Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 5 REPORT

ÖMER ÇEVİK
161044004

Course Assistant: Özgü GÖKSU

1 INTRODUCTION

1.1 Problem Definition

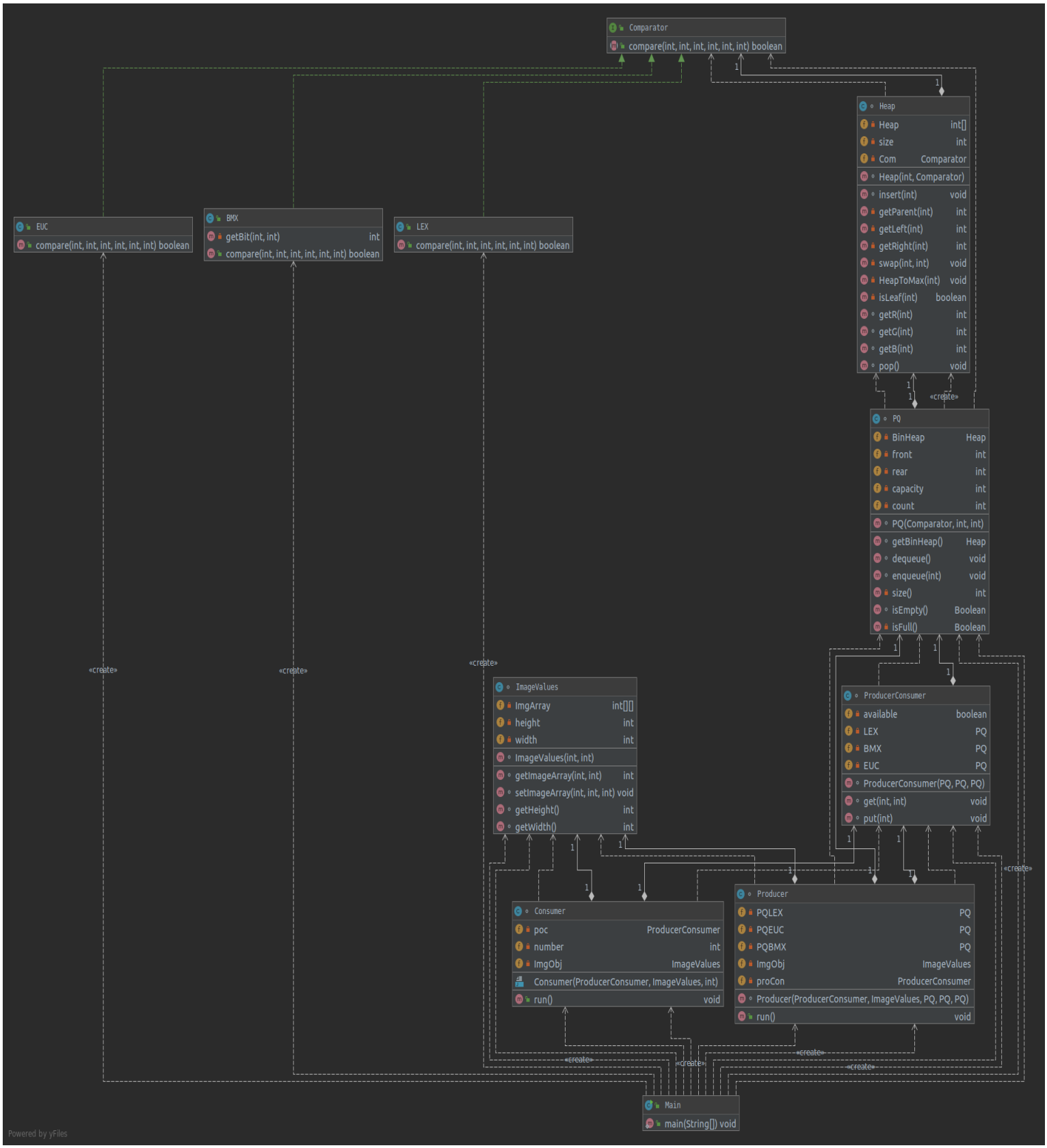
In that programme, the problem is there is an image and we want to sort the pixels' colors (red, green and blue) of that image using three methods: BMX, EUC and LEX. But all these method uses one thread to calculate. We have to use priority queue data structure but that priority queue must keep a binary tree heap data structure. First 100 pixels will be inserted in thread 1. After that other 3 threads will be awake to calculate. They will delete pixels while thread 1 inserts.

1.2 System Requirements

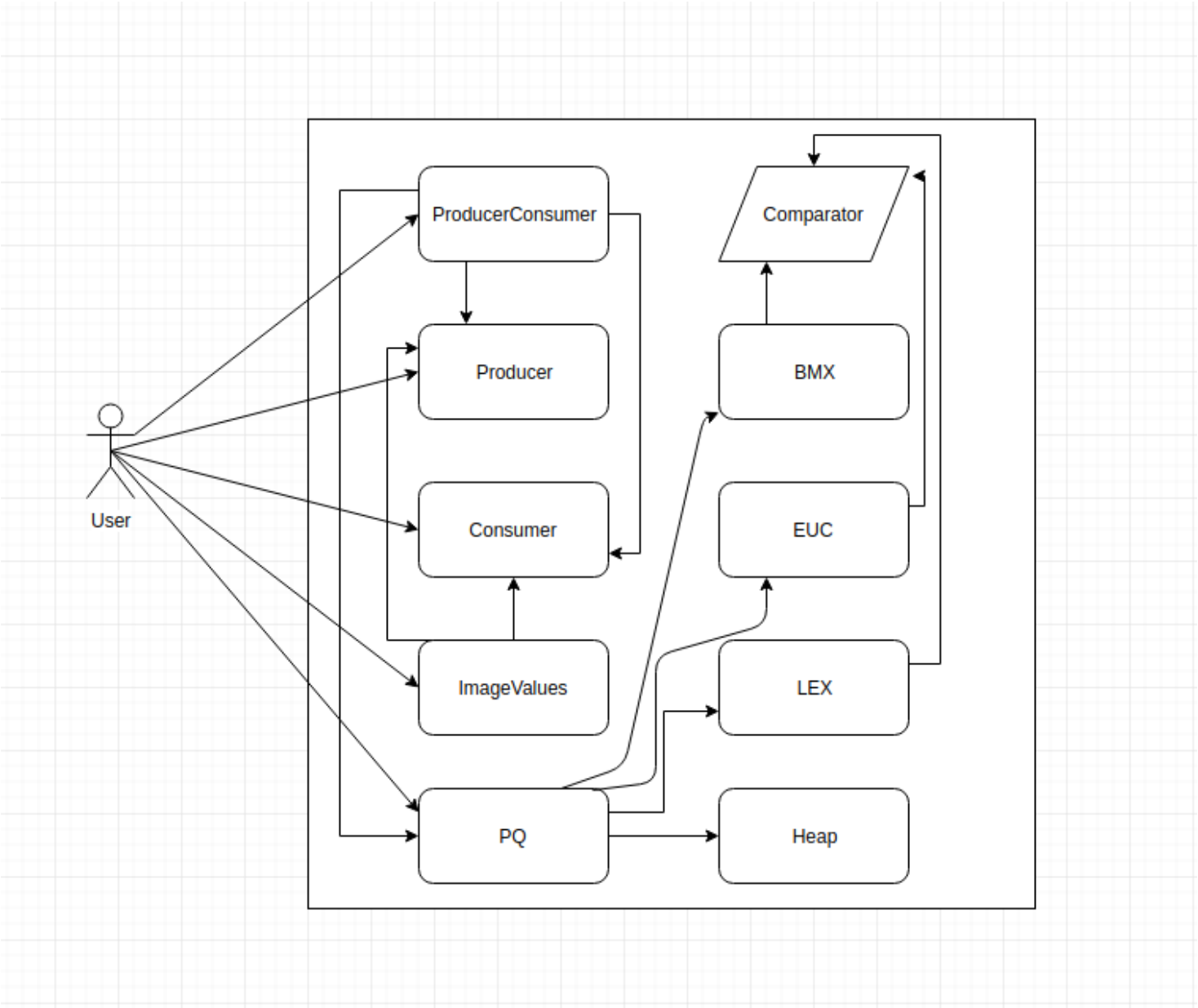
My solution doesn't require a specific piece of hardware, or maybe a certain minimal amount of memory, or a certain operating system, or a software library to be installed in order to run properly. It built by IntelliJ IDEA and used java programming language. It might run correctly if you use Java Virtual Machine and JDK is at least 8. Will my program work with 128KB of memory? I don't know, I didn't work with 128KB of memory but maybe it can work. If your nephew's smartphone's operating system is Android then it must run on it.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams



2.3 Problem Solution Approach

In my programme, I used Priority Queue and Binary Heap Tree data structures. I created priority queue class using that heap class. I used integer array to create heap class. I had one interface called Comparator. That Comparator interface has only one method called compare(). I also created BMX, EUC and LEX classes which implements Comparator interface and overrides that interface's compare() methods. First of all my priority queue class takes comparator object to know what kind of queue will it saves. And of course it gets size information. User creates three priority queues and gives as parameter its priority. All of these queues keeps the maximum priority of pixel specification on top. Maximum binary heap is represented. While inserting an element to queue it checks if it is bigger than its parent till to head node. And insert it to its right place. I didn't add as a result page but I've checked the tree structure giving values in main method. The most hard part was to creating and synchronizing the threads. It was hard to think because of producer-consumer problem. Producer inserts the pixels into queue, consumer deletes the pixels from the queue. While producer inserts the pixels, if inserted pixels are 100 then other threads awakes to delete their pixels. But it must to be synchronized to delete till no pixels have exist. Consumers waits the producer and deletes.

Time Complexity:

- Heap class : Has some getters and setters O(1). Insert an element is O(logn). Delete an element is O(logn).
- PQ class : Has some getters and setters O(1). Insert an element is O(logn). Delete an element is O(logn).
- ProducerConsumer class : Makes relations between producer and consumer O(1).
- Producer class : Inserts image pixels to queue is O(n).
- Consumer class : Deletes image pixes from queue is O(n).
- ImageValues class : Just getters and setters O(1).
- EUC class : compare() method overrides O(1).
- BMX class : compare() method overrides O(1).
- LEX class : compare() method overrides O(1).

Space Complexity:

- Heap class : ImageSize x 4 + 4 + ComparatorSize
- PQ class : HeapSize + 16
- ProducerConsumer class : 3 x PQSize + 1
- Producer class : ProducerConsumerSize + 3 x PQSize + ImageValuesSize
- Consumer class : ProducerConsumerSize + ImageValuesSize + 4
- ImageValues class : 4 x ImageSize + 8
- EUC class : 1
- BMX class : 5
- LEX class : 1

3 RESULT

3.1 Test Cases

Main Test

I used main method to test my programme. In main method, I called my methods and creating objects. Then runned them. The results are in **3.2 Running Results** part.

3.2 Running Results