Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

GÖKTUĞ ALİ AKIN
161044018

Course Assistant:ÖZGÜ GÖKSU

1 INTRODUCTION

1.1 Problem Definition

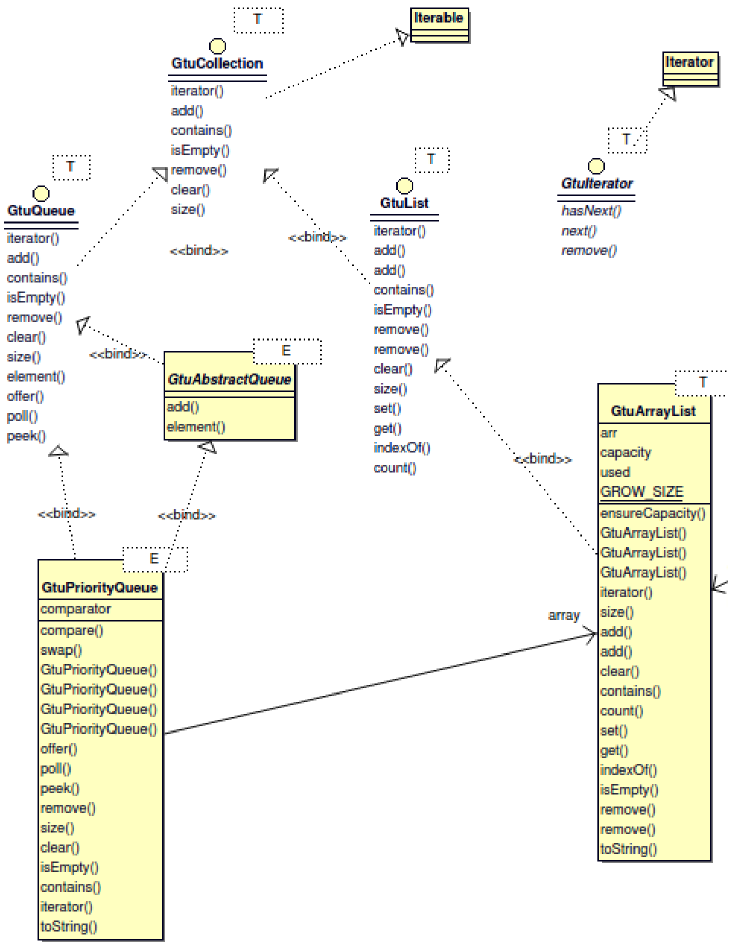
In this homework, the problem is design and implement a program that reads the input image file and add pixels of image into the different priortiy queues and at the same time empty the priority queues synchronously. This priority queues give three kind of priorities that are **LEX**,**EUC** and **BMX** comparision methods.

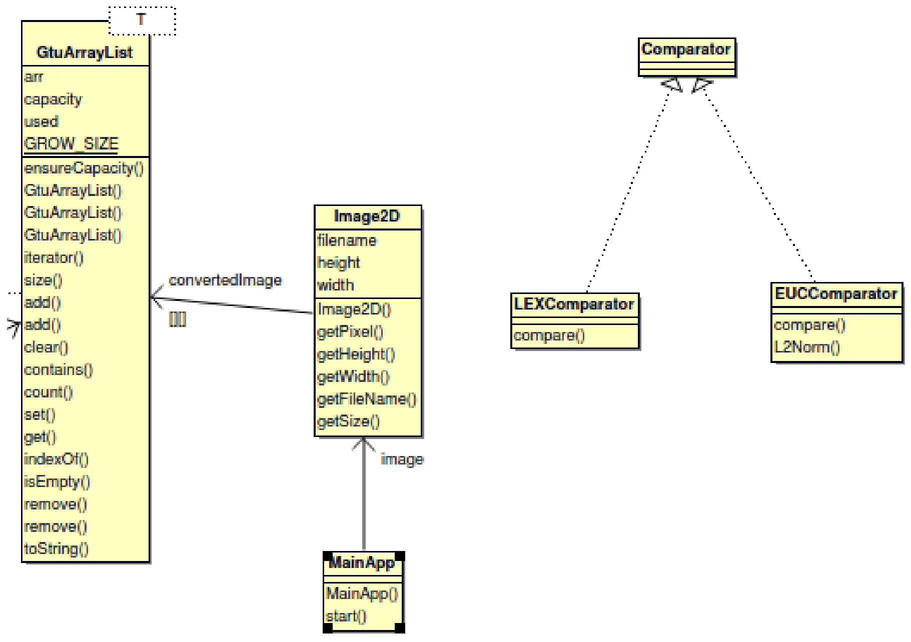
1.2 System Requirements

A user which has already installed Java her/his computer can run this program properly. In addition, this program process the image with different fields at the same. This means, if this program process an image which has X pixels, while the value of X is getting greater, program will runs slowly according to the number of pixels. There is no specific plug-in must be installed to run this program.

2 METHOD

2.1 Class Diagrams





2.2 Use Case Diagrams

There is no graphical user interface designed for this program. User can run this program with giving the path of the image that will be processed with command line or IDE enviroment for now.

2.3 Problem Solution Approach

Loading image

To read the image from file and reads pixel from this image, I used a BufferedImage data structure.

Manipulating the image

In order to easly to manipulate image that is readed by BufferedImage, I implement a Image2D class that converts BufferedImage to 2D array that contains 3D ArrayList which stores RGB values. ArrayList data structure is useful for storing the three color values within the range 0-255. While adding the pixels into priortiy queues, pixels are readed from this class by getPixel(i,j) method.

Priority Queue

I implement Priority Queue using max heap property. To implement priortiy queue with max heap priortiy, I used ArrayList data structure. By using this, I can easly manipulate the data, deal with adding or removing operations and get a linear data structure.

Comparators for Priority Queues

EUCComparator and **LEXComparator** are implemented to use when Priortiy Qeues are constructed. By this comparators, pixels can be compared with different comparision methods such as LEX and EUC. Benefits of this is, one priority queue implementation can be used with different meanings by comparators.

Multi-threading with inserting and removing pixel

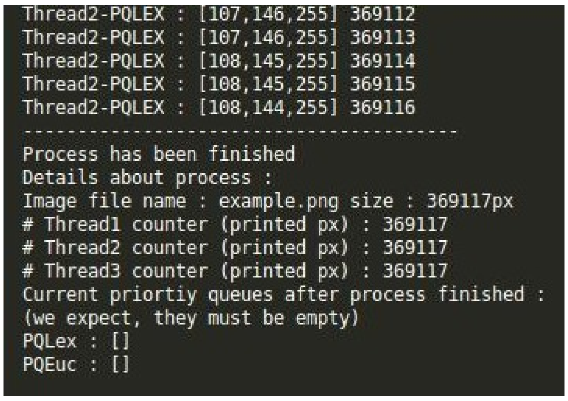
There are 4 threads that runs in this program. First thread, named Thread1, adds pixels into the priority queues which are **PQLEX**,**PQEUC**,**PQBMX**. Second thread, named Thread2, removes the pixel from PQLEX, third thread, named Thread3 removes the pixel from PQEUC, fourth thread, named Thread4, removes the pixel from PQBMX. In this situation, Thread1-Thread2, Thread1-Thread3, Thread1-Thread4 shares the same data which are priortiy queues. To synchronize the threads, I used a data named lock type Object. This data provides a synchronisation to the threads by **wait()** and **notify()** calss which are inherited from **Object** class. If thread2,thread3,thread4 are tries the remove element from empty queue, this process has blocked by wait() method call until thread 1 inserts an element to the related priority queue. These are synchronized between each other queue by queue. So, for example, thread 2, which is responsible for remove pixel from PQLEX, is not wait the thread 1 for adding item to PQEUC. Thread 2 is just wait for PQLEX is empty, Thread 3 is just wait for PQEUC is empty. This is the low-level producer-consumer problem, in this homework, I can deal with this synchronisation issues with synchronized keyword and lock(),notify() methods.

3 RESULT

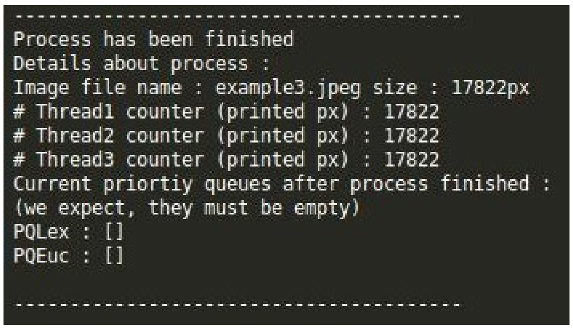
3.1 Test Cases

I tried this program with giving different sized images.

3.2 Running Results



```
Thread2-PQLEX : [107,146,255] 369112
Thread2-PQLEX : [107,146,255] 369113
Thread2-PQLEX : [108,145,255] 369114
Thread2-PQLEX : [108,145,255] 369115
Thread2-PQLEX : [108,144,255] 369116
----------------------------------------
Process has been finished
Details about process :
Image file name : example.png size : 369117px
# Thread1 counter (printed px) : 369117
# Thread2 counter (printed px) : 369117
# Thread3 counter (printed px) : 369117
Current priortiy queues after process finished :
(we expect, they must be empty)
PQLex : []
PQEuc : []
```



```
----------------------------------------
Process has been finished
Details about process :
Image file name : example3.jpeg size : 17822px
# Thread1 counter (printed px) : 17822
# Thread2 counter (printed px) : 17822
# Thread3 counter (printed px) : 17822
Current priortiy queues after process finished :
(we expect, they must be empty)
PQLex : []
PQEuc : []

----------------------------------------
```

After execution of program, priority queues must be empty since thread1 fills the queues,thread 2 and thread 3 are empties the queues.