**package** ECTE331\_ProjA;

**import** java.awt.Color;

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.io.IOException;

**import** javax.imageio.ImageIO;

**public** **class** Aproj{

**public** **static** **void** main(String[] args) {

String fileName1= "C:\\Users\\Talal\\OneDrive\\Desktop\\ECTE331 Project A\\Rain\_Tree.jpg";

String fileName2= "C:\\Users\\Talal\\OneDrive\\Desktop\\ECTE331 Project A\\Wr.jpg";

colourImage ImgStruct= **new** colourImage();

imagereadwrite.readJpgImage(fileName1, ImgStruct); //here the image is 'Rain\_Tree' is read.

BufferedImage grey= convertTo(ImgStruct); //grey scaling

BufferedImage equalized= histogramequalize(grey); //histogram equalization

imagereadwrite.writeimage(equalized, fileName2); //new modified image file is saved here and renamed

}

//this section converts the image into grey scale

**public** **static** BufferedImage convertTo(colourImage imgStruct) {

**int** width= imgStruct.width;

**int** height= imgStruct.height;

**short**[][][] pixels= imgStruct.pixels;

BufferedImage greyimg= **new** BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

**for** (**int** y= 0; y < height; y++) {

**for** (**int** x= 0; x < width; x++) {

**short** red= pixels[y][x][0];

**short** green= pixels[y][x][1];

**short** blue= pixels[y][x][2];

// here the grey scale is calculated with the given requirements

**int** greys= ((red >> 16) & 255) + ((green >> 8) & 255) + (blue & 255);

**int** rgbgrey= (greys << 16) | (greys << 8) | greys;

// the calculated grey scale value is now applied to the new image

greyimg.setRGB(x, y, rgbgrey);

}

}

**return** greyimg;

}

// In this part, we apply equalization to the histogram

**public** **static** BufferedImage histogramequalize(BufferedImage image) {

**int** width= image.getWidth();

**int** height= image.getHeight();

**int**[] histogram= histogramcalc(image); // histogram calculation

**int**[] cumulativehistogram= calccumulativehist(histogram);

BufferedImage equalizedimg= **new** BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB); // histogram equalization is applied on the new image after creation

**int** totalpixels= width\*height;

**for** (**int** y= 0; y<height; y++) {

**for** (**int** x= 0; x<width; x++) {

**int** pixel= image.getRGB(x, y);

**int** greys= pixel & 255;

**int** equalizegrey= (**int**) (cumulativehistogram[greys]\*255.0 / totalpixels); // histogram equlization applied to grey scale value

**int** eqRGB= (equalizegrey << 16) | (equalizegrey << 8) | equalizegrey;

equalizedimg.setRGB(x, y, eqRGB); //new grey scaled pixel into the new image 'Wr'

}

}

**return** equalizedimg;

}

**private** **static** **int**[] histogramcalc(BufferedImage image) { //histogram calculation using utilization

**int**[] histogram= **new** **int**[256];

**int** width= image.getWidth();

**int** height= image.getHeight();

**for** (**int** i= 0; i < 256; i++) { //array for the histogram

histogram[i]= 0;

}

**for** (**int** y= 0; y < height; y++) { //histogram calculation

**for** (**int** x= 0; x < width; x++) {

**int** pixel= image.getRGB(x, y);

**int** greys= pixel & 255;

histogram[greys]++;

}

}

**return** histogram;

}

**private** **static** **int**[] calccumulativehist(**int**[] histogram) { //cumulative histogram calculation using utilization

**int**[] cumulativehistogram= **new** **int**[256];

cumulativehistogram[0]= histogram[0];

**for** (**int** i= 1; i < 256; i++) { // cumulative histogram calculation

cumulativehistogram[i]= cumulativehistogram[i - 1] + histogram[i];

}

**return** cumulativehistogram;

}

}

**class** imagereadwrite {

**public** **static** **void** readJpgImage(String fileName, colourImage ImgStruct) {

**try** {

File file= **new** File(fileName); //reading the image file

BufferedImage image= ImageIO.read(file);

System.out.println("File: " + file.getCanonicalPath());

**if** (!image.getColorModel().getColorSpace().isCS\_sRGB()) { // here it check if the image is in sRGB colour space

System.out.println("Image is not in sRGB Colour Space");

**return**;

}

// below gets the height and width of the image

**int** width= image.getWidth();

**int** height= image.getHeight();

ImgStruct.width= width;

ImgStruct.height= height;

ImgStruct.pixels= **new** **short**[height][width][3];

**for** (**int** y= 0; y < height; y++) { // loops each pixel, store RGB in the array

**for** (**int** x= 0; x < width; x++) {

// gets color of the current pixel

**int** pixel= image.getRGB(x, y);

Color color= **new** Color(pixel, **true**);

// stores the red, green, and blue colour pixel in the array

ImgStruct.pixels[y][x][0]= (**short**) color.getRed();

ImgStruct.pixels[y][x][1]= (**short**) color.getGreen();

ImgStruct.pixels[y][x][2]= (**short**) color.getBlue();

}

}

} **catch** (IOException e) {

System.out.println("Error Reading Image File: " + e.getMessage());

}

}

**public** **static** **void** writeimage(BufferedImage image, String fileName) {

**try** {

// buffer image written to the JPG file

File outputFile= **new** File(fileName);

ImageIO.write(image, "jpg", outputFile);

} **catch** (IOException e) {

System.out.println("Error Writing Image File: " + e.getMessage());

}

}

}

**class** matManipulation {

/\*\*

reshape a matrix to a 1-D vector

\*/

**public** **static** **void** mat2Vect(**short**[][] mat, **int** width, **int** height, **short**[] vect) {

**for** (**int** i= 0; i < height; i++) {

**for** (**int** j= 0; j < width; j++) {

vect[j + i\*width]= mat[i][j];

}

}

}

}

**class** colourImage {

/\*\*

A data structure to store a colour image

\*/

**public** **int** width;

**public** **int** height;

**public** **short** pixels[][][];

}