```
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Arrays;

public class Image {
    BufferedImage image;

    //------------------------constructor---------

    public  Image(String filename){
        try{
            this.image = ImageIO.read(new File(filename));
        }catch (IOException e){
            System.out.println("error" + e);
        }
    }

    public Image ( BufferedImage img){
        this.image = img;
    }

    public Image(int[][]A ){ // grey scale image form array
        image = new BufferedImage(A.length, A[0].length,BufferedImage.TYPE_BYTE_GRAY);
        for (int x =0 ; x< image.getWidth(); x++){
            for(int y =0 ; y < image.getHeight(); y++){
                Color newColour = new Color(A[x][y],A[x][y],A[x][y]);
                image.setRGB(x,y,newColour.getRGB());
            }
        }

    }

    //----------------------basic io function----------------


    public BufferedImage getImage() {
        return this.image;
    }

    void saveToFile(String filename , String extension){
        try{
            ImageIO.write(image,"jpg",new File(filename+"."+extension));
        }catch (IOException e){
            System.out.println("error : "+ e);
        }
    }

    public  static  void saveToFile(int [][]f, String filename , String extension){
        Image im = new Image(f);
        im.saveToFile(filename,extension);
    }
```

```
public static void  displayPixelsIntensity(int[][] f){

    for(int x = 0 ; x <f.length ; x++){
        for(int y = 0; y < f[0].length ; y++){
            System.out.print(f[x][y]+",");
        }
        System.out.println();
    }
}

void display (String title){
    ImageIcon icon = new ImageIcon(this.image);
    JFrame frame = new JFrame(title);

    frame.setLayout(new FlowLayout());
    frame.setSize(this.image.getWidth(),this.image.getHeight());
    JLabel lbl = new JLabel();
    lbl.setIcon(icon);
    frame.add(lbl);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public  static  void display (int [][]f , String title){

    //clip intensities to the range [0,255]

    for(int x = 0 ; x <f.length ; x++){
        for(int y = 0; y < f[0].length ; y++){
            if (f[x][y] > 255){
                f[x][y] =255;
            }
            if (f[x][y] < 0 ){
                f[x][y] = 0;
            }
        }
    }
    Image img = new Image(f);
    img.display(title);
}

public  static  void display (double [][]f , String title){

    //clip intensities to the range [0,255]

    int [][] F = new int[f.length][f[0].length];

    for(int x = 0 ; x <f.length ; x++){
        for(int y = 0; y < f[0].length ; y++){
            F[x][y] = (int) Math.round(f[x][y]);

            if (F[x][y] > 255){
                F[x][y] =255;
            }
```

```
            if (F[x][y] < 0 ){
                F[x][y] = 0;
            }
        }
    }

    Image img = new Image(F);
    img.display(title);
}

int [][] getPixelsArray(){
    int [][] A = new int[image.getWidth()][image.getHeight()];
    for (int x =0 ; x< image.getWidth(); x++){
        for(int y =0 ; y < image.getHeight(); y++){
            Color c = new Color(image.getRGB(x,y));
            A[x][y] = (int) ( c.getRed() + c.getBlue() + c.getGreen())/3;
        }
    }
    return A;
}


//=======================extra functions==============================

int[][] logTransformArray (int[][] inputArray, int scale){
    int [][]finalArray = new int[inputArray.length][inputArray[0].length];
    int max = 255;
    for(int x = 0 ; x <inputArray.length ; x++){
        for(int y = 0; y < inputArray[0].length ; y++){

            finalArray[x][y] = scale * (int) ( max * Math.log(1+inputArray[x][y])/Math.log(1+max));
        }
    }
    return finalArray;
}

int[][] histogramArray(int[][] grayImageArray){
    double maxIntensity = 255;
    int [][] finalArray = new int [grayImageArray.length][grayImageArray[0].length];
    int[] transformedIntensity = new int [256];
    int[]intensityCount = new int [256];
    double[] probabilityCount = new double [256];

    //counting the number of pixels that have certain intensity
    for(int x =0 ; x< grayImageArray.length ; x++){
        for (int y = 0; y< grayImageArray[0].length;y++){
            intensityCount[grayImageArray[x][y]]++;
        }
    }

    //calculating  probability
    for(int x = 0; x< probabilityCount.length; x++){
        probabilityCount[x]= (double) intensityCount[x]/(grayImageArray.length*
grayImageArray[0].length);
    }
```

```
    //redefining intensity
    double sumOfPrev =0;
    for(int x = 0; x< transformedIntensity.length ; x++){
        sumOfPrev = sumOfPrev + probabilityCount[x];
        transformedIntensity[x]= (int)(maxIntensity * sumOfPrev);
    }

    for(int x =0 ; x< finalArray.length ; x++){
        for (int y = 0; y< finalArray[0].length;y++){
            finalArray[x][y] = transformedIntensity[grayImageArray[x][y]];
        }
    }
    return finalArray;
}

//==================for laplacian transform ==========================
int[][] getLaplacianKernel(){
    return new int[][]{{0,1,0},{1,-4,1},{0,1,0}};
}

int[][] convolutionKernel(int[][]kernel){
    int[][] finalArray = new int [kernel.length][kernel[0].length];
    for(int x =0 ;x< kernel.length ; x++ ){
        for(int y=0; y<kernel[0].length;y++){
            finalArray[x][y] = kernel[kernel.length -1 -x][kernel[0].length-1-y];
        }
    }
    return finalArray;
}

int[][] getPadded2DSection(int[][]imageArray, int xCord, int yCord , int sizeOfKernel){
    int[][] finalArray = new int[sizeOfKernel][sizeOfKernel];

    int x = xCord - (sizeOfKernel /2);

    for(int i = 0; i< sizeOfKernel; i++){
        int y = yCord - (sizeOfKernel /2);
        for(int j = 0; j< sizeOfKernel; j++){
            if(x<0 || y< 0 || x>=imageArray.length || y>= imageArray[0].length){
                finalArray[i][j]=0;
            }else {
                finalArray[i][j] = imageArray[x][y];
            }
            y++;
        }
        x++;
    }
    return finalArray;
}

int  sumOfDotMatrix(int[][] sectionArray,int [][] kernel){
    int sum =0;
    for(int x=0;x<sectionArray.length;x++){
        for(int y=0;y<sectionArray[0].length;y++){
```

```
            sum = sum + (sectionArray[x][y]*kernel[x][y]);
        }
    }
    return  sum;
}

public  int[][] correlationConvolutionTransform(int[][] imageArray,int[][]kernel){
    int[][] unscaledArray = new int[imageArray.length][imageArray[0].length];
    int[][] finalArray = new int[imageArray.length][imageArray[0].length];
    int[][] sectionArray ;

    for(int x=0; x< imageArray.length ; x++){
        for(int y=0; y< imageArray[0].length;y++){
          sectionArray = getPadded2DSection(imageArray,x,y, kernel.length);
          unscaledArray[x][y]= sumOfDotMatrix(sectionArray,kernel);
        }
    }

    //to find max and min intensity
    int maxIntensity = -999999999 ;
    int minIntensity =  999999999;
    for(int x =0; x< unscaledArray.length;x++){
        for(int y=0; y<unscaledArray[0].length;y++){
            if(maxIntensity<unscaledArray[x][y]){
                maxIntensity=unscaledArray[x][y];
            }
            if(minIntensity>unscaledArray[x][y]){
                minIntensity = unscaledArray[x][y];
            }
        }
    }
    //rescaling the intensity
    for(int x =0; x< unscaledArray.length;x++){
        for(int y=0; y<unscaledArray[0].length;y++){
            finalArray[x][y] =  255 * (unscaledArray[x][y]-minIntensity)/maxIntensity;
        }
    }
    return finalArray;
}

//====================for enhancement ===============


public  int getEnhancementConstantScaling(int[] [] kernel){
    /*

    where c = -1 if we take a Laplacian kernel with negative center, and c = 1 if we take a Laplacian
kernel with positive center.
     */
    int centerRow = (kernel.length /2)+1;
    int centerColm = (kernel[0].length /2)+1;
    if(kernel[centerRow][centerColm] < 0 ){
        return -1;
    }else {
        return 1;
```

```
        }
    }

    public  int[][] imageLaplacianEnhancement(int[][]grayImageArray, int[][]correlationArray,int scale){

        /*
        The value of image Laplacian is higher where there is sharp transition in intensity, i.e. where the image is more detailed.
         Thus, addition of the Laplacian to the original image enhances the details in an image.
         */
        int [][] finalArray = new int [grayImageArray.length][grayImageArray[0].length];

        for(int x =0; x< grayImageArray.length ; x++){
            for(int y=0 ; y< grayImageArray[0].length;y++){
                finalArray[x][y] = (grayImageArray[x][y]+ scale * correlationArray[x][y])%255 ;
            }
        }
        return  finalArray;
    }


    //====================for averaging and bluing image==============

    double[][] getAverageKernel(){
        double[][] kernel = new double[3][3];
        for(int x =0 ;x<3;x++){
            for(int y=0;y<3;y++){
                kernel[x][y]= (double) 1/9;
            }
        }
        return kernel;
    }

    int  sumOfDotMatrixForAverage(int[][] sectionArray,double [][] kernel){
        double sum =0;
        for(int x=0;x<sectionArray.length;x++){
            for(int y=0;y<sectionArray[0].length;y++){
                sum = sum + (sectionArray[x][y]*kernel[x][y]);
            }
        }
        return (int) sum ;
    }

    public  int[][] averageTransform(int[][] imageArray,double[][]kernel){
        int[][] unscaledArray = new int[imageArray.length][imageArray[0].length];
        int[][] finalArray = new int[imageArray.length][imageArray[0].length];
        int[][] sectionArray ;

        for(int x=0; x< imageArray.length ; x++){
            for(int y=0; y< imageArray[0].length;y++){
                sectionArray = getPadded2DSection(imageArray,x,y, kernel.length);
                unscaledArray[x][y]= sumOfDotMatrixForAverage(sectionArray,kernel);
            }
        }
```

```
        //to find max intensity
        int maxIntensity = -999999999 ;
        for(int x =0; x< unscaledArray.length;x++){
            for(int y=0; y<unscaledArray[0].length;y++){
                if(maxIntensity<unscaledArray[x][y]){
                    maxIntensity=unscaledArray[x][y];
                }
            }
        }
        //rescaling the intensity
        for(int x =0; x< unscaledArray.length;x++){
            for(int y=0; y<unscaledArray[0].length;y++){
                finalArray[x][y] = 255 * unscaledArray[x][y]/maxIntensity;
            }
        }
        return finalArray;
    }

    //====================== image gradient using Sobel's masks==============
    public int[][] getHorizontalKernel(){
        return new int[][]{{-1,2,-1},{0,0,0},{1,2,1}};
    }
    public int[][] getVerticalKernel(){
        return new int[][]{{-1,0,1},{-2,0,2},{-1,0,1}};
    }

    public int[][] getMagnitudeOfSobelOperator(int[][] grayImageArray,int[][]horizontalKernel,int[][]
verticalKernel){
        int [][] finalArray = new int[grayImageArray.length][grayImageArray[0].length];
        int[][] sectionArray;
        int magnitudeX , magnitudeY;
        for(int x =0; x<grayImageArray.length;x++){
            for(int y =0 ; y<grayImageArray[0].length;y++){
                sectionArray = getPadded2DSection(grayImageArray ,x,y,horizontalKernel.length);
                magnitudeX = sumOfDotMatrix(sectionArray,horizontalKernel);
                magnitudeY = sumOfDotMatrix(sectionArray,verticalKernel);
                finalArray[x][y]= (int) Math.sqrt(magnitudeX * magnitudeX + magnitudeY * magnitudeY);
            }
        }
        return finalArray;
    }

    public static int getMedian(int [][]array ){
        // how we do is we flatten 1d array into 2d and sort the array
        //then if odd number of elements are odd then value in length/2 is median
        //if even we take out average of 2

        int [] oneDArray = new int[array.length * array[0].length];
        int index = 0;
        for (int[] ints : array) {
            for (int y = 0; y < array[0].length; y++) {
                oneDArray[index] = ints[y];
                index++;
            }
        }
```

```
        Arrays.sort(oneDArray);

        if(oneDArray.length % 2 == 0){//is even
            int num1 = oneDArray[(oneDArray.length-1)/2];
            int num2 = oneDArray[((oneDArray.length-1)/2)+1];
            return (num1+num2)/2;
        }
        else {//is odd
            return oneDArray[oneDArray.length/2];
        }
    }

    int [][] binaryImagePixelsArray (int[][]inputArray , int threshold){

        //if the intensity is greater than  threshold up it to 255 else downgrade it to 0

        int [][] finalArray = new int[inputArray.length][inputArray[0].length];

        for(int x = 0 ; x <inputArray.length ; x++){
            for(int y = 0; y < inputArray[0].length ; y++){
                if (inputArray[x][y] > threshold){
                    finalArray[x][y] =255;
                }else{
                    finalArray[x][y] = 0;
                }
            }
        }
        return finalArray;
    }

}


//if f[][] then f.length is number of rows and f[0].length is number of columns
//255 is white 0 is black




-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------

-----------------------------------------------------------------------------------

-----------------------------------------------------------------------------------

-----------------------------------------------------------------------------------
public class Main {
    public static void main(String[] args) {

        Image img = new Image("D:\\drive\\OneDrive - Everest Engineering College\\7th sem\\Image
Processing and Pattern Recognization\\labs\\lab2 image Processing\\062.png" );
        int[][] grayImageArray = img.getPixelsArray();
```

```
        Image.display(grayImageArray,"Original");
        Image.saveToFile(grayImageArray,"1 grayScaled","jpg");

        System.out.println("Grey scaling complete ");

        int[][] arrayLogTransform = img.logTransformArray(grayImageArray,1);
        Image.display(arrayLogTransform,"log transform image");
        Image.saveToFile(arrayLogTransform,"2 log transform image","jpg");

        System.out.println("Log transform complete ");

        int[][] arrayHistogram= img.histogramArray(grayImageArray);
        Image.display(arrayHistogram,"Histogram equalization image");
        Image.saveToFile(arrayHistogram,"3 Histogram equalization image","jpg");

        System.out.println("Histogram equalization complete ");


        int[][] kernel = img.getLaplacianKernel();
        int[][] correlationArray = img.correlationConvolutionTransform(grayImageArray,kernel);
        Image.display(correlationArray,"correlation image");
        Image.saveToFile(correlationArray,"4 correlation image","jpg");

        System.out.println("Laplacian complete ");

        int enhancementConstantScaling = img.getEnhancementConstantScaling(kernel);
        int[][]enhancedImage
=img.imageLaplacianEnhancement(grayImageArray,correlationArray,enhancementConstantScaling);
        Image.display(enhancedImage,"enhanced image");
        Image.saveToFile(enhancedImage,"5 enhanced image","jpg");

        System.out.println("Laplacian enhancement complete ");

        double[][] blurKernel = img.getAverageKernel();
        int[][] blurArray = img.averageTransform(grayImageArray,blurKernel);
        Image.display(blurArray,"blurring image ");
        Image.saveToFile(blurArray,"6 blurring image","jpg");

        System.out.println("Image blurring complete ");


        int[][] horizontalKernel = img.getHorizontalKernel();
        int[][] verticalKernel = img.getVerticalKernel();
        int[][] magnitudeOfSobelOperator =
img.getMagnitudeOfSobelOperator(grayImageArray,horizontalKernel,
                verticalKernel);
        int median  = Image.getMedian(magnitudeOfSobelOperator);
        int[][] binaryImageMedian = img.binaryImagePixelsArray(magnitudeOfSobelOperator,median);
        Image.display(binaryImageMedian," image gradient using Sobel masks");
        Image.saveToFile(binaryImageMedian,"7 image gradient using Sobel masks","jpg");



    }
```

}