Summary
Add a method to the Picture class to give the image a vignette effect. See the samples at the end.

Deadline
This assignment is due Saturday, March 3, at 11:59PM. Late submissions are accepted for up to 7 days after that, with the score reduced by 14% per day late, computed almost continuously.

Prerequisites
Lecture notes on loops, problem solving and pixels; Chapter 4 of the textbook and necessary material from previous chapters.

Description
Create a new method, named `vignette`, taking no parameters, at the bottom of `Picture.java` (just before the final closing curly bracket). The new method must be programmed to do the computations described mostly in English below whose effect is to convert the image into one with the vignette effect applied. Since your method must be an "instance" method (like the draw letter methods you gave to `Turtles` of Project 1), your code uses the keyword `this` to get at the `Picture` to be modified.

The new method MUST be named `vignette` and take no parameters: Otherwise, your TA will work harder to test your work and you will lose points.

Besides the `vignette` method, you must re-program the main method so that when the TA runs the program, it (1) brings up the `FileChooser` and, if an image file is successfully chosen, it (2) makes a `Picture` from the file and (3) shows the `Picture` after it was modified by the computer calling your `vignette` method upon it.

Extra Credit
10 points: Besides the regular method, create a parametrized version declared `void vignette(int x1, int y1, int x2, int y2, Color cornerColor)`. When this (polymorphic) version of vignette is run on a `Picture`, it transforms precisely the part of the image inside the rectangle with opposite corners `(x1,y1)` and `(x2,y2)`, leaving the rest of the image unchanged. And, the corner color is given, not always white. You can make your regular credit method simply call your extra credit one: This will save you time if you understand parametrized methods well.

OR 20 points: Like the 10 point problem, but instead of vignetting all of of the rectangle, it vignettes precisely the contents of a circular disk. That disk is centered at the center of the rectangle. The radius of the disk is the minimum of half the disk's width and half the disk's height. If you do this successfully, you may make your whole Picture vignetting method do circular disk vignetting instead.

Grading
The basis is 100 points, including extra credit. However, the score will be weighted so 80 points means 100%.
To make a vignette--What to program the computer to compute:

Before the loop
1. The computer must know, more exactly, write down, the width and the height of the Picture
2. From the width and height, she or he can calculate the x and y coordinates of the center point. Just divide the width and the height by 2.0. Let's use variables \textit{xC} and \textit{yC} for those coordinates. They should be of type \texttt{double} because dividing an integer by 2 can leave a fractional part.
3. From the coordinates of the center point, he or she must calculate the distance of the center to each corner. Since the center is in the middle, it doesn't matter which corner. So the easiest corner is the one with coordinates (0.0, 0.0), the upper left corner.

The distance to that corner is calculated by the Pythagorean Theorem applied to the right triangle shown below.

4. So the calculation is (1) compute the square of each center point coordinate (\textit{xC*xC} and \textit{yC*yC}) (2) add those squares and (3) thinking that the sum is \textit{S}, take the square root returned by \texttt{Math.sqrt(S)} That square root is the maximum distance \textit{distMax} of all \texttt{Pixel}s in the Picture to the center point. All \texttt{Pixel}s are within that distance to the center.

The above common computation is done ONLY ONCE, to make a single vignette. You have to arrange that the remaining computations be controlled by a loop. We now describe the computation done once for each \texttt{Pixel}. See lecture notes and Sec. 4.3 for several kinds and applications of loops to \texttt{Picture}s and arrays of \texttt{Pixel}s. The only result of the above (common) computation that will be used over and over for each \texttt{Pixel} is \textit{distMax}'s value. Sec. 4.3 also has many examples of taking \texttt{int} values, converting them to \texttt{double}, doing the arithmetic possibly with fractions, and casting the resulting \texttt{doubles} to \texttt{int} with \texttt{(int)} so certain G&E methods accept those parameters.

Inside the loop body
1. The \textit{x} and \textit{y} coordinates of the \texttt{Pixel} are needed. A G&E \texttt{Pixel} object has the methods you can use to return these coordinates: Supposing variable \texttt{pixel} refers to the \texttt{Pixel}, calls to those methods are coded by \texttt{pixel.getX( )} and \texttt{pixel.getY( )} (Or you can use nested loops to set the two variables \textit{x} and \textit{y}. Both ways are explained and applied in the textbook.)
2. The original \texttt{Color}'s three intensities (red, green and blue) are needed too. Use method calls \texttt{pixel.getRed( )}, \texttt{pixel.getGreen( )} and \texttt{pixel.getBlue( )} to return those three numbers.
3. How the color is changed depends on the distance of the pixel to the center point. Your computer will have to compute that distance using the Pythagorean theorem, again. The sides of the relevant triangle have lengths \((x - xC)\) and \((y - yC)\). So program the computer to (1) square each of these side lengths (Just multiply each number by itself), (2) add the two squares and (3) compute the square root of the sum. Let's say he or she wrote that square root onto ticket `distPix`

4. The key quantity that determines the color change is the ratio of `distPix` to `distMax` (actually, their values.) The ratio will range from 0.0 for the center pixel to 1.0 for a corner pixel. So, your computer must use the Java `double` type of number. For the reason explained below, let's store this ratio as `whiteColorWeight = distPix/distMax`

5. Now think about how your weighted GPA (grade point average) is computed. Suppose you took only two courses: Course P with say 9 credits and another course Q with say 13 credits. Course P has weight of \(9.0/21\) or about 0.43 and Course Q's weight is \((1.00-0.43)\) or 0.57. So, if you got a 4.0 (A) in course P and a 3.0 (B) in course Q, your weighted GPA would be \((0.43*4.0) + (0.57*3.0)\). You do not divide this sum by anything because the weights already add up to 1. The familiar non-weighted average of two numbers N1 and N2 is the weighted average with equal weights 0.5: \((0.5*N1 + 0.5*N2)\).

Here is the answer: The new red intensity is the weighted average of 255 with weight `whiteColorWeight` averaged with the original red intensity with weight \((1.0-\text{whiteColorWeight})\) Of course, 255 is the red intensity for color white.

The new blue intensity and the new green intensity are separately computed in similar fashion.

6. Remember that after the new red, green and blue intensities are computed, the G&E `Pixel` methods that change those three intensities must be called. Call those methods like `pixel.setRed( ?? );` etc., with the correct variable or expression in place of the `??`. These three method calls are of course inside the loop body.

7. And just in case you forget to add 1 to the correct variable or variables for each run of the loop body, it's very helpful to know how to stop DrJava when it's running a program with an infinite loop: Click on "Reset" (just right of the "Compile" button.)
(Vignetted region is chosen by the parameters.)