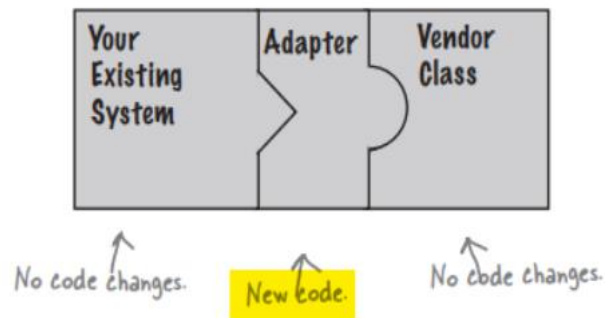


**GTU Department of Computer Engineering
CSE443 Object Oriented Analysis and Design
Fall 2021 - Homework 3 Report**

**Akif KARTAL
171044098**

Question 1

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



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**.

The Proxy Pattern provides a surrogate or placeholder for another object to control access to it.

Therefore, we will use Proxy Design Pattern.

Solution

1.1 Implement simple BestDSEver Class

```

4  public class BestDSEver {
5      private ArrayList<Object> arrayList;
6
7      public BestDSEver() {
8          arrayList = new ArrayList<>();
9      }
10
11     public void insert(Object o) {
12         arrayList.add(o);
13     }
14
15     public void remove(Object o) {
16         arrayList.remove(o);
17     }
18
19     public Object get(int index) {
20         return arrayList.get(index);
21     }
22 }

```

1.2 Test BestDSEver Class

```
3 public static void main(String[] args) {  
4     BestDSEver ever = new BestDSEver();  
5     Integer a = 5;  
6     Integer b = 10;  
7     ever.insert(a);  
8     ever.insert(b);  
9     ever.remove(b);  
10    System.out.println("Result: " + ever.get(0));  
11 }  
12 }
```

Output:

```
Result: 5
```

```
Process finished with exit code 0
```

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.

Thread t1	Thread t2	
lock (m)	lock (m)	
push (v)	v = pop ()	// critical section
unlock (m)	unlock (m)	

*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**.

```
1  /**
2   * Simple interface for proxy class.
3   */
4  public interface ThreadSafe {
5      public void insert(Object o);
6      public void remove(Object o);
7      public Object get(int index);
8  }
```

1.3.3 Creating Proxy Class

```
1  import java.util.concurrent.locks.ReentrantLock;
2  public class ProxyBestDSEver implements ThreadSafe{
3      private BestDSEver service;
4      private ReentrantLock mutex;
5
6      public ProxyBestDSEver() {
7          service = new BestDSEver();
8          mutex = new ReentrantLock();
9      }
10
11     @Override
12     public void insert(Object o) {
13         mutex.lock(); //lock(m)
14         try {
15             service.insert(o);
16         } finally {
17             mutex.unlock(); //unlock(m)
18         }
19     }
20
21     @Override
22     public void remove(Object o) {
23         mutex.lock(); //lock(m)
24         try {
25             service.remove(o);
26         } finally {
27             mutex.unlock(); //unlock(m)
28         }
29     }
30
31     @Override
32     public Object get(int index) {
33         mutex.lock(); //lock(m)
34         try {
35             return service.get(index);
36         } finally {
37             mutex.unlock(); //unlock(m)
38         }
39     }
40 }
```

*Here, we have a mutex for threads and we are using it in critical regions. We are making unlock in finally 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

```

9  ▼ public static void main(String[] args) {
10      ThreadSafe buffer = new ProxyBestDSEver();
11      Integer[] arr = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
12      buffer.insert(arr[0]);
13      buffer.insert(arr[1]);
14      buffer.insert(arr[2]);
15      buffer.insert(arr[3]);
16  ▼  Thread thread0 = new Thread() {
17  ▼      public void run() {
18          System.out.println("Thread0 -> get(0): " + buffer.get(0));
19          System.out.println("Thread0 -> insert(Integer(4))");
20          buffer.insert(arr[4]);
21          System.out.println("Thread0 -> remove(Integer(4))");
22          buffer.remove(arr[4]);
23      }
24  };
25  Thread thread1 = new Thread() {
26  ▼      public void run() {
27  ▼          System.out.println("Thread1 -> insert(Integer(8))");
28          buffer.insert(arr[8]);
29          System.out.println("Thread1 -> remove(Integer(8))");
30          buffer.remove(arr[8]);
31          System.out.println("Thread1 -> get(1): " + buffer.get(1));
32      }
33  };
34  Thread thread2 = new Thread() {
35  ▼      public void run() {
36  ▼          System.out.println("Thread2 -> remove(Integer(0))");
37          buffer.remove(arr[0]);
38          System.out.println("Thread2 -> get(0): " + buffer.get(0));
39          System.out.println("Thread2 -> insert(Integer(5))");
40          buffer.insert(arr[5]);
41      }
42  };
43  };

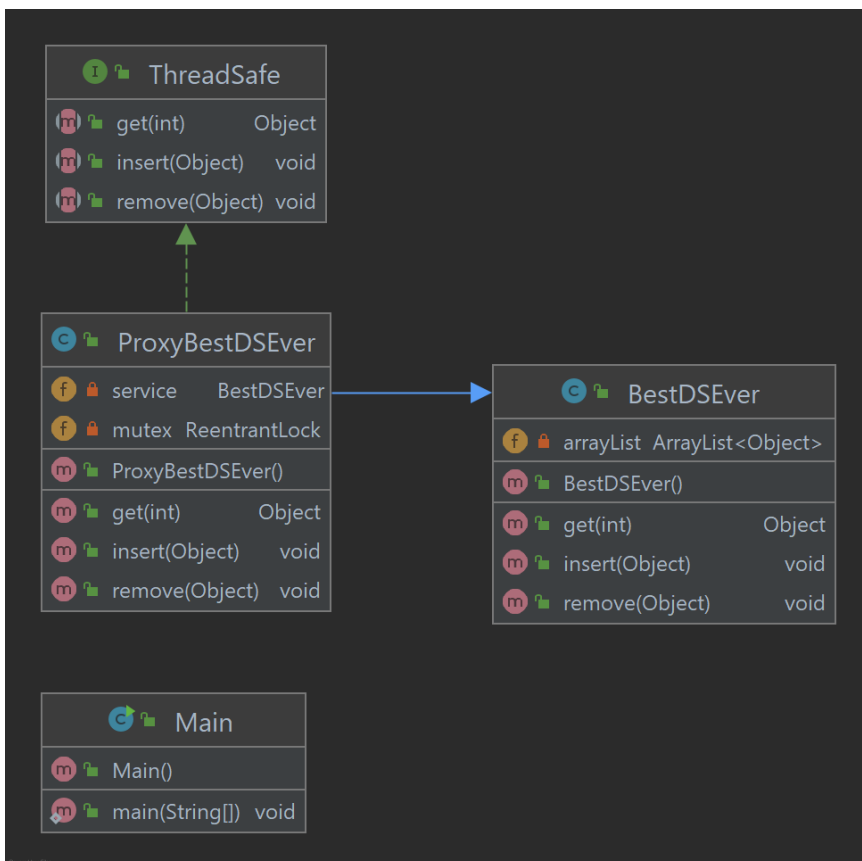
38      thread0.start();
39      thread1.start();
40      thread2.start();
41      System.out.println("Main Thread -> get(1): " + buffer.get(1));
42      System.out.println("Main Thread -> insert(Integer(7))");
43      buffer.insert(arr[7]);
44      try {
45          /*Make sure all threads have finished.*/
46          thread0.join();
47          thread1.join();
48          thread2.join();
49      } catch (Exception e) {
50          e.printStackTrace();
51      }
52      System.out.println("Good Bye...");
53  }
54  }

```

1.5 Output

```
Thread2 -> remove(Integer(0))
Thread1 -> insert(Integer(8))
Thread1 -> remove(Integer(8))
Main Thread -> get(1): 1
Main Thread -> insert(Integer(7))
Thread1 -> get(1): 2
Thread2 -> get(0): 1
Thread2 -> insert(Integer(5))
Thread0 -> get(0): 0
Thread0 -> insert(Integer(4))
Thread0 -> remove(Integer(4))
Good Bye...
```

1.6 Class Diagram



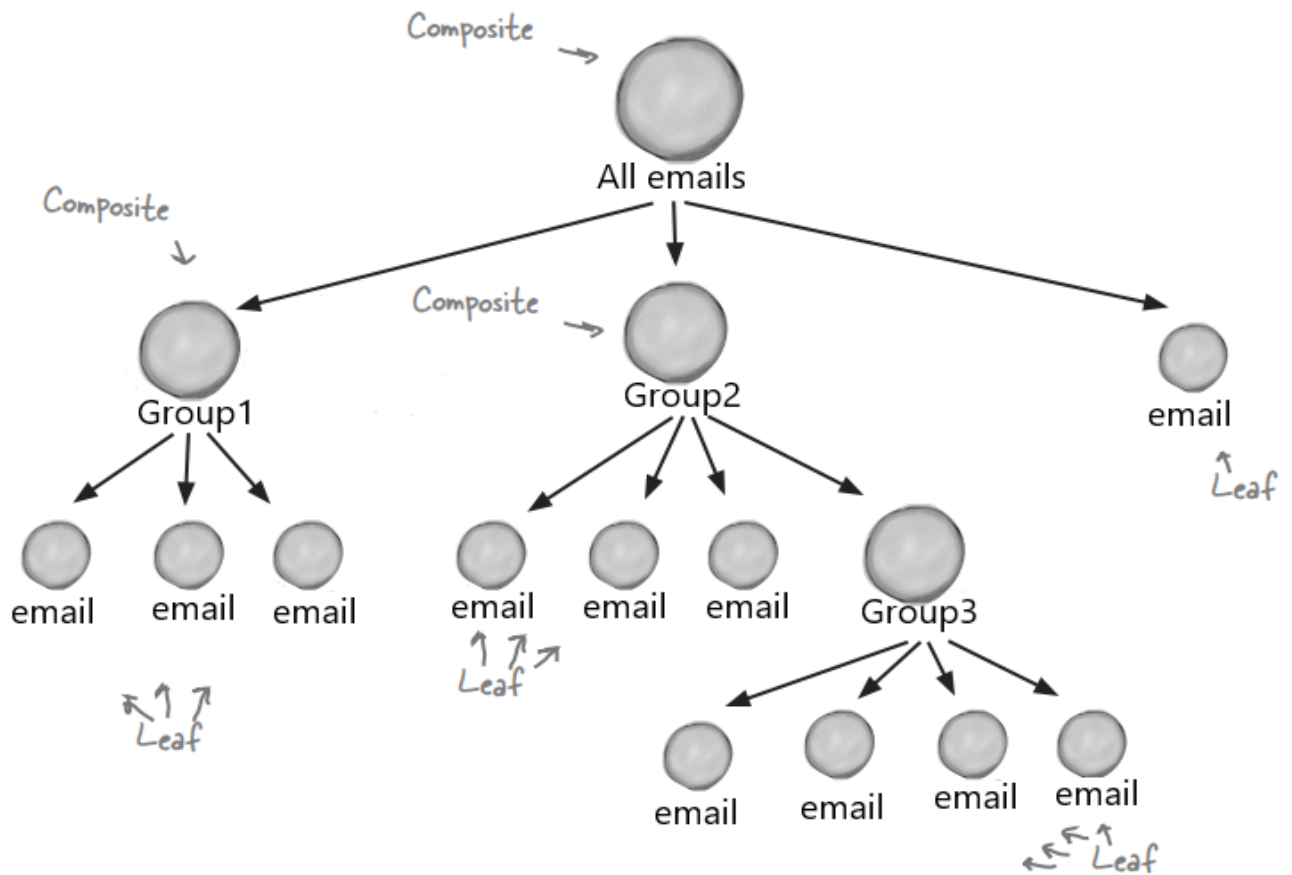
*Here, normally `BestDSEver` class can implement `ThreadSafe` interface but from question we have only a concrete `BestDSEver` class therefore it didn't implement `ThreadSafe` interface.

Question 2 – Composite and Iterator

Solution

2.1 Understanding the problem

The problem is like the following picture;



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**.

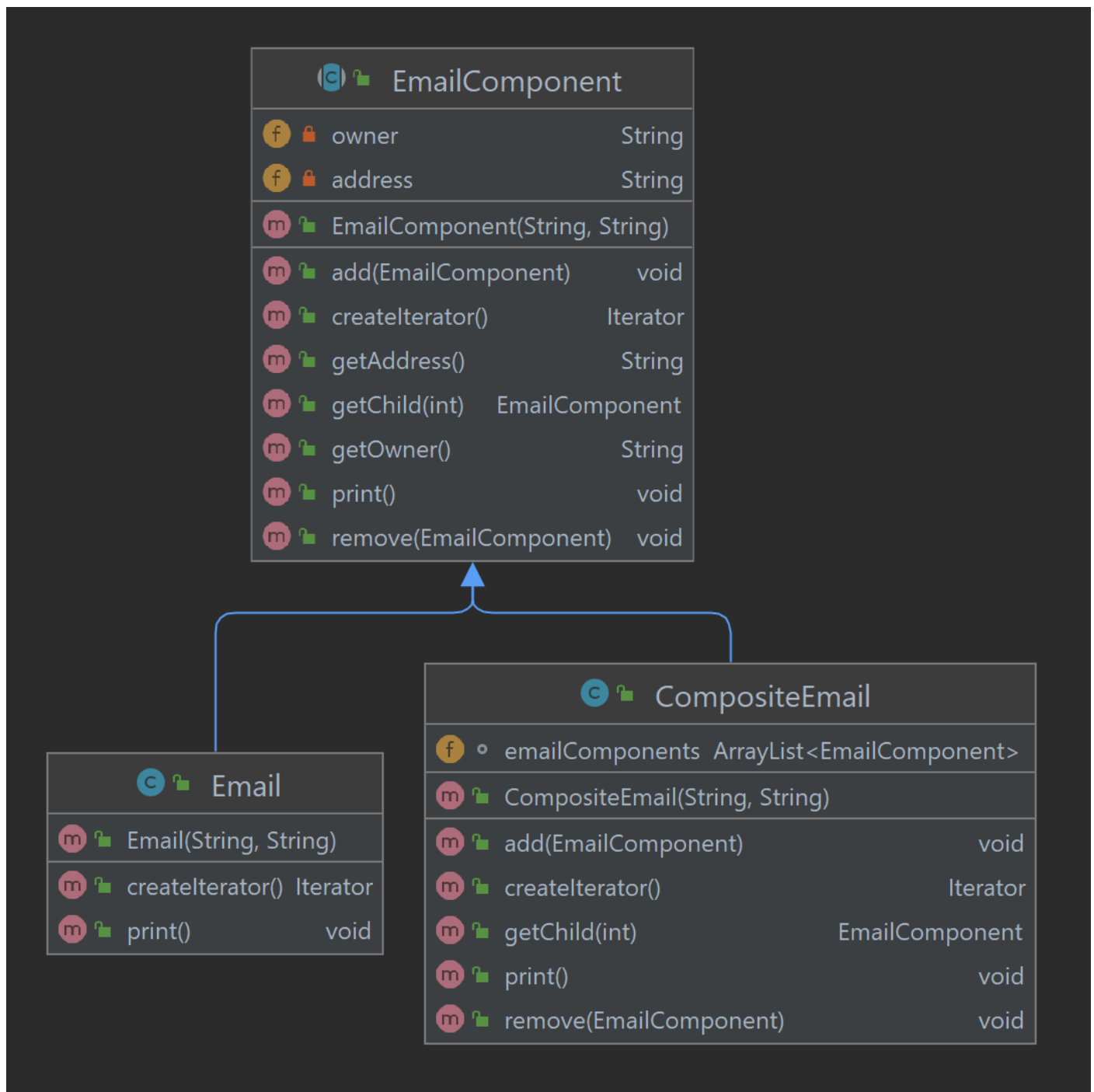
2.2 Class Diagram

2.2.1 Classes in my solution

```

com.Akif
├── CompositeEmail
├── CompositeIterator
├── Email
├── EmailComponent
├── Main
└── NullIterator
    
```

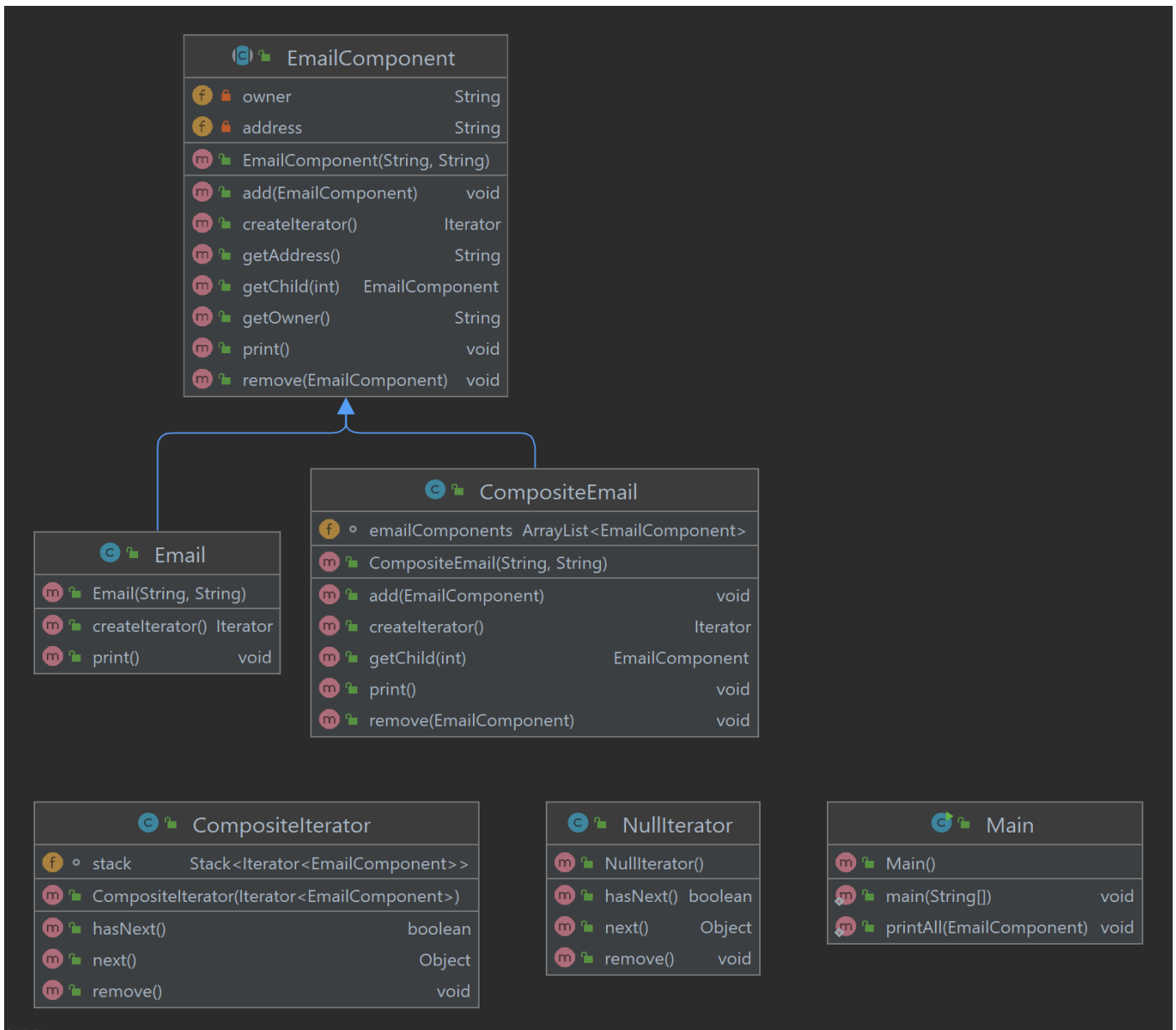
2.2.2 Composite Pattern Class Diagram



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.



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;

```

Owner: GTU Ceng All
-----
akif.kartal2017@gtu.edu.tr Akif Kartal
djuro2017@gtu.edu.tr Djuro RADUSINOVIC
mustafa.tokgoz2017@gtu.edu.tr Mustafa TOKGÖZ
m.karakaya2018@gtu.edu.tr Muhammed Emin KARAKAYA
m.kurtcebe2018@gtu.edu.tr Mehdi KURTCEBE
sinan.sari2016@gtu.edu.tr Sinan SARI
hboubati@gtu.edu.tr Alp Eser
-----

```

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;

Example: **synchronization barrier** with N threads.

Condition variable `c`, mutex `m`, `arrived = 0`

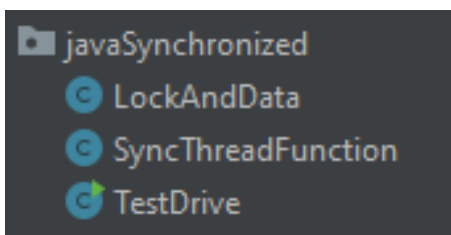
```
lock (m)
++arrived
if (arrived < N)    // if this thread is not last
    cwait (c,m)     // then wait for others
else
    broadcast (c)   // i'm last, awaken the other N-1
unlock (m)
```

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



3.2.2 Shared Data and Lock Object Class

```

7  /**
8   * This class is used for both shared data and
9   * as a lock object for synchronized.
10  */
11  public class LockAndData {
12      private ComplexNumber[][] matrixA;
13      private ComplexNumber[][] matrixB;
14      private ComplexNumber[][] matrixSum;
15      private AtomicInteger arrived;
16
17      public LockAndData(AtomicInteger arrived) {
18          this.arrived = arrived;
19      }
20  }

```

*Atomic integer is much better between threads in java.

3.2.3 Common Thread Function between Threads

```

7  public class SyncThreadFunction implements Runnable{
8      private final LockAndData lockData;
9      private Coordinates coordinates;
10
11      public SyncThreadFunction(LockAndData lockData, Coordinates coordinates) {
12          this.lockData = lockData;
13          this.coordinates = coordinates;
14      }
15
16      @Override
17      public void run() {
18          System.out.println("Task1 -> XStart: " + coordinates.getXLow() + " YStart: "+
19              synchronized (lockData){
20                  try{
21                      lockData.getArrived().getAndIncrement(); // ++arrived
22                      if(lockData.getArrived().get() < 4){
23                          lockData.wait(); // cwait(c,m)
24                      }
25                      else{
26                          lockData.notifyAll(); // broadcast(c)
27                      }
28                  } catch (InterruptedException e) {
29                      e.printStackTrace();
30                  }
31              }
32          System.out.println("Task2 -> XStart: " + coordinates.getXLow() + " YStart: "+
33      }
34  }

```

*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

```
14 public class TestDrive {
15     public static void main(String[] args) {
16         // create thread shared data number of arrived
17         AtomicInteger arrivedCount = new AtomicInteger(0);
18
19         //set common data and lock object
20         LockAndData data = new LockAndData(arrivedCount);
21
22         //create threads and inject shared data and its responsible coordinates in matrix
23         Thread thread0 = new Thread(new SyncThreadFunction(data, new Coordinates(0, 4096, 0)));
24         Thread thread1 = new Thread(new SyncThreadFunction(data, new Coordinates(0, 4096, 4096)));
25         Thread thread2 = new Thread(new SyncThreadFunction(data, new Coordinates(4096, 8192, 0)));
26         Thread thread3 = new Thread(new SyncThreadFunction(data, new Coordinates(4096, 8192, 4096)));
27
28         //start threads
29         thread0.start();
30         thread1.start();
31         thread2.start();
32         thread3.start();
33
34         try {
35             /*Make sure all threads have finished.*/
36             thread0.join();
37             thread1.join();
38             thread2.join();
39             thread3.join();
40         } catch (Exception e) {
41             e.printStackTrace();
42         }
43         System.out.println("All threads are finished. Good Bye...");
44     }
45 }
46 }
```

3.3.5 Output

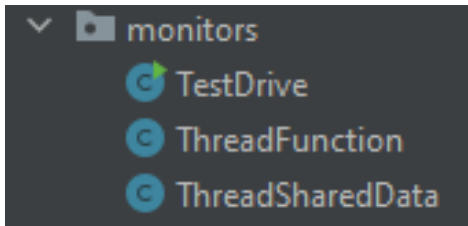
```
Task1 -> XStart: 4096 YStart: 0
Task1 -> XStart: 4096 YStart: 4096
Task1 -> XStart: 0 YStart: 0
Task1 -> XStart: 0 YStart: 4096
Task2 -> XStart: 0 YStart: 4096
Task2 -> XStart: 4096 YStart: 0
Task2 -> XStart: 0 YStart: 0
Task2 -> XStart: 4096 YStart: 4096
All threads are finished. Good Bye...

Process finished with exit code 0
```

*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

```
5 import java.util.concurrent.atomic.AtomicInteger;
6 import java.util.concurrent.locks.Condition;
7 import java.util.concurrent.locks.ReentrantLock;
8
9 public class ThreadSharedData {
10     //private ComplexNumber[][] matrixA;
11     //private ComplexNumber[][] matrixB;
12     //private ComplexNumber[][] matrixSum;
13     private AtomicInteger arrived;
14     private ReentrantLock mutex;
15     private Condition cond;
16
17     public ThreadSharedData(AtomicInteger arrived, ReentrantLock mutex, Condition cond) {
18         this.arrived = arrived;
19         this.mutex = mutex;
20         this.cond = cond;
21     }
22 }
```

*Atomic integer is much better between threads in java.

3.3.3 Common Thread Function between Threads

```
6 public class ThreadFunction implements Runnable{
7     private ThreadSharedData data;
8     private Coordinates coordinates;
9
10    public ThreadFunction(ThreadSharedData data ,Coordinates coordinates) {
11        this.data = data;
12        this.coordinates = coordinates;
13    }
14
15    @Override
16    public void run() {
17        System.out.println("Task1 -> XStart: " + coordinates.getXLow() + " YStart: " + coordinates.getYLow());
18        data.getMutex().lock(); // lock(m)
19        try{
20            data.getArrived().getAndIncrement(); // ++arrived
21            if(data.getArrived().get() < 4){
22                data.getCond().await(); // cwait(c,m)
23            }
24            else{
25                data.getCond().signalAll(); // broadcast(c)
26            }
27        } catch (InterruptedException e) {
28            e.printStackTrace();
29        } finally {
30            data.getMutex().unlock(); // unlock(m)
31        }
32        System.out.println("Task2 -> XStart: " + coordinates.getXLow() + " YStart: " + coordinates.getYLow());
33    }
34 }
35 }
```

3.3.4 Creating Threads and Testing

```

12 public class TestDrive {
13     public static void main(String[] args) throws IOException, InterruptedException {
14         // create thread shared data
15         ReentrantLock mutex = new ReentrantLock();
16         Condition cond = mutex.newCondition();
17         AtomicInteger arrivedCount = new AtomicInteger(0);
18
19         //set common data
20         ThreadSharedData data = new ThreadSharedData(arrivedCount, mutex, cond);
21
22         //create threads and inject shared data and its responsible coordinates in matrix
23         Thread thread0 = new Thread(new ThreadFunction(data, new Coordinates(0, 4096, 0, 4096));
24         Thread thread1 = new Thread(new ThreadFunction(data, new Coordinates(0, 4096, 4096, 4096));
25         Thread thread2 = new Thread(new ThreadFunction(data, new Coordinates(4096, 8192, 0, 4096));
26         Thread thread3 = new Thread(new ThreadFunction(data, new Coordinates(4096, 8192, 4096, 4096));
27
28         //start threads
29         thread0.start();
30         thread1.start();
31         thread2.start();
32         thread3.start();
33
34         try {
35             /*Make sure all threads have finished.*/
36             thread0.join();
37             thread1.join();
38             thread2.join();
39             thread3.join();
40         } catch (Exception e) {
41             e.printStackTrace();
42         }
43         System.out.println("All threads are finished. Good Bye...");
44     }
45 }

```

3.3.5 Output

```

Task1 -> XStart: 4096 YStart: 4096
Task1 -> XStart: 0 YStart: 4096
Task1 -> XStart: 0 YStart: 0
Task1 -> XStart: 4096 YStart: 0
Task2 -> XStart: 4096 YStart: 0
Task2 -> XStart: 4096 YStart: 4096
Task2 -> XStart: 0 YStart: 4096
Task2 -> XStart: 0 YStart: 0
All threads are finished. Good Bye...

Process finished with exit code 0

```

*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

```
for (int i = coordinates.getxLow(); i < coordinates.getxUp() ; i++) {
    for (int j = coordinates.getyLow(); j < coordinates.getyUp() ; j++) {
        lockData.setSumByIndex(i,j, Helper.addNumbers(lockData.getAByIndex(i,j),lockData.getBByIndex(i,j)));
    }
}
```

3.4.2 Discrete Fourier Transform Formula and Implementation

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn}$$

$$= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N}kn\right) - i \cdot \sin\left(\frac{2\pi}{N}kn\right) \right], \quad (\text{Eq.1})$$

Implementation

```
37     int n = (coordinates.getxUp() - coordinates.getxLow());
38     int k = 0;
39     for (int i = coordinates.getxLow(); i < coordinates.getxUp() ; i++) {
40         double sumReal = 0;
41         double sumImag = 0;
42         for (int j = coordinates.getyLow(); j < coordinates.getyUp() ; j++) {
43             double angle = (2 * Math.PI * k * coordinates.getPortion()) / n;
44             sumReal += lockData.getSumByIndex(i,j).getReal() * Math.cos(angle) +
45                 lockData.getSumByIndex(i,j).getImg() * Math.sin(angle);
46             sumImag += -1*lockData.getSumByIndex(i,j).getReal() * Math.sin(angle) +
47                 lockData.getSumByIndex(i,j).getImg() * Math.cos(angle);
48         }
49         lockData.setResByIndex(coordinates.getPortion(),k,new ComplexNumber((int)sumReal,
50             (int)sumImag));
51         k++;
52     }
```

*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

```
Threads are starting...  
Time Taken in java synchronized: 1632 ms  
All threads are finished. Good Bye...  
  
Process finished with exit code 0
```

Java monitor result

```
Threads are starting...  
Time Taken in java monitor: 1601 ms  
All threads are finished. Good Bye...  
  
Process finished with exit code 0
```

Using a single thread result

```
Calculating is starting...  
Time Taken in single thread: 3340 ms  
Single thread is finished. Good Bye...  
  
Process finished with exit code 0
```

*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

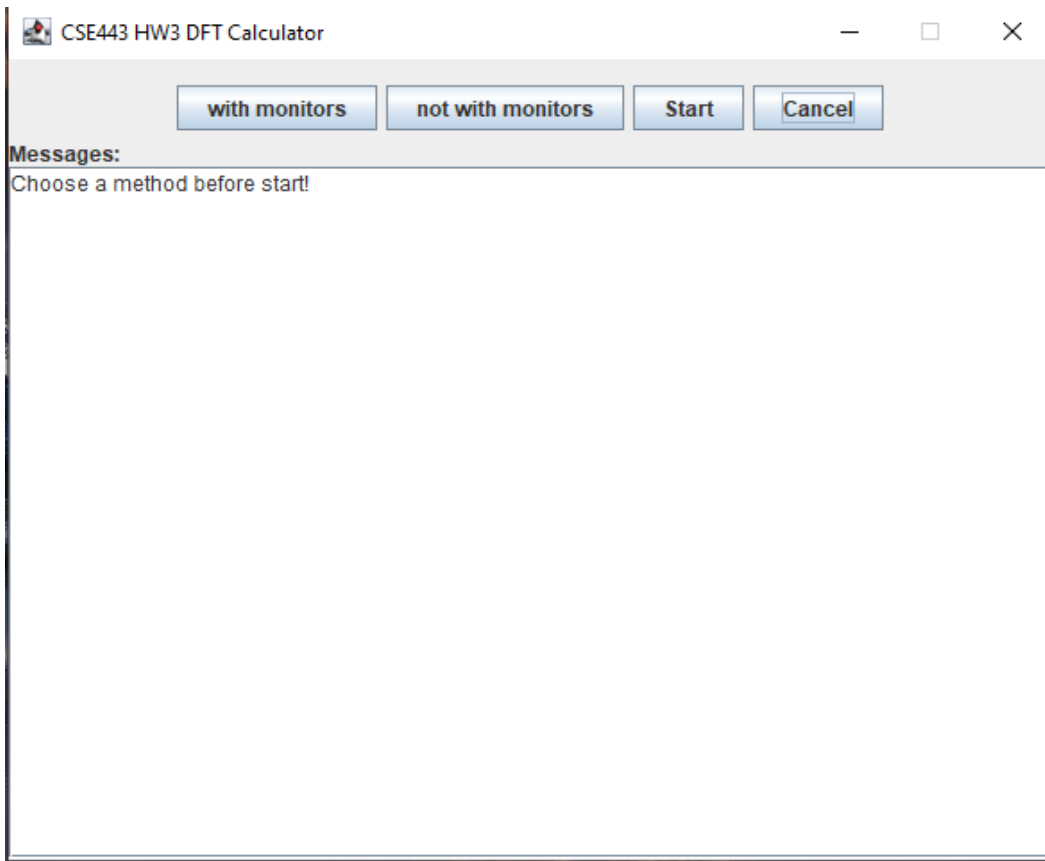
In java Swing GUI, **if we use these implementations directly** our gui **will not be respond** until calculation is done.

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



*with monitors = multi thread

* not with monitor = single thread

3.5.2 Making it responsive while calculating

First See start button action listener code

```
btnNewButton_2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (type == 0) {
            JOptionPane.showMessageDialog(null, "Choose a method!"
                , "Warning", JOptionPane.INFORMATION_MESSAGE);
        } else {
            calculator = new SwingWorker<Long, Void>() {
                @Override
                public Long doInBackground() {
                    isWorking = true;

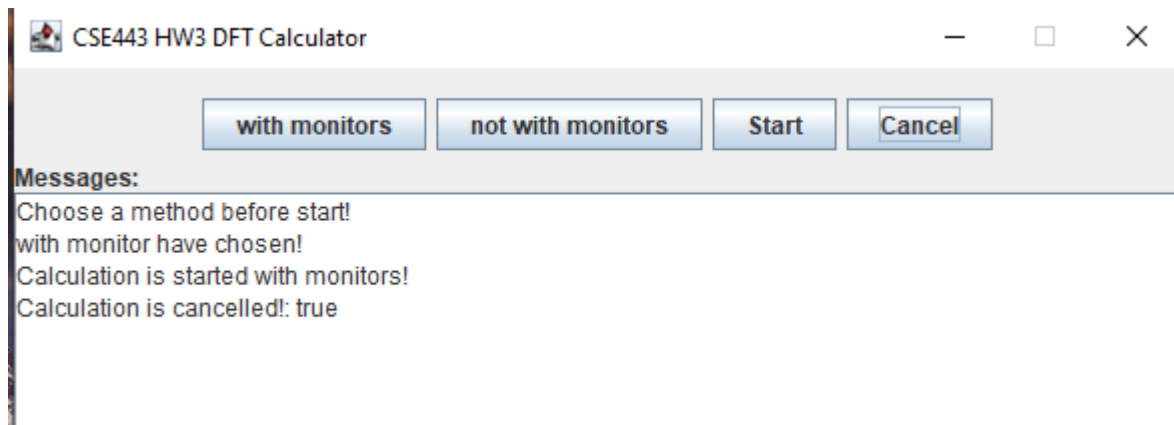
                    //logArea.append(String.valueOf(type));
                    if (type == 1) {
                        logArea.append("\nCalculation is started with monitors!");
                        Long start = System.currentTimeMillis();
                        TestDrive.main(null);
                        return System.currentTimeMillis() - start;
                    } else {
                        logArea.append("\nCalculation is started with single thread!");
                        Long start = System.currentTimeMillis();
                        SingleThread.main(null);
                        return System.currentTimeMillis() - start;
                    }
                }
            };
        }
    }
});
```

Canceling operation

To cancel current operation, SwingWorker cancel method is used

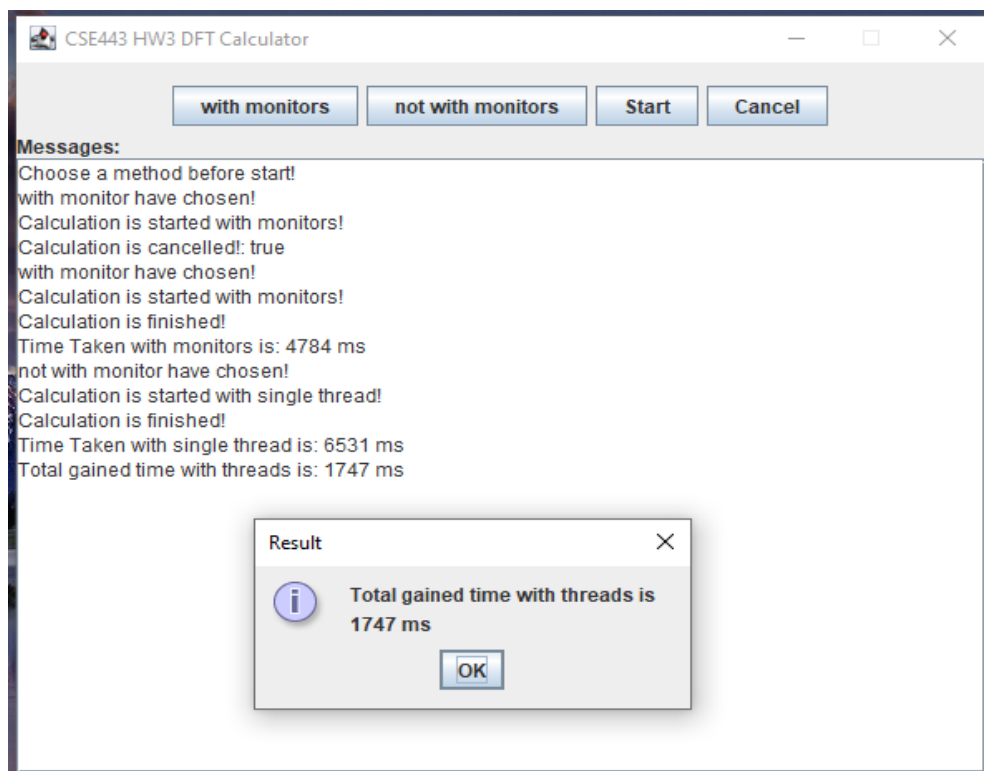
```
if (isWorking) {
    logArea.append("\nCalculation is cancelled!: " + calculator.cancel(false));
} else {
    JOptionPane.showMessageDialog(null, "Nothing is working to cancel!"
        , "Warning", JOptionPane.INFORMATION_MESSAGE);
}
```

Output:



3.5.3 See the total time gained by using multiple threads

Output:



*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

ThreadSharedData	
matrixA	ComplexNumber[]
matrixB	ComplexNumber[]
matrixSum	ComplexNumber[]
dftResult	ComplexNumber[]
arrived	AtomicInteger
mutex	ReentrantLock
cond	Condition
ThreadSharedData(AtomicInteger, ReentrantLock, Condition)	
getAByIndex(int, int)	ComplexNumber
getArrived()	AtomicInteger
getBByIndex(int, int)	ComplexNumber
getCond()	Condition
getDftResult()	ComplexNumber[]
getMatrixA()	ComplexNumber[]
getMatrixB()	ComplexNumber[]
getMatrixSum()	ComplexNumber[]
getMutex()	ReentrantLock
getSumByIndex(int, int)	ComplexNumber
setArrived(AtomicInteger)	void
setCond(Condition)	void
setDftResult(ComplexNumber[])	void
setMatrixA(ComplexNumber[])	void
setMatrixB(ComplexNumber[])	void
setMatrixSum(ComplexNumber[])	void
setMutex(ReentrantLock)	void
setResByIndex(int, int, ComplexNumber)	void
setSumByIndex(int, int, ComplexNumber)	void

LockAndData	
matrixA	ComplexNumber[]
matrixB	ComplexNumber[]
matrixSum	ComplexNumber[]
dftResult	ComplexNumber[]
arrived	AtomicInteger
LockAndData(AtomicInteger)	
getAByIndex(int, int)	ComplexNumber
getArrived()	AtomicInteger
getBByIndex(int, int)	ComplexNumber
getDftResult()	ComplexNumber[]
getMatrixA()	ComplexNumber[]
getMatrixB()	ComplexNumber[]
getMatrixSum()	ComplexNumber[]
getResByIndex(int, int)	ComplexNumber
getSumByIndex(int, int)	ComplexNumber
setArrived(AtomicInteger)	void
setDftResult(ComplexNumber[])	void
setMatrixA(ComplexNumber[])	void
setMatrixB(ComplexNumber[])	void
setMatrixSum(ComplexNumber[])	void
setResByIndex(int, int, ComplexNumber)	void
setSumByIndex(int, int, ComplexNumber)	void

Coordinates	
portion	int
xLow	int
xUp	int
yLow	int
yUp	int
Coordinates(int, int, int, int, int)	
getPortion()	int
getxLow()	int
getxUp()	int
getyLow()	int
getyUp()	int
setPortion(int)	void
setxLow(int)	void
setxUp(int)	void
setyLow(int)	void
setyUp(int)	void

MainWindow	
calculator	SwingWorker<Long, Void>
times	Stack<TakenTime>
contentPane1	JPanel
contentPane2	JPanel
logArea	JTextArea
chatScroll	JScrollPane
chatPanel	JPanel
btnNewButton_2	JButton
btnNewButton_3	JButton
type	int
isWorking	boolean
MainWindow()	
main(String[])	void

ComplexNumber	
real	int
img	int
ComplexNumber(int, int)	
getImg()	int
getNumber()	String
getReal()	int
setImg(int)	void
setReal(int)	void
toString()	String

TakenTime	
time	Long
type	int
TakenTime(int, Long)	
getDiffer(TakenTime, TakenTime)	Long
getTime()	Long
getType()	int
setTime(Long)	void
setType(int)	void

ThreadFunction	
data	ThreadSharedData
coordinates	Coordinates
ThreadFunction(ThreadSharedData, Coordinates)	
run()	void

SyncThreadFunction	
lockData	LockAndData
coordinates	Coordinates
SyncThreadFunction(LockAndData, Coordinates)	
run()	void

Helper	
Helper()	
addNumbers(ComplexNumber, ComplexNumber)	ComplexNumber
createRandomMatrix()	ComplexNumber[]

TestDrive	
TestDrive()	
main(String[])	void

SyncTestDrive	
SyncTestDrive()	
main(String[])	void

SingleThread	
SingleThread()	
main(String[])	void

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.