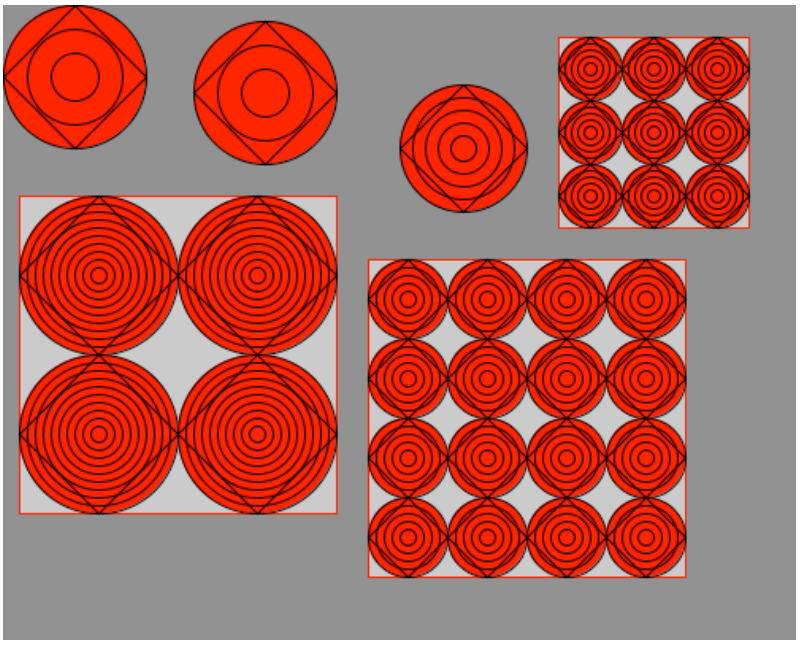
CSE 142, Winter 2016
Programming Assignment #3: Illusion/Doodle (40 points)
Due: Tuesday, January 26, 2016, 11:30 PM

Program Description:
This assignment covers parameters and graphics. Turn in Java files named `Doodle.java` and `Illusion.java`.

To compile and run this assignment, you must download the file `DrawingPanel.java` from the Homework section of the class web page and save it in the same folder as your code. Do not turn in `DrawingPanel.java`.

Part A: `Doodle` (4 points):
For the first part of this assignment, turn in a file `Doodle.java` that draws a figure using the `DrawingPanel` provided in class. You may draw any figure you like that is at least 100 x 100 pixels, contains at least three shapes i.e. three calls to draw…( ) or fill…( ), uses at least two distinct colors, is your own work, and is not highly similar to your figure for Part B. Your program also should not have any infinite loops and should not read any user input. Your score for Part A will be based solely on external correctness as just defined; it will not be graded on internal correctness.

Part B: `Illusion` (36 points):


For the second part of this assignment, turn in a file named `Illusion.java` that draws several instances of the Ehrenstein illusion. In this illusion, the sides of a square placed inside a set of concentric circles look curved. Your program should exactly reproduce the image at left.

This image has several levels of structure. There is a basic "subfigure" that occurs throughout, containing concentric circles inside it. The subfigure is repeated to form grids.

The overall drawing panel is size **500 x 400**. Its background is gray. The square area behind the grids is light gray (`Color.LIGHT_GRAY`), and the background of the circles is red. The circles are outlined in black and the squares in red. A black diamond outline covers each set of concentric circles.

The six figures on the panel should have the following properties:

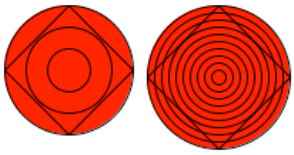
Description	(x, y) position	size of subfigure	circles per subfigure	rows/cols
top-left	(0, 0)	90 x 90	3	N/A
top-second-left	(120, 10)	90 x 90	3	N/A
top-middle	(250, 50)	80 x 80	5	N/A
bottom-left	(10, 120)	100 x 100	10	2
top-right	(350, 20)	40 x 40	5	3
bottom-right	(230, 160)	50 x 50	5	4

You can use the DrawingPanel's image comparison feature (File, Compare to Web File...) to check your output. Different operating systems draw shapes in slightly different ways, so it is normal to have some pixels different between your output and the expected output. You do not need to achieve 0 pixels difference to get full credit for your output. If there is no visible difference to the naked eye, your output is considered correct. (If your figure looks the same but has "thicker" black lines, you may be re-drawing the same shapes multiple times.)

Implementation Guidelines for Part B:

To receive full credit on Part B, you are required to have two particular static methods described below.

1. Method to draw a subfigure



Your first method should draw one single concentric circle subfigure. A subfigure is one set of red and black concentric circles and a black diamond, such as those at left. Different subfigures have different sizes, positions, and so on. Therefore, your method should accept several parameters so that it is possible to call it many times to draw the many different subfigures on the screen.

You should assume that every subfigure has equal width and height, and that the subfigure's size is a multiple of its number of circles, so that all coordinates are integers.

2. Method to draw a grid

Once you have completed the method that produces one subfigure, write another method that produces a square grid of subfigures. Each square grid has a light gray background and a red outline. It will need a lot of parameters to be flexible enough to draw each of the grids in the final image. The key point is that this single method can be called multiple times to produce all the grids in the overall figure. Your two methods should work together to remove redundancy. Assume each grid has an equal number of rows and columns.

Place the following statement at the top of your Java files, so that your code can use graphics:

```
import java.awt.*;    // so that I can use Graphics
```

Development Strategy (How to Get Started):

This program does not require as many lines of code as past ones. But the numeric computations and parameters are not simple. Write your code in stages, repeatedly making small improvements. Start by having your first method draw only the upper-left subfigure, then generalize it by **adding one parameter at a time**. For example, add parameters to change the x/y position. Test the parameter by passing different values. Once it works, move on to the next.

It may help to compute a value that we'll call the **"gap,"** or the distance between neighboring pairs of circles in a subfigure. For example, the 90x90 top-left subfigure has 3 circles, and each circle has a gap of 15 pixels from the others (a total of 6 gaps). Look at the subfigures to be drawn and try to find the relationship between their various properties and the resulting gap. Each subfigure uses a different gap value based on its parameters. Notice that the diamond has not yet been added in the screenshot to the left.

Debugging:

Part of the challenge of assignment is managing the complexity of drawing several figures. We've provided you with a special version of the Graphics object to help you debug your program. More information is linked below

http://courses.cs.washington.edu/courses/cse142/16wi/hw3_faq.shtml

Style Guidelines:

For this assignment you are limited to the language features in Chapters 1-3 of the textbook.

We require at least the two methods named previously. You may use additional methods if you like. You may receive a deduction if your methods accept too many parameters or unnecessary parameters. An "unnecessary" parameter in this case is one whose value is redundant with another's or could be computed using others' values.

Give meaningful names to methods, variables, and parameters, and properly indent your code. Follow Java's naming standards as specified in Chapter 1. Limit the lengths of your lines to fewer than 100 characters. Include meaningful comment headers at the top of your program and at the start of each method.