ITIS 1212 – Assignment 4 – Pixel Fun

The main idea behind this assignment is to demonstrate your understanding of nested for loops and the 2D array structure that underlies Picture objects. You will show that you know how to work with the indices in for loops to get access to different pixels in an image. After completing Labs 10 and 11, you should have all the skills you need to complete this assignment. Remember that assignments are to be done individually.

Part 1 – Shrink! Shrink images to ½, ⅓ or ¼ of their original size. (15%)
Part 2 – Reveal! Peel back one picture to reveal another. (15%)
Part 3 – Checkerboard! Add a colored checkerboard over your picture. (30%)
Part 4 – Alpha Checkerboard! Add a transparent checkerboard over your picture. (10%)
Part 5 – Dual-Picture Checkerboard! Merge two pictures through a checkerboard. (10%)
(Note: 5 parts may seem like a lot, but parts 4 & 5 are quick once you figure out Part 3!)

Note: Your name MUST appear in a comment in the first line of every java file you submit. If your name is not in the first line of every java file you submit, a total of 10 points will be deducted. No exceptions.

Part 1 – Shrink! (15%)

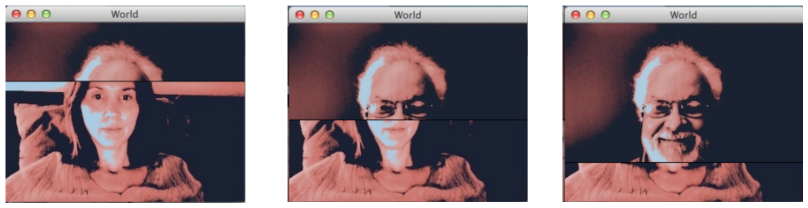
For Part 1 of the assignment, use the base code Assignment4Part1.java as a test file. Write a **shrink(int factor)** method in the Picture.java file that takes an integer parameter called ‘factor', creates a new Picture object that is scaled down from the original and return the new Picture object. If factor is 2, shrink the picture to half and return a new picture that is half the size. If factor is 3, shrink the picture to a third the original size and return the smaller image. If the factor is 4, shrink the picture to a quarter the original size and return the smaller image. Your shrink method will need to do the following things:

1. Check the precondition that the factor passed in is 2, 3, or 4.
2. Create a new Picture object that is the correct size. (Important Hint: in creating a smaller Picture object, add one pixel to the width and one pixel to the height because when you divide the original height by two or three, Java will not round up, and you want to make sure your target image is large enough).
3. Copy every second, third or fourth pixel into the new image from this image object, depending on factor.
4. Return the new Picture object.

You will save yourself a lot of time if you look carefully at program 30 in the textbook, which shows how to shrink an image to half the size (factor of 2). Your job is to generalize this. Use the Assignment4Part1.java file to test that your method works correctly.

Part 2 – Reveal! (15%)

For Part 2 of the assignment, you are to create a method in Picture.java called **reveal(Picture underPic, double division_height)**, that takes a second Picture object as the first parameter and a double percentage value as the second parameter. What reveal does is show part of the original picture and part of the revealed picture. The percentage value passed in determines how much of each image to show: a low percentage shows mostly the original, and a high percentage shows mostly the underPic. By repeatedly calling the reveal function with different percentages, we can create an effect of peeling away the top (original photo) to reveal the second (underPic) underneath. The peeling is vertical, with one row of black pixels in between the two images, like this where Celine's image is peeled back to reveal Bruce's image:

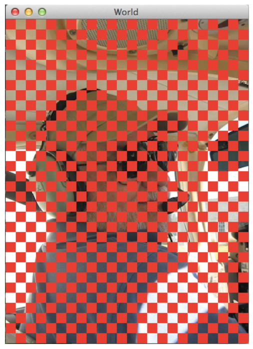


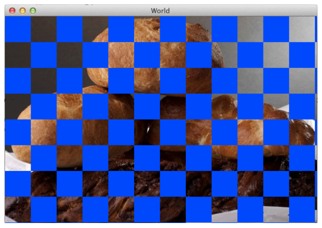
To write this method, you will need to do the following:

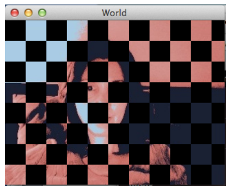
1. Check the precondition that the percentage passed in is between 0.0 and 1.0.
2. Check the precondition that the two images are the same size (use your private method from Lab 11!)
3. Create an integer variable that keeps track of where the dividing line should be, based on the percentage that is passed in.
4. Write a nested for loop that iterates through the image. Pixels above the dividing line should be copied from the passed in picture. The dividing row of pixels should be set to black. The rest of the pixels should be left alone.

The file Assignment4Part2.java has code that will test your reveal method by calling it 100 times with a slight pause in between, so you should see an animation of the first picture being peeled down to reveal the second picture.

Part 3 – Checkerboard! (30%)



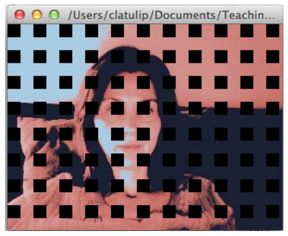




In this part of the assignment, you need to write a **checkerboard(int width, Color c)** method that takes two parameters: the size of the checks and the color of the checks. This method should iterate through the image pixels, turning the appropriate pixels the color passed in. The three images above show the method run on three different input photos, with a different check size and color passed in each time. With the picture PilotBruce.jpg, we called checkerboard(15, Color.RED). With the bread picture, we called checkerboard(50, Color.BLUE) and with the warholized picture of Celine, we called checkerboard(30, Color.BLACK). In order to write this method, you will need to:

1. Check the precondition that the check width passed in is less than ½ the width of the picture object.
2. Iterate through every pixel in the image.
3. Check each pixel to see if it should be checkerboarded (this is the hard part, sketch this out on paper to try figure it out)
4. If the pixel should be checkerboarded, change it's color to the passed in color.

Hint: work on figuring out how to put the checks in the odd rows first. So, get something like this going:

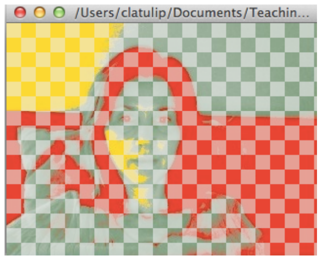


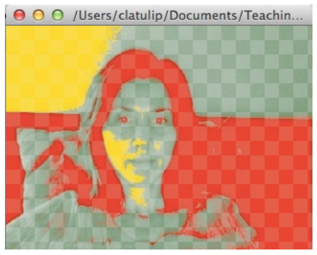
Then, once you get that working, figure out how to also put checks in the even rows, but staggered over.

Further hint: you'll really need to get out some paper and work on this with a pencil to figure out the pattern and how to program it. Labeling graph paper with array indices would be useful for this. You need to come up with a way to mathematically determine when the checks should be drawn. The modulus operator will be useful. Look back at the process we used in Lab 10, Part B, that may be helpful to you for a way to think this through.

Part 4 – Alpha Checkerboard! (10%)

In this part of the assignment, you need to write a **checkerboard(int width, int alpha)** method that instead of making color checks, makes checks by adjusting the transparency value of the pixels in the checked area. Here is an example of what this looks like, with an alpha value of 100 on the left and an alpha value of 200 (more transparent) on the right:



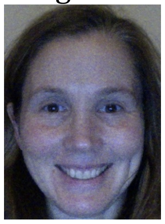
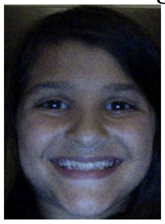
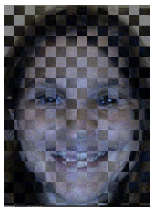


We haven't talked about transparency in pictures, but image pixels can have a fourth piece of information, which is the transparency value, which ranges either from 0 to 255 (this is the way the Pixel class works) or sometimes stored as a decimal value between 0.0 and 1.0. The In order to work with the transparency of pixels, the image must have an 'alpha channel' added to it before you open it up in Dr.Java. Lots of image editing programs allow you to add an alpha channel to an image (usually this is located in the same menu as the color adjustment tools). We have provided the image above as a test image for you to use, but it's good to find out for yourself how to add an alpha channel to your images. This is a very useful feature when you want to make images that appear to float on top of other images (such as on web pages).

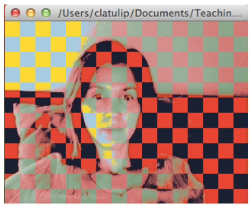
For this part of the assignment, you'll be making a copy of the first checkerboard method, and changing the second parameter to be an integer that should range from 0 to 255. Then, use that value with the setAlpha() method that is part of the pixel class, instead of using the setColor() method that you used in the previous Part. Test your new checkerboard method with a picture that has an alpha channel, such as Assignment4AlphaPic.png. Try it out with various alpha levels.

Part 5 – Dual Picture Checkerboard! (10%)

In this part of the assignment, you need to write another checkerboard method. This one will be **checkerboard(int width, Picture pic)** and takes two parameters: the size of the checks and a second picture that is the same size as the current picture object. Instead of the checkerboard having colored in or transparent checks, the area of the checks is going to be taken from the second photo, like below where Celine's headshot is merged with her daughter's headshot using the checkerboard(…) method:

Or, in this example, two different 'warholized' pictures of Celine are checkerboarded together for a different kind of effect:



To do this part of the assignment, you can copy and paste the Part 4 method, and change the second parameter so that a Picture is passed in. To copy pixels from the passed in picture to the first picture, you can use the private copyPixel(…) method that you created in Lab 11. This should be a very simple quick change. Make sure you test for the precondition that the two pictures are the same size, in addition to checking the width of the check.

Coding Style – (15%)

This grade is awarded for proper coding style. This includes:

- Appropriate method, variable, field, object and class names
- Proper indentation
- Good commenting (explains what code is doing)
- Well-organized, elegant solution

What to submit:

Note: Your name MUST appear in a comment in the first line of every java file you submit. If your name is not in the first line of every java file you submit, a total of 10 points will be deducted. No exceptions.

- Picture.java (Yep, that's it!)

Grading Rubric:

Part 1 Shrink(), checks precondition	Not checked *0 points*	Checks lower bound *2 points*	Checks upper bound *3 points*	Checks both *5 points*
Part 1 Shrink(), creates new picture	No picture created *0 points*	Picture created but without using factor parameter *2 points*	Picture uses factor parameter, but without hint *3 points*	Picture creation uses factor parameter and hint *5 points*
Part 1 Shrink(), copies pixels into new pic	Pixels not copied into new pic *0 points*	Pixels copied but not correctly *2 points*	Pixels copied in correctly *5 points*	
Part 2 Reveal(), preconditions checked	Not checked *0 points*	Checks pictures same size *2 points*	Checks division number *3 points*	Checks size and number *5 points*
Part 2 Reveal(), division line calculated correctly	Division parameter not used *0 points*	Not calculated correctly *2 points*	Calculated correctly *5 points*	
Part 2 Reveal(), pixels adjusted appropriately	Pixels not adjusted at all *0 points*	Black line added, but not underPic *2 points*	UnderPic added but not black line *3 points*	Both underPic and black line show correctly *5 points*

Part 3 Checkerboard(), preconditions checked	Not checked *0 points*	Checked but not correctly *2 points*	Checked correctly *5 points*	
Part 3 Checkerboard(), color parameter used	Not used *0 points*	Used, but not correclty *2 points*	Used correctly *5 points*	
Part 3 Checkerboard(), pattern calculated correctly	Not done at all *0 points*	Checks drawn without calculation *5 points*	Checks drawn through some calculation but checksize is hardcoded, or calculation only partially works *10 points*	Calculated correctly, adjusts to different checksize parameters *20 points*
Part 4 Alpha Checkerboard, preconditions	Not checked *0 points*	Check width checked *2 points*	Alpha level checked *3 points*	Both check width and alpha level checked *5 points*
Part 4 Alpha Checkerboad, alpha changes appropriately	Alpha not used *0 points*	Alpha used, but not correctly *2 points*	Alpha used correctly *5 points*	

(continued on next page)

Part 5 Dual Picture Checkerboard, preconditions	Not checked ***0 points***	Check width checked ***2 points***	Picture size checked ***3 points***	Both check width and picture size checked ***5 points***
Part 5 Dual Picture Checkerboard, pixels change	Not working ***0 points***	Pixels from other picture used, but not correctly ***2 points***	Pixels from two image merge appropriately ***5 points***	
Coding Style	Poor indentation, bad naming style, poor or no commenting ***0 points***	Mediocre style, some okay commenting, indentation and naming, but lots that is problematic. ***5 points***	Good style, but solution is not as clear or organized as it could be ***10 points***	Excellent style - appropriate commenting, good naming conventions, excellent indentation, clear and organized solution ***15 points***
Excellence. most people will get zero, we will give at most 10% of the class these final five points, which demonstrate excellence). This is purely at the discretion of the TAs and Professors.	Normal effort. ***0 points***		Excellence - for effort above and beyond the assignment requirements. ***5 points***	