Project 1: Image Manipulation

1. Brief Overview of Assignment and Goals

The goal of this assignment was to create multiple filters to alter an image. The various filters include changing the shades of each image, reflecting the image, and changing the size of the image. This assignment wants to test our knowledge of basic programming. Each image is read in as an two dimensional array so that the code can individually manipulate each unit in the array (manipulate each pixel). Personal goal of this assignment was to refresh my memory of basic programming and Java.

2. Solution Design

A. NoRed, NoGreen, and NoBlue
These three methods are similar in implementation. ImageEffect has a method that lets the user control the specific RGB values for each pixel. So for each method, the color not wanted was replaced with 0 and the other values remained the same (like in the example code given in the assignment).

B. RedOnly, GreenOnly, and BlueOnly
These three methods are similar in implementation to the methods is part A. However, instead of only replacing one RGB value with zero, two values are replaced with zero, leaving only the desired shade remaining.

C. BlackAndWhite
Going off the fact the black is (0, 0, 0) and white it (255, 255, 255), it was evident that to create an image in grayscale, the RGB value would have to be (x, x, x), where x was the average of the red value, green value, and blue value in the image.

D. VerticalReflect and HorizontalReflect
Both methods are similar in nature as they are both creating a reflection of the original image. Since the methods are basically copying the first half (left half for VerticalReflect) to the second half of the image, a new 2D array of the same size is needed to be created so that no data of the original is lost.

VerticalReflect reflects the image over the vertical line down the middle of the image, so that for each pixel location, it’s row location does not change, but its column location flips to the other side (width minus original column location). However, Horizontal Reflect reflects the image over the horizontal line down the middle of the image so that for each location, it’s column location does not change, but its row location flips to the other side of the middle axis, becoming(height - original row location).
E. Grow and Shrink
The goals of Grow and Shrink were to change the size of the original image. Therefore, the first step in the solution was to create 2D arrays of the needed new size to fit the assignment specifications. Once with the correct size array, I went through each column and row to find the pattern for which pixel(s) corresponded with the new array.

Since Grow’s new array has 2 times the height and width, there will be four copies of each of the original pixel. The odd columns would be an exact copy of the even column directly to the left of it. For example, the 1st column is exactly like the 0th column. Therefore, it is only necessary, to figure out odd columns in this array and just have the odd column be a copy of the even column to the left of it. In each column, the odd rows will be a copy of the previous even row, so odd rows can just be a copy of the even row above it.

A similar methodology was applied to Shrink. With a new array half the height and original array, it is populated by containing the average value of four original pixels in one pixel. Tracing through the array would give you that for every new array[y][x] it would contain the average values of [2 * y][2 * x], [2 * y][2 * x + 1], [2 * y][2 * x], and [2 * y][2 * x]. Average values of the red shade, green shade, and blue shade were found individually to finally be put together to be the color the new pixel was supposed to be.

F. Threshold
Since Threshold asked for a parameter to be used, a constructor based off the Dummy class was created in order to add an ImageEffectParam (an ArrayList). The value the user chose could therefore be accessed through the parameter and be used to implement the code. Method itself wants only 8 colors in the solution, RGB values either at 0 or 255. Using if statements, method could test if the parameter threshold value is above an R, G, or B value, at which the shade was turned completely on (at 255). If the parameter threshold value read in was below, the shade/color would be turned off (at 0).

3. Scope and Quality of Solution, Interesting Results, Enhancements

A. NoRed, NoGreen, NoBlue, RedOnly, GreenOnly, and BlueOnly
Solution was thorough and does what is intended and no assumptions were made besides the fact it was operating on the basis of RGB values.

B. BlackAndWhite
Solution was pretty thorough and does what is intended. It could be better x in makePixel(x, x, x) was created in a more tested way. Here it was simply an average of the red value, blue value, and green value; perhaps, one value should be given more weight to create a better image but I currently I don’t know which without further testing.
C. VerticalReflect and HorizontalReflect
Solution was thorough and method does what is intended. It is however not the most efficient as it has to completely populate a new array and copy it over. It would be more efficient if it only had to populate half of a new array and just copy itself to the original array once the original array has copied one half to the other side.

D. Grow and Shrink
Solution for Grow was thorough and method does what is intended, creating an image twice the size and height of the original. However, for Shrink, the assumption that there would be an even number of rows and columns was made. If there are odd number of rows or columns, the last row and column of pixels would be lost. In addition, for Shrink, after a certain number of times of applying the method, the image would no longer shrink its width, but just its height, because the JIP constrained it to me a minimum width to fit in the words “File and Effects”.

E. Threshold
Solution was thorough as method does what is intended based off the assignment instructions. It creates an image of only 8 colors, making the image very bright and white if a low threshold value is chosen, and vice versa.

F. Problems Encountered
Some problems encountered initially when writing the code, was ArrayIndexOutOfBoundsException when creating new sized arrays and populating them, especially for Grow and Shrink.

G. Extra Enhancements
Erode and Dilate were created. They are both similar in nature. Erode expands the darker regions of the image, while dilate expands the lighter regions of the image. In both, the borders are excluded when changing the image as to simplify the code and avoid possible ArrayIndexOutOfBoundsException. Both were implemented similarly. For each pixel[y][x], a 3 x 3 neighborhood around it was identified and its red, green, and blue values were identified. For each color, the 9 values of each color were put into an Array to be sorted. Erode simply used the first values from each array sort for the new images color value, while Dilate used the last values from each array sort. It should be noted that there is a slight pinkish tint when it was tested on Dr. Lin’s face.

4. Software Test Methodology

A. Black Box Testing
Multiple images for provided for black box testing. Some images were more useful for some methods than others. For example, I found image 14 to be very useful in testing NoRed, NoGreen, NoBlue, RedOnly, GreenOnly, and BlueOnly as it had only the four basic colors in the image and I could clearly see which colors remained and which colors were removed.
Image 14 was also easy to test VerticalReflect and HorizontalReflect as the image is nicely separated into quadrants and I can see where each quadrant has been reflected to.

However, image 14 other tests such as Threshold since image 14 is already contain only 4 colors max. For the other methods I enjoyed using image 11 to test the rest of the methods including the “good karma” methods. (Grow and Shrink can be tested using any image)

It was interesting the test what the difference the threshold value did to each picture. At a lower threshold value, it was evident that more colors would be turned completely on, forming a
picture with many white pixels. However at a higher threshold value, more colors were turned off, forming a picture with more black pixels.

After Dilate (applied twice)*                   After Erode (applied twice)*
It was especially interesting to see Dilate and Erode compared directly to each other since they in effect accomplished opposite goals.

It should also be noted that it was useful to use rectangles to test the methods (e.g. images 3, 4, 5) to make sure there were no unnoticed ArrayIndexOutOfBoundsExceptions when testing with a square image.

*Effect applied to new image each time

B. White Box Testing
Some white box testing was used, but not as much as black box testing. For example, I used System.out.println() to print out array width and height of the original array and the array width and height of the new array created for Grow and Shrink to make sure I created the array of the desired size. However, these statements are now commented out.