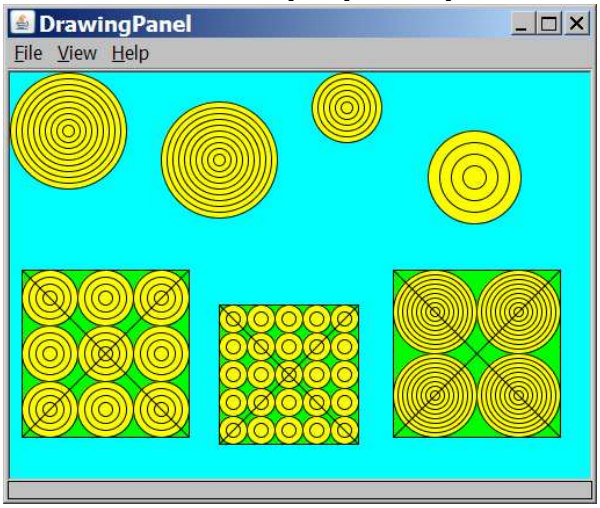
CSE 142, Autumn 2009
Programming Assignment #3: Doodle (20 points)
Due: Tuesday, October 20, 2009, 11:30 PM

Program Description:
This assignment covers parameters and graphics. Turn in Java files named `Doodle.java` and `Circles.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` (2 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, 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.

Be creative! Student doodles can be posted on our Facebook application as part of a voting contest.

Part B: `Circles` (18 points):


The second part of this assignment asks you to turn in a file named `Circles.java` that draws a specific figure of grids of concentric circles. Your program should exactly reproduce the image at left. (The image at left was taken on Windows; your window may look slightly different.)

The Part B 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 larger grids.

The overall drawing panel is size **500 x 350**. Its background is cyan. The rectangular area behind the grids is green, and the background of the circles is yellow. The rectangles and circles are outlined in black. Each grid also has a pair of lines drawn across it in an "X" pattern.

The seven figures on the panel should have the following properties.

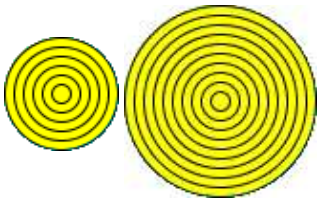
Description	(x, y) position	size of subfigure	circles per subfigure	rows/cols
top-left	(0, 0)	100 x 100	10	N/A
top-middle 1	(130, 25)	100 x 100	10	N/A
top-middle 2	(260, 0)	60 x 60	6	N/A
top-right	(360, 50)	80 x 80	4	N/A
bottom-left	(10, 170)	48 x 48	4	3 x 3
bottom-middle	(180, 200)	24 x 24	2	5 x 5
bottom-right	(330, 170)	72 x 72	9	2 x 2

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. These methods use a great deal of parameter-passing and perform the program's complex numeric computations.

1. Method to draw a subfigure



Your first_method should draw one single concentric circle subfigure. A subfigure is one set of yellow and black concentric circles, 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's width and height are the same, 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. You will call this method three different times from `main` to produce the grids of the overall figure. It will need a lot of parameters to be flexible enough to draw each of these grids. 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 (our solution is under 65 lines). But the numeric computations and parameters are not simple. You might be overwhelmed with the amount of detail you have to handle all at once. As famous computer scientist Brian Kernighan once said, "Controlling complexity is the essence of computer programming." To make things easier, begin with a smaller piece of the problem.

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 100x100 top-left subfigure has 10 circles, and each circle has a gap of 5 pixels from the others (a total of 20 gaps in each direction). 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. You could also store the gap in a variable and use it in your subfigure code.

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.

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.