MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Flujos y ficheros

6

ÍNDICE

OBJETIVOS

Comprender el concepto de flujo.

Conocer los diferentes flujos existentes.

Entender el concepto de fichero y las diferentes formas de acceso a los mismos.

Conocer las jerarquías de flujos.

Dominar las diferentes excepciones asociadas a flujos.

/ 1. Introducción y contextualización práctica

En esta unidad, vamos a tratar los conceptos de flujo y fichero. Vamos a ver qué es un flujo, para qué sirve y los diferentes tipos de flujos que nos podemos encontrar, y que darán lugar a los ficheros que todos conocemos.

También, vamos a ver las diferentes formas de acceso a ficheros que existen, con sus ventajas e inconvenientes correspondientes.

Aprenderemos las varias formas de referenciar a los ficheros, lo que se conoce como rutas.

Por último, analizaremos el tratamiento de ficheros, donde observaremos que este es susceptible a provocar errores, y estudiaremos cuáles son estos.

Fig. 1. Control de información con flujos y ficheros.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Audio intro. "Persistencia de datos"

https://bit.ly/31Vengm

/ 2. Concepto de flujo

Hasta el momento, nuestros programas siempre han funcionado de la misma forma: solicitando información al usuario para que la introduzca por teclado, procesando dicha información y mostrando por pantalla el resultado del procesamiento.

Una vez que nuestros programas se han ejecutado y cerrado, toda esa información que se ha introducido se pierde, ya que se almacenaba en la memoria RAM del ordenador y el programa, al cerrarse, se desaloja de memoria, borrando toda información relativa al mismo.

Pero ¿qué ocurre si queremos que esa información introducida no se pierda al cerrar el programa? Puede ser que la necesitemos para futuras ejecuciones.

Es ahora donde entran en juego los flujos a ficheros. **Un flujo de datos es una vía de comunicación entre dos puntos, un origen y un destino.** Una buena analogía a un flujo de datos es un sistema de **tuberías** con entradas (sumideros) y salidas (grifos) de agua (información), o una **carretera**, en ella circulan vehículos que podemos imaginar que son la información que se transmite a través del flujo. Esta información transmitida podrá ir en dos direcciones, siendo estas los flujos de entrada y de salida de información.

Mediante el uso de flujo, podremos crear ficheros en disco, pudiendo almacenar y recuperar información de ellos cada vez que lo necesitemos.

Los flujos nos van a aportar una gran ventaja, que es la **persistencia de datos.** Con esta nos referimos a que se **va a poder guardar información en disco**, y gracias a esto, cuando cerremos el programa, **la información no se va a perder**, siempre y cuando la hayamos almacenado en un fichero antes de cerrar, permitiéndonos recuperarla cuando lo necesitemos para volver a operar con ella.

Los flujos a ficheros también tienen un gran inconveniente, y es que el acceso a disco, tanto para almacenar como para recuperar información, es muy ineficiente en cuanto a tiempo se refiere, es decir, se tarda mucho en almacenar y recuperar información de un fichero.

Lo más recomendable es realizar los mínimos accesos para, así, paliar la ineficiencia del acceso.

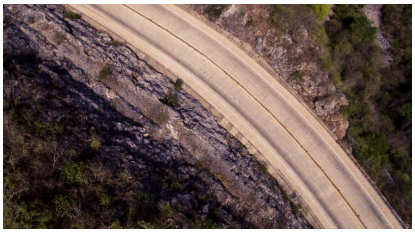

Fig. 2. Símil de la carretera y los flujos.

/ 3. Tipos de clases de flujo

Cualquier lenguaje de programación actual provee el uso de flujos de datos para poder operar y utilizar la persistencia de datos en sus programas.

En nuestro caso, el lenguaje de programación Java nos provee de una serie de clases que forman una jerarquía para poder tener acceso a ficheros e implementar la persistencia de datos.

Esta jerarquía se va a dividir en dos partes:

- Clases para lectura de ficheros.
- Clases para escritura de ficheros.

La ventaja de utilizar flujos de datos es que estos van a ser independientes del tipo de dispositivo que utilicemos, es decir, que va a ser indiferente si estamos trabajando con un flujo que obtenga información de teclado o un flujo que recoja información de un fichero en una determinada ruta. Si lo hemos implementado de forma correcta, no debería haber problema en ello, habiendo liberado al programador que lo implementa de saber con qué está actuando.

Esta vinculación del sistema físico con el tipo de flujo la hará la propia máquina virtual de Java, es decir, el JDK.

Resumiendo, quien tendrá que encargarse de comunicarse con el sistema operativo va a ser el fujo, de esta manera, no vamos a tener que cambiar mucho en nuestra aplicación para hacer que un programa que originalmente obtenía información de teclado, ahora la obtenga de un fichero de datos almacenado en disco.

Esta gran capacidad es una característica vital para un lenguaje de programación multiplataforma como es Java.

Todas estas clases se encuentran en el paquete *java.io*.

Fig. 3. Comunicación.

/ 4. Caso práctico 1: "Almacenando datos"

Planteamiento: Pilar y José acaban de recibir un nuevo encargo para desarrollar una nueva aplicación. Esta deberá poder almacenar de forma persistente los datos de una serie de empleados de un taller mecánico.

El jefe de proyecto les ha encargado que diseñen un pseudocódigo que realice la acción de almacenar los empleados en el menor tiempo posible.

Nudo: ¿Cómo crees que podrán realizar dicha tarea nuestros amigos? ¿Habrá que realizar control de excepciones para el almacenamiento de los datos de los empleados?

Desenlace: La forma más sencilla y rápida de llevar a cabo el almacenamiento de datos en un fichero que nos permita tener un almacenamiento persistente es ir recorriendo dichos datos (los empleados del taller en este caso) uno a uno, e ir almacenándolos a la vez que se recorren.

Una cosa que tenemos que tener muy en cuenta, y no podemos olvidar, es que toda operación con ficheros podrá lanzar una serie de excepciones que tenemos que controlar mediante un bloque try-catch, siendo de vital importancia esto, ya que si no lo hacemos, el programa podrá generar un fallo en tiempo de ejecución, y además de cerrarse, nos arriesgaremos a perder toda la información que estábamos almacenando, ya que la que no se escribiera, se habrá perdido, y la que se pudiera haber escrito, puede perderse porque el fichero de guardado puede corromperse al no cerrarse de forma adecuada.

Un ejemplo del pseudocódigo que se ha solicitado podría ser este:

```
// Abrimos el fichero de los datos
fichero = abrirFichero()
// Recorremos un array con los datos
desde 0 hasta cantidad_datos
    // Obtenemos el dato a guardar
    empleado = obtenerEmpleado()
    // Guardamos los datos del empleado
    escribirEnFichero(fichero, empleado)
// Una vez terminemos cerramos el fichero
cerrarFichero(fichero)
```

Fig. 4. Pseudocódigo para almacenar los empleados.

/ 5. Flujos predeterminados

Todos los lenguajes de programación que han existido y existen se han provisto de flujos para poder introducir y mostrar información, lo que se conoce como los flujos de entrada y salida estándar. Esto permite que se pudieran leer datos de teclado y mostrar datos por pantalla.

Ya sabemos qué son los flujos y que hay principalmente dos tipos, los de entrada y los de salida de información.

Hasta el momento, hemos estado utilizando flujos de entrada y salida sin ser conscientes de ellos.

Los flujos que hemos estado utilizando son los estándares, tanto para la entrada como para la salida:

Tipo de flujo	Instrucción
Estándar de salida	*System.out*
Estándar de salida de errores	*System.err*
Estándar de entrada	*System.in*

Tabla 1. Tabla de flujos estándar en Java.

El flujo estándar de salida de errores sirve para mostrar un error por pantalla. Su resultado es exactamente igual que la salida normal (*System.out*) salvo que las letras aparecerán en color rojo.

Un ejemplo puede ser:

System.err.println("Error inesperado");

El flujo estándar de entrada de datos *System.in* es el que hemos estado utilizando desde el principio con la clase *Scanner* y sirve para poder leer datos de teclado.

Cuando utilicemos los flujos para abrir un fichero, tendremos que indicarle a qué fichero hacen referencia, ya que si no, abrirán los flujos de datos por defecto.

Audio 1. "Entrada y salida de información"
https://bit.ly/32XPm3q

/ 6. Flujos basados en bytes

Los flujos basados en *bytes* están orientados tanto a la lectura como a la escritura de información organizada en *bytes*.

Este tipo de flujos, por su configuración y funcionamiento, son ideales para gestionar ficheros que contengan información estructurada de forma binaria.

El lenguaje de programación Java provee dos clases que nos van a proporcionar toda la funcionalidad necesaria para poder tratar con este tipo de flujos:

- *InputStream*
- *OutputStream*

Con la clase ***InputStream***, podremos crear flujos de entrada de datos a ficheros basados en *bytes*, es decir, podremos **leer** información de ellos.

Con la clase ***OutputStream***, podremos crear flujos de salida de datos a ficheros basados en *bytes*, es decir, podremos **escribir** información en ellos.

A los objetos instanciados de estas clases, podremos indicarles dónde se encuentra el fichero al que queramos abrir el flujo, es decir, tendremos que indicarles la ruta del fichero para que puedan acceder a él.

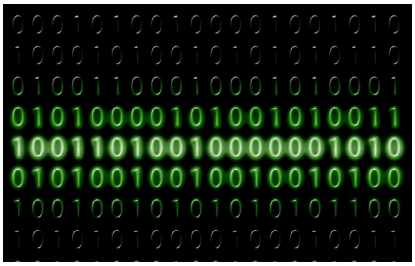

Fig. 5. Leer y escribir bytes.

Al igual que podemos indicar la ruta del fichero al que queremos abrir un flujo, podremos indicarles, también, los tipos predeterminados, por lo que tendremos lectura y escritura 'normal'.

Las operaciones con este tipo de flujos siempre van a constar de cuatro partes:

1. Abrir el flujo al fichero.
2. Comprobar que el flujo se ha abierto correctamente.
3. Operar con el fichero.
4. Cerrar el fichero.

Vídeo 1. "Jerarquía de clases de flujos 1"
https://bit.ly/355LNKY

/ 7. Flujos basados en caracteres

Con los flujos orientados a *bytes*, podemos tener la funcionalidad necesaria para poder trabajar con la información de una forma muy sencilla, pero estos tienen una limitación importante, y es que no pueden trabajar con caracteres Unicode, ya que estos utilizan 2 *bytes* para su codificación.

En principio, esta consideración puede parecer no tener demasiada importancia, pero debido a esto, se consideró el crear un nuevo tipo de flujo de datos, el flujo basado en caracteres.

Este nuevo tipo de flujo de datos ofrece la funcionalidad necesaria para poder trabajar directamente con caracteres, pudiendo operar con Unicode y todas sus respectivas variaciones.

Dependiendo de la funcionalidad que necesitemos, existen muchos tipos de flujos de datos.

Para la implementación de los flujos de caracteres, Java nos va a proporcionar dos clases base:

1. *Reader*.
2. *Writer*.

Estas dos clases pueden tratar estos flujos de carácter en carácter, lo cual hace que su tratamiento sea muy lento, y más teniendo en cuenta la ineficiencia del acceso a disco.

Para solucionar esto, lo que haremos será crear otros flujos a partir de estos dos. Estos nuevos flujos van a ser:

1. *InputStreamReader.*
2. *FileOutputStream*.

Con la clase *InputStreamReader*, podremos crear flujos de entrada de datos a ficheros basados en caracteres, es decir, podremos leer información de ellos.

Con la clase *FileOutputStream*, podremos crear flujos de salida de datos a ficheros basados en caracteres, es decir, podremos escribir información en ellos.

Al igual que con los flujos anteriores, las operaciones con este tipo de flujos siempre van a constar de cuatro partes:

1. Abrir el flujo al fichero.
2. Comprobar que el flujo se ha abierto correctamente.
3. Operar con el fichero.
4. Cerrar el fichero.

Vídeo 2. "Jerarquía de clases de flujos 2"
https://bit.ly/2Z5J8gu

/ 8. Ficheros. Formas de acceso a un fichero

En otros lenguajes de programación, existe el concepto de registro. **Un registro es una estructura de datos ya definida que representa un fichero**, y mediante esta representación, podemos saber perfectamente **dónde se encuentran todos sus elementos, cómo pueden ser los metadatos, la parte de información, detalles,** etc.

En Java, esto no ocurre, por lo que, al no haber una estructura de datos ya definida para indicar cómo son los ficheros, son los propios programadores los que tienen que decidir cómo se van a estructurar estos.

Esto tiene tanto ventajas como inconvenientes. Por ejemplo, proporciona un **nivel de personalización muy alto,** pero, por el contrario, **hace que tengamos que explicar cómo están estructurados los ficheros de nuestros programas** para que otros los puedan utilizar.

Con respecto a los ficheros, debemos tener claros los siguientes conceptos antes de adentrarnos en ellos:

- Un **registro lógico** es un conjunto de información de uno de los elementos que hace referencia al fichero.
- Un **bloque o registro físico** es el conjunto de información que puede ser escrito o leído de una sola vez. Por norma general, se podrán escribir o leer varios registros de una vez.
- El número de registros contenidos en un bloque se denomina **factor de bloque**, el cual es muy importante en el diseño de los mismos, ya que influirá de forma directa en la velocidad de lectura/escritura.
- La **dirección lógica** es la dirección relativa dentro del fichero que ocupa un bloque, mientras que la **dirección física** es la dirección real que ocupa dicho bloque en el sistema de almacenamiento que se encuentre.

Fig. 6. Acceso.

En los ficheros, hay dos formas de acceso, principalmente:

1. Los ficheros de acceso secuencial.
2. Los ficheros de acceso aleatorio.

8.1. Acceso secuencial

Los ficheros de acceso secuencial fueron los primeros que aparecieron.

En este tipo de ficheros, los registros que **almacenan la información** se almacenan **de forma secuencial o consecutiva,** es decir, un dato detrás de otro.

La gran consecuencia de este tipo de almacenamiento es que **cualquier operación** que necesitemos hacer sobre uno de estos ficheros se hará sobre el **mismo orden lógico en el que fueron insertados sus registros.**

Con estos ficheros, vamos a poder realizar las siguientes operaciones:

1. **Añadir elementos:** Cuando vayamos a añadir un nuevo elemento en este tipo de ficheros, solo lo podremos hacer al final del mismo, justo después del último registro que tengamos.

2. **Eliminar elementos:** Para poder eliminar un registro en este tipo de ficheros, primero tendremos que localizarlo, y una vez localizado, se podrá suprimir, teniendo que mover los registros de delante una posición atrás para seguir con la estructura lógica.

3. **Modificar elementos:** Para poder modificar un elemento, primero tendremos que localizarlo, y luego podremos modificar su valor.

4. **Consultar elementos:** Para poder consultar un elemento, tendremos que hacerlo en orden secuencial, es decir, habrá que recorrer todos los registros anteriores al registro que queremos localizar.

Este tipo de ficheros son buenos para operaciones de programas que no sean interactivos, ya que tendrán pocos accesos a ellos y los que tengan, serán fáciles de implementar.

Fig. 7. Elementos ordenados.

Tienen la ventaja de que aprovechan bien el espacio en memoria reservado para ellos.

8.2. Acceso aleatorio

Los ficheros de acceso aleatorio surgieron como una mejora de los de acceso secuencial.

Este tipo de ficheros son mucho más versátiles que los de **acceso secuencial y permiten poder acceder a cualquier parte del fichero, en cualquier momento y de forma directa,** como si se tratase de un acceso a memoria.

La **flexibilidad** que nos otorga esta propiedad es muy alta, ya que no tendremos que acceder a todos los registros anteriores para llegar al registro al que queremos acceder, ahorrando una gran cantidad de tiempo.

Con estos ficheros, vamos a poder realizar las siguientes operaciones:

1. **Añadir elementos:** Cuando vayamos a añadir un nuevo elemento en este tipo de ficheros, lo podremos hacer en cualquier punto del mismo. Basta con indicar dónde lo queremos añadir, teniendo que mover los registros de delante una posición hacia adelante para seguir con la estructura lógica.

Fig. 8. Orden aleatorio.

2. **Eliminar elementos:** Al igual que con el acceso secuencial, para poder eliminar un registro en este tipo de ficheros, primero tendremos que localizarlo, y una vez localizado, se podrá suprimir, teniendo que mover los registros de delante una posición atrás para seguir con la estructura lógica.

3. **Modificar elementos:** La modificación también comparte el mismo proceso que en el acceso secuencial, es decir, para poder modificar un elemento, primero tendremos que localizarlo y luego modificarlo.

4. **Consultar elementos:** Para poder consultar un elemento, lo podemos hacer directamente indicando la posición del registro.

Cuando creamos un flujo a este tipo de ficheros, tendremos que indicar qué tipo de operación queremos realizar con ellos, si la escritura o la lectura.

/ 9. Clases para acceso a ficheros

Ya hemos visto que tenemos varios tipos de flujos a ficheros.

El lenguaje de programación Java nos va a permitir trabajar con todos y cada uno de los tipos de flujos que hemos visto, y tiene ya integradas clases para tal propósito.

Todas las clases que necesitaremos estarán dentro del paquete *java.io*, por lo que necesitaremos importarlas en el momento en el que las usemos.

Si recordamos lo visto en unidades anteriores, no es recomendable importar un paquete entero, ya que importaremos clases que no necesitamos, sino que lo que tendremos que hacer es importar una a una las clases vayamos a usar.

Las clases que vamos a necesitar para poder trabajar con flujos son las siguientes:

- ***File:*** Con esta clase, podremos **abrir un fichero.**
- ***FileReader*:** Con esta, generar un flujo de **lectura carácter a carácter** a un fichero de texto.
- ***BufferedReader*:** Con esta, podremos generar un flujo de **lectura línea a línea** a un fichero de texto.
- ***FileWriter*:** Con esta clase, abrir un flujo de **escritura carácter a carácter** a un fichero de texto.
- ***PrintWriter*:** Con esta, podremos abrir un flujo de **escritura línea a línea** a un fichero de texto.
- ***FileOutputStream*:** Con esta, podremos generar un flujo de **escritura carácter a carácter a un fichero binario.**
- ***BufferedOutputStream*:** Con esta clase, podremos generar un flujo de **escritura línea a línea a un fichero binario.**
- ***FileInputStream*:** Con esta, generar un flujo de **lectura carácter a carácter a un fichero binario.**
- ***BufferedInputStream*:** Con esta, podremos generar un flujo de **lectura línea a línea a un fichero binario.**
- ***RandomAccessFile*:** Con esta clase, podremos generar un flujo **tanto de lectura como de escritura a un fichero de acceso aleatorio.**

Fig. 9. Ficheros.

/ 10. Caso práctico 2: "Escribiendo en ficheros"

Planteamiento: Una vez que Pilar y José entienden cómo va el tema de ficheros, están seguros de que no será tan tedioso como pensaban al principio.

El profesor de programación les ha enviado una relación de ejercicios para practicar antes del examen.

Uno de estos ejercicios pide que se realice una función que permita escribir de una forma sencilla un texto dentro de un fichero, pero que deberá controlar cualquier excepción que pueda darse.

Nudo: ¿Cómo crees que puede resolverse este ejercicio? ¿Tiene sentido realizar una función para una operación tan básica como esta?

Desenlace: Cuando estamos programando cualquier función que auguremos nos pueda simplificar las tareas, por sencillas que estas puedan ser, merece la pena hacerla, o por lo menos, intentar hacerla, ya que si lo conseguimos de una manera más o menos fácil, nos ahorrará mucho tiempo de desarrollo a futuro.

En este caso, podemos hacer que esta función ya controle algunos de los posibles errores que puede ocasionar el uso de ficheros, concretamente, la escritura en un fichero. Dicho esto, esta función deberá controlar y lanzar las excepciones pertinentes, aunque si no tenemos claro cuáles son estas, podemos hacer que controle la excepción general, chequeando así los errores producidos de una manera sencilla.

Un ejemplo de la función solicitada podría ser este:

```
public static void escribirFichero(String texto) throws IOException {
    try {
        //Escribimos el texto en el fichero
        fw.write(texto);
    } catch(IOException e) {
        System.out.println("Problemas en la escritura E/S " + e);
    }
}
```

Fig. 10. Función para escribir en fichero.

/ 11. Tipos de rutas

Los ficheros se van a almacenar en disco dentro de la jerarquía de carpetas en la que dividamos nuestro sistema de archivos.

Sea cual sea el sistema operativo que usemos, Windows, GNU/Linux o MacOS, los ficheros van a ser accesibles mediante su ruta.

En informática, llamamos **ruta a la forma que tenemos de referenciar un archivo dentro de un sistema de archivos.** La ruta **va a señalar la posición exacta de nuestro fichero mediante una cadena de caracteres.**

Existen dos tipos de rutas en informática:

- **Rutas relativas:** Indica la posición del fichero desde la posición donde nos encontremos en el sistema de directorios.
- **Rutas absolutas:** Indica la posición del fichero desde el directorio raíz.

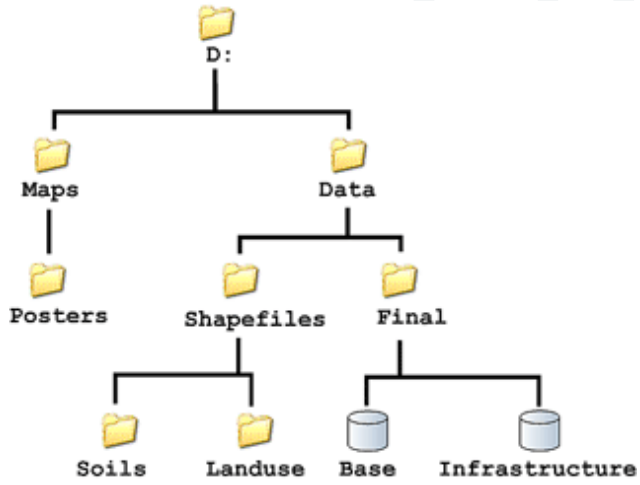

Fig. 11. Jerarquía de ficheros.
Fuente: https://desktop.arcgis.com/es/arcmap/latest/tools/supplement/pathnames-explained-absolute-relative-

Por ejemplo, en la figura anterior, podemos indicar la ruta al fichero "Base" de las dos formas:

- **Ruta relativa:** Si nos encontramos en la carpeta Data, sería Final\Base
- **Ruta absoluta:** D:\Data\Final\Base

Para indicar la ruta de los ficheros, podremos hacerlo tanto con la ruta relativa como con la absoluta.

La forma de indicar la ruta va a cambiar según el sistema operativo, y es más recomendable usar la ruta relativa, aunque los separadores van a cambiar también según el sistema operativo. Para ello, en Java, tenemos la posibilidad de hacer que se ponga el separador automáticamente mediante: ***File.separator.***

/ 12. Excepción en ficheros

Cuando trabajamos con ficheros, vamos a tener que lidiar con multitud de posibles fallos. Entre los más comunes, nos podemos encontrar:

- El fichero al que queremos acceder no existe.
- El fichero al que queremos acceder está corrompido.
- No se tienen permisos de lectura sobre el fichero.
- No se tienen permisos de escritura sobre el fichero.

Todo esto lo vamos a poder solucionar mediante el uso de excepciones, así que el uso de ficheros va a traer consigo el tener que utilizar los bloques *try-catch* y gestionar varias excepciones para asegurar el funcionamiento del programa.

Las **excepciones** que nos vamos a encontrar cuando trabajamos con ficheros son:

- ***FileNotFoundException:*** Esta se lanzará siempre que la ruta del fichero al que queramos acceder no sea correcta o dicho fichero no exista.
- ***IOException:*** Esta se lanzará siempre que el fichero al que queremos acceder no tenga permisos de lectura en el caso de querer leer información de él, o no tenga permisos de escritura, en el caso de querer escribir información en él. O el fichero esté corrupto por cualquier motivo. También puede ocurrir que el usuario que estemos usando no tenga los permisos de lectura o escritura sobre ficheros.

En la siguiente figura, podemos ver un esqueleto básico de bloque *try-catch* que podremos usar para la gestión de ficheros, aunque le falta un pequeño, pero importante, detalle, que veremos en la siguiente unidad.

```
public static void main(String[] args) {
    try
    {
        // Tratamos el fichero
    }
    catch(FileNotFoundException | IOException error)
    {
        System.out.println("Error: " + error.toString());
    }
}
```

Fig. 12. Bloque try-catch.

/ 13. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad, hemos aprendido el concepto de flujo, qué nos permite hacer y los diferentes tipos de flujos que existen.

Otro concepto que hemos estudiado ha sido el de **persistencia de datos,** que va íntimamente ligado al de flujo.

Hemos visto cuáles son los **flujos de datos predeterminados**, aunque los hayamos estado usando sin saberlo desde la primera unidad, y que existen ficheros tanto basados en ***bytes*** como en **caracteres.**

También, hemos expuesto las diferentes **formas de acceso** que tenemos para trabajar con ficheros, analizando sus ventajas e inconvenientes, así como las **clases** que vamos a necesitar en Java para poder tener acceso a los ficheros y poder operar con ellos de forma cómoda.

Finalmente, hemos conocido los dos tipos de **rutas** que existen y las **excepciones** que nos vamos a encontrar cuando trabajemos con ficheros.

Resolución del caso práctico inicial

Uno de los mayores problemas que tenemos hasta el momento es que todos los datos que introducimos en nuestros programas se encuentran únicamente en memoria. De esta forma, mientras el programa está ejecutándose, no hay ningún problema, pero cuando este se cierra, ya sea por un fallo, por una equivocación o porque necesitemos cerrarlo, todos esos datos que había en memoria se pierden y no habrá forma de recuperarlos.

Este es el gran problema que nos solucionan los ficheros, ya que estos nos proporcionan persistencia de datos, o lo que es lo mismo, hacen que podamos almacenar información en disco para, posteriormente, poder recuperarla y seguir trabajando con ella.

También es cierto que los ficheros no son lo más eficiente que podamos tener. Hemos visto sus limitaciones con respecto a la eficiencia en los tiempos de acceso, pero eso lo arreglaremos en futuras unidades.

Fig. 13. Almacenamiento de información.

/ 14. Bibliografía

Flujos de datos. (s. f.). Recuperado 27 de mayo de 2020, de http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/archivos/flujos.htm

C++ con Clase Ficheros en C y C++. (s. f.). Recuperado 27 de mayo de 2020, de http://c.conclase.net/ficheros/?cap=004