COMP0008 Written Java Coursework 3 (2020/21)

NOTE: Your final Moodle submission will consist of a zip file in a similar structure to the one you uncompressed that contains:

1. **The final version of the Java code for your application (in src subdirectory).**
2. **A PDF file named "answers.pdf" placed in the "doc" subdirectory which gives answers to the questions below (given with bold stars). You can write this description using Word or Latex - as long as you submit a PDF version in the end.**
3. **You may leave this "2021_COMP0008_CW3_Description.pdf" file in the doc subdirectory if you wish.**

Getting going ...

Unzip the template Java code which is provided on Moodle. Template source code is within the "src" subdirectory and you should find this coursework description as the file "2021_COMP0008_CW3_Description.pdf" in the "doc" subdirectory. Take your favourite Java IDE and make a new project using this code. (You should be familiar with how to do this.)

Look through the Java code and analyse how it is meant to work. It is attempting to add up a sequence of double precision floating point numbers which are given as Strings in an array. The addition of these numbers is carried out in a separate background thread. Running the code calculates the sum of the array of numbers and also gives the time that the system took to calculate this in seconds.

Task 1: Getting familiar with the code.

Unfortunately the developer only half finished the code! You can see that the current data employed is {"1.0", "2.0", "3.0", "4.0"}. Running the code gives an answer of 0.0 which is clearly not correct. Work out how the Adder class is meant to behave to give the correct answer of 10.0 when used in the way the client is currently using it. **Find the bug and add suitable synchronization and conditional synchronization to the SerialAdder.java class so that it generates the correct answer and does not have any concurrency issues. Note that you should not change either the Main.java or Adder.java interface since these are defining how the SerialAdder should be working.**

***** QUESTION 1: Describe what caused this bug and how you fixed it. Also describe other changes you did to the code to make it thread safe.**

Task 2: Trying out different data

Hopefully you have the SerialAdder.java code producing the correct answer of 10.0 for DATA1. Now change Main.java so that it uses DATA2 instead of DATA1. Hopefully your new code will produce the correct answer of 3.0 ? Now change Main.java to use DATA3 (which contains exactly the same numbers as DATA2 but in a different order). Does it produce the correct answer of 3.0 in this case ?

***** QUESTION 2: Explain in detail why the code it is producing a different answer for DATA3 compared to DATA 2 when exactly the same values are being added together ! Use this fact to explain why you may get different results when running numerical calculations with multi-threaded code.**

Task 3: Speed it up challenge!

This part is more challenging and the instructions are less detailed. So you will have to work out how certain aspects need to be structured yourself and also worry about whether your code is thread safe.

Now change Main.java to use DATA4. The creation of DATA4 shows static initialization using “class initializer” code using the static keyword with a block of code. Static initialization of data values is guaranteed to be visible to all threads running in the system.

Change the POWER variable (currently set at 10) to create a suitable sized arrays of numbers that the SerialAdder takes a number of seconds (say between 4 seconds to 20 seconds ?) to sum this array. This variable represents powers of two ... so “10” would give 1024 data items, whereas 11 will give 2048 and 12 will give 4096. Values between 18 and 24 seem suitable for most machines. (This really depends on the speed of your machine.) In all cases the arrays should roughly sum to 1.0 so it is worth checking your code using this.

***** QUESTION 3: State in your report the time that Java says it takes to run your code after this adjustment of the number of data items to use. Also measure the actual time the Java code runs for (using a watch or mobile phone) and also note this time down in your report. You will probably find that the actual time it runs is much longer than the time it reports. Given that the reported time is from the start of the main() method to almost the end – can you explain the discrepancy in terms of these times?**

***** QUESTION 4: Do five runs and write down the different times your system takes to add the numbers together. Put this table into your report together with the average time taken for serial addition of the numbers. (If you wish, you may want to modify Main.java so it runs the Adder multiple times and produces a data table which you could analyse, potentially producing a plot of the distribution of different times taken).**

Now the final challenge! You probably have a multicore processor and the current system is only making use of a single core since it is running a single thread for all the calculations.

Change the Adder to use MultithreadedAdder (this only requires one line of code to be changed where this object is created since the rest of the code is employing the Adder interface. This is why “programming to the interface” is useful in software engineering since it decouples dependencies between how objects are used and how they are created.

Running using MultithreadedAdder will not work since someone hasn’t implemented it yet !

Implement a multithreaded version of the code where the number of threads to use, N, is specified via the API. The basic implementation should divide the data array into N parts and then get N threads to add up the N different parts. Finally, the thread running within the run() method of MultithreadedAdder should add together all these sub-totals to get a final total. You should use a CountDownLatch to control this thread having to wait for the other threads to complete.

Your challenge is to work out the detail required to get this architecture to work and also to worry about concurrency aspects – where might you need to add volatiles or synchronization? You should be accurate in your analysis and not just add volatiles and synchronization everywhere!

***** QUESTION 5: Do your timing analysis again with different numbers of threads (say 2, 4, 8, 16 threads). Is your MultithreadedAdder faster than the SerialVersion ? Give a brief (half a page at most) explanation of all your results.**

***** Please use the Moodle Forum to ask any questions about this coursework *****

***** Upload your zip file containing your code and your answers to Moodle *****