramienta de optimización. No añade métodos de formato nuevos, simplemente agrupa los caracteres que le pasa el programador y hace escrituras masivas en el disco para mejorar el rendimiento.
- **PrintWriter:** Es la herramienta de conveniencia. Su propósito es facilitar la vida del programador permitiendo escribir texto formateado (como se hace por consola con System.out.println). A menudo, PrintWriter se utiliza envolviendo a un BufferedWriter que, a su vez, envuelve a un FileWriter.

4. Gestión de errores y adición de texto (FileWriter vs. PrintWriter)

¿Cuál lanza excepciones de tipo IOException?

FileWriter. Sus constructores y métodos (como write()) lanzan obligatoriamente IOException, lo que exige que el programador maneje la excepción con un bloque try-catch o la declare en la firma del método con throws.

¿Cómo gestiona los errores PrintWriter?

PrintWriter suprime las excepciones de tipo IOException en sus métodos de escritura. Si ocurre un error al escribir, el programa no se detiene ni lanza la excepción; en su lugar, PrintWriter activa una bandera interna (un *flag* booleano). El programador debe llamar manualmente al método checkError() para saber si algo falló durante el proceso de escritura.

¿Cuál usarías para añadir/escribir texto al final de un fichero (Append)?

Técnicamente, **debes usar FileWriter** (a menudo envuelto por PrintWriter o BufferedWriter).

PrintWriter por sí solo no tiene un constructor directo para abrir un archivo en modo *append* pasándole solo un String o File. La responsabilidad de decidir si un archivo se sobrescribe o si se añade texto al final recae en la clase que se conecta directamente al disco. Por lo tanto, la sintaxis correcta para añadir texto de forma eficiente y cómoda requiere instanciar ambas:

```
// FileWriter en modo append (true) envuelto por PrintWriter
PrintWriter out = new PrintWriter(new FileWriter("archivo.txt", true));
```