I. Manejo de ficheros

Persistencia de datos en ficheros

Actualmente se utilizan en aplicaciones para guardar información simple como un fichero de configuración o un fichero log.

También en sistemas de correo electrónico, para intercambio de datos entre sistemas, (como los estándares XML y JSON que estudiaremos) o para copias de seguridad.

Además, muchísimas aplicaciones almacenan los datos con los que trabajan en ficheros con infinidad de formatos diferentes.

Tipos de ficheros según su contenido

Según la información que contienen se diferencian en:

- Ficheros de texto: contienen únicamente una secuencia de caracteres (visibles, como letras, números o signos de puntuación, y no visibles como espacios y separadores) codificados (UTF-8, ASCII, ISO-8859...). Su contenido se puede visualizar y modificar con cualquier editor de texto.
- Ficheros binarios: son el resto de los ficheros. Pueden contener cualquier tipo de información (texto, imágenes, vídeos, ficheros...). En general, requiere de programas especiales para mostrar la información que contienen. Los programas también se almacenan en ficheros binarios.

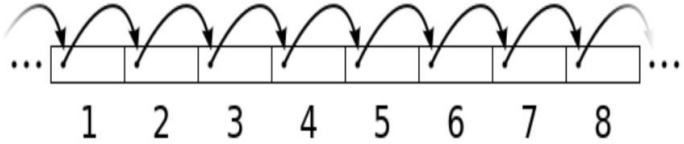
Tipos de ficheros según su contenido

Tipo de fichero	Extensión	Descripción
Ficheros de texto	.txt	Fichero de texto plano
	.xml	Fichero XML
	.json	Fichero de intercambio de información
	.props	Fichero de propiedades
	.conf	Fichero de configuración
	.sql	Script SQL
	.srt	Fichero de subtítulos
Ficheros binarios	.pdf	Fichero PDF
	.jpg	Fichero de imagen
	.doc, .docx	Fichero de Microsoft Word
	.avi	Fichero de vídeo
	.ppt, .pptx	Fichero de Microsoft PowerPoint
	.bin, .dat	Ficheros específicos de aplicación

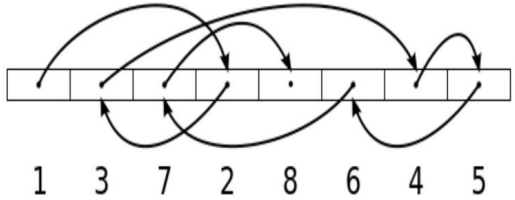
Tipos de ficheros según su forma de acceso

Existen dos formas de acceder a los contenidos de un fichero:

- Acceso secuencial: se accede comenzando desde el principio del fichero. Para llegar a cualquier parte del fichero, hay que pasar antes por todos los contenidos anteriores.



- Acceso aleatorio: se accede directamente a los datos situados en cualquier posición del fichero.



Ficheros en Java

Java proporciona varias API para leer y escribir ficheros:

- Java.io (Java Input Output)
- Java.nio (Java Non-blocking Input Output)

Ambas pueden ser utilizadas, pero NIO supone una mejora sobre IO al corregir muchas de sus deficiencias.

Conviene conocer java.io porque podemos enfrentarnos a proyectos que lo utilicen, pero si realizamos uno desde cero se recomienda usar java.nio.

JAVA.IO

Es la librería inicial de entrada y salida, que se encarga de gestionar las operaciones de entrada y salida de nuestro programa (ficheros, operaciones de entrada y salida por pantalla, etc).

Trabaja con streams, un flujo de datos (en formato de bytes).

La clase principal se llama File, que es una representación abstracta de un fichero o un directorio. Esta clase no se utiliza para escribir o leer datos, sino para trabajar a alto nivel, es decir, se utiliza para crear, buscar, eliminar archivos y para trabajar con directorios y rutas.

JAVA.NIO

Es una librería avanzada para implementar operaciones de entrada y salida de alta velocidad.

Trabaja con buffers, dando soporte de entrada/salida asíncrono.

Incluye nuevas clases para trabajar con ficheros:

- Path, es una actualización sobre la clase File. Se refiere a una ruta dentro de un sistema de ficheros.
- Files, es una clase de utilidad permite realizar operaciones sobre ficheros y directorios de una forma mucho más eficiente que su antecesora.
- FileSystems, para obtener referencias a sistemas de ficheros.

Lectura de ficheros de texto

Los métodos más comunes para leer ficheros de texto son:

- Files.lines, que devuelve un Stream. Desde Java 8 y funciona bien en ficheros grandes y pequeños.
- Files.readString, devuelve un String. Desde Java 11 y para ficheros pequeños (< 2 GB).
- Files.readAllBytes, devuelve un byte[]. Desde Java 7 y para ficheros pequeños (< 2 GB).
- Files.readAllLines, devuelve un List<String>. Desde Java 8.
- BufferedReader, el más recomendable de java.io al almacenar los datos en un buffer → operaciones E/S más eficientes .

Escritura de ficheros de texto

Los métodos más comunes para escribir ficheros de texto son:

- Files.writeString: Desde Java 11, escribe caracteres tal cuál. No quita nada ni añade nada. UTF-8 por defecto
- Files.write: Desde Java 7. Escribe tanto caracteres como datos binarios.
- FileWriter: Escribe directamente a fichero enteros, arrays de bytes o String.
- BufferedWriter: Utiliza buffers para hacer más eficiente la escritura de grandes cantidades de información a un fichero.
- PrintWriter: Escribe texto formateado.

Añadir datos a ficheros de texto

A diferencia de los métodos anteriores, para añadir a final de fichero, se requiere de un segundo argumento explícito que así lo indique.

- Files.writeString y Files.write añaden el argumento StandardOpenOption.APPEND durante la escritura.
- FileWriter, segundo argumento como true.
- BufferedWriter y PrintWriter, envuelven un recurso FileWriter, el cuál debe tener el segundo argumento como true.

Lectura de ficheros binarios

Los métodos más comunes para leer ficheros binarios son:

- Files.readAllBytes, devuelve un byte[]. Desde Java 7 y para ficheros pequeños (< 2 GB).
- FileInputStream, se utiliza para leer byte a byte un fichero, sin buffer.
- DataInputStream, diseñado para leer tipos de datos primitivos.
- BufferedInputStream, que lee bytes en un buffer haciendo más eficientes las operaciones de lectura al minimizar accesos a disco.

Escritura de ficheros binarios

Los métodos más comunes para escribir ficheros binarios son:

- Files.write: Desde Java 7. Escribe tanto caracteres como datos binarios. Podemos añadir bytes al final del fichero con la opción StandardOpenOption.APPEND.
- FileOutputStream, de la librería java.io, se utiliza para escribir secuencias de bytes sin procesar en un fichero, sin buffer.
- DataOutputStream, diseñado para escribir tipos de datos primitivos.
- BufferedOutputStream, para escribir bytes en el fichero mediante buffers, lo que lo hace más eficiente.

Serialización de objetos

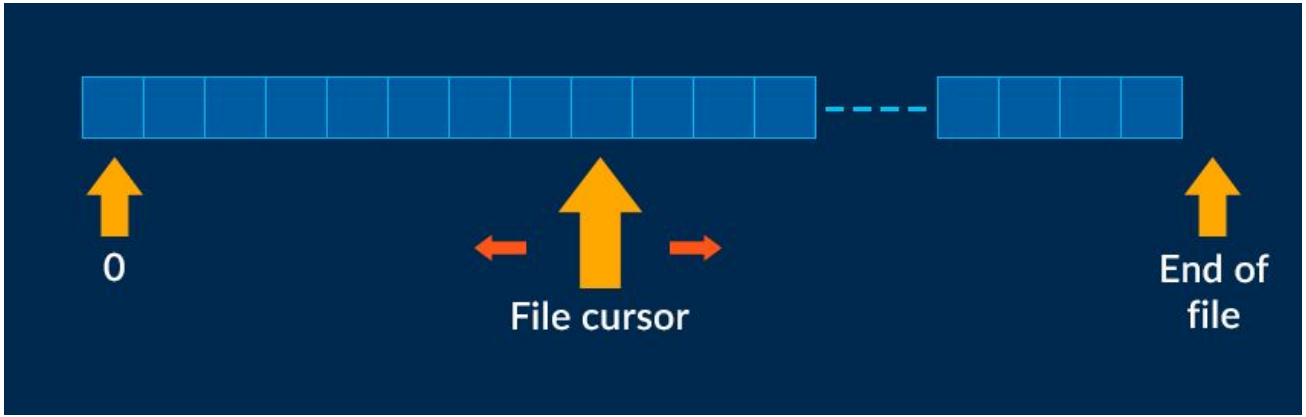
Hemos visto cómo escribir bytes y tipos de datos primitivos en un archivo binario, … pero ¿cómo guardar un objeto en un fichero?

- La clase debe implementar la interfaz Serializable.
- Utilizar las clases ObjectInputStream para leer objetos del fichero con el método readObject, o ObjectOutputStream para escribirlos en él con el método writeObject.
- Se recomienda declarar la constante serialVersionUID de tipo long para asegurar la compatibilidad de los objetos en distintos sistemas.

Al proceso de convertir un objeto en bytes se le llama serializar. A la reconstrucción del objeto a partir de bytes se le llama deserialización.

Acceso aleatorio

Existen situaciones donde necesitaremos acceder a un byte o grupo de bytes concretos para comprobar su valor o modificarlo. En vez de ir byte a byte secuencialmente hasta llegar a la posición deseada, podemos utilizar ficheros de acceso aleatorio.



Acceso aleatorio

La clase RandomAccessFile de Java.io permite leer y/o escribir un fichero binario y moverse hasta una posición dada para leer/modificar la información almacenada. Para esto, los métodos más útiles son:

- read, readInt, readFloat: lee información de un tipo básico dado
- write, writeInt, writeFloat: escribe información de un tipo básico dado
- getFilePointer(): devuelve la posición del puntero
- seek(long pos): establece la posición del puntero
- length(): devuelve el tamaño del fichero en bytes
- skipBytes(int salto): desplaza el puntero *salto* posiciones

Sistema de ficheros en Java

Java proporciona clases para gestionar los sistemas de ficheros y realizar operaciones básicas como:

- Comprobar si un fichero existe o no, si es regular o un directorio
- Copiar, mover y borrar ficheros
- Moverse a través del sistema de ficheros

Clase File

La clase File es la más básica. Pertenece al paquete java.io y ofrece métodos para operaciones como:

- Creación de ficheros y directorios: createNewFile, mkdir, createTempFile
- Eliminación de ficheros y directorios: delete
- Listar directorios y ficheros en un directorio: list, listFiles
- Existencia, permisos y otra información: canRead, canWrite, canExecute... exists, isDirectory, isFile ... getName, length.

Java.io no tiene disponibles las operaciones de copia y movimiento.

Para poder utilizar estas dos operaciones, disponemos de la clase FileUtils, de la librería ApacheCommons

Clase Path

La clase Path es una interfaz que representa una ruta. Pertenece al paquete java.nio, ofreciendo métodos para operaciones como:

- startsWith, endsWith: comprueba si la ruta actual empieza o acaba con una subruta dada.
- getParent: obtiene el directorio padre de la ruta actual.
- getRoot: obtiene el directorio raíz de la ruta actual.
- iterator: para explorar cada directorio y subdirectorio de la ruta.
- toAbsolutePath: para obtener la ruta absoluta de la ruta.

La clase Paths es otra clase de java.nio que contiene métodos estáticos para tratar con rutas.

Clase Files

La clase Files nos permite manejar ficheros reales del disco desde Java. Esta clase tiene métodos que ya hemos visto para leer y escribir en ficheros, pero también otros muy interesantes:

- copy (Path, Path): copia un archivo (solo archivos, no directorios)
- move(Path, Path): mueve un archivo (solo archivos, no directorios)
- delete(Path): elimina un archivo (no directorio)
- createFile(Path): crea un nuevo fichero especificado por Path
- createDirectory(Path): crea un nuevo directorio especificado por Path
- exists(Path): comprueba que una ruta exista