

# Manejo de Ficheros

Todos sabemos de sobra que los programas usan variables para tratar la información. Por ejemplo, usamos variables para:

- Almacenar valores de entrada: al leer del teclado.
- Resultados calculados: al calcular una suma.
- Valores intermedios: para almacenar precios en una lista.

Pero, todas estas variables desaparecen al terminar el programa. No existe una **persistencia** de la información. Por tanto, hace falta emplear otra serie de herramientas para conseguir que esa información permanezca en el tiempo.

Definimos la **persistencia** como la capacidad de una aplicación para guardar información de forma que puedan ser recuperada y utilizada posteriormente, incluso después de que la aplicación se haya cerrado.

La forma más sencilla de almacenar información es mediante el uso de ficheros o archivos que se almacenan en un dispositivo físico.

## Ficheros en Java

En Java existen diferentes tipos de ficheros. Se diferencian por las clases que se utilizan para representarlos y manipularlos. Estas clases están ubicadas en el paquete **java.io** y pertenecen a la propia JDK, por lo tanto, nos basta con importarlos en nuestro programa.

En la asignatura de Programación de 1ro ya hemos trabajado con ficheros de forma básica. Por tanto, ya sabemos que para acceder a un fichero pueden ocurrir un montón de problemas:

- Ruta de fichero incorrecta
- Nombre del fichero incorrecto
- Espacio de disco insuficiente
- ... y un largo etcétera.

Dado que no tiene sentido preparar el código para todo lo que pueda salir mal, lo que haremos será emplear **excepciones** para gestionar todos los posibles errores de nuestro código.

Mencionar a demás que existen varias formas de acceder a la información de un fichero. En estos apuntes se va a ver la forma más teórica; no obstante, con una simple búsqueda en Google podrás encontrar diversas soluciones para un mismo problema (como siempre, vamos).

## La Clase File

Cada vez que queremos usar ficheros, en algún momento usaremos la clase **File**. Esta clase se encuentra en el paquete **java.io** y representa de forma abstracta a un fichero o un directorio.

Existen varios constructores disponibles en la clase **File**, entre ellos:

```
File file1 = new File ("prueba.txt");

File file2 = new File ("c://trastero/", "prueba");
```

Nótese que ninguno de los constructores genera un fichero o un directorio en ningún sitio. Simplemente son objetos en memoria con una serie de rutas (path) y nombres hasta el momento en el que decides hacer algo con ellos. De hecho, es posible incluso que contengan rutas erróneas, incompletas o inexistentes.

Para trabajar con la clase File, tenemos una serie de métodos. No vamos a listar todos, pero los más relevantes son:

```
file.createNewFile()    // Crea un nuevo fichero vacío si no existe a partir de File
file.isDirectory()      // Retorna true si File se refiere a un directorio
file.canRead()          // Retorna true si puedo leer en el fichero (permisos)
file.canWrite()         // Retorna true si puedo escribir en el fichero (permisos)
file.getAbsolutePath()  // Retorna el path absoluto del File referenciado
file.delete()           // Elimina el fichero o directorio
```

Puedes encontrar más información sobre la clase File acudiendo a las API de Java. Te dejo el enlace para la [API](#) de Java 22. Consulta la correspondiente a la versión de Java que estás utilizando.

Por tanto, la clase **File** se emplea para manipular un fichero o directorio.

Sin embargo, lo que NO podemos hacer con File es algo tan elemental como escribir o leer información de ese fichero o directorio.

## Ejercicio Propuesto - 1

---

Crea un programa en Java que acceda a una carpeta específica de tu disco duro (p.ej. c:/trastero/) y liste todos los ficheros, carpetas y subcarpetas del mismo. Indica también si es un fichero o un directorio, o si se puede escribir y/o leer del mismo. Acuérdate de comprobar siempre si el fichero o directorio existe antes de hacer nada con él.

No necesitas más que la clase File.

## Leer y escribir de un File

Java dispone de varias clases para trabar con File dependiendo de cómo queramos acceder a él:

Si queremos acceder a él según su tipo de contenido, tenemos:

- Ficheros de caracteres: Los famosos ficheros de texto.
- Ficheros binarios: Literalmente, guardamos en ellos 0s y 1s.

Según cómo queramos acceder a ellos:

- Ficheros secuenciales: Se empieza por el principio, se va avanzando.
- Ficheros aleatorios: Se puede acceder a cualquier punto del fichero.

Por lo tanto, tendremos que conocer todos y decidir cuál es el método más adecuado para nuestro programa.

## Flujos

En Java, un **flujo (stream)** es una secuencia de datos que se puede leer o escribir de forma ordenada. Para Java, todo son flujos. No existe un teclado, una pantalla, un fichero o una base de datos: hay flujos. Y un flujo puede ser cualquier cosa. Esto se diseñó así desde el principio dado que los primeros desarrolladores de Java fueron capaces de prever (allá en 1992) hasta cierto punto el auge de Internet y la conexión entre diversos dispositivos.

En resumen: en el fondo, java no sabe de dónde está leyendo o escribiendo, porque para Java todo es lo mismo. Lo que es una buena idea.

Java ofrece dos tipos de flujos principales:

- Flujos de caracteres: Usados para ficheros de texto.
- Flujos binarios: Para datos binarios (PDF, imágenes, sonido...)

En la asignatura de PSP de 2º veremos que, para leer y escribir de un **socket** se utilizan también flujos de forma similar a ficheros.

Hoy en día existen muchas librerías que facilitan estas tareas de acceso a datos (en general). Aunque en esos casos no hagamos un uso directo de los flujos, dichas librerías en el fondo te están ocultando la complejidad de su manejo, de forma que no tengas que preocuparte por ellos.

## Texto Plano - Escritura

La clase **FileWriter** nos permite definir un flujo que escribe caracteres en un fichero de modo secuencial. Esta clase hereda de la clase **Writer**. Ofrece dos constructores principales:

```
// Flujo de salida, coloca el texto a final del fichero
FileWriter fileWriter = new FileWriter (path, true);

// Flujo de salida, sobrescribe el fichero
FileWriter fileWriter = new FileWriter (path, false);

// Flujo de salida sobre un objeto File
FileWriter (file).
```

Obviamente, el método principal de **FileWriter** es **write ()**.

Un ejemplo de escritura secuencial de un fichero de texto podría ser:

```
public void writeFile() {
    String cesta[] = { "Peras", "Manzanas", "Plátanos" };
    File file = null;
    FileWriter writer = null;
    try {
        // Create a File
        file = new File("example.txt");

        // Create a FileWriter
        writer = new FileWriter(file);

        // Loop and write the text
        for (int i = 0; i < cesta.length; i++) {
            writer.write(cesta[i]);
            writer.write("\n");
        }
    } catch (Exception e) {
        System.out.println("An error occurred.");
    } finally {
        // Close the writer
        try {
            if (writer != null)
                writer.close();
        } catch (IOException e) {
            // Nothing to do here...
        }
    }
}
```

Este código escribe el contenido de cesta en example.txt, línea a línea.

## Texto Plano - Lectura

La clase **FileReader** nos permite definir un flujo que lee caracteres en un fichero de modo secuencial. Esta clase hereda de la clase **Reader** y ofrece dos constructores principales:

```
FileReader fileReader = new FileReader (path)

FileReader (file)
```

Obviamente, el método principal de **FileReader** es **read ()**, aunque la lectura es un proceso un poco más complicado.

Un ejemplo de lectura secuencial de un fichero de texto podría ser:

```
public void readFile(String[] args) {
    File file = null;
    FileReader reader = null;
    try {
        // Create a File
        file = new File("example.txt");

        // Create a FileWriter
        reader = new FileReader(file);

        // Loop to read a single character until
        // file is empty (-1)
        int i = 0;
        while ((i = reader.read()) != -1) {
            System.out.println( (char) i + "==">" + i);
        }
    } catch (Exception e) {
        System.out.println("An error occurred.");
    } finally {
        // Close the reader
        try {
            if (reader != null)
                reader.close();
        } catch (IOException e) {
            // Nothing to do here...
        }
    }
}
```

Este código escribe por pantalla el contenido del fichero `example.txt`, carácter a carácter. No es que sea particularmente útil, pero es lo que hay...

---

## Ejercicio Propuesto - 2

Crea un programa que permita escribir y leer un fichero `diario.txt`. Entre las opciones tiene que estar: añadir al final, reescribir, y leer.

## Texto Plano - BufferedWriter

Trabajar con ficheros 'a mano' siempre ha sido costoso para los programas. Por tanto, se decidió implementar nuevas clases que hacían uso de un **buffer** para una mejorar los procesos de lectura y escritura.

La clase **BufferedWriter** escribe texto de forma eficiente en un flujo de salida. Funciona almacenando internamente el texto en un búfer antes de escribirlos en el fichero, lo que reduce la cantidad de operaciones de escritura y mejora el rendimiento, especialmente cuando tiene mucho texto.

Esta clase hereda de la clase **Writer** y ofrece un constructor principal:

```
BufferedWriter BufferedWriter = new BufferedWriter (fileWriter);
```

Obviamente, el método principal de BufferedWriter es **write ()** y **writeLine ()**.

Un ejemplo de escritura secuencial de un fichero de texto podría ser:

```
public void writeFileBuffered () {
    String cesta[] = { "Peras", "Manzanas", "Plátanos" };
    File file = null;
    FileWriter writer = null;
    BufferedWriter buffer = null;
    try {
        // Create a File
        file = new File("example.txt");

        // Create a FileWriter
        writer = new FileWriter(file);

        // Create a BufferedWriter
        buffer = new BufferedWriter(writer);

        // Loop and write the text
        for (int i = 0; i < cesta.length; i++) {
            buffer.write("Contenido cesta " + i + ": " + cesta[i]);
            buffer.newLine();
        }
    } catch (Exception e) {
        System.out.println("An error occurred.");
    } finally {
        // Close the buffer
        try {
            if (buffer != null)
                buffer.close();
        } catch (IOException e) {
            // Nothing to do here...
        }

        // Close the writer
        try {
            if (writer != null)
                writer.close();
        } catch (IOException e) {
            // Nothing to do here...
        }
    }
}
```

## Texto Plano - BufferedReader

La clase **BufferedReader** lee texto de forma eficiente en un flujo de entrada. Funciona almacenando internamente el texto en un búfer, lo que nos permite (por ejemplo) leer texto de golpe hasta que nos encontramos con un salto de línea.

Esta clase hereda de la clase **Reader** y ofrece un constructor principal:

```
BufferedReader bufferedReader = new BufferedReader (fileReader);
```

Obviamente, el método principal de **BufferedReader** es **readLine ()**.

Un ejemplo de lectura secuencial de un fichero de texto podría ser:

```
public void readFileBuffered () {
    File file = null;
    FileReader reader = null;
    BufferedReader buffer = null;
    try {
        // Create a File
        file = new File("example.txt");

        // Create a FileWriter
        reader = new FileReader(file);

        // Create a BufferedWriter
        buffer = new BufferedReader(reader);

        // Loop to read a row
        String linea;
        while((linea = buffer.readLine())!=null) {
            System.out.println(linea);
        }
    } catch (Exception e) {
        System.out.println("An error occurred.");
    } finally {
        // Close the reader
        try {
            if (reader != null)
                reader.close();
        } catch (IOException e) {
            // Nothing to do here...
        }
    }
}
```

Este código escribe por pantalla el contenido del fichero example.txt, línea a línea. Toma el carácter de salto de línea ("/n") para diferenciar las filas.

## Ejercicio Propuesto - 3

---

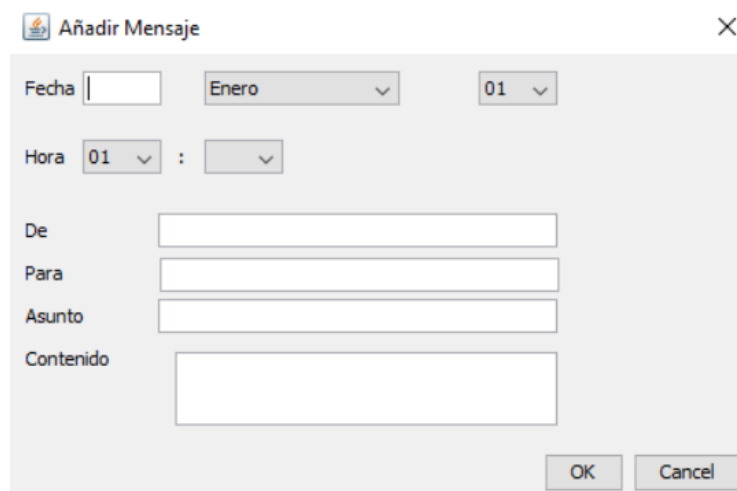
Usando **BufferedWriter** y **BufferedReader**, crea un programa que permita escribir y leer un fichero diario.txt. Entre las opciones tiene que estar: añadir al final, reescribir, y leer.

## Práctica - 1

Crea un sistema de mensajería visual similar al mail que permita a varios usuarios comunicarse entre ellos. Existirá una ruta de red en la que habrá un fichero llamado **Mensajer.txt**. Cuando un usuario abra el programa, podrá hacer:

- Cargar mensajes: Se cargarán todos los mensajes en un ArrayList en memoria del Programa.
- Guardar mensaje: Se volcarán en el fichero todos los mensajes guardados en memoria.
- Nuevo mensaje: Se añadirá un mensaje nuevo en memoria.
- Imprimir mensaje: Carga en un JTable todos los mensajes de memoria.

La ventana para nuevo mensaje debe ser similar a esta:



Y el JTable igual que este:

De	Para	Fecha	Hora	Asunto	Contenido
Eneritz	Aner	2021-03-04	10:45:53	Quedada	A las 5 de la tarde ...
Aner	Eneritz	2021-01-31	12:15:53	Respuesta	Ok.

Ten en cuenta que tienes que mostrar todos los mensajes de error adecuados, comprobar que el fichero existe, controlar las excepciones, etc.