Exercise 1 Introductory lesson

Objectives

The main goals of this work are:

1. Install the necessary Java development tools
2. Compare the structure of a program in Java and another in Python
3. Understand the organization of Java projects and programs
4. Edit, compile and run Java programs (based on the tools previously installed)
5. Compile and execute using the command line *(optional)*
6. Prepare a Java project for the next class

Exercise 1.1 Installing the Java development tools

There is a high probability that you already have Java installed on your personal computer. However, that version is composed of a set of components (e.g., libraries) needed to run Java programs on your computer (i.e., the Java Runtime Environment or JRE). For you to build your programs, you need to install another version, called the Java Development Kit (JDK), which is the development system for Java (contains libraries, compiler, interpreter, etc.). With the compiler and a common text editor (i.e., Notepad), you can write and compile Java programs using the command line. However, Java is widely used in projects of large size, making the use of integrated software project development tools, also known as the Integrated Development Environment, or IDE, appealing. There are multiple IDEs, such as Eclipse, Intellij, NetBeans, CLion, Geany, among others.

In this course, we will privilege the use of **Visual Studio Code (VS Code)**.

Since IDEs do not intend to address any particular programming language, they rarely come with pre-installed compilers. Usually, IDEs allow the installation of extensions, which enable them to interpret and compile languages selected by users.

In the case of VS Code, there is a pre-designed package, which already includes the IDE, the JDK, and an extension ("Coding Pack for Java") for development in Java. You can obtain this package from:

Windows - https://aka.ms/vscode-java-installer-win
macOS - https://aka.ms/vscode-java-installer-mac
Linux: *Different alternatives (Note that you must install the JDK by yourself according to your distribution and processor architecture)*
- VSCode (only) *deb or .rpm packages*:
  https://code.visualstudio.com/download
- VSCode (only) *distribution-based packages*:
  https://code.visualstudio.com/docs/setup/linux
- "Java Extension Pack"*you can download it from the VS Code in-app marketplace or via*
https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack

Once the program is installed, we can go to the menu **View □ Explorer**, and choose the option **Create Java Project**. Then, select the option **No build tools**. After that, a file explorer window appears, so that we can indicate where we want our project to be hosted. Finally, another text box appears so that we can identify the name of our project (e.g., OOP).

On the left side, you can observe a file structure with the "lib" and "src" folders. The "lib" folder will contain the libraries needed for our project, and the "src" folder will contain the source code (i.e., .java files) of our project. We can verify that inside the folder "src" exists a file "App.java" that has an example of a program in Java.

To compile and run the program we can go to the menu **Run □ Run Without Debugging**, or simultaneously press the **Ctrl** and **F5** keys on the keyboard, or simply right-click on the "App.java" file and select the **Run** option. After this, the program is compiled and the result appears in a terminal window embedded in the VS Code, with the phrase "Hello, World!".

To familiarize yourself with the VS Code, you can read some sources. For example:
https://code.visualstudio.com/docs/introvideos/basics
https://code.visualstudio.com/docs/java/java-tutorial

Exercise 1.2 Comparing code in Java and Python

Let's look at the code of the file "App.java"

```
public class App {
   public static void main(String[] args) throws Exception {
      System.out.println("Hello, World!");
   }
}
```

Now we have the Python equivalent:

```
def main():
   print("Hello, World!");
```

We intended to verify the differences between Python and Java syntax. There are many other differences, namely Python is an interpreted language, whereas Java needs to compile the source code and then run it. Both languages have performance differences as well.

From a syntax point of view, check the code for the following differences:

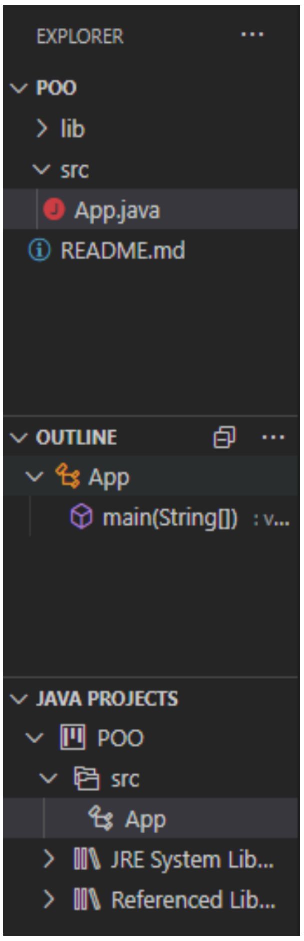
- A Java program needs to define a class and all the code is within that class. That is, in Java, everything is an object.
- Everything in Java (functions, variables, etc.) belongs to a data type (e.g., void, int, etc.). In Python, the data type is dynamic. Because of that, we say that Java is **Strongly Typed** and Python is **Weakly Typed**. Check the following Java example:

```
public class App {
 public static void main(String[] args) throws Exception {
    String frase = "Hello, World!";
    System.out.println(frase);
 }
}
```

- A Java program needs to have a function called `public static void main(String[] args);`
- Python's syntax is based on indentation. In Java, indentation has no value: the scope of each instruction (e.g., function, loop, class) is defined within a set of "{" and "}". Usually, we say that what is included among the "{….}" is within a **block**. Besides, each expression in Java is terminated by a ";". If these elements fail, there is a syntax error signaled by the compiler.

More details on the syntax differences between both languages can be checked in the Python-Java-Comparison.pdf file available in eLearning.

Exercise 1.3 Understand the organization of Java projects and programs



In the VS Code (with the extensions coming from the previously installed package), there are three areas of interaction with the project, shown in the figure on the left. The first, *EXPLORER*, shows you the file structure in the folder (i.e., Workspace) where you have the code for your project. The second, *OUTLINE*, presents in a contextual way the methods and elements of the class presented in the editor. The third, *JAVA PROJECTS*, allows you to manage the Java project.

We are going to create a new class in the existing "POO" project. To do this, in the "JAVA PROJECTS" component, move the "src" folder with the mouse pointer, where the "+" symbol will appear. Click on this symbol and select the option "Create New Class". Create a class called "MyFirstClass" (without the quotes).

a) Write the following code in the editor and run the program.

```
public class MyFirstClass {

    public static void main(String[] args) {
           System.out.println("Hello VS Code!");
    }
}
```

b) Modify the code according to the following example and execute. Analyze its functioning. Make other changes to the program (values, operations, ...) and check for errors / results.

```
public class MyFirstClass {

  public static void main(String[] args) {
    System.out.println("Hello VS Code!");
    int sum = 0;
    for (int i = 1; i <= 100; i++) {
      sum += i;
    }
    System.out.println(sum);
  }
}
```

c) Outside of VS Code, open the file management program (*Explorer, File manager, Finder,* ...) and check the structure of folders and files you have created so far. What folders are there? What are they used for? Compare with the "EXPLORER" structure in VS Code.

Exercise 1.4 Command line compilation and execution

Note: This exercise is optional.
The package installed in exercise 1.1, in addition to installing the VS Code and Java extensions, also installs the JDK (in this case, version 11). This means that we have access to the compiler through the command line (and not only through the VS Code).

After installing the JDK you can start writing and running programs, with the help of a simple text editor, the compiler (javac), and the interpreter (java). You can use these three programs through the command line.

Using any text editor (vim, notepad, etc.), create the Hello.java file with the following content:

```java
public class Hello {
   public static void main(String[] args) {
      System.out.println("O nosso primeiro programa!");
   }
}
```

a) Using the command line (*Terminal on Linux, COM on Windows, etc.*), compile this file using the command: **javac Hello.java**

Note: if you have problems running the javac and java programs, set the PATH environment variable to tell the operating system the location of the Java compiler (look for solutions online depending on the operating system you are using).

After compiling the code, make sure that the Hello.class file has been created in the same folder.

b) Use the java command to run the created program: **java Hello** Make sure the program does what you want.

Exercise 1.5 Other code examples

In the aula01 folder of elearning there are several java files. Briefly, analyze each program, execute and check its result.

Exercise 1.6 Prepare the Java project for the next class (Aula2)

Taking into account the procedure performed in Exercise 1.3, in the "EXPLORER" component, move the mouse pointer over the folder name "src" and of the 4 possible options that appear, choose to add a new folder, giving it the name of "Aula2". Mouse over the "Aula2" folder, now select the option to add a new file and name it "Ex1.java" (i.e., Java code file from the first exercise in *Exercise 2*). A basic code template is automatically produced by VS Code in the text editor. Notice that the first line of this program now says "package

Aula2;”. This means that all the classes you produce in this "folder" belong to a "package" (i.e., set) of Java classes that, in this case, are associated with "Aula2". This mechanism allows us to access other classes. In the next practical lesson, you can continue on this file, creating new classes for each of the exercises in *Exercise 2*, and so on.