

Unidad 1. ACCESO A LOS FICHEROS

Parte 1. Introducción, revisión de Java y acceso básico a archivos

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

CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)

Abelardo Martínez

Año 2023-2024

Créditos



- Notas realizadas por Abelardo Martínez.
- Basado y modificado a partir de Sergio Badal (www.sergiobadal.com).
- Las imágenes e iconos utilizados están protegidos por la licencia [LGPL](#) y se han obtenido de:
 - https://commons.wikimedia.org/wiki/Crystal_Clear
 - <https://www.openclipart.org>

Contenido

1. ¿POR QUÉ UTILIZAR JAVA?
2. RECURSOS RECOMENDADOS
3. INSTALACIÓN DE JAVA E IDE
4. FUNDAMENTOS DE JAVA
5. FICHEROS: TIPOS Y ACCESO
6. ACCESO A ARCHIVOS CON JAVA
 1. Clase de archivo
 2. Constructores
 3. Excepciones
 4. Lectura de archivos de texto en Java
 5. Escritura de archivos de texto en Java
7. ACTIVIDADES PROPUESTAS
8. BIBLIOGRAFÍA



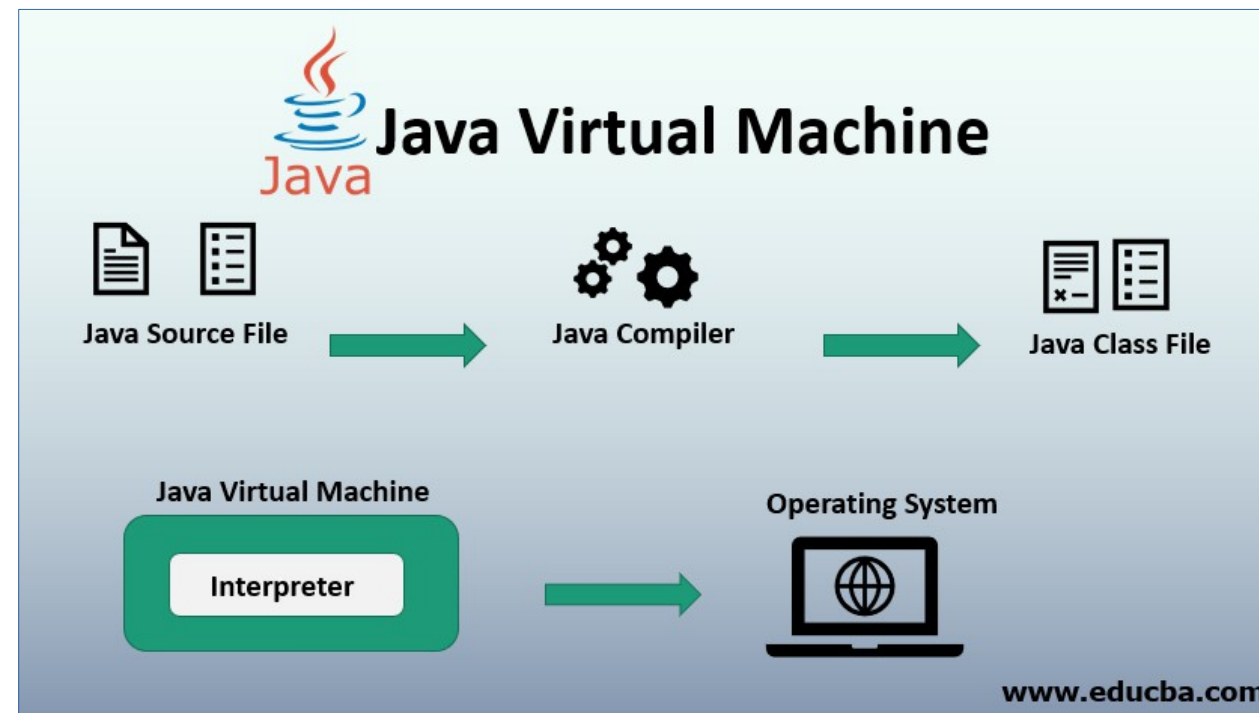
1. ¿POR QUÉ UTILIZAR JAVA?

Lenguaje Java

Java es un **LENGUAJE de programación de propósito general, basado en clases y orientado a objetos**, diseñado para tener menos dependencias de implementación. También es una PLATAFORMA informática para el desarrollo de aplicaciones.

Ventajas:

- 1) Java es fácil de escribir y de ejecutar -ésta es la fuerza fundamental de Java y la razón por la que muchos desarrolladores programan en él-. Cuando escribes Java una vez, puedes ejecutarlo prácticamente en cualquier lugar y en cualquier momento.
- 2) Java permite crear aplicaciones completas que pueden ejecutarse en un solo ordenador o distribuirse entre servidores y clientes de una red.
- 3) Como resultado, puede utilizarlo para crear fácilmente aplicaciones móviles o ejecutarlas en aplicaciones de escritorio que utilicen diferentes sistemas operativos y servidores, como Linux o Windows.



Lenguaje Python

- Java ya no es el rey de los lenguajes OO.
- Por primera vez desde que comenzó a elaborarse el índice TIOBE hace casi 20 años, Java y C no ocupan las dos primeras posiciones.
- Python, un lenguaje nacido el mismo año que Java, viene para quedarse.

Ventajas:

- Se tarda menos en escribir en un editor de texto, y las líneas de código son de tres a cinco veces más cortas que en Java.
 - En Java podría necesitar una licencia comercial para mantener las actualizaciones periódicas de seguridad del lenguaje.
- Desventajas:**

- Python es más lento que Java cuando se compila.
- Java está mejor equipado para el desarrollo móvil.



Aug 2023	Aug 2022	Change	Programming Language		Ratings	Change
1	1			Python	13.33%	-2.30%
2	2			C	11.41%	-3.35%
3	4	▲		C++	10.63%	+0.49%
4	3	▼		Java	10.33%	-2.14%
5	5			C#	7.04%	+1.64%
6	8	▲		JavaScript	3.29%	+0.89%
7	6	▼		Visual Basic	2.63%	-2.26%
8	9	▲		SQL	1.53%	-0.14%
9	7	▼		Assembly language	1.34%	-1.41%
10	10			PHP	1.27%	-0.09%

[Índice TIOBE actual](#)

Para más información:
[Java frente a Python](#)

2. RECURSOS RECOMENDADOS

¿Cuál es su situación?



Nivel	Descripción
0	No tengo ni idea de programación. Prueba los 2 cursos básicos de Alex y continúa con el siguiente nivel si lo necesitas.
1	Mi última línea de código fue hace años. No te preocupes, prueba el vídeo de Alex de 14 minutos y decide el siguiente paso.
2	Soy bueno codificando pero es mi primera vez con OO. Prueba con los cursos 2 y 3 de Alex y me cuentas.
3	Puedo considerarme un experto en OO, pero ninguna línea escrita en Java . No pasa nada. Una vez que montas en bici, todas las bicis son casi iguales.
4	Puedo considerarme un experto en programación Java. Buenas noticias, ¡relájate y hasta la semana que viene!





Recursos Java

Recursos recomendados





- Java Básico (Alex Lee) Cursos 1º y 2º: [Enlace 1](#)
- Refresca tu mente en sólo 14 minutos. (Alex Lee): [Enlace 2](#)
- Java Avanzado (Alex Lee) Cursos 3º y 4º: [Enlace 3](#)

adicionales (en español)

- Un libro de un profesor de PRG DAM/DAW (sólo en "papel") (200 actividades resueltas) (15€): [Enlace 1](#)
- Un libro en línea (sólo en PDF) (350 actividades resueltas) (5€): [Enlace 2](#)
- Un curso en línea (sólo en YouTube) (gratuito): [Enlace 3](#)

3. INSTALACIÓN DE JAVA E IDE

IDE para crear aplicaciones Java:

IDE	Descripción
<div>Eclipse</div> <div></div>	<p>Es uno de los Entornos de Desarrollo Integrado (IDE) más utilizados para crear aplicaciones Java. También puede utilizarse para desarrollar aplicaciones en otros lenguajes de programación mediante plug-ins.</p> <p>https://www.eclipse.org Cómo instalar</p>
<div>NetBeans</div> <div></div>	<p>Es un entorno de desarrollo integrado (IDE) para Java y funciona en Windows, macOS, Linux y Solaris. Dispone de extensiones para otros lenguajes.</p> <p>https://netbeans.apache.org https://www.geeksforgeeks.org/how-to-install-netbeans-java-ide-on-windows/ https://www.howtoforge.com/how-to-install-netbeans-ide-on-ubuntu-2004/</p>
<div>Código de Visual Studio</div> <div></div>	<p>Es un editor de código fuente creado por Microsoft para Windows, Linux y macOS y puede utilizarse con una gran variedad de lenguajes de programación. Para instalar el programa debe descargar el paquete DEB (Linux) o el paquete EXE (Windows).</p> <p>https://code.visualstudio.com</p>
<div>IntelliJ</div> <div></div>	<p>IntelliJ IDEA es un entorno de desarrollo integrado (IDE) escrito en Java para desarrollar software informático escrito en Java, Kotlin, Groovy y otros lenguajes basados en JVM. Está desarrollado por JetBrains (antes conocido como IntelliJ) y está disponible como una edición comunitaria con licencia Apache 2, y en una edición comercial propietaria. Ambas pueden utilizarse para el desarrollo comercial.</p> <p>https://www.jetbrains.com/es-es/idea/download/other.html</p>

IDE Eclipse

Java: https://java.com/en/download/help/download_options.html

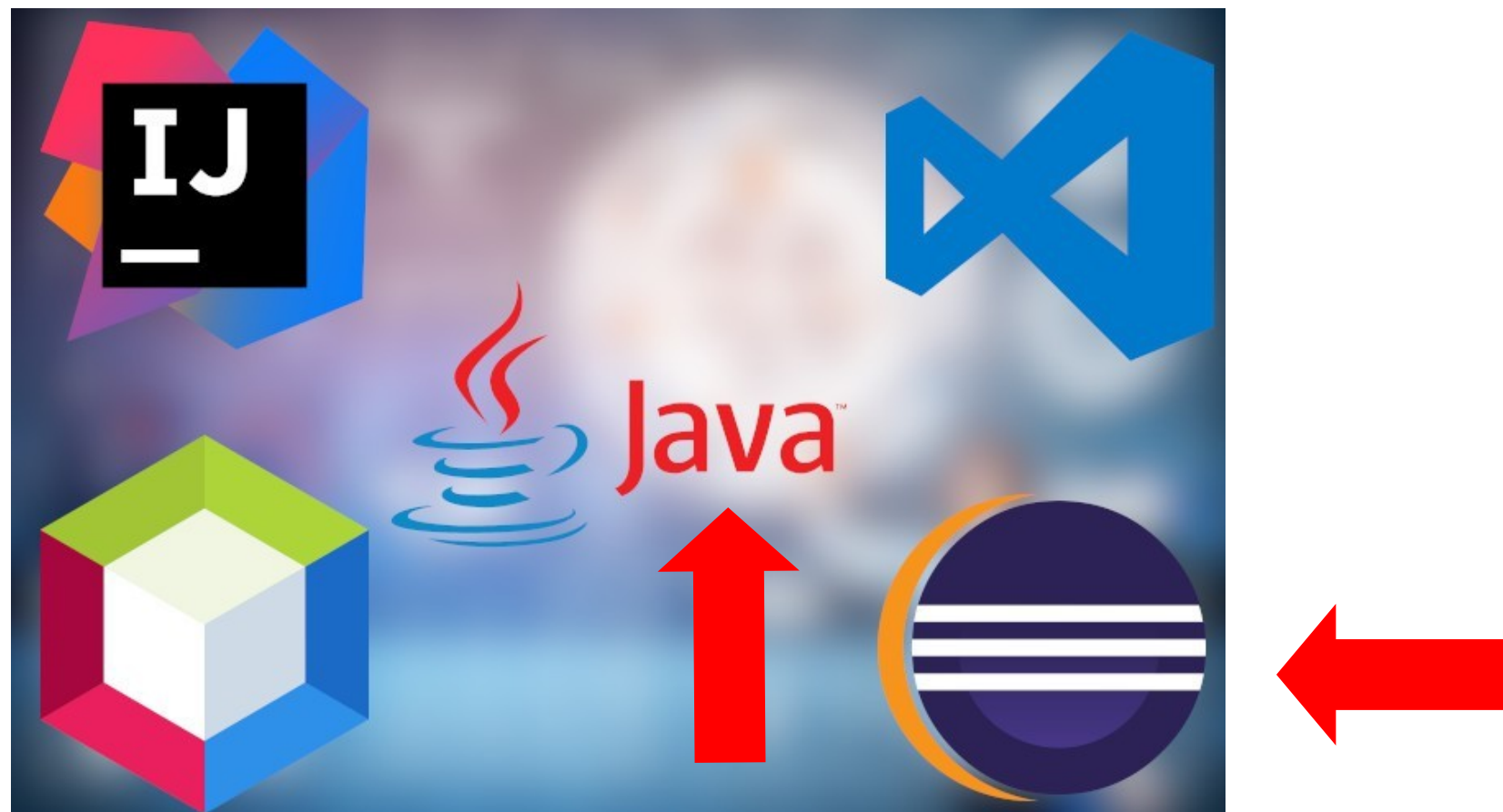
Eclipse: <https://wiki.eclipse.org/Eclipse/Installation>

Ayuda adicional en español:

Eclipse en Linux: <https://conpilar.es/como-instalar-eclipse-ide-en-ubuntu-20-04/>

Habilitar interfaz gráfica en Eclipse:

<https://www.cablenaranja.com/java-como-activar-el-editor-visual-en-eclipse/>



4. FUNDAMENTOS DE JAVA

Historia de Java

- **Historia completa del lenguaje de programación Java:**

<https://www.geeksforgeeks.org/the-complete-history-of-java-programming-language/>

- **Breve historia de Java:** <https://dzone.com/articles/a-short-history-of-java>

- **Java (lenguaje de programación):**

[https://en.wikipedia.org/wiki/Java_\(lenguaje_de_programaci3n\)#:~:text=Java%20fue%20originalmente%20desarrollado%20por,by%20Sun%20bajo%20licencias%20de%20propiedad.](https://en.wikipedia.org/wiki/Java_(lenguaje_de_programaci3n)#:~:text=Java%20fue%20originalmente%20desarrollado%20por,by%20Sun%20bajo%20licencias%20de%20propiedad.)



Estructura básica del código

Basic code structure

text file named HelloWorld.java

name

main() method

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.print("Hello, World");
        System.out.println();
    }
}
```

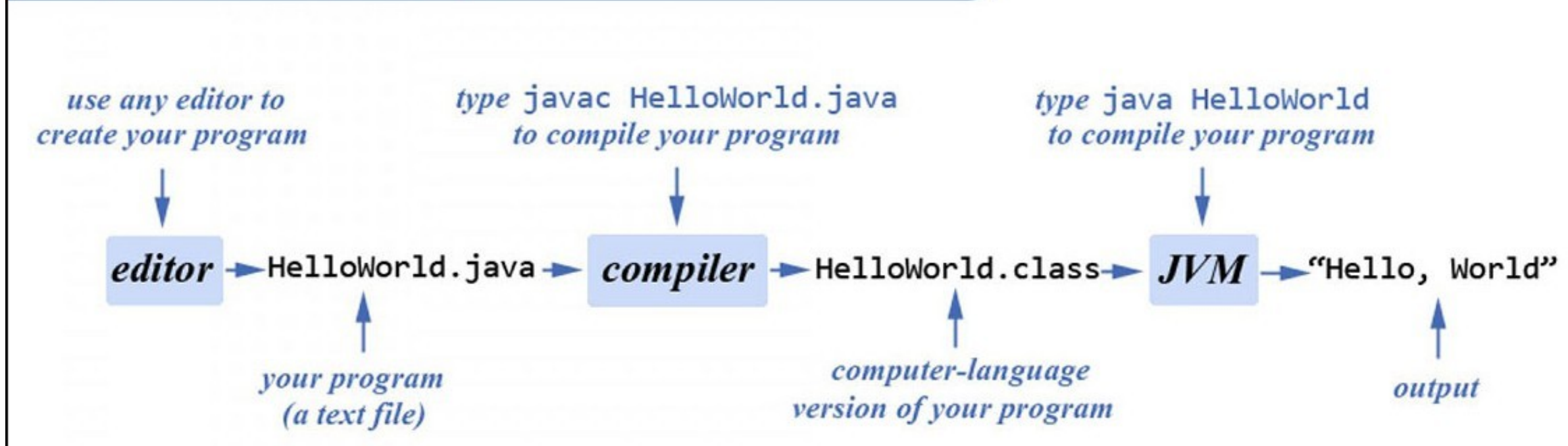
statements

body



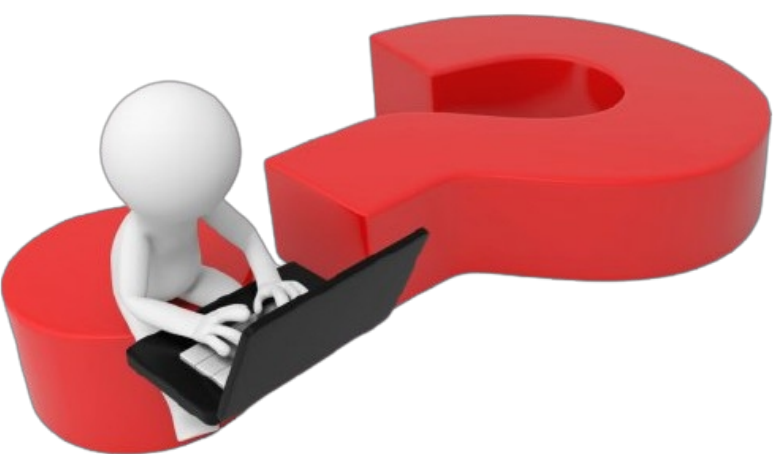
Entrada y salida. Tipos de datos

Input and output



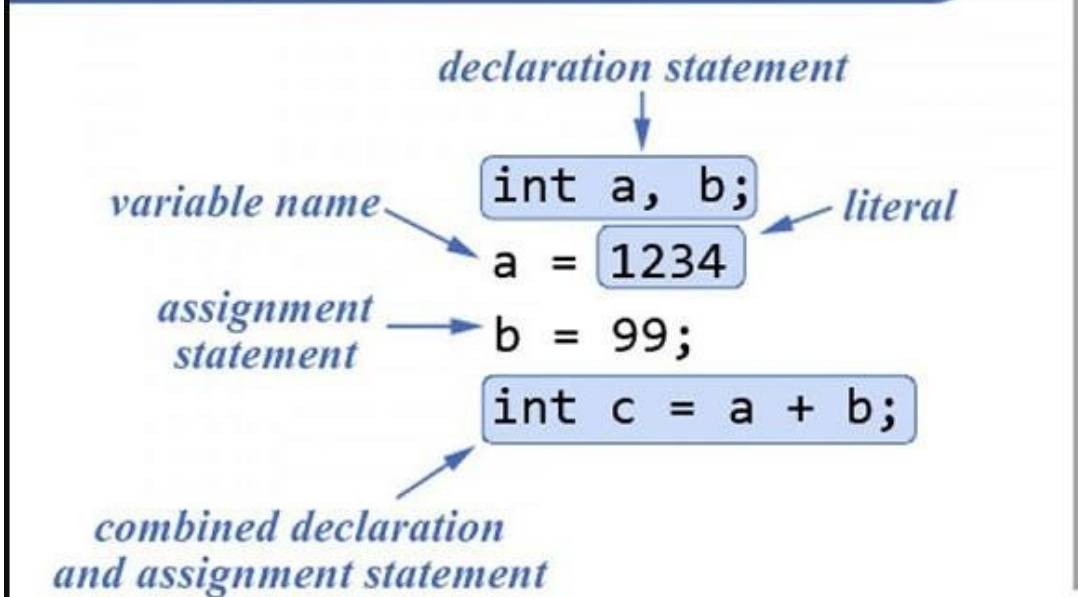
Date types

<i>type</i>	<i>set of values</i>	<i>common operators</i>	<i>sample literal values</i>
int	intergers	+ - * / %	99 -12 2147483647
double	floating-point numbers	+ - * /	3.14 -2.5 6.022e23
boolean	boolean values	&& !	true false
char	characters		'A' '1' '%' '/n'
String	sequence of characters	+	"AB" "Hello" "2.5"



Estado de la asignación. Booleanos. Comparación

Assignment status



Booleans

<i>values</i>	true or false		
<i>literals</i>	true false		
<i>operations</i>	and	or	not
<i>operators</i>	&&		!

Comparison

<i>op</i>	<i>meaning</i>	true	false
<code>==</code>	<i>equal</i>	<code>2 == 2</code>	<code>2 == 3</code>
<code>!=</code>	<i>not equal</i>	<code>3 != 2</code>	<code>2 != 2</code>
<code><</code>	<i>less than</i>	<code>2 < 13</code>	<code>2 < 2</code>
<code><=</code>	<i>less than or equal</i>	<code>2 <= 2</code>	<code>3 <= 2</code>
<code>></code>	<i>greater than</i>	<code>13 > 2</code>	<code>2 > 13</code>
<code>>=</code>	<i>greater than or equal</i>	<code>3 >= 2</code>	<code>2 >= 3</code>



Recursos y ejercicios para repasar

- Los tutoriales de Java™. <https://docs.oracle.com/javase/tutorial/index.html>
- W3resource. Ejercicios de Programación Java, Práctica, Solución. <https://www.w3resource.com/java-exercises/>
- Tutorial de Java. <https://www.w3schools.com/java/default.asp>
- EJERCICIOS DE CODIFICACIÓN. Ejercicios de programación en Java. <https://code-exercises.com>
- Practica 133 ejercicios en Java. <https://exercism.org/tracks/java/exercises>

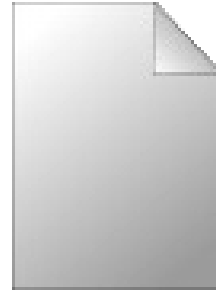
Nivel Java	Ejercicios recomendados de cada sección en función de la duración de la última codificación		
	> 1 año	> 3 meses	< 3 meses
Básico	5	4	3
Medio	4	3	2
Avanzado	3	2	1

List of Java Exercises:

- [Basic Exercises Part-I \[150 Exercises with Solution \]](#)
- [Basic Exercises Part-II \[93 Exercises with Solution \]](#)
- [Data Types Exercises \[15 Exercises with Solution \]](#)
- [Conditional Statement Exercises \[32 Exercises with Solution \]](#)
- [Array \[74 Exercises with Solution \]](#)
- [String \[107 Exercises with Solution \]](#)
- [Date Time \[44 Exercises with Solution \]](#)
- [Methods \[16 Exercises with Solution \]](#)
- [Numbers \[28 Exercises with Solution \]](#)
- [Input-Output-File-System \[18 Exercises with Solution \]](#)
- [Collection \[126 Exercises with Solution \]](#)

5. FICHEROS: TIPOS Y ACCESO

Tipo de archivos



Los datos, es decir, los objetos almacenados en memoria, se pierden cuando se cierra la aplicación. Si queremos asegurarnos de que los datos de la aplicación son **persistentes**, tendríamos que guardarlos cuando se cierre la aplicación y volver a cargarlos cuando la volvamos a ejecutar. Para ello utilizamos ficheros.

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).

Hay muchas formas de almacenar los datos de una aplicación, y cada una tiene sus ventajas e inconvenientes. Generalmente, **almacenamos los datos de la aplicación de una de las siguientes maneras:**

- **Archivos de texto con una estructura plana** (por ejemplo, archivos .txt y .csv) (**¡YA!**)
- Archivos de texto con jerarquía interna (json, XML, HTML, etc.) (**próximamente**)

- Archivos binarios (volcado simple de memoria en un archivo) (no en este módulo)
- Bases de datos (Unidad 2)

Archivos de texto frente a archivos binarios

Archivos de texto. Están diseñados para ser leídos por seres humanos y pueden leerse o escribirse con un editor. Los archivos de texto también suelen denominarse archivos planos o archivos [ASCII](#).



- **Positivo:** suelen ser los mismos en todos los ordenadores, por lo que pueden pasar de un ordenador a otro.
- **Negativo:** no son tan eficientes de procesar como los binarios.

Archivos binarios. Están diseñados para ser leídos por programas y consisten en una secuencia de dígitos binarios.



- **Positivo:** son más eficientes de procesar que los archivos de texto. A diferencia de la mayoría de archivos binarios, los archivos binarios Java tienen la ventaja de ser independientes de la plataforma.
- **Negativo:** están diseñados para ser leídos en el mismo tipo de ordenador y con el mismo idioma que el ordenador que creó el archivo. No es posible visualizar

directamente el contenido.

Archivos de texto frente a archivos binarios

Aquí puedes consultar un amplio estudio sobre ambos tipos de archivos con w/o buffering:

https://funnelgarden.com/java_read_file/

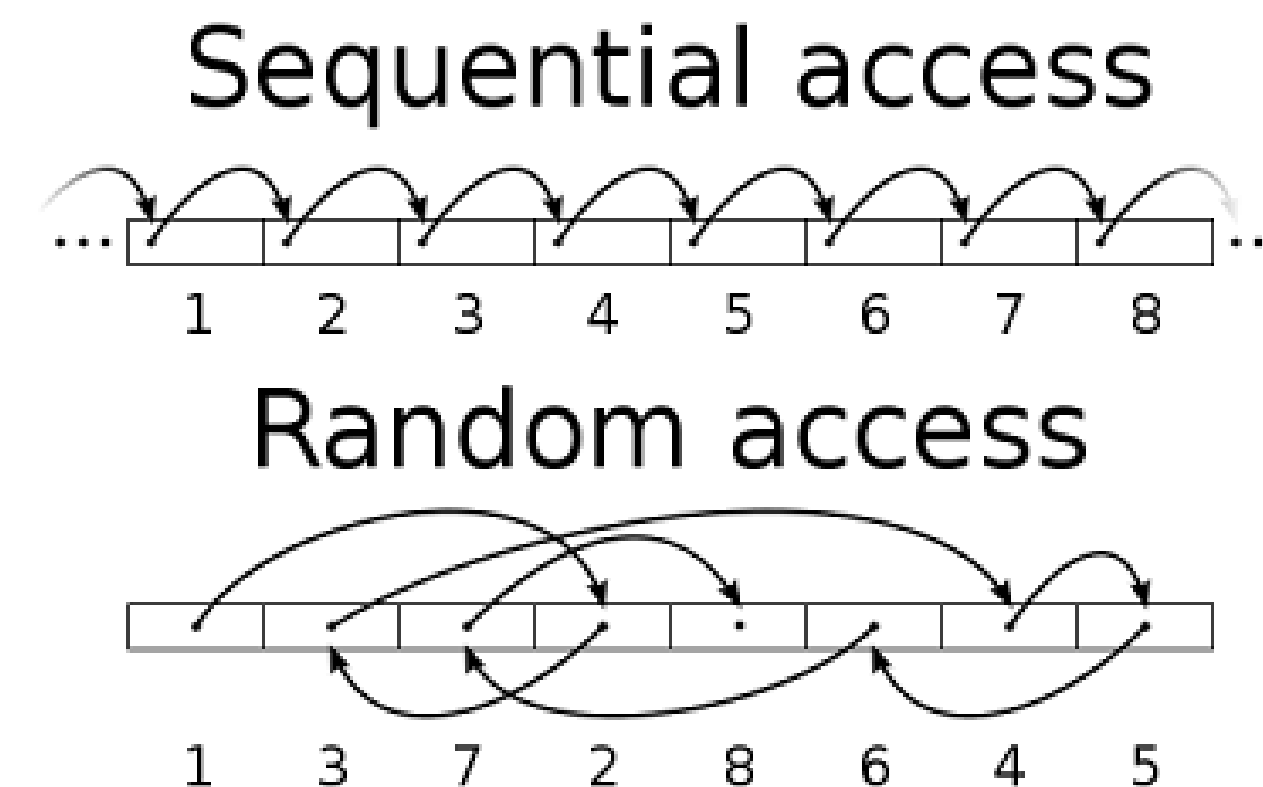
Reading Method	Time to read data file (in milliseconds)						
	1KB	10KB	100KB	1MB	10MB	100MB	1GB
FileReader.read()	3	9	29	95	512	4,279	43,635
BufferedReader.readLine()	1	2	8	30	81	492	4,498
FileInputStream.read()	2	13	133	1,247	12,603	124,413	1,261,190
BufferedInputStream.read()	0	1	6	24	122	1,138	24,643
Files.readAllBytes()	3	3	4	4	15	102	969
Files.readAllLines()	5	6	12	39	120	866	OutOfMemoryError
Files.lines()	26	31	35	59	112	465	3,588
Scanner.nextLine()	6	15	38	107	376	2,346	21,539
Commons-FileUtils.readLines()	29	29	35	61	143	823	OutOfMemoryError
Guava-Files.readLines()	43	44	52	96	243	1,493	OutOfMemoryError

Tipo de acceso

Aunque un mecanismo de acceso **secuencial** recorre los registros de un fichero de forma lineal, el acceso **aleatorio** en un fichero nos permite acceder directamente a registros individuales sin buscar en otros registros.

Los archivos planos en general no están pensados para ser accedidos de esta manera; además, Java no impone ninguna estructura a un archivo. En consecuencia, no existe una técnica específica para crear archivos de acceso aleatorio.

Para más información sobre el acceso aleatorio puede consultar este enlace: <https://www.developer.com/database/random-file-access-using-java/>



Flujos de entrada/salida

Un **stream** es un objeto que permite el flujo de datos entre un programa y algún dispositivo de E/S o archivo:

- Si los datos se introducen en un programa, el flujo se denomina **flujo de entrada**.
- Si los datos salen de un programa, el flujo se denomina **flujo de salida**.

Los flujos de entrada pueden proceder del teclado, de un archivo, etc.

- **System.in** es un **flujo de entrada** que se conecta al **teclado**

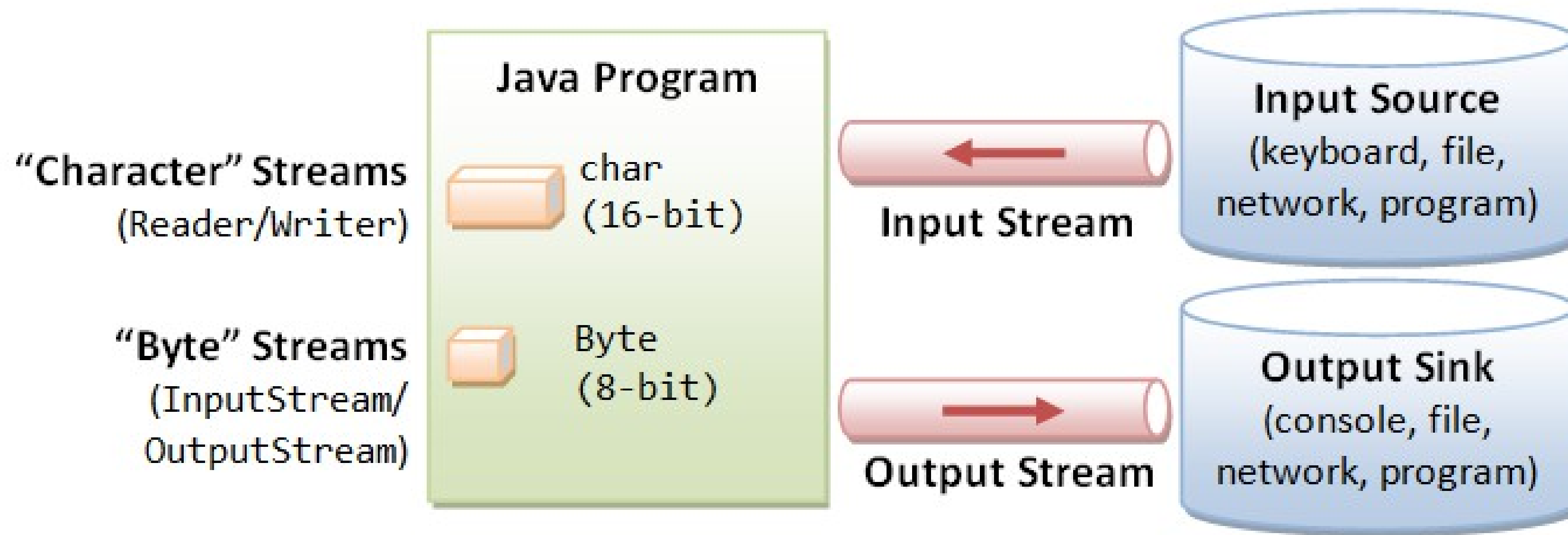
```
Teclado Scanner = nuevo Scanner(System.in);
```

Los flujos de salida pueden ir a una pantalla, a un archivo, etc.

- **System.out** es un **flujo de salida** que se conecta a la **pantalla/consola**

```
System.out.println("Flujo de salida");
```

Flujos de entrada/salida



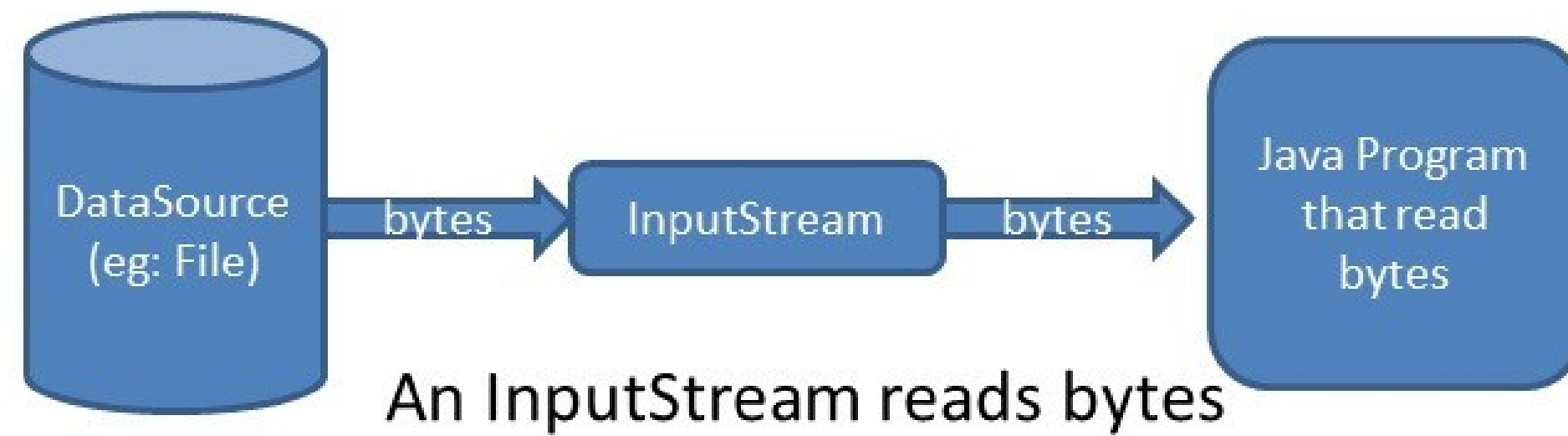
Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

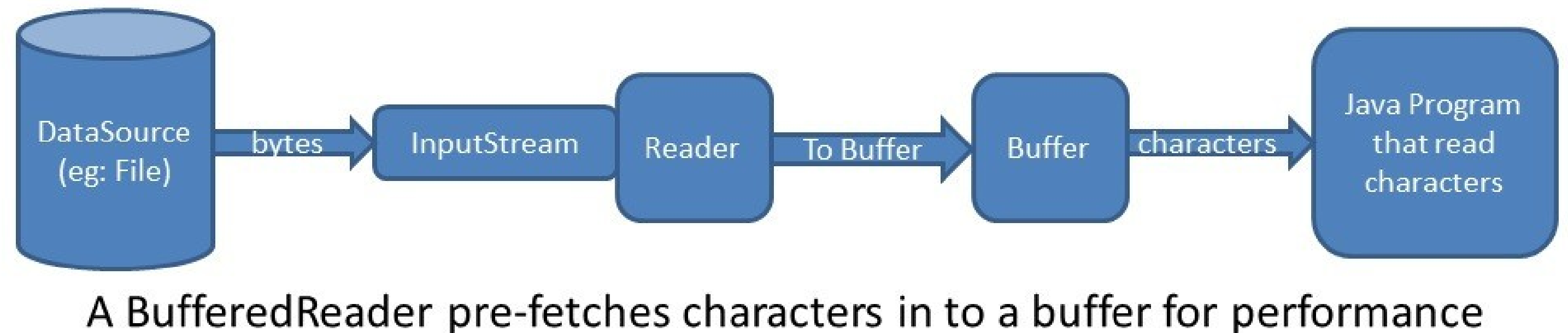
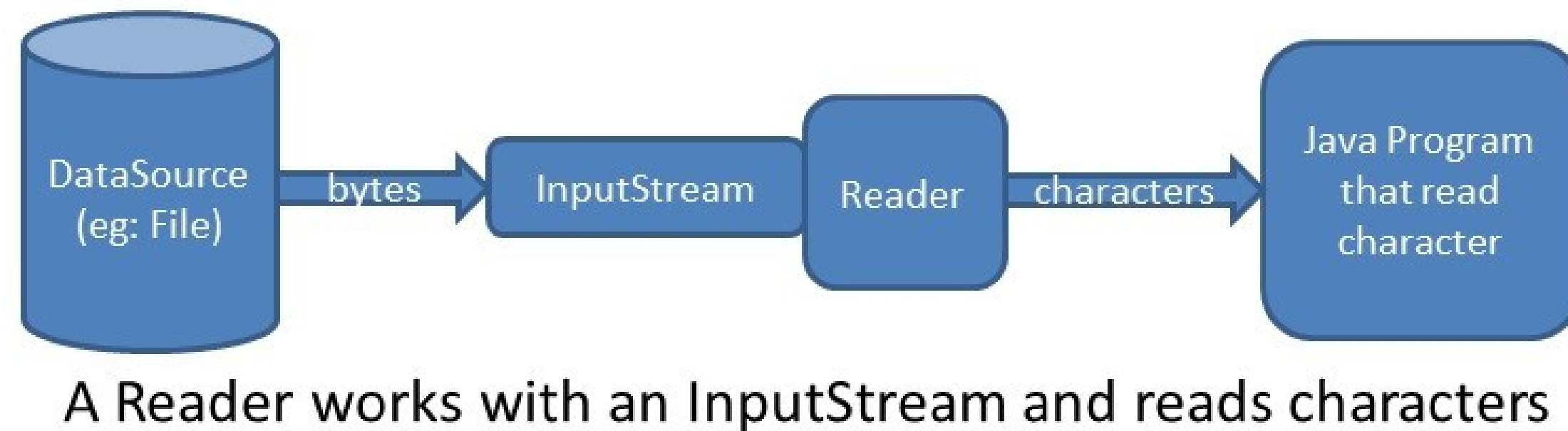
External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

Flujos de entrada/salida (mediante búferes)

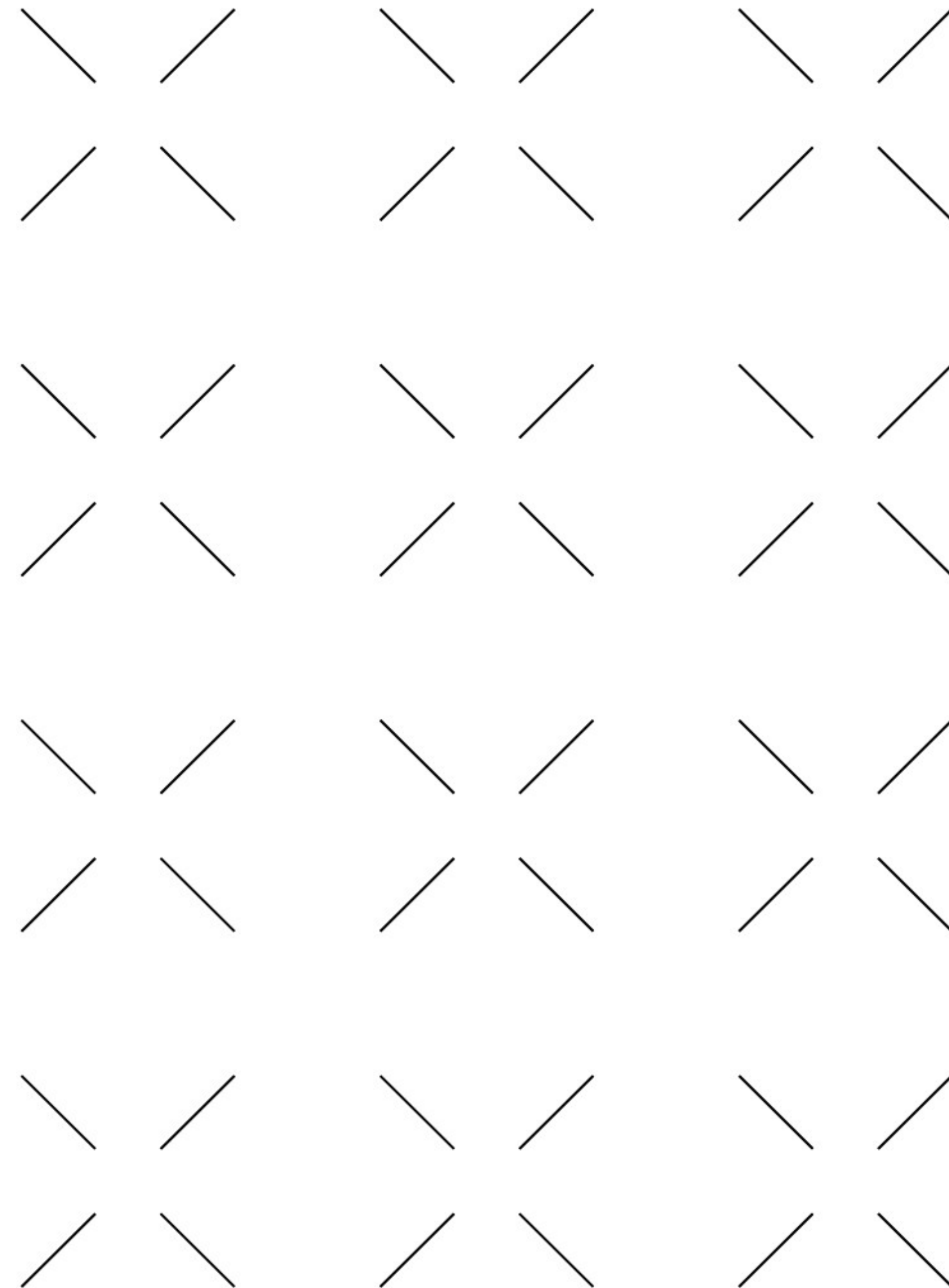


Ejemplo de uso de
búferes para leer datos
binarios



6. ACCESO A ARCHIVOS CON JAVA

6.1 Clase de archivo



Archivos y carpetas



En esta lección presentaremos la **clase File**, sus métodos y usos.

- La clase File ha estado en Java desde la primera versión.
- Encapsula prácticamente toda la funcionalidad para gestionar un sistema de archivos organizado en árboles de directorios.
- Esta clase no representa el contenido de ningún fichero, sino la ruta donde se encuentra. Por lo tanto, la clase **representa tanto un archivo como una carpeta**.
- La clase File no tiene ningún tipo de utilidad para obtener una lista ordenada.

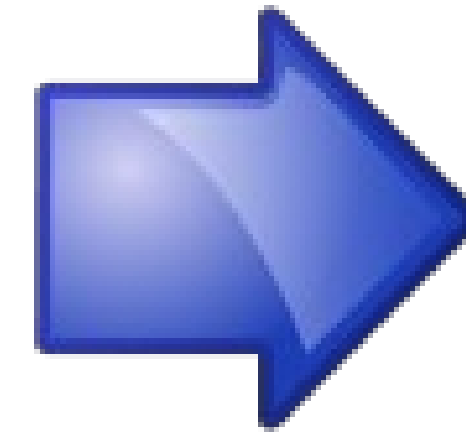
Ventajas

- Con esta clase conseguimos la independencia total de la notación OS.
- La clase Archivo, en colaboración con la máquina virtual, se adaptará de forma transparente al sistema operativo.

Métodos de la clase File en Java

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

- GET
- SET
- CAN
- IS
- FUNCIONAL



Estos son sólo algunos de los métodos. Para más información, consulte la documentación de Oracle en:

<https://docs.oracle.com/javase/8/docs/api/java/io/File.html>

Métodos GET

| GET methods

We have the following get methods available, which we'll all try:

- `getAbsolutePath(): String` - Returns the absolute path to the file
- `getCanonicalPath(): String` - Returns the canonized file path
- `getFreeSpace(): long` - Returns the number of free bytes in the partition where the file is located
- `getName(): String` - returns the file name
- `getParent(): String` - returns the absolute path to the parent, or `null` if the file itself is the parent
- `getParentFile(): File` - Returns an instance of the `File` class representing the parent of the current file
- `getPath(): String` - returns the path to the file (since we don't know in which format we get it, it's better to use `getCanonicalPath()` instead)
- `getTotalSpace(): long` - returns the total number of bytes in the partition where the file is located
- `getUsableSpace(): long` - Returns the number of used bytes for the current virtual machine; the result is more accurate than from the `getFreeSpace()` method

Métodos SET

| SET methods

Set methods, as the name suggests, set some properties to the files. These are:

- `setExecutable(boolean executable, boolean ownerOnly): boolean` - sets whether the file is executable; the second parameter is optional (there's an overload where this parameter is set to `true` automatically); if the second parameter is `true`, then the executability is set for the current user only
- `setLastModified(long time): boolean` - sets the date when the file has been lastly modified
- `setReadable(boolean readable, boolean ownerOnly): boolean` - sets whether the file can be read; the second parameter works the same as with the first method
- `setReadOnly(): boolean` - a one-way method to set a file as read-only -> it won't be possible to write it anymore
- `setWritable(boolean writable, boolean ownerOnly): boolean` - sets whether it's possible to write to the file; the second parameter works the same as with the first method

Métodos CAN e IS

| CAN methods

The Can methods are the following:

- `canExecute(): boolean` - Returns `true` if the file can be executed, `false` otherwise
- `canRead(): boolean` - return `true` if the file is readable, `false` otherwise
- `canWrite(): boolean` returns `true` if writable, `false` otherwise

| IS methods

Using the "IS" methods we can ask the following questions:

- `isAbsolute(): boolean` - Returns `true` if the instance was created using an absolute path
- `isDirectory(): boolean` - Returns `true` if it's a folder
- `isFile(): boolean` - Returns `true` if it's a file
- `isHidden(): boolean` - Returns `true` if the file is hidden

Métodos FUNCIONALES

| FUNCTIONAL methods

We're now missing only the "functional" that do something with the file itself and which we'll use most often.

- `toURI(): URI` - Creates a URI from the file instance used
- `createNewFile(): boolean` - creates a new file if it doesn't exist; returns `true` if the file was created, `false` otherwise
- `delete(): boolean` - deletes the file; returns `true` if the file was deleted, `false` otherwise
- `deleteOnExit(): void` - deletes the file only after the program has finished
- `exists(): boolean` - returns `true` if the file exists, `false` otherwise
- `length(): long` - returns the file size in bytes
- `list(): String []` - returns an array of absolute paths of the files in the folder
- `listFiles(): File []` - Returns an array of file instances in the folder
- `mkdir(): boolean` - attempts to create the folder; returns `true` if the folder was created, `false` otherwise
- `mkdirs(): boolean` - attempts to create all folders in the path; returns `true` if all the folders have been created, `false` otherwise
- `renameTo(File dest): boolean` - renames the file to a new name; can be understood as "moving" a file from one location to another; this method is platform dependent; cannot be used to move the file between two file systems
- `toPath(): Path` - creates a new `Path` instance, which we'll discuss in the next lesson

Nueva API de E/S (nio)

Los problemas de la API de archivos ha evolucionado en la **nueva API de IO (nio)** y se puede ir más lejos este tema en el enlace: <https://jenkov.com/tutorials/java-nio/nio-vs-io.html>

Orientado al flujo IO: (A → ventaja, D → desventaja).

(A) Se leen uno o varios bytes a la vez, de un flujo.

(A) No se almacenan en caché en ningún sitio.

(D) No puedes avanzar y retroceder en los datos en un flujo. Si necesitas hacerlo, tendrás que almacenarlo primero en caché en un buffer.

Búfer de NIO orientado: (A → ventaja, D → desventaja).

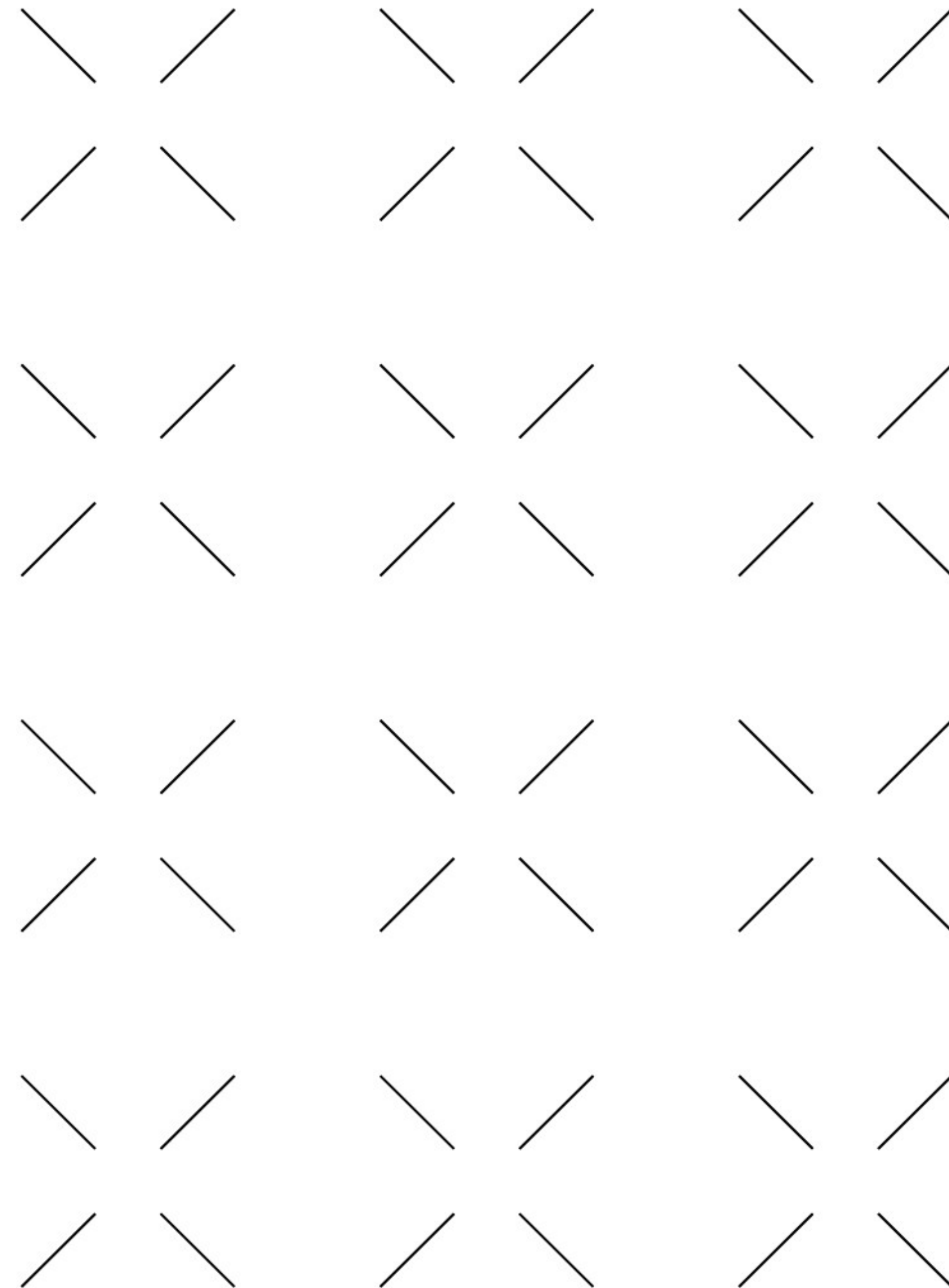
(A) Los datos se leen en una memoria intermedia desde la que se procesan posteriormente.

(A) Puedes avanzar y retroceder en la memoria intermedia según necesites. Esto te da un poco más de flexibilidad durante el procesamiento.

(D) También hay que comprobar si la memoria intermedia contiene todos los datos necesarios para procesarla completamente.

(D) Tienes que asegurarte de que cuando lees más datos en la memoria intermedia, no sobrescribes los datos de la memoria intermedia que aún no has procesado.

6.2 Constructores



¿Cómo crear un objeto de archivo?



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**:

```
public File(String pathname);  
public File(String parent, String child);  
public File(File parent, String child);  
public File(URI uri);
```

Dependiendo de la familia de sistemas operativos para la que estemos programando, deberemos utilizar una nomenclatura u otra para las rutas de los ficheros:

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

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

- **Ventanas**:

```
Archivo fiWindows = nuevo Archivo("C:³Usuarios³Administrador³DocumentosyArchivos³.txt");
```


¿Cómo crear un objeto de archivo?



1) La **primera sobrecarga** recibe una cadena (nombre de ruta abstracto): ex. `Archivo("archivo.txt")`

```
File file = new File("file.txt");
```

2) La **segunda sobrecarga** recibe dos cadenas: ex. `Archivo("/home/ada", "archivo.txt")`

1º = ruta al padre (relativa o absoluta).

2nd = ruta relativa al padre en el primer parámetro.

3) La **tercera sobrecarga** es una variante de la primera: ex. `File(new File("/home/ada"), "archivo.txt")`

4) La **cuarta sobrecarga** acepta un URI (Uniform Resource Identifier) con esta sintaxis: ej. `file:/ada/file.txt`

```
scheme: [//[user:password@]host[:port]][/]path[?query] [#fragment]
```

Ejemplo sencillo

Una vez que tenemos el archivo o carpeta, podemos comprobar sus propiedades, tales como **exists** o **getAbsolutePath**:

```
String stCurDir = new File("").getAbsolutePath();  
  
    System.out.println("Carpeta actual: "+ stCurDir);  
  
    File fiFile1 = new File("ejemplo1.txt");  
  
    if(fiArchivo1.existe()) {  
        // hacer algo  
    }
```

Ejemplo completo

Lee un fichero para acceder a su información. A continuación, comprueba si el archivo existe.

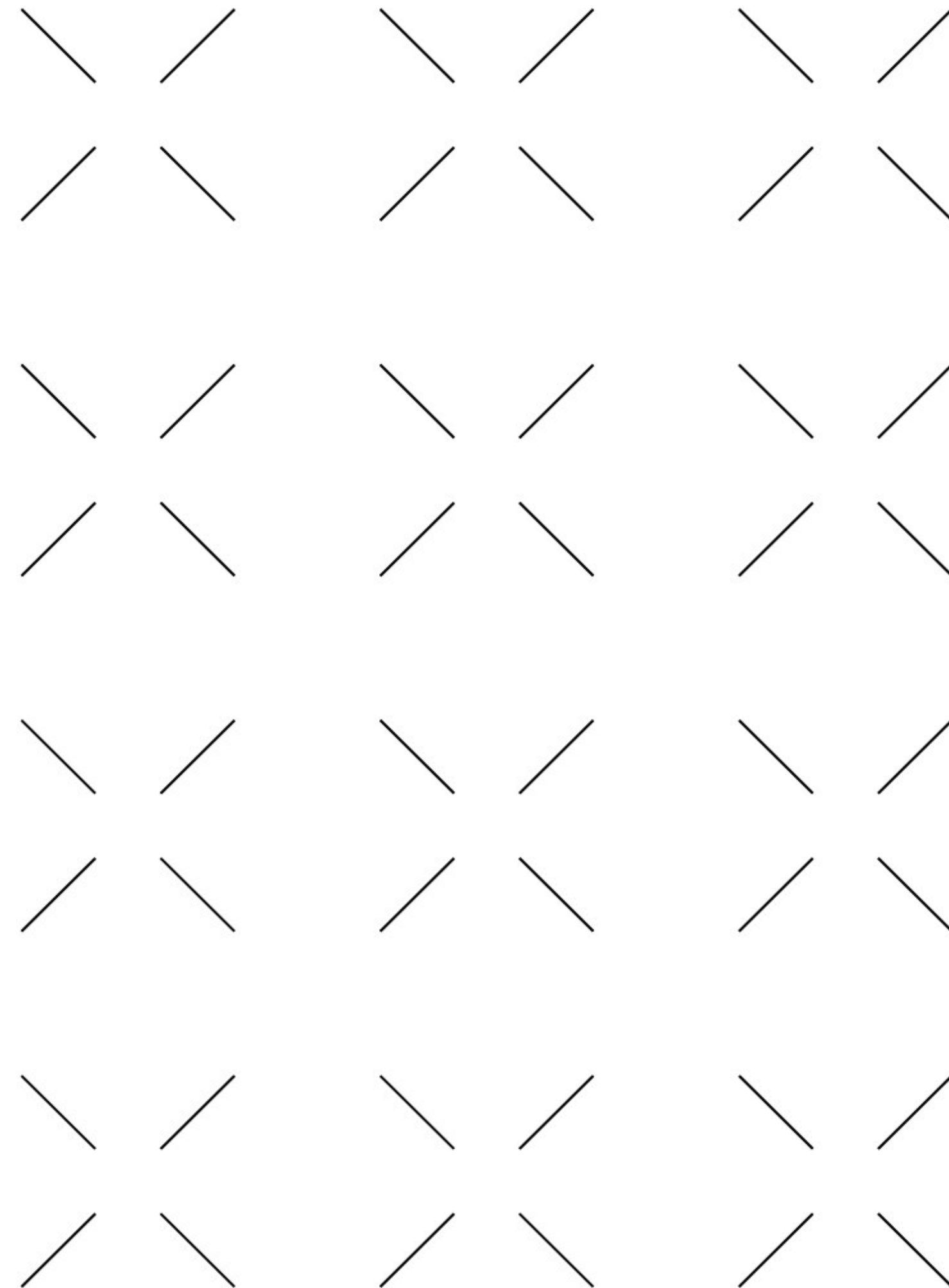
Pruébalo colocando un archivo llamado "ejemplo1.txt" en la carpeta raíz de un nuevo proyecto Java.

```
// DAM.ADA U1 EJEMPLO 1: COMPROBACIÓN DE LA EXISTENCIA DE UN FICHERO
import java.io.*;

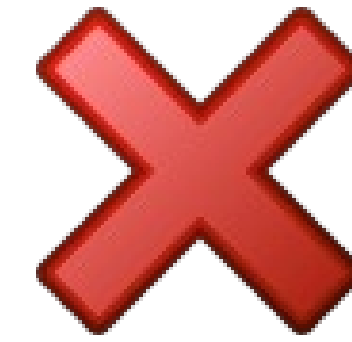
public class ADA_Ficheros
{
    public static void main(String[] stArgs)
    {
        String stCurDir = new File("").getAbsolutePath();
        System.out.println("Carpeta actual: "+ stCurDir);

        Fichero fiFichero1 = nuevo Fichero("ejemplo1.txt");
        if(fiArchivo1.existe()) {
            System.out.println("Archivo encontrado");
        } else {
            System.out.println("Archivo no encontrado");
        }
    }
}
```

6.3 Excepciones



Errores en las operaciones con archivos



Antes de que podamos empezar a leer y escribir archivos, tendremos que hablar sobre **el manejo de estados de error**, que se producirán muchas veces al trabajar con archivos.

A menudo se producen errores en nuestros programas, principalmente errores causados por operaciones de entrada/salida, a menudo denominadas E/S.

En todos estos casos, hay un usuario que puede introducir datos no válidos o archivos no existentes o no válidos, o un archivo se ha movido o borrado inesperadamente.

Sin embargo, no dejaremos que bloqueen nuestros programas debido a errores. En su lugar, informaremos al usuario de la situación.

Excepciones

Usamos excepciones cuando la operación es compleja y sería demasiado difícil sanear todos los posibles estados de error. Las excepciones se **denominan gestión pasiva de errores**.

No tenemos que lidiar con la lógica interna del método que llamamos de ninguna manera. Todo lo que tenemos que hacer es intentar ejecutar la parte vulnerable del código en "modo protegido".

```
try
{
    System.out.println(Mathematics.divide(a, b));
}
catch (Exception e)
{
    System.out.println("Error occurred.");
}
```



Para más información: <https://www.ictdemy.com/java/files/exceptions-in-java>

Ejemplo

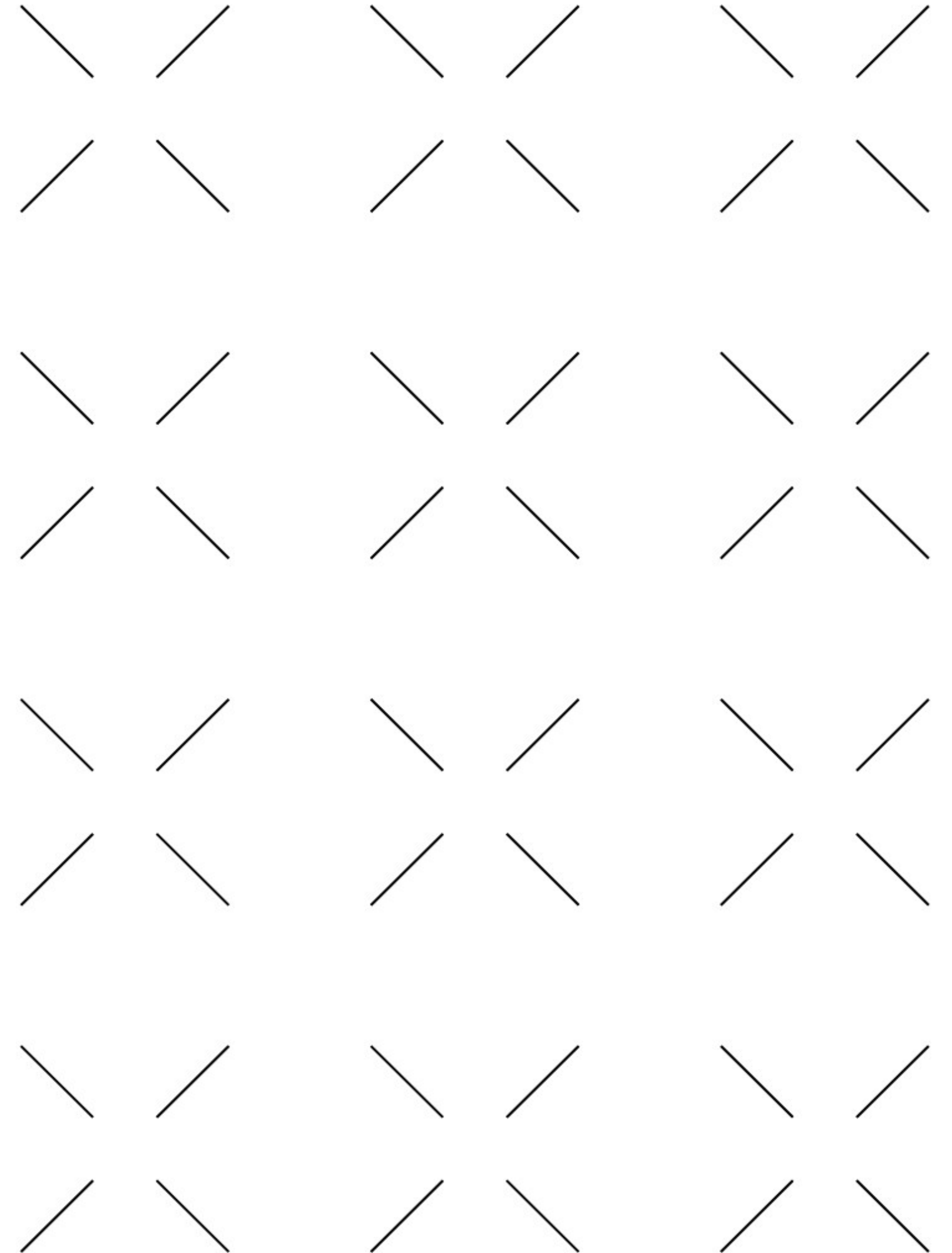
Cuando tratamos con ficheros, como veremos más adelante, necesitamos atrapar la excepción y cerrar el fichero tanto si la operación va bien como si va mal.

Usando **finally**, Java recuerda que el bloque try-catch contenía finally y llama al bloque finally incluso después de salir del bloque try o catch.

Tenga en cuenta que *openFile*, *writeToFile* sólo se colocan para que sea más fácil de explicar.

```
public boolean saveSettings()
{
    try
    {
        openFile();
        writeToFile();
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
    finally
    {
        if (fileIsOpened())
            closeFile();
    }
}
```

6.4 Lectura de archivos de texto en Java



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.

Lectura de archivos de texto en Java



Existen varios métodos para leer archivos como puedes comprobar aquí:

- <https://www.baeldung.com/reading-file-in-java>
- <https://www.geeksforgeeks.org/different-ways-reading-text-file-java/>
- <https://www.stackchief.com/blog/FileReader%20vs%20BufferedReader%20vs%20Scanner%20%7C%20Java>

Le mostraremos la más fácil, pero debería comprobarlas todas.

Para más información:

- Ictdemy. Lección 3 - Trabajar con archivos de texto en Java
<https://www.ictdemy.com/java/files/working-with-text-files-in-java>

Lectura de archivos de texto con `BufferedReader`



Este método realiza el almacenamiento en búfer para una lectura eficiente de caracteres, matrices y líneas.

Se puede especificar el tamaño del búfer (en bytes) o utilizar el tamaño por defecto. Por defecto es lo suficientemente grande para la mayoría de los propósitos.

Por lo tanto, es aconsejable envolver un `BufferedReader` alrededor de cualquier `Reader` cuyas operaciones `read()` puedan ser costosas, como `FileReaders` e `InputStreamReaders`. Por ejemplo:

```
BufferedReader in = new BufferedReader(Reader in, int size);
```

Ejemplo de uso de BufferedReader



```
// DAM.ADA U1 EJEMPLO 2: LECTURA DE ARCHIVOS DE TEXTO
import java.io.*;
clase pública ReadFromFile
{
    public static void main(String[] stArgs)throws Exception
    {
        Archivo fiFile = nuevo Archivo("ejemplo2.txt");
        BufferedReader brBuffer = new BufferedReader(new FileReader(fiFile));

        Cadena stLine;
        while ((stLine = brBuffer.readLine()) != null)
            System.out.println(stLine);
        brBuffer.close();
    }
}
```

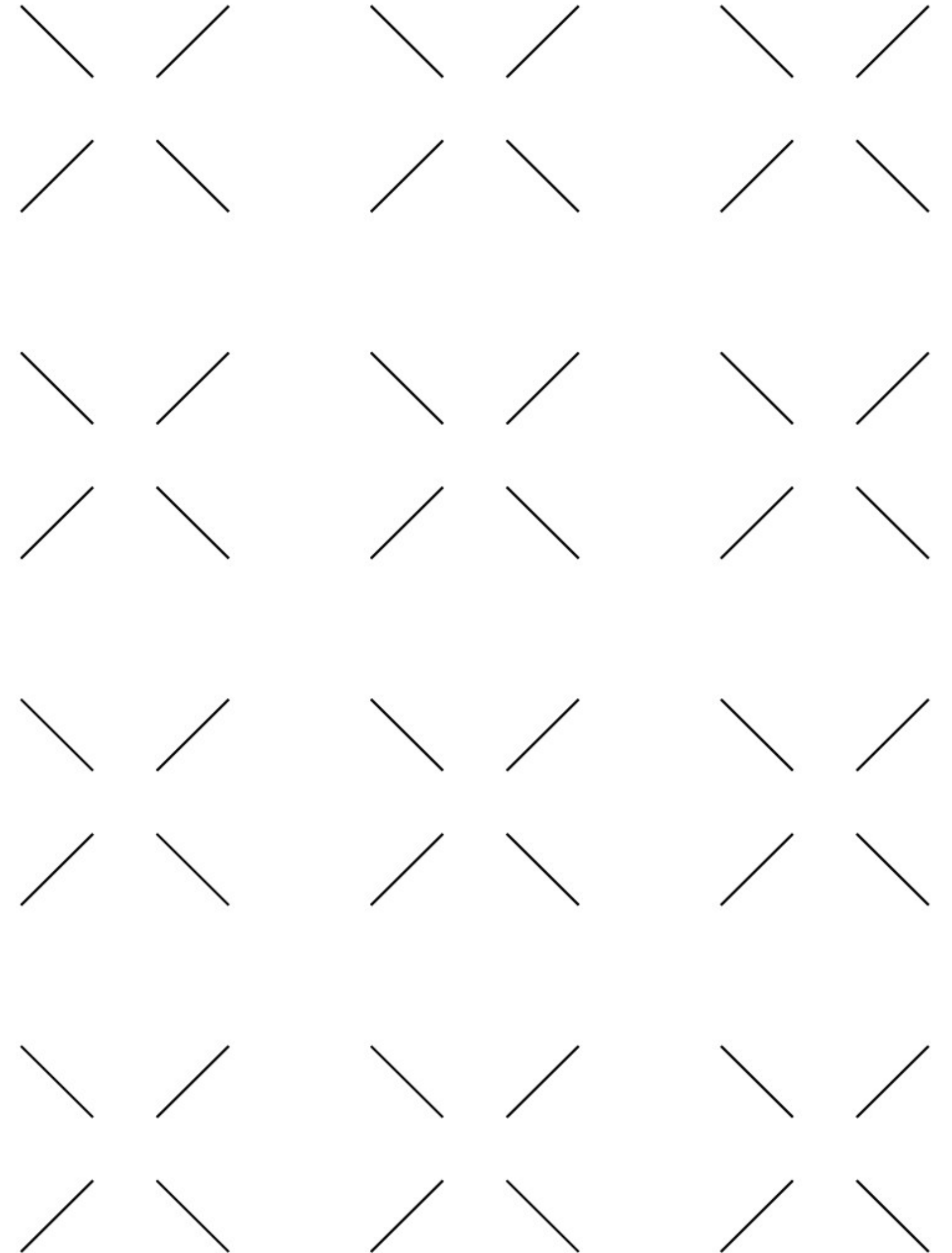
Ejemplo de uso de BufferedReader (CON EXCEPCIONES)



```
// DAM.ADA U1 EJEMPLO 2: LECTURA DE ARCHIVOS DE TEXTO
import java.io.*;
clase pública ReadFromFile
{
    public static void main(String[] stArgs)throws Exception
    {
        pruebe
        {
            Archivo fiFile = nuevo Archivo("ejemplo2.txt");
            BufferedReader brBuffer = new BufferedReader(new FileReader(fiFile));

            Cadena stLine;
            while ((stLine = brBuffer.readLine()) != null)
                System.out.println(stLine);
            brBuffer.close();
        }
        catch (FileNotFoundException fne) { System.out.println("Archivo no encontrado"); }
        catch (IOException ioe) { System.out.println("Error de E/S"); }
    }
}
```


6.5 Escritura de archivos de texto en Java



Escritura de archivos de texto en Java

Lo mismo ocurre con la escritura. Consulta aquí varios métodos para escribir archivos:

- <https://www.geeksforgeeks.org/java-program-to-write-into-a-file/>

Le mostraremos la más fácil, pero debería comprobarlas todas.

Para más información:

- Ictdemy. Lección 3 - Trabajar con archivos de texto en Java
<https://www.ictdemy.com/java/files/working-with-text-files-in-java>



Escritura de archivos de texto con BufferedWriter



Se utiliza para escribir texto en un flujo de salida de caracteres. Tiene un tamaño de búfer por defecto, pero se puede asignar el tamaño de búfer grande.

Es útil para escribir caracteres, cadenas y matrices. Es mejor envolver esta clase con cualquier clase escritora para escribir datos en un archivo si no se requiere una salida rápida.

Por ejemplo:

```
BufferedWriter out = new BufferedWriter(Writer out);
```

Ejemplo de uso de BufferedWriter



```
// DAM.ADA U1 EJEMPLO 3: ESCRITURA DE ARCHIVOS DE TEXTO
import java.io.*;
public class EscribirDesdeArchivo
{
    public static void main(String[] stArgs) throws Exception
    {
        String stText = "Ejemplo de ADA para escribir archivos de texto"; //texto para
        el archivo

        BufferedWriter bwBuffer = new BufferedWriter(new FileWriter("ejemplo3.txt"));
        bwBuffer.write(stText);

        System.out.print(stText);
        System.out.print("El archivo se ha creado correctamente con el contenido");

        bwBuffer.close();
    }
}
```

Ejemplo de uso de BufferedWriter (CON EXCEPCIONES)



```
// DAM.ADA U1 EJEMPLO 3: ESCRITURA DE ARCHIVOS DE TEXTO
import java.io.*;
public class EscribirDesdeArchivo
{
    public static void main(String[] stArgs)throws Exception
    {
        intentar {
            String stText = "Ejemplo de ADA para escribir archivos de texto";

            BufferedWriter bwBuffer = new BufferedWriter(new FileWriter("ejemplo3.txt"));
            bwBuffer.write(stText);

            System.out.print(stText);
            System.out.print("El archivo se ha creado correctamente con el contenido");

            bwBuffer.close();
        }
        catch (FileNotFoundException fne) { System.out.println("Archivo no encontrado"); }
        catch (IOException ioe) { System.out.println("Error de E/S"); }
    }
}
```


7. ACTIVIDADES PROPUESTAS

Actividades propuestas



Consulta los ejercicios propuestos que encontrarás en el "Aula Virtual". Estas actividades son **opcionales y no evaluables, pero la** comprensión de estas actividades no evaluables es esencial para resolver la tarea evaluable que tienes por delante.

"La mejor forma de aprender cualquier cosa es mediante la práctica y las preguntas de ejercicios. Aquí tienes la oportunidad de practicar los conceptos del lenguaje de programación Java resolviendo los ejercicios desde los más básicos hasta los más complejos". W3CSchools

En breve encontrará las soluciones propuestas.

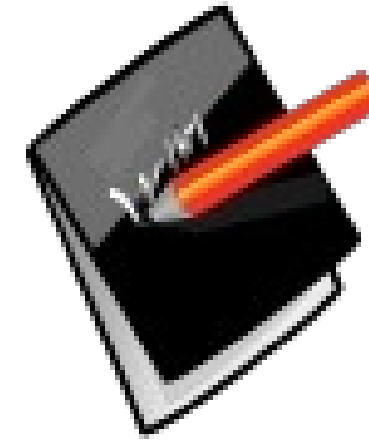
¿Y ahora qué?



Esta semana deberías...

- 1) Instala tu IDE favorito y Java en tu Sistema Operativo favorito (revisando la documentación proporcionada).
- 2) Intenta hacer los ejercicios de repaso (en función de tus conocimientos).
- 3) Intenta hacer los ejercicios propuestos.
- 4) Revise los materiales para la próxima semana ANTES de asistir a la siguiente CT.

8. BIBLIOGRAFÍA



Recursos

- Documentación de Oracle. <https://docs.oracle.com/javase/tutorial/essential/io/index.html>
- Geeksforgeeks. <https://www.geeksforgeeks.org/java-program-to-write-into-a-file/>
- Geeksforgeeks. <https://www.geeksforgeeks.org/different-ways-reading-text-file-java/>
- Java absoluto. Capítulo 10 E/S de archivos. Walter Savitch. Pearson.
- Josep Cañellas Bornas, Isidre Guixà Miranda. Acceso a datos. Desarrollo de aplicaciones multiplataforma. Creative Commons. Departamento de Enseñanza, Instituto Obert de Catalunya. Dipòsit legal: B. 29430-2013. <https://ioc.xtec.cat/educacio/recursos>
- Alberto Oliva Molina. Acceso a datos. UD 1. Manejo de ficheros. IES Tubalcaín. Tarazona (Zaragoza, España).

