PART 1 OF THE FINAL HOMEWORK

MAIN CLASS;

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Scanner;

//@AUTHOR: GORKEM TOPRAK
//DATE: January 22, 2021 Friday

public class Main {

    public static int position = 0;

    public static void main(String[] args) {

        Scanner scan=new Scanner(System.in);
        System.out.print("Please enter a txt name:");
        String fileName= scan.nextLine();

        ReadFile readFile = new ReadFile();
        readFile.scanFile(fileName+".txt");

        int width = readFile.width;
        int height = readFile.height;

        System.out.println(width + " " + height);

        //THIS IS FOR ORIGINAL IMAGE (TAB 0)
        int[][][] pixelsForOriginal = readFile.pixels;
        //gx= vertical gy=horizontal

        //THIS IS FOR GRAYSCALE. BECAUSE ORIGINAL IMAGE IS COLORED IMAGE
SO I HAVE TO CREATE GREYSCALE IMAGE..
        GrayScale greyScale = new GrayScale();
        greyScale.createGrayScale(width, height, pixelsForOriginal);
        int[][] pixelsForGrayScale = greyScale.pixels;

        //THIS IS FOR GX EDGE IMAGE (TAB 1)
        GxEdgeImage gxEdgeImage = new GxEdgeImage();
        gxEdgeImage.createGxImage(width,height,pixelsForGrayScale);
        int [][] gX = gxEdgeImage.gY;

        //THIS IS FOR GY EDGE IMAGE (TAB 2)
        GyEdgeImage gyEdgeImage = new GyEdgeImage();
        gyEdgeImage.createGyImage(width,height,pixelsForGrayScale);
        int [][] gY = gyEdgeImage.gX;

        //THIS IS FOR GX and GY EDGE IMAGE (TAB 3)
        GEdgeImage createGEdge = new GEdgeImage();
        createGEdge.createGImage(gX,gY,width,height);
        int [][] g = createGEdge.g;

        //THIS IS FOR HORIZONTAL EDGE IMAGE (TAB 4)
        HOGEdge hogEdge = new HOGEdge();
        hogEdge.createHOGImage(gX, gY, width, height);
        int[][] pixelsForHorizantal = hogEdge.hogEdge;

        //THIS IS THE LAST PART OF THE HOMEWORK..(TAB 5)
        // (I created an array of 2 named posX and posY for x and y coordinates.)
        final int[] posX = new int[2];
        final int[] posY = new int[2];

        SketchPanel clickableTab1 = new SketchPanel(width, height, pixelsForGrayScale);
        ClickableImage clickableImage = new ClickableImage();

        //THIS IS THE SECOND OPTION FOR THE LAST PART OF THE
HOMEWORK..(TAB 5)
        ClickableJPanel clickableJPanel = new ClickableJPanel();

//        SketchPanel tab1 = new SketchPanel(width,height,pixelsForGrayScale);
        SketchPanel tab2 = new SketchPanel(width,height,gX);
        SketchPanel tab3 = new SketchPanel(width,height,gY);
        SketchPanel tab4 = new SketchPanel(width,height,g);
        HOGSketch tab5 = new HOGSketch(width, height, pixelsForHorizantal);
        SketchPanel tab6 = new SketchPanel(width, height, new int[width][height]);

        final JFrame frame = new JFrame("Gorkem Toprak Final HW");
        frame.setSize(600, 600);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new GridLayout(1, 1));

        JTabbedPane tabbedpane = new JTabbedPane(JTabbedPane.TOP);
        tabbedpane.addTab("Original",clickableTab1);
        tabbedpane.addTab("Gx Edge",tab2);
        tabbedpane.addTab("Gy Edge",tab3);
        tabbedpane.addTab("G Edge",tab4);
        tabbedpane.addTab("HOG Edge",tab5);
        tabbedpane.addTab("Tab6", tab6);
        frame.getContentPane().add(tabbedpane);

        // Here I have created a clickable panel using a addmouselistener. Here I went
completely according to the oracle document
        // An abstract adapter class for receiving mouse events. The methods in this class are
empty.
```

```
        // This class exists as convenience for creating listener objects.
        // After defining the mouse adapter class, I used the two methods in it. Named
mousePressed and mouseReleased
        clickableTab1.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent me) { // Invoked when a mouse button has
been pressed on a component.
                if (position < 2) {
                    // A method from the getX and getY mouse event class.
                    posX[position] = me.getX(); // It returns the horizontal x position of the event
relative to the source component.
                    posY[position] = me.getY(); // It returns the vertical y position of the event
relative to the source component.
                }
            }

            public void mouseReleased(MouseEvent me) { // Invoked when a mouse button has
been released on a component.
                if (me.getX() < pixelsForOriginal.length && me.getY() <
pixelsForOriginal[0].length) {
                    position++; // If I do not increase the position, it will remain at 0 and a black
screen will remain on the panel.
                    // If the positions at x and y are taken, it takes the positions of those selected
                    // pixels in the original image and draws them to tab 6.
                    if (position == 2) {
                        int[][] tab6Pixels = clickableImage.createClickableTab(posX, posY,
pixelsForGrayScale);
                        tabbedpane.removeTabAt(5);
                        tabbedpane.addTab("Tab6", new SketchPanel(tab6Pixels.length,
tab6Pixels[0].length, tab6Pixels));
                    }
                }
            }
        });
    }
}
```

<u>GRAYSCALE CLASS;</u>

```
public class GrayScale {

    public int[][] pixels = null;

    public void createGrayScale(int width, int height, int[][][] pixelsForRGB) {

        pixels = new int[width][height];
        for (int col = 0; col < width; col++) {
            for (int row = 0; row < height; row++) {
                pixels[col][row] = (int) ((0.2126 * pixelsForRGB[col][row][0]) + (0.7152 *
pixelsForRGB[col][row][1]) + (0.0722 * pixelsForRGB[col][row][2]));
            }
        }

    }
}
```

READ FILE CLASS;

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadFile {
    public int width;
    public int height;
    public int[][][] pixels=null;

    public void scanFile(String fileName){
        Scanner inFile=null;
        try {
            inFile = new Scanner(new File(fileName));
            int fileType = inFile.nextInt();
            width = inFile.nextInt();
            height = inFile.nextInt();
            inFile.nextInt();
            System.out.printf("type: %d, width: %d, height:%d\n",
                    fileType, width, height);
            pixels = new int[width][height][3];
            for(int col = 0; col < height; col++)
                for(int row = 0; row < width; row++) {
                    for(int rgb=0;rgb<3;rgb++) {
                        pixels [row][col][rgb] = inFile.nextInt();
                    }
                }
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

SKETCH PANEL CLASS;

```
import javax.swing.*;
import java.awt.*;

public class SketchPanel extends JPanel {
    int width,height;
    int[][]pixels;
    public SketchPanel(int width,int height,int[][]pixel){

        this.width = width;
        this.height = height;
        this.pixels = pixel;

    }
    public void paintComponent(Graphics g){
        for(int row = 0; row < height; row++)
            for(int col = 0; col < width; col++){
                if(pixels[col][row]>=0 && pixels[col][row]<256){
                    g.setColor(new Color(pixels[col][row],pixels[col][row],pixels[col][row]));
                    g.fillRect(col, row, 1, 1);
                }
            }
    }
}
```

GX EDGE IMAGE CLASS;

```
//@AUTHOR: GORKEM TOPRAK
//DATE: January 22, 2021 Friday

public class GxEdgeImage {
    public int max = 0;
    public int[][] gXEdge;
    public int[][] gY;
    public int threshold = 0;

    public void createGxImage(int width, int height, int[][] pixel) {
        gXEdge = new int[width][height];
        gY= new int[width][height];
        for (int i = 0; i < width-1; i++) {
            for (int j = 0; j < height-1; j++) {
                if(i != 0 && j != 0){
                    int index00 = pixel[i - 1][j - 1];
                    int index01 = pixel[i - 1][j];
                    int index02 = pixel[i - 1][j + 1];
                    int index10 = pixel[i][j - 1];
                    int index11 = pixel[i][j];
                    int index12 = pixel[i][j + 1];
                    int index20 = pixel[i + 1][j - 1];
                    int index21 = pixel[i + 1][j];
                    int index22 = pixel[i + 1][j + 1];

                    int calculationX = Math.abs(((-1 * index00) + (-2 * index01) + (-1 * index02)) +
((0 * index10) + (0 * index11) + (0 * index12))
                            + ((1 * index20) + (2 * index21) + (1 * index22)));
                    if(calculationX>max){
                        max = calculationX;
                    }
                    gXEdge[i][j] = calculationX;
                }
            }
        }
        threshold = max / 255;
        for (int i = 1; i < width - 1; i++) {
            for (int j = 1; j < height - 1; j++) {
                gY[i][j] = gXEdge[i][j] / threshold;

            }
        }
    }
}
```

GY EDGE IMAGE CLASS;

```
//@AUTHOR: GORKEM TOPRAK
//DATE: January 22, 2021 Friday

public class GyEdgeImage {

    public int max = 0;
    public int[][] gYEdge;
    public int[][] gX;
    public int threshold = 0;

    public void createGyImage(int width, int height, int[][] pixel) {
        gYEdge = new int[width][height];
        gX= new int[width][height];
        for (int i = 0; i < width-1; i++) {
            for (int j = 0; j < height-1; j++) {
                if(i != 0 && j!= 0){
                    int index00 = pixel[i - 1][j - 1];
                    int index01 = pixel[i - 1][j];
                    int index02 = pixel[i - 1][j + 1];
                    int index10 = pixel[i][j - 1];
                    int index11 = pixel[i][j];
                    int index12 = pixel[i][j + 1];
                    int index20 = pixel[i + 1][j - 1];
                    int index21 = pixel[i + 1][j];
                    int index22 = pixel[i + 1][j + 1];

                    int calculationY = Math.abs(((-1 * index00) + (0 * index01) + (1 * index02)) +
((-2 * index10) + (0 * index11) + (2 * index12))
                        + ((-1 * index20) + (0 * index21) + (1 * index22)));

                    if(calculationY>max){
```

```
                max = calculationY;
            }
            gYEdge[i][j] = calculationY;
          }
        }
      }
      threshold = max / 255;
      for (int i = 1; i < width - 1; i++) {
        for (int j = 1; j < height - 1; j++) {
          gX[i][j] = gYEdge[i][j] / threshold;

        }
      }
    }
}
```

<u>G EDGE IMAGE CLASS;</u>

```
public class GEdgeImage {
  public int max=0;
  public int[][] gEdge;
  public int[][] g;
  public int threshold = 0;
  public void createGImage(int gXPixels[][], int gYPixels[][],int width,int height){
    gEdge = new int[width][height];
    g= new int[width][height];
    for (int i = 0; i < width; i++) {
      for (int j = 0; j < height; j++) {

        int gradientValue = (int) Math.sqrt(Math.pow(gXPixels[i][j], 2) +
Math.pow(gYPixels[i][j], 2));
        if(gradientValue>max){
          max=gradientValue;
        }
        gEdge[i][j]=gradientValue;
      }
    }
    threshold =max/255;
    for (int i = 1; i < width - 1; i++) {
      for (int j = 1; j < height - 1; j++) {
        g[i][j]= gEdge[i][j]/ threshold;

      }
    }

  }
}
```

HORIZONTAL EDGE IMAGE CLASS;

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.HashMap;

public class HOGEdge {

   public int max=0;
   public int[][] hogEdge;
//   public int[][] g;
//   public int threshold = 0;
//   public int sideHeight = 0;
//   public int sideWidth = 0;
//   public int countWidth = 0;
//   public int countHeight = 0;

   public void createHOGImage(int gXPixels[][], int gYPixels[][], int c, int b) {
//       hogEdge = new int[width][height];
//       g = new int[width][height];
      BufferedImage bi = null;
      try {
         bi = ImageIO.read(new File("circle.jpg"));
      } catch (IOException e) {
         e.printStackTrace();
      }
      int width =bi.getWidth();
      int height =bi.getHeight();
      int anzpixel= width*height;
      int[] histogram = new int[255];
      int[] iarray = new int[1];
      int i =0;

      //read pixel into histogram
      for (int x = 1; x < width; x++) {
         for (int y = 1; y < height; y++) {
            int valueBefore=bi.getRaster().getPixel(x, y,iarray)[0];
            histogram[valueBefore]++;
         }
      }

      int sum =0;
      float[] lut = new float[anzpixel];
      for ( i=0; i < 255; ++i )
      {
         sum += histogram[i];
         lut[i] = sum * 255 / anzpixel;
      }
```

```
        for (int x = 1; x < width; x++) {
            for (int y = 1; y < height; y++) {
                int valueBefore=bi.getRaster().getPixel(x, y,iarray)[0];
                int valueAfter= (int) lut[valueBefore];
                iarray[0]=valueAfter;
                bi.getRaster().setPixel(x, y, iarray);
            }
        }

//        sideHeight = height - 2;
//        sideWidth = width - 2;
//
//        for (int i = 0; i < width; i++){
//            i = i+8;
//            if((i + 7 <= sideWidth)){
//               countWidth++;
//            }
//            else if((i + 7 <= sideHeight)){
//                countHeight++;
//            }
//            else{
//                i = countHeight * countWidth;
//            }
//        }
//        GxEdgeImage gxEdgeImage = new GxEdgeImage();
//        gxEdgeImage.createGxImage(width,height,gXPixels);
//
//        for (int i=0; i<countWidth; i++){
//            for (int j=0; j<countHeight; j++){
//
//            }
//        }

//        for (int i = 0; i < width; i++) {
//            for (int j = 0; j < height; j++) {
//                int gradientValue = (int) Math.toDegrees(Math.atan(-gYPixels[i][j] /
(gXPixels[i][j]+1)));
//                if (gradientValue < max) {
//                    max = gradientValue;
//                }
//                hogEdge[i][j] = gradientValue;
//            }
//        }
//        threshold = max / 255;
//        for (int i = 1; i < width - 1; i++) {
//            for (int j = 1; j < height - 1; j++) {
//                g[i][j] = hogEdge[i][j] / (threshold + 1);
//
//            }
//        }
    }
}
```

<u>HORIZONTAL EDGE IMAGE SKETCH CLASS;</u>

```
import javax.swing.*;
import java.awt.*;

public class HOGSketch extends JPanel {

    int[][] histogram;
    int width, height;

    public HOGSketch(int width, int height, int[][] histogram) {
        this.histogram = histogram;
        this.width = width;
        this.height = height;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLACK);

        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                if (col % 9 == 0) {
                    g.setColor(Color.RED);
                    g.drawLine(row * 9 + col + 10, 300, row * 9 + col + 10, 250);
                    g.setColor(Color.BLACK);
                }
                g.drawLine(row * 9 + col + 10, 300, row * 12 + col + 10, 300);

            }
            // g.setColor(new Color(pixels[row], pixels[row], pixels[row]));
        }
    }
}
```

<u>CLICKABLE IMAGE CLASS (ACTUAL PART5 OF THE HW);</u>

```
//@AUTHOR: GORKEM TOPRAK
//DATE: February 3, 2021 Wednesday

//THIS IS THE ACTUAL PART OF THE TAB5
public class ClickableImage {

    public static int[][] createClickableTab(int[] x, int[] y, int[][] pixelsForGrayScale) {
        int x1 = Math.min(x[0], x[1]);
        int y1 = Math.min(y[0], y[1]);
        int x2 = Math.max(x[0], x[1]);
        int y2 = Math.max(y[0], y[1]);

        int xLength = (x2 - x1);
        int yLength = (y2 - y1);

        int[][] originalImagePixels = new int[xLength * 2][yLength * 2];

        for (int i = x1, positionX = 0; i < x2; i++, positionX += 2) {
            for (int j = y1, positionY = 0; j < y2; j++, positionY += 2) {
                originalImagePixels[positionX][positionY] = pixelsForGrayScale[i][j];
                originalImagePixels[positionX + 1][positionY] = pixelsForGrayScale[i][j];
                originalImagePixels[positionX][positionY + 1] = pixelsForGrayScale[i][j];
                originalImagePixels[positionX + 1][positionY + 1] = pixelsForGrayScale[i][j];
            }
        }
        return originalImagePixels;
    }
}
```

PART 2 OF THE FINAL HOMEWORK

CUSTOMER CLASS;

```
//@AUTHOR: GORKEM TOPRAK
//DATE: JANUARY 23, 2021 SATURDAY
//TOPIC: JAVA-8 STREAMS

public class Customer {

    private String name;
    private String surname;
    private int year;
    private String city;
    private int purchase;

    public Customer(String name, String surname, int year, String city, int purchase){
        this.name = name;
        this.surname = surname;
        this.year = year;
        this.city = city;
        this.purchase = purchase;
    }

    public String getName(){
        return this.name;
    }

    public String getSurname(){
        return this.surname;
    }

    public int getYear(){
        return this.year;
    }

    public String getCity(){
        return this.city;
    }

    public int getPurchase(){
        return this.purchase;
    }

    public void setCity(String newCity){
        this.city = newCity;
    }


    public String toString(){
        return "{" + "Customer Name: " + this.name + " Surname: " + this.surname + " Year: " +
this.year + " City: " + this.city + " Amount of Purchase: " + this.purchase +"}";
    }
}
```

<u>MAIN CLASS;</u>

```
import java.util.*;
import java.util.stream.Stream;
import java.util.function.Predicate;

import static java.util.Comparator.comparing;
import static java.util.stream.Collectors.toList;

//@AUTHOR: GORKEM TOPRAK
//DATE: JANUARY 23, 2021 SATURDAY
//TOPIC: JAVA-8 STREAMS

public class Main {

    public static void main(String[] args) {

        Customer cust1 = new Customer("Gorkem", "Toprak",2011, "Istanbul", 10);
        Customer cust2 = new Customer("Mert","Toprak",2011, "Istanbul", 24);
        Customer cust3 = new Customer("Volkan","Ozer",2012, "Ankara", 34);
        Customer cust4 = new Customer("Tolga","Ozer",2016, "Erzincan", 120);
        Customer cust5 = new Customer("Baris","Manco",1999, "Tekirdag", 900);
        Customer cust6 = new Customer("Cem","Karaca",2008, "Mersin", 530);
        Customer cust7 = new Customer("Ahmet","Mehmet",2018, "Yozgat", 1);
        Customer cust8 = new Customer("John", "Snow",2000, "Manisa", 65);
        Customer cust9 = new Customer("Snoop", "Dogg",2010, "Los Angeles", 90);
        Customer cust10 = new Customer("Marie","Plassard",2011, "San Francisco", 650);

        List<Customer> newCustomerList =
Arrays.asList(cust1,cust2,cust3,cust4,cust5,cust6,cust7,cust8, cust9, cust10);
        //This is optional, i can use also this one.. (Just for try)
//        Stream<NewCustomer> customerStream = customerList.stream();

        //THIS IS FOR QUESTION 1
        //1. Find all transactions in the year 2011 and sort them by value (small to high).
        List<Customer> sortYear = newCustomerList.stream().filter(transaction ->
transaction.getYear() == 2011).sorted(comparing(Customer::getPurchase)).collect(toList());
        System.out.println("1. " + sortYear + "\n");

System.out.println("*************************************************************************************** \n");

        //THIS IS FOR QUESTION 2
        //2. What are all the unique cities where the customers live?
        List<String> cities = newCustomerList.stream().map(transaction ->
transaction.getCity()).distinct().collect(toList());
        System.out.println("2. " + cities + "\n");

System.out.println("*************************************************************************************** \n");

        //THIS IS FOR QUESTION 3
        //3: Find all customers from Istanbul and sort them by name.
        List<Customer> customersList = newCustomerList.stream().filter(trader ->
trader.getCity().equals("Istanbul")).distinct().sorted(comparing(Customer::getName)).collect(
toList());
        System.out.println("3. " + customersList + "\n");

System.out.println("*************************************************************************************** \n");

        //THIS IS FOR QUESTION 4
        //4: Return a string of all customers' names sorted alphabetically.
        String namesSorted = newCustomerList.stream().map(transaction ->
transaction.getName()).distinct().sorted().reduce("Sorted:", (n1 , n2) -> n1 + " " + n2);
        System.out.println("4. " + namesSorted + "\n");

System.out.println("*************************************************************************************** \n");

        //THIS IS FOR QUESTION 5 [I used boolean type for this question]
        //5: Are any customers living in Ankara?
        boolean isLivingAnkara = newCustomerList.stream().anyMatch(transaction ->
transaction.getCity().equals("Ankara"));
        System.out.println("5. " + isLivingAnkara + "\n");

System.out.println("*********************************************************
************************ \n");

        //THIS IS FOR QUESTION 6
        //6: Print all transactions' values from the customers living in Istanbul.
        Predicate<Customer> Condition = customer -> customer.getCity().equals("Istanbul");

newCustomerList.stream().filter(Condition).sorted(comparing(Customer::getPurchase)).forEa
ch(customer -> System.out.println("6. " + customer + "\n"));

System.out.println("*********************************************************
************************ \n");

        //THIS IS FOR QUESTION 7
        //7: What's the highest value of all the transactions?
        int highestValue = newCustomerList.stream().map(Customer::getPurchase).reduce(0,
Integer::max);
        System.out.println("7. Max Value: " + highestValue + "\n");

System.out.println("*********************************************************
************************ \n");

        //THIS IS FOR QUESTION 8
        //8. Find the transaction with the smallest value.
        Customer smallestValue =
newCustomerList.stream().min(comparing(Customer::getPurchase)).orElseThrow(NoSuchEle
mentException::new);
//                      .map(Transaction::getValue)
//                      .reduce(0, Integer::min);
        System.out.println("8. " + smallestValue + "\n");

System.out.println("*********************************************************
************************ \n");

        //THIS IS FOR QUESTION 9 [Is there any value less than 5 here? I checked it. If there
is, it prints the element.]
        //9. Is there any transaction less than a certain value?
        Customer isCertainValue = newCustomerList.stream().filter(customer ->
customer.getPurchase() < 5).reduce((a, b) -> { throw new IllegalStateException("Multiple
elements: " + a + ", " + b); }).get();
        System.out.println("9. " + isCertainValue + "\n");

System.out.println("*********************************************************
************************ \n");
```

```
        //THIS IS FOR QUESTION 10
        Predicate<Customer> mypred = new Filter<>();
        Stream<Object> str2 = newCustomerList.stream().filter(mypred).map(d->
d.getCity().equals("Ankara"));
//        System.out.println("10. " + str2 + "\n");
    }
}
```

FILTER CLASS;

```
import java.util.function.Predicate;

//@AUTHOR: GORKEM TOPRAK
//DATE: JANUARY 23, 2021 SATURDAY
//TOPIC: JAVA-8 STREAMS

class Filter<T> implements Predicate<T> {
    @Override
    public boolean test(T t) {
        return true;
    }
}
```

FILTER INTERFACE;

```
//@AUTHOR: GORKEM TOPRAK
//DATE: JANUARY 23, 2021 SATURDAY
//TOPIC: JAVA-8 STREAMS

@FunctionalInterface
interface MyPredicate<T>{
    boolean mytest(T arg);
}
```