CS 314H Honors Data Structures — Fall 2019 **Programming Assignment #1**
Image Manipulation **Due September 6, 2019**

In this assignment you will write code that manipulates digital images. We will provide routines that read images (in JPEG, PNG, or GIF formats) and write images (in JPEG or PNG formats), and you will write classes that alter the image in various ways. This assignment does not require you to write a lot of code, but it does encourage you to be creative in showing off the capabilities of your resulting tool, and it does require you to carefully explain what you've done and how you've done it.

1 Your Assignment

Digital images are stored on disks in a variety of different formats. Once you read them into your computer's main memory, you can manipulate these images in many interesting ways. The first part of your assignment is to implement the following effects.

- `NoRed`: Remove all red shades from the image.
- `NoGreen`: Remove all green shades from the image.
- `NoBlue`: Remove all blue shades from the image.
- `RedOnly`: Preserve any red shades in the image, but remove all green and blue.
- `GreenOnly`: Preserve any green shades in the image, but remove all red and blue.
- `BlueOnly`: Preserve any blue shades in the image, but remove all red and green.
- `BlackAndWhite`: Turn a color image into a black and white image. You should give some thought as to how you want to do this.
- `VerticalReflect`: Reflect the image around a vertical line down the middle of the original image. If the input image were a picture of your face, the output image would be what you see when you look in the mirror.
- `HorizontalReflect`: Reflect the image around a horizontal line across the middle of the original image. If the input image were a picture of your face, the output image would be an upside-down picture of your face.
- `Grow`: Create an image that is twice as high and twice as wide as your original picture. You should simply map one pixel in the original image to 4 pixels in the larger image.
- `Shrink`: Create an image that is half as high and half as wide as your original picture. The pixel values of the smaller image should be the average of four values in the original image.
- `Threshold`: Reduce the number of colors used in the image from millions to eight. In particular, set each of the R, G, and B values in the output image to either fully on (255) or fully off (0), depending on whether its value in the input image is above some threshold. This threshold value should be a parameter to this function, and a typical value is $\frac{1}{2}$ of the maximum value, or 127.

The second part of your assignment is to use your code to produce interesting images. You can use any input images that you like, but be sure to provide both "before" and "after" images, and be sure to let us know what effects you applied to produce the "after" image. Feel free to be creative. We would like to publish on the class web page some of the more interesting and artistic images that we get. Feel free to make collages, posters, or whatever.

The third part of your assignment is to describe what you've done and to describe your software in English prose. In addition to describing your code, you should explain how you tested your code. See Section **??** for more details.

2 Details

2.1 Representing Images

Realize that you will use two distinct representations of images: a `BufferedImage` and a 2D array of pixels.

The `BufferedImage` is how the image will be represented on disk. You do not need to know anything about the details of this representation. Instead, you will manipulate the image as a 2D array of pixels.

The 2D array of pixels is a convenient representation for manipulating the images. Each pixel consists of 3 components—a Red, Green, and Blue component—where each component is an integer between 0 and 255, with 0 indicating no intensity and 255 indicating the highest intensity. In reality, each pixel will be represented in your Java program as an integer; to access the 3 components, we provide the following three routines:

`int getRed(int pixel)`	Get the amount of red in the image.
`int getGreen(int pixel)`	Get the amount of green in the image.
`int getBlue(int pixel)`	Get the amount of blue in the image.

To create a pixel, use the following method to specify the Red, Green, and Blue components of a pixel:

```
int makePixel(int red, int green, int blue)
```

Create a pixel with the specified amount of red, green, and blue.

2.2 Your Code

We will provide the JIP (Java Image Processing) class, which is available from the class webpage. This class provides a GUI interface and handles all sorts of low-level details that you can ignore.

We will also provide an abstract class, the `ImageEffect` class. You should create a different subclass of the `ImageEffect` class for each effect that you want to create. As you can see from the definition of `ImageEffect`, each of your subclasses should implement an `apply()` method that takes a 2D array of pixels as input and produces a 2D array of pixels as output.

Below is an example of a class that inverts the value of each pixel in the image.

```
class Invert extends ImageEffect
{
    public int[][] apply(int[][] pixels)
    {
        int width = pixels[0].length;
        int height = pixels.length;

        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                pixels[y][x] = ~pixels[y][x];
            }
        }
        return pixels;
    }
}
```

As another example, if you wanted to remove the red pigment from the pixel in the fourth row and the fifth column, your `apply()` method could include the following line of code:

```
public int[][] apply(int[][] pixels)
{
    pixels[3][4] = makePixel(0, getGreen(pixels[3][4]), getBlue(pixels[3][4]));
}
```

This class removes red from (3,4). Note that in this 2D array of pixels, the row number is given first, followed by the column number, and note that arrays are indexed starting at 0.

2.3 Naming Conventions

The names of your subclasses should match the names of the effects as given above. Thus, the `VerticalReflect` subclass should implement the `VerticalReflect` effect.

The file `Transformations.java` is provided to get you started. Take a look.

2.4 Interfaces

Your code should not alter the interfaces that we provide. If you believe that you have a better interface than what we provide, you may explain the changes that you would make in your report, along with your argument for its superiority. (If you want to use your new interface in your code, come see the instructors to see how you can do so while respecting the provided interfaces.)

2.5 Programming Environments

The CS labs on the first and third floors of GDC have a number of development environments, such as CodeWarrior, BlueJ, and Eclipse. You may use whatever you want.

2.6 Testing Your Software

Please think carefully about how you will test your code. To assist you, we will provide set of images that you may use to perform **black box testing**. Black box testing treats the software as a black box and determines correctness by observing what inputs produce what outputs. The use of these supplied images by themselves does not necessarily constitute an effective test strategy, so you are free to use other images, and in general we encourage you to be creative in finding ways to test your code.

To further stimulate your thought process, here are just some of the questions that you might consider. Feel free to discuss these and other questions as you write your report, focusing of course on your most important observations.

- For some effects, such as `NoRed()`, some of the images may be more useful than others. For a given effect, can you identify useful and non-useful images?
- You might also consider asking the question the other way: For a given image, what effects is it useful for testing? What effects is it not useful for testing?
- If you were to apply your test methodology to someone else's code, how confident would you be that their code is correct? Can you think of other testing approaches that would increase your confidence? (If you have some good ideas, it might be good to discuss these ideas in your report even if you don't have time to execute them.)

3 Your Report

You should write a report that explains what you've done and how you did it. Remember that the report is likely the first thing that your audience will read.

Your report should have four sections: (1) a brief overview of the assignment and your goals in completing this assignment, (2) a description of the solution design, including assumptions made, algorithms used, etc., (3) a discussion of the completed assignment, including scope of solution, quality of solution, interesting results, problems encountered, enhancements/extra credit completed, etc., and (4) a discussion of your software test methodology. This third section is a good place to point out any interesting insights that you learned or surprises that you found.

The third section is important, as it asks you to evaluate your software and its behavior. Here are some questions to ask yourself when thinking about your software.

- What does your code do? For example, how does your code turn a color image into a black and white image?
- What assumptions do you make in your code? Do you assume that the input image is a square? A power of 2? What special cases have you identified? How do you handle these special cases? Are there any special cases that you do not address?

Your report does not have to be long. On the contrary, you should strive for conciseness and clarity in your writing. We will give you feedback on your reports over the course of the semester. One general suggestion: Focus on your most important points, rather than trying to exhaustively describe every detail.
As for all assignments in this class, you should clearly acknowledge any submitted code that is not your own.

4 Good Karma

If you'd like an extra challenge, implement some of the effects below. You can get an A in the course without doing any of the extra problems, but if you're not pushing yourself or learning anything, then you're wasting your time, and that would be a shame.

Though you can create a wide variety of effects where each pixel in the original image contributes to just one pixel in the final image, more interesting effects can be achieved if each original pixel contributes to many final pixels. One class of such effects are called *neighborhood filters*. A neighborhood filter bases each final pixel on a group, or "neighborhood", of original pixels. For example, each final pixel could be the median pixel value among the corresponding original pixel and its eight adjacent pixels. The shape and size of the neighborhood are variable, and the function utilizing the neighborhood pixels is limited only by the imagination.

One special class of neighborhood filter is a *convolution*. The neighborhood function of a convolution is a linear combination of the pixel values in the neighborhood. For example, each final pixel could be the average pixel value of the corresponding original pixel and its eight adjacent pixels. By adjusting the coefficients of the linear function, many different effects can be created. The following are some common neighborhood filters and convolutions.

- Smooth: Replace each pixel with the average pixel value of a small neighborhood (3×3 grid, 5×5 grid, etc., you're free to choose).
- Sharpen: Light pixels near dark pixels should be made lighter, and dark pixels near light pixels should be made darker. This effect will highlight neighborhoods of high contrast.
- Highlight edges: Each pixel should be replaced with a light pixel if it came from a neighborhood of high contrast (an edge), or replaced with a dark pixel if it came from a neighborhood of low contrast.
- De-noise: Replace each pixel with the median pixel value of a small neighborhood.
- Erode: Replace each pixel with the minimum pixel value of a small neighborhood. This effect expands the darker regions of an image, "eroding" the edges of lighter regions.
- Dilate: Replace each pixel with the maximum pixel value of a small neighborhood. This effect expands the lighter regions of an image.

Of course, there are many other image effects that you could implement. For example, you might implement *transparency*, in which a parameter would specify the degree of transparency, with 0% being completely opaque and 100% being completely transparent. Feel free to be creative as you think of new effects to implement, and feel free to do your own research to figure out how to implement such effects.

5 What To Turn In

Use the **Canvas** website to submit a single ZIP file that contains your report, your source code, and optionally any links to images you used. Canvas is available at `http://canvas.utexas.edu/`—the TA will post the directions on Piazza in a few days.

Important Details.

- Canvas has rather small quotas, so your submissions should *not* include image files. For karma, make your interesting images accessible on the web and include the URL in your report. Make available both the "before" and "after" images, and describe the transformations needed to produce the final image.
- As with all reports for this class, the electronic version should be in Adobe PDF format. Please do **not** submit Word files.
- The .zip file should be named **progN.zip**, where N is the number of the assignment.
- This assignment is due at **5:00pm** on the due date. Late assignments will be penalized 10% per day.

Acknowledgments. We thank Ashlie Martinez, Arthur Peters, and Josh Eversmann for their revisions of this assignment.