Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

FURKAN ÖZEV
161044036

Course Assistant: Özgü GÖKSU

1 INTRODUCTION

1.1 Problem Definition

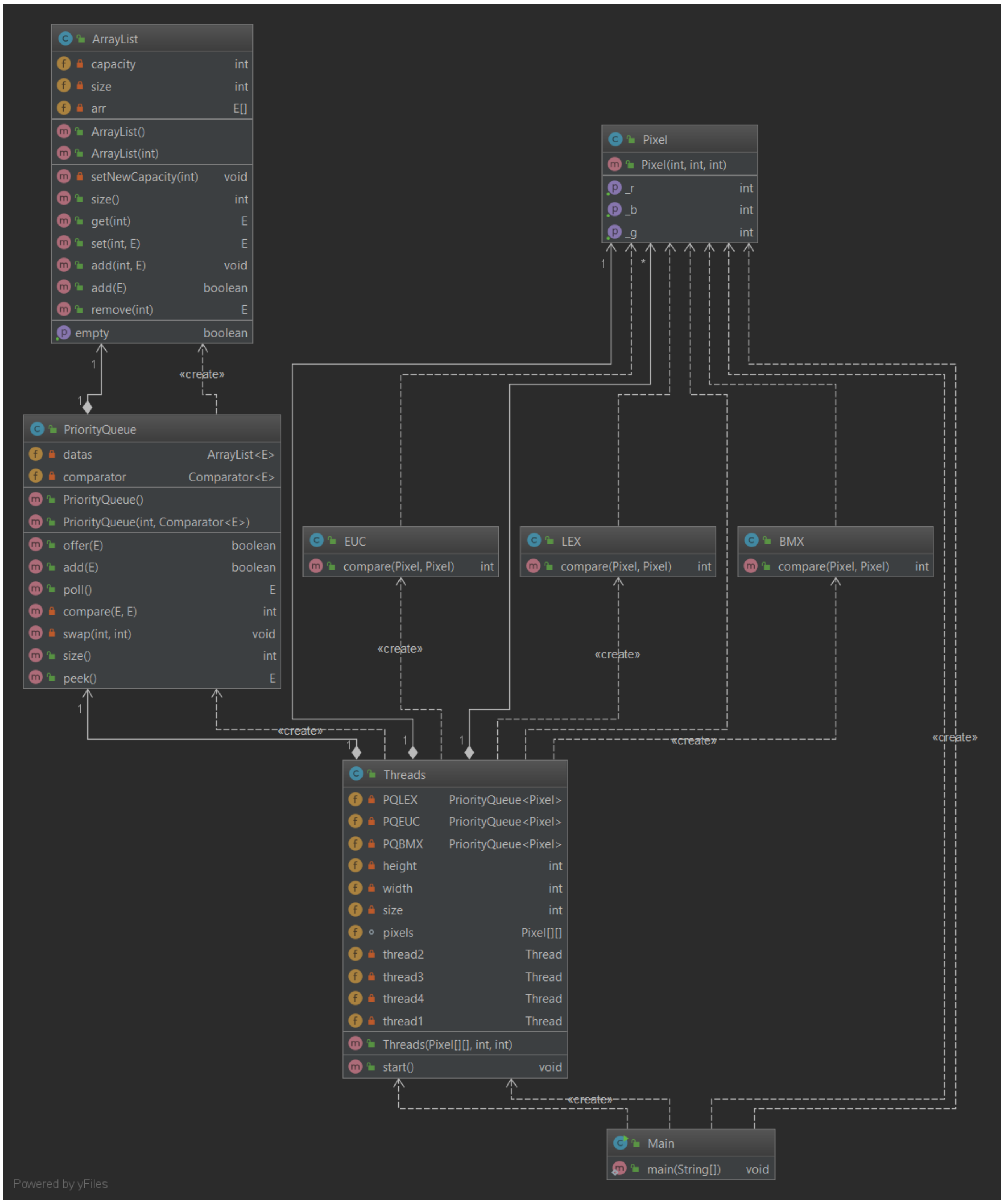
In this assignment, There is a digital image as data. We will compare the pixels of a digital image according to certain conditions. We want to construct max priority queues for color pixels. We want to enter those pixels one by one into priority queues, and then extract them according to different priority schemes. Program will support 3 comparison methods: lexicographical comparison (LEX), Euclidean norm based comparison (EUC) and bitmix comparison (BMX). For each one, 1 priority queue is needed. Totally, 3 priority queue is needed. Also, program will have 4 threads. Thread 1 responsible for reading the pixels from the image starting from the top-left corner and advancing column first until the bottom-right pixel. Every color pixel read, will be inserted to each of the 3 priority queues (PQLEX, PQEUC and PQBMX). As soon as the first 100 pixels are inserted, it will create and start the following 3 threads, and then continue inserting the remaining pixels. Thread 2,3,4 will remove from related queue the color pixels and print them on screen one after the other, up until a total of WxH pixels are printed. Since threads will work according to the CPU's preference, certain problems may arise. One of these problems is that the queue is temporarily empty, while the thread tries to pull the element. In this case, the thread must wait until the element is added to the queue without the CPU busy. When the element is added, it must continue where it left off. Another problem is to try to extract elements before the end of the process while adding elements to the queue. In such a case, the sequence may be disrupted. In order to avoid this situation, when adding elements, the element must not be removed and the same must be valid for opposed situation. Adding to 3 queues and printing the screen must be simultaneously, one element removing from queue and printing the screen must be done at the same time. Thread design should be done properly and synchronized with each other.

1.2 System Requirements

You must have java installed on your operating system to run programs. Running programs with the current version of Java will be useful for efficiency. Programs can work in both windows, linux and other operating systems installed java. There is a need for a digital image that the program can take as an argument for its operation. For the programs to work properly, image must be appropriate for the program. You must enter the path of the image when running the program. The program was written using jdk-11.0.2 in IntelliJ IDEA. It requires 19 kb memory to keep source files. Project size is 1.91 mb with javadoc.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

2.3 Problem Solution Approach

Firstly we need to read the data from the image and keep them. Each pixel has 3 color values. So I created a class called Pixel and I was able to keep these values. I've created a 2D pixel array to hold all the pixels. The first dimension of the array contains row information the second dimension of the array contains column information. I transferred all the pixels in the picture to the 2-dimensional array with the corresponding methods. We wanted to create a maximum order of priority for color pixels, and I used binary heap to implement the priority queue. I chose binary heap because it's easy to implement and the time complexity is quite low. I designed only one priority queue implementation, the constructor of which I customize with a Comparator object parameter to make it prioritize accordingly the elements inside it. So It can do 3 comparisons. For each comparison(LEX, EUC, BMX) I have implemented my comparator class according to the required rules. Then I created the objects of these classes and formed 3 priority queues(PQLEX, PQEUC, PQBMX). Then I created a class called Threads, which takes the 2D pixel array and height and width information. I created 4 threads in this class. The start method of this class is responsible for running thread1. When the start method is called, thread 1 starts to add element to three previously created queue. When the number of elements added is 100, it starts the other three threads. For certain problems that may occur in threads, certain synchronizations and solutions were made: 1. Synchronized for adding and removing elements in the priority queue. 2 operations will not be done at the same time to wait for another one to finish. Thus, The process can wait without keeping the CPU busy. 2. If the order is empty, the working element removal method will wait for the element to be added to the queue. Thus, The process can wait without keeping the CPU busy. 3. At the same time in 3 queues will be added element and the added element will be printed on the screen. It cannot switch to other threads before this is finished. To ensure this, I synchronized the queues. After this is finished, the OS can continue calling the thread it wants. Thus, without conflicts, it will be able to do the operations we want. So we reach our goal. If we examine at our program with time complexity: The time complexity of inserting and removing elements to the queue is O(logn). Because every time, it need to update parent, childrens and their places. The time complexity of accessing max element of queue is O(1). Because max element is always at the top.

3 RESULT

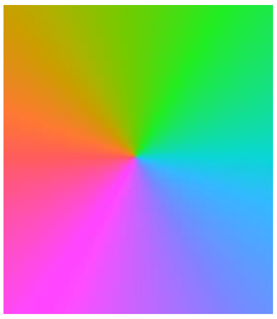
3.1 Test Cases

I run the program with a small digital image of different colors that I created. First I printed the pixels and RGB values on the screen. So I've checked and confirmed if I read the pixels correctly from the picture. I then started adding elements to 3 different queues that I created. I tested and checked the elements after a few additions to test if the comparisons were working correctly. After making sure that they were working properly, I went to test threads. First I started thread1, but for each test phase, when the number of elements passed 100, I started only 1 of the 3 threads. I checked the processes printed on the screen from the beginning to the end and checked that it was working correctly. Any problems of slippage, mess, synchronization etc. I made sure it wasn't. I've checked the number of elements added and removed to make sure it works correctly. Then I ran all the threads so I ran the program as it should. To make sure that all threads are running without any problems, I have done the same controls from the beginning to the end for the processes that are printed on the screen. I've tried the same actions for another picture when I'm sure there's no problem for this picture. When I decided that there was nothing wrong with it, I decided the program was valid. Of course, in all these tests I've encountered certain errors, after correcting these errors I started to test from the beginning. I was convinced that the program was valid.

3.2 Running Results

Test images:

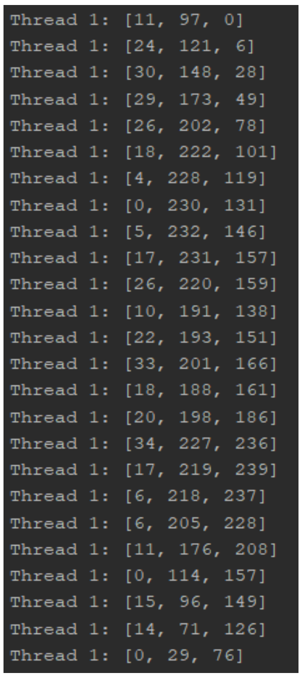
These images have been tested.



Outputs:

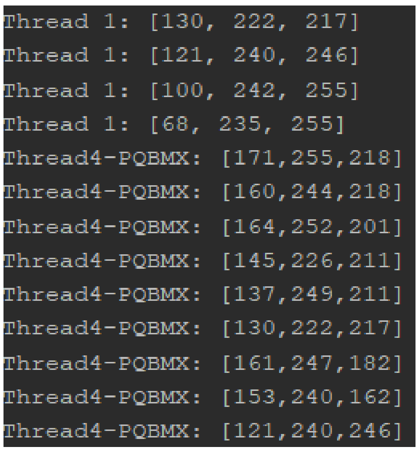
For: 25x19

1. Thread 1 start add element to priority queues.

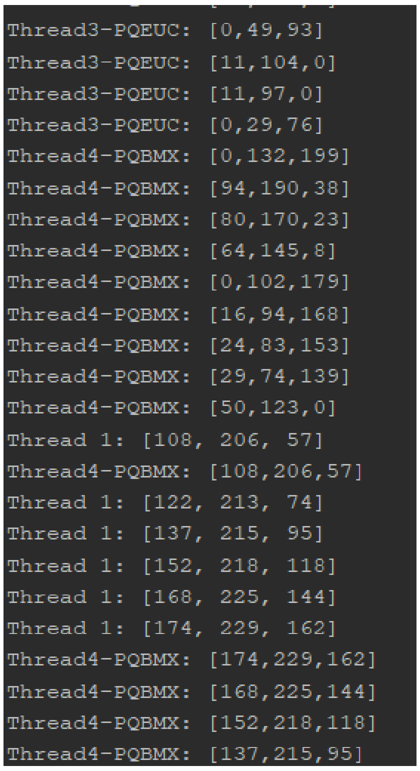


```
Thread 1: [11, 97, 0]
Thread 1: [24, 121, 6]
Thread 1: [30, 148, 28]
Thread 1: [29, 173, 49]
Thread 1: [26, 202, 78]
Thread 1: [18, 222, 101]
Thread 1: [4, 228, 119]
Thread 1: [0, 230, 131]
Thread 1: [5, 232, 146]
Thread 1: [17, 231, 157]
Thread 1: [26, 220, 159]
Thread 1: [10, 191, 138]
Thread 1: [22, 193, 151]
Thread 1: [33, 201, 166]
Thread 1: [18, 188, 161]
Thread 1: [20, 198, 186]
Thread 1: [34, 227, 236]
Thread 1: [17, 219, 239]
Thread 1: [6, 218, 237]
Thread 1: [6, 205, 228]
Thread 1: [11, 176, 208]
Thread 1: [0, 114, 157]
Thread 1: [15, 96, 149]
Thread 1: [14, 71, 126]
Thread 1: [0, 29, 76]
```

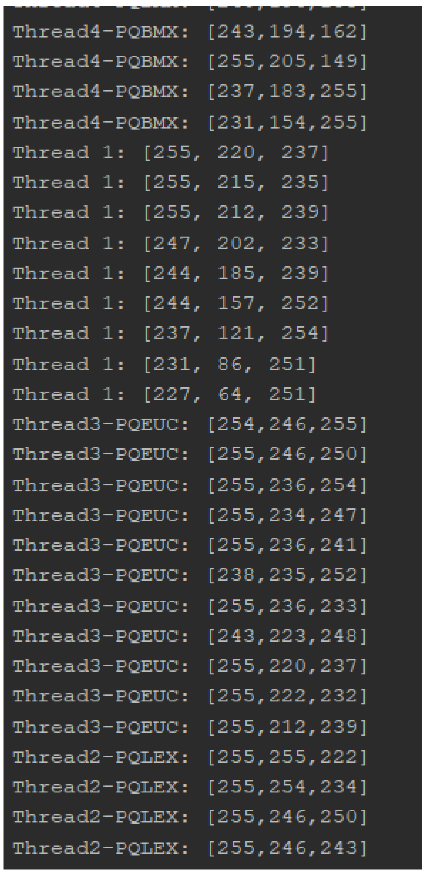
2. After adding 100 elements, starts other threads.



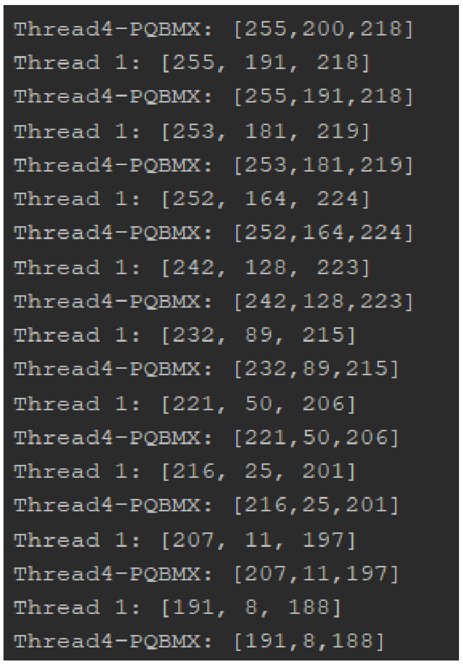
```
Thread 1: [130, 222, 217]
Thread 1: [121, 240, 246]
Thread 1: [100, 242, 255]
Thread 1: [68, 235, 255]
Thread4-PQBMX: [171,255,218]
Thread4-PQBMX: [160,244,218]
Thread4-PQBMX: [164,252,201]
Thread4-PQBMX: [145,226,211]
Thread4-PQBMX: [137,249,211]
Thread4-PQBMX: [130,222,217]
Thread4-PQBMX: [161,247,182]
Thread4-PQBMX: [153,240,162]
Thread4-PQBMX: [121,240,246]
```



```
Thread3-PQEUC: [0,49,93]
Thread3-PQEUC: [11,104,0]
Thread3-PQEUC: [11,97,0]
Thread3-PQEUC: [0,29,76]
Thread4-PQBMX: [0,132,199]
Thread4-PQBMX: [94,190,38]
Thread4-PQBMX: [80,170,23]
Thread4-PQBMX: [64,145,8]
Thread4-PQBMX: [0,102,179]
Thread4-PQBMX: [16,94,168]
Thread4-PQBMX: [24,83,153]
Thread4-PQBMX: [29,74,139]
Thread4-PQBMX: [50,123,0]
Thread 1: [108, 206, 57]
Thread4-PQBMX: [108,206,57]
Thread 1: [122, 213, 74]
Thread 1: [137, 215, 95]
Thread 1: [152, 218, 118]
Thread 1: [168, 225, 144]
Thread 1: [174, 229, 162]
Thread4-PQBMX: [174,229,162]
Thread4-PQBMX: [168,225,144]
Thread4-PQBMX: [152,218,118]
Thread4-PQBMX: [137,215,95]
```



```
Thread4-PQBMX: [243,194,162]
Thread4-PQBMX: [255,205,149]
Thread4-PQBMX: [237,183,255]
Thread4-PQBMX: [231,154,255]
Thread 1: [255, 220, 237]
Thread 1: [255, 215, 235]
Thread 1: [255, 212, 239]
Thread 1: [247, 202, 233]
Thread 1: [244, 185, 239]
Thread 1: [244, 157, 252]
Thread 1: [237, 121, 254]
Thread 1: [231, 86, 251]
Thread 1: [227, 64, 251]
Thread3-PQEUC: [254,246,255]
Thread3-PQEUC: [255,246,250]
Thread3-PQEUC: [255,236,254]
Thread3-PQEUC: [255,234,247]
Thread3-PQEUC: [255,236,241]
Thread3-PQEUC: [238,235,252]
Thread3-PQEUC: [255,236,233]
Thread3-PQEUC: [243,223,248]
Thread3-PQEUC: [255,220,237]
Thread3-PQEUC: [255,222,232]
Thread3-PQEUC: [255,212,239]
Thread2-PQLEX: [255,255,222]
Thread2-PQLEX: [255,254,234]
Thread2-PQLEX: [255,246,250]
Thread2-PQLEX: [255,246,243]
```



```
Thread4-PQBMX: [255,200,218]
Thread 1: [255, 191, 218]
Thread4-PQBMX: [255,191,218]
Thread 1: [253, 181, 219]
Thread4-PQBMX: [253,181,219]
Thread 1: [252, 164, 224]
Thread4-PQBMX: [252,164,224]
Thread 1: [242, 128, 223]
Thread4-PQBMX: [242,128,223]
Thread 1: [232, 89, 215]
Thread4-PQBMX: [232,89,215]
Thread 1: [221, 50, 206]
Thread4-PQBMX: [221,50,206]
Thread 1: [216, 25, 201]
Thread4-PQBMX: [216,25,201]
Thread 1: [207, 11, 197]
Thread4-PQBMX: [207,11,197]
Thread 1: [191, 8, 188]
Thread4-PQBMX: [191,8,188]
```

The elements added thread1 was printed on the screen. In the same way, The elements removed from thread2, thread3 and thread4 was printed on the screen. Printed 475 times for each thread. A total of 1900 times were printed on the screen.