Acceso a Datos

UT1. MANEJO DE FICHEROS DE TEXTO Y BINARIOS

FICHEROS DE TEXTO

1. Introducción

- Los programas usan variables para almacenar **información**: los datos de entrada, los resultados calculados y valores generados a lo largo del cálculo. Toda esa información es **efímera**: cuando se acaba el programa, todo desaparece. Pero, para muchas aplicaciones, es importante poder almacenar datos de manera **permanente**.
- Cuando se desea guardar información más allá del tiempo de ejecución de un programa, lo habitual es organizar esa información en uno o varios **ficheros** almacenados en algún soporte de almacenamiento persistente. Otras posibilidades como el uso de **bases de datos** utilizan archivos como soporte para el almacenamiento de la información.

1. Introducción

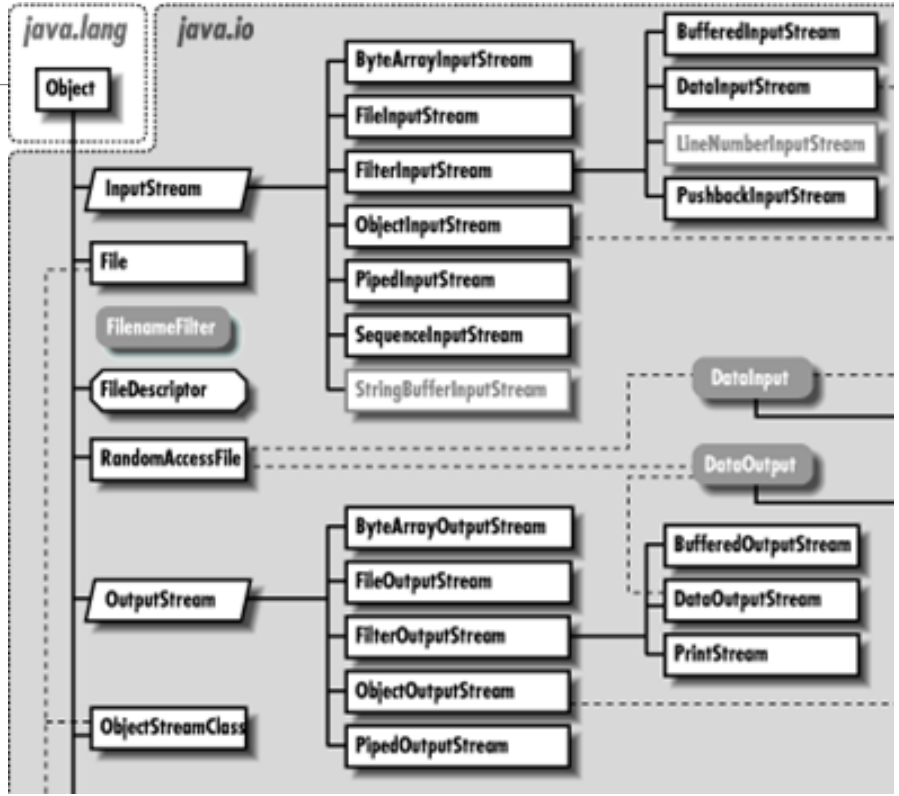
Tipos de ficheros.

- Podemos diferenciar varios tipos de archivos en función de los siguientes criterios:
  - **En función del contenido**: archivos de **texto** (de caracteres) y archivos **binarios** (de bytes).
    - Aunque en el fondo ambos tipos acaben almacenando conjuntos de bits en el archivo, en el caso de los ficheros de texto solo encontraremos caracteres que podrán visualizarse usando cualquier editor de texto. Por el contrario, los ficheros binarios almacenan conjuntos de bytes que pueden representar cualquier cosa: número, imágenes, sonido, etc.
  - **En función del modo de acceso**: archivos **secuenciales** y de a**cceso directo** (o de acceso aleatorio).
    - En el modo secuencial la información del archivo es una secuencia de bytes de forma que para poder acceder al byte i-ésimo se ha de haber accedido previamente a todos los bytes anteriores. Por el contrario, el modo de acceso directo nos permite acceder directamente a la información del byte i-ésimo sin necesidad de recorrerlos todos.

1. Introducción

- En Java, disponemos de varias clases dentro de los paquetes **java.io** y **java.nio.file** para manejar los distintos tipos de archivos que existen.

- Veamos las clases más importantes, en primer lugar, del paquete **java.io**.



2. La clase File

- La clase **File** proporciona un conjunto de utilidades relacionadas con los ficheros que nos van a proporcionar información acerca de los mismos:
  - Su nombre
  - Sus atributos
  - Sus directorios
  - Etc.

- Un objeto de la clase **File** puede representar:
  - Un archivo en particular.
  - Una serie de archivos ubicados en un directorio.
  - Un nuevo directo para ser creado.

2. La clase File

- Para crear un objeto de la clase File podemos usar cualquiera de los siguientes constructores:

a) `File(String rutaCompleta)`

```
File fichero= new File("./resources/textFiles/file001.txt");
```

a) `File(String rutaDirectorio, String nombreArchivo)`

```
File fichero2= new File("./resources/textFiles","file001.txt" );
```

2. La clase File

- Dependiendo de la familia SO operativos para los que estemos programando debemos utilizar una nomenclatura u otra para las rutas de ficheros:
  - Windows

```
File ficheroWindows= new File("C:\\Users\\Admin\\DocumentsandFiles\\file001.txt");
```

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

```
File ficheroNix= new File("/Users/Admin/DocumentsandFiles/file001.txt");
```

2. La clase File

Métodos de la clase

Método	Función
`String[] list()`	Devuelve un array de String con los nombres de los ficheros y directorios asociados al objeto File
`File[] listFiles()`	Devuelve un array de objetos File conteniendo los ficheros que estén dentro del directorio representado por el objeto FIle
`String getName()`	Devuelve el nombre del fichero o directorio
`String getPath()`	Devuelve la ruta del fichero o directorio
`String getAbsolutePath()`	Devuelve la ruta absoluta al fichero o directorio
`boolean exists()`	Devuelve true si el fichero o directorio existe
`boolean canWrite()`	Devuelve true si el fichero se puede escribir

2. La clase File

Métodos de la clase

Método	Función
`boolean canRead()`	Devuelve true si el fichero o directorio se puede leer.
`boolean isFile()`	Devuelve true si el objeto File corresponde a un fichero
`boolean isDirectory()`	Devuelve true si el objeto File corresponde con un directorio.
`long length()`	Devuelve el tamaño del fichero en bytes
`boolean mkdir()`	Crea un directorio con el nombre del objeto File (En caso de que no exista)
`boolean renameFileTo(File newName)`	Renombre el fichero representado por el objeto File, asignándole el nombre de este.
`boolean delete()`	Borra el fichero o directorio
`boolean createNewFile()`	Crea un fichero vacío asociado al objeto File (Solo se creará en caso de que no exista)

2. La clase File

- Ejemplo de uso de la clase File

Hemos creado un fichero llamado file001.txt en la carpeta src de nuestro proyecto.

1. Creamos una cadena para la ruta
2. Creamos una cadena para el nombre del fichero
3. Creamos el fichero
4. Mostramos algunas de sus propiedades
5. Comprobamos que el fichero existe
   a. En caso de existir mostramos sus permisos y su tamaño
   b. En caso de no existir se indica por pantalla.

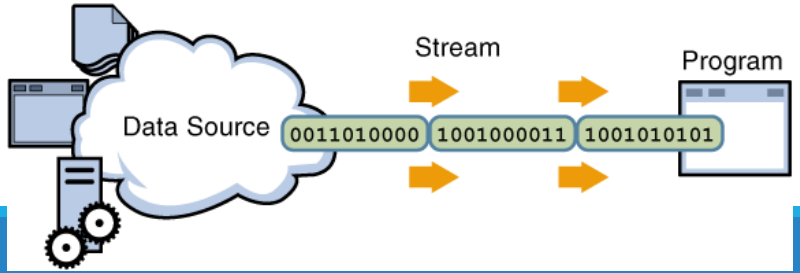
```
public class Main {

    public static void main(String[] args) {

        String ruta = "src/";
        String nombre = "file001.txt";
        File f = new File(ruta+nombre);

        System.out.println("Nombre: "+f.getName());
        System.out.println("Ruta: "+f.getAbsolutePath());
        System.out.println("Directorio padre: "+f.getParent());
        if (f.exists()) {
            System.out.println("¡El fichero existe!");
            System.out.println("Permisos(rwx)=>"+f.canRead()+f.canWrite()+f.canExecute());
            System.out.println("Longitud del fichero: "+f.length()+" bytes");
        }
        else {
            System.out.println("El fichero no existe");
        }



    }

}
```

2. La clase File

- Estos son tan solo algunos de los métodos de la clase File. Para más información consultar la documentación de Oracle en
- https://docs.oracle.com/javase/7/docs/api/java/io/File.html
- https://docs.oracle.com/javase/8/docs/api/java/io/File.html

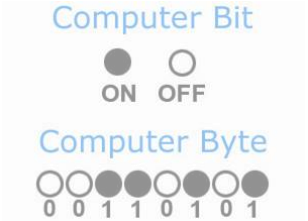
3. Flujos o Streams

- El sistema de I/O de Java dispone de diferentes clases definidas en el paquete java.io
- Emplea la abstracción del flujo (Stream) para tratar la comunicación entre una fuente y un destino.
  - Dicha fuente puede ser:
    - El disco duro.
    - La memoria principal.
    - Una ubicación en 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 stream se escribirá y leerá en serie.



3. Flujos o Streams

- Se definen dos tipos de flujos:
  - **Flujos de Bytes (8 bits)** .
    - Realiza operaciones de I/O de bytes.
    - Orientado a operaciones con datos binarios
    - Todas las clases de flujos de bytes descienden de InputStream y OutputStream.
  - **Flujos de Caracteres (16 bits)**
    - Realizan operaciones de I/O de caracteres.
    - Definido en las clases Reader y Writer
    - Soporta caracteres Unicode de de 16 bits.



3. Flujos o Streams

Flujos de Bytes (Byte Streams)

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

Clase	Función
ByteArrayInputStream	Permite usar el espacio de almacenamiento intermedio de memoria.
StringBufferInputStream	Convierte un String en un InputStream
FileInputStream	Se emplea para leer datos de un fichero.
PipedInputStream	Implementa el concepto de tubería
FilterInputStream	Proporciona funcionalidad adicional a otras streams
SequenceInputStream	Concatena dos o más objetos InputStream

3. Flujos o Streams

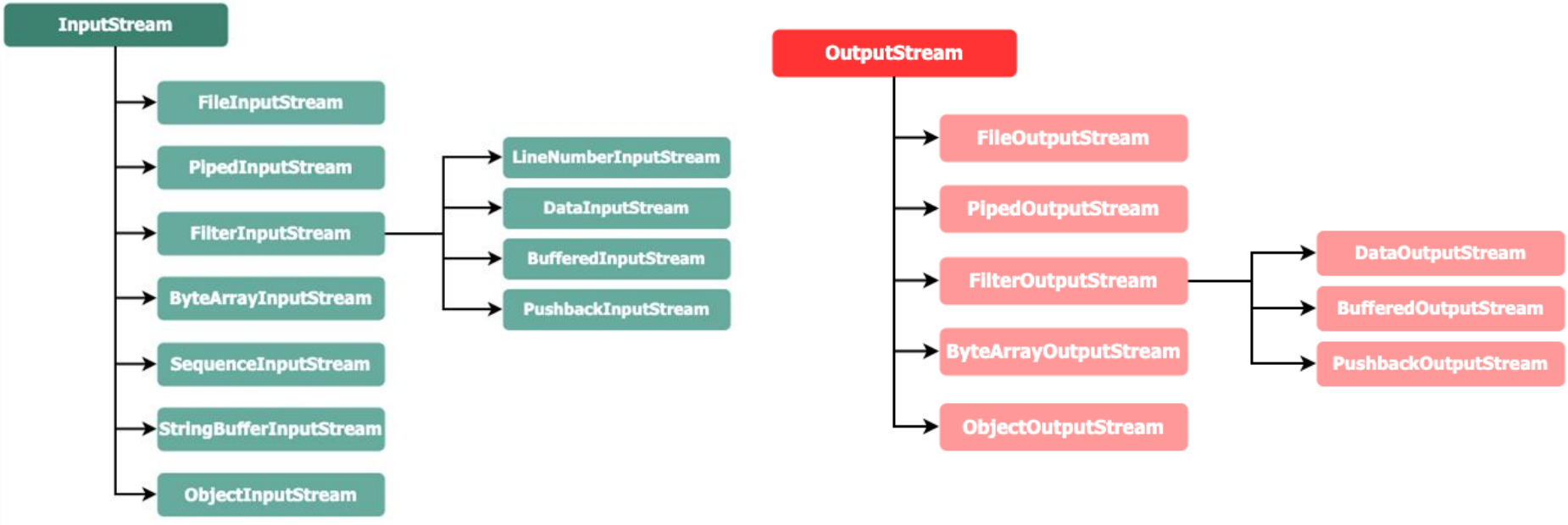
Flujos de Bytes (Byte Streams)

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

Clase	Función
ByteArrayOutputStream	Crea un espacio de almacenamiento en memoria. Todos los datos que se envían a este flujo se almacenan en él.
FileOutputStream	Se emplea para escribir datos en un fichero.
PipedOutPutStream	Cualquier información escrita en un objeto de esta clase deberá ser usada como entrada de un PipedInputStream asociado a él. Implementa el concepto de tubería.
FilterOutputStream	Proporciona funcionalidad adicional a otras OutputStreams

3. Flujos o Streams

Flujos de Bytes (Byte Streams)



3. Flujos o Streams

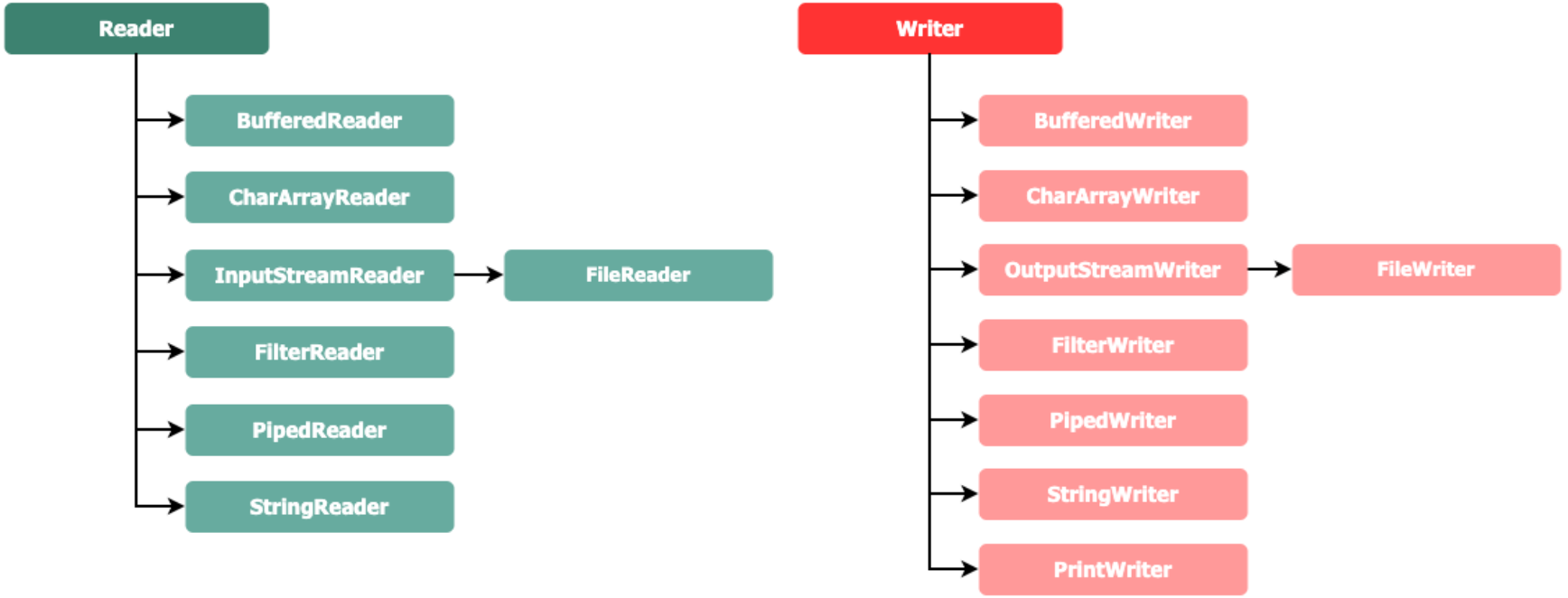
Flujos de Caracteres (Character Streams)

- Las clases **Reader** y **Writer** manejan flujos de caracteres Unicode.
- Es habitual usar estos flujos en combinación con los flujos que manejan Bytes.
  - Para ello contamos con las llamadas “clases puente”, que convierten de byteStream a characterStream
    - **InputStreamReader** → Convierte un InputStream en un Reader.
    - **OutputStreamWriter** → Convierte un OutputStream en un Writer.

- Las clases de flujos de caracteres más importantes:
  - **FileReader y FileWriter** → Lectura y escritura de caracteres en ficheros.
  - **CharArrayReader y CharArrayWriter** → Lectura y escritura de Array de caracteres.
  - **BufferedReader y BufferedWriter** → Se emplean para utilizar un buffer intermedio entre el programa y el fichero de origen o destino.

3. Flujos o Streams

Flujos de Caracteres (Character Streams)

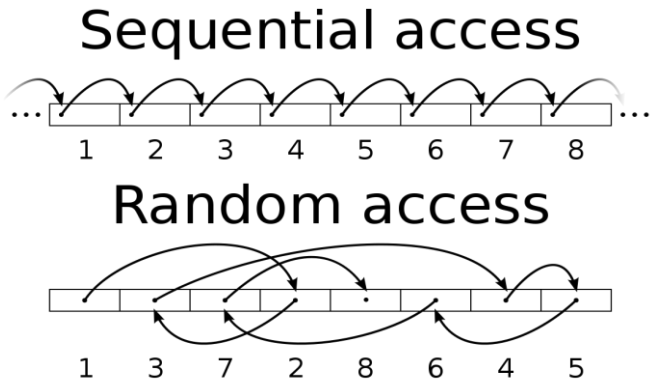


4. Formas de acceso a un fichero

- De forma genérica podemos decir que existen dos formas básicas de acceso a un fichero:
  - **Acceso Secuencial.**
    - Los datos se leen y escriben en orden, de forma similar a como se hace en una antigua cinta de video.
    - Para acceder a un dato es necesario leer primero todos los anteriores.
    - Solo se pueden insertar nuevos datos el final del fichero.
  - **Acceso directo o aleatorio.**
    - Permite el acceso directo a un dato en concreto, esto es, se puede acceder a la información en cualquier orden.
    - Los datos se almacenan en registros de tamaño conocido, por tanto podemos inferir la posición de un dato o registro y acceder a él para leerlo o modificarlo.

4. Formas de acceso a un fichero

- En Java, el acceso secuencial puede ser:
  - **Binario** → FileInputStream y FileOutputStream
  - **Caracteres** → FileReader y FileWriter
- Para el acceso aleatorio usaremos RandomAccessFile. (Se estudiará más adelante)



5. Operaciones sobre ficheros

- Las operaciones que podemos realizar sobre cualquier fichero (Independientemente de su tipo):
  - Creación de fichero
  - Apertura del fichero
  - Cierre del fichero
  - Lectura de datos del fichero
  - Escritura de datos en el fichero
- Una vez abierto, las operaciones que podremos realizar se clasifican en:
  - Altas
  - Bajas
  - Modificaciones
  - Consultas

5. Operaciones sobre ficheros

Operaciones sobre ficheros secuenciales

- **Consultas** → Para consultar un determinado registro es necesario comenzar la lectura desde el primer registro y continuar leyendo hasta llegar al deseado.
- **Altas** → Las nuevas altas deben hacerse al final del fichero.
- **Bajas** → Para dar de baja un registro es necesario:
  - Copiar todos los registros menos el que deseamos borrar a un fichero auxiliar
  - Borrar completamente el fichero de origen
  - Volcar el contenido del fichero auxiliar al fichero original.
- **Modificaciones** → Se realiza a través de un fichero auxiliar, de forma similar a como se realizar las bajas.

5. Operaciones sobre ficheros

Operaciones sobre ficheros aleatorios

- En los ficheros aleatorios, para poder llevar a cabo cualquier operación, antes es necesario localizar la posición del registro sobre el cual queremos operar.
  - Para posicionarnos en un registro es necesario aplicar una función de conversión, que de forma general estará relacionada con el tamaño del registro y con la clave del mismo.
    - Ejemplo:
      - Contamos con un fichero que contiene información de alumnos 3 campos (ID, nombre, apellido). Usaremos el ID como campo clave del mismo estableciendolo como campo autonumérico; de esta forma, para localizar a un alumno cuyo ID sea *N* deberemos acceder a la posición *tamaño*(N-1)*

5. Operaciones sobre ficheros

Operaciones sobre ficheros aleatorios

- **Consulta** → Aplicamos la función de conversión para obtener la dirección del registro y lo leemos.
- **Alta** → Aplicamos la función de conversión para obtener la dirección del registro y escribimos el registro en la posición obtenida.
- **Bajas** → Aplicamos la función de conversión para obtener la dirección del registro y:
  - Bajas lógicas → Marcamos el registro como "eliminado" mediante un campo booleano, aunque realmente lo mantenemos en la estructura.
  - Bajas físicas → Eliminamos el registro del fichero.
- **Modificaciones** → Aplicamos la función de conversión para obtener la dirección del registro y modificamos los valores deseados.

6. Ficheros de texto en Java

- Para trabajar con ficheros de texto usaremos:
  - **FileReader** para leer.
  - **FileWriter** para escribir.

- Siempre que trabajemos con estas clases debemos realizar una correcta gestión de excepciones ya que pueden producirse
  - **FileNotFoundException** → En caso de no encontrar el fichero.
  - **IOException** → Cuando se produce algún tipo de error de escritura.

6. Ficheros de texto en Java

FileReader

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

Método	Función
int read()	Lee un carácter y lo devuelve.
int read(char[] buf)	Lee hasta buf.length caracteres de datos. Los caracteres leídos se almacenan en buf
int read(char buf, int desplazamiento, int n)	Lee hasta n caracteres de datos a partir de la posición de desplazamiento.

6. Ficheros de texto en Java

FileReader

- En Java para abrir y leer un fichero de texto debemos
  - Crear una instancia de la clase File
  - Crear un flujo de entrada con la clase FileReader
  - Realizamos las operaciones de lectura pertinentes.
  - Cerramos el fichero con el método .close()

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

```
//Creamos el fichero
File f= new File("myFiles/SampleFile.txt");


try {
    //Creamos el stream de entrada
    FileReader fr= new FileReader(f);
    int caracter;

    //Leemos el fichero caracter a caracter
    while((caracter=fr.read())!=-1) {
        System.out.print((char)caracter);
    }
    //Cerramos el fichero
    fr.close();

} catch (FileNotFoundException e) {

    e.printStackTrace();
} catch (IOException e) {

    e.printStackTrace();
}
```

6. Ficheros de texto en Java

FileWriter

- La siguiente tabla muestra los métodos empleados por la clase FileWriter para escribir en ficheros:

Método	Función
void write(int c)	Escribe un carácter
void write(char[] buf)	Escribe un array de caracteres
void write(char buf, int desplazamiento, int n)	Escribe n caracteres de datos a partir de la posición de desplazamiento.
void write(String str)	Escribe una cena de caracteres
void append(char c)	Añade un carácter al final del fichero.

6. Ficheros de texto en Java

FileWriter

- En Java, escribir en un fichero de texto:
  - Crear una instancia de la clase File
  - Crear un flujo de salida con FileWriter
  - Realizamos las operaciones de escritura
  - Cerramos el fichero con el método .close()

```
public static void main(String[] args) {
    //Creamos el fichero
    File f= new File("myFiles/newFile001.txt");
    try {
        //Creamos un stream de salida
        FileWriter fw= new FileWriter(f);
        String cadena="Esta es mi primera escritura en disco.";
        //Escribimos en el fichero
        fw.write(cadena);
        //Cerramos el fichero
        fw.close();
    } catch (IOException e) {

        e.printStackTrace();
    }

}
```

6. Ficheros de texto en Java

FileWriter

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

¿Con qué formato crees que se escribirá?

```
File f= new File("myFiles/newFile002.txt");
try {

    FileWriter fw= new FileWriter(f);
    String provincias[]={
        "Albacete", "Avila", "Badajoz",
        "Caceres", "Huelva", "Jaén", "Toledo"
    };
    for (int i = 0; i < provincias.length; i++) {
        fw.write(provincias[i]);
    }

    //Cerramos el fichero
    fw.close();
} catch (IOException e) {

    e.printStackTrace();
}
```

6. Ficheros de texto en Java

FileWriter

- Debemos tener en cuenta que si el fichero ya contenía información anteriormente, al realizar la nuevas escritura sobrescribiremos su contenido.
  - Si nuestra intención es agregar nueva información debemos crear el stream de salida de la siguiente manera:

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

El segundo parámetro del constructor es el parámetro de append:

true → Añade la información al final del fichero.

false → Borra el contenido anterior para introducir el nuevo.

6. Ficheros de texto en Java

BufferedReader

- Con la clase BufferedReader podemos hacer una gestión más avanzada de la lectura de fichero.
  - Con el método readLine() podremos leer una línea de nuestro documento, especialmente útil si el separador de registros es el carácter \n.
    - readLine() devuelve una String con la cadena leída o null si nos encontramos al final del fichero (EOF).
- Para crear un BufferedReader necesitamos partir de la clase FileReader. Esto es debido a que la clase que realmente gestiona el stream es FileReader. BufferedReader se vale de ella para ir haciendo una lectura con buffer.

```
public static void main(String[] args) {
    File f= new File("myFiles/newFile002.txt");
    try {
        BufferedReader br= new BufferedReader(new FileReader(f));
        String linea="";
        while((linea=br.readLine())!=null) {
            System.out.println(linea);          }
        br.close();

    } catch (IOException e) {

        e.printStackTrace();
    }
```

6. Ficheros de texto en Java

BufferedWriter

- Del mismo modo contamos con la clase BufferedWriter, que nos permite realizar escrituras línea a línea.
  - El método .write() escribe una línea en nuestro fichero.
  - El método .newLine() escribe un salto de línea.

```
File f= new File("myFiles/newFile003.txt");
try {
    BufferedWriter bw= new BufferedWriter(new FileWriter(f));
    String provincias[]={
            "Albacete", "Avila", "Badajoz",
            "Caceres", "Huelva", "Jaén", "Toledo"
        };
    for (int i = 0; i < provincias.length; i++) {
        bw.write(provincias[i]);
        bw.newLine();
    }

} catch (IOException e) {

    e.printStackTrace();
}
```

6. Ficheros de texto en Java

PrintWriter

- La clase PrintWriter posee los métodos **print(String)** y **println(String)** cuyo comportamiento se asemeja a los de System.out.
- Ambos reciben un String y lo escriben en el fichero. println, además, inserta un salto de línea.
- Para construir un PrintWriter, una vez más, necesitamos un FileWriter.

```
File f= new File("myFiles/newFile004.txt");
try {
    PrintWriter pw= new PrintWriter(new FileWriter(f));
    String provincias[]={
            "Albacete", "Avila", "Badajoz",
            "Caceres", "Huelva", "Jaén", "Toledo"
        };
    for (int i = 0; i < provincias.length; i++) {
        pw.println(provincias[i]);

    }
    pw.close();

} catch (IOException e) {

    e.printStackTrace();
}
```

Dudas y preguntas