Assignment 5

Learning Goals: Every student should understand the concepts of:

- input from and output to an external file
- loops
- inputting and outputting multi-dimensional lists
- passing multidimensional arrays/arraylists as parameters/arguments
- a little about image formats

PPM Image Format

The PPM (Portable Pix Map) image format is encoded in human-readable ASCII text. If you'd like to try out reading real documentation,, the formal image specification can be found here.

This is an example of a sample PPM file:

```
P3
4 4
255
0   0   0    100 0   0        0   0   0     255    0 255
0   0   0      0 255 175      0   0   0       0    0   0
0   0   0      0   0   0      0 15  175     0    0   0
255 0 255    0   0   0        0   0   0     255  255 255
```

Image Header

You can think of the image as having two parts, a **header** and a **body**.
The **header** consists of four entries:

```
P3
4 4
255
```

P3 is a "magic number". It indicates what type of PPM (full color, ASCII encoding) image this is. For this assignment it will always be P3.
Next comes the number of columns and the number of rows in the image (**4** x **4**).
Finally, we have the maximum color value **255**. This can be any value, but a common value is 255.
The way you see the header presented is how it should be spaced out.

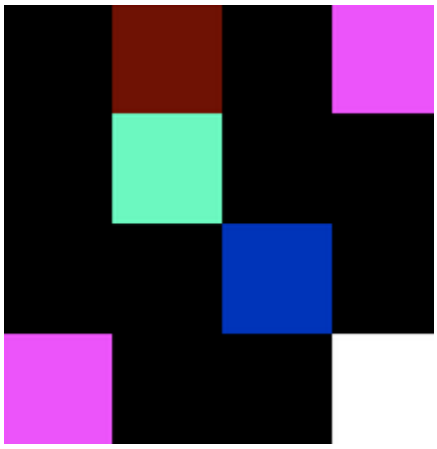
Image Body

The **image body** contains the actual picture information. Each pixel of the image is a tiny, colored square. The color is determined by how much red, green, and blue are present. So, **0 0 0** is the first color of the image, which is black, and the last pixel in the image is **255 255 255**, which is white. By varying the levels of the RGB values you can come up with any color in between.

Note that color values must be separated by a space, but after than additional whitespace is ignored by the image viewer. In the sample PPM file above we used additional whitespace to format the image so that it is easy for a human to understand, but the computer doesn't care if everything is on one line, if there is one line per line of the image, or some mix. For this assignment, we will keep 1 line of text = 1 line of the image for this assignment to make the images easier to read in a text editor, so your solution's outputs should meet this requirement.

Putting it all together

The example image above would look something like this:



Keep in mind, each square is one pixel, so the real thing is much smaller (the rendered image was blown up by 5000%).

How to view PPM files

While PPM files are easy to view as text (you can use Notepad, for instance), and easy to work with in code, they are highly inefficient. Most modern image formats use some kind of compression to make their size reasonable while preserving the image appearance. This is not to say that PPMs don't still have some life in them--one modern use for PPM is an intermediate format when converting

images from one type to another.

You may need to install a program to view these images visually depending on your operating system. For Windows, we find that Irfanview is a small download and works quite well. It will also allow you to convert your own images to PPM so you can practice with pictures you took in the past (keep in mind that you may need to make them very small or the resulting PPM will be **quite large in size**!).

The Assignment

Whew! That may sound like a lot of work, but it's really pretty simple. To make it easy to code up we've broken the program into three phases. You should move towards an eventual final version of the solution by completing each phase in order, then modifying what you did in the previous phase.

In the first you are reading the whole image and writing it to a file without making any changes to the image in the process. Believe it or not, at this point you've written most of the program! In phase II we add some effects, and in phase III we add a menu for the user.

Phase I:

The user will specify the name of the image file. The file will be a text file in PPM format as described in the discussion above. Remember an image file has three dimensions: (i) rows, (ii) columns, and (iii) channel, where channel is either the red, green, or blue channel. So to store this, you have a few choices:

- Use 3 2-dimensional Arrays
- Use a single 3-dimensional Array (recommended)
- Either of the above, but with ArrayLists (not recommended)

If the image file cannot be opened, the program should report that fact and abort the program.

The user will specify an output filename. The purpose of the program in Phase I is to make an exact copy of the input file as the output file.

Your output file should be formatted such that each new line of the file as text corresponds to each new line in the image. Beyond that, the formatting is up to

you, but you may want some space between the RGB channel trios. Remember the file format is unaffected by extraneous whitespace.

Example interaction with the user:

```
Portable Pixmap (PPM) Image Editor!

Enter name of image file:  feep.ppm
Enter name of output file: out.ppm
out.ppm created.
```

And the output file created would be in a file called out.ppm and would be identical to feep.ppm. Your program is NOT responsible for displaying the image in the file, just to manipulate the pixels and create an output file in the proper PPM format.

Test this with small files and large files. You can check to see if they are identical by loading them both into Notepad and comparing number by number.

Example ppm files can be downloaded here: ZIP Archive (1.2 MB) containing the following images:

- cake.ppm - A picture of a slice of cake on a plate
- squares.ppm - Some boxes
- blocks.ppm - Some solid blocks of color
- tinypix.ppm - The example image from above

Phase II:

Write a static method called `negateRed`. It will change just the RED color numbers into their "negative". That is, if the red number is low, it should become high and vice versa. The maximum color depth number is useful here. If the red were 0, it would become 255; if it were 255 it would become 0. If the red were 100, it would become 155. It should make changes to the list as described above.

When you have this method written, insert it into Phase I so that every pixel of the picture has had its red color negated in the output file every time.

View this picture - does it look as you expected?

Write another method called `flipHorizontal` which will flip the picture horizontally. That is, the pixel that is on the far right end of the row ends up on the far left of the row and vice versa (remember to preserve RGB order!).

Write a function called `grayScale` which will change the picture into a grayscale image. This is done by averaging the values of all three color numbers for a pixel, the red, green and blue, and then replacing them all by that average. So if the three colors were 25, 75 and 250, the average would be 116, and all three numbers would become 116.

Write a function called `flattenRed` which will set the red value to zero. Write similar methods for the other color numbers.

In summary, you must implement the following functionality:

- **negateRed** - Its job is to negate the red number of each pixel.
- **negateGreen**, as above but change the green
- **negateBlue**, as above but change the blue
- **flipHorizontal** that flips each row horizontally
- **grayScale** sets each pixel value to the average of the three
- **flattenRed** sets the red value to zero
- **flattenGreen** sets the green value to zero
- **flattenBlue** sets the blue value to zero

Phase III:

The last part of the problem is to add a menu in `main` so that the user can decide which of these effects will be applied to an image.

You should apply the desired effects to the image one at a time.

You should also give your solution "robustness" to the program against errors in this phase. See the Testing section below for details.

Testing:
Your main method should avoid reading from a failed input stream. What if the file is not as large as the row and column numbers indicate it will be? Does your code handle that gracefully (without falling into infinite loops or crashing, for example)?

All your manipulations should NOT cause a color number to be less than 0 nor larger than the maximum color depth specified in the file.

Adapted from
http://nifty.stanford.edu/2012/guerin-image-editor/image_editor.html

Sample run:

```
Portable Pixmap (PPM) Image Editor

Enter name of image file:  cake.ppm
Enter name of output file: outcake.ppm

Here are your choices:
[0] exit
[1] convert to greyscale
[2] flip horizontally
[3] negative of red
[4] negative of green
[5] negative of blue
[6] just the reds
[7] just the greens
[8] just the blues

Enter choice: 1
outcake.ppm created with effect, “convert to greyscale.”

Do you want to do more operations (y or n): y
Do you want to change input file (y or n): n
Do you want to change output file (y or n): y

Enter name of output file: outcake2.ppm

Here are your choices:
[0] exit
[1] convert to greyscale
[2] flip horizontally
[3] negative of red
[4] negative of green
[5] negative of blue
[6] just the reds
[7] just the greens
[8] just the blues

Enter choice: 2
Outcake2.ppm created with effect, “flip horizontally.”
Do you want to do more operations (y or n): n
Bye bye.
```

Notes on Grading

- All variables should have informative names.
- Your input/output should match the examples.
- Every program must have comments. There should be header comments in your program including your name and a brief introduction of the program. You should have at least one comment besides that.
- Any plagiarism/cheating would be strictly dealt with.
- Use correct indentation and Java code conventions.
- Late penalties apply for late submissions, per the syllabus.
- Failure to send the required files would result in a zero for the assignment.

Rubric

- (10pts) Code cleanliness, conventions, and comments
  - (10/10pts) for >95% correctness and all required comments
  - (7.5/10pts) for repeated minor mistakes or one egregious mistake, or for missing either overall or inline comments
  - (5/10pts) for code that is sloppy and hard to understand or is missing all comments
  - (<=2.5/10pts) for code that has no comments, is hard to understand, and consistently does not follow Java conventions
- (30pts) for Phase I functionality
  - (2.5pts) for allowing user to type in input and output file names
  - (5pts) for creating File and Scanner objects to read from input
  - (10pts) for storing input file data into an appropriate data structure representation, such as 3 2D arrays or a 3D array
  - (2.5pts) for creating a PrintWriter object for output with valid output file name
  - (10pts) for converting this data structure representation to an output file
- (30pts) for Phase II functionality
  - (3.5pts) per method
  - (2pts) for any method being called on the data structure between reading in and outputting to files
    - Since Phase III involves making the method call dependent on the user's choice, as long as at least one method is callable via some input in solutions that have a menu system, these points will be given. If the solution only made it to Phase II, as long as a method is called by the program, these points will be given
- (30pts) for Phase III functionality
  - (5pts) for prompting a user for which mode, calling the appropriate methods
  - (5pts) for program not crashing upon user inputting a non-numeric input, warning them that their choice was invalid
  - (10pts) for program not crashing when an input file isn't found, warning the user that the path was not valid and re-doing the main loop

- (10pts) for the program not crashing when an input file is malformed, e.g. when the header's image width property does not match the actual image width