ASSIGNMENT 3
McGill Paint

COMP-202, Summer 2011

Due: Monday, June 13, 2011 (23:30)

You must do this assignment individually and, unless otherwise specified, you should follow all the general instructions and regulations for assignments. Graders have the discretion to deduct up to 10% of the value of this assignment for deviations from the general instructions and regulations.

Part 1	65 points
Part 2	35 points
Bonus	20 points
	100 points total

Part 1 : Matrix Utilities

Beginning in this assignment, the 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 Dan or one of the TAs or consult each other or the discussion boards.

In this part of the question, you will write several static methods that operate on a matrix (i.e. a 2d array) Don't worry if you don't remember much about matrices as the math in this is not very related to that done prior.

All your code for this section should go into a file `MatrixUtilities.java` in a class `MatrixUtilities`

Update Note: In the course of this assignment, you will deal with array of arrays very often. You can think of the array of arrays as a matrix with the first index representing the vertical row and the second index representing the horizontal row. You may assume that this matrix is rectangular although it is not necessarily square. (No "jagged" arrays)

You should write the following methods:

- `public static int[][] add(int[][] m1, int[][] m2)` This method should return a new matrix computed by adding the contents of m1 to the contents of m2.
- `public static int[][] createFilledMatrix(int value, int height, int width)` This method should return a new matrix of size `height` and `width` in which each entry in the matrix has value `value`
- `public static int[][] scaleMatrix(int[][] m, double scale)` This method should take as input a matrix `m` and a double `scale` and return a new matrix computed by taking each of the original values of m and multiplying it by the factor `scale`. In the case of fractional values, it should truncate the results (i.e. 1.99999 should be come 1)

- `public static int[][] rotateLeft(int[][] m)` This method should return a new matrix which is the same as the original matrix but rotated to the left by 90 degrees. Your method should work for any rectangular matrix.
- `public static int[][] rotateRight(int[][] m)` This method should return a new matrix which is the same as the original matrix but rotated to the right by 90 degrees. Your method should work for any rectangular matrix.
- `public static int[][] horizontalMirror(int[][] m)` This method should return a new matrix which is the same as the original matrix but mirrored across the horizontal axis. In other words, the top row of the original matrix should become the bottom row, the 2nd row should become the 2nd most bottom row, etc.
- `public static int[][] verticalMirror(int[][] m )` This method should do the same as `horizontalMirror` except it should mirror vertically instead of horizontally.

  Hint: Try to use some of the methods you've already written for this one.
- `public static int[][] doubleSize(int[][] m)` This method should return a new matrix with each of its dimensions doubled in size. The way to do this is that every single value in the original matrix should be duplicated 3 additional times so that it is present 4 times in the new matrix. **Update: This method is being moved to assignment 4. It is not required for assignment 3**
- `public static int[][] halfSize(int[][] m)` This method should return a new matrix with each of its dimensions halved in size. The way to do this is to "compress" the matrix by taking each 2 by 2 block of the original matrix, averaging the 4 values, and replacing it by the 1 average. You may assume that the matrix has even dimensions in both directions.**Update: This method is being moved to assignment 4. It is not required for assignment 3**

Part 2 : Image Manipulation

In this section, you will use the Matrix tools you wrote in the first part, along with some other tools provided to you, to write an image manipulation program.

Before getting started writing code for this section, you should familiarize yourself with what is provided to you.

First, you should first download the file Assignment3.zip and extract the files "Comp202ImageIO.java", "ImageManipulation.java", "McGillPaint.java" and "rangers.jpg" from the course website. For this part, you will only make changes to "ImageManipulation.java" Put all of these files in the same directory as the `MatrixUtilities.java`

The next thing to do is make sure you can compile and run the program as it is. Try compiling the program by typing:

```
javac Comp202ImageIO.java ImageManipulation.java McGillPaint.java MatrixUtilities.java
```

or

```
javac *.java
```

Note that the above operation will try to compile every java file in the current folder you are in, so it will be slow if you have many files (if for example, you put all java files you've ever written into the same folder)

The main method is in the class `McGillPaint` Try running this by typing `java McGillPaint` If you have trouble running it from the console on a Mac, try running it via the program x11 instead. `http://support.apple.com/downloads/X11_Update_2006_1_1_3` On windows it should be identical as before. The program works if you see the Rangers beautiful logo on your screen.

The point of this question is going to ask you to make various manipulations to the image in memory. For example, one method you write will convert the image from a colour image to black and white. These methods sound complex, but will mostly be applications of the matrix utilities method you wrote in the previous section.

Background on images

There are many ways to store a colour image on a computer. Normally, however, one thinks of the screen as a a matrix of pixels. Each pixel is a certain colour, and the colour is stored by representing a red, a green, and a blue component. These three colours are sort of like the primary colours of computers. Each one of these components is an integer from 0 to 255. Here are some example colours

Red component	Green Component	Blue Component	Colour
255	0	0	red
0	255	0	green
0	0	255	blue
0	0	0	black
255	255	255	white
255	255	0	yellow
0	255	255	aqua
255	0	255	violet

(You can try out more values at `http://www.colorschemer.com/online.html`)

Each of these numbers ranges from 0-255. The number of bits needed to fit this is 8. Since there are 3 components, the total size of a pixel would be 24 bits. One odd thing about this, is that and int on a computer is normally 32 bits. People working with images decided that rather than wasting the additional 8 bits, they could use the remaining 8 bits to represent an *alpha* channel. The alpha channel controls whether an image is transparent or not. We will not worry about the alpha channel for this assignment.

So typically an image in memory can be thought of as 4 two dimensional arrays. One for alpha, one for red, one for green, and one for blue.

The upper left corner of an image is normally the “origin” of an image and the x values increase to the right while the y values increase down.

What has been provided

You have been provided with a couple of utility methods to make reading and writing from files more natural. These are inside the file `Comp202ImageIO.java`

The methods provided are the following:

- `public static int[][] loadImage(String inputFile)` This method will load the image with the path `inputFile` It will return a 2d array. Recall that a 2d array is an array where every element is an array. The first array is the alpha array, the second array is the red array, the third array is the green array, and the fourth array is the blue array.

  One thing interesting is that each of these separate arrays is actually a 1d array. This is an optimization that Java does to make things run faster. This 1d array is the normal image grid rearranged so that all the columns, are following each other on one row. For example, if the image is 5 rows by 3 columns, then the resulting array would be size 15 where the first 3 columns are row 1 the next 3 columns are row 2, etc.

  If you want to go from 1d coordinates to 2d coordinates, a very useful transformation is that the value stored in the ith row and jth column shows up in location $i * width + j$ in the 1 dimensional array.

- `public static int getImageWidth (String inputFile)` **Note: NOT getWidthImage as previously written** This method will calculate the width of the given image file.
- `public static int getImageHeight (String inputFile)` **Note: NOT getHeightImage as previously written** This method will calculate the height of the given image file.
- `public static BufferedImage getBufferedImage(int[][] imageData, int height, int width)` You do not need to call this method although it is used by other code given to you.
- `public static void savePng(int[][] imageData, int height, int width, String filename)` You do not need to use this method at all. If you wish to play around with things (for fun) you can use it to save a png file.

In addition to what is given here, the program is set up, so that whatever is present in the instance variable `imageData` of the `ImageConverter` class and the end of the constructor will be displayed to the screen. So when you are testing your methods, you should modify the constructor so that it calls the appropriate methods you've written.

What you have to do

You will have to write several methods. If you did the first part correctly, most of these methods will not be very long. Most of the methods will modify the instance variable `imageData` by individually modifying each of the 4 fields in it. Feel free to add any additional instance variables or helper methods.

- A constructor `ImageConverter()` . The constructor should take as input a `String` representing the filename to load and a `String[]` args. (Don't worry about args for now)

  The method should call the method `loadImage` provided to you with the given file and store the results into the instance variable imageData. It should also store the width and height as instance variables somewhere. You may, if you wish, add additional instance variables if you find them useful.
- Two getters, `getWidth()` and `getHeight()` which should return the width and height of the image respectively. These methods are already present in the file ImageConverter.java but you will have to update them to no longer be fixed sizes.
- A method `doubleSize` (see update!) This method should double the size of the image. It can do this by calling the method you wrote in `MatrixUtilities.java` on `imageData[0]` , `imageData[1]` , `imageData[2]` , and `imageData[3]` . It should then produce a new 2d array from the results and store a reference to this array into `imageData` **Update: This method is being moved to assignment 4. It is not required for assignment 3**
- A method `halfSize` (see update!) This method should halve the size of the image referenced by imageData using a similar transformation to that done on matrices in part 1. **Update: This method is being moved to assignment 4. It is not required for assignment 3**
- A method `rotate90DegreesClockwise` This method should rotate the image referenced by imageData 90 degrees clockwise. Remember that this will affect the image width and height and you should change instance variables accordingly.
- A method `rotate90DegreesCounterclockwise` This method should rotate the image referenced by imageData 90 degrees counter-clockwise. Remember that this will affect the image width and height and you should change instance variables accordingly.
- A method `flipHorizontal()` This method should flip the image referenced by imageData across the horizontal axis.
- A method `flipVertical()` This method should flip the image referenced by imageData across the vertical axis.

- A method `convertToBlackAndWhite()` (see update!) The way to convert an image to black and white is by assigning to each pixel the average value of the red,green, and blue components. For example, if the original image had 200 for red, 150 for blue, and 100 for green, then the new image should have 150 for all 3. **Update: This method is being moved to assignment 4. It is not required for assignment 3**

  This method should convert the image referenced by imageData to black and white.

- A method `getNegativeImage()` (see update!) One can convert a black and white image to a negative image by "flipping" the values of all components. To do this, each component of each pixel should have the value $255 - x$ where x was the original value. Write a method `getNegativeImage()` that converts a black and white image to the negative of the image. **Update: This method is being moved to assignment 4. It is not required for assignment 3**

  Hint: If you are clever about things, you can use the methods in `MatrixUtilities` here once again to avoid writing much code.

Bonus (20 points)

The bonus will involve adding additional functionality to both the `MatrixUtilities` class as well as the `ImageConverter` class. It will require you to write 2 methods, one in each class.

In `MatrixUtilities.java` add the following method:

`public static int[][] rotate(int[][] m, double degrees)` This method should return a new matrix which is the original matrix rotated *clockwise* by `degrees` . You may assume for this that the matrix is square. Elements of the matrix that are away from the middle and would not be part of the resulting matrix (such as corner elements) can be assigned to be 0.

To figure out what value to put, you may use the following formula:

In the resulting matrix, the value at the ith row and jth column will come from the value in the k row and l column of the original matrix where:

$$k = (i - (width/2)) \times cos(degrees \times \frac{\pi}{180}) + (j - height/2) \times -sin(degrees \times \frac{\pi}{180}) + width/2$$

$$l = (i - (width/2)) \times sin(degrees \times \frac{\pi}{180}) + (j - height/2) \times cos(degrees \times \frac{\pi}{180}) + height/2$$

Note that since we are assuming a square matrix, that width and height are the same.

Then, in `ImageConverter.java` add the following method:

`rotate(double degrees)` This method should rotate a square image referenced by imageData by `degrees` clockwise. It is okay to set the new corner areas to be black.

What To Submit

`confession.txt` - You should write in this file any information that you think is useful for the TA to mark the assignment. This should include things you were not sure of as well as parts of your code that you don't think it will work. Of course, like a confession, this will draw the TA's attention to the part of your code that doesn't work, but he/she will probably be more lenient than if he/she has to spend a lot of time looking for your error. It demonstrates that even though you couldn't solve the problem, you understand roughly what is going on.
`MatrixUtilities.java`
`ImageConverter.java`