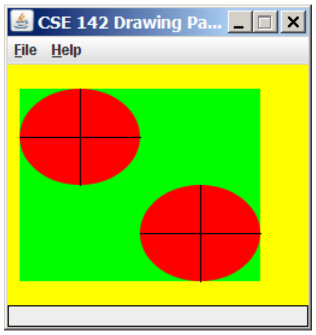
CSE 142, Winter 2007
Programming Assignment #4: Doodle and Checkers (20 points)
Due: Thursday, February 1, 9:00 PM

Program Description:

This assignment will give you practice with parameters, graphics, and `if/else` statements. This assignment has two parts; turn in two Java files named `Doodle.java` and `Checkers.java`. To compile and run this assignment, you must download the file `DrawingPanel.java` from the Assignments section of the class web page and save it in the same folder as your code. You should 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 not highly similar to your figure for Part B, and runs without user intervention (e.g. it should not read any user input from the `Scanner`). Be creative! After the assignment is turned in, we will anonymously post student figures on the course web site as part of a just-for-fun Doodle voting contest. See the course web site for details.

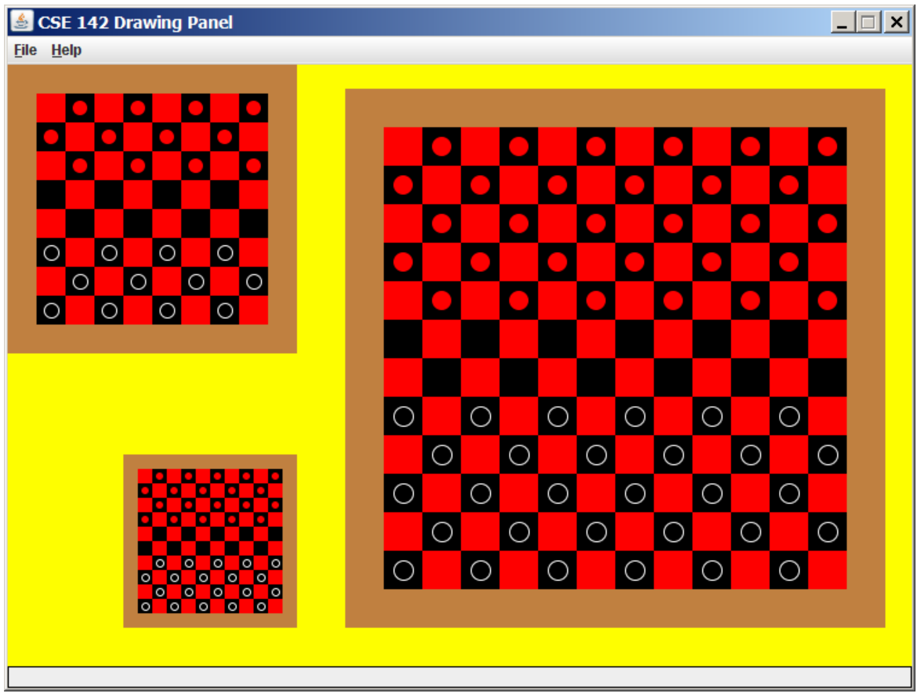


If you do not want to create a unique figure, at right is a default figure you may draw for Part A. If your Doodle program draws this figure, you will receive full credit for Part A. This figure contains a `DrawingPanel` of size 250 x 200 with a yellow background, a green rectangle with top-left corner at (10, 20) and size 200 x 160, red ovals with top-left corners at (10, 20) and (110, 100) of size 100 x 80, and black lines from (10, 60) to (110, 60), from (60, 20) to (60, 100), from (110, 140) to (210, 140), and from (160, 100) to (160, 180).

You may optionally wish to try using parameterized methods to capture the repetition in the figure. However, your score for Part A will be based solely on its external correctness as defined above; it will not be graded for internal correctness or style.

Part B: Checkers (16 points)

The second part of this assignment asks you to turn in a file `Checkers.java` that draws a specific complex figure that consists of several checkerboards at various locations and sizes, containing differing numbers of rows and columns. Your program should exactly reproduce the image shown here.



The images in this document were taken on Windows; if you use another platform, the window may look slightly different.

The program should produce graphical output only; no text output should be printed.

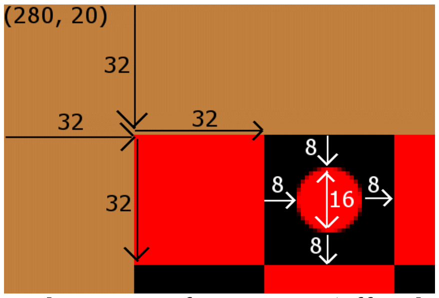
The `DrawingPanel` provides functionality to compare your image against the expected output file posted on the course web site. Detailed specifications of the appearance of the figure are on the next page.

Implementation Guidelines (Part B):

The overall drawing panel is size 750 x 500. Its background color is yellow. The panel contains several checkerboards of red and black squares. Each checkerboard is surrounded by a brown border (brown is represented as (red=192, green=128, blue=64). The border is exactly the size of one board square and spans around the entire board. The (x, y) positions given below are the top-left corners of each board's brown border.

The three checkerboards on your drawing panel should have the following properties.

Description	(x, y) position	size of each square	size of checkers	number of rows/cols
top-left	(0, 0)	24 x 24	12 x 12	8 x 8
bottom-left	(96, 324)	12 x 12	6 x 6	10 x 10
right	(280, 20)	32 x 32	16 x 16	12 x 12



The checkerboards have a varying number of rows and columns. All rows except the middle 2 of each checkerboard are occupied by checkers on their black squares, represented as red filled ovals for red checkers on top of the board, and white drawn ovals for black checkers on the bottom of the board. So in a sense, the board can be thought of as having three "regions": a top group of rows that contain red checkers, a middle group of exactly two rows with no checkers, and a bottom group of rows that contain black checkers.

Each checker has a diameter exactly half as large as the square it occupies and lies in the center of its square (offset by one quarter of the square's size in each direction). The diagram at left shows an enlarged view of the top-left corner of the large right checkerboard, with various pixel distances labeled.

Stylistic Guidelines (Part B):

An important part of this program is using `if/else` statements as necessary to draw your checkerboards. A solution receiving full credit will be able to draw a complete "region" of a checkerboard as described above (or even an entire checkerboard) in a single pass using nested `for` loops that contain various `if` and `if/else` statements.

To receive full credit on Part B, you should write static methods that use a great deal of parameter-passing and complex numeric computations. You are required to have **two useful static methods besides `main`** as described below.

You are required to have a static method that draws one checkerboard each time it is called. You will call this method three times from `main` to produce the overall figure. It will need several parameters to be flexible enough to draw each of these boards. The key point is that a single method can be called three times to produce the three grids. The method should be general enough to work for any checkerboard with an even number of rows and columns.

If all of the checkerboard-drawing code were placed into the preceding method, it would be quite large. Therefore you should decide on a second useful method to add to the program that offloads a nontrivial portion of the drawing logic.

For this assignment you are limited to the language features in Chapters 1 through 4; you are not allowed to use more advanced Java features to solve the problem. For reference, our solution is 61 lines long.

You should properly indent your code and use whitespace to make your program readable. Give meaningful names to methods and variables in your code, and follow Java's naming standards as specified in Section 1.2 of the textbook. Localize variables whenever possible; that is, declare them in the smallest scope in which they are needed. Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method explaining its behavior.

How to Get Started (Part B):

Because it is difficult to correctly implement such a complex program all at once, instead start with a smaller piece of the problem. For example, you can start with drawing just the checkerboard in the upper-left part of the screen, without any checkers on it. Write the code incrementally and make gradual improvements, eventually creating a parameterized method that allows different sizes, (x, y) locations, and so on.