ASSIGNMENT 4
Image Manipulation: 2D Arrays and Objects

COMP-202, Fall 2016

Due: Monday, December 5th, 2016 (23:55)

You must do this assignment individually and, unless otherwise specified, you should follow all the instructions. Graders have the discretion to deduct up to 15% of the value of this assignment for deviations from the general instructions and regulations. A 75% deduction will be applied for any classes that do not compile. Thus you must make sure that you upload code that compiles and runs, whether or not it generates the correct output.

Part 1:	0 points
Pixel:	10 points
Image:	40 points
ImageFileUtilities	35 points
Comp202Photoshop	15 points
Total	100 points

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)
Write a class for an object called Car that has 3 private attributes: a color, a drive and a modelname. The color and modelname should be of type String, while the drive is an int that takes on a value of 2 or a value of 4. Write two constructor methods for the class. In the first there are no arguments passed to the constructor method and a new Car is simply created. In the second the color, drive and modelnames are passed as arguments to the constructor method and a new Car is created with its private fields initialized accordingly.

Warm-up Question 2 (0 points)
Write a main method to read from a text file of your choice using a Scanner object. The program should take a single command line argument as input, which corresponds to the name of a text file. It should print all of the lines that start with a # symbol to the screen. Recall that you will need to use a try-catch block in order to handle any file related exceptions.

Warm-up Question 3 (0 points)
Write a main method to read from a text file of your choice using a Scanner object. The program should take a single command line argument as input, which corresponds to the name of a text file. The program should use a try-catch block to catch a FileNotFound exception if the filename does not exist. Otherwise it should read from the file line by line. Each line that it reads should be printed but

in reversed order. For example, a line "The quick brown fox" would be printed "xof nworb kciuq ehT". Test your program on a sample ascii text file to make sure that it works properly.

Warm-up Question 4 (0 points)
Write a method that takes a 2D square array of integers as an input and generates a new 2D array as output. The new array should be the original array, but reflected along its diagonal. In other words an entry at location [i][j] in the source array should be copied to location [j][i] in the destination array. Your method should first check to see if the input 2D array is indeed a square array, i.e., its two dimensions are the same. If the input array is not a square array the method should throw an exception.

Part 2

The questions in this part of the assignment will be graded.

In this assignment, a 75% non-compilation penalty will apply. This means that if your code does not compile, you will receive a maximum of 25% for the question. If you are having trouble getting your code to compile, please contact your instructor or one of the TAs or consult each other on the discussion boards. Also make sure that you submit .java files and not .class files!

Background Information

In this assignment, you will write code to read, write, and manipulate images.

Every point in an image is a combination of red, green, and blue light intensities. This (r,g,b) tuple is called a *pixel*. For the purposes of this assignment, each value in a pixel is constrained to be between 0 and 255 (inclusive). Note that if all of the r,g,b values are the same in each pixel, then the image will appear as a greyscale image.[1]

Below is a sample pnm image file taken from Wikipedia to illustrate this idea:

```
P3
# "P3" means this is a RGB color image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value each number could be
# The part below is image data: RGB triplets
# The first number in each triplet is the red intensity, followed by green, then blue
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

The first line (P3) above is a format code that is used by the computer as a flag to indicate that the image is in pnm format.

Directly after the image format may follow one or more comment lines. These are lines starting with a # symbol and can store image metadata, such as the author's name or date modified. Note that some files may not have a comment line.

The next line has the image width and height (in that order). The third line has the maximum range of the pixel intensities.

All subsequent lines contain pixel (r,b,g) values. Note that the numbers can be on the same line or on separate lines. The numbers are listed in "row order." This means that the first 3 numbers are the 3 values

[1]Colloquially, we refer to images such as these as being "black and white." But in reality, these images have more than 2 colours and a more accurate terminology would be "grey image."

(red,green, and blue in that order) for the upper left pixel. The next 3 values are the pixel in the top row, 2nd column. This continues until the first row is complete at which point the next numbers start referring to the 2nd row.

If we know that our image is in greyscale (no colour) we can use the pgm format instead.

```
P2
# "P2" means this is a greyscale image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value each number could be
# The part below is image data. Each number is the intensity value of a single pixel.
# A value of 0 would display as black and a value of 255 as white.
3 2
255
85 85 85
170 255 0
```

Further examples can be found at the wikipedia page at `https://en.wikipedia.org/wiki/Netpbm_format#PGM_example`.

The entry on the first line, P2, is the format code. If it is `P2`, then the format is pgm. If it is `P3`, then the format is pnm. The rest is very similar to the `P3` format described earlier with one difference. Since we know that P2 is grey scale, we only use one number to store the r,g,b values in a pixel. So each number in the P2 data represents an entire pixel, whereas in P3 format, we needed 3 numbers.

You can create and read these image files in a Java program using `Scanner` and `BufferedWriter` as you would any other file. You can view the contents of the files in a text editor such as `notepad++`. To view the files as pictures, you can use a software tool such as Gimp, which can be downloaded at `https://www.gimp.org/`. It is not required to use Gimp if you know of another software that reads .pnm or .pgm image files.

For the purposes of testing your methods, we have provided a .pnm and a .pgm image file of a cat skyping with a rodent who has a teddy bear. You can test your method on other .pnm images as well. If you create your own files to work with we suggest you keep them small (about 256 x 256). In this case. If you convert png, jpg images using gimp, make sure also to select "ascii" (and not raw) as the format.

Before starting this assignment, open the provided image files using both an image viewer and text editor to make sure you can see the contents. **Also, be aware that the code you write should be able to handle rectangular (not necessarily square) images**

Question 1: Pixel Class (10 points)

Define a new type `Pixel`. A `Pixel` has 3 private integer attributes, each representing a red, a green and a blue field. There should be two *overloaded* constructors:

1. One constructor should take as input 3 values: red, green, and blue (in that order). It should initialize the 3 properties of the `Pixel` to be those 3 values respectively.
2. The second constructor should take as input just 1 value `intensity`. It should initialize all 3 properties of the `Pixel` to be that same value of `intensity`.

Both of the above constructors should throw an IllegalArgumentException if any of the intensities are outside of the expected range $[0 - 255]$ (inclusive).

Next, write 3 get methods, getRed(), getGreen() and getBlue() to return the corresponding data field values of a Pixel.

Finally, write a method `grey()` which returns an `int` representing the average of the 3 properties. If the average is not an integer, you should truncate the decimal point. This value will be used later to convert a colour image to greyscale.

Note: You *cannot* add any additional public methods to this class.

Optional (0 points): For those who want to play around with the greyscale a bit, you can get slightly different visual results by using a weighted average instead. Since all colours of light have different wavelengths, our eyes don't process them all the same way - green tends to be more visually significant. You could try using, eg: ((0.3 * Red) + (0.59 * Green) + (0.11 * Blue)) instead of a standard average in the `grey` method above.

Question 2: Image Class (40 points)

Define an `Image` class which is meant to represent the abstract notion of an Image.

An `Image` consists of several private properties:

1. A `String metadata` which can store information about the image in a text format. For example, one might include in metadata the author's name, and the time at which the image was created.
2. An `int maxRange` which stores the upper bound of the values of the pixels. You can assume (that is, no check is required) maxRange is greater than or equal to all of the values in the pixel array.
3. A `Pixel[][] data` which stores the values of the various pixels. The first "row" of the array (i.e. the first array inside the 2D array) stores the first row of the image pixel values. This means that the length of this array is the `height` of the image. The length of each 'sub-array' is the `width` of the image.

Note that the Image class does *not* store the format code. This allows an Image object to be easily exported to multiple formats.

Now add the following methods to this class. **For full marks, you may not add any additional public methods to the Image class. You may add private helper methods if you like though.**

1. A constructor that takes values for all of the attributes as input in the order in which they are listed above. Be sure to create and store a *copy* of the input `Pixel[][]` array (and copy each Pixel in the array as well). Your constructor should throw an `IllegalArgumentException` if the maxRange value is negative.
2. `getMetadata()`: a get method for the metadata attribute
3. `getMaxRange()`: a get method for the maxRange attribute
4. `getWidth()`: a get method for the image width
5. `getHeight()`: a get method for the image height
6. `getPixel(int i, int j)`: a get method for the Pixel object at index i,j in the data array. Note that the above `get` methods should be non-static.
7. A non-static method `flip`, which will be used to flip an image either vertically or horizontally. The method should take as input a `boolean horizontal` and return void. If the boolean is true, it will flip the image horizontally. If the boolean value is false, the flip should be done vertically. Here, we consider a vertical flip to be one in which we swap values across the `x-axis`. A horizontal flip is one in which we swap across the `y-axis`.

   Hint: This method is easier if you create a new 2D Pixel array, populate it with the flipped values, and then update the 2D Pixel array of the calling image.
8. A non-static method `toGrey`, which converts the `Image` to a greyscale image. A grey scale image is one in which the red, green, and blue components in each pixel are the same. Write a non-static method `toGrey` to turn a color Image into a grey scale one. The method should take nothing as input and updates the pixels of the calling image so that it becomes grey scale. This must be done by calling the `grey` method in the `Pixel` class.

For example, if the pixel originally stored at a location is 0 for red, 100 for green, and 200 for blue, the new pixel would store 100 for all 3 attributes. Test your method using a main method to make sure that it creates a grey version of the original .pnm image.

9. A non-static method `crop` which crops a rectangular section of the original `Image`. The method should take as input 4 int values: `startX` (inclusive), `startY` (inclusive), `endX` (exclusive), and `endY` (exclusive). The method should then "cut" the Image so that only the parts of the Image within the coordinates specified by the input are left. If the input arguments are invalid, then your method should throw an `IllegalArgumentException`.

   Note we count the pixels in an image starting from 0. Note also which parameters are inclusive and which are exclusive. For example, to take the first 10 rows and 15 columns, you could call the method with input `crop(0,0,10,15)`

   We have provided some sample code in the file `ImageGenerator.java` to help test this class. You may ignore this if you like. However, we strongly recommend adapting the sample code to test your methods before moving on to the next section.

Image Reading and Writing (35 points)

Now you will write code to actually read and write images. The following should go into a class `ImageFileUtilities`.

1. Write a static method `read` which takes as input a `String filename` representing the relative path to a file on your computer. The method should populate an `Image` object based on the contents of this file.

   Your `read` method must support two types of files: PNM and PGM. Recall: a PPM file is a colour image and a PGM file is a greyscale image.

   Recall the format of the file: First we have a format code (either P2 or P3). Next we have one or more lines starting with a # symbol which represent comments or metadata. Then we have 3 entries which are to be interpreted as the width, height, and maxRange of the image, respectively. You may assume that these values all exist (ie, there is a format code, there is at least one comment, there are values for height, width, and maxRange). You may also assume that the file contains the correct number of pixel values, and that each pixel is between 0 and the specified `maxRange`.

   After this we have several numbers with two possible meanings:

   - If the format is P3, then the numbers are to be viewed in groups of 3 numbers: the red, green, blue triplets for each pixel in the image, which have integer values between 0 and maxRange. In order to construct a single pixel, we need to read in three numbers.
   - If the format is `P2`, then the numbers are to be viewed individually as the number to distribute to all 3 values of an individual `Pixel` (since a grey scale image has the same value for red, green, and blue.) In order to construct a single pixel, we only need to read in one number.

   Your method should throw an `IOException` (remember to import the type IOException from `java.io`) if the format code is not one of the two expected values, or if there is a standard reading error. Since the method throws an `IOException`, you need to add to your method header `throws IOException`.

   Hint: In order to read in all the comments without knowing how many there are, consider using the `hasNext(String s)` method in the Scanner class.

   Refer to the examples in the intro if you're having trouble following how this method works.

2. Write a static method `writePnm` which takes as input an `Image` and a `String filename` and returns void. This method should produce a pnm file (P3 format) based on the contents of the `Image` provided.

3. Write a static method `writePgm` which takes as input an `Image` and a `String filename` and returns void. This method should produce a pgm file (P2 format) based on the contents of the `Image` provided. Be sure to convert the pixels to grey scale before you write this new file.

Note that the last two methods will already, based on the behavior of the `BufferedWriter` class, throw an `IOException` if there is an issue writing the file. It should not be necessary to explicitly add code to throw an exception, but since you are using a method that generates an exception and don't catch it, you will need to add to your method header `throws IOException`.

Comp202Photoshop (15 points)

Lastly, define a class Comp202Photoshop which will contain your main method. The main method should load an image, perform one transformation on it, and then save a new image based on the results. Its task is determined via command line arguments.

The command line arguments should be processed according to the following rules:

1. args[0] represents the name of the input file.
2. args[1] represents the name of the output file.
3. args[2] represents the format of the output file. It should be either `pgm` or `pnm`.
4. args[3] represents the operation to perform. It can be any of 4 values: `-fh` for horizontal flip, `-fv` for vertical flip, `-gs` for greyscale conversion, or `-cr` for crop.
5. args[4], args[5], args[6], args[7] These arguments are present only when the option `-cr` is specified. In this case, they determine where in the image to crop.

For example, if the usage is:

```
run Comp202Photoshop cat.pnm cat-changed.pgm pgm -gs
```

then the program should load the file `cat.pnm` into an Image object, convert it to grey scale, and then save the resulting image into a file `cat-changed.pgm` as a pgm format image.

If the usage is:

```
run Comp202Photoshop cat.pnm cat-cropped.pnm pnm -cr 10 15 30 40
```

then the program should load the file `cat.pnm` into an Image object, crop it by keeping the region from the 11th row, 16th column until the 31th row, 41th column (recall that the counting starts from 0), and then save the resulting image into a file `cat-cropped.pnm` as a pnm format image.

Feel free to assume that the extension of the output file matches the specified output format.

If `args.length` is less than 4, your program should exit gracefully by printing an error message reminding the user of the expected input format. If the output format is invalid, or if the requested operation does not exist, your program should print a helpful error message and exit.

If the operation is `-cr`, then it is required that there be 8 arguments total, the last 4 of which should be `int`. You do not need to check in the main method that 4 pieces of the coordinates are valid for the image, however, you do need to catch the exception that the `crop` method would throw in this event.

If any of the other methods you call generate an error (for example, the file is not found, or the file does not match the proper format), then your program should catch the exception generated and print a user-friendly error message.

Hint: Only make the error 'friendly' after you have finished debugging. Otherwise, you will turn run-time errors into bugs, which makes it much harder to fix your code.

Optional (0 points): add an -id flag to represent the identity operation. This flag indicates that you just want to make a copy of your image. If this flag is used, the code should essentially just read in the image, and then immediately call write, without making any alterations to the Image. This might be extremely useful in testing your code, as it allows you to check that read and write are working, independently of `flip`, `crop`, and `grey`.

What To Submit

You have to submit one zip file called `Assignment4_YourName.zip` with the Pixel.java, Image.java, ImageFileUtilities.java, and Comp202Photoshop.java files in it to MyCourses under Assignment 4.

These files must all be inside your zip. Double check to ensure you have included the `.java` source files rather than the .class files. Also, please ensure after submitting that you have received an email receipt from mycourses confirming that your submission was successful. Even if your submission was successful you should still verify that the files you submitted were the ones you intended to submit.