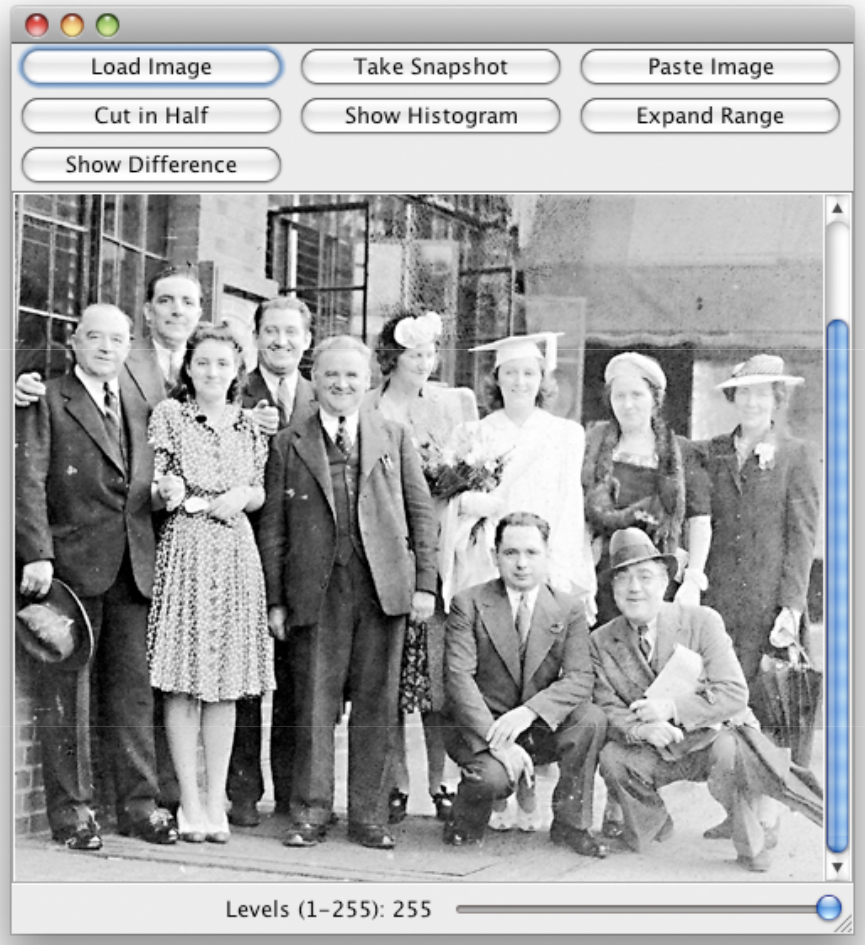
You Can Make a Difference!

Due November 11/12 (Implementation plans due in class on 11/9)

In last week's lab, we introduced some of the basic mechanisms used to manipulate images in Java programs. In this week's lab, we ask you to use your knowledge of arrays to:

- explore additional image processing operations including cropping and pasting, and
- implement several common algorithms for processing arrays.

The screen shot below shows the main window that the program you write this week should display.



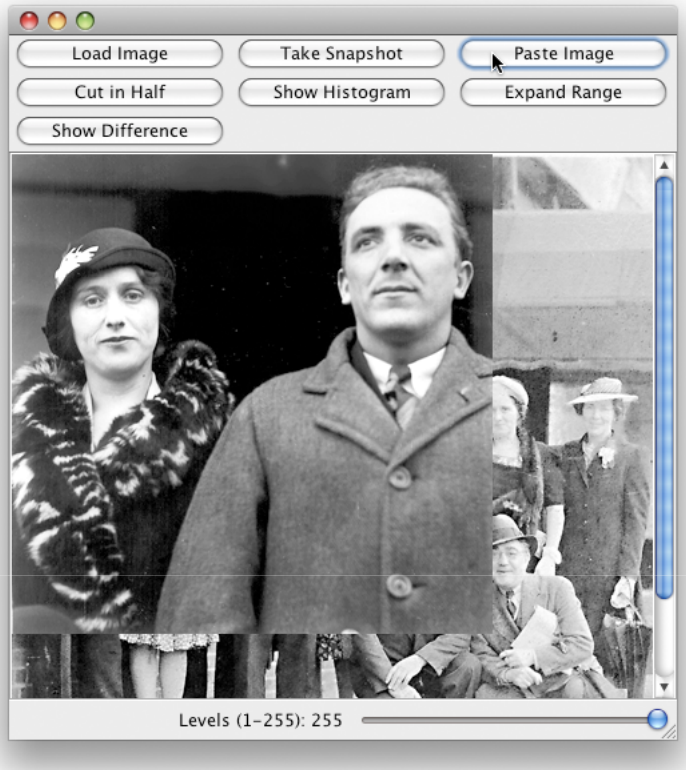
You should recognize the basic layout of the program's window from last week's lab. In fact, we will use the `ImageViewer` and `Quantizer` classes we asked you to write last week as a starting point for this program. The "Load Image" and "Take Snapshot" buttons will function exactly as they did last week. We will discuss the functions performed by the other buttons shortly.

The slider at the bottom of the window control the color resolution with which an image is displayed. That is, it functions just like the "Levels" slider you implemented in the last lab. In the sample window shown above, the image is displayed using 256 brightness values, the standard resolution for grayscale images. The illustration at the top of the next page shows the same image displayed using only 3 distinct brightness values.

The "Paste Image" and "Cut in Half" buttons perform very limited versions of the common paste and crop operations.

The "Paste Image" button tells the program to paste a copy of a second image into the image currently displayed on the screen. When this button is pressed, the program displays the same file selection dialog box used when the "Load Image" button is pressed. Rather than simply loading the image selected, however, the program pastes a copy of this image into the upper left corner of the image currently displayed in the window. A somewhat silly example is shown below on the right in which we have pasted a copy of the picture used in last week's lab handout over the picture shown on the front of this handout.

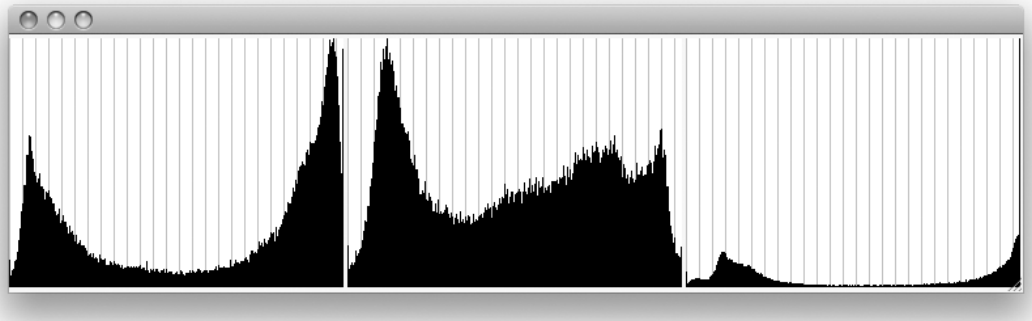
The paste operation should not increase the size of the image displayed in the window. Therefore, if the width and/or height of the image selected after "Paste Image" is pressed exceed those of the image currently displayed in the program's window, the image being pasted will effectively be cropped and only as much of the image as will fit will be pasted.



The "Cut in Half" button causes the program to divide the currently displayed image in half and display the left half. The window below shows the result of applying this transformation to our example image.

The code you write to implement "Paste Image" and "Cut in Half" will play an important role in next week's lab. For the purpose of that lab, the definition of "half" you use is critical. If an image's width is 2N, then the width of a half is obviously N. If the width is 2N+1, then your program should produce a "half" whose width is N+1.

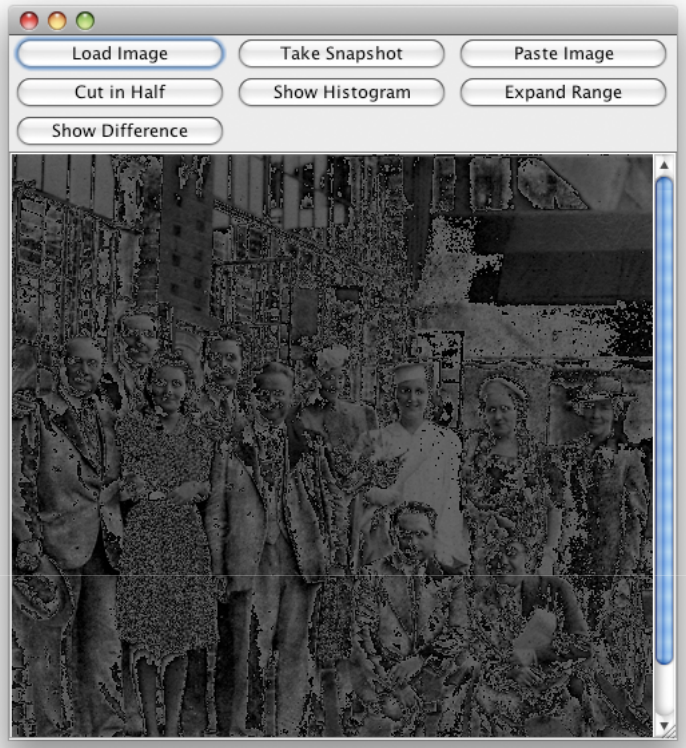
Three additional buttons are found after "Cut in Half". The "Show Histogram" button creates a new window containing histograms of the brightness values associated with red, green, and blue components the displayed image's pixels. An example of such a histogram window is shown below. The graph on the left shows the distribution of redness values in the image, the graph in the middle shows the distribution of greenness values, and the rightmost graph shows the distribution of blueness values. This particular set of histograms was created using an image showing pinkish flowers with quite a bit of green foliage.



The "Expand Range" button adjusts the brightness values of an image's color components linearly so that they range all the way from 0 to 255. This operation has no effect on images with a histogram like the ones shown above since the pixel values already stretch from 0 to 255. This operation is valuable, however, on images that are either too dark (the largest brightness value is significantly less that 255) or too bright.

The only remaining button is the "Show Difference" button. This week's `ImageViewer` class will save a copy of the last image placed in the display using either the "Load Image", "Take Snapshot", or "Cut in Half" buttons. This saved image may be different from the image actually displayed in the window if the user adjusts the image resolution using the slider or presses the "Paste Image" or "Expand Range" buttons. When the user presses the "Show Difference" button, the program creates an image with brightness values determined by the differences between the image currently displayed in the window and the image

that was originally loaded. This image is then displayed in a new, independent image viewer window like the one shown on the right. This window shows the differences between the original image shown on the first page of this handout and the version of the image produced by setting the Levels slider to 3 (as shown on the second page of the handout).



Class Structure

The program you will write this week will be composed of 5 distinct classes. The good news is that we will give you working versions of three of them (`ImageViewer`, `Quantizer`, and `DisplayHistograms`). The versions of `ImageViewer` and `Quantizer` we provide are very similar to what you completed last week. They differ from what we had you construct last week only in small ways that make it easier for them to support the larger collection of image operations you will implement this week. To complete this program, you will:

- revise the `ImageViewer` class we provide in the starter folder,
- define a new class named `DisplayDifference` that extends `ImageViewer`, and
- define a class named `Histogram` that will provide the information about an image our `DisplayHistograms` class needs to draw pictures of an image's color histograms.

Extending `ImageViewer`

The most obvious change you will make to the `ImageViewer` class is to add a lot of new buttons. In last week's lab, the class displayed just two buttons: "Load Image" and "Take Snapshot". This week you will add 5 additional buttons. In addition, you will add private methods to the ImageViewer class to implement the "Paste Image, "Cut in Half", and "Expand Range" buttons. The code for "Show Difference" and "Show Historgram" will be implemented primarily in two of the other classes that will be included in your program.

Using a GridLayout

To avoid having the new buttons stretch all the way across your computer's screen, our `ImageViewer` class uses a new layout manager to control the placement of components within the `JPanel` that will eventually hold the seven buttons. The layout manager we use is called a `GridLayout`. As its name suggests, it divides the space within a panel into a two-dimensional grid of cells and places one component in each cell. Components are placed in cells from left to right and from top to bottom as they are added to the panel. If you look back at the screenshots we included above, you can see that we used a 3 by 3 grid to hold the buttons in our `ImageViewer`. The "Load Image" button was added first while the "Show Difference" button was added last.

The GridLayout constructor expects two `int` parameters specifying the width and height of the desired grid. Therefore, we tell the computer to use a 3 by 3 grid for your buttons by executing an instruction that looks like

```
controlPane.setLayout( new GridLayout( 3, 3 ) );
```

Expanding an Image's Brightness Range

When the "Expand Range" button is pressed, the `ImageViewer` will adjust the brightness of the pixels of the displayed image so that for each of the primary colors, the smallest brightness value is 0 and the largest is 255. The code that handles this button will have to determine the smallest and largest brightness values used for each of the color components of the image and then linearly scale the brightness values so that they extend from 0 to 255. To do this you will probably want to include at least two private methods in the class. One of these methods will determine the largest value in a two dimensional array of values. The other will determine the smallest value in an array.

In addition, we suggest that you divide the code to expand the range of an image into two methods. The first method should take an `SImage` as a parameter and return an `SImage` with all of its brightness values appropriately expanded. This method will depend on a second method that takes an `SImage` and an `int` specifying which color component to process (`SImage.RED`, `SImage,GREEN`, or `SImage.BLUE`). This second method will return an `int` array containing the expanded brightness values for just the specified color component of the image. We suggest using similar pairs of methods to implement all of the image transformations in this lab.

To help you understand the approach we have in mind, we have used it in the version of the Quantizer class provided in this lab's starter folder. In Lab 7, we suggested that you define an `adjustPixels` method that takes three parameters: the number of brightness levels to be used, an `SImage`, and an `int` specifying a color layer. This method returned an array of `int`s describing the adjusted brightness values for one layer of the image. We then told you to invoke this method three times within a single construction of the form:

```
new SImage( adjustPixels( sliderValue, getImage(), SImage.RED ),
            adjustPixels( sliderValue, getImage(), SImage.GREEN ),
            adjustPixels( sliderValue, getImage(), SImage.BLUE ),
)
```

used within your `sliderChanged` method to create a new SImage with all of its layers appropriately modified.

In our version of the Quantizer class, we have included an additional method named `requantize` that produces a `new` version of an image given just two parameters: the number of brightness levels to be used and the `SImage` to be processed. The requantized image is produced using a construction like the one shown above that includes three invocations of `adjustPixels`. This new method makes our `sliderChanged` method a bit shorter since a single, concise invocation of `requantize` replaces the complex construction shown above.

Honestly, in the `Quantizer` class, the use of the `requantize` method is superfluous. The `sliderChanged` methods you wrote last week were already concise. By the end of this lab, however, the buttonClicked method in your `ImageViewer` class will contain an `if` statement designed to handle seven separate buttons. If more than one or two lines of code are required to handle each button, `buttonClick` will get large and difficult to read. It is in this method, therefore, that the technique we have illustrated in our `Quantizer` will really pay off.

The Paste Operation
When the "Paste Image" button is pressed, the `ImageViewer` will first pop up a file selection dialog box as it does when "Load Image" is pressed. It will then create a new `SImage` formed by replacing the brightness values describing the upper left corner of the displayed image with those of the image selected using the dialog box, and finally display the result. Again, you will probably want to write two private method to implement this transformation. One of these methods will take the two `SImage`s that should be combined as parameters and return the resulting `SImage`. The other will take the two images and an int specifying one of the three color layers in an image. The second method will return an "`int [][]`" describing just the specified layer of the combined image.

As mentioned above, the code for the paste operation must work correctly even if the images being pasted is wider or taller than what will become the background image. While writing this code, it may be helpful to recall that the boolean expressions used as conditions in while and for loops can include boolean operators. In particular, it is possible to write a loop header like:

```
for (int x = 0; x < image1.getWidth() && x < image2.getWidth(); x++) {
```

Maintaining the Original Image
Our `ImageViewer` class has two `SImage` instance variables named `original` and `updated`. The variable `original` should always be associated with the `SImage` most recently placed in the `ImageViewer` by pressing the "Load Image", "Take Snapshot" or "Cut in Half" button. The variable `updated` should refer to the `SImage` that is currently being displayed. These two images will frequently be slightly different since the displayed image can be modified by clicking the "Expand Range" button or by adjusting the slider displayed at the bottom of a `Quantizer` window.

When asked to display a histogram, expand the range of an image, cut an image in half, or paste one image over another, your code should use the displayed image rather than the original. That is, the code for these operations should start with the value associated with the variable `updated`. When the slider at the bottom of the window are adjusted, a new image to be displayed should be computed starting with the original image. Therefore, the `getImage` method provided by our `ImageViewer` returns the image associated with the variable `original`.

The `Histogram` Class

You should define a class named `Histogram` to hold the array of pixel counts that provide the data used to draw a histogram. The constructor for the `Histogram` class should take two parameters. The first parameter should be an `SImage`. The second parameter should be an `int` specifying the layer of the `SImage` from which the histogram should be constructed. That is, the second parameter's value will either be `SImage.RED`, `SImage.BLUE`, or `SImage.GREEN`. The constructor should create an array of 256 integer values. It should include a loop that will examine all the elements in the pixel array for the specified layer, setting each of the 256 elements of the histogram array so that the element at index b is equal to the number of entries in the pixel array that are equal to *b*.

The histogram class should provide one public methods named `frequency`. This methods should take a brightness value, b, and return the count stored at position b in the histogram array. If the parameter value is out of range (i.e., less than 0 or greater than 255), your `frequency` method should return -1.

Our `DisplayHistogram` class will use your `Histogram` class to display histograms for the three color layers of an `SImage`. The constructor for the `DisplayHistogram` class takes an `SImage` as a parameter. It creates three `Histogram`s, one for each color layer, and then creates a new window displaying graphs of the brightness distributions for these three color layers. Your `ImageViewer` class should create a new `DisplayHistogram` window when the "Show Histogram" button is pressed. The histograms displayed should be based on the image currently displayed in the `ImageViewer`.

The `DisplayDifference` Class

When the "Show Difference" button is clicked, your `ImageViewer` should create a new `DisplayDifference` object. The constructor for the `DisplayDifference` class should take two `SImages` as parameters. In the constructor, you should include code to compute the numerical difference between the brightness values of corresponding pixels for each of the three color layers and create a new SImage whose brightness values are based on these differences. The difference between corresponding pixels of two images can range anywhere from -255 to 255. The brightness values of an image are supposed to fall between 0 and 255. Since we will want to display the differences you compute as images, you should actually compute the absolute values of the differences between pixels. In Java, the absolute value of x can be produced by evaluating the expression `Math.abs( x )`.

You should define two private methods to assist in this computation. One will take two SImages and return an SImage based on their differences. The other method will take two SImages and an int specifying a color layer and return a table of brightness values for one layer of the difference image.

The `DisplayDifference` class should be defined to extend `ImageViewer`. This provides a convenient way to display the difference image and to examine its histogram and/or expand its brightness range. Our `ImagerViewer` class provides a public method named `setImage` that can be used to set the "original" image displayed within an `ImageViewer`. The `DisplayDifference` constructor should invoke `setImage` to display the image it has computed. Of course, several of the buttons included in ImageViewer (like "Load Image") will be pointless in a window created to display a difference image.

Implementation Plan?

This lab handout is missing one very familiar feature. There is no "Implementation Plan."

This week, we want you to take a stab at making your own step-by-step plan for completing this program similar to the implementation plans we have presented in previous weeks. We will collect these plans in class on Monday. This is a dry run! We will provide you with copies of our implementation plan in class once you have turned in your own. The implementation plan that you turn in, however, will count as part of your grade for this lab.

Your implementation plan should be about 1 or 2 TYPED pages. In preparing your implementation consider how you will test the correctness of the code you write in each step of the plan. That is, remember that one of your main goals in developing your plan is to ensure that as you develop the program there is a way to test the correctness of each addition you make before moving on to the next step.

Getting Started

To start this lab, you should download a copy of the starter project described above.

- Launch Safari (you can use another browser, but these instructions are specific to Safari) and go to the "Labs" section of the CS 134 web site (http://www.cs.williams.edu/~cs134).
- Find the link that indicates it can be used to download the `Lab7Starter` program.
- Point at the link. Hold down the control key and depress the mouse button to make a menu appear.
- Find and select the "Download Linked File As ..." item in the menu.
- Using the dialog box that appears, navigate to your "`Documents`" folder and save the `Lab7Starter.zip` file in that folder.
- Return to the Finder, locate the `Lab7Starter.zip` file in your Documents folder, and double-click on it to create a `Lab7Starter` folder.
- Rename the folder using a name including "Lab7" and your name (e.g., FloydLab7). Remember not to include any blanks in the new folder's name.
- Launch BlueJ and use the "Open Project" item in the "File" menu to open your Lab 7 project.

Submission Instructions

As usual, make sure you include your name and lab section in a comment in each class definition. Find the folder for your project. Its names should be something like FloydLab7.

- Click on the Desktop, then go to the "Go" menu and "Connect to Server."
- Type "cortland" in for the Server Address and click "Connect."
- Select Guest, then click "Connect."
- Select the volume "Courses" to mount and then click "OK." (and then click "OK" again)
- A Finder window will appear where you should double-click on "cs134",
- Drag your project's folder into the appropriate dropoff folder.

You can submit your work up to 11 p.m. two days after your lab (11 p.m. Wednesday for those in the Monday Lab, and 11 p.m. Thursday for those in the Tuesday Lab). If you submit and later discover that your submission was flawed, you can submit again. The Mac will not let you submit again unless you change the name of your folder slightly. Just add something to the folder name (like the word "revised") and the re-submission will work fine.

Grading

This labs will be graded on the following scale:

++	An absolutely fantastic submission of the sort that will only come along a few times during the semester.
+	A submission that exceeds our standard expectation for the assignment. The program must reflect additional work beyond the requirements or get the job done in a particularly elegant way.
✓+	A submission that satisfies all the requirements for the assignment --- a job well done.
✓	A submission that meets the requirements for the assignment, possibly with a few small problems.
✓-	A submission that has problems serious enough to fall short of the requirements for the assignment.
-	A submission that is significantly incomplete, but nonetheless shows some effort and understanding.
--	A submission that shows little effort and does not represent passing work.

Completeness / Correctness

- `ImageViewer` determines minimum and maximum brightness values before expanding range
- `ImageViewer` adjusts brightness values proportionately to stretch from 0 to 255 when expanding range
- `ImageViewer` implements "Cut in Half" correctly
- ImageViewer performs paste correctly
- DisplayDifferences computes absolute values of pixel color differences
- ImageViewer GUI interface displays correctly
- Histogram class computes pixel counts correctly

Style

- Commenting
- Good variable names
- Good, consistent formatting
- Correct use of instance variables and local variables
- Good use of blank lines
- Uses names for constants