ALGORITHMICS - Bachelor of Software Engineering, 2024

LAB GUIDE. SESSION 1.1

GOALS:

- **Measure execution times**

1. Introduction

In the previous session it was established that there were two ways to measure execution times in Java: WITHOUT OPTIMIZATION and WITH OPTIMIZATION. Then, we are faced with the dilemma: do we measure times WITHOUT OPTIMIZATION, which are more reliable with the theoretical complexity of the algorithms although greater? or, do we measure times WITH OPTIMIZATION, which are shorter but sometimes they give us surprises related to the expected time complexity that is analyzed on paper?

We will make a Solomonic decision: in the first sessions (until lab 3) we will take times WITHOUT OPTIMIZATION as a priority, thus prioritizing to match the time complexity. After that, the times will be usually measured WITH OPTIMIZATION, seeking to get the best results in terms of execution times.

2. Code to start working on this lab

You will now work with the `Vector1.java` class provided. This way of working, **packaging classes**, has the advantage of being able to structure and use the information much better, so it will be **our way of working from now on.**

If we use the JDK with the command line, we just need to place ourselves in the folder with the class we want to compile and type:

- `javac Vector1.java`

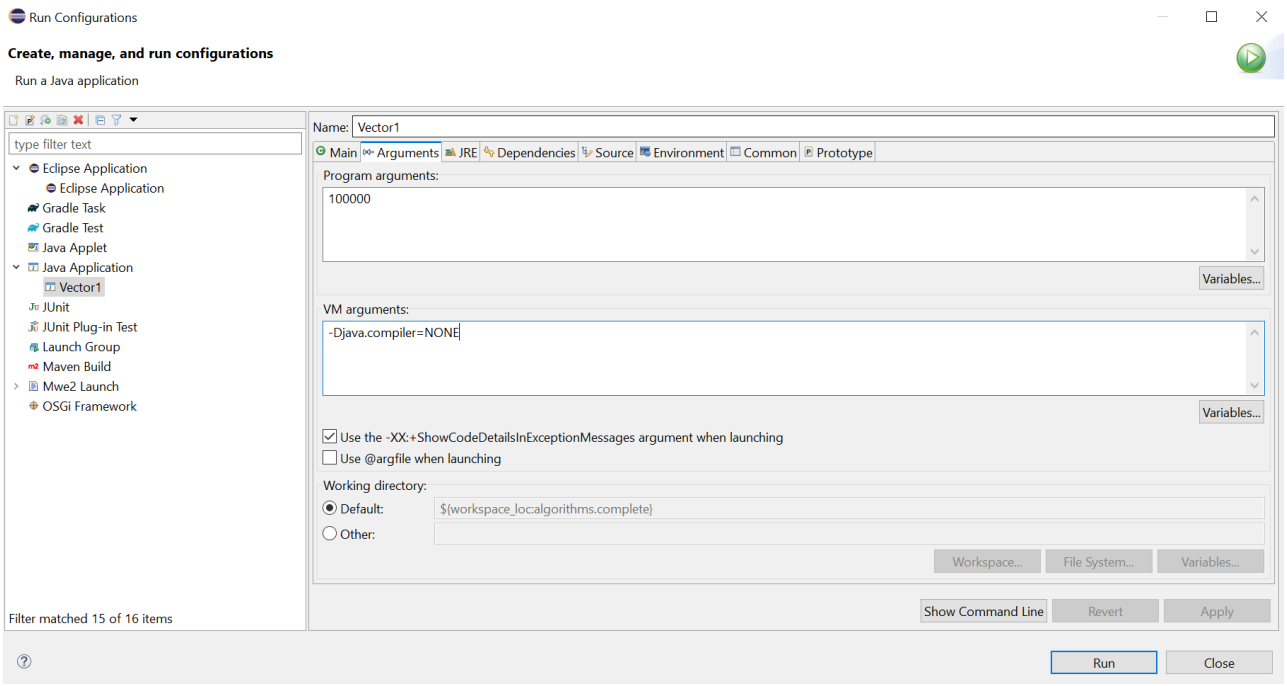
Next, we can verify that the `Vector1.class` file appears in that same folder, doing:

- `dir`

Since the package of the class is `algstudent.s11` we should create that folder structure and place the file in there. That is, we should create the folders `algstudent/s11`. Then, we can copy and paste the `Vector1.class` file there. Thus, we can type:

- `java -Djava.compiler=NONE algstudent.s11.Vector1`
- `java -Djava.compiler=NONE algstudent.s11.Vector1 5`
- `java -Djava.compiler=NONE algstudent.s11.Vector1 50`
- `java -Djava.compiler=NONE algstudent.s11.Vector1 500`
- `java -Djava.compiler=NONE algstudent.s11.Vector1 5000`
- …

However, for this course we will use the **Eclipse IDE**. To compile and execute the code, we use the option "**Run as ...**". To add the arguments in the execution, you must set up them in "**Run configurations ...**". For example:



3. Measuring execution times

The idea is to perform an **empirical study** of the execution time of some programs. That way, we can determine whether they match the theoretical behavior obtained from the **analytical study** of their time complexity.

We are going to use the Java method called `currentTimeMillis()` from the `System` class of the `java.lang` package (that is the only Java package that is preloaded, without the need for explicitly importing it using the `import` statement). The `currentTimeMillis()` method returns an integer of type `long` (64 bits) which is the current millisecond that the computer is living in that moment (the value of 0 has been more than 50 years ago). Put another way, it is the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC. So, if we call such a method twice, at the beginning and at the end of the measurement process, and if we subtract those two values, we will get the time spent between the calls in milliseconds. Considering that:

YOU ARE REQUESTED TO:

Calculate how many more years we can continue using this way of counting.

If, when taking times, made as explained in the previous paragraph, we get only a few milliseconds (by putting a threshold, we will consider a number less than 50 milliseconds) we will not use it for lack of reliability. The reason is that there are internal processes of the system (e.g., the so-called "garbage collector"), which are executed with a higher priority than our program (in fact, they stop it, although we may not realize that fact). The time of those system processes, in

addition to what it takes to end our process, sensitively distort the time obtained in the case of low times.

> CONCLUSION: We don't use times below 50 milliseconds, and the longer the time, the more reliable it is (3578 is more reliable than 123).

Next, we will use the class called `Vector2.java`, which is a program that should be able to measure the time of an operation (algorithm), namely the **addition of the `n` elements of a vector**, that we have in the `Vector1` class. We will pass an argument to the program with the size of the vector as in the previous case.

That it does it so fast and as is under 50 milisecond it does not take into account

YOU ARE REQUESTED TO:

What does it mean that the time measured is 0?

From what size of problem (n) do we start to get reliable times? 6000000

4. Growing of the problem size

It is impractical to vary the size of the problem by hand, especially if we consider that we usually want to draw a graph of the time depending on the size of the problem for an operation. So, how can we obtain the different values to draw the graph?

Use the class `Vector3.java` that will increase the size of the vector, obtaining times for each case. In this way, you will be able to follow more conveniently the evolution of the execution time.

However, it is observed that the times are so low that we cannot measure them, so we are going to take another step to be able to measure times as low as necessary in each case.

5. Taking small execution times (<50 ms)

When the process takes very little time, we can run the process to be measured several `repetitions`, adjusting this parameter, since on the one hand, we must ensure that the total execution time exceeds the 50 milliseconds and on the other hand, the process must end in a reasonable time.

Although `repetitions` can be any value, it is advisable to test with values that are powers of 10 because the conversion of times is as easy as applying the following table:

nTimes	*Time units*	
1	Milliseconds	(10^{-3} sg.)
10	Tens of millisec.	(10^{-4} sg.)
100	Hundreds of millisec.	(10^{-5} sg.)
1 000	Microseconds	(10^{-6} sg.)
10 000	Tends of micros.	(10^{-7} sg.)
100 000	Hundreds of micros.	(10^{-8} sg.)

The `Vector4.java` class introduce the previous idea.

If for any reason (e.g., it takes too long) an execution should be aborted, press **Control + C**. In Eclipse, you can use the red square button above the Console panel.

YOU ARE REQUESTED TO:

What happens with the time if the problem size is multiplied by 2? that the time is *2

What happens with the time if the problem size is multiplied by a value k other than 2? (try it, for example, for k=3 and k=4 and check the times obtained) si es 3 es el triple si es 4 es *4

Explain whether the times obtained are those expected from the linear complexity O(n) si

From what we saw in **Vector4.java** measuring the times for **sum**, create the following three java classes:

- **Vector5.java** to measure times for **maximum**.
- **Vector6.java** to measure times for **matches1**.
- **Vector7.java** to measure times for **matches2**.

With the times obtained from the previous classes (in milliseconds), fill in the following two tables:

TABLE1 (times in milliseconds WITHOUT OPTIMIZATION):

n	*Tsum*	*Tmaximum*
10000	…..	…..
20000	…..	…..
40000	…..	…..
80000	…..	…..
160000	…..	…..
320000	…..	…..
640000	…..	…..
1280000	…..	…..
2560000	…..	…..
5120000	…..	…..
10240000	…..	…..
20480000	…..	…..
40960000		
81920000	…..	…..

TABLE2 (times in milliseconds WITHOUT OPTIMIZATION):

n	*Tmatches1*	*Tmatches2*
10000	…..	…..
20000	…..	…..
40000	…..	…..
80000	…..	…..
160000	…..	…..
320000	…..	…..
640000	…..	…..
1280000	…..	…..
2560000	…..	…..
5120000	…..	…..
10240000	…..	…..
20480000	…..	…..
40960000		
81920000	…..	…..

Indicate the main features (processor and memory) of the computer where times have been measured.

Once both tables are filled in, conclude whether the times obtained meet what was expected, given the computational time complexity of the different operations.

6. Work to be done

- An `algstudent.s11` **package** in your course project. The content of the package should be the Java files used and created during this session.
- A `session11.pdf` **document** using the course template (the document should be included in the same package as the code files). You should create one activity each time you find a "YOU ARE REQUESTED TO" instruction (e.g., in this document you should do 3 different activities answering the different questions you will find in each point).

Deadline: The delivery of this lab will be made on the same date as the next one. More instructions will be given.