CSC9A3 Practical 4

The Debugger, and Introducing Complexity!

Intro

In this practical, we will check how to use the debugger mode in Eclipse. We are algo going to check the utility of logging concepts. **NOTE THAT, WE WILL USE THE MAIN METHOD TO RUN THIS PRACTICAL. THEREFORE, THERE IS NO JUNIT CLASS OR @TEST IN THIS PRACTICAL.**

Getting Started

A skeleton code for this practical has been created for you. It can be download on Canvas.

Copy these files to a new project directory and open them in your desired IDE.

The Problem

In this code, we have a class, called `Module`, which represents the courses offered by the university, such as the A3. This class has two attributes: a String that stores the module name and a list of integers that stores the registration number of the students enrolled in that course. This class implements several methods, including: the default setters and getters, a method for adding new students, a method for removing students, a method for sorting the list of students, and a method for sequential searching a student in the list of students.

In this code, we also have the `Main` class which implements two methods: the main one and the `test_sequential_search` method that creates a new module with a random array of students and evaluates how long (in **nanoseconds**) the searching algorithm takes to find an element.

Try to run the main. You will notice that the code is currently raising an ArrayIndexOutOfBoundsException. Moreover, note that just a few lines are printed in the log, making it difficult to understand the source of the problem. It would be interesting to get more information in order to understand and fix the error. Go to the constructor of the `Module` class. You'll see that we are using a Logger to log the messages. Currently, this Logger is set to use the Level SEVERE, which is employed for printing messages that indicates a serious failure. There are other log levels, such as: WARNING, which is used for printing messages indicating potential problems, and INFO, which is employed for printing informational messages. Change the log level to WARNING. Run the program and observe that more messages are printed. Then, change the log level to INFO and run the program again. Observe that all messages, from INFO to SEVERE, are printed. Now, you can better analyse the error and problem of the code.

The error that is raising such an Exception is not so easy to spot and fix. Use the interactive debugger to spot and fix the error. You might need to check the previous lectures in order to do this. Ask for help if stuck. After fixing the bug, the method output should be [171, 954, 857, 486, 176] (note that the value 999 has been removed and no other value - such as 0 - has been added).

Now that you fixed this, your code should run without raising Exceptions. Set the Logger to Level SEVERE again, run the code and analyse the printed messages. Is everything correct? No! You still have one bug in your code, given that the array is not being sorted. Again, this error is not so easy to spot and fix. Use the interactive debugger to spot and fix the error. When you finish fixing the code, run it. Is everything working properly?

[Checkpoint]

When you finish fixing these two bugs, call a demonstrator and show your code.

Complexity

The `Module` class implements the sequential search. Now, let's implement the binary search. Check your lectures to remember this searching algorithm.

After you implement this search, try to analyse the complexity of each searching algorithm by just looking at the code. In particular, think about how often you think the loops will run each time. Which one do you think is quicker?

Let's try to answer this in practice. Create or amend the `Main` class so it can evaluate the running time of the new binary search using random arrays, as for the sequential search. Now, for **each** searching algorithm, vary the number of elements in the array (for example, from 1,000 to 100,000,000, every 10,000 or 100,000). For each case, run the program **3-5 times** and calculate the average time, in terms of nanoseconds, of these runs.

You will end up with a table starting like this (but you will have different numbers, these are just for illustration):

Array Size	Average time for sequential search	Average time for binary search
1,000	18300	2100
10,000	227200	7100
100,000	694300	15400
…	…	…

Then, create a plot of the average running time x the array size, using Excel or another application if you prefer. Now, which searching algorithm do you think is the best one?

[Checkpoint]

When you finish plotting your measures, call a demonstrator and show your plot and code.