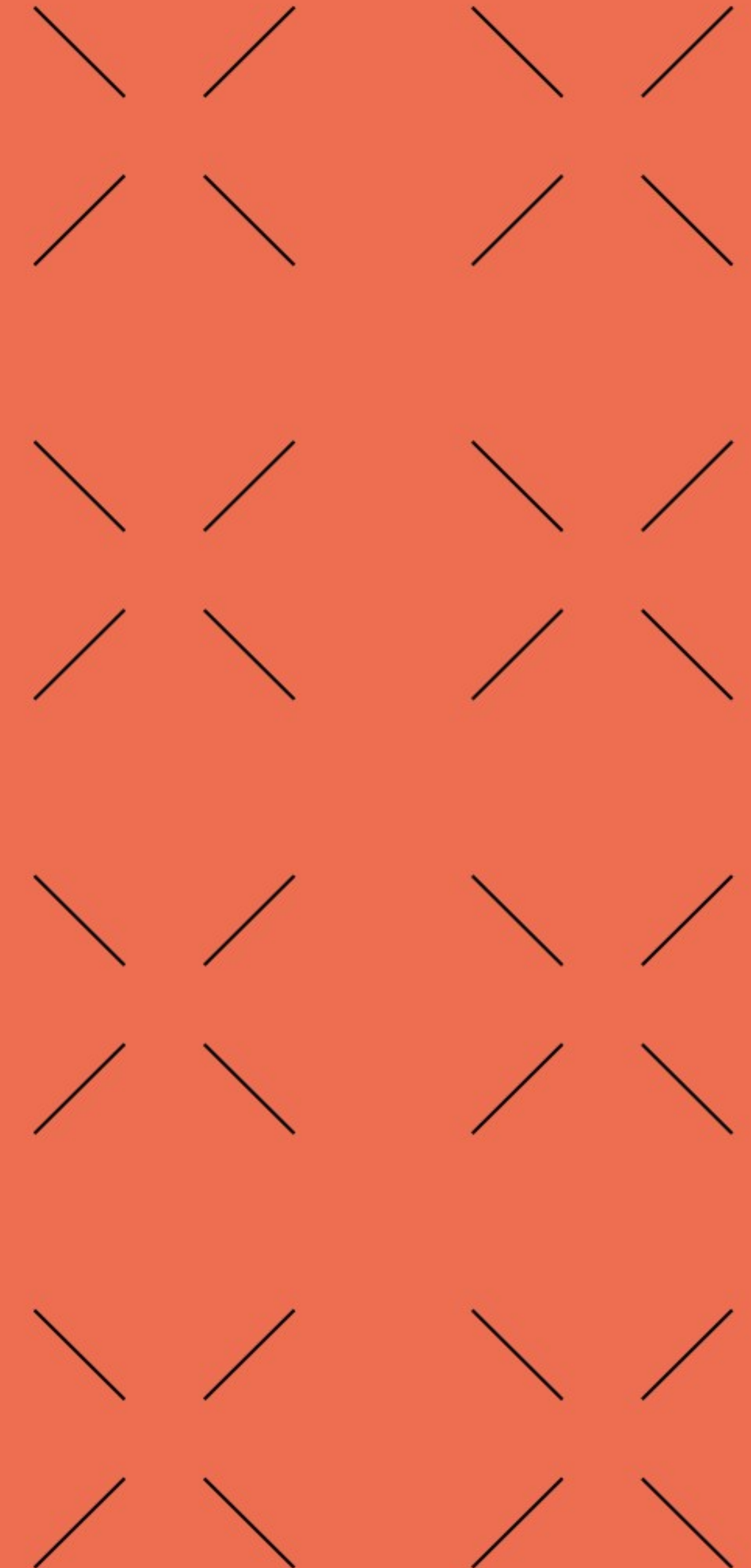# Unit 1. ACCESS TO FILES

Part 1. Intro, Java review and basic file access

Acceso a Datos (ADA) (a distancia en inglés)
CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)

Abelardo Martínez
Year 2023-2024

Credits

- Notes made by Abelardo Martínez.
- Based and modified from Sergio Badal (www.sergiobadal.com).
- The images and icons used are protected by the LGPL licence and have been obtained from:
  - https://commons.wikimedia.org/wiki/Crystal_Clear
 - https://www.openclipart.org

Contents

1. WHY USING JAVA?

Java language

Java is a **general-purpose, class-based, object-oriented programming LANGUAGE** designed for having lesser implementation dependencies. It is also a computing PLATFORM for application development.

Advantages:

1) Java is easy to write and easy to run—this is the foundational strength of Java and why many developers program in it—. When you write Java once, you can run it almost anywhere at any time.

2) Java can be used to create complete applications that can run on a single computer or be distributed across servers and clients in a network.

3) As a result, you can use it to easily build mobile applications or run on desktop applications that use different operating systems and servers, such as Linux or Windows.
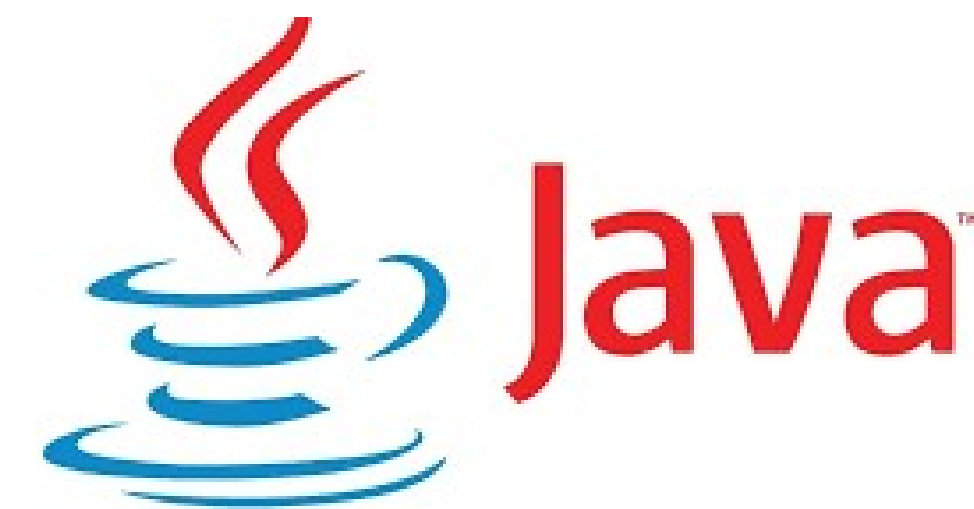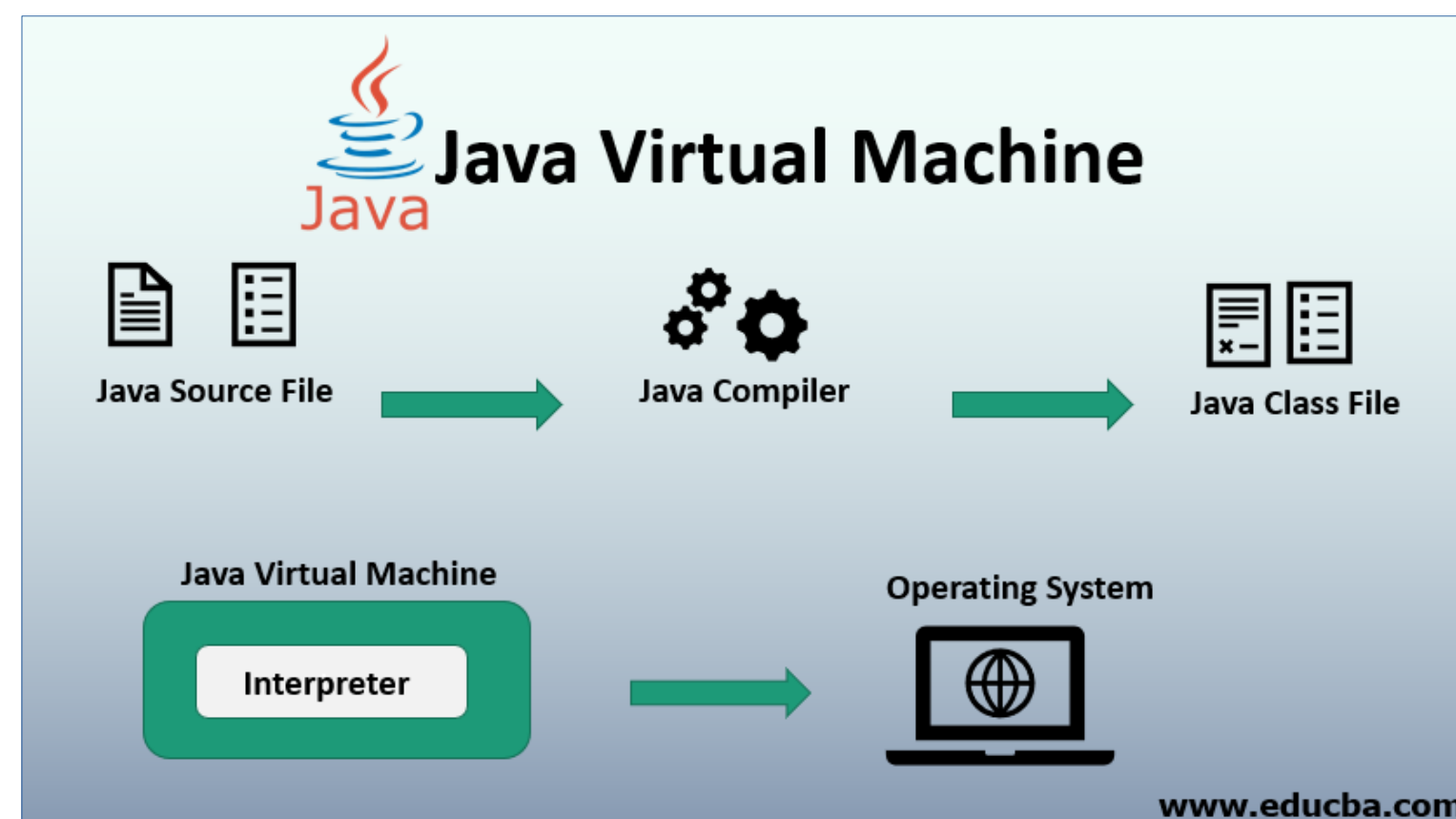
Python language

- Java is no longer the king of OO languages.
- For the first time since the TIOBE index began almost 20 years ago, Java and C do not get the first two positions.
- Python, a language born the same year as Java, is coming to stay.

Advantages:

- It takes less time to write in a text editor, with the lines of code actually being three-to-five times shorter than Java.
- In Java you could need a commercial license to maintain the regular security updates of the language.

Disadvantages:

- Python runs slower than Java when compiled.
- Java is better equipped for mobile development.

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	v	Java	10.33%	-2.14%
5	5		C#	7.04%	+1.64%
6	8	^	JavaScript	3.29%	+0.89%
7	6	v	Visual Basic	2.63%	-2.26%
8	9	^	SQL	1.53%	-0.14%
9	7	v	Assembly language	1.34%	-1.41%
10	10		PHP	1.27%	-0.09%

[Current TIOBE Index](#)

For further information:
[Java vs Phyton](#)

2. RECOMMENDED RESOURCES

What's your situation?

Level	Description
0	I have no idea of programming. Try Alex basic 2 courses and continue to next level if you need it.
1	My last line of code was ages ago. Don't worry, try the 14 min Alex video and decide next step.
2	I'm good at coding but that's my first time with OO. Try with Alex courses 2 and 3 and let me know.
3	I can consider myself an expert on OO, but no line written on Java . It's OK. Once you ride a bike, all bikes are almost the same.
4	I can consider myself an expert on Java programming. Good news, relax and see you next week!

Java resources

Recommended resources

- Java Basics (Alex Lee) Courses 1st and 2nd: Link 1
- Refresh your mind in just 14 minutes. (Alex Lee): Link 2
- Advanced Java (Alex Lee) Courses 3rd and 4th: Link 3

Additional resources (in Spanish)

- A book from a PRG DAM/DAW teacher (only on "paper") (200 solved activities) (15€): Link 1
- An online book (only on PDF) (350 solved activities) (5€): Link 2
- An online course (only on YouTube) (free): Link 3

3. INSTALLING JAVA & IDE

IDE's for building Java applications:

IDE	Description
Eclipse	Is one of the most widely used Integrated Development Environment (IDE) for building Java applications. It may also be used to develop applications in other programming languages via plug-ins. https://www.eclipse.org How to install
NetBeans	Is an integrated development environment (IDE) for Java and runs on Windows, macOS, Linux and Solaris. It has extensions for other languages. 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/
Visual Studio Code	Is a source-code editor made by Microsoft for Windows, Linux and macOS and it can be used with a variety of programming languages. To install the program you must download the DEB package (Linux) or the EXE package (Windows) https://code.visualstudio.com
IntelliJ	IntelliJ IDEA is an integrated development environment (IDE) written in Java for developing computer software written in Java, Kotlin, Groovy, and other JVM-based languages. It is developed by JetBrains (formerly known as IntelliJ) and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. Both can be used for commercial development. https://www.jetbrains.com/es-es/idea/download/other.html

IDE Eclipse

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

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

Additional help in Spanish:

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

Enable graphical interface in Eclipse:

https://www.cablenaranja.com/java-como-activar-el-editor-visual-en-eclipse/
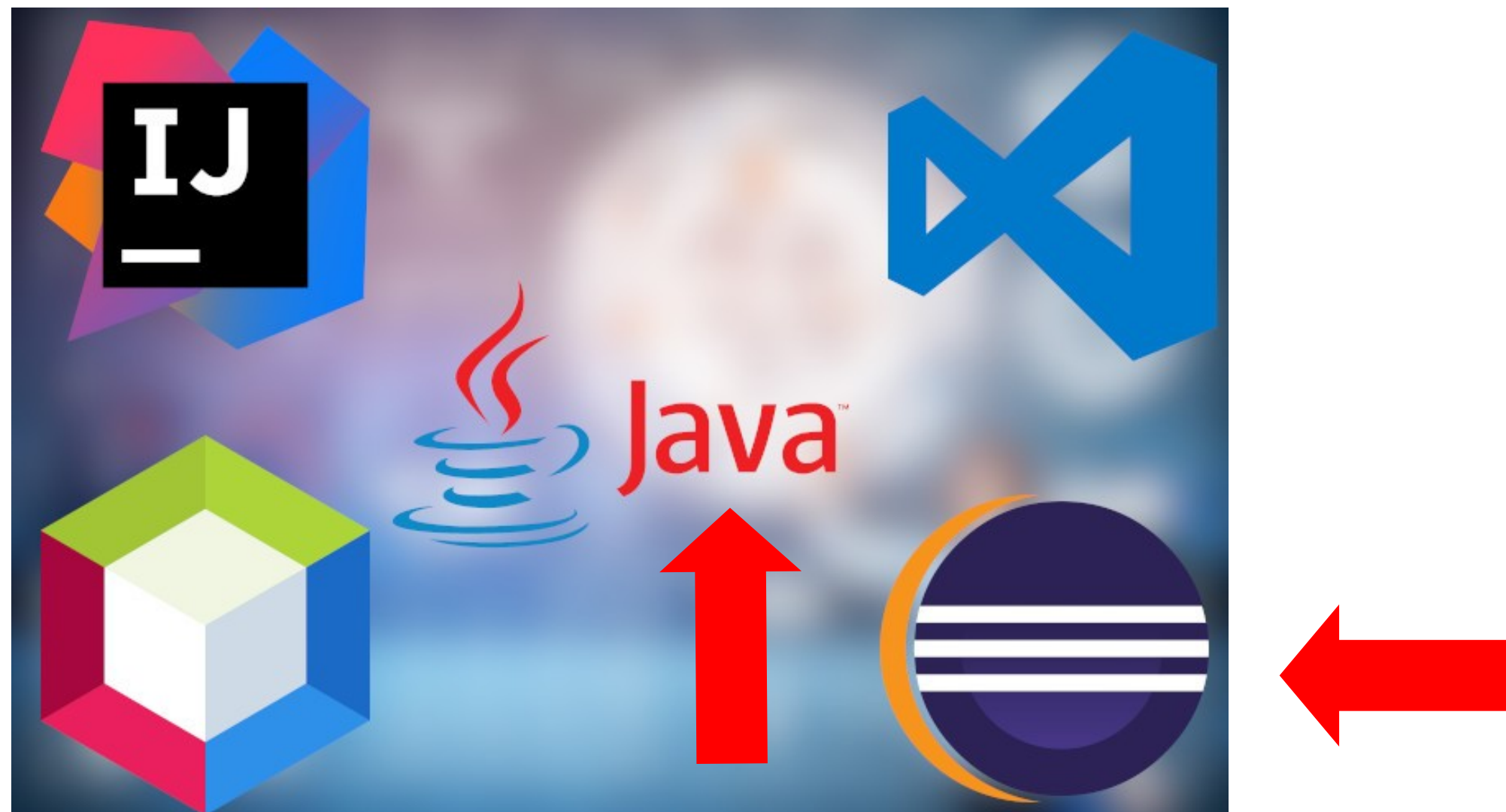
4. BASICS OF JAVA

History of Java

- **The complete History of Java Programming Language:**
  https://www.geeksforgeeks.org/the-complete-history-of-java-programming-language/
- **A Short History of Java:** https://dzone.com/articles/a-short-history-of-java
- **Java (programming language):**
  https://en.wikipedia.org/wiki/Java_(programming_language)#:~:text=Java%20was%20originally%20developed%20by,by%20Sun%20under%20proprietary%20licenses.
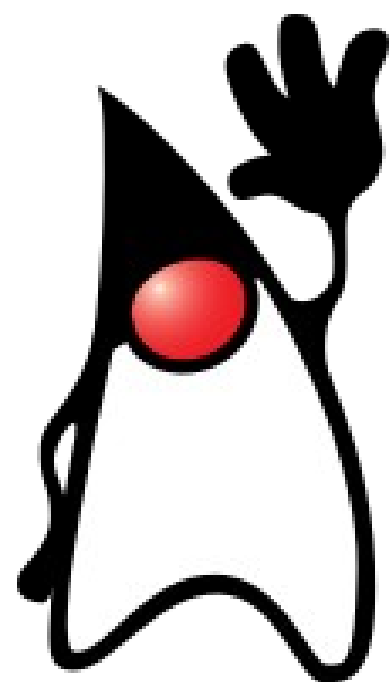
Basic code structure

Basic code structure

text file named `HelloWorld.java`

name

`main()` method

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

statements

body

Complete infographic: https://www.infogrades.com/technology-infographics/java-cheat-sheet-for-programmers/

Input and output. Data types

Input and output

use any editor to
create your program

type `javac HelloWorld.java`
to compile your program

type `java HelloWorld`
to compile your program

editor → `HelloWorld.java` → *compiler* → `HelloWorld.class` → *JVM* → `"Hello, World"`

your program
(a text file)

computer-language
version of your program

output
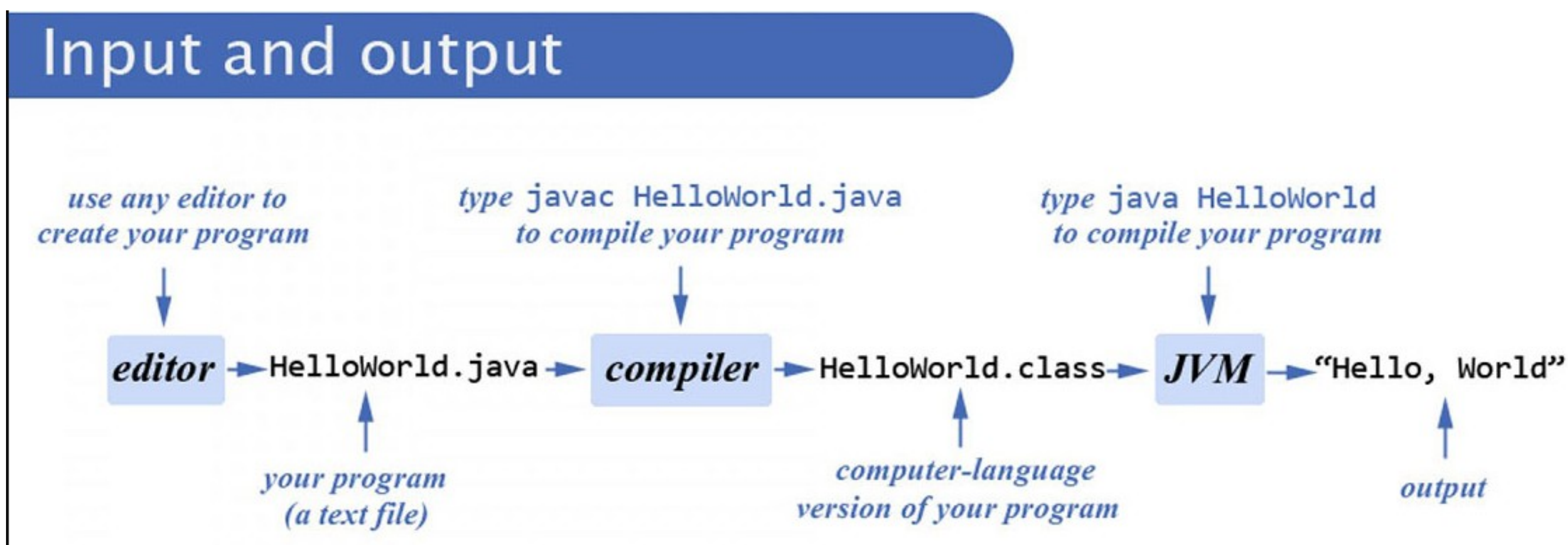
Date types

type	set of values	common operators	sample literal values
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"

Complete infographic:

Assignment status. Booleans. Comparison

Assignment status

declaration statement

```
int a, b;
a = 1234
b = 99;
int c = a + b;
```

variable name
literal
assignment statement
combined declaration and assignment statement
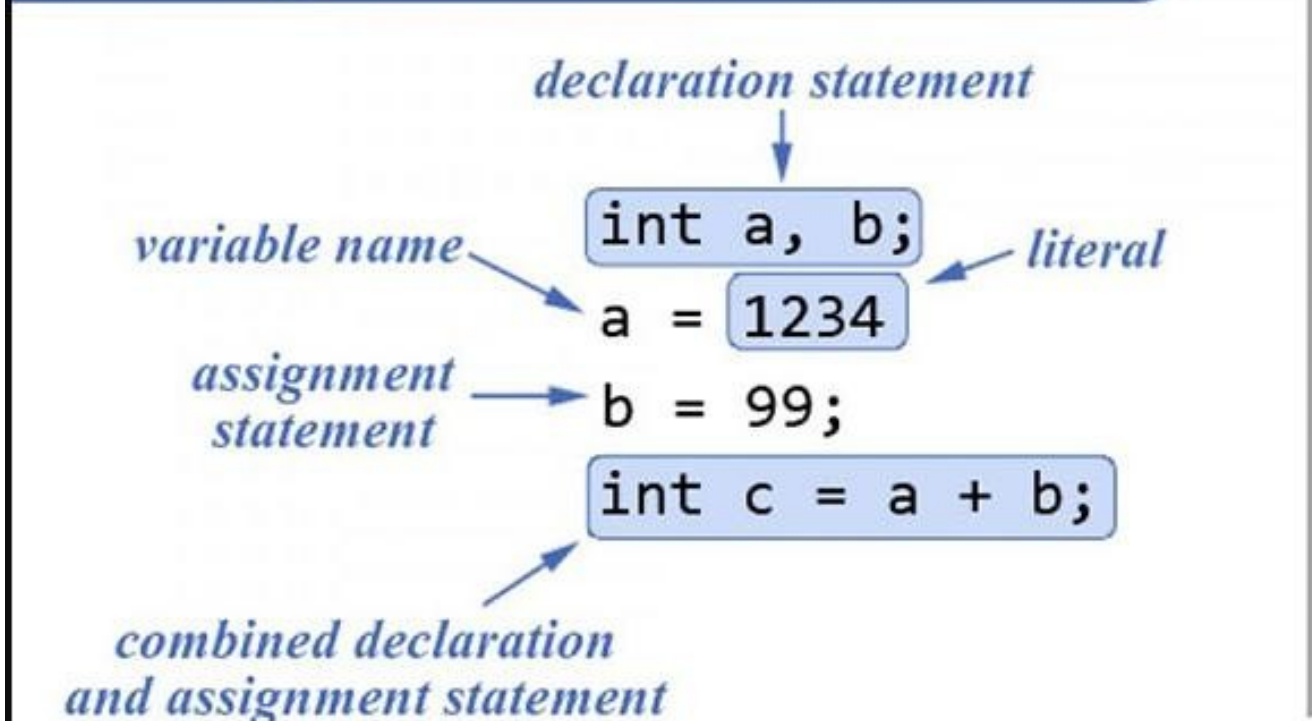
Booleans

values	true or false
literals	true false
operations	and or not
operators	&& || !

Comparison

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

Complete infographic: https://www.infogrades.com/technology-infographics/java-cheat-sheet-for-programmers/
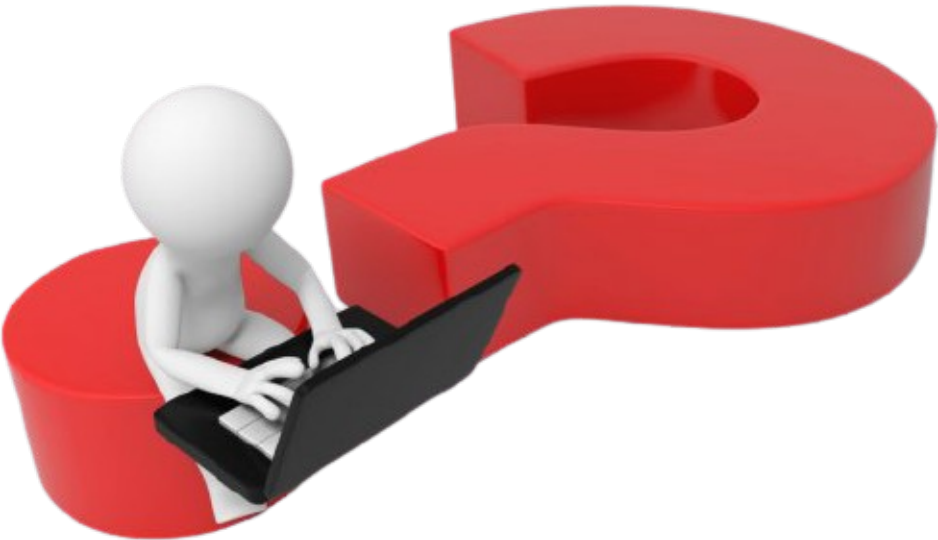
Resources and exercises to review

- The Java™ Tutorials. https://docs.oracle.com/javase/tutorial/index.html
- W3resource. Java Programming Exercises, Practice, Solution.
  https://www.w3resource.com/java-exercises/
- Java tutorial. https://www.w3schools.com/java/default.asp
- CODE EXERCISES. Java Programming Exercises. https://code-exercises.com
- Practice 133 exercises in Java. https://exercism.org/tracks/java/exercises

Java Level	Recommended exercises of each section depending on how long you coded last time		
	> 1 year	> 3 months	< 3 months
Basic	5	4	3
Medium	4	3	2
Advanced	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. FILES: TYPES AND ACCESS

Type of Files
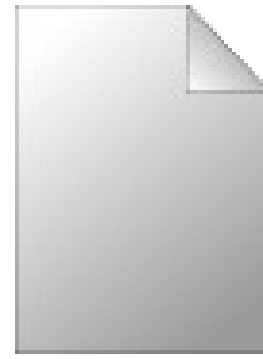
Data, i.e. objects stored in memory, is lost once the application is terminated. If we want to ensure that the application data is **persistent**, we'd have to save it when the application is closing and load it back when we run it again. For this purpose we use files.

Files are structured data warehouses and it could be considered as an exchanging data resources between 2 systems: one volatile (RAM memory) and another permanent (storage devices).

There are a lot of ways to store application data, each has some advantages and disadvantages. Generally, **we store application data in one of the following ways**:

- **Text files with a flat structure** (e.g. .txt and .csv files) (NOW!)
- Text files with internal hierarchy (json, XML, HTML, etc.) (coming soon)
- Binary files (simple memory dump into a file) (not at this module)
- Databases (Unit2)

Text Files vs Binary Files

==**Text files**==. They are designed to be read by human beings and can be read or written with an editor. Text files are often also called flat files or [ASCII](#) files.

- **Positive**: they are usually the same on all computers, so that they can move from one computer to another.
- **Negative**: they are not as efficient to process than the binary ones.

==**Binary files**==. They are designed to be read by programs and consist of a sequence of binary digits.

- **Positive**: they are more efficient to process than text files. Unlike most binary files, Java binary files have the advantage of being platform independent.
- **Negative**: they are designed to be read on the same type of computer and with the same language as the computer that created the file. It is not possible to view the content directly.

Text Files vs Binary Files

Here you can check an extensive study on both types of files with 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
BufferedFileReader.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

Type of Access

Although a **sequential** access mechanism traverses file record in a linear fashion, **random** access in a file enables us to access individual records directly without searching through other records.

Flat files in general are not meant to be accessed in this manner; moreover, Java does not impose any structure on a file. As a result, there is no one specific technique to create random access files.

For more information on random access you can check this link:

https://www.developer.com/database/random-file-access-using-java/
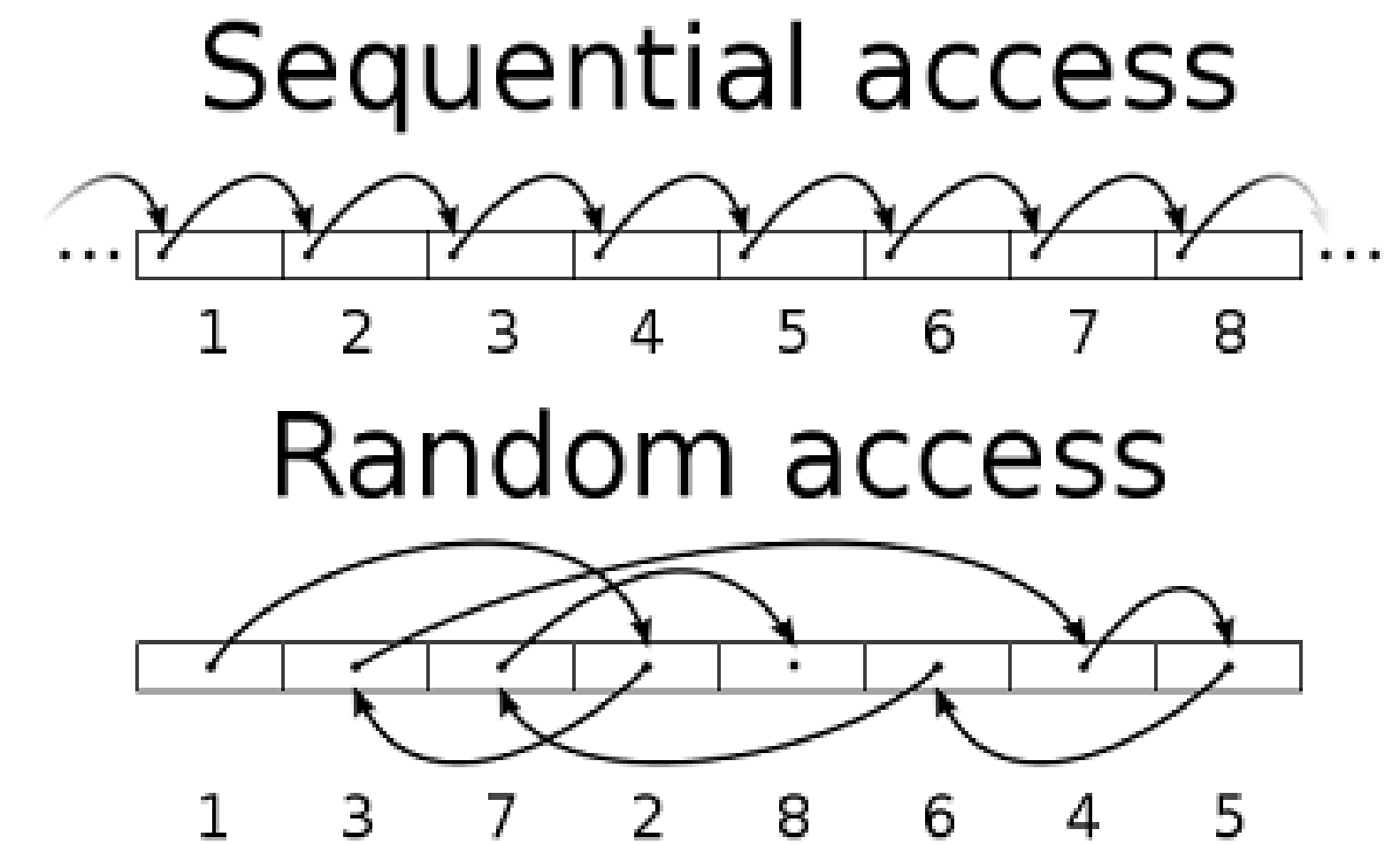
Input/Output Streams

A **stream** is an object that enables the flow of data between a program and some I/O device or file:

- If the data flows into a program, then the stream is called an **input stream**
- If the data flows out of a program, then the stream is called an **output stream**

Input streams can flow from the keyboard, from a file, etc.

- **System.in** is an **input stream** that connects to the **keyboard**

```java
Scanner keyboard = new Scanner(System.in);
```

Output streams can flow to a screen, to a file, etc.

- **System.out** is an **output stream** that connects to the **screen/console**

```java
System.out.println("Output stream");
```

Input/Output Streams

"Character" Streams
(Reader/Writer)

"Byte" Streams
(InputStream/
OutputStream)

Java Program

char
(16-bit)

Byte
(8-bit)

Input Stream

Output Stream

Input Source
(keyboard, file,
network, program)

Output Sink
(console, file,
network, program)
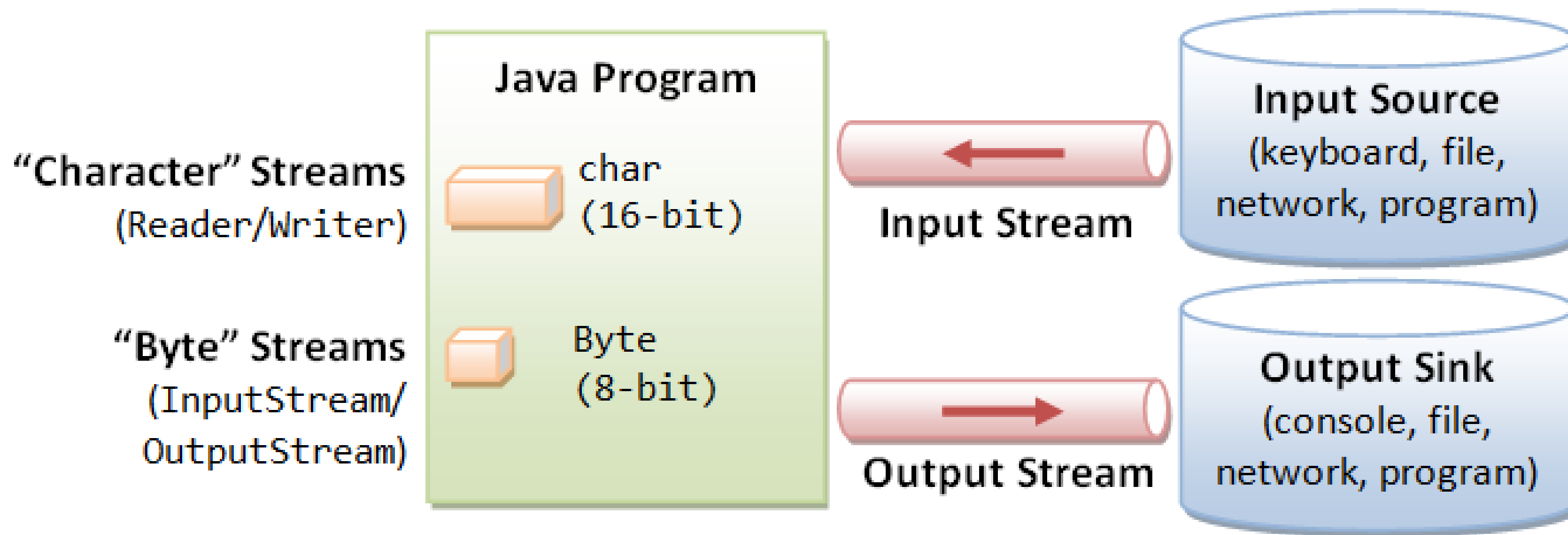
Internal Data Formats:
- Text (char): UCS-2
- int, float, double, etc.

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)

Input/Output Streams (using Buffers)
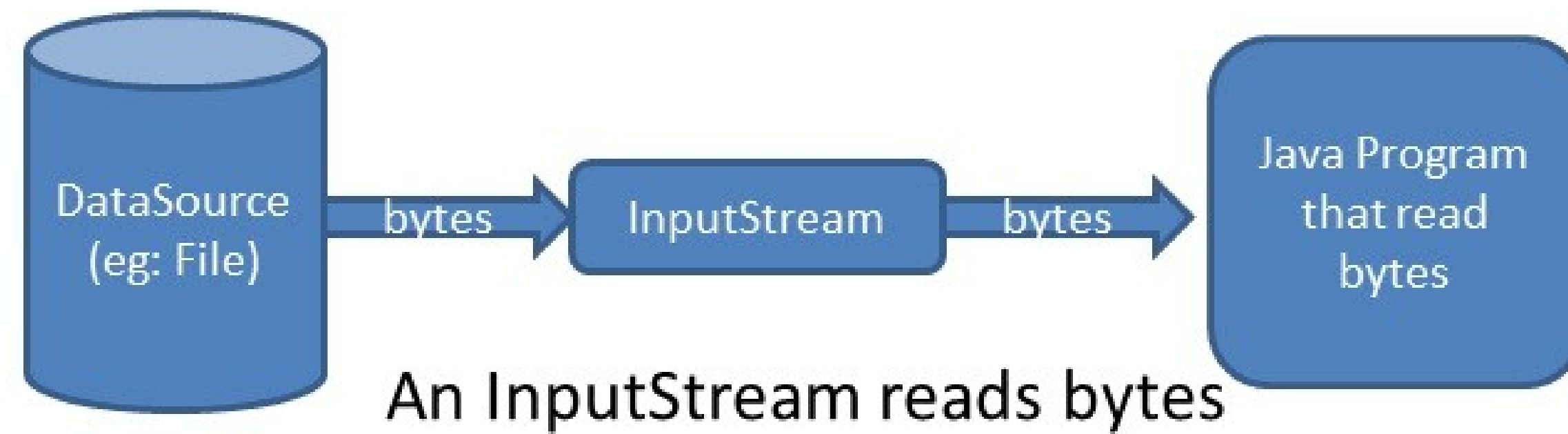
Example using buffers to read binary data

An InputStream reads bytes

A Reader works with an InputStream and reads characters

A BufferedReader pre-fetches characters in to a buffer for performance
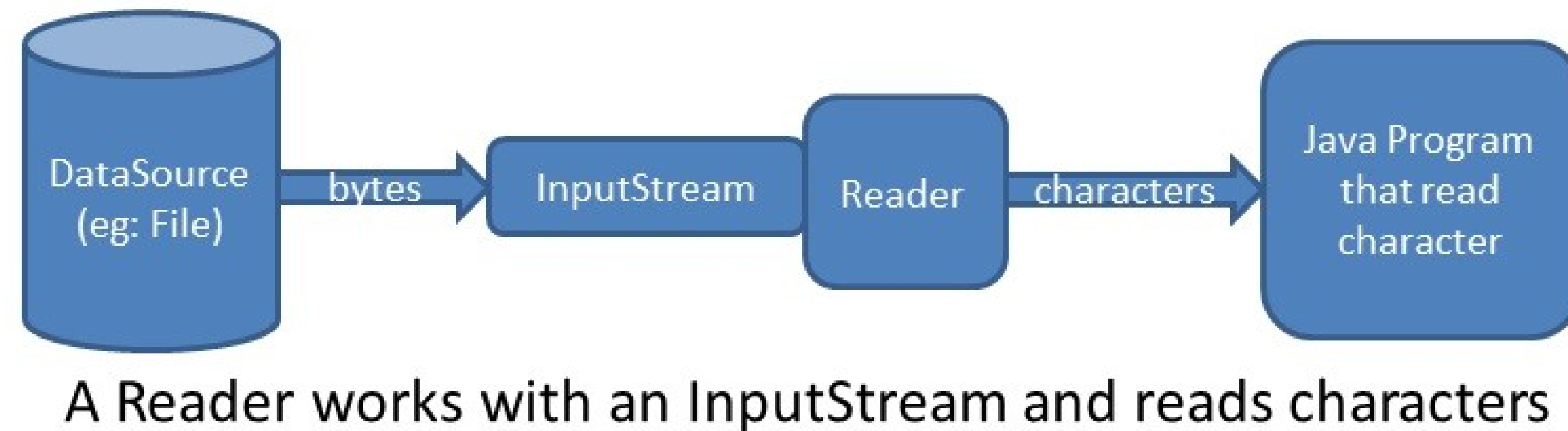
6. FILE ACCESS WITH JAVA

6.1 File Class
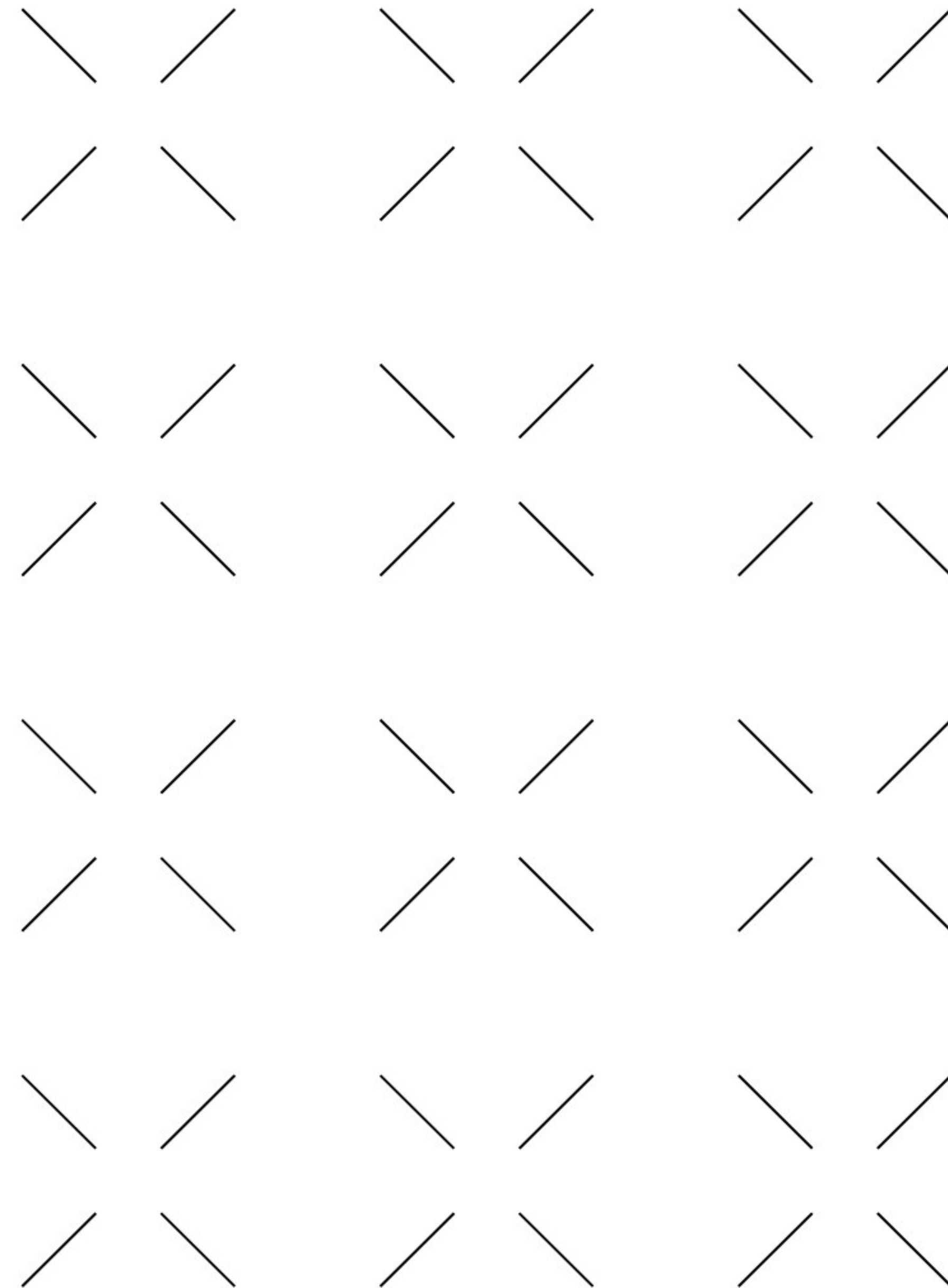
Files & Folders

In this lesson we'll introduce the **File class**, its methods and uses.

- The File class has been in Java since the first release.
- It encapsulates virtually all the functionality for managing a file system organised in directory trees.
- This class doesn't represents the content of any file, but rather the path where it is located. Therefore, the class **represents either a file or folder**.
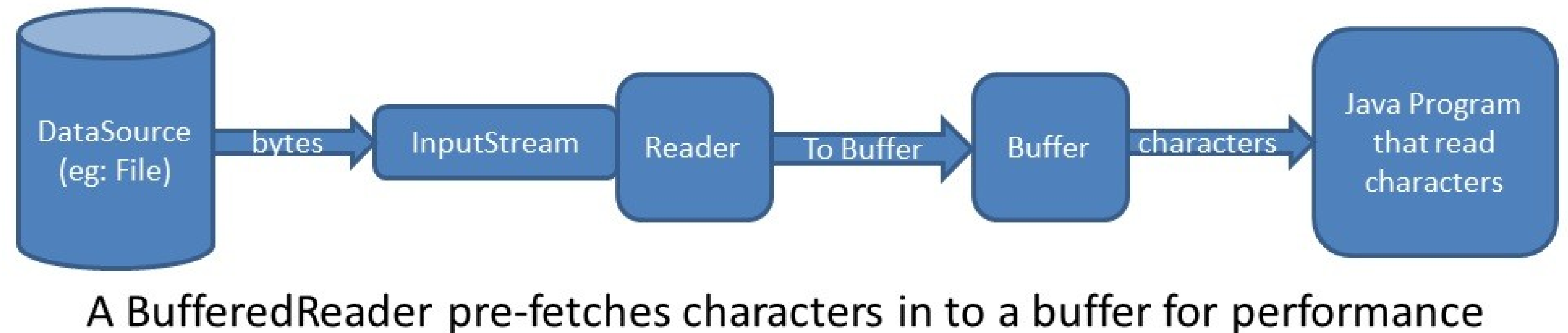- The File class doesn't have any kind of utility to obtain a sorted list.

Advantages

- With this class we achieve full independence of the OS notation.
- The File class, in collaboration with the virtual machine, will adapt itself transparently to the OS.

Methods of File Class in Java

The **File class methods** can be categorized into several categories:

- GET
- SET
- CAN
- IS
- FUNCTIONAL
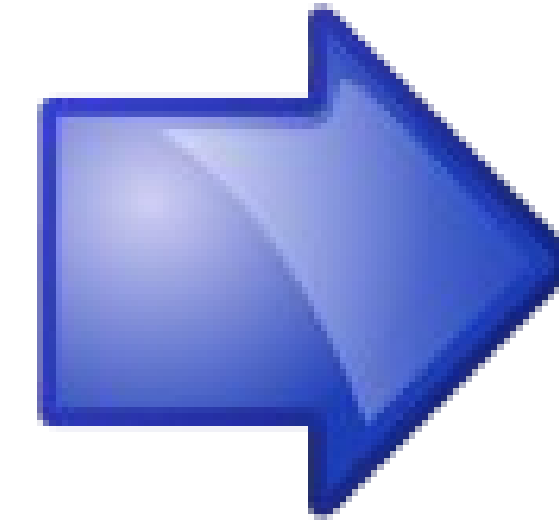
These are just some of the methods. For more information, please refer to the Oracle documentation at:

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

GET Methods

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

SET Methods

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

CAN & IS Methods

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

FUNCTIONAL Methods

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

New IO API (nio)

The File API problems has evolved into the **new IO API (nio)** and you can go further this topic in the link:

https://jenkov.com/tutorials/java-nio/nio-vs-io.html

IO stream oriented: (A → advantage, D → disadvantage)

(A) You read one or more bytes at a time, from a stream.

(A) They are not cached anywhere.

(D) You cannot move forth and back in the data in a stream. If you need to do it, you will need to cache it in a buffer first.

NIO's buffer oriented: (A → advantage, D → disadvantage)

(A) Data is read into a buffer from which it is later processed.

(A) You can move forth and back in the buffer as you need to. This gives you a bit more flexibility during processing.
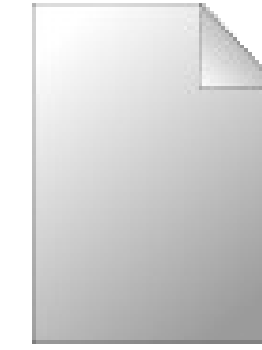
(D) You also need to check if the buffer contains all the data you need in order to fully process it.

(D) You need to make sure that when reading more data into the buffer, you do not overwrite data in the buffer you have not yet processed.

6.2 Constructors
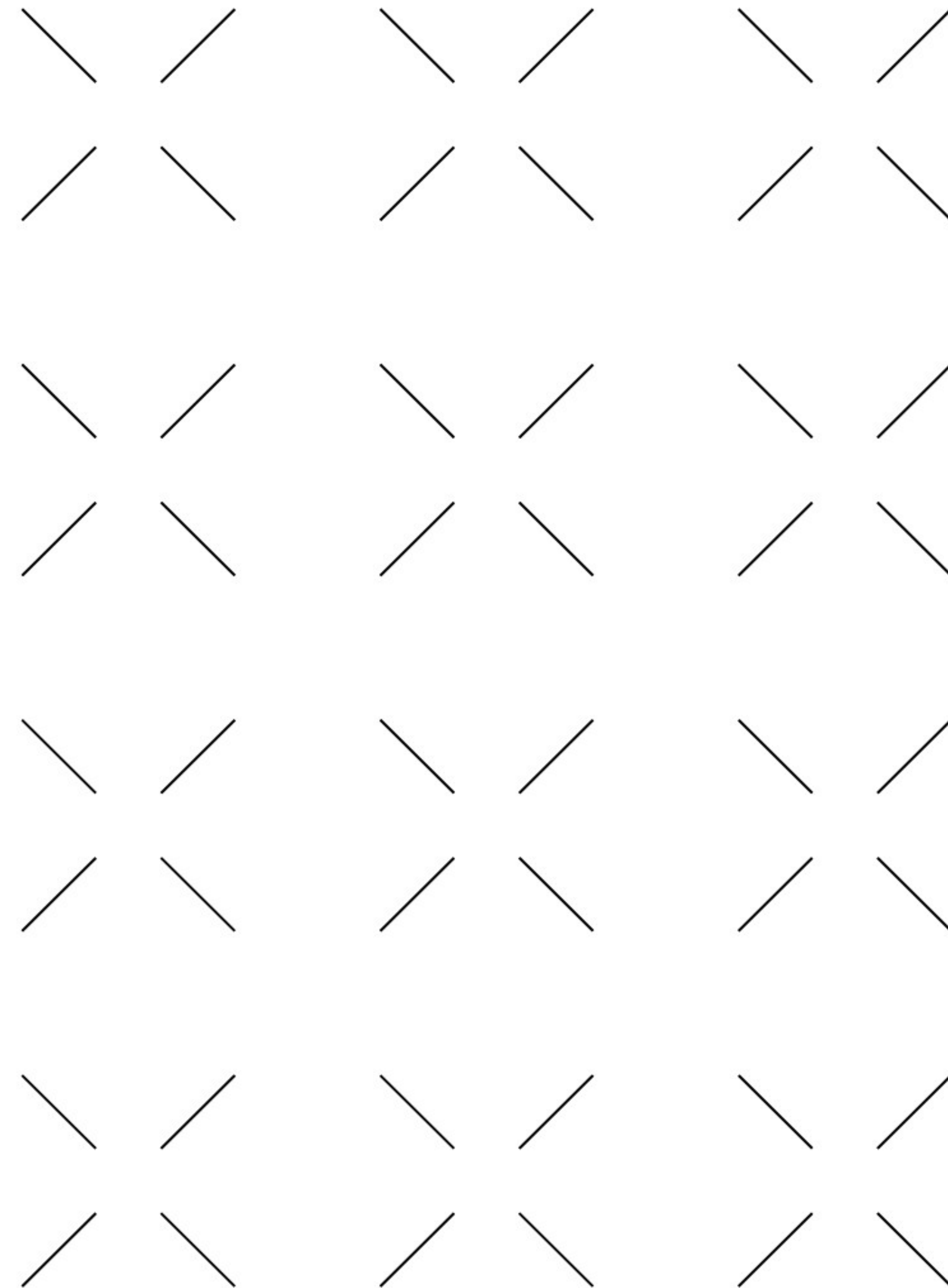
How to Create a File Object?

We need to create a new instance to work with the file/folder. This can be done very simply and **its constructor has 4 overloads**:

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

Depending on the operating OS family for which we are programming, we must use one nomenclature or another for the file paths:
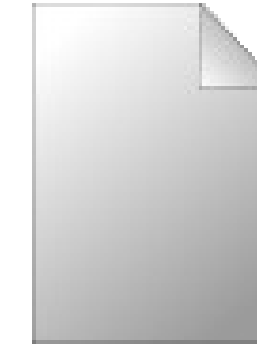
- **Unix like** (Linux, freeBSD, Mac OS, Android, IOS, etc.):

```java
File fiUnix = new File("/usr/local/bin/ada.txt");
```

- **Windows**:

```java
File fiWindows = new File("C:\\Users\\Admin\\DocumentsandFiles\\ada.txt");
```

How to Create a File Object?

1) The **first overload** receives one string (abstract pathname): ex. <mark>File("file.txt")</mark>

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

2) The **second overload** receives two strings: ex. <mark>File("/home/ada", "file.txt")</mark>
   1st = path to the parent (relative or absolute).
   2nd = path relative to the parent in the first parameter.

3) The **third overload** is a variant of the first: ex. <mark>File(new File("/home/ada"), "file.txt")</mark>

4) The **fourth overload** accepts an URI (Uniform Resource Identifier) with this syntax: ex. <mark>file:/ada/file.txt</mark>

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

Simple example

Once we have the file or folder, we can check its properties, such us **exists** or **getAbsolutePath**:

```java
String stCurDir = new File("").getAbsolutePath();

    System.out.println("Current folder: "+ stCurDir);

    File fiFile1 = new File("example1.txt");

    if(fiFile1.exists()) {
        // do something
    }
```

Full example

Reads a file to access its information.

Then checks if the file exists.

Try it placing a file called "example1.txt" on the root folder of a new Java project.

```java
// DAM.ADA U1 EXAMPLE 1: CHECKING IF FILE EXISTS
    import java.io.*;

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

            File fiFile1 = new File("example1.txt");
            if(fiFile1.exists()) {
                System.out.println("File found");
            } else {
                System.out.println("File not found");
            }
        }
    }
```

6.3 Exceptions
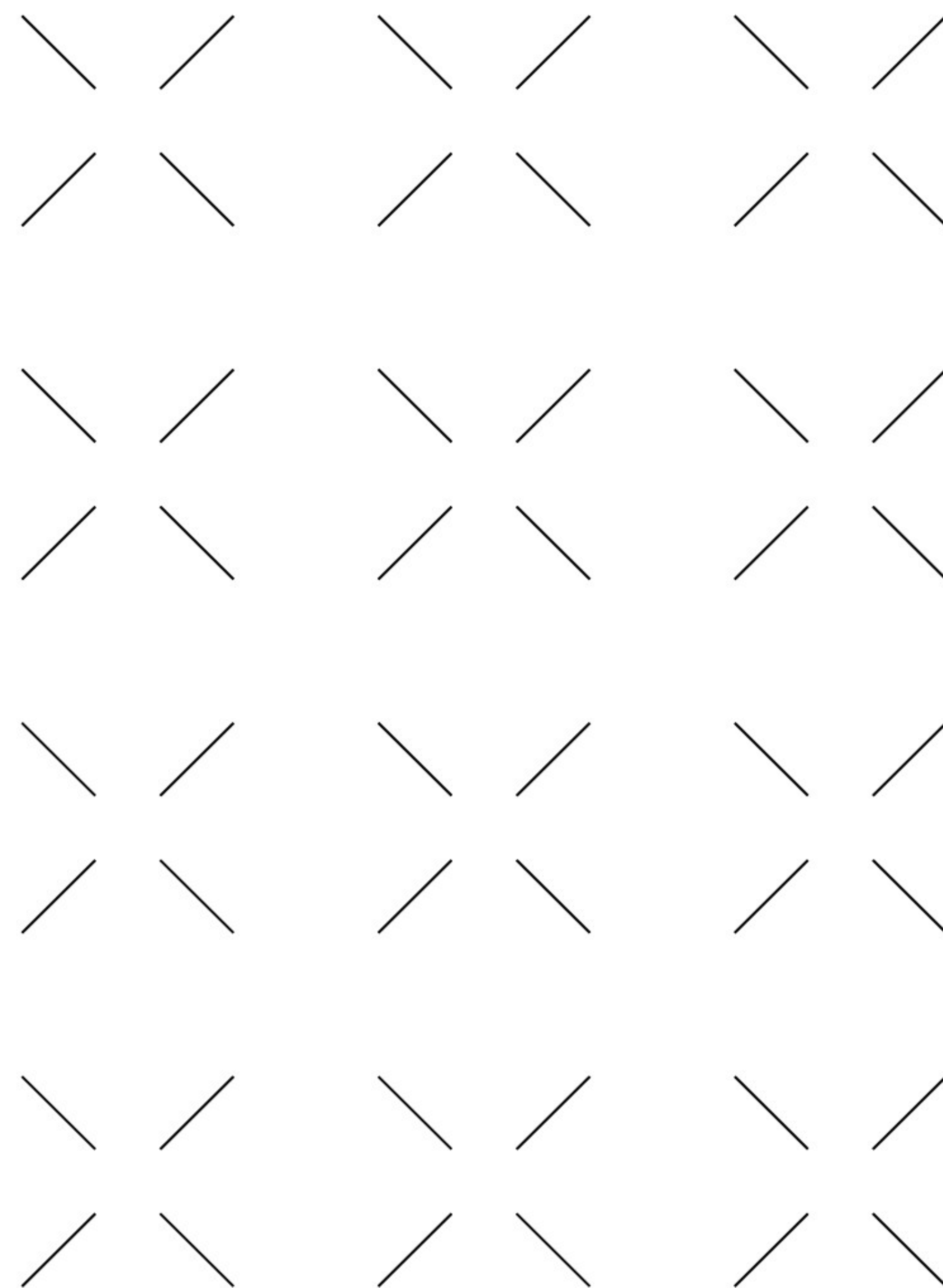
Errors with file operations
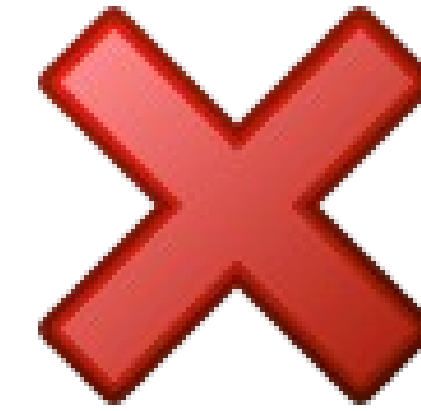
Before we can begin to read and write files, we'll have to talk about **handling error states**, which will occur a lot when working with files.

Errors will often occur in our programs, mainly errors caused by input/output operations, often referred to as I/O.

In all of these cases, there is a user who can enter invalid input or non existent/invalid files or a file has been moved or deleted unexpectedly.

However, we won't let them crash our programs due to errors. Instead, we'll inform the user about the situation.

Exceptions

We use exceptions when the operation is complex and it'd be too difficult to sanitize all of the possible error states. **Exceptions are referred to as a passive error handling**.

We don't have to deal with the internal logic of the method we call in any way. All we have to do is try to run the vulnerable part of the code in a "protected mode".

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

For further information: https://www.ictdemy.com/java/files/exceptions-in-java

Example

When dealing with files, as we will see later, we need to catch the exception and close the file either if the operation goes well or bad.

Using **finally**, Java remembers that the try-catch block contained finally and calls the finally block even after leaving the try or catch block.

Note that the *openFile*, *writeToFile* are just placed to make it easier to explain.

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

6.4 Reading text files in Java
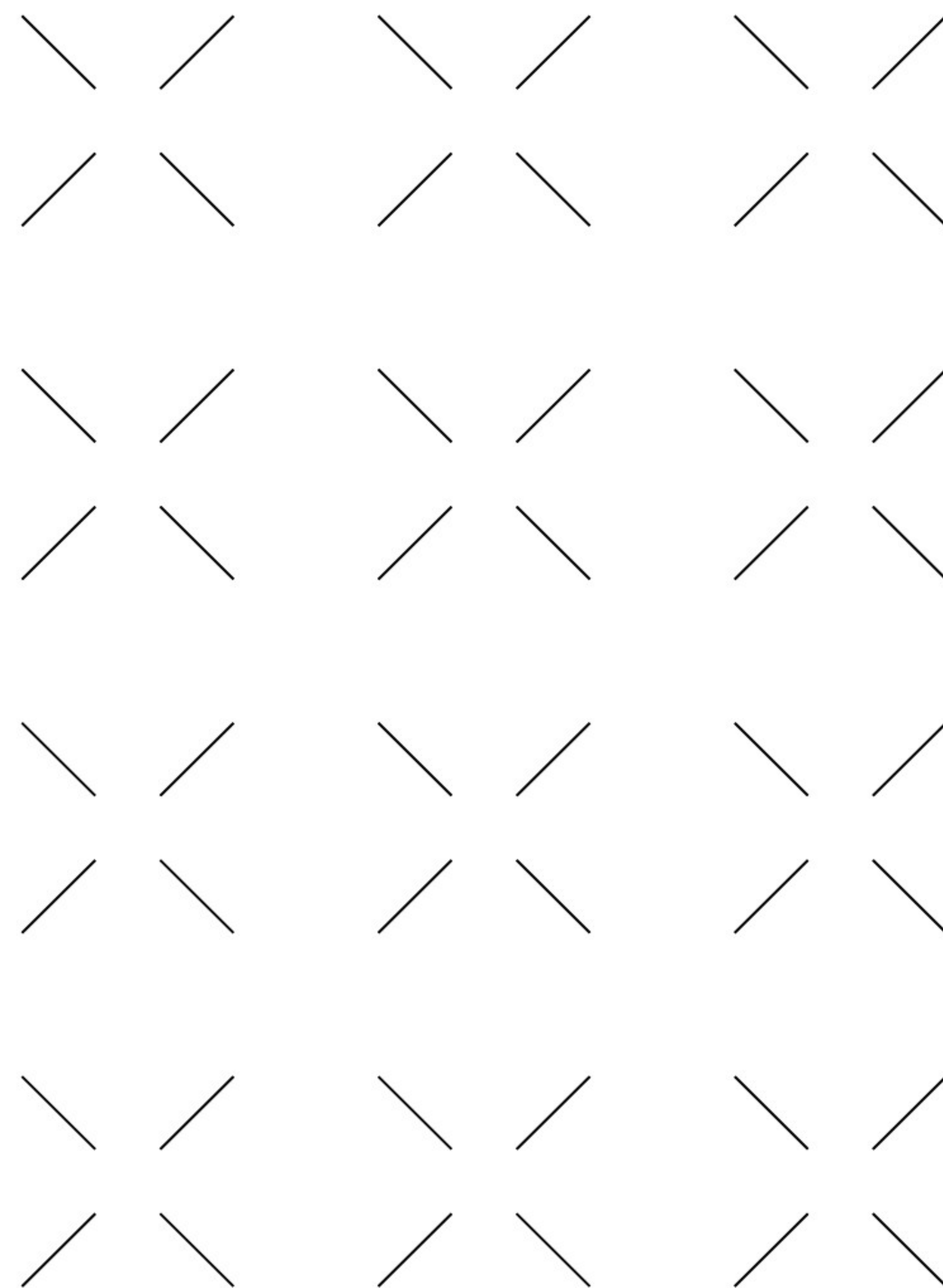
Text Files In Java

To work with text files we will use:

- **FileReader** to read.
- **FileWriter** to write.

Whenever we work with these classes **we must always carry out a correct management of exceptions** since they can produce:

- **FileNotFoundException** → In case of not finding the file.
- **IOException** → When some kind of writing error occurs.

Reading Text Files In Java

There are several methods to read files as you can check here:

- https://www.baeldung.com/reading-file-in-java

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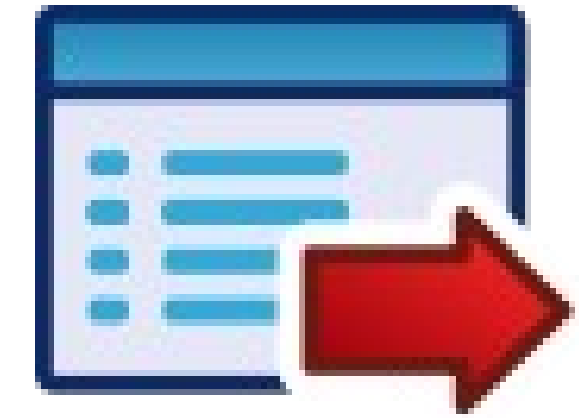
- https://www.stackchief.com/blog/FileReader%20vs%20BufferedReader%20vs%20Scanner%20%7C%20Java

We will show the most easy one but you should CHECK them all.

For further information:

- Ictdemy. Lesson 3 - Working with text files in Java

  https://www.ictdemy.com/java/files/working-with-text-files-in-java

Reading Text Files using BufferedReader

This method does buffering for efficient reading of characters, arrays, and lines.

The buffer size may be specified, or the default size may be used. The default is large enough for most purposes.

It is therefore advisable to wrap a BufferedReader around any Reader whose read() operations may be costly, such as FileReaders and InputStreamReaders. For example:

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

Example using BufferedReader

```java
// DAM.ADA U1 EXAMPLE 2: READING TEXT FILES
    import java.io.*;
    public class ReadFromFile
    {
      public static void main(String[] stArgs)throws Exception
      {
        File fiFile = new File("example2.txt");
        BufferedReader brBuffer = new BufferedReader(new FileReader(fiFile));

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

Example using BufferedReader (WITH EXCEPTIONS)

```java
// DAM.ADA U1 EXAMPLE 2: READING TEXT FILES
    import java.io.*;
    public class ReadFromFile
    {
      public static void main(String[] stArgs)throws Exception
      {
        try
          {
          File fiFile = new File("example2.txt");
          BufferedReader brBuffer = new BufferedReader(new FileReader(fiFile));

          String stLine;
          while ((stLine = brBuffer.readLine()) != null)
            System.out.println(stLine);
          brBuffer.close();
        }
        catch (FileNotFoundException efn) { System.out.println("File not found"); }
        catch (IOException eio) { System.out.println("IO Error"); }
      }
    }
```
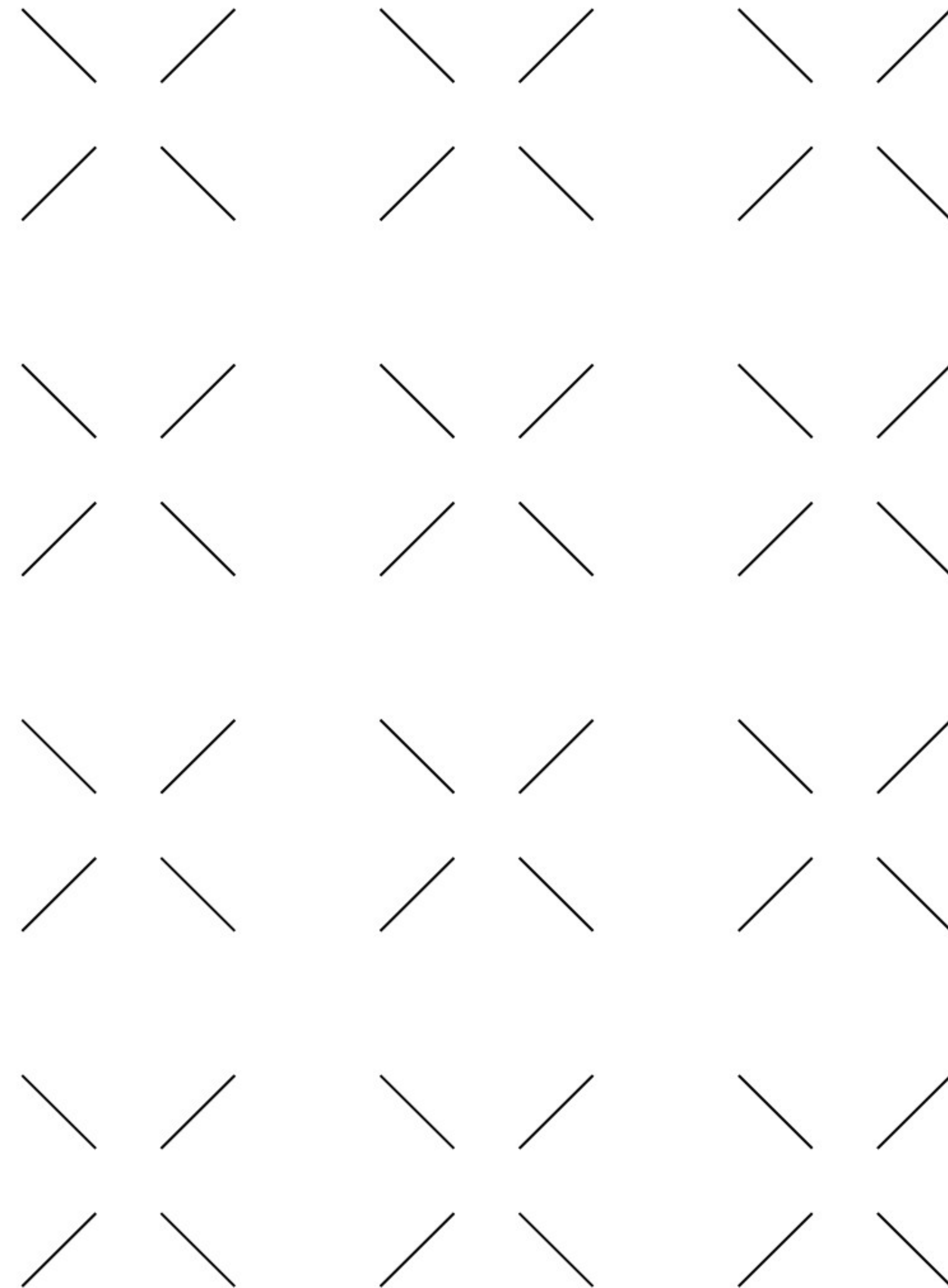
6.5 Writing text files in Java

Writing Text Files In Java

The same happens with writing. Check several methods to write files here:

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

We will show the most easy one but you should CHECK them all.

For further information:

- Ictdemy. Lesson 3 - Working with text files in Java

    https://www.ictdemy.com/java/files/working-with-text-files-in-java

Writing Text Files using BufferedWriter

It is used to write text to a character-output stream. It has a default buffer size, but the large buffer size can be assigned.

It is useful for writing characters, strings, and arrays. It is better to wrap this class with any writer class for writing data to a file if no prompt output is required.

For example:

BufferedWriter out = new BufferedWriter(Writer out);

Example using BufferedWriter

```java
// DAM.ADA U1 EXAMPLE 3: WRITING TEXT FILES
    import java.io.*;
    public class WriteFromFile
    {
      public static void main(String[] stArgs)throws Exception
      {

            String stText = "ADA example for writing text files";

        BufferedWriter bwBuffer = new BufferedWriter(new FileWriter("example3.txt"));
        bwBuffer.write(stText);

        System.out.print(stText);
        System.out.print("File is created successfully with the content.");

        bwBuffer.close();
      }
    }
```

Example using BufferedWriter (WITH EXCEPTIONS)

```java
// DAM.ADA U1 EXAMPLE 3: WRITING TEXT FILES
    import java.io.*;
    public class WriteFromFile
    {
      public static void main(String[] stArgs)throws Exception
      {
          try {
        String stText = "ADA example for writing text files";

        BufferedWriter bwBuffer = new BufferedWriter(new FileWriter("example3.txt"));
        bwBuffer.write(stText);

        System.out.print(stText);
        System.out.print("File is created successfully with the content.");

        bwBuffer.close();
      }
      catch (FileNotFoundException efn) { System.out.println("File not found"); }
      catch (IOException eio) { System.out.println("IO Error"); }
    }
  }
```

7. PROPOSED ACTIVITIES

Proposed activities

Check the suggested exercises you will find at the "Aula Virtual". **These activities are optional and non-assessable but** understanding these non-assessable activities is essential to solve the assessable task ahead.

"The best way we learn anything is by practice and exercise questions. Here you have the opportunity to practice the Java programming language concepts by solving the exercises starting from basic to more complex exercises". W3CSchools

If you look for the solutions surfing the Internet or asking the oracle of ChatGPT you will be fooling yourself. Please note that **ChatGPT is neither infallible nor all-powerful**. If you use it, check the proposed solutions carefully. So, try to solve the problem using the resources we gave you and the extended documentation you will find at the "Aula Virtual".

Shortly you will find the proposed solutions enclosed in a single PDF.

What Now?

This week you should …

1) Install your favourite IDE and Java in your favourite Operating System (checking the provided documentation).

2) Try to do the review exercises (depending on your expertise).

3) Try to do the suggested exercises.

4) Check the materials for next week BEFORE attending to the next TC.

8. BIBLIOGRAPHY

Resources

- Oracle Documentation. 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/
- Absolute Java. Chapter 10 File I/O. Walter Savitch. Pearson.
- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desenvolupament d'aplicacions multiplataforma. Creative Commons. Departament d'Ensenyament, Institut 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).