CSE443 – Object Oriented Analysis and Design

Homework 3 – Report

Yasir Nacak – 161044042

Q1.

In this part, I first created the BestDSEver structure as an array that has add, remove and get methods that are not thread safe. I then created an adapter class for this called BestDSEverThreadSafe which contained an instance of the BestDSEver. I provided the same interface as the BestDSEver with the same method names and the parameters. I called the methods of the BestDSEver. This class provides thread safety with the Java keyword *synchronized*. With adding this keyword to the adapter's method signatures, I created a thread safe version of the BestDSEver without modifying it.

Test Results (500 threads using class methods 500 times):

1. BestDSEver (Thread Unsafe):

```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...
Exception in thread "Thread-22" java.lang.ArrayIndexOutOfBoundsException: 10240
    at Q1.BestDSEver.Add(BestDSEver.java:31)
    at Q1.Main.lambda$TestDS$0(Main.java:32)
    at java.lang.Thread.run(Thread.java:748)
Exception in thread "Thread-46" java.lang.ArrayIndexOutOfBoundsException
Exception in thread "Thread-176" java.lang.ArrayIndexOutOfBoundsException
Exception in thread "Thread-411" java.lang.ArrayIndexOutOfBoundsException
Exception in thread "Thread-205" java.lang.ArrayIndexOutOfBoundsException
Exception in thread "Thread-206" java.lang.ArrayIndexOutOfBoundsException

Process finished with exit code 0
```

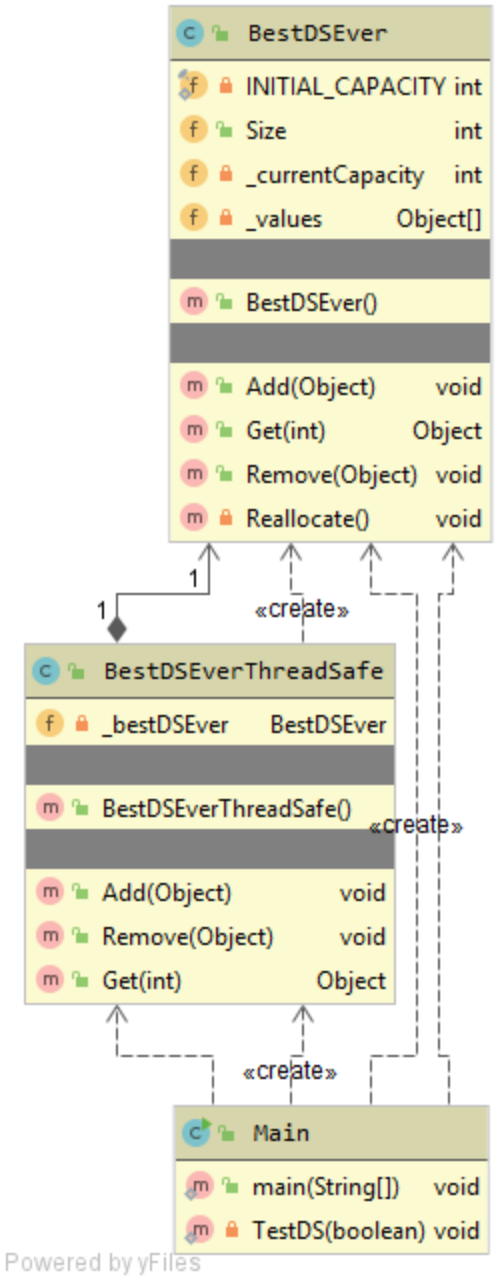
2. BestDSEverThreadSafe (Thread Safe):

```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...

Process finished with exit code 0
```

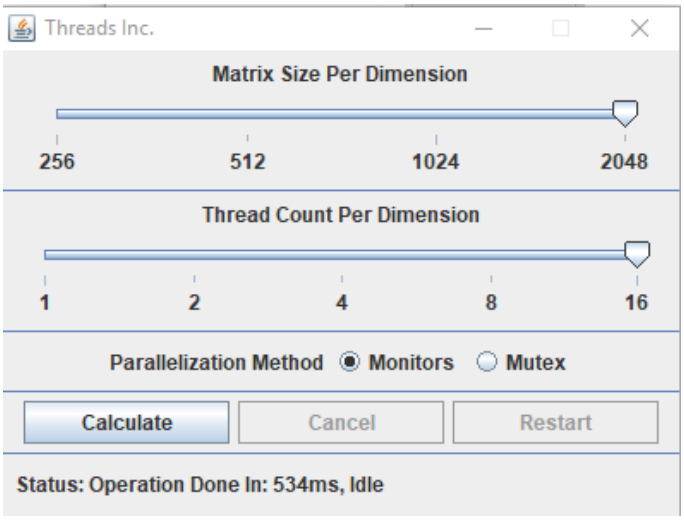
We can see that while the thread unsafe implementation crashes while multiple threads try to add elements to the data structure while the thread safe adapter gets no crashes and finishes successfully.

Class Diagram:



Q2.

In this part, I created a complex number class, a class that holds a matrix of complex numbers that performs calculations on it and a calculator class called AddAndDFT which takes matrix sizes and thread count then calculates the DFT(A+B). I randomized the values of the matrices, so the user doesn't need to provide them by hand when using the GUI. The calculator creates given amount of threads as worker classes. Worker class get constructed with given a range of index values for rows and columns. It then operates addition on the given range of indices in matrices A and B. After that, it waits for the synchronization barrier to end to start the DFT step. The implementation of the barrier mechanism left as abstract which makes the AddAndDFT class an abstract class. Classes that derive from AddAndDFT are expected to implement a barrier mechanism. This gives us the template method pattern where the matrix generation, thread instancing and calculations are in the base class while the barrier logic left as a template method. I then inherited two classes from AddAndDFT called AddAndDFTMonitor and AddAndDFTMutex where I implemented parallelization mechanisms using Java monitors and mutexes respectively. After this, I created a GUI class called ApplicationWindow where I put sliders for thread count and matrix size and provided two radio buttons to choose between monitors and mutexes for the parallelism method. In this application window, I hold an AddAndDFT field which gets constructed when the user pressed the calculate button. The type of the AddAndDFT depends on the radio button selected. With the help of a separate calculation thread, I kept my GUI responsive throughout the calculation. I also added a cancel and a restart button. The cancel button calls the cancel method off the AddAndDFT which changes a control variable from false to true, so the worker threads breaks from their calculation loops and returns immediately. The restart button first cancels the operation then starts it again. I added a status bar to the bottom of the window that tells the user the current state of the program. Whenever a calculation ends, the status bar tells how much time has elapsed since the start of the program.



The Application Window

Throughout my runs, I concluded that the number of threads start to cause an overhead after a certain limit which causes the program to utilize more resources while gaining so little time improvements. I also find an interesting result where the performance peaks at the number of threads that is equal to the cores of the CPU running the program.

Class Diagram: