Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 5 REPORT

EMRE KAVAK
151044085

Course Assistant:

1 INTRODUCTION

1.1 Problem Definition

Problem was reading a jpf or png file from console and store this image pixels different Queues. We have had to use one Binary heap impelementation for this Queues and we have had to use 3 different comparasion method named LEX, EUC and BMX. Also, we should have to use threads. There were 4 different threads and these threads task were different. First threads was has to read pixels and add they into 3 different Queues. Another threads have had to remove pixels from these Queues

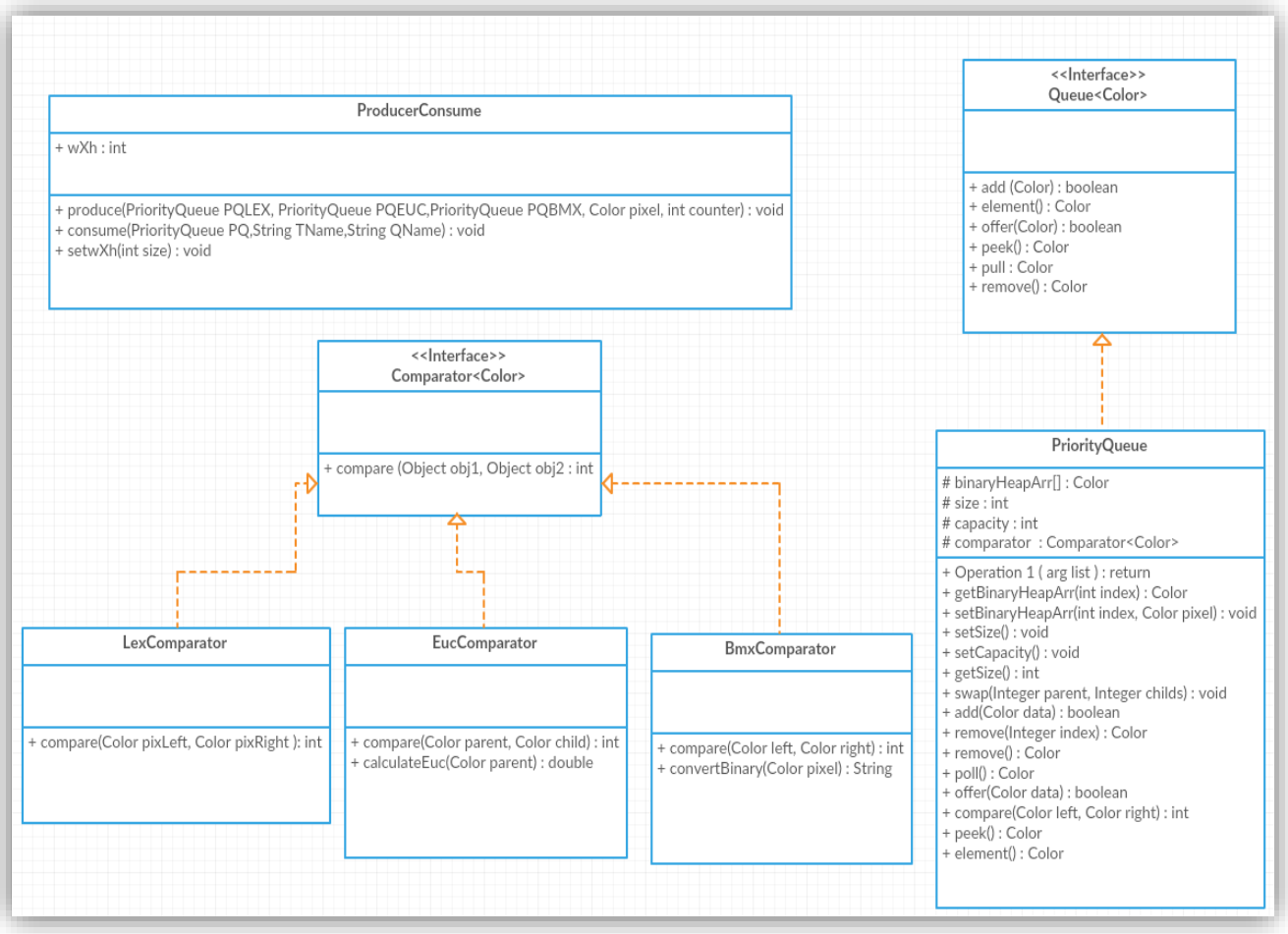
1.2 System Requirements

This program run on intelliJ IDE. I mean, Project file will be intelliJ file. So, for run this program there must be intelliJ IDE. So, you can just run this program on computer or laptop. Also you have enter a text file path via command line (it must be full path from root). This file should have contain a jpg or png file.

IntelliJ IDE Reguirements : Minimum 2GB Ram, At least 2 GB harddisk space,Windows operating System.

2 METHOD

2.1 Class Diagrams



2.2 Problem Solution Approach

For read image file I use **BufferedImage** class object. I used **Binary Heap** implementation for hold pixels. Because we have to hold max element in the root and binary heap hold elements dicreasing order. So, its **complexity is O(logn).** I used 3 different comparasion methods named **LEX**, **EUC** and **BMX**. LEX comparasion method compare 2 Color objects r,g,b values according to lower or bigger each other. EUC method compare objects according to r,g,b colors values power sum and square. Finally BMX method, convert integer values into binary and create an object order **r1g2b1r2g2b2… r24g24b24** and compare two object according to this object. I used 3 comparator class named **LexComparator, EucComparator and BmxComparator.** This classes implements **Comparator interface** and compare methods. These compare methods above methods (LEX, EUC, BMX).

I implement PriorityQueue class, this class have binary heap implementation with just hold array named **binaryHeapArr.** This array's 0. Element always max element **(root).** Childs indexs start (last item) size -1 and parent index is always (size-1) / 2.

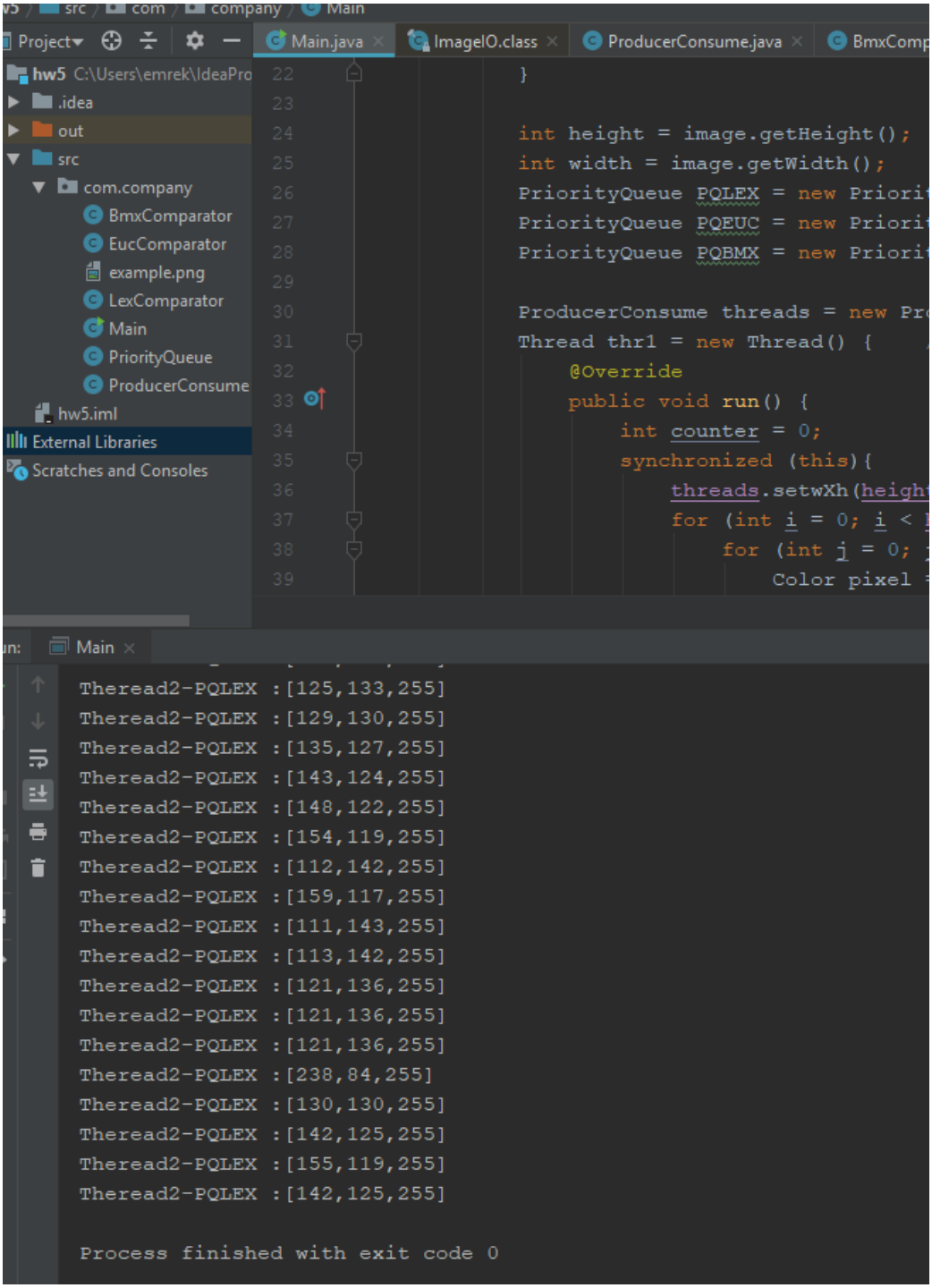
I used 4 different **thread**. For use Producer and consumer methods, I created **ProducerConsume** class. This class have 2 methods named **Producer** and **consume**. I used just one object this class named **threads** and when I start **thread1,** this thread use **threads** object producer method and I send 3 PriorityQueue object named **PQLEX**, **PQEUC** and **PQBMX.** This object just and PriorityQueue object. PriorityQueue class constructor parameter have an comparator object parameter. I creat comparator class objects (**LexComparator, EucComparator, BmxComparator**) and I send it with PriorityQueue objects parameter. When I did it, PriorityQueu object compare objects according to comparator.When producer method reach 100 pixel saved, I notify() another threads and these threads remove elements from 3 different Queue and print they until reach Width * height.

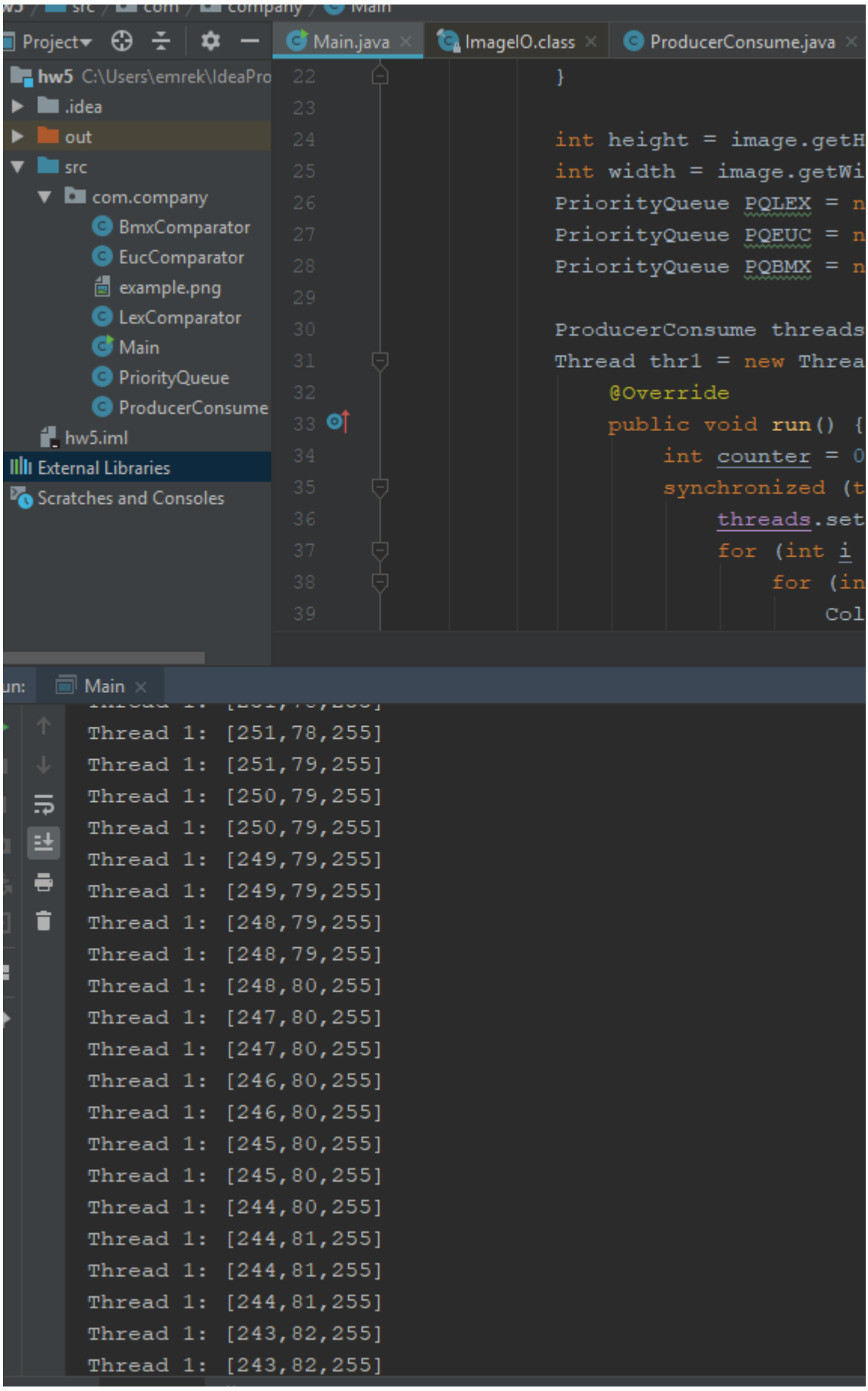
3 RESULT

3.1 Test Cases

I used pictures and read it and print its pixels values

3.2 Running Results




Project
hw5 C:\Users\emrek\IdeaPro
.idea
out
src
com.company
BmxComparator
EucComparator
example.png
LexComparator
Main
PriorityQueue
ProducerConsume
hw5.iml
External Libraries
Scratches and Consoles
Main.java
ImageIO.class
ProducerConsume.java
}
int height = image.getH
int width = image.getWi
PriorityQueue PQLEX = n
PriorityQueue PQEUC = n
PriorityQueue PQBMX = n
ProducerConsume threads
Thread thr1 = new Threa
@Override
public void run() {
int counter = 0
synchronized (t
threads.set
for (int i
for (in
Col
Main
Thread 1: [251,78,255]
Thread 1: [251,79,255]
Thread 1: [250,79,255]
Thread 1: [250,79,255]
Thread 1: [249,79,255]
Thread 1: [249,79,255]
Thread 1: [248,79,255]
Thread 1: [248,79,255]
Thread 1: [248,80,255]
Thread 1: [247,80,255]
Thread 1: [247,80,255]
Thread 1: [246,80,255]
Thread 1: [246,80,255]
Thread 1: [245,80,255]
Thread 1: [245,80,255]
Thread 1: [244,80,255]
Thread 1: [244,81,255]
Thread 1: [244,81,255]
Thread 1: [244,81,255]
Thread 1: [243,82,255]
Thread 1: [243,82,255]