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

Akif KARTAL
171044098

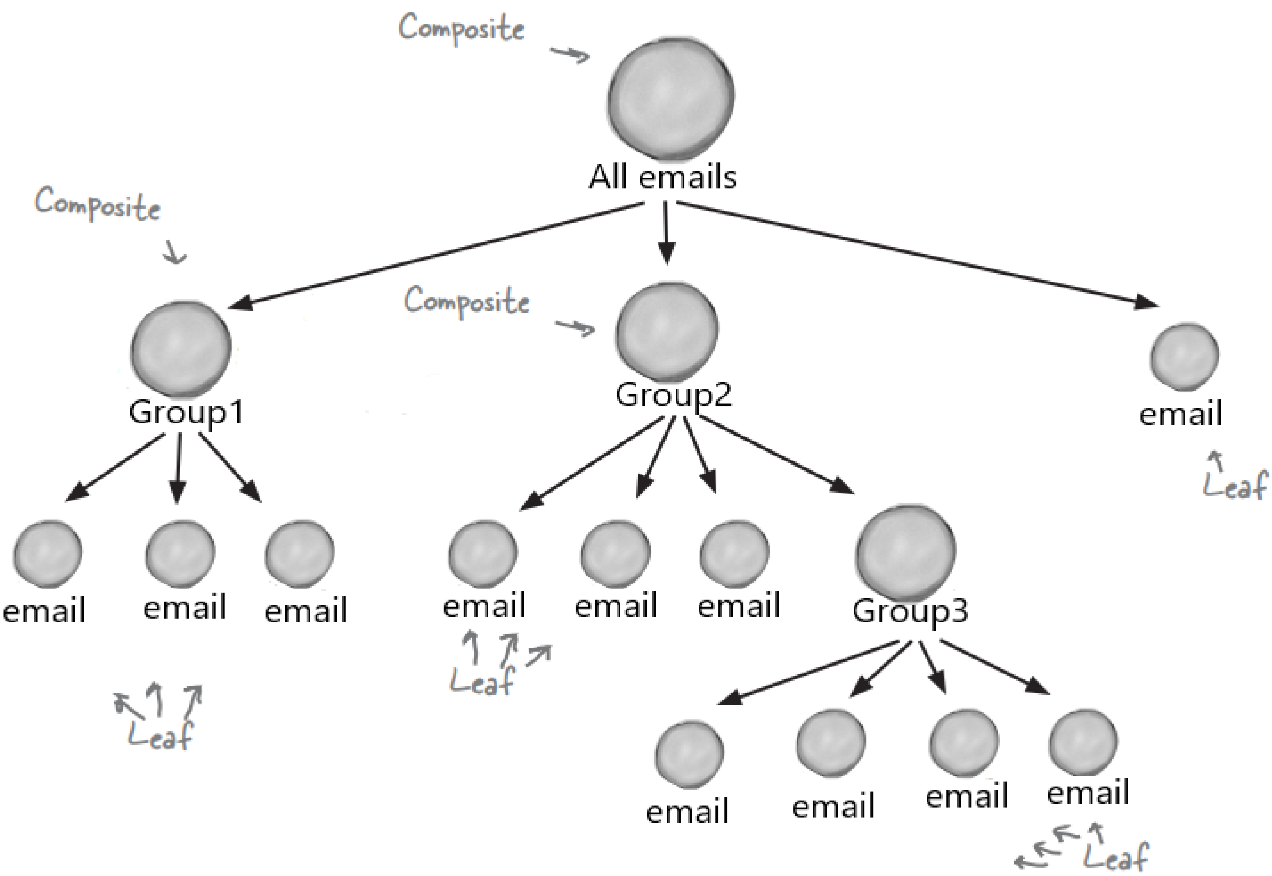
Question 1

Question 2 – Composite and Iterator

Solution

2.1 Understanding the problem

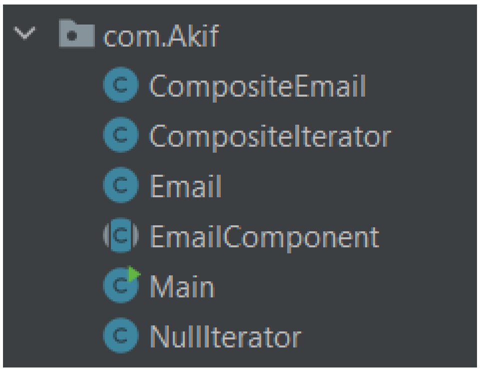
The problem is like the following picture;



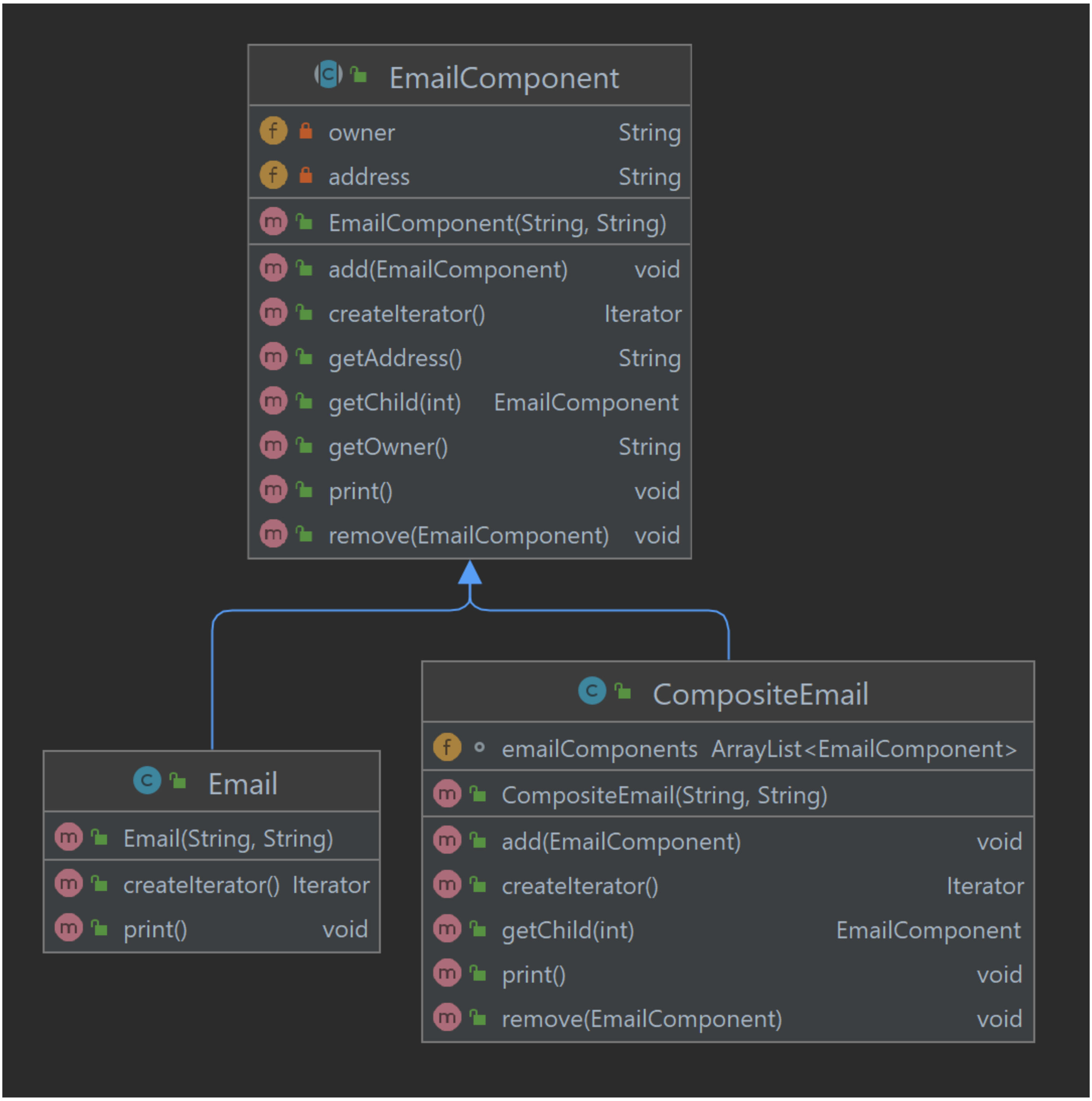
Here, emails contains both address and name of its owner, groups contains an arbitrary number of personal or group addresses also **groups are composite, emails are leaf**.

2.2 Class Diagram

2.2.1 Classes in my solution



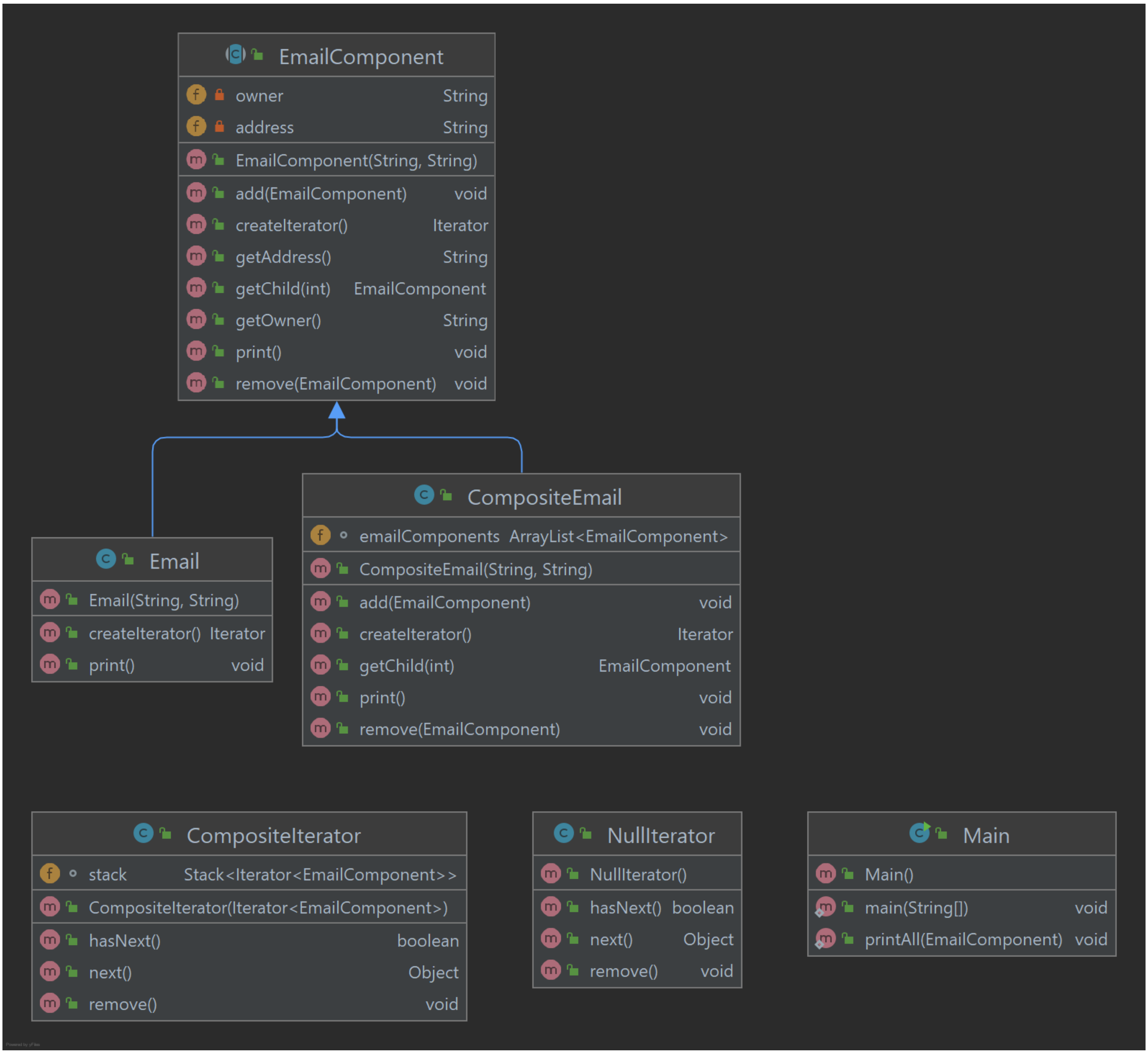
2.2.2 Composite Pattern Class Diagram



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. A 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.
- 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

3.3 Using mutex(es) and monitor(s)

3.3.1 Classes in my solution

- monitors
  - Coordinates
  - TestDrive
  - ThreadFunction
  - ThreadSharedData

*These are general classes in my solution. Note that all threads **will use same thread function** with different coordinates.

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 Example Thread Class

```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());
    }
}
```

*Here, we are injecting shared data and coordinates. Also we are overriding run method from runnable interface. See the comments of critical codes. This is java version of synchronization barrier problem solution.

3.3.4 Creating Threads and Testing

```java
public class TestDrive {
    public static void main(String[] args) {
        // 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(0,4096,0,4096)));
        Thread thread1 = new Thread(new ThreadFunction(data,new Coordinates(0,4096,4096,8192)));
        Thread thread2 = new Thread(new ThreadFunction(data,new Coordinates(4096,8192,0,4096)));
        Thread thread3 = new Thread(new ThreadFunction(data,new Coordinates(4096,8192,4096,8192)));

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

3.3.5 Output

```
Task1 -> XStart: 4096 YStart: 4096
Task1 -> XStart: 0 YStart: 4096
Task1 -> XStart: 4096 YStart: 0
Task1 -> XStart: 0 YStart: 0
Task2 -> XStart: 4096 YStart: 0
Task2 -> XStart: 4096 YStart: 4096
Task2 -> XStart: 0 YStart: 4096
Task2 -> XStart: 0 YStart: 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

3.4.2 Discrete Fourier Transform Formula

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