Topic: The use of File class.
OOP concepts involved: Classes, Objects, Constructors.
Programming generic concepts involved: Functions, Variables, Data Types, Arrays, Control Statements, Access Modifiers.

➢ Theoric introduction

FILE MANAGEMENT

A file is a collection of information, located or stored as a unit somewhere on the computer.

- ➔ The files are the organized set of information of the same type. As a material support of this information.
- ➔ The files as data collection are used for the input and output to the computer and are managed through programs.

The files can be contrasted with ***arrays*** and registers. However, for arrays, you must specify the fields, the number of elements in an array, the number of characters in a string; This is why it is denoted as ***static structures***. A predetermined size is not required in the files; This means that you can make larger or smaller data files, as needed.

- ➔ Each file is referenced by its id (its name).

BINARY AND TEXT FILES

In general, you could say that all files are binary, but they have a radical difference.

Text file

Flat text files are those that are **composed only of plain text, only characters**. These characters can be encoded in different ways depending on the language used.

- ➔ They are also known as plain text files or plain text because they lack information to generate formats and fonts.
- ➔ In a text file, each series of 8 bits corresponds to an equivalent in the ASCII code that can be a letter, a number, a symbol, etc. (It also depends on the type of coding used for a said file).

Binary file

A binary file is a file stored in binary format. A binary file is computer-readable but not human-readable. All executable programs are stored in binary files, as are most numeric data files. In contrast, text files are stored in a form (usually ASCII) that is human-readable.

In a binary file, the set of bits has an array of bits and a length that can vary from one file to another, and even from one length to another.

Binary files are also known as "typed" (with type) files, they contain data of a simple or structured type, such as integer, float, double, etc., except for other types of files.

Consequently, a binary file is a file that contains information of any type, encoded in binary form for the purpose of storage and processing.

- ➔ Many binary formats contain parts that can be interpreted as text.
- ➔ A binary file that only contains information of textual type without information about the format of it, said to be a plain text file.
- ➔ Usually, the terms binary file and text file are contrasted so that the first ones do not contain only text.
- ➔ A binary file is a file that reads byte by byte without assuming any structure.

Binary files are not a new type of file, but a new way to manipulate any type of file. The binary file techniques allow to read or change any byte of a file.

JAVA FILE CLASS

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class has several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory, and determining several common attributes of files and directories.

- ➔ First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- ➔ Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

How to create a File Object?

A File object is created by passing in a String that represents the name of a file, or a String or another File object. For example,

```
File a = new File("/user/Desktop/example");
```

defines an abstract file name for the example file in directory /user/Desktop/example. This is an absolute abstract file name.

CONSTRUCTORS OF THE FILE CLASS

Constructor Header	Description
File(File parent, String child)	Creates a new File instance from a parent abstract pathname and a child pathname string.

File(String pathname)	Creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child)	Creates a new File instance from a parent pathname string and a child pathname string.
FIle(URI uri)	Creates a new File instance by converting the given file: URI into an abstract pathname.

IMPORTANT METHODS OF THE FILE CLASS

- ➔ **static File** *createTempFile(String prefix, String suffix)*
  It creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
- ➔ **boolean** *createNewFile()*
  It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
- ➔ **boolean** *isAbsolute()*
  It tests whether this abstract pathname is absolute.
- ➔ **boolean** *isDirectory()*
  It tests whether the file denoted by this abstract pathname is a directory.
- ➔ **boolean** *isFile()*
  It tests whether the file denoted by this abstract pathname is a normal file.
- ➔ **String** *getName()*
  It returns the name of the file or directory denoted by this abstract pathname.
- ➔ **File[]** *listFiles()*
  It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
- ➔ **long** *getFreeSpace()*
  It returns the number of unallocated bytes in the partition named by this abstract pathname.
- ➔ **String[]** *list(FilenameFilter filter)*

It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfies the specified filter.

➔ **boolean** *mkdir()*

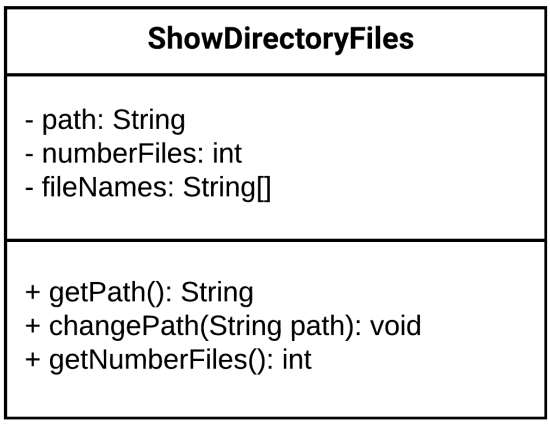
It creates the directory named by this abstract pathname.

➢ Statement

Create a class called **ShowDirectoryFiles**, which receives a String as the constructor's parameter. This String will be the path from which we will show the contained files.

ShowDirectoryFiles must have a *showFiles()* method which we will call to show the files in a certain path, as well as the number of files present.

➢ Class design (UML)



➢ Program Code

ShowDirectoryFiles.java

```
import java.io.File;

public class ShowDirectoryFiles {
	private String path;
	private int numberFiles;
	private String[] fileNames;
```

```
	public ShowDirectoryFiles(String path) {
		this.path = path;
		numberFiles = 0;
	}

	public void showFiles() {
		File file = new File(path);
		fileNames = file.list();		// Getting a list of files

		for(String fileName : fileNames) {
			System.out.println(fileName);
			numberFiles++;
		}

		System.out.println("\nNumber of files in the directory:
"+numberFiles);
	}

	// Getters and setters
	public String getPath() {
		return path;
	}

	public void changePath(String path) {
		this.path = path;
		numberFiles = 0;
	}

	public int getNumberFiles() {
		return numberFiles;
	}
}
```

Main.java

```
public class Main {
	public static void main(String[] args) {
		ShowDirectoryFiles sdf = new ShowDirectoryFiles(".");	// Showing
files of the actual relative directory
		sdf.showFiles();

		System.out.println("\n\n");
		sdf.changePath("./.settings");
		sdf.showFiles();
	}
}
```

➢ Program execution

This program creates an object of type **ShowDirectoryFiles** where a relative path is passed as a parameter, in this case, it is the current path: ".".

Then the *showFiles()* method is used to show the name of the files contained in the route, as well as the number of files in that directory. Following this, the path is changed so that there is a reference to the path "./.settings". When finished, the *showFiles()* method is used again to show the files available in the given route.

```
.classpath
.project
.settings
bin
File Demos
src

Number of files in the directory: 6



org.eclipse.jdt.core.prefs

Number of files in the directory: 1
```

➢ Conclusions

As users, we can write and read files whether in text or binary format. In essence, the content is the same, the only thing that changes is the representation we give to it.

In Java, there are two classes that allow us to read binary data and text data. The **InputStream** class allows us to create a flow to read binary data, this class has a diversity of more specialized subclasses.

The **Reader** class allows us to read data in text format and, as the **InputStream** class, it has more specialized subclasses.