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

Akif KARTAL
171044098

Question 1

Use Adapter Design pattern.

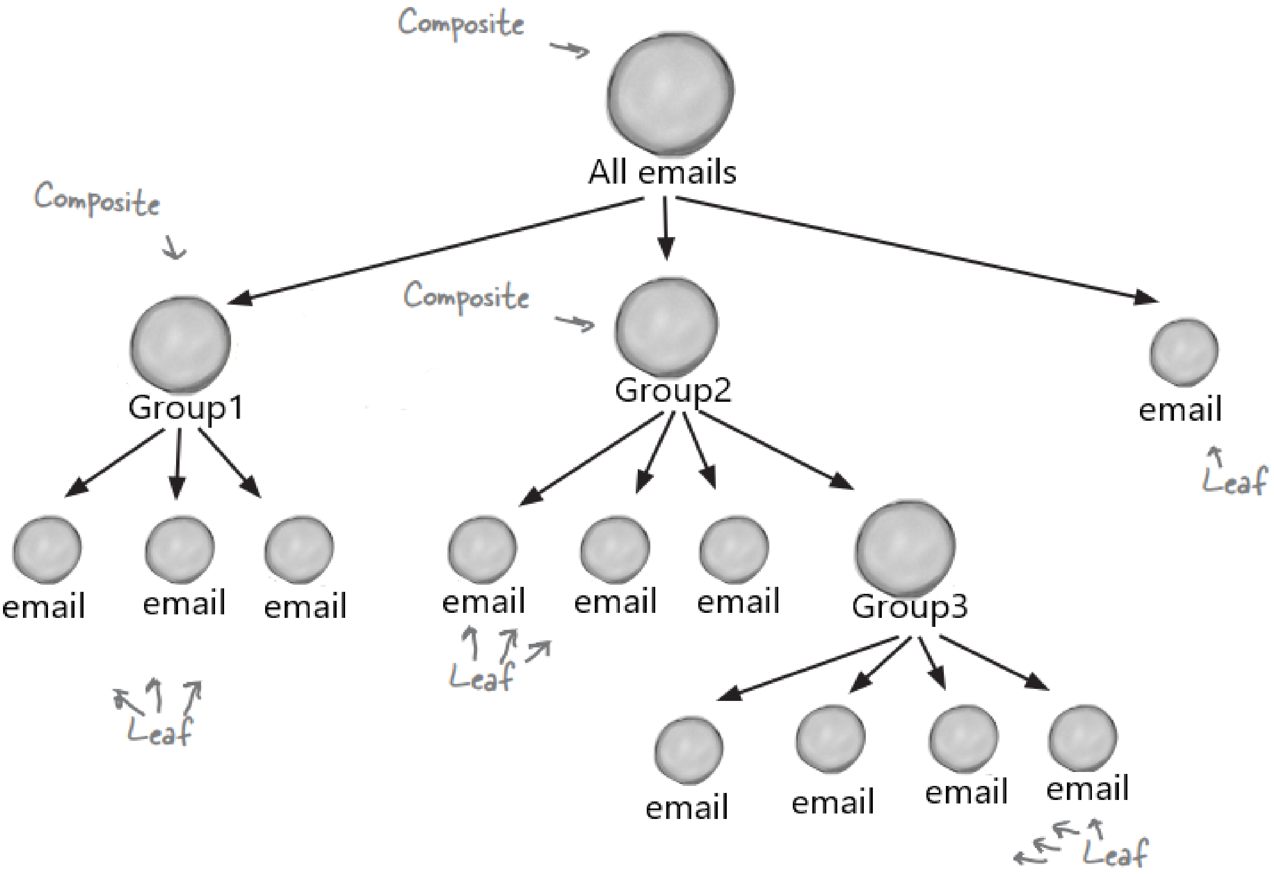
Implement a simple lineer data structure which will be a linked list and then Using adapter pattern convert that class to a thread safe class.

Question 2 – Composite and Iterator

Solution

2.1 Understanding the problem

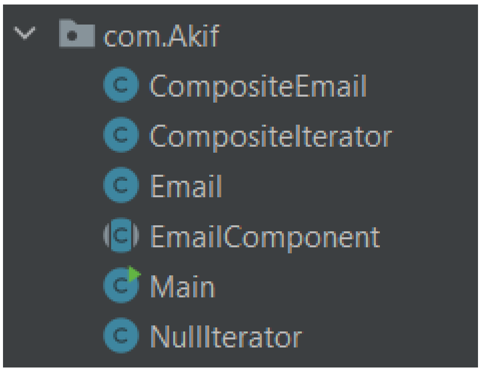
The problem is like the following picture;



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**.

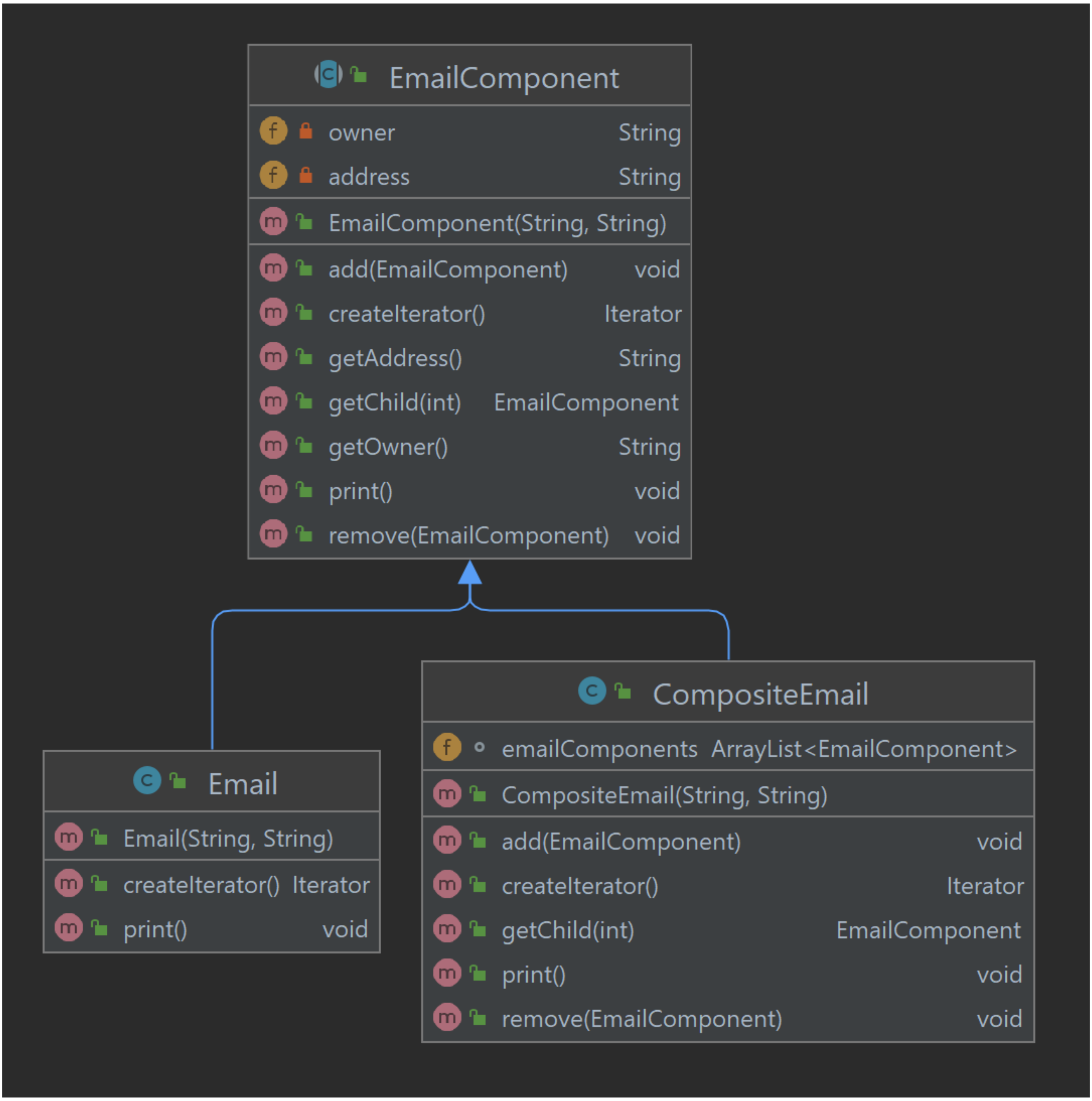
2.2 Class Diagram

2.2.1 Classes in my solution



com.Akif
- CompositeEmail
- CompositeIterator
- Email
- EmailComponent
- Main
- NullIterator

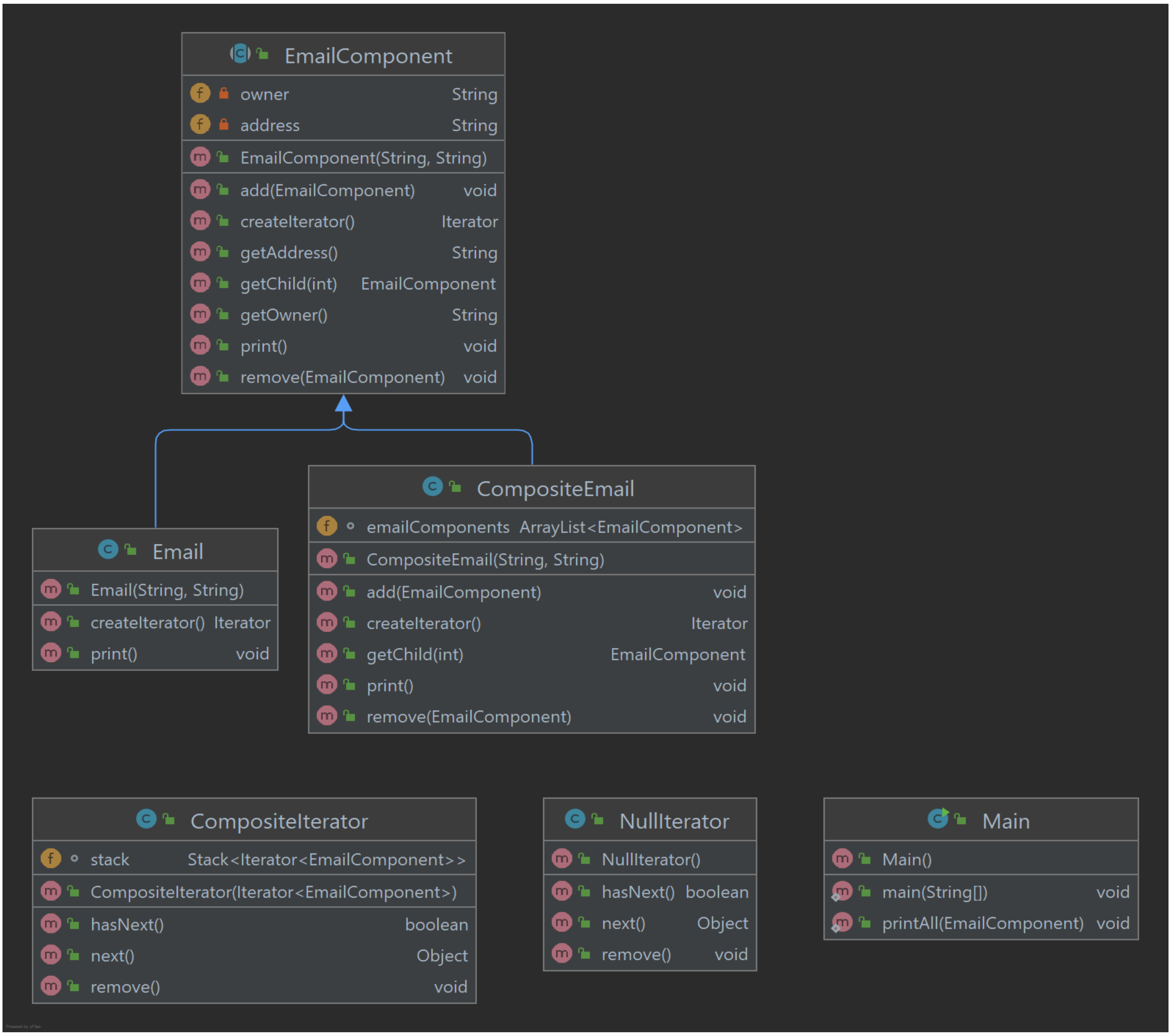
2.2.2 Composite Pattern Class Diagram


EmailComponent
owner String
address String
EmailComponent(String, String)
add(EmailComponent) void
createIterator() Iterator
getAddress() String
getChild(int) EmailComponent
getOwner() String
print() void
remove(EmailComponent) void
Email
Email(String, String)
createIterator() Iterator
print() void
CompositeEmail
emailComponents ArrayList<EmailComponent>
CompositeEmail(String, String)
add(EmailComponent) void
createIterator() Iterator
getChild(int) EmailComponent
print() void
remove(EmailComponent) void

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

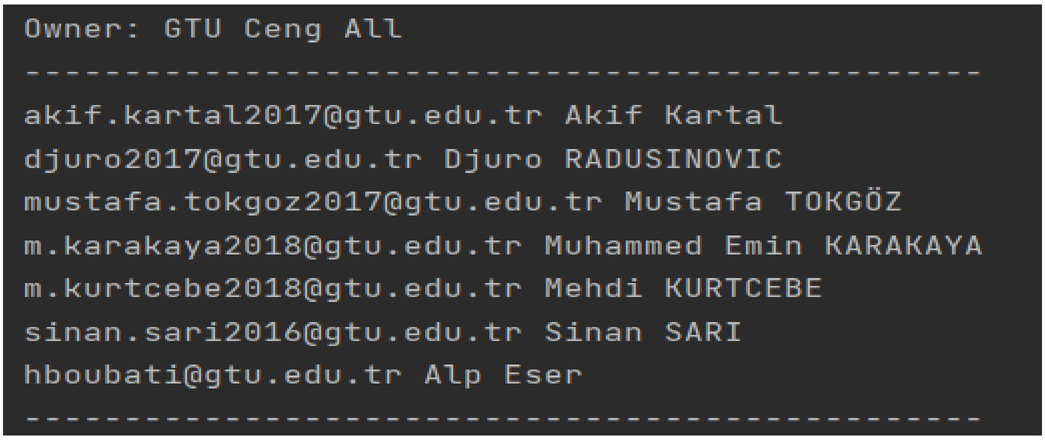
Here, we will see full diagram note that I used iterator design pattern to traverse all tree easily as expected in homework pdf file.


EmailComponent
owner String
address String
EmailComponent(String, String)
add(EmailComponent) void
createIterator() Iterator
getAddress() String
getChild(int) EmailComponent
getOwner() String
print() void
remove(EmailComponent) void
CompositeEmail
emailComponents ArrayList<EmailComponent>
CompositeEmail(String, String)
add(EmailComponent) void
createIterator() Iterator
getChild(int) EmailComponent
print() void
remove(EmailComponent) void
Email
Email(String, String)
createIterator() Iterator
print() void
CompositeIterator
stack Stack<Iterator<EmailComponent>>
CompositeIterator(Iterator<EmailComponent>)
hasNext() boolean
next() Object
remove() void
NullIterator
NullIterator()
hasNext() boolean
next() Object
remove() void
Main
Main()
main(String[]) void
printAll(EmailComponent) void

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;



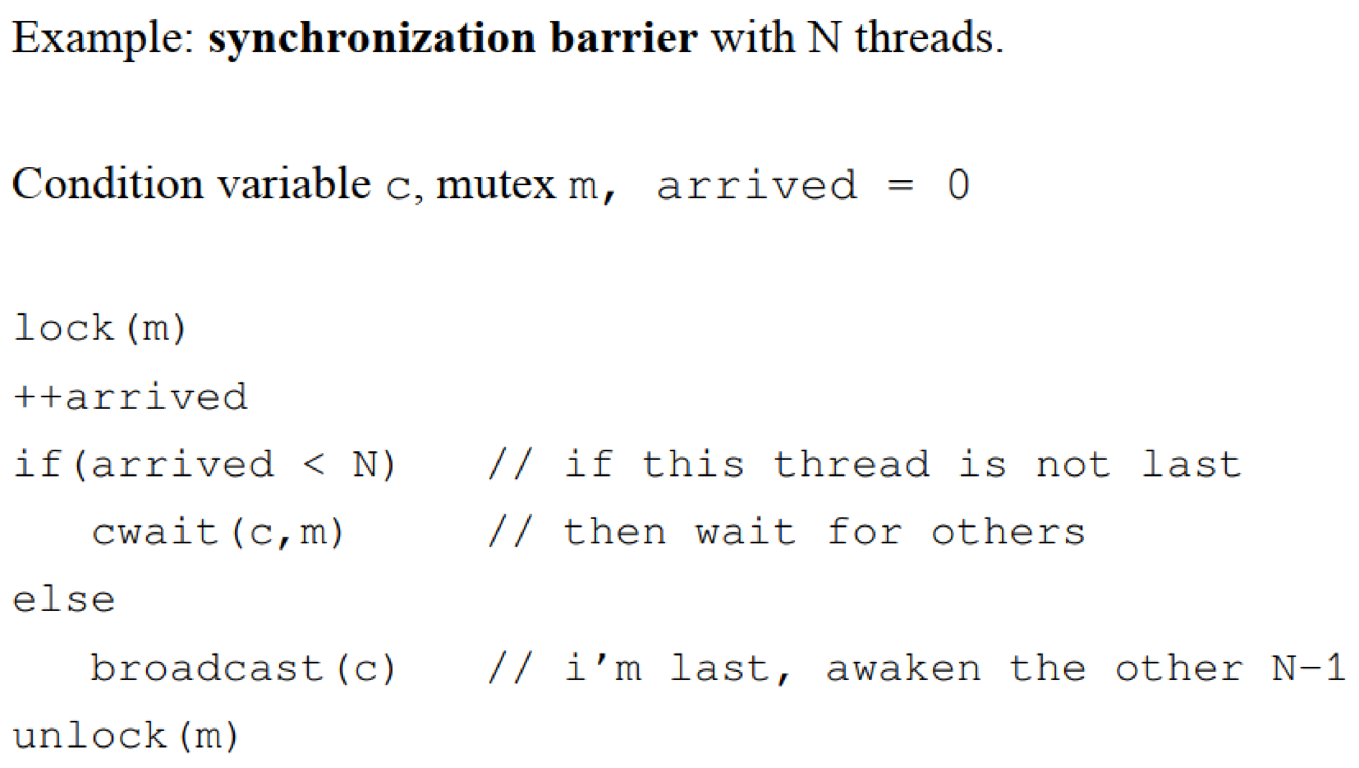
```
Owner: GTU Ceng All
------------------------------------------------
akif.kartal2017@gtu.edu.tr Akif Kartal
djuro2017@gtu.edu.tr Djuro RADUSINOVIC
mustafa.tokgoz2017@gtu.edu.tr Mustafa TOKGÖZ
m.karakaya2018@gtu.edu.tr Muhammed Emin KARAKAYA
m.kurtcebe2018@gtu.edu.tr Mehdi KURTCEBE
sinan.sari2016@gtu.edu.tr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

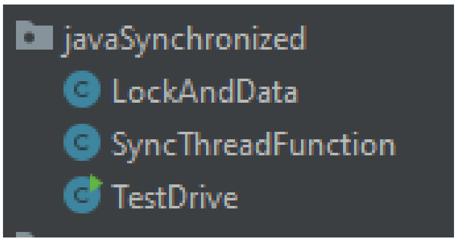
*This solution was taken from CSE344 System Programming Course slides.

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.

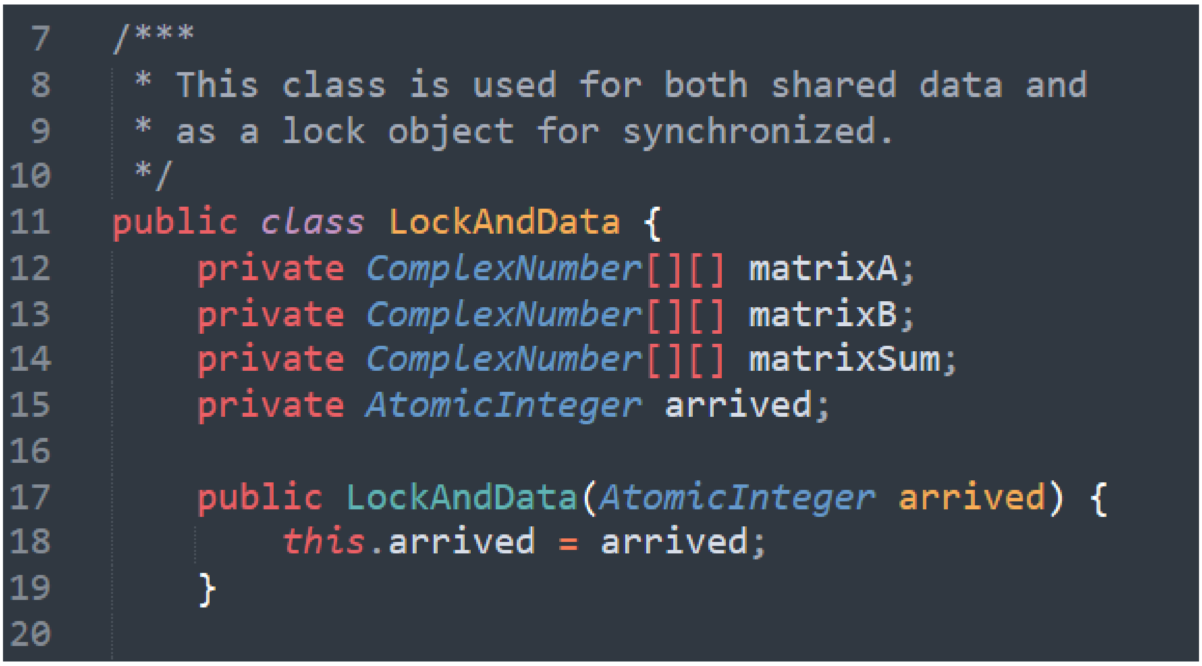
3.2 Using Java's synchronized

3.2.1 Classes in my solution



javaSynchronized
LockAndData
SyncThreadFunction
TestDrive

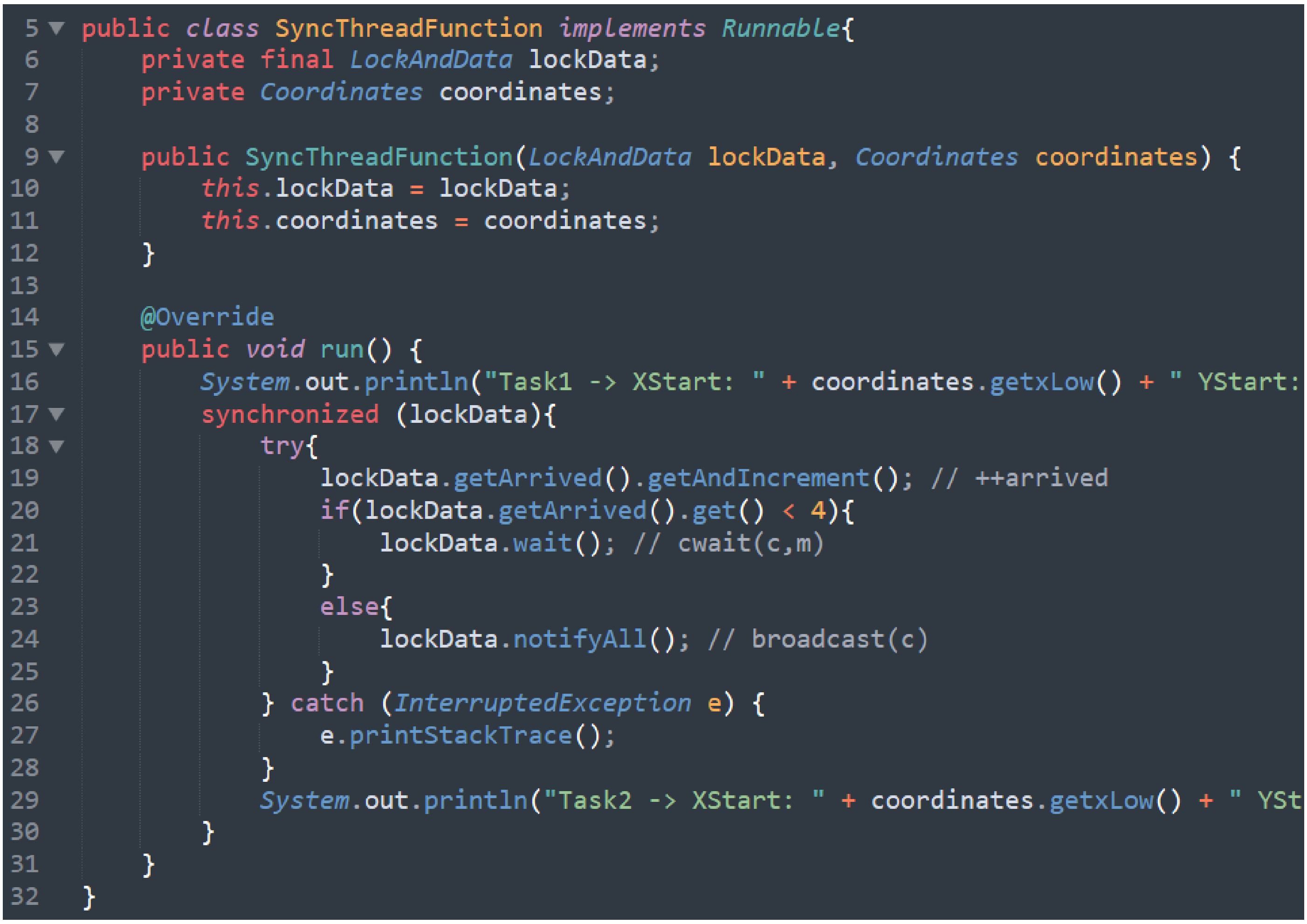
3.2.2 Shared Data and Lock Object Class



```java
/***
 * This class is used for both shared data and
 * as a lock object for synchronized.
 */
public class LockAndData {
    private ComplexNumber[][] matrixA;
    private ComplexNumber[][] matrixB;
    private ComplexNumber[][] matrixSum;
    private AtomicInteger arrived;

    public LockAndData(AtomicInteger arrived) {
        this.arrived = arrived;
    }

```

*Atomic integer is much better between threads in java.

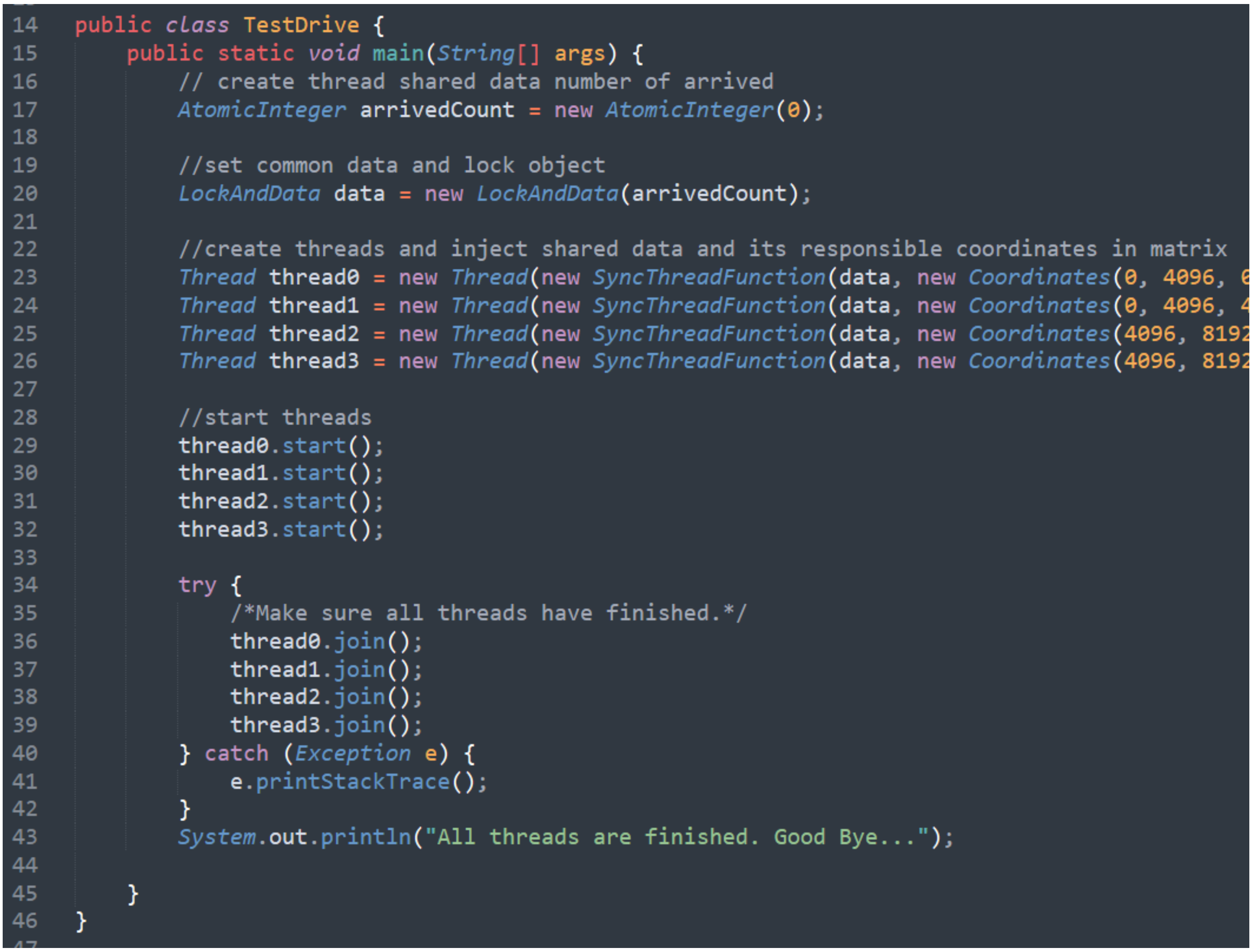
3.2.3 Common Thread Function between Threads



```java
public class SyncThreadFunction implements Runnable{
    private final LockAndData lockData;
    private Coordinates coordinates;

    public SyncThreadFunction(LockAndData lockData, Coordinates coordinates) {
        this.lockData = lockData;
        this.coordinates = coordinates;
    }

    @Override
    public void run() {
        System.out.println("Task1 -> XStart: " + coordinates.getxLow() + " YStart:
        synchronized (lockData){
            try{
                lockData.getArrived().getAndIncrement(); // ++arrived
                if(lockData.getArrived().get() < 4){
                    lockData.wait(); // cwait(c,m)
                }
                else{
                    lockData.notifyAll(); // broadcast(c)
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Task2 -> XStart: " + coordinates.getxLow() + " YSt
        }
    }
}
```

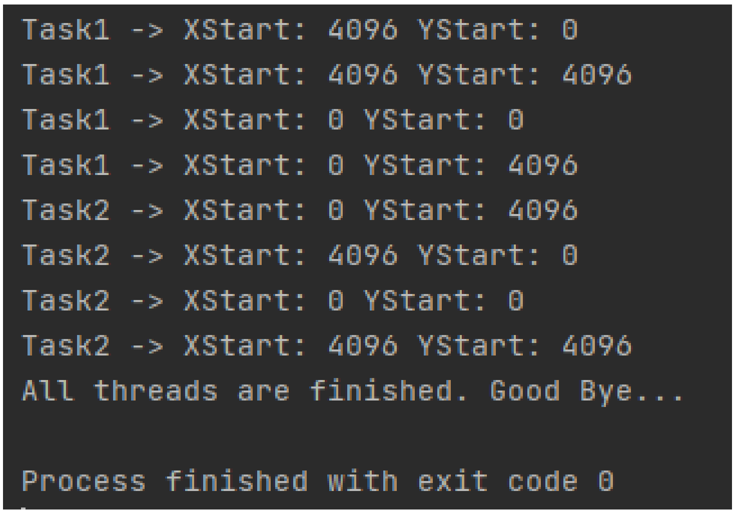
*Here, synchorized keyword acts like a mutex and it locks and unlock code in its scope. We are using lockData object like a mutex(with the help of object class) since it is common and shared between all threads.

3.2.4 Creating Threads and Testing



```
public class TestDrive {
    public static void main(String[] args) {
        // create thread shared data number of arrived
        AtomicInteger arrivedCount = new AtomicInteger(0);

        //set common data and lock object
        LockAndData data = new LockAndData(arrivedCount);

        //create threads and inject shared data and its responsible coordinates in matrix
        Thread thread0 = new Thread(new SyncThreadFunction(data, new Coordinates(0, 4096, 0
        Thread thread1 = new Thread(new SyncThreadFunction(data, new Coordinates(0, 4096, 4
        Thread thread2 = new Thread(new SyncThreadFunction(data, new Coordinates(4096, 8192
        Thread thread3 = new Thread(new SyncThreadFunction(data, new Coordinates(4096, 8192

        //start threads
        thread0.start();
        thread1.start();
        thread2.start();
        thread3.start();

        try {
            /*Make sure all threads have finished.*/
            thread0.join();
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("All threads are finished. Good Bye...");

    }
}
```

3.3.5 Output

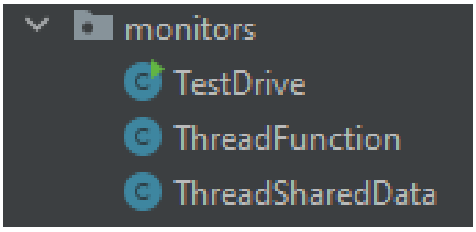


```
Task1 -> XStart: 4096 YStart: 0
Task1 -> XStart: 4096 YStart: 4096
Task1 -> XStart: 0 YStart: 0
Task1 -> XStart: 0 YStart: 4096
Task2 -> XStart: 0 YStart: 4096
Task2 -> XStart: 4096 YStart: 0
Task2 -> XStart: 0 YStart: 0
Task2 -> XStart: 4096 YStart: 4096
All threads are finished. Good Bye...

Process finished with exit code 0
```

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

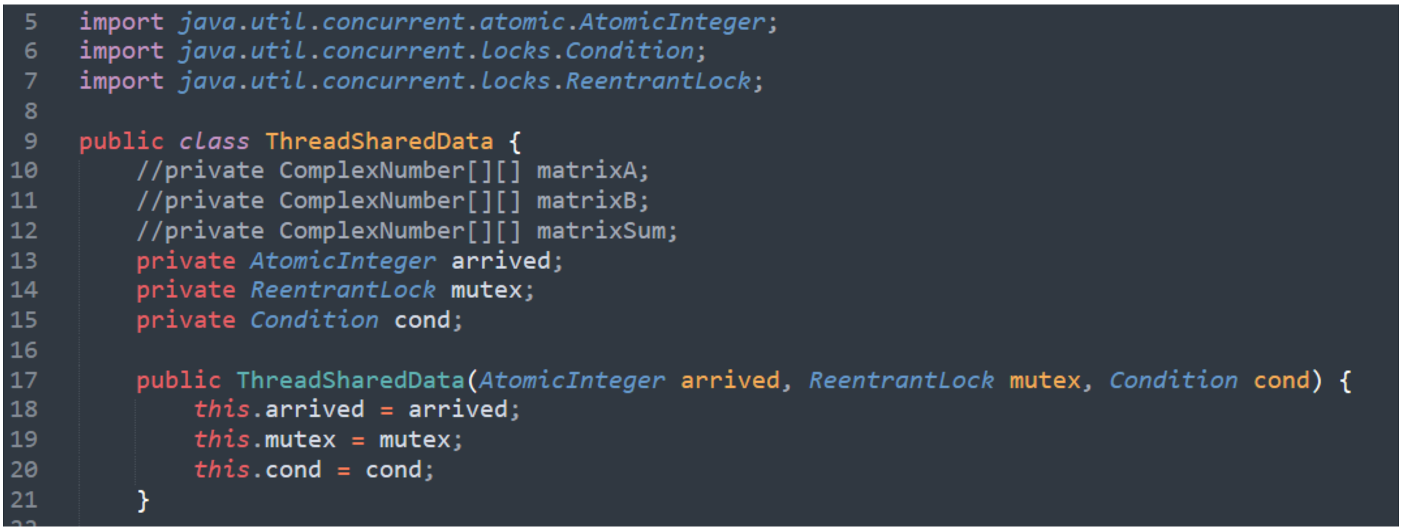
3.3 Using mutex(es) and monitor(s)

3.3.1 Classes in my solution



```
monitors
    TestDrive
    ThreadFunction
    ThreadSharedData
```

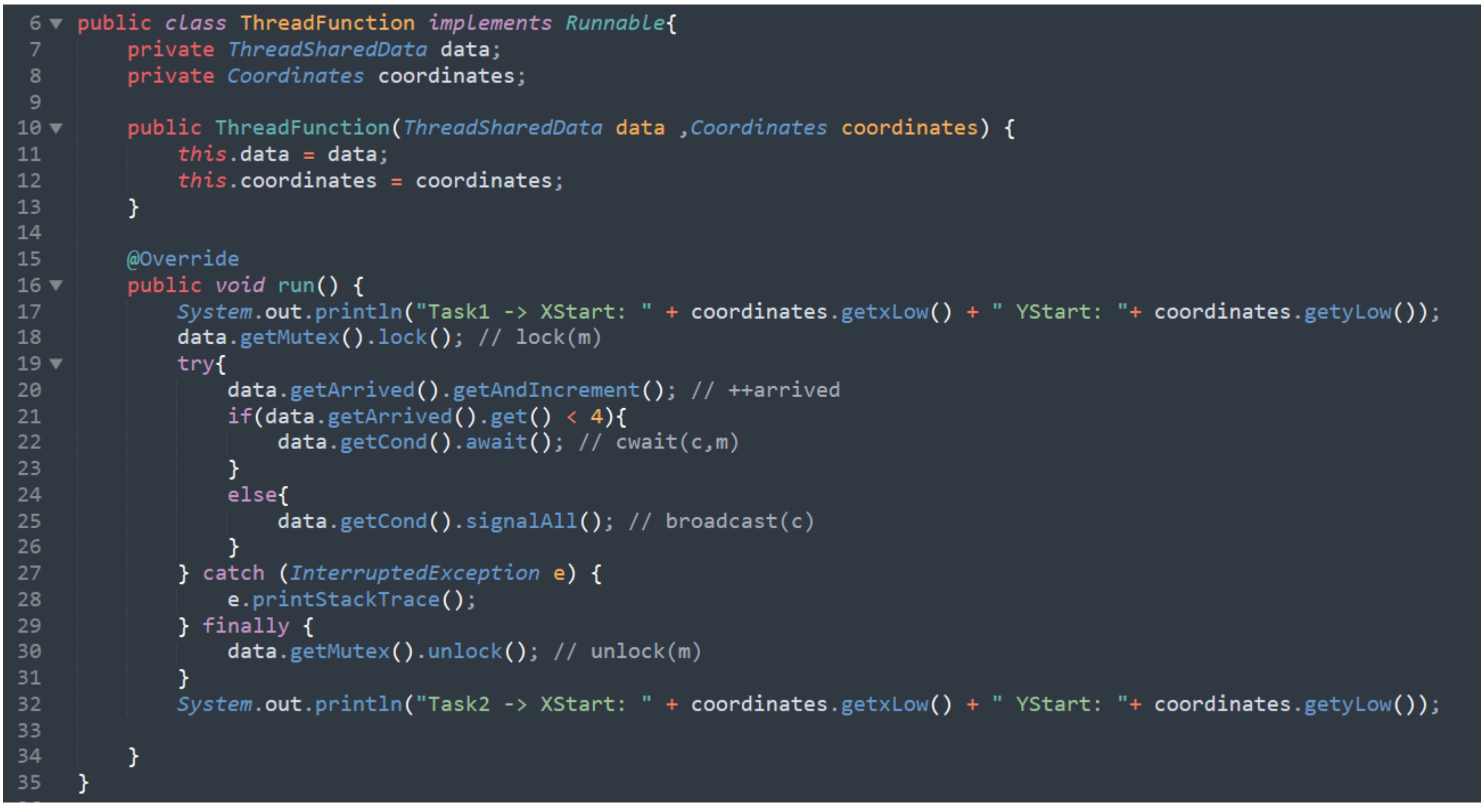
3.3.2 Shared Data between Threads



```java
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

public class ThreadSharedData {
    //private ComplexNumber[][] matrixA;
    //private ComplexNumber[][] matrixB;
    //private ComplexNumber[][] matrixSum;
    private AtomicInteger arrived;
    private ReentrantLock mutex;
    private Condition cond;

    public ThreadSharedData(AtomicInteger arrived, ReentrantLock mutex, Condition cond) {
        this.arrived = arrived;
        this.mutex = mutex;
        this.cond = cond;
    }
```

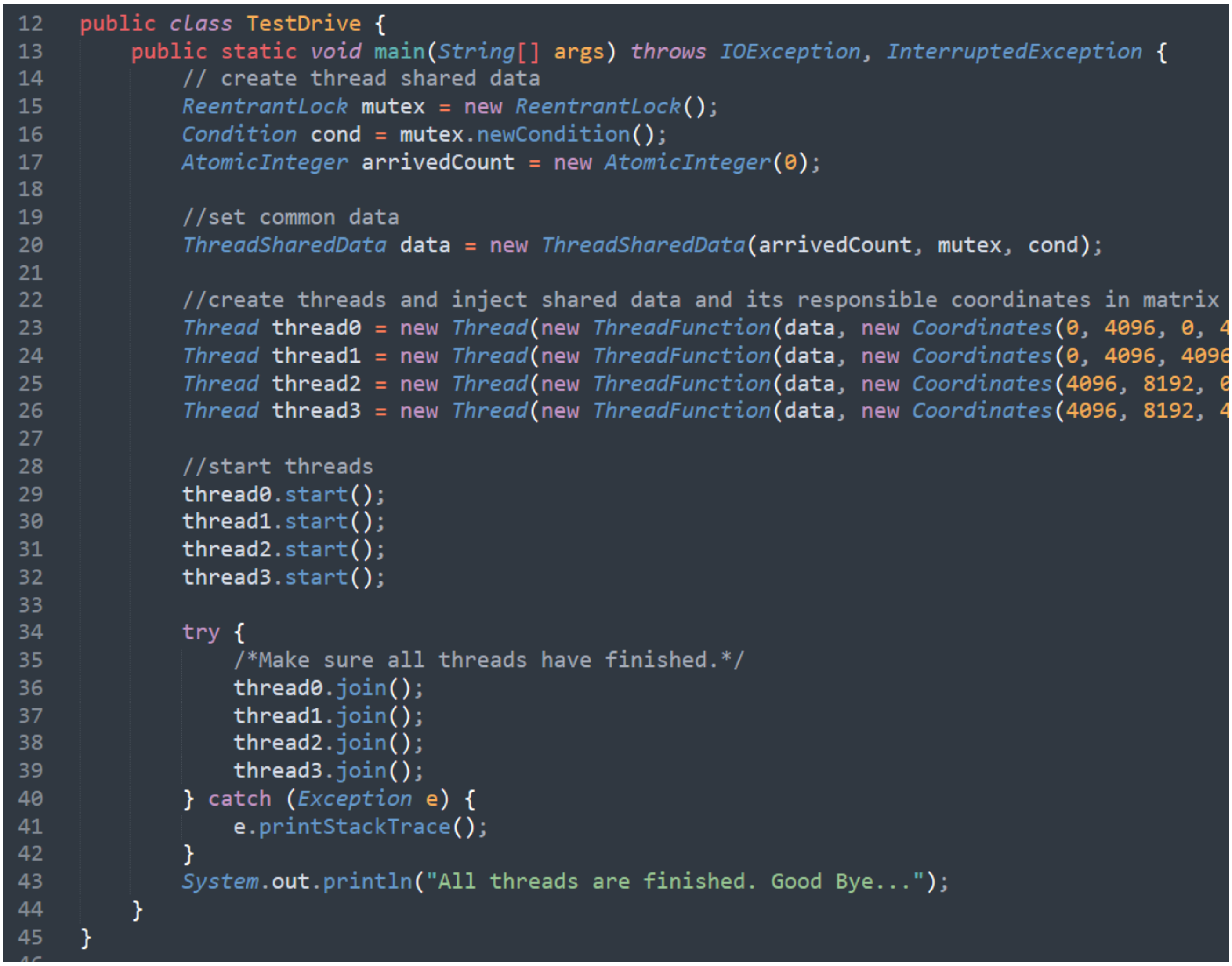
*Atomic integer is much better between threads in java.

3.3.3 Common Thread Function between Threads



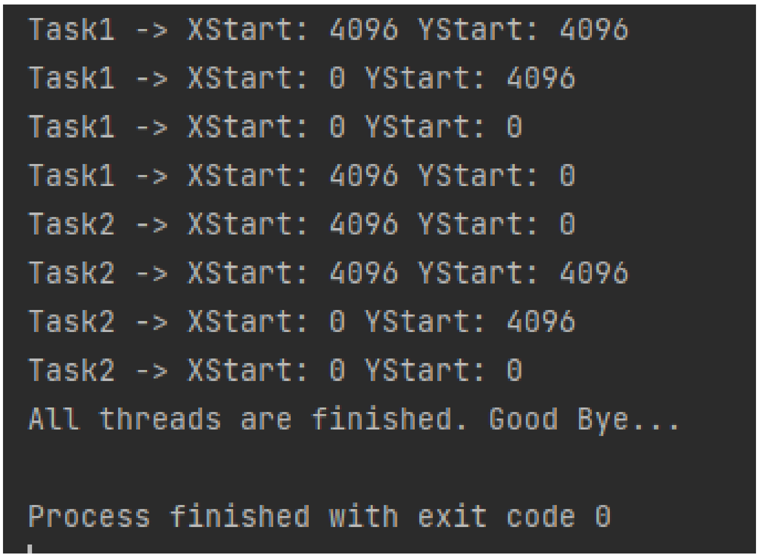
```java
public class ThreadFunction implements Runnable{
    private ThreadSharedData data;
    private Coordinates coordinates;

    public ThreadFunction(ThreadSharedData data ,Coordinates coordinates) {
        this.data = data;
        this.coordinates = coordinates;
    }

    @Override
    public void run() {
        System.out.println("Task1 -> XStart: " + coordinates.getxLow() + " YStart: "+ coordinates.getyLow());
        data.getMutex().lock(); // lock(m)
        try{
            data.getArrived().getAndIncrement(); // ++arrived
            if(data.getArrived().get() < 4){
                data.getCond().await(); // cwait(c,m)
            }
            else{
                data.getCond().signalAll(); // broadcast(c)
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            data.getMutex().unlock(); // unlock(m)
        }
        System.out.println("Task2 -> XStart: " + coordinates.getxLow() + " YStart: "+ coordinates.getyLow());

    }
}
```

3.3.4 Creating Threads and Testing



```java
public class TestDrive {
    public static void main(String[] args) throws IOException, InterruptedException {
        // create thread shared data
        ReentrantLock mutex = new ReentrantLock();
        Condition cond = mutex.newCondition();
        AtomicInteger arrivedCount = new AtomicInteger(0);

        //set common data
        ThreadSharedData data = new ThreadSharedData(arrivedCount, mutex, cond);

        //create threads and inject shared data and its responsible coordinates in matrix
        Thread thread0 = new Thread(new ThreadFunction(data, new Coordinates(0, 4096, 0, 4
        Thread thread1 = new Thread(new ThreadFunction(data, new Coordinates(0, 4096, 4096
        Thread thread2 = new Thread(new ThreadFunction(data, new Coordinates(4096, 8192, 0
        Thread thread3 = new Thread(new ThreadFunction(data, new Coordinates(4096, 8192, 4

        //start threads
        thread0.start();
        thread1.start();
        thread2.start();
        thread3.start();

        try {
            /*Make sure all threads have finished.*/
            thread0.join();
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("All threads are finished. Good Bye...");
    }
}
```

3.3.5 Output

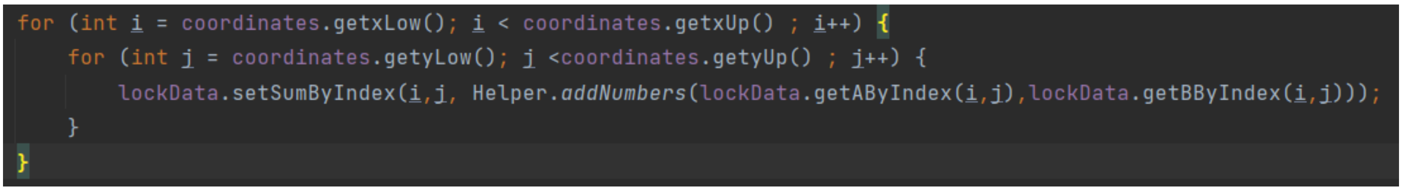


```
Task1 -> XStart: 4096 YStart: 4096
Task1 -> XStart: 0 YStart: 4096
Task1 -> XStart: 0 YStart: 0
Task1 -> XStart: 4096 YStart: 0
Task2 -> XStart: 4096 YStart: 0
Task2 -> XStart: 4096 YStart: 4096
Task2 -> XStart: 0 YStart: 4096
Task2 -> XStart: 0 YStart: 0
All threads are finished. Good Bye...

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

$$
\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \\
&= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N}kn\right) - i \cdot \sin\left(\frac{2\pi}{N}kn\right)\right],
\end{aligned}
\qquad \text{(Eq.1)}
$$