MANEJO DE FICHEROS EN JAVA

Erik Amo Toquero

18 DE FEBRERO DE 2025

ERIK AMO TOQUERO

Contenido

Introducción

Las aplicaciones normalmente leen, procesan, editan y eliminan datos, datos que tienen que ser almacenados fuera de la aplicación para ser utilizados posteriormente y que no se vayan con la memoria. Hay distintas opciones: las **bases de datos** y las **lecturas de ficheros** (mejor bases de datos pero ficheros se suele usar)

Flujos de entrada de información

Los objetos sobre ficheros se encuentran en java.io.

La lectura de información se hace con la clase FileReader

```
FileReader fr = new FileReader("src/resources/fichero.txt")
```

Hay que tener en cuenta que los FileReader leen los ficheros directamente del String de su constructor. Este string puede tener dos tipos de rutas de acceso:

- Ruta del proyecto: La que sale de la raíz del proyecto: “src/resources/fichero.txt”
- Ruta del sistema: La que coge la lectura de la ruta de acceso del sistema operativo:
    - **Windows:** “C:\\Users\\Usuario\\Desktop\\Fichero.txt”
    - **Linux:** “/home/user/Escritorio/Fichero.txt”

La apertura del fichero causa problemas de tipo IOException, por lo que hay que controlarlo por medio de la captura de excepciones try-catch o try-catch-resources.

Los FileReader cuentan con dos métodos:

- read(): Devuelve un valor entero que representa el carácter codificado. Cuando la lectura no lee más caracteres, devuelve un valor nulo.

```
String texto = "";
FileReader lector = new FileReader("src/resources/texto.txt");
int c = lector.read();
while (c != -1){ //Mientras encuentre un caracter al leer
    texto = texto + (char) c;
    c=lector.read();
}
System.out.println(texto);
```

- close(): Cierra el FileReader. Es una buena práctica para evitar consumir demasiados recursos durante la ejecución de un programa

```
lector.close(); //Cerrar lector una vez acabe
```


BufferedReader

El FileReader tarda mucho y pide mucha cosa para ser usado, por lo que se utilizan más otros objetos para leer, como el BufferedReader, que permite leer más de un carácter a la vez. En su declaración, se tiene que meter dentro el FileReader que utilizará como referencia.

```
try(FileReader fr = new FileReader("src/resources/fichero.txt");
    BufferedReader bReader = new BufferedReader(fr)){
    texto ="";
    String linea = bReader.readLine();
    while (linea!=null){
        texto += linea + "\n";
        linea = bReader.readLine();
    }
    System.out.println(texto);
}catch (IOException e){
    System.out.println(e.toString());
}
```

Tiene dos métodos:

- read(): Igual que el lector normal, da un número codificado de un carácter.
- readLine(): Devuelve un String con toda una línea del texto (Hasta el salto de línea)

Scanner

Normalmente se utiliza para leer lo que un usuario introduce en la consola por teclado, pero se puede hacer lo mismo con un fichero ya escrito. En este caso, tendremos que añadir en su declaración un elemento de tipo File("url"). Tiene que ir guardado en un control de excepciones para evitar problemas.

```
try (Scanner sc = new Scanner(new File("src/resources/fichero.txt"))){

}catch (IOException e){
    System.out.println(e.toString());
}
```

Tiene varios métodos, pero se puede resumir en dos:

- hasNext___(): Para saber si hay más de lo que se quiere leer
- next____(): Lee el siguiente valor que interesa al usuario.

Cabe recalcar que tiene que ir con Int para enteros, Double, Float, Boolean y Line para strings.

```
while (sc.hasNext()){
    System.out.println(sc.nextLine());
}
```


Flujos de salida

Cuando se quiere escribir, se recurre a los flujos de salida o FileWriter, que genera el stream de escritura hacia el fichero. Si el archivo seleccionado no existe, lo crea. Necesita controlar la excepción IOException.

Cuenta con dos constructores:

- FileWriter(String ruta): Crea el flujo y sobreescribe todo el archivo
- FileWriter(String ruta, boolean append): Crea el flujo y decide si sobreescribe lo ya escrito (false), o continua desde la última línea (true).

```
try {
    FileWriter fw = new FileWriter("src/fichero.txt", true);
    FileWriter fw2 = new FileWriter("src/fichero.txt");

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

Para mejorar la eficacia, se usa la clase BufferedWriter, que dejará cerar un buffer y así escribir más rápido, entre otras cosas. Cuenta con varios métodos:

- write(int carácter): Escribe un carácter escrito en Unicode
- write(String cadena): Escribe una cadena entera
- newLine(): Salto de línea
- flush(): Escribe todo lo que tenga pendiente y vacía el buffer
- close(): Cierra el buffer

```
try(BufferedWriter bw = new BufferedWriter(new
FileWriter("src/resources/datos.txt", true))){
    System.out.println("Escribe tu nombre");
    String nombre = sc.nextLine();
    System.out.println("Escribe ahora tu edad");
    int edad = sc.nextInt();
    bw.write(nombre + " " + edad);
    bw.newLine();
    System.out.println("Introducido correctamente");
}catch (IOException e){
    System.out.println("Error de archivo");
}
```