CSE 142, Autumn 2007
Programming Assignment #3: Doodle / Circles (20 points)
Due: Tuesday, October 16, 2007, 4:00 PM

Program Description:

This assignment will give you practice with parameters and graphics. This assignment has 2 parts; turn in two Java files named `Doodle.java` and `Circles.java`.

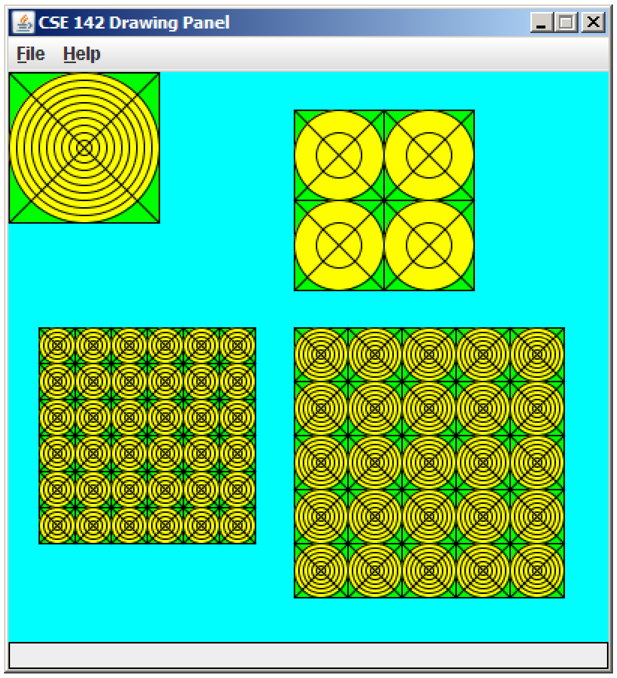
To compile and run this assignment, you must download the file `DrawingPanel.java` from the Homework section of the class web page and save it in the same folder as your code. Do not turn in `DrawingPanel.java`.

Part A: Doodle (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, uses at least two distinct colors, is your own work, and is not highly similar to your figure for Part B. Be creative! Student doodles will be posted anonymously on the course web site as part of a voting contest.

You may optionally use parameterized methods for any repetition in your figure. However, your score for Part A will be based solely on external correctness as defined above; it will not be graded on internal correctness.

Part B: Circles (16 points):



The second part of this assignment asks you to turn in a file `Circles.java` that draws a specific complex figure of grids of concentric circles. Your program should exactly reproduce the image at left. (The image in this document was taken when running the program on Windows; if you use another platform, the border around the 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 grid figures.

The overall drawing panel is size **400 x 380**. Its background color is cyan. The rectangular area behind the circles is green, and the background of the circles is yellow. The rectangles and circles are outlined in black. Each subfigure also has a pair of lines drawn across it in an "X" pattern.

The four figures on your drawing panel should have the following properties.

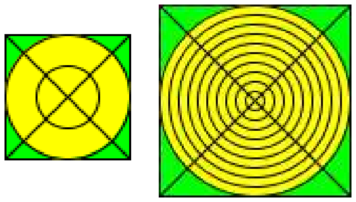
Description	**(x, y) position**	**size of subfigure**	**circles per subfigure**	**rows/cols**
top-left	(0, 0)	100 x 100	10	1 x 1
bottom-left	(20, 170)	24 x 24	4	6 x 6
top-right	(190, 25)	60 x 60	2	2 x 2
bottom-right	(190, 170)	36 x 36	6	5 x 5

You can use the `DrawingPanel`'s image comparison feature to check your graphical output.

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.

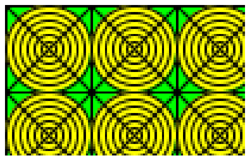
1. Method to draw a subfigure



Your first method should draw one single concentric circle subfigure. A subfigure is one green rectangle with its 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.

Assume that every subfigure's size is a multiple of the number of rows, so that all coordinates are integers.

2. Method to draw a grid figure



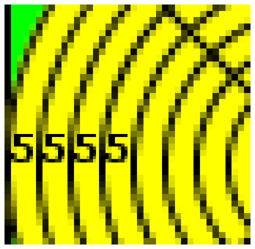
Once you have completed the static method that produces one subfigure, write another static method that produces a square grid of subfigures. You will call this method several different times from `main` to produce the grids of the overall figure. It will need a lot of parameters to be flexible enough 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. This method should work together with your first method to remove redundancy.

Don't forget to place the following statement at the top of your Java files:

```
import java.awt.*;
```

How to Get Started:

This program does not require many lines of code (our solution is 60 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 you to compute a value that we'll call **"the gap,"** or the distance between each neighboring pair of concentric circles in a subfigure. The 100x100 top-left subfigure has 10 circles, and each circle has a gap of 5 from any others (a total of 20 gaps in each direction). You can store the gap into a variable and use it in your subfigure drawing code. Each grid figure uses a different gap value for its subfigures, based on the parameters.

Write your code incrementally, repeatedly making small improvements. Start out by having your first method draw only the subfigure in the upper-left part of the screen. Then generalize it by **adding parameters one at a time**. For example, add parameters to change the subfigure's x/y position. Test the code by passing different values. Once one parameter works, move on to the next.

Stylistic Guidelines:

For this assignment you are limited to the language features in Chapters 1 through 3.

Continue to use static methods to structure your solution; this time, write methods that use parameters. In grading, we require at least the two methods named previously. You may use additional methods if necessary.

Give meaningful names to methods and variables, and properly indent your code. Follow Java's naming standards as specified in Chapter 1. Localize variables when possible; declare them in the smallest scope needed. Include meaningful comment headers at the top of your program and at the start of each method.