Assignment 5: Image Processing (Part 1)

Due Jun 11 by 8:59pm **Points** 0

IME: Image Manipulation and Enhancement

Starter files: code

1 Introduction

Many applications use color images. A good number of these applications provide a way to change their appearance in different ways. For example, Instagram has "filters" that convert a picture into something more interesting. They do this by editing the colors of individual dots in the image (called pixels).

1.1 Images

Whatever the file format (jpg, png, bmp, etc.) an image is basically a sequence of pixels. Each pixel has a position in the image (row, column) and a color. For a color image, the pixel's color is stored using three numbers: red, green, blue. Any color is a combination of these three *base* colors. To see how various colors can be made using these base colors: open Intellij -> Type "Ctrl+Shift+A" on Windows or "Command+Shift+A" on Mac and type "show Color Picker". When selected, you can pick different colors and observe their red, green and blue values to get an intuition for how this works).

Each of these colors is called a "channel" in the image (i.e. a color image has 3 channels, ignoring transparent images). Each channel is usually stored using 8-bits, that is 256 distinct values. This is where the name "24-bit image" comes from (3 8-bit channels)!

1.2 Image Processing

The ubiquity of images is matched by the plethora of image manipulation programs and services. All phones with a camera also include apps that allow us to brighten, darken, blur or crop photos. Instagram has "filters" that convert a picture into something more interesting. Programs on traditional computers range from very simple photo manipulators to sophisticated Photoshop-like applications. The technical field of image processing itself is deeply rooted in math and signal processing theory. But the basics of image representation and manipulation are quite simple to understand.

In this series of assignments, you will build an image processing application. You will create text-based and interactive GUI-based user interfaces for this program. Although you will only implement some of the above effects, your system will have "good bones" through design that will streamline adding more image manipulations.

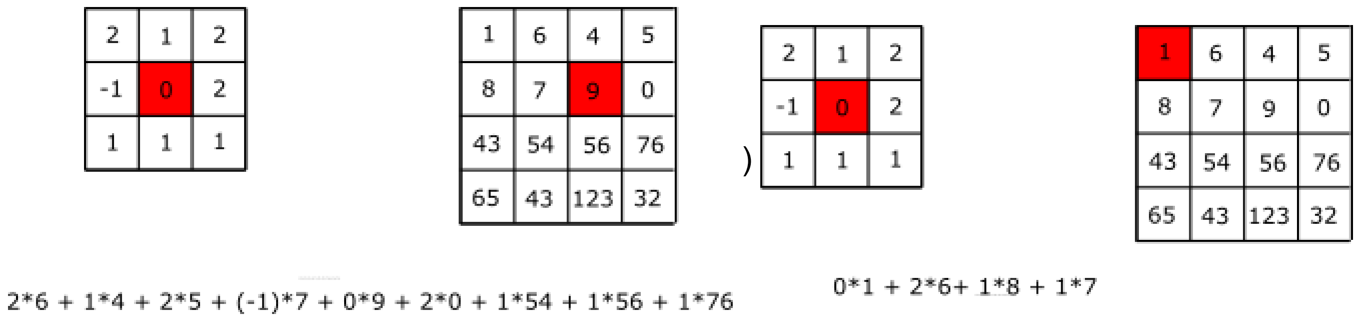
1.3 Sample Image Application

If you have not worked with image processing applications before, we recommend Gimp(http://www.gimp.org). Gimp is a free, open-source powerful image processing application. If you have access to Photoshop then that will be enough as well. We encourage you to find and try out the image processing operations that we will introduce in these assignments using these applications, to understand them better.

2 Example Image Processing: Filtering

A basic operation in many image processing algorithms is filtering. A filter has a "kernel," which is a 2D array of numbers, having odd dimensions (3x3, 5x5, etc.). Given a pixel in the image and a channel, the result of the filter can be computed for that pixel and channel.

As an example, consider a 3x3 kernel being applied for the pixel at (5,4) of the red channel of an image(row 5, column 4). We place the center of the kernel at the particular pixel (e.g. kernel[1][1] overlaps pixel (5,4) in channel 0. This aligns each number in the kernel with a corresponding number in that channel (kernel[0][0] aligns with pixel (4,3), kernel[1][2] aligns with pixel(5,5), etc.). The result of the filter is calculated by multiplying together corresponding numbers in the kernel and the pixels and adding them. If some portions of the kernel do not overlap any pixels, those pixels are not included in the computation. The picture below illustrates the filter operation (not using the same numbers as above).



2.1 Applications of filtering

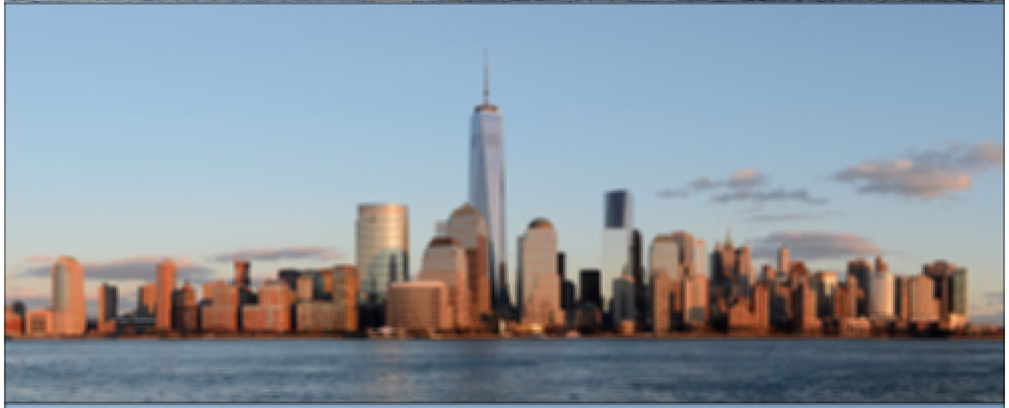
Many image processing operations can be framed in terms of filtering (i.e. they are a matter of designing an appropriate kernel and filtering the image with it).

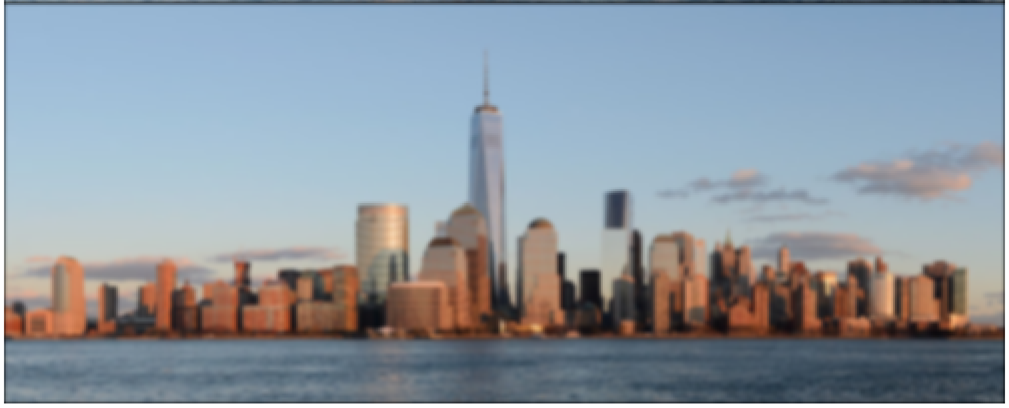
2.1.1 Image blur

Blurring an image is probably the simplest example of filtering. We can use a 3x3 filter as follows:

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

Blurring can be done by applying this filter to every channel of every pixel to produce the output image. Here is an example of blurring an image this way (in order: original image, blurred, blurred again):





)

The filter can be repeatedly applied to an image to blur it further. The last image above shows the result of applying the same filter to the second image.

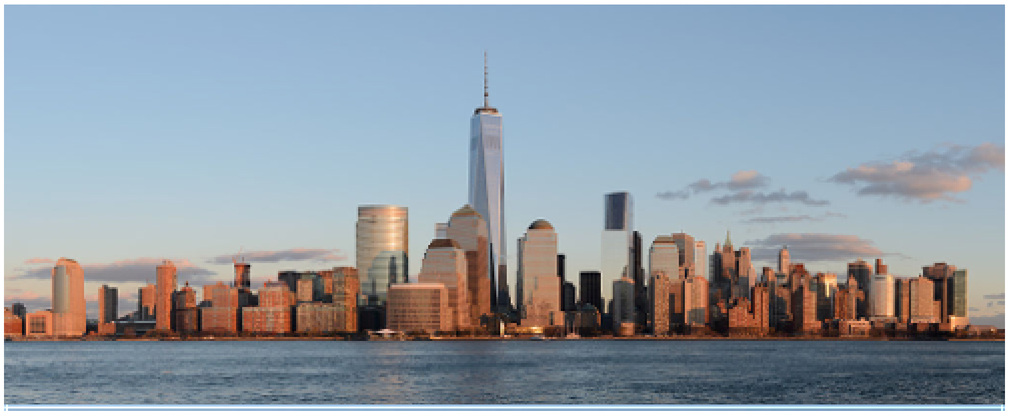
2.1.2 Image sharpening

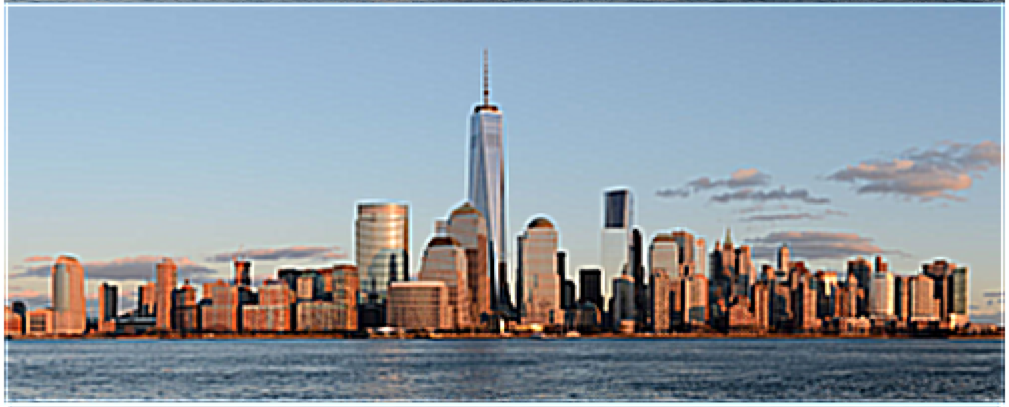
Image sharpening is in some ways, the opposite of blurring. Sharpening accentuates edges (the boundaries between regions of high contrast), thereby giving the image a "sharper" look.

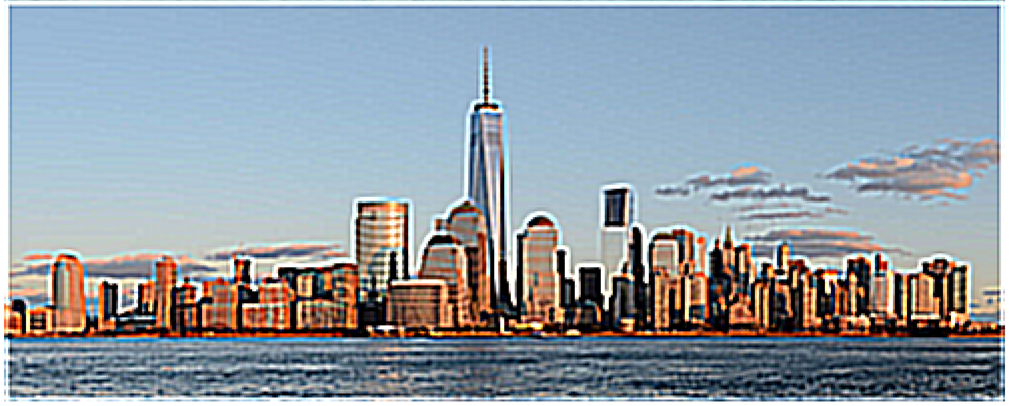
Sharpening is done by using the following filter:

$$\begin{bmatrix} \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{1}{4} & 1 & \frac{1}{4} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} & \frac{-1}{8} \end{bmatrix}$$

As with blurring, it is possible to repeatedly sharpen the image. Here is an example of sharpening an image (in order: original image, sharpened, sharpened more):







2.2 Clamping values

Often filtering results in an image where some values are beyond their range. For example, using 8 bits per channel means each value in each channel is between 0 and 255. Filtering such an image may cause some values to be outside this range. We must "clamp" these values to avoid overflow and underflow while saving, and to display such images properly. Clamping requires two values: the permissible minimum and maximum. Then each value in an image that is lesser than the minimum is assigned to the minimum, and each value greater than the maximum is assigned to the maximum.

Clamping is implemented as the last step of any filtering operation, to ensure that the resulting image can be properly saved and displayed.

3 Color transformations

Filtering modifies the value of a pixel depending on the values of its neighbors. Filtering is applied separately to each channel. In contrast, a color transformation modifies the color of a pixel based on its own color. Consider a pixel with color (r,g,b). A color transformation results in the new color of this pixel to be (r',g',b') such that each of them are dependent only on the values (r,g,b).

A simple example of a color transformation is a linear color transformation. A linear color transformation is simply a color transformation in which the final red, green and blue values of a pixel are linear combinations of its initial red, green and blue values. For example, if the initial color is (r,g,b), then the final red value being $0.3r + 0.4g + 0.6b$ is a linear color transformation.

Linear color transformations can be succinctly represented in matrix form as follows:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

where a_{ij} are numbers. Using the above notation, the final values will be:

$$r' = a_{11}r + a_{12}g + a_{13}b \; g' = a_{21}r + a_{22}g + a_{23}b \; b' = a_{31}r + a_{32}g + a_{33}b$$

Similar to filtering, clamping may be necessary after a color transformation to save and display an image properly.

3.1 Applications of color transformations

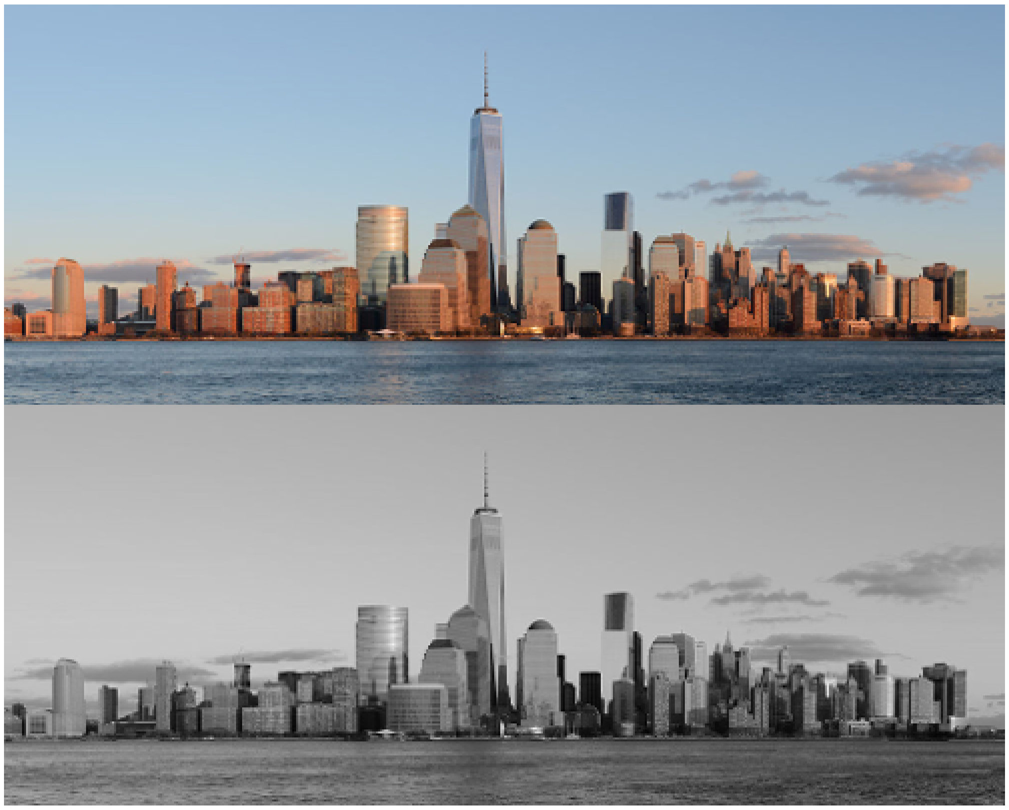
There are many image processing operations that can be expressed as color transformations on individual pixels (i.e. implementing them is a matter of using the correct numbers a_{ij} on all pixels of the image).

3.1.1 Greyscale

A simple operation is to convert a color image into a greyscale image. A greyscale image is composed only of shades of grey (if the red, green and blue values are the same, it is a shade of grey). There are many ways to convert a color image into greyscale. A simple way is to use the following color transformation: $r' = g' = b' = 0.2126r + 0.7152g + 0.0722b$. This is a standard formula to compute the "luma" for high-definition television (**read more about it here (https://en.wikipedia.org/wiki/Rec._709)**). This can be expressed as a color transformation in matrix form as:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.2126 & 0.7152 & 0.0722 \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

The effect of this conversion is illustrated below:

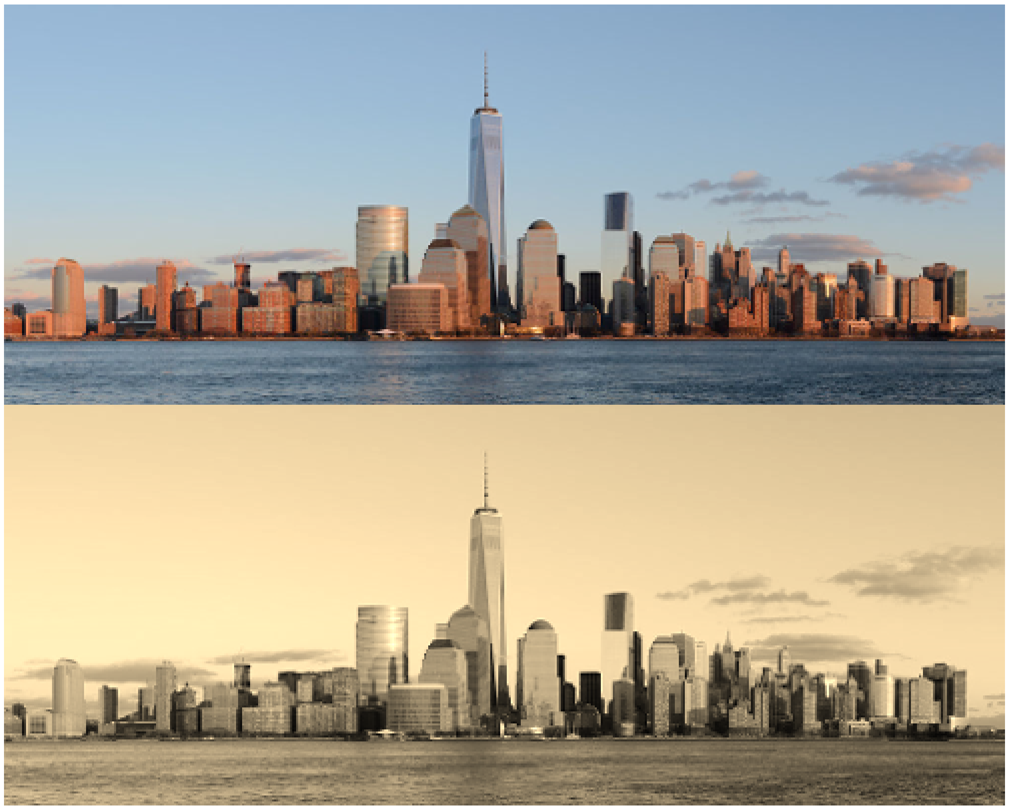


3.1.2 Sepia tone

Photographs taken in the 19th and early 20th century had a characteristic reddish brown tone. This is referred to as a sepia tone (see the **origin of this term (https://en.wikipedia.org/wiki/Sepia_(color))**). Most image processing programs allow an operation to convert a normal color image into a sepia-toned image. This conversion can be done using the following color transformation:

$$\begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

The effect of this conversion is illustrated below (original image, sepia-toned result):



4 Creating Images

In order to work with images, you have two options: create images programmatically (an image is basically pixel data), or load images stored in the simple PPM format. We will add support for other file formats later: keep this in mind as you incorporate this in your design.

4.1 Creating images programmatically

Many images can be created programmatically. For example, a checkerboard pattern can be create quite easily by writing a program for it. A reasonable way to create a checkerboard image would be specify the size of each square tile, the number of tiles in the checkerboard, and possibly the colors to use.

Another example of a programmatic image is a rainbow.

4.2 The PPM image format

The PPM format is a simple, text-based file format to store images. It basically contains a dump of the red, green and blue values of each pixel, row-wise.

Some code has been provided to you to read a PPM file. You may use and manipulate this code in any way you see fit.

5 What to do

In this assignment, you must design the model of your application. Your work in this assignment should also include any relevant data representations, classes and interfaces. Specifically, your application must have :

- A suitable representation of images
- Support for blurring and sharpening images
- Support for creating monochrome and sepia images
- Support for creating images programmatically, at least a configurable checkerboard image.
- Support for importing and exporting images in PPM format

Here are some aspects to think about:

- What does the model represent?
- What are possible ways to represent image data? Which ones seem more helpful for this application?
- Although you are implementing two specific operations in this assignment, you will likely implement others in subsequent assignments. The same goes for different ways to create images programmatically.
- Remember to think from not only the implementor's perspective (the person that is implementing the model) but also the client perspective (the people or classes that are using the model). What operations might they reasonably want to perform? What observations might they reasonably want to make?
- Remember not to fall into the trap of assuming that there is only one way to use your model. You do not know if there will be only one view, and what kind of user interaction will need to be supported.
- There may come a time in the semester when you will be asked to share your code with other students, or see others' code. So your design, code and documentation is not "for your eyes only".

6 High-level Design Principles

This assignment's open-endedness is strategic, to simulate a real-world situation where you may have a high-level idea of what is to be accomplished, but details are known progressively. This is what makes design challenging. As you make your design, keep the following in mind:

- Try to extrapolate from this assignment. What kinds of things may be coming? How can you make your design flexible to withstand changes?
- Your design may need to be enhanced as you add features. Adding a new feature without changing any of your design is ideal: prepare to achieve this goal as much as possible. But in cases where you

cannot, strive towards isolating and minimizing changes. Editing an existing design is the worst option, because it can break other things.

- This series of assignments expects you to *combine* the design techniques you have learned in this and previous semesters. It would help to review what you have learned, and identify when they may be useful.

7 What to submit

A complete submission must contain:

- All your code in the src/ folder
- All your tests in the test/ folder
- At least two example PPM images, their blurred and its sharpened versions, and greyscale and sepia versions in a res/ folder
- An image of a class diagram showing your design (excluding tests). The class diagram should show names of classes and interfaces, signature of methods and inheritance relationships. Do not show field names or "has-a" relationships because they may clutter the diagram.
- A text README file explaining your design. Your README file should give the graders an overview of what the purposes are for every class, interface, etc. that you include in your model, so that they can quickly get a high-level overview of your code. It does *not* replace the need for proper Javadoc!
- A citation for the source of each image (in the README file). It is your responsibility to ensure that you are legally allowed to use any images, and your citation should be detailed enough for us to access the image and read its terms of usage. If an image is your own photograph or other original work, specify that you own it and are authorizing its use in the project. **Please do not use images provided or shown in this assignment as your examples.**

8 Grading standards

For this assignment, you will be graded on

- the design of your model interface(s), in terms of clarity, flexibility, and how plausibly it will support needed functionality;
- the appropriateness of your representation choices for the data, and the adequacy of any documented class invariants (please comment on why you chose the representation you did in the code);
- the forward thinking in your design, in terms of its flexibility, use of abstraction, etc.
- the correctness and style of your implementation, and

- the comprehensiveness and correctness of your test coverage.