**GTU Department of Computer Engineering**

**CSE443 Object Oriented Analysis and Design**

**Fall 2021 - Homework 3 Report**

**Akif KARTAL**

**171044098**

**Question 1**

Understanding this problem is tricky. Why? Check following image.

Timeline

Description automatically generated

As you can see adapter design pattern is looking good for this. **But,** it works for different incompatible interfaces. In this problem we don’t have such a situation.

In this problem **we need to control access** to BestDSEver class therefore check definition of the **proxy design pattern.**

Graphical user interface, text, application

Description automatically generated **Therefore, we will use Proxy Design Pattern.**

**Solution**

**1.1 Implement simple BestDSEver Class**



**1.2 Test BestDSEver Class**

Text

Description automatically generated

**Output:**

Text

Description automatically generated

**1.3 Implement Proxy Design Pattern**

**1.3.1 Understanding synchronization problem**

Since each thread is trying to use same data structure, we have critical sections. We will solve this problem by applying following solution.

Text

Description automatically generated

\*You can think push and pop methods as insert and remove methods.

**1.3.2 Creating Interface**

I will create a simple interface **for proxy class**. But in question we don’t know whether BestDSEver implements it or not. **We will not touch** **BestDSEver class** as the question denoted. Also, I will use **Protection Proxy(Access control) method**.

Text

Description automatically generated

**1.3.3 Creating Proxy Class**

A screenshot of a computer

Description automatically generated with medium confidence

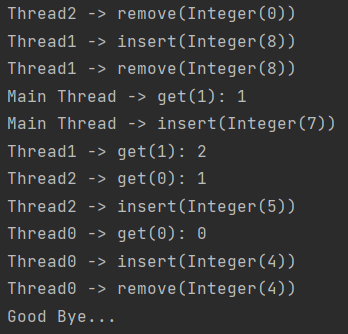
\*Here, we have a mutex for threads and we are using it in critical regions. We are making unlock in finaly block because if we get an exception in try block, still we should be able to unlock otherwise we will leave without unlocking. Also we have composition with BestDSEver class to use its methods.

**1.4 Testing with multiple Threads**

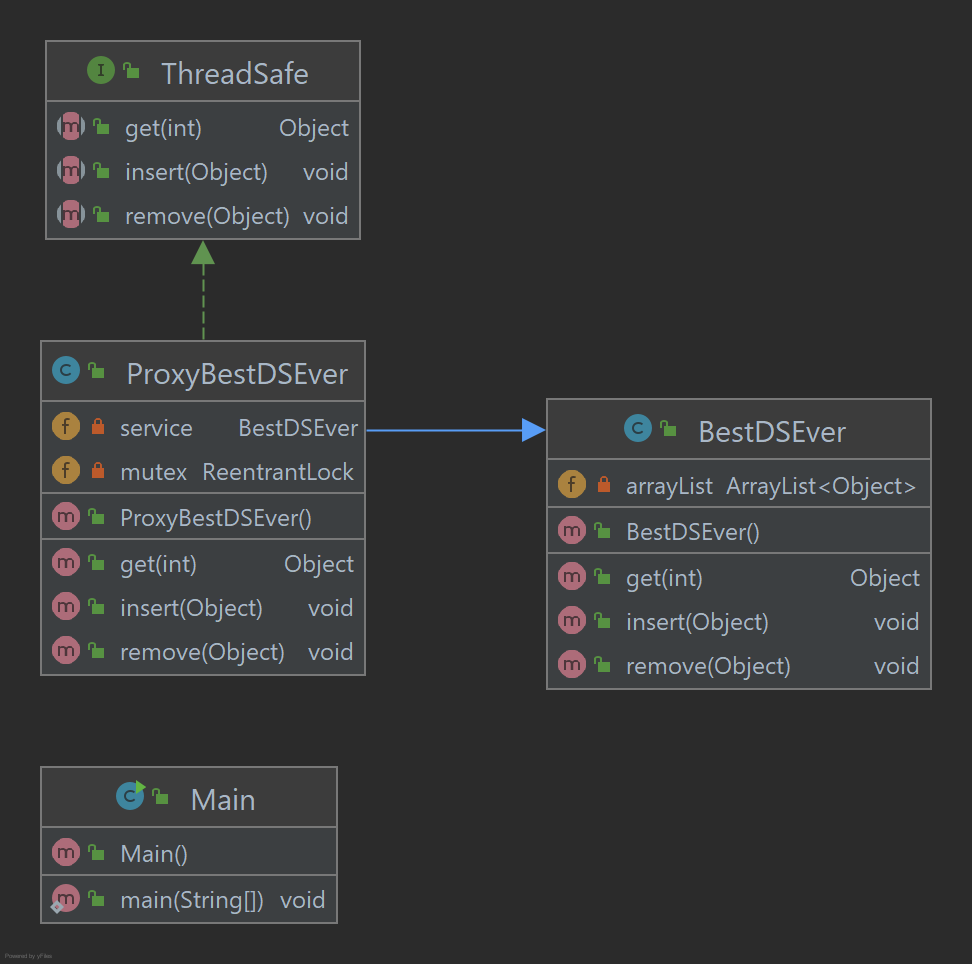
**Text

Description automatically generated**

**1.5 Output**



**1.6 Class Diagram**

****

\*Here, normally BestDSEver class can implement ThreadSafe interface but from question we have only a concrete BestDSEver class therefore it didn’t implement ThreadSafe interface.

**Question 2 – Composite and Iterator**

**Solution**

**2.1** **Understanding the problem**

The problem is like the following picture;

Diagram

Description automatically generated

Here, emails contains both address and name of its owner, groups contains an arbitrary number of personal or group addresses also **groups are composite, emails are leaf**.

**2.2 Class Diagram**

**2.2.1 Classes in my solution**

Graphical user interface, application

Description automatically generated

**2.2.2 Composite Pattern Class Diagram**

Graphical user interface, text

Description automatically generated

Here, EmailComponent class is abstract class, Email is leaf class and CompositeEmail is a composite class with add, remove and get methods.

**2.2.3 Full Class Diagram**

Here, we will see full diagram note that I used iterator design pattern to traverse all tree easily as expected in homework pdf file.

Graphical user interface

Description automatically generated

Here, Composite iterator implements **java.util iterator** interface to use in composite email class. NullIterator class is used in Email leaf class. Main class is for test purpose.

**2.3 Test Results**

Check Main.java class and run to see results. Some part of result is following;

**A screenshot of a computer

Description automatically generated with medium confidence**

**Question 3 – Concurrency patterns**

**Solution**

**3.1 Understanding the** **synchronization barrier problem**

In order to solve this problem, we will apply following solution in 2 different ways with java;

Text

Description automatically generated

**Some notes on my solution**

* In order **to create thread** in java I will use **Runnable interface** for both solutions.
* Note that, since all threads works with **different portion of the matrix** which means different buffer, we don’t have any synchronization problem other than synchronization barrier.
* In my solution, all threads will use **same thread function** with different coordinates.
* I will explain my solutions with simple example. You can check the source code after reading.

**3.2 Using Java’s synchronized**

**3.2.1 Classes in my solution**

Text

Description automatically generated with medium confidence

**3.2.2 Shared Data and Lock Object Class**

**Text

Description automatically generated**

\*Atomic integer is much better between threads in java.

**3.2.3 Common Thread Function between Threads**

****

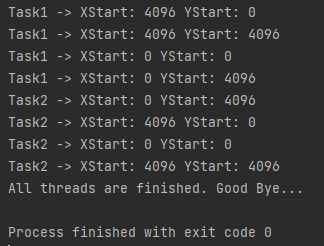
**\***Here, synchorized keyword acts like a mutex and it locks and unlock code in its scope. We are using **lockData object is** like a mutex(with the help of object class) since **it is common and shared between** all threads.

**3.2.4 Creating Threads and Testing**

Text

Description automatically generated

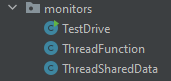
**3.3.5 Output**



**\***As you can see all task2s **didn’t start** all task1s are finished.

**3.3 Using mutex(es) and monitor(s)**

**3.3.1 Classes in my solution**



**3.3.2 Shared Data between Threads**

**Text

Description automatically generated**

\*Atomic integer is much better between threads in java.

**3.3.3 Common Thread Function between Threads**

**Text

Description automatically generated**

**3.3.4 Creating Threads and Testing**

**Text

Description automatically generated**

**3.3.5 Output**

**Text

Description automatically generated**

**\***As you can see all task2s **didn’t start** all task1s are finished.

**3.4 Calculating A+B and Discrete Fourier Transform**

**3.4.1 A+B Implementation**

**Text

Description automatically generated with low confidence**

**3.4.2 Discrete Fourier Transform Formula and Implementation**

Text, letter

Description automatically generated

**Implementation**

Text

Description automatically generated

\*Here, as you can see thread calculates dft for only its responsible portion of the matrix.

**3.5 Testing full implementation as console application**

In order to test we need to create 8192x8192 size of matrix and calculate dft on this matrix. But if I choose this size, I am getting **java.lang.OutOfMemoryError**. Therefore, **I tested my application with 4096x4096 size of matrix.**

**Java synchronized result**

**Text

Description automatically generated**

**Java monitor result**

**Text

Description automatically generated**

**Using a single thread result**

Text

Description automatically generated

\*As you can see in multi thread version we have gained about 1700 ms.

**Note:** As you have seen, main thread is used only for test purpose such as creating threads etc.

**3.5 Responsive GUI**

In java Swing GUI, **if we use these implementations directly** our gui **will not be respond** until calculation is done.

Therefore in order to solve this problem, I have used **java SwingWorker<T,V> class** and it worked.

**Definition:**

SwingWorker is designed for situations where you need to have a long running task run in a background thread and provide updates to the UI either when done, or while processing. \*docs.oracle.com\*

**3.5.1 My Simple GUI**

Graphical user interface, text, application

Description automatically generated

\*with monitors = multi thread

\* not with monitor = single thread

**3.5.2 Making it responsive while calculating**

First See start button action listener code

**Text

Description automatically generated**

**Canceling operation**

To cancel current operation, SwingWorker cancel method is used

Text

Description automatically generated

**Output:**

Graphical user interface, application

Description automatically generated

**3.5.3 See the total time gained by using multiple threads**

**Output:**

Graphical user interface, text, application

Description automatically generated

\*As you can see, we gained **nearly same amount of time as in console test**. Note that in this test I have used 4096x4096 size of matrix.

**3.5 Full Class Diagram**

Graphical user interface

Description automatically generated

**References**

* CSE344 System Programming Course slides
* Head First Design Patterns, 2nd Edition.
* <https://en.wikipedia.org/wiki/Discrete_Fourier_transform>
* <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantLock.html>
* <https://docs.oracle.com/javase/7/docs/api/javax/swing/SwingWorker.html>

**Java Version**

Project default java version is **17**. Therefore, executables(jar files) may not be working, if java 17 is not installed.