Lab 8
Implementation Plan

1. Download the sample code and test it. Create a new instance of ImageViewer and use it to load an image. Slide the “Brightness Levels” control all the way to the left. Then, move it slightly to the right by pressing the right arrow key several times. You should see clear changes in the image.

2. Create a new class named Expander that extends ImageFilter. The easiest part of this class is the constructor. It should take no parameters and do nothing. Like our Quantizer class, this class will inherit the filter and layerFilter methods from the ImageFilter class. Therefore, the main method you will have to define in this class is the pixelFilter method. The first thing to do in pixelFilter is determine the largest and smallest values in the pixel array. We suggest defining two private methods that will return the minimum and maximum values in a pixel array and invoking these methods from filter. Once it has the minimum and maximum values, your pixelFilter should process every value in the pixel array. If a particular element in the pixel array has value, b, you should replace it with

\[
\frac{b - \text{minimum}}{\text{maximum} - \text{minimum} + 1} \times 256
\]

3. Add an “Expand Range” button to your interface. Create a new JPanel to hold the button and add this JPanel to the controlPane right after the load button. This JPanel will eventually also hold the button that displays histograms.

4. Add code to the buttonClicked method to invoke the filter method of your Expander class when the “Expand Range” button is clicked. This code should apply the filter method to the image currently associated with the variable displayed. Then, both display the result filter returns and associates it with the variable displayed. Test this new functionality on the images from the ExpandExamples folder under the AllImages folder (except for the RISD.png image which is damaged).

5. Create a DualImageViewer class that extends GUIManager. It should use a BorderLayout with two Imageviewers in the CENTER and a JPanel holding the three control buttons in the SOUTH. The Imageviewers should be placed in a JPanel that uses GridLayout. Add the three buttons now, even though they won’t do anything for a while. Remove the createWindow call from ImageViewer. Test to ensure that each of your two image viewers appears correctly and can load images individually.

6. Add a getPic method to ImageViewer that returns the value of displayed. Use this new method together with the setPic method to implement the “Move Left -> Right” button. This will require adding a buttonClicked method to the DualImageViewer class.

7. Use the Paster class we provided to implement the “Insert Left -> Right” button’s functionality.

8. Modify the buttonClicked method in the DualImageViewer class so that the “Show Difference” button creates a new ImageViewer, tells the new ImageViewer to create a window, and sets the picture in the viewer to be the left image from the DualImageViewer. Test that this works.

9. Create and define the Differencer class. This class will extend ImageViewer like your Expander class, but its implementation will resemble the Paster class included with the starter project more than the Expander or Quantizer class. Its constructor will take an SImage as a parameter and associate it with an instance variable. It will include the definition of a layerFilter method that will take another SImage as a parameter. It will also take an int identifying a color level and extract the pixel arrays for that color level from both the image passed to the constructor and to layerFilter. It should return a pixel array that holds the absolute values of the differences between the two pixel arrays.

10. Modify the buttonClicked method in DualImageViewer to use the filter method of the Differencer class so that it displays the difference between the left and right images. Test this code by computing the difference between the two images in the Difference folder of AllImages.
11. Define the DisplayHistogram class. This class should extend GUIManager. In the handout, we suggested you should write a private method that draws a single line to simplify the implementation of this class. A good way to start therefore is to write this method and a simplified version of the constructor that will test it. This initial constructor will expect no parameter, create a 256 by 200 pixel array, use the private method to draw a few vertical lines of different sizes within the pixel array, create an SImage using the pixel array, and finally display this SImage in a JLabel in its window. If you include an invocation of createWindow in the constructor, you will be able to test this class and your line drawing method independently of the rest of your program.

12. Since you will need a working version of the DisplayHistogram class to tell if your Histogram class is really working, next define a trivial Histogram class that should cause your final DisplayHistogram class to display a window like this (actually, you are not required to draw the gray scale lines):

To do this, define the Histogram method that is supposed to return the number of pixels of a given brightness so to instead return the brightness value itself. Define the method that is supposed to return the most frequently occurring brightness value to returns 255. Make the constructor should expect no parameters and do nothing.

13. Next, create one of your trivial Histograms in the DisplayHistogram constructor and add a loop to the constructor to draw 256 lines based on the values returned by this Histogram. Test this until the result looks like the image above.

14. Now you are ready to work on the real Histogram class. First, change the constructor in the current version of the class to expect an SImage as a parameter. Next, change the DisplayHistogram constructor to expect a Histogram as a parameter. Finally, add a “Show Histogram” button to your ImageViewer class and modify its buttonClicked method so that it constructs and new Histogram using the image displayed and a new DisplayHistogram using this Histogram. Test this code. Since you did not modify the methods in the Histogram class, it should still display a triangle like the one shown above.

15. Write the real Histogram class. The constructor should create an array of 256 ints declared as an instance variable. It should then extract a pixel array of the gray levels of the SImage that was passed as a parameter. It should examine each value in the array, adding one to the corresponding position in the 256 element array as it does so. With this array, it should then be easy to define methods to return the most frequently occurring brightness value and the number of pixels of any brightness. Test the result on some of the images provided. You may find it interesting to look at the histograms of image differences.

16. Finally, you should complete the Blocker class. This class will extend ImageFilter and resemble Quantizer and Expander in its style of implementation. The key is to write two private methods to fill a block and to find the average brightness of a block. Since it is simpler, you might want to start by defining the method to fill a block with a particular value. You can test this method by defining a version of the pixelFilter method that fills each block with the brightness of the pixel in its upper left corner. Write this code and modify ImageViewer so that it displays and responds to the “Block size” slider. Test this code.

17. Once this is done, write the method to compute the average of a block and change pixelFilter to fill each block with its average rather than with the brightness of its upper left corner.

18. Check over all of your work before handing it in: formatting, comments, good variable names, vertical whitespace, and design. Consider what would happen to each of your methods if the input image data was not what you expected. Would your expandRange method work on an all-white image?