Gestión de Entrada/Salida

Índice:

1. Introducción.

Podemos definir un flujo como un conjunto de bytes o caracteres que salen o entran en nuestra aplicación. Si esos bytes o caracteres los provoca nuestra aplicación, serán de salida. Sin embargo, si la aplicación toma datos externos para procesarlos, estamos hablando de flujos de entrada.

Existen por tanto flujos de entrada y de salida, además ambos tipos pueden ser de bytes o de caracteres.

Las clases asociadas a estos flujos están en el paquete java.io.

2. Flujos de salida de caracteres

Para gestionar los flujos de salida de caracteres existen una gran cantidad de clases. Estudiaremos algunas.

La gestión de un flujo de salida es la siguiente:

- Abrir el flujo
- Mientras existan datos, escribirlos en el flujo
- Cerrar el flujo.

2.1. La clase FileWriter.

La clase FileWriter permite escribir carácter a carácter en un archivo. El archivo se crea en la carpeta del proyecto.

En el ejemplo, el constructor de File recibe el nombre del archivo a crear (String). Existe también la posibilidad de pasarle un objeto File que representa a un archivo (o directorio). Estudiamos la clase File más adelante en esta unidad.

```
FileWriter fw = null;
String cadena= "El dinero no puede comprar la vida";
try {
    fw = new FileWriter("frase.txt");
    for (char c: cadena.toCharArray()) {
        fw.write(c);
    }
} catch (IOException ex) {
    System.out.println(ex.getMessage());
} finally {
    if (fw!=null){
        try {
            fw.close();
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

2.2. La clase BufferedWriter.

Esta clase permite escribir líneas en un flujo. El archivo se crea en la carpeta del proyecto. Observa que se usan los métodos write() y newLine(), para escribir y saltar de línea respectivamente. Al final del proceso (finally), se cerrará el flujo.

```
List<String> cadenas = Arrays.asList(new String[] { "En un lugar de la Mancha",
        "de cuyo nombre no quiero acordarme,", "no ha mucho tiempo que vivía ",
        "un hidalgo de los de lanza en ",
        "astillero, adarga antigua, rocín flaco y galgo corredor." });

BufferedWriter bw=null;
try {
    bw = new BufferedWriter(new FileWriter("quijote.txt"));
    for (String string : cadenas) {
        bw.write(string);
        bw.newLine();
    }

} catch (IOException e) {
    // TODO Auto-generated catch block
    System.out.println(e.getMessage());
} finally {
    try {
        System.out.println("Archivo creado");
        bw.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        System.out.println(e.getMessage());
    }
}
```

3. Flujos de entrada de caracteres

Para gestionar flujos de entrada debemos seguir el patrón:

- Abrir el flujo
- Mientras existan datos que leer, leer datos del flujo y procesarlos
- Cerrar el flujo

3.1. La clase BufferedReader

Mediante esta clase leemos líneas de un flujo de entrada. El método readLine lee líneas del flujo de entrada y si no quedan más datos que leer devuelve nulo (null).

Descarga desde aquí el archivo para pruebas.

```
BufferedReader bufferEntrada=null;
try {
    bufferEntrada = new BufferedReader(
            new FileReader("src/recursos/quijote.txt",
                    standardCharsets.UTF_8));

    String linea="";
    while((linea=bufferEntrada.readLine())!=null) {
        System.out.println(linea);
    }

} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    System.out.println(e.getMessage());
} catch (IOException e) {
    // TODO Auto-generated catch block
    System.out.println(e.getMessage());
} finally {
    try {
        bufferEntrada.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        System.out.println(e.getMessage());
    }
}
```

4. Flujos de salida y entrada de bytes

Los flujos de salida de bytes dan lugar a archivos binarios, es decir, archivos que no pueden ser leídos por editores de texto.

Centraremos el estudio de estos flujos, tanto de entrada como de salida, en archivos que guarden objetos.

Las clases que necesitamos están en el paquete java.io y serán:

- Para la salida (creación de archivos):
  - FileOutputStream y ObjectOutputStream.
  - Para escribir objetos en un archivo se utiliza el método writeObject(objetop) de la clase ObjectOutputStream.
  - Con close() se cierra el ObjectOutputStream.
- Para la entrada (lectura de archivos):
  - FileInputStream y ObjectInputStream.
  - El método readObject() de la clase ObjectInputStream permite leer objetos de un archivo binario.
  - Con close() se cierra el ObjectInputStream.

En el ejemplo creamos un archivo “personas.dat” en el que se guardarán objetos de la clase Persona (id y nombre). Ojo, esta clase debe implementar la interfaz java.io.Serializable, para que las instancias (objetos) de la misma puedan guardarse en un archivo binario.

Importante, al leer datos de un archivo binario, se genera la excepción IOException al alcanzar el fin del archivo.

En moodle, tenéis las clases ejemplos para probar un proyecto con objetos de la clase Persona en un archivo.

5. Rutas a archivos.

Mediante la interfaz Path podemos hacer referencia a una ruta del sistema de archivos, ya se trate de sistemas Windows, Linux o Mac. Un objeto que implemente esta interfaz contiene el nombre del archivo y la lista de directorios usada para construir la ruta. Estas rutas pueden existir o no.

Con Path podemos, obtener información de un path, unir dos paths, compararlos, …

La clase Paths nos permite crear un path, mediante su método get(). En la imagen se observan varias formas de crear un path.

Ambas, clase e interfaz, se encuentran en el paquete java.nio.

```
Path path1 = Paths.get("documentos", "palabras.txt");
Path path2 =  FileSystems.getDefault().getPath("documentos2", "marzo", "datos.txt");
Path path3 = Paths.get(System.getProperty("user.home"), "a2020", "marzo", "gastos.txt");

System.out.println("Ruta 1: " + path1.toAbsolutePath().toString());
System.out.println("Ruta 1: " + path2.toAbsolutePath().toString());
System.out.println("Ruta 1: " + path3.toAbsolutePath().toString());
```

De esa forma la ruta 1 es D:\Programacion2122\unidad9\documentos\palabras.txt

Entre los métodos asociados a esta clase tenemos: toString(), getFileName(), getParent(), getRoot(), …

6. La clase Files.

También pertenece al paquete java.nio y contiene una gran cantidad de métodos estáticos para gestionar operaciones con archivos y directorios.

Algunos de sus métodos. Ojo son estáticos, se llaman mediante Files.metodo(), es decir, nombre de la clase . nombre del método. No es ningún objeto en que llama al método.

- isDirectory(). Devuelve true si la ruta apunta a un directorio.
- isRegularFile(). Devuelve true si se trata de un archivo regular o normal (no temporal).
- isSameFile(path1, path2). Devuelve true si ambos paths apuntan al mismo fichero.
- newBufferedWriter(path). Crea un flujo de salida de caracteres sin necesidad de usar FileWriter.
- copy(pathOrigen, pathDestino, StandarCopyOption). Copiar archivos.
- move(pathOrigen, pathDestino, StandarCopyOption). Mover archivos
- deleteIfExists(path). Eliminar archivo.
- size(). Devuelve el tamaño de un archivo en bytes.
- Files.newDirectoryStream(path). Obtener los archivos y directorios de un path. Devuelve un stream de directorios (un stream es una secuencia).

```
DirectoryStream<Path> directoryStream =
        Files.newDirectoryStream(ruta);

for (Path path : directoryStream) {
    if (Files.isDirectory(path))
        System.out.println("Directorio: " + path);
    else
        System.out.println("Archivo: "+ path);

}
```

- readAllLines(path). Lee todas las líneas de un archivo y las devuelve en formato List<String>, es decir, un ArrayList. Está pensado para archivos que no sean de un gran tamaño.