DAM. UNIDAD 1. ACCESO A FICHEROS. PRIMERA PARTE. INTRODUCCIÓN, REPASO DE JAVA Y ACCESO BÁSICO A FICHEROS

DAM. Acceso a Datos (ADA) (a distancia en inglés)

Unidad 1. ACCESO A LOS FICHEROS

Parte 1. Introducción, repaso de Java y acceso básico a ficheros

Abelardo Martínez

Año 2023-2024

1. Introducción

En informática, a partir del objetivo del tratamiento automatizado de la información, surge una necesidad: hacer que los datos persistan más allá de la ejecución del proceso o aplicación que los creó. Los programas utilizan variables para almacenar información: los datos de entrada, los resultados calculados y los valores generados durante el cálculo. Toda esta información es efímera: cuando el programa termina, todo desaparece. Las aplicaciones deben poder almacenar y recuperar datos, por lo que éstos deben ser persistentes.

Estos datos pueden persistir en diferentes sistemas de gestión y tratamiento de datos, como archivos, bases de datos (ya sean relacionales, no relacionales, nativas, etc.) o incluso otros sistemas de procesamiento, y no sólo en dispositivos de almacenamiento.

Veremos que las aplicaciones que trabajan con datos se enfrentan a la dificultad de cómo almacenar y volver a acceder a los datos que se encuentran en diferentes sistemas de información. Este tipo de aplicaciones que trabajan con datos que han de perdurar crean la obligación de utilizar y conocer diferentes técnicas de acceso para cada uno de los sistemas de gestión que se utilicen.

El objetivo inicial de este módulo es abordar conceptualmente el problema de la persistencia de datos en las aplicaciones, analizando diferentes situaciones en las que es necesario que los datos sean almacenados y recuperados considerando diversos enfoques.

2. Gestión del sistema de archivos

21. Sistemas de archivos

Cuando se habla de ordenadores, se suele decir que, en última instancia, lo que hay en un ordenador son unos y ceros. Con el tiempo, se han ido añadiendo diferentes capas de software "encima" de estos ceros y unos para facilitar el uso de los ordenadores.

Un fichero es una abstracción del sistema operativo para el almacenamiento genérico de datos. La parte del sistema operativo que gestiona los ficheros se denomina "sistema de ficheros".

Los sistemas de archivos tradicionales se encargan de organizar, almacenar y nombrar (identificar por su nombre) los archivos almacenados en dispositivos de almacenamiento permanente, como discos duros, memorias USB de estado sólido, DVD, etc.

Hoy en día, los sistemas de archivos más populares son:

Sistema de archivos	Aplicación
ext4	Sistemas operativos Linux
NTFS	Sistemas basados en Windows
ISO9660, UDF	Dispositivos ópticos como CD y DVD

Aunque cada sistema de archivos ofrece su propia visión de los datos almacenados en un disco y los gestiona y organiza a su manera, todos comparten algunos aspectos generales:

- Los archivos suelen organizarse en estructuras jerárquicas de directorios (también llamadas carpetas). Estos directorios son contenedores de archivos (y otros directorios) que ayudan a organizar los datos en el disco.
- El nombre de un fichero está relacionado con su posición en el árbol de directorios que lo contiene, lo que permite no sólo identificar unívocamente cada fichero, sino también encontrarlo en el disco a partir de su nombre.
- Los ficheros suelen llevar asociados una serie de metadatos, como su fecha de creación, la de su última modificación, su propietario o los permisos que los distintos usuarios tienen sobre ellos (lectura, escritura, etc.). Esto convierte al sistema de archivos en una base de datos de los contenidos almacenados en el disco que puede consultarse según múltiples criterios de búsqueda.

22. Rutas: nomenclatura de ficheros

En los sistemas informáticos actuales, en los que un solo ordenador puede tener más de un millón de archivos, es imprescindible disponer de un sistema que permita una gestión eficaz de la localización, de forma que los usuarios puedan moverse cómodamente entre tantos archivos. La mayoría de los sistemas de archivos han incorporado contenedores jerárquicos que actúan como directorios, facilitando la clasificación, identificación y localización de los archivos. Los directorios se han popularizado bajo la versión gráfica de las carpetas.

También hay que tener en cuenta que la excesiva necesidad de espacio de almacenamiento ha llevado a los sistemas operativos a trabajar con un gran número de dispositivos y a permitir el acceso remoto a otros sistemas de archivos distribuidos por la red.

El nombre de un archivo se conoce como su "ruta" (la traducción más cercana en español es "ruta", pero incluso en español utilizamos su nombre en inglés). Hay que tener en cuenta que la ruta de un fichero cambia en función del sistema de ficheros utilizado.

Por ejemplo, veamos el nombre completo de un fichero (su ruta) en un ordenador con sistema Unix:

```
/home/admin/documents/AccessFiles/unit1.txt
```

En este ordenador, no puede haber otro archivo con el mismo nombre (nombre completo de la ruta), aunque puede haber muchos archivos con el nombre corto "unidad1.txt".

221. Linux

Para gestionar tal variedad de sistemas de ficheros, algunos sistemas operativos como Linux o Unix adoptan la estrategia de unificar todos los sistemas en uno solo, para conseguir una forma de acceso unificada y con una única jerarquía que facilite la referencia a cualquiera de sus componentes, independientemente del sistema de ficheros en el que se encuentren realmente. En Linux, sea cual sea el dispositivo o el sistema remoto real donde se almacenará el archivo, la ruta siempre tendrá la misma forma.

En los sistemas operativos Linux, todos los archivos están contenidos en un directorio raíz o en directorios que cuelgan de él. Por ejemplo:

```
/home/admin/documents/AccessFiles/unit1.txt
```

222. Windows

En cambio, la estrategia de otros sistemas operativos como Windows consiste en mantener muy diferenciados cada uno de los sistemas y dispositivos a los que tiene acceso. Para distinguir el sistema al que se va a hacer referencia, Windows utiliza un nombre específico que se incorpora a la ruta del elemento a referenciar. Aunque Microsoft ha apostado claramente por la convención UNC, la evolución de este sistema operativo, que tiene su origen en MS-DOS, le ha llevado a convivir con otra convención también muy extendida. Se trata de la identificación de dispositivos y sistemas con una letra del alfabeto seguida de dos puntos. Esta unidad puede contener a su vez archivos o directorios.

Tradicionalmente, además, en el mundo Windows, los nombres de archivo suelen tener una extensión (normalmente un "." y tres letras) que ayuda a identificar el tipo de contenido almacenado en ese archivo.

A continuación ilustramos ambas convenciones con un ejemplo. Hemos marcado en negrita el nombre específico que identifica al sistema de archivos o dispositivo concreto:

```
Opción a --> F:\Documents\AccessFiles\Unit1.txt
Opción b --> \ServerHS\Documents\AccessFiles\Unit1.txt
```

223. Ejemplos

En ambos sistemas podemos localizar un fichero por su nombre (su ruta). Veamos un par de ejemplos:

Linux:

```
/home/admin/miarchivo
```

Bajo el directorio raíz (/), existe un directorio /home/, dentro del cual hay un directorio /home/admin/ (nombre corto "admin"), dentro del cual hay un archivo /home/admin/miarchivo, nombre corto "miarchivo".

Ventanas:

```
C:\Mis documentos\myfile.txt
```

Bajo la unidad C, hay un directorio C:\Mis documentos, dentro del cual hay un fichero C:\Mis documentos\myfile.txt, de nombre abreviado "myfile.txt", que tiene una extensión típica de los sistemas Windows antiguos de tres letras ("txt").

Como puedes ver en estos ejemplos, las rutas tienen un aspecto muy diferente según el sistema operativo. En Linux, todo parte del directorio raíz y se utiliza el carácter / para separar directorios y archivos entre sí. En Windows, en cambio, se parte de una letra de unidad y se separan los directorios con el carácter \.

23. Rutas relativas y absolutas

Hasta ahora sólo hemos visto ejemplos de rutas absolutas, que son las que permiten identificar de forma unívoca un fichero y su ubicación sin necesidad de más datos. Una ruta relativa especifica la ruta a un fichero desde un directorio de referencia, que se conoce como directorio de trabajo.

Veamos algunos ejemplos:

- Si el directorio de trabajo es /home/admin/, la ruta relativa mi_archivo identifica el archivo /home/admin/mi_archivo.
- Si el directorio de trabajo es /home/admin/, la ruta relativa mi_proyecto/mi_fichero identifica el fichero /home/admin/mi_proyecto/mi_fichero.
- Si el directorio de trabajo es /home/admin/, la ruta relativa ../ada/mi_archivo identifica el archivo /home/ada/mi_archivo. El directorio ../ es un alias del directorio padre del directorio al que sucede.
- Si el directorio de trabajo es /home/admin/, la ruta relativa ./mi_proyecto/mi_fichero identifica el fichero /home/admin/mi_proyecto/mi_fichero. El directorio ./ es un alias del directorio de trabajo.

Para distinguir una ruta relativa de una absoluta, la clave está en darse cuenta de que una ruta relativa no va precedida de la raíz del sistema de archivos (/ en Linux) ni de una letra de unidad en Windows (C:\ por ejemplo).

3. Ficheros: Tipos y acceso

Podemos diferenciar entre varios tipos de archivos según los siguientes criterios:

3.1. En función del contenido

- Archivos de texto (caracteres).
- Archivos binarios (bytes).

3.1.1. Archivos de texto (caracteres)

Aunque ambos tipos acaban almacenando conjuntos de bits en el archivo, en el caso de los archivos de texto sólo encontraremos caracteres que pueden visualizarse con cualquier editor de texto. Los archivos binarios, en cambio, almacenan conjuntos de bytes que pueden representar cualquier cosa: números, imágenes, sonido, etc.

Los ficheros de texto son aquellos que sólo contienen texto en su interior y, por tanto, podemos visualizar y/o manipular su contenido utilizando una herramienta básica del sistema operativo: el editor de texto.

En este punto es necesario distinguir entre un editor de texto y un procesador de textos. Un editor de texto es un programa que permite ver y/o editar archivos de texto, es decir, archivos que sólo contienen texto, mientras que un procesador de textos permite ver y/o editar archivos en los que, además de texto, puede haber otro tipo de información que el procesador de textos tiene que interpretar: fuentes, imágenes, etc.

3.1.2. Archivos binarios (bytes)

En cambio, también tenemos los archivos binarios, que son aquellos que no se pueden ver y/o editar con un editor de texto, y pueden ser de muchos tipos diferentes: Hojas de cálculo (.ods, .xls, .xlsx.) imágenes (.jpg, .png, tiff), sonidos (.mp3, .wav), vídeos (.mp4, .avi, .mkv), procesadores de texto (.odt, .doc, .docx), etc.

32. Según el modo de acceso

Tanto para los archivos de texto como para los archivos binarios, existen dos formas de acceder a los datos que contienen:

- Secuencial.
- Archivos de acceso directo (o aleatorio).

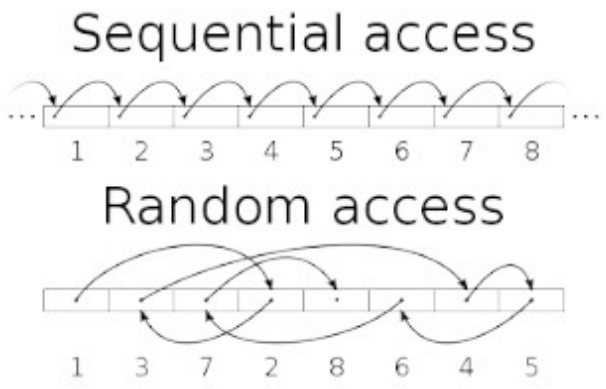
321. Acceso secuencial

En el modo secuencial, la información del fichero es una secuencia de bytes, de modo que para acceder al i-ésimo byte es necesario haber accedido previamente a todos los bytes anteriores. Es decir, es posible leer o escribir una determinada cantidad de datos empezando siempre por el principio del fichero.

También es posible añadir datos empezando por el final del fichero (habrá que recorrer todo el fichero, lo que supone poca eficacia). Por lo tanto, necesitamos herramientas que permitan el acceso directo a cualquier posición dada en un fichero. Para ello es necesario utilizar ficheros de acceso aleatorio.

En Java, el acceso secuencial puede ser:

- Binario → FileInputStream y FileOutputStream
- Caracteres → FileReader y FileWriter

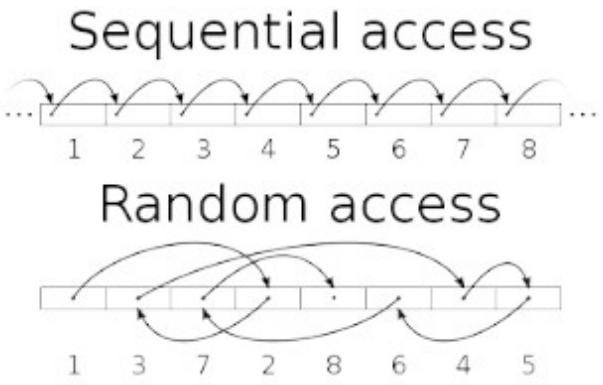


322. Acceso aleatorio

Por el contrario, el modo de acceso directo nos permite acceder directamente a la información del i-ésimo byte sin necesidad de pasar por todos ellos. Es decir, permiten ir a cualquier parte del fichero para actualizar determinados valores.

Por lo tanto, la distinción entre archivos de acceso secuencial y archivos de acceso aleatorio es simplemente una distinción entre las operaciones que se utilizan para acceder a los datos.

En Java, para el acceso aleatorio utilizaremos RandomAccessFile.



3.3. Operaciones de archivo

Las operaciones que podemos realizar sobre cualquier fichero (independientemente de su tipo):

- Creación de ficheros
- Abrir el expediente
- Cierre del expediente
- Lectura de datos del fichero
- Escritura de datos en el archivo

Una vez abierto, las operaciones que se pueden realizar se clasifican de la siguiente manera:

- Adiciones
- Supresiones
- Modificaciones
- Consultas
- Cualquier otra operación, como la búsqueda o la clasificación, se compone de una combinación de las anteriores.

4. Gestión de archivos Java

Resulta evidente que cada sistema operativo puede organizar sus archivos de formas diferentes y utilizar distintos sistemas para identificar su ubicación. La consecuencia de esto es que no existe una forma estándar de referenciar un fichero, sino que depende de cada sistema operativo.

Para superar esta dependencia, Java ha desarrollado la clase File, que permite abstraer la referencia de un fichero independientemente de la notación propia del sistema operativo.

Las instancias de tipo Archivo pueden representar tanto un archivo como un directorio (o carpeta). En cualquiera de los dos casos, nos permiten interrogarlas para obtener información sobre el elemento representado. Así, por ejemplo, podemos saber si se trata de un fichero o de un directorio, u obtener información sobre su nombre en cualquiera de sus formas (absoluto o relativo), su tamaño, la fecha de creación, etc.

File no tiene ninguna utilidad para obtener una lista ordenada. La ordenación se conseguirá utilizando las utilidades Java de la clase System, para ordenar colecciones utilizando un Comparador.

4.1. La clase Archivo. Aspectos generales

En Java, la clase "File" se utiliza básicamente para gestionar el sistema de ficheros. Es una clase que debe **entenderse** como una referencia a la ruta o ubicación de los ficheros en el sistema. NO representa el contenido de ningún fichero, sino la ruta del sistema donde se encuentran. Al ser una **ruta**, la clase puede representar tanto ficheros como carpetas o directorios.

Un objeto de la clase File puede representar:

- Un archivo concreto
- Una serie de archivos ubicados en un directorio
- Un nuevo directorio a crear

Los objetos de la clase File representan rutas del Sistema de Archivos. Si utilizamos una clase para representar las rutas, conseguimos total independencia respecto a la notación que cada sistema operativo utilice para describirlas. Recordemos que Java es un lenguaje multiplataforma y, por tanto, puede ocurrir que tengamos que realizar una aplicación sin conocer el SO donde se ejecutará.

La estrategia utilizada por cada SO no afecta a la funcionalidad de la clase Fichero, ya que ésta, en colaboración con la máquina virtual, adaptará las llamadas al SO anfitrión de forma transparente al programador, es decir, sin que éste tenga que indicar ni configurar nada.

Las instancias de la clase Archivo están estrechamente ligadas a la ruta con la que se crean. Esto significa que las instancias durante todo su ciclo de vida sólo representarán una única ruta, la que se les asoció en el momento de su creación. La clase File no tiene ningún método o mecanismo para modificar la ruta asociada. Si se necesitan nuevas rutas, siempre habrá que crear una nueva instancia y no será posible reutilizar las ya creadas vinculándolas a diferentes rutas.

En implementaciones tan cercanas al SO, los programadores Java tienen que hacer un esfuerzo para que las aplicaciones implementadas sean independientes de las plataformas en las que se ejecutarán. Por ello, hay que procurar utilizar técnicas de parametrización que eviten escribir las rutas directamente en el código, de forma que al trasladar las aplicaciones de una plataforma a otra sólo haya que modificar las rutas del sistema de configuración.

La clase File encapsula prácticamente toda la funcionalidad necesaria para gestionar un sistema de ficheros organizado en un árbol de directorios. Es una gestión completa que incluye:

1. Funciones de manipulación y consulta de la propia estructura jerárquica (creación, supresión, obtención de la ubicación, etc., de ficheros o carpetas).

2. Funciones para manipular y consultar las características particulares de los elementos (nombres, tamaño o capacidad, etc.).
3. Funciones de manipulación y consulta de atributos específicas de cada sistema operativo y que, por tanto, sólo serán funcionales si el sistema operativo anfitrión también soporta la funcionalidad. Nos referimos, por ejemplo, a permisos de escritura, permisos de ejecución, ocultación de atributos, etc.

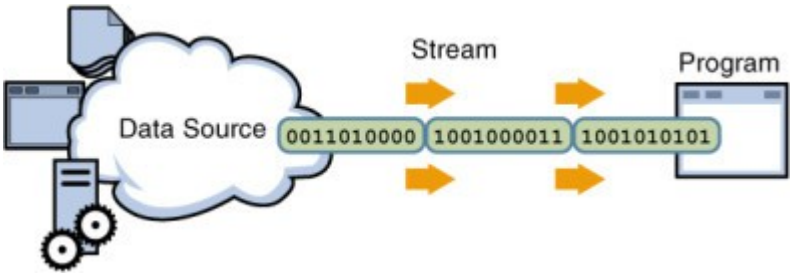
42. Operaciones con ficheros. Flujos de entrada/salida

Los ficheros son almacenes de datos estructurados y podrían considerarse como un intercambio de recursos de datos entre 2 sistemas: uno volátil (memoria RAM) y otro permanente (dispositivos de almacenamiento).

El sistema de E/S de Java tiene diferentes clases definidas en el paquete java.io. Utiliza la abstracción Stream para manejar la comunicación entre un origen y un destino. Esta fuente puede ser:

- El disco duro.
- La memoria principal.
- Una ubicación de red.
- Otro programa.

Cualquier programa que necesite obtener o enviar información a cualquier fuente necesita hacer uso de un stream. Por defecto, la información de un flujo se escribirá y leerá en serie.



Se definen dos tipos de flujos:

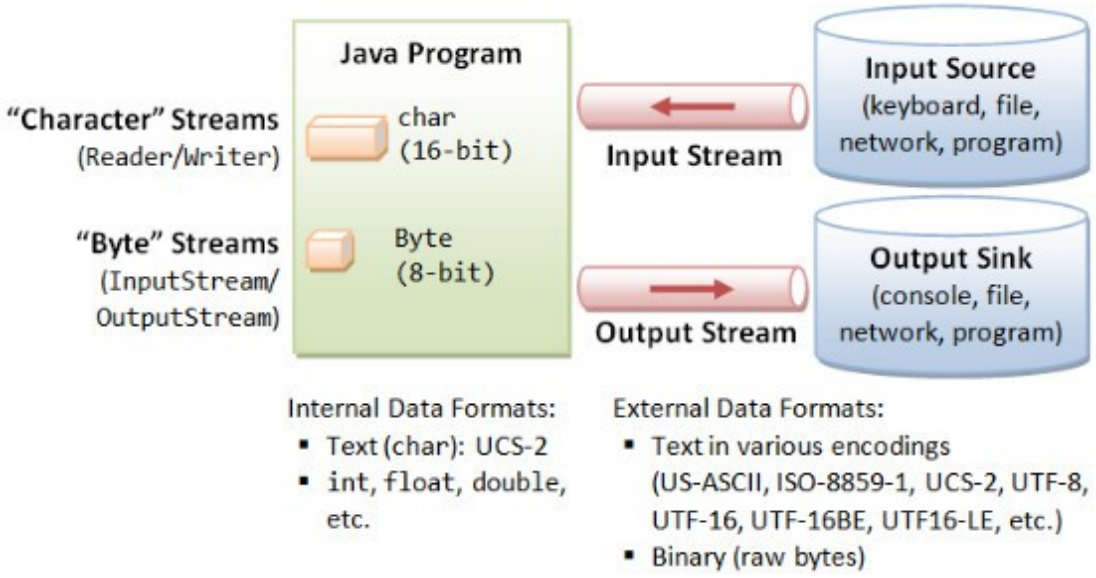
a) Flujos de bytes (8 bits)

- Realiza operaciones de E/S de bytes.
- Orientado a operaciones con datos binarios.
- Todos los tipos de flujos de bytes descienden de InputStream y OutputStream.

b) Secuencias de caracteres (16 bits)

- Realizar operaciones de E/S de caracteres.
- Definido en las clases Reader y Writer.

- Admite caracteres Unicode de 16 bits.

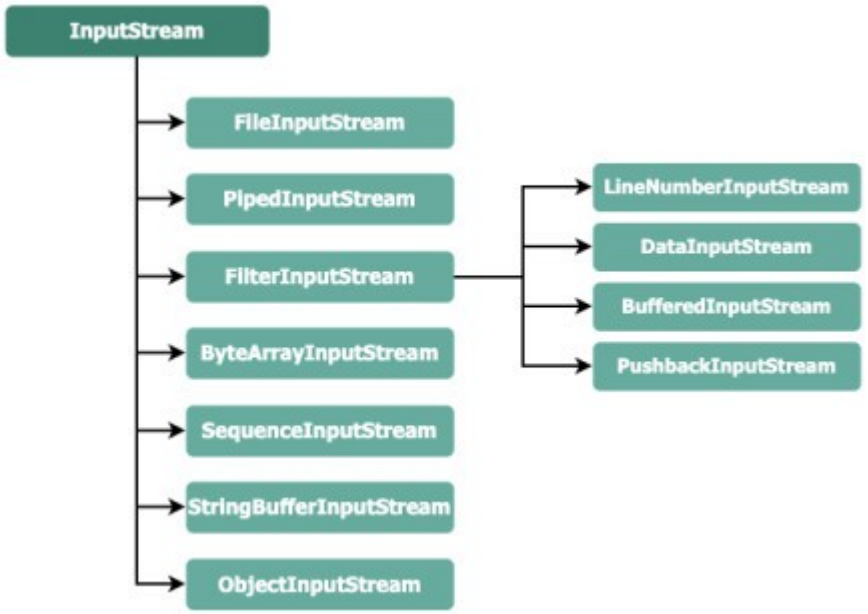


421. Flujos de bytes

a) InputStream

Representa clases que producen entradas desde diferentes fuentes: un Array de Bytes, un objeto String, un fichero, una tubería, un stream, una conexión de red, etc. La siguiente tabla muestra las diferentes clases que heredan de InputStream:

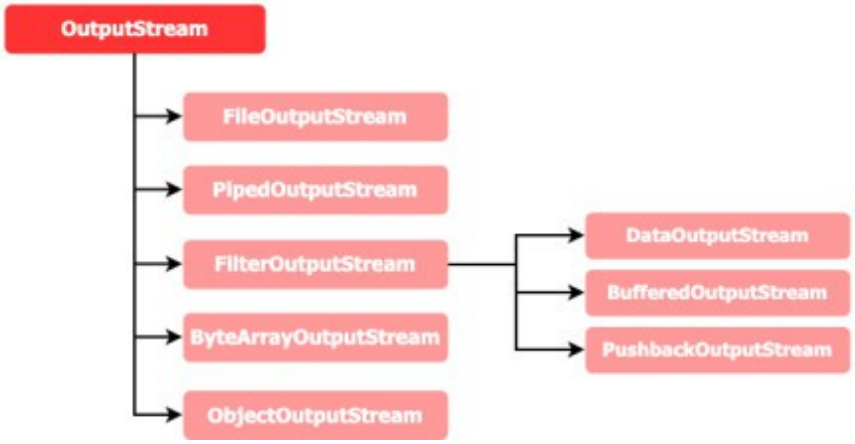
Clase	Descripción
ByteArrayInputStream	Permite utilizar el espacio del bu&#65533;er de memoria.
StringBu&#65533;erInputStream	Convierte una cadena en un InputStream.
FileInputStream	Se utiliza para leer datos de un archivo.
PipedInputStream	Implementa el concepto de canalización.
FiltroInputStream	Proporciona funciones adicionales a otros flujos.
SecuenciaInputStream	Concatena dos o más objetos InputStream.



b) OutputStream

Representa los flujos de salida de un programa. La siguiente tabla muestra las diferentes clases que heredan de OutputStream:

Clase	Descripción
ByteArrayOutputStream	Crea un espacio de almacenamiento en la memoria. Todos los datos enviados a este flujo se almacenan en él.
FileOutputStream	Se utiliza para escribir datos en un archivo.
PipedOutPutStream	Cualquier información escrita en un objeto de esta clase se utilizará como entrada a un PipedInputStream asociado a él. Implementa el concepto de pipeline.
FilterOutputStream	Proporciona funcionalidad adicional a otros OutputStreams.



422. Flujos de caracteres

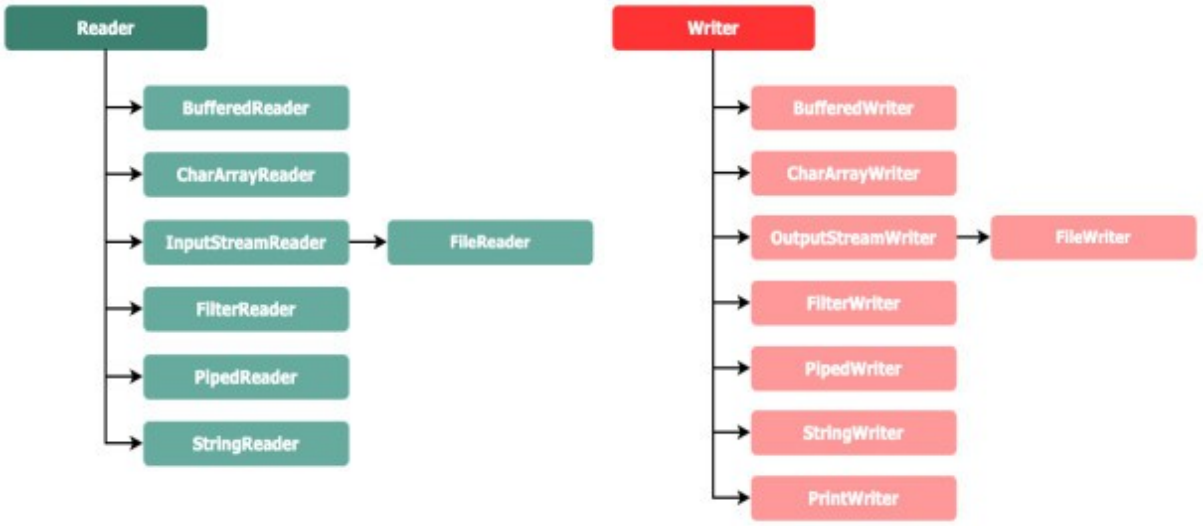
Las clases Reader y Writer manejan flujos de caracteres Unicode. Es habitual utilizar estos flujos en combinación con los flujos que manejan Bytes.

Para ello disponemos de las llamadas "clases puente", que convierten de byteStream a characterStream

- InputStreamReader → Convierte un InputStream en un lector.
- OutputStreamWriter → Convierte un OutputStream en un Writer.

Las clases de flujo de caracteres más importantes:

- FileReader y FileWriter → Lee y escribe caracteres en ficheros.
- CharArrayReader y CharArrayWriter → Leer y escribir matrices de caracteres.
- BufferedReader y BufferedWriter → Se utilizan para utilizar un búfer intermedio entre el programa y el archivo de origen o de destino.

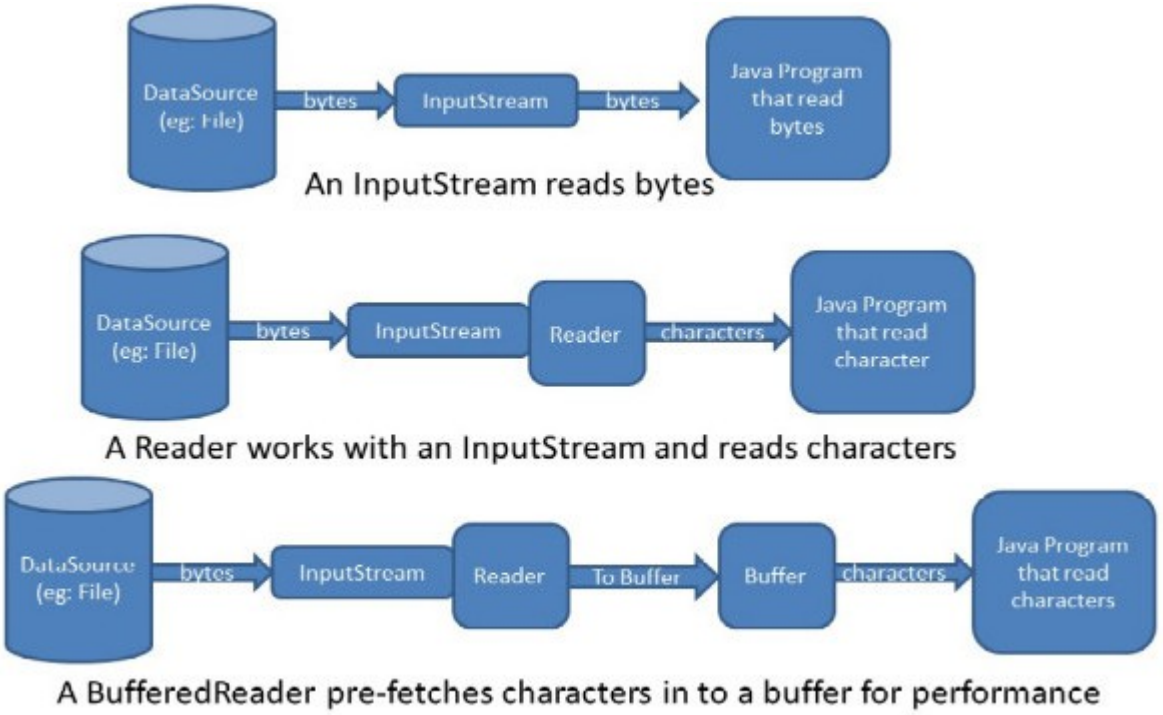


423. Uso de Bu&#xfffd;ers

Un búfer es una estructura de datos que permite el acceso por trozos a una colección de datos. Los búferes son útiles para evitar almacenar grandes cantidades de datos en memoria; en su lugar, se solicitan pequeñas porciones de datos al búfer y se procesan por separado.

También son muy útiles para que la aplicación pueda ignorar los detalles específicos de la eficiencia del hardware subyacente, la aplicación puede escribir en el búfer cuando quiera, y el búfer escribirá en el disco a las velocidades más apropiadas y eficientes.

Ejemplo de uso de búferes para leer datos binarios:



5. Acceso a archivos con Java

La clase Java File es la representación en Java de un nombre de archivo o directorio. Debido a que los nombres de archivos y directorios tienen diferentes formatos en diferentes plataformas, una simple cadena no es adecuada para nombrarlos. La clase Java File contiene varios métodos para trabajar con la ruta de acceso, borrar y renombrar archivos, crear nuevos directorios, listar el contenido de un directorio y determinar varios atributos comunes de archivos y directorios.

- Se trata de una representación abstracta de los nombres de ruta de archivos y directorios.
- Un nombre de ruta, ya sea abstracto o en forma de cadena, puede ser absoluto o relativo. El padre de un nombre de ruta abstracto puede obtenerse invocando el método getParent() de esta clase.
- En primer lugar, debemos crear el objeto de clase Fichero pasándole el nombre del fichero o directorio. Un sistema de archivos puede implementar restricciones a ciertas operaciones en el objeto real del sistema de archivos, como leer, escribir y ejecutar. Estas restricciones se conocen colectivamente como permisos de acceso.
- Las instancias de la clase File son inmutables; es decir, una vez creadas, el nombre de ruta abstracto representado por un objeto File nunca cambiará.

5.1. ¿Cómo crear un objeto de archivo?

Un objeto File se crea pasando una cadena que representa el nombre de un fichero, un String u otro objeto File. Dependiendo de la familia de sistemas operativos para la que estemos programando, deberemos utilizar una nomenclatura u otra para las rutas de los ficheros. Por ejemplo:

- Unix like (Linux, freeBSD, Mac OS, Android, IOS, etc.):

```
Archivo fiUnix = nuevo Archivo("/usr/local/bin/ada.txt");
```

Define un nombre de archivo abstracto para el archivo ada.txt en el directorio /usr/local/bin. Se trata de un nombre de archivo abstracto absoluto.

- Ventanas:

```
Archivo fiWindows = nuevo
Archivo("C:³³Usuarios³Administrador³DocumentosyArchivos³.txt");
```

Esto define un nombre de archivo abstracto para el archivo ada.txt en el directorio C:\Users\Admin
\DocumentosyArchivos\. Se trata de un nombre de archivo abstracto absoluto.

5.2. Campos de la clase File en Java

Resumen de campo:

Modifier y tipo	Campo y descripción
String estático	separador de rutas El carácter separador de rutas dependiente del sistema, representado como una cadena por conveniencia.
carácter estático	pathSeparatorChar El carácter separador de rutas dependiente del sistema.
String estático	separador El carácter separador de nombres por defecto dependiente del sistema, representado como una cadena por conveniencia.
carácter estático	separadorChar El carácter separador de nombres por defecto dependiente del sistema.

5.3. Métodos de la clase File en Java

Los métodos de la clase File pueden clasificarse en varias categorías:

- GET
- SET
- CAN
- IS
- FUNCIONAL

Para más información, consulte la documentación de Oracle en https://docs.oracle.com./javase/8/docs/api/java/io/File.html

5.3.1. Métodos GET

Tipo de devolución	Método	Descripción
Archivo	getAbsoluteFile()	Devuelve la forma absoluta de este nombre de ruta abstracto.
Cadena	getAbsolutePath()	Devuelve la cadena de ruta absoluta de este nombre de ruta abstracto.
Archivo	getArchivoCanónico()	Devuelve la forma canónica de este nombre de ruta abstracto.
Cadena	getCanonicalPath()	Devuelve la cadena de ruta canónica de este nombre de ruta abstracto.
largo	getFreeSpace()	Devuelve el número de bytes no asignados en la partición nombrada por este nombre de ruta abstracta.
Cadena	getName()	Devuelve el nombre del archivo o directorio indicado por esta ruta abstracta.
Cadena	getParent()	Devuelve la cadena de ruta del directorio padre de este nombre de ruta abstracto, o null si este nombre de ruta no nombra un directorio padre.
Archivo	getParentFile()	Devuelve la ruta abstracta del directorio padre de esta ruta abstracta, o null si esta ruta no tiene un directorio padre.
Cadena	getPath()	Convierte este nombre de ruta abstracto en una cadena de nombre de ruta.
largo	getTotalSpace()	Devuelve el tamaño de la partición nombrada por esta ruta abstracta.
largo	getUsableSpace()	Devuelve el número de bytes disponibles para esta máquina virtual en la partición nombrada por esta ruta abstracta.

5.3.2. Métodos SET

Tipo de devolución	Método	Descripción
booleano	setExecutable(boolean ejecutable)	Un método práctico para establecer el permiso de ejecución del propietario para esta ruta abstracta.
booleano	setExecutable(boolean ejecutable, boolean ownerOnly)	Establece el permiso de ejecución del propietario o de todos para este nombre de ruta abstracto.
booleano	setLastModified(long time)	Establece la última hora modificada del archivo o directorio nombrado por esta ruta abstracta.
booleano	setReadable(boolean readable)	Un método práctico para establecer el permiso de lectura del propietario para esta ruta abstracta.
booleano	setReadable(boolean readable, boolean ownerOnly)	Establece el permiso de lectura del propietario o de todos para esta ruta abstracta.
booleano	setReadOnly()	Marca el archivo o directorio nombrado por esta ruta abstracta para que sólo se permitan operaciones de lectura.
booleano	setWritable(boolean writable)	Un método práctico para establecer el permiso de escritura del propietario para esta ruta abstracta.
booleano	setWritable(boolean writable, boolean ownerOnly)	Establece el permiso de escritura del propietario o de todos para esta ruta abstracta.

5.3.3. Métodos CAN

Tipo de devolución	Método	Descripción
booleano	puedeEjecutar()	Comprueba si la aplicación puede ejecutar el archivo indicado por esta ruta abstracta.
booleano	canRead()	Comprueba si la aplicación puede leer el archivo indicado por esta ruta abstracta.
booleano	canWrite()	Comprueba si la aplicación puede modificar el archivo indicado por esta ruta abstracta.

5.3.4. Métodos IS

Tipo de devolución	Método	Descripción
booleano	isAbsoluto()	Comprueba si esta ruta abstracta es absoluta.
booleano	isDirectory()	Comprueba si el archivo indicado por esta ruta abstracta es un directorio.
booleano	isArchivo()	Comprueba si el archivo indicado por esta ruta abstracta es un archivo normal.

5.3.5. Métodos FUNCIONALES

Tipo de devolución	Método	Descripción
int	compareTo(Nombre de archivo)	Compara dos nombres de ruta abstractos lexicográficamente.
booleano	crearNuevoArchivo()	Crea atómicamente un nuevo archivo vacío con el nombre de esta ruta abstracta sólo si aún no existe un archivo con este nombre.
Archivo estático	createTempFile(Cadena prefix, Cadena su&#65533;x)	Crea un archivo vacío en el directorio de archivos temporales predeterminado, utilizando el prefix y el su&#65533;x dados para generar su nombre.
Archivo estático	createTempFile(String prefix, String su&#65533;x, File directory)	Crea un nuevo archivo vacío en el directorio especificado, utilizando las cadenas prefix y su&#65533;x dadas para generar su nombre.
booleano	suprimir()	Elimina el archivo o directorio indicado por esta ruta abstracta.
void	deleteOnExit()	Solicita que el archivo o directorio denotado por esta ruta abstracta sea borrado cuando la máquina virtual termine.
booleano	equals(Objeto obj)	Comprueba la igualdad de este nombre de ruta abstracto con el objeto dado.
booleano	existe()	Comprueba si el archivo o directorio indicado por esta ruta abstracta existe.
int	código hash()	Calcula un código hash para este nombre de ruta abstracto.
largo	lastModified()	Devuelve la hora a la que se modificó por última vez el archivo indicado por este nombre de ruta abstracto.
largo	longitud()	Devuelve la longitud del archivo indicado por esta ruta abstracta.
Cadena[ ]	lista()	Devuelve una matriz de cadenas que nombran los archivos y directorios del directorio indicado por esta ruta abstracta.
Cadena[ ]	list(FilenameFilter filter)	Devuelve una matriz de cadenas que nombran los archivos y directorios del directorio indicado por este nombre de ruta abstracto que satisfacen el filtro especificado.
Archivo[ ]	listFiles()	Devuelve una matriz de nombres de ruta abstractos que indican los archivos del directorio indicado por este nombre de ruta abstracto.
Archivo[ ]	listFiles(FileFilter filter)	Devuelve una matriz de nombres de ruta abstractos que indican los archivos y directorios del directorio indicado por este nombre de ruta abstracto que satisfacen el filtro especificado.
Archivo[ ]	listFiles(FiltroDeNombresDeArchivo filtro)	Devuelve una matriz de nombres de ruta abstractos que indican los archivos y directorios del directorio indicado por este nombre de ruta abstracto que satisfacen el filtro especificado.
static Archivo[ ]	listRoots()	Lista las raíces de filesystem disponibles.

booleano	mkdir()	Crea el directorio nombrado por esta ruta abstracta.

booleano	mkdirs()	Crea el directorio nombrado por esta ruta abstracta, incluyendo cualquier directorio padre necesario pero inexistente.
booleano	renameTo(Archivo dest)	Renombra el fichero indicado por esta ruta abstracta.
Ruta	toPath()	Devuelve un objeto java.nio.file.Path construido a partir de esta ruta abstracta.
Cadena	toString()	Devuelve la cadena de ruta de este nombre de ruta abstracto.
URI	toURI()	Construye un file: URI que representa este nombre de ruta abstracto.

5.4. Constructores de la clase Java File

Necesitamos crear una nueva instancia para trabajar con el archivo/carpeta. Esto se puede hacer de forma muy sencilla y su constructor tiene 4 sobrecargas:

Constructor	Descripción
Archivo(Archivo padre, Cadena hijo)	Crea una nueva instancia de Archivo a partir de una ruta abstracta padre y una cadena de ruta hija.
File(String pathname)	Crea una nueva instancia de Archivo convirtiendo la cadena de ruta dada en una ruta abstracta.
Archivo(String padre, String hijo)	Crea una nueva instancia de Archivo a partir de una cadena de ruta padre y una cadena de ruta hija.
Archivo(URI uri)	Crea una nueva instancia de File convirtiendo el file: URI en un nombre de ruta abstracto.

5.5. Notación húngara

A los datos utilizados por un programa se les asigna un nombre identificador. En Java -como en otros lenguajes de programación- podemos dar cualquier nombre a una variable o constante (excepto palabras reservadas). Sin embargo, este sistema flexible puede dar lugar a código complejo de leer y entender.

Para resolver este problema, Charles Simonyi inventó la notación húngara. Esta notación es un sistema de uso común para crear nombres de variables y consiste en prefijos en minúsculas que se añaden a los nombres de las variables para indicar su tipo. El resto del nombre indica, lo más claramente posible, la función que realiza la variable.

En nuestro caso, nos basaremos en la notación húngara para los nombres de las variables, aunque con algunas diferencias. Se recomienda seguir la notación de la tabla siguiente:

Tipo de datos	Notación húngara
Matriz	ar
ArrayList	arl
booleano	b
Bu&#xFFFD;eredReader	br
Bu&#xFFFD;eredWriter	bw
char	ch
Fecha	dt
doble	d
Archivo	fi
float	f
int	i
Lista	lst
largo	l
Objeto	obj
Escáner	sc
Cadena	st

5.6. Ejemplos de clases de archivos Java

Ejemplo 1:

Programa para comprobar si un archivo o directorio existe físicamente o no.

```
import java.io.File;
import java.util.Scanner;

/*
 * En este programa Java, aceptamos un nombre de archivo o directorio desde el teclado.
 * A continuación, el programa comprobará si ese archivo o directorio existe físicamente o
 * y muestra la propiedad de ese archivo o directorio.
 */
public clase Ejemplo1 {

    /*
     * ------------------------------
     * VARIABLES Y CONSTANTES GLOBALES
     * ------------------------------
     */
    /Constante PATH. Suponemos que nuestro archivo se encuentra en la carpeta "Docume final static String PATH =
    "/home/"+System.getProperty("user.name")+"/Doc
    //Sólo podemos usar UN escáner.
    //Podemos pasar la instancia a cada método o crear esta estática global public static Scanner scKeyboard = new Scanner(System.in);

    /*
     * ---------
     * MÉTODOS
     * ---------
     */
    //leer el fichero o directorio por teclado
    public static String ReadUserFile ()
```

```
    {
        System.out.print("Introduzca el nombre del
        archivo: "); String stFilename =
        scKeyboard.next(); return PATH+stFilename;
    }

    /*
     * --------------
     * PROGRAMA PRINCIPAL
     * --------------
     */
     public static void main(String[] stArgs) {
        //leer nombre de archivo
        String stFilename = LeerArchivoUsuario();
        // pasar el nombre de fichero o directorio al objeto
        File File fiFile = new File(stFilename);
        // aplicar los métodos de la clase File al objeto File
        System.out.println("Nombre del fichero: " + fiFile.getName());
        System.out.println("Ruta: " + fiFile.getPath());
        System.out.println("Ruta absoluta: " + fiFile.getAbsolutePath());
        System.out.println("Padre: " + fiFile.getParent());
        System.out.println("Existe: " + fiFile.exists());
        // Visualización de la
        propiedad del fichero if
        (fiFile.exists()) {
            System.out.println("Se puede escribir: " + fiFile.canWrite());
            System.out.println("Se puede leer: " + fiFile.canRead());
            System.out.println("Es un directorio: " + fiFile.isDirectory());
            System.out.println("Tamaño del archivo en bytes: " +
            fiFile.length());
        }
        System.out.println();
        scKeyboard.close(); //cerramos el escáner al final
     }
}
```

Salida:

```
Input the file name: filetest.txt
File name: filetest.txt
Path: /home/lliurex-admin/Documentos/filetest.txt
Absolute path: /home/lliurex-admin/Documentos/filetest.txt
Parent: /home/lliurex-admin/Documentos
Exists: true
Is writable: true
Is readable: true
Is a directory: false
File Size in bytes: 13
```

Ejemplo 2:

Programa para mostrar todo el contenido de un directorio. Aquí aceptaremos un nombre de directorio desde el teclado y luego mostraremos todo el contenido del directorio. Para este propósito, el método list() puede ser usado como:

```
String arArchivo[ ]=fiArchivo.list();
```

En la sentencia anterior, el método list() hace que todas las entradas de directorio se copien en el array arFile[ ]. A continuación, pasa estos elementos del array arFile[ii] al objeto File y los comprueba para saber si representan un fichero o un directorio.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException; import
java.io.InputStreamReader;

/*
 * Programa Java para mostrar todo el contenido de un directorio
 */
public clase Ejemplo2 {

    // Visualizar el contenido de un directorio
    public static void main(String[] stArgs) throws IOException {
        // introduzca la ruta y el nombre desde el teclado
```

```
BufferedReader brLine = new BufferedReader(new InputStreamReader(Syst

System.out.println("Introducir ruta de
directorio: "); String stPath = brLine.readLine();
System.out.println("Introducir nombre de
directorio: "); String stName = brLine.readLine();

// crear objeto File con ruta y nombre File
fiFile = new File(stPath, stName);

// si el directorio
existe,entonces if
(fiFile.exists()) {
    // obtener el contenido en arFolders[]
    // ahora arFolders[i] representa un Archivo o Directorio String
    arstFolders[] = fiFile.list();

    // encontrar el número de entradas en
    el directorio int iNum =
    arstFolders.length;

    // visualización de las entradas
    for (int ii = 0; ii < iNum; ii++) {
        System.out.println(arstFolders[ii]);
        // crear objeto File con la entrada y
        // comprobar si se trata de un archivo o directorio
        File fiCurrent = new File(arstFolders[ii]);
        if (fiCurrent.isFile())
            System.out.println(": es un
        archivo"); if
        (fiCurrent.isDirectory())
            System.out.println(": es un directorio");
    }
    System.out.println("Número de entradas en este directorio " +
    iNum)
}
si no
    System.out.println("Directorio no encontrado");
}
```

```
}
```

6. Acceso a archivos de texto en Java

Para trabajar con ficheros de texto utilizaremos:

- Lector de archivos para leer.
- FileWriter para escribir.

Siempre que trabajemos con estas clases debemos realizar una correcta gestión de excepciones ya que se pueden producir

- FileNotFoundException → En caso de no encontrar el archivo.
- IOException → Cuando se produce algún tipo de error de escritura.

6.1. Lector de archivos

La siguiente tabla muestra los métodos utilizados por la clase FileReader para leer ficheros, en caso de estar al final del fichero (EOF) todos ellos devuelven -1.

Método	Descripción
int leer()	Lee un carácter y lo devuelve.
int leer(char[ ] buf)	Lee hasta buf.length caracteres de datos. Los caracteres leídos se almacenan en buf.
int read(char buf, int o□set, int n)	Lee hasta n caracteres de datos desde la posición o□set.

En Java, para abrir y leer un archivo de texto, debemos:

- Crea una instancia de la clase File.
- Crear un flujo de entrada con la clase FileReader.
- Realice las operaciones de lectura pertinentes.
- Cierra el archivo con el método .close().

Ejemplo (lectura de un fichero carácter por carácter):

```
//Creamos el fichero en nuestra carpeta personal
Archivo fiFile= new File("/home/"+System.getProperty("nombre.usuario")
+ "/Sa

intentar {
    //Creamos el flujo de entrada
    FileReader frFile= new FileReader(fiFile); int
    iCaracter;

    //Leemos el fichero carácter a carácter
    while((iCaracter=frFile.read())!=-1) {
        System.out.print((char)iCaracter);
    }
    //Cierra el fichero
    frFile.close();
```

```
        } catch (FileNotFoundException fnfe) {
            fnfe.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
```

Ejemplo (lectura de un fichero de 20 por 20):

```
//Leemos el fichero de 20 en 20
char[] chBuffer= new char[20];
while((iCaracter=frFile.read(chBuffer))!=-1) {
    System.out.print(chBuffer);
}
```

62. Escritor de archivos

La siguiente tabla muestra los métodos utilizados por la clase FileWriter para escribir en archivos:

Método	Descripción
void write(int c)	Escribe un personaje.
void write(char[ ] buf)	Escribe una matriz de caracteres.
void write(char buf, int o□set, int n)	Escribe n caracteres de datos a partir de la posición o□set.
void write(String str)	Escribe una cadena de caracteres.
void append(char c)	Añade un carácter al final del archivo.

En Java, al escribir en un archivo de texto debemos:

- Crea una instancia de la clase File.
- Crear un flujo de salida con FileWriter.
- Realizamos las operaciones de escritura.
- Cierra el archivo con el método .close().

Ejemplo 1:

```
public static void main(String[] stArgs) {
    //Creamos el fichero en nuestra carpeta personal
    File fiFile= new File("/home/"+System.getProperty("nombre.usuario") +
    "/Fi try {
        //Crear un flujo de salida
        FileWriter fwFile= new FileWriter(fiFile);
        String stTexttowrite="Esta es mi primera escritura en disco";
        //Escribimos en el fichero
        fwFile.write(stTexttowrite);
        //Cierra el fichero
        fwFile.close();
    } catch (IOException ioe) {
        ioe.printStackTrace();
```

```
        }
    }
```

Ejemplo 2:

En el siguiente ejemplo vamos a escribir todo el contenido de un array en un fichero.

```
//Creamos el fichero en nuestra carpeta personal
Archivo fiFile= new File("/home/"+System.getProperty("nombre.usuario") +
"/Archivo02.txt try {

FileWriter fwFile= new FileWriter(fiFile); String
stRomanprovinces[]={
    "Baetica", "Balearica", "Carthaginensis",
    "Gallaecia", "Lusitania", "Tarraconensis"
};
for (int ii = 0; ii < stRomanprovinces.length; ii++) {
    fwFile.write(stRomanprovinces[ii]);
}

//Cierra el fichero
fwFile.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

Debemos tener en cuenta que si el fichero ya contenía información anteriormente, sobrescribiremos su contenido cuando escribamos en él. Si nuestra intención es añadir nueva información, debemos crear el flujo de salida de la siguiente manera:

```
Filewriter fw = new FileWriter(f, true)
```

El segundo parámetro del constructor es el parámetro append:

- true → Añade la información al final del archivo.
- falso → Borra el contenido anterior para insertar el nuevo.

6.3. Bu?eredReader

Con la clase BufferedReader podemos hacer una gestión más avanzada de la lectura de ficheros. La clase java.io.BufferedReader es ideal para leer archivos de texto y procesarlos. Permite leer eficientemente caracteres sueltos, matrices o líneas completas como cadenas.

Cada lectura de un BufferedReader desencadena una lectura del fichero al que está asociado. Es el propio BufferedReader el que recuerda la última posición del fichero leído, de forma que las lecturas posteriores acceden a posiciones consecutivas del `fichero`. El método readLine() lee una línea `del fichero` y la devuelve en forma de String. Es especialmente útil si el separador de registros es el carácter \n. Este método devuelve un String con la cadena leída o null si estamos al final del fichero (EOF).

Para crear un BufferedReader necesitamos partir de la clase FileReader. Esto se debe a que la clase que realmente gestiona el flujo es FileReader. BufferedReader lo utiliza para hacer una lectura en buffer.

Ejemplo:

```
public static void main(String[] stArgs) {
    //Creamos el fichero en nuestra carpeta personal
    File fiFile= new File("/home/"+System.getProperty("nombre.usuario") +
    "/File06 try {
        BufferedReader brFile= new BufferedReader(new FileReader(fiFile));
        String stLine="";
        while((stLine=brFile.readLine())!=null) {
            System.out.println(stLine);<br>  }
        brFile.close();

    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

6.4. Bu?eredWriter

También tenemos la clase BufferedWriter, que nos permite escribir línea a línea. Al igual que con la lectura, la escritura desde buffers es mucho más eficiente que el uso de matrices de bytes para archivos grandes.

- El método .write() escribe una línea en nuestro fichero.
- El método .newLine() escribe un salto de línea.

Ejemplo:

```
//Creamos el fichero en nuestra carpeta personal
Archivo fiFile= new File("/home/"+System.getProperty("nombre.usuario") +
"/Archivo07.txt try {
    BufferedWriter bwFile= new BufferedWriter(new FileWriter(fiFile)); String
    stRomanprovinces[]={
            "Baetica", "Balearica", "Carthaginensis",
            "Gallaecia", "Lusitania", "Tarraconensis"
        };
    for (int ii = 0; ii < stRomanprovinces.length; ii++) {
        bwFile.write(stRomanprovinces[ii]);
        bwFile.newLine();
    }
    bwFile.close();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

6.5. Impresora

La clase PrintWriter tiene los métodos print(String) y println(String) cuyo comportamiento es similar a los métodos System.out.

- Ambos reciben un String y lo escriben en el fichero.
- println también inserta un salto de línea.

Para construir un PrintWriter, una vez más, necesitamos un FileWriter.

Ejemplo:

```
//Creamos el fichero en nuestra carpeta personal
Archivo fiFile= new File("/home/"+System.getProperty("nombre.usuario") +
"/Archivo08.txt try {
    PrintWriter pwFile= new PrintWriter(new FileWriter(fiFile)); String
    stRomanprovinces[]={
            "Baetica", "Balearica", "Carthaginensis",
            "Gallaecia", "Lusitania", "Tarraconensis"
        };
    for (int ii = 0; ii < stRomanprovinces.length; ii++) {
        pwFile.println(stRomanprovinces[ii]);
    }
    pwFile.close();

} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

7. Bibliografía

Fuentes

- Wikipedia. Acceso secuencial a ficheros. https://en.wikipedia.org/wiki/Sequential_access
- Wikipedia. Acceso aleatorio a archivos. https://en.wikipedia.org/wiki/Random_access
- Tipos de ficheros. https://www.palabrasbinarias.com/articles/2022-03-31-tipos-de-ficheros/
- Josep Cañellas Bornas, Isidre Guixà Miranda. Acceso a datos. Desarrollo de aplicaciones multiplataforma. Creative Commons. Departamento de Enseñanza, Instituto Obert de Cataluña. Dipòsit legal: B. 29430-2013. https://ioc.xtec.cat/educacio/recursos
- Alberto Cortés. Entrada/Salida con ficheros en Java. Universidad Carlos III de Madrid.
- Alberto Oliva Molina. Acceso a datos. UD 1. Trabajo con ficheros XML. IES Tubalcaín. Tarazona (Zaragoza, España).
- Geeksforgeeks. https://www.geeksforgeeks.org/file-class-in-java/
- Documentación de Oracle. Archivo de clase. https://docs.oracle.com/javase/8/docs/api/java/io/Archivo.html