Photoslop^{JK}

1 Introduction

Hello everyone! This week, we will be working with classes we provide to construct your very own classes, as well as getting some nice 2D array practice. Behold your next assignment: Photoslop^{JK}!

2 Problem Description

You're quite fed up with paying big corporations hundreds for sub-par products. Okay actually, you're a broke college student and too poor to afford anything anyway, but it sounds cooler to say you're boycotting them.

Anyway, you've decided to develop a revolutionary photo editing tool: Photoslop^{JK}

3 Solution Description

Before you get started, prepare yourself. This is a significant step up from previous assignments, and involves several classes and their interactions, so before you start coding, be sure you understand how everything works in theory. Draw out the connections on paper. Your overall application will have three components, plus one additional one we provide for you.

3.1 `Pixel`

You'll need to write the mutable `Pixel` class representing a single pixel of a particular image. `Pixel`s are composed of four values: red, green, blue, and alpha (transparency), oft abbreviated RGBA. These should be instance data of type `int` in `Pixel`.

1. Include a constructor that will allow the developer to initialize the pixel to whatever color he/she desires. (Hint: would this be a no-arg constructor?)
2. Include mutator and accessor methods for *every* instance variable of `Pixel`. Remember to provide proper encapsalation for the instance data.

3. `Pixels` have the class invariant that their component values (RGBA) are between 0 and 255, inclusive.
4. **Hint:** Make sure your constructor accepts RGBA in that order.
5. **Method Naming:** if you adopt Java standard, camelCase, method naming conventions for your mutators/accessors, your code will work with the provided `Pic` (see below).

3.2 `Pic`

We've provided `Pic.class` for you. `Pic`s are our images - they represent a collection of pixels.

`Pic`s contain an internal 2D array of pixels. Review the lecture slides if you need to: 2D arrays are just arrays *of* arrays.

You will use the `Pic` class to represent the images you manipulate.

Hints:

- if `Pic`s are composed of mutable `Pixel`s, are `Pic`s mutable or immutable?
- See the included documentation (`Pic.html` in the `javadoc/` folder) to see the methods `Pic` has.
- `Pic` contains both `Pixel[][] getPixels()` and `getPixel(int x, int y)`. They're two different ways of accessing pixels: use whatever works for you.
- pssst. You may need `Pic`'s `Pic deepCopy()` method...

3.3 `ImageProcessor`

This is where most of your time will be spent. An `ImageProcessor` is a processor for a specific `Pic`, i.e, you may have several different instances of `ImageProcessor` for different `Pic`s. Accordingly, it will need a constructor to specify which `Pic` to use.

`ImageProcessor` has several ways of processing the image (enumerated below), but these shouldn't modify the original image. That is, `ImageProcessor` should be immutable.

`ImageProcessor` should have the following methods, which are transformations upon its underlying `Pic`. Instead of modifying the underlying image, we'll return our transformation. Note that the transformations listed below are listed loosely, without return type or visibility modifier. Consider the requirements of `ImageProcessor` and determine for yourself what the return type of each method should be. (Hint: it should be the same for all of them)

1. `add(int increment)`: Add the specified value to the red, green, and blue values for each pixel. Remember `Pixel`'s class invariant. Make sure to respect that here - max out the values at 255 and don't let them drop below 0.

2. `multiply(double scale)`: Multiply the red, green, and blue values of each pixel by the scaling factor. Note that these values are doubles. It is okay to round to an integer, but only as the final step. Same caution as above applies.

3. `chroma(Pixel key, int dr, int dg, int db)`: For every pixel, if the red, green, and blue values are within the delta (dr, dg, or db) of the r, g, b of the key, then set the r, g, b, and alpha to 0. In other words, if, if `red - key.getRed() < dr`, then the red difference is within the red delta. If the blue and green differences are within their respective deltas, set the pixel's alpha to zero.

4. `background(Pic bg)`: For every pixel, if its alpha value is zero, then set its RGBA values equivalent to the RGBA values of the corresponding pixel in bg.

3.4 `Photoslop`

Lastly, we've provided some skeleton code for running/testing your code. You will need to fill in the indicated spots with code to create and use the `ImageProcessor`

You will use add/multiply individually, and then combine chroma and background to create a greenscreen effect. See the provided `Photoslop.java` file for further details.

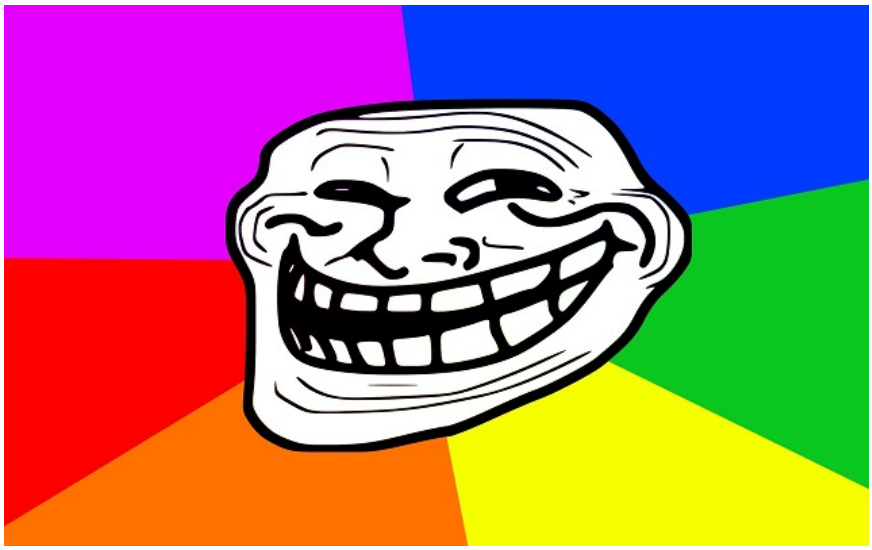
Use `Pic`'s `save(String filename)` method to save the image to the location requested via the command line args.

We've provided some sample images for you to work with, but you're more than welcome to use your own.

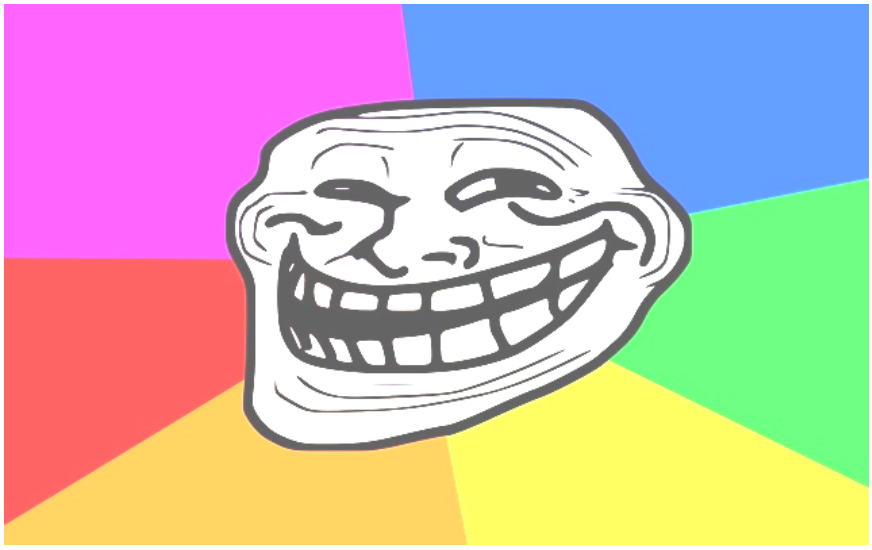
4 Example Output

4.1 Add

Default:



Adding 100 to it with `java Photoslop -a troll.png 100 trollplus`



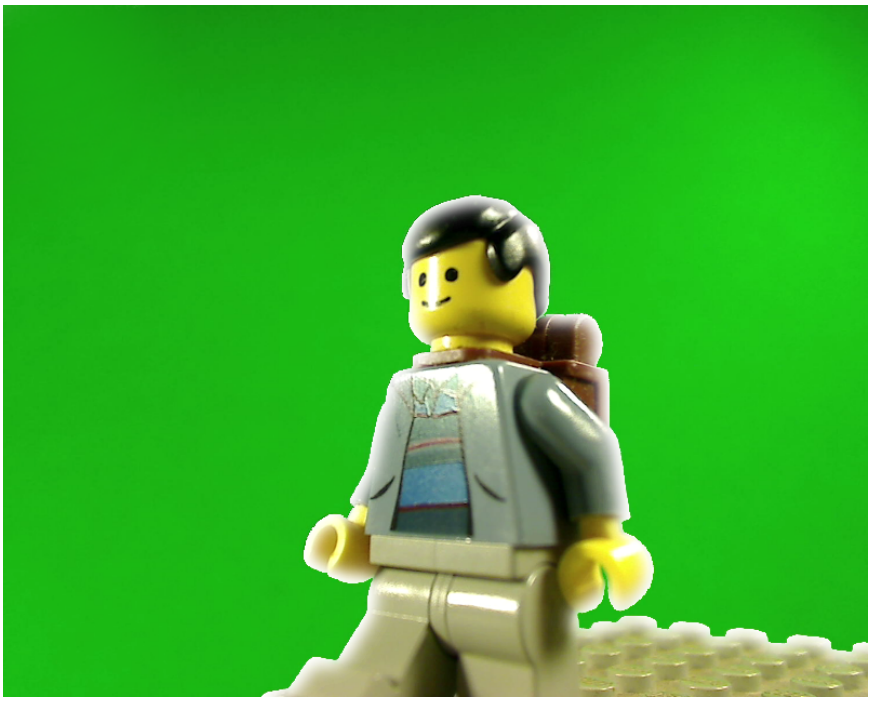
4.2 Multiply

Default:

Multiplying it by 15 with `java Photoslop -m darklion.png 15 brightlion`

4.3 Chroma + Background

Foreground:



Background:

Chroma-keyed out green, and added background with `java Photoslop -c legoguy.png explosions.png myawesomegreenscreen`

5 Recap

That was a lot. Let's recap:

1. You'll be writing the `ImageProcessor` class, which performs transformations on the underlying `Pic` object. **Hint:** you should need a mininum of five methods.
2. You are provided a `Pic` class, but it's compiled, so in order to see its methods, you'll need to view the documentation provided in the `javadoc/` folder.
3. `Pics` are a collection of `Pixels`, which you have to write.
4. `Pixels` have red, green, blue, and alpha `int` values. Write accessor/mutator methods for all of these, but be sure to name them properly in order for `Pic` to see them.
5. Finally, you'll write a "driver" - a class that exists just to run or test the code. `Photoslop`'s main method must perform the add, multiply, or chroma/background combination on the image specified by the command line arguments, and then save it. We've provided some of this functionality for you.

6 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and

why they are awesome, the online documentation for them is very detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

   /**
    * Creates an awesome dog (NOT a dawg!)
    */
   public Dog(){
      ...
   }

   /**
    * This method takes in two ints and returns their sum
    * @param a first number
    * @param b second number
    * @return sum of a and b
    */
   public int add(int a, int b){
      ...
   }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

6.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

7 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **25** points. Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.1.jar *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **25** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

8 Turn-in Procedure

Submit all of the Java source files you modified and resources your program requires to run to T-Square. Do not submit any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

Verify the Success of Your Submission to T-Square
Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!