CS 117: Introduction to Computer Science	**Winter 2006**
Handout 6: Problem Set #3 (Due: Wednesday, 8 February 2006, 11:59p)	*27 January 2006*

When I'm feeling blue, all I have to do
Is take a look at you; then I'm not so blue.
— Phil Collins (b. 1951), "A Groovy Kind Of Love."

-1. **[Required!]** Go to the course website, and use the anonymous feedback form to give me some comments about the class so far. Is the pace too fast? Too slow? Are lectures clear? Are they interesting? What would you change if you could?

Movies—particularly action movies that use a lot of special effects—often use a technique called "blue screening" to generate some scenes. In blue screening, the actors in a scene are filmed as they perform in front of a blue screen. Later, the special-effects crew removes the blue from the scene and replaces it with another background (an ocean, a skyline, the Carleton campus). The same technique is used for television weather reports; when Al Roker gestures at cold fronts and snow storms, he's really gesturing at another part of the blank screen behind him; later, the computer-generated weather map replaces the blue background. (Sometimes "green screening" is used instead.)[1]

There are three goals for this assignment: (1) to give you a chance to use Java, especially loops in Java, to do something interesting; (2) to let you create interesting images; and (3) to introduce you to Dave Musicant and Amy Csizmar Dalal, two other Carleton CS professors. (You'll see how soon.)

Before you get started, there are some things that you should know about images and how they are typically stored. You can think of a digital image as a long array of pixels, each of which has a red (R) value, a green (G) value, and a blue (B) value. An R, G, or B value is just an integer between 0 and 255. Suppose, for example, that you have an image that is 200 pixels wide and 100 pixels high. Pixel numbers 0 through 199 in our imaginary array form the top row of the image, pixels 200–399 form the second row, etc. In order to modify the image, you need to "loop through" all of these pixels, one at a time, and do something to some or all of the pixels. For example, to remove all of the green color value from each pixel, you could use the following loop:

```
for each pixel in the image
    set the green value of the current pixel's color to 0
done
```

To facilitate reading in, storing, and manipulating image files using "standard" formats (like `.jpg`, `.gif`, and `.png`), we will use a class called `EzImage`. Copy `EzImage.java` from the course webpage into a new directory called `bluescreen`. You can also find documentation on EzImage on the course webpage. Also on the course webpage are a handful of sample images, including images of Dave Musicant and Amy Csizmar Dalal standing in front of a cinder-block wall. (Plus, now you know over half of the CS faculty.)

0. Estimate the amount of time you spent on this problem set, and write it at the top of your `ps3.txt`.

1. Your first task is to write some basic image manipulations. Write a class `PhotoLab` that contains the following following methods:

   - `public PhotoLab(EzImage image)`
     The constructor; the input parameter is the source image.

[1] What we're going to be doing in this assignment is closer to the technology that's used to put the "yellow line" (first-down marker) on football broadcasts. From `http://ask.yahoo.com/20001012.html`: *The virtual line is drawn on video based on the position [of the] first down marker, ridiculously exact details [of] the live camera's position (including altitude and lens angle), a digital 3D model of the field, and two palettes of colors for the field and the players. The player's colors automatically override the virtual line's colors, so it appears as though they're stepping over it.*

- `public EzImage onlyRed()`
  Creates and returns a new image containing only the red aspects of the original image. You can do this by setting all the values in the green and blue pixel arrays to 0. Note that you shouldn't change the original image! You may find the method `copy()` in `EzImage` helpful.
- `public EzImage negate()`
  Creates and returns a new image that is a conversion of the original image to "negative" form (like a photographic negative). You should modify all colors in a way that turns black into white, white into black, dark colors into light, light into dark, etc. (Don't worry if you don't exactly match the color correspondence between a photo negative and its print.) Again, don't change the original image!

Create a class `PhotoLabTester` to test your program. (You should add to the same file additional tests for the functionality you develop in the next question.)

2. Now that you've got the hang of manipulating images, let's go back to the idea of blue screening. The pictures named `background?.jpg` on the course web page depict some sort of scenery. The others (`amy.jpg` and `dave.jpg`) are pictures of CS faculty standing around in front of a plain background (in this case, a wall instead of a blue screen). We can "combine" these images into a single image by replacing the "wall" pixels in one of the people pictures with pixels from one of the scenery pictures. To do this, we have to figure out which pixels correspond to the wall (and can be changed) and which ones correspond to the person (and should be left alone).

   Identifying which pixels are in the wall is tricky. Here's a formula that works pretty well for these pictures. Any pixel that satisfied the following counted as part of the wall:

   ```
   red value >= 100
   and green value >= 100
   and absolute value of difference between red value and green value < 30
   and absolute value of difference between red value and blue value < 30
   and absolute value of difference between green value and blue value < 30
   ```

   (The last three conditions say that the values of each of the colors must be similar—i.e., gray.)

   Add the following methods to your PhotoLab class:

   - `public EzImage replaceWall(Color replacementColor)`
     Creates and returns a new image that replaces all "wall" pixels in the original image by the specified color. The trick here is to decide what it means for a pixel to be part of the "wall" (in terms of its RGB values). Hint: the Color class contains the methods `getRed()`, `getGreen()`, and `getBlue()`, which you may find useful here.
   - `public EzImage replaceWall(EzImage replacementImage)`
     Creates a new image that replaces all "wall" pixels in the original image with pixels taken from the replacement image. Note that the main image and the replacement image need to have the same dimensions for this technique to produce the desired effect.

   Test your program by putting Amy and Dave in front of the provided backgrounds. Feel free to use your own pictures, too. If you have extra time, try to figure out how to remove the background of your own pictures, too. (Assuming that you aren't standing in front of a gray cinder-block wall in yours ...)

   Also create a `ps3.txt` file, and include any discussion of anything interesting that happened while you were working on this assignment. Specifically, discuss some of the problems that you encountered while trying to remove the wall pixels and retain the "person" pixels? Can you have done better than using the suggested formula?