CSE142—Computer Programming I
Programming Assignment #3
due: Tuesday, 7/14/15, 9 pm

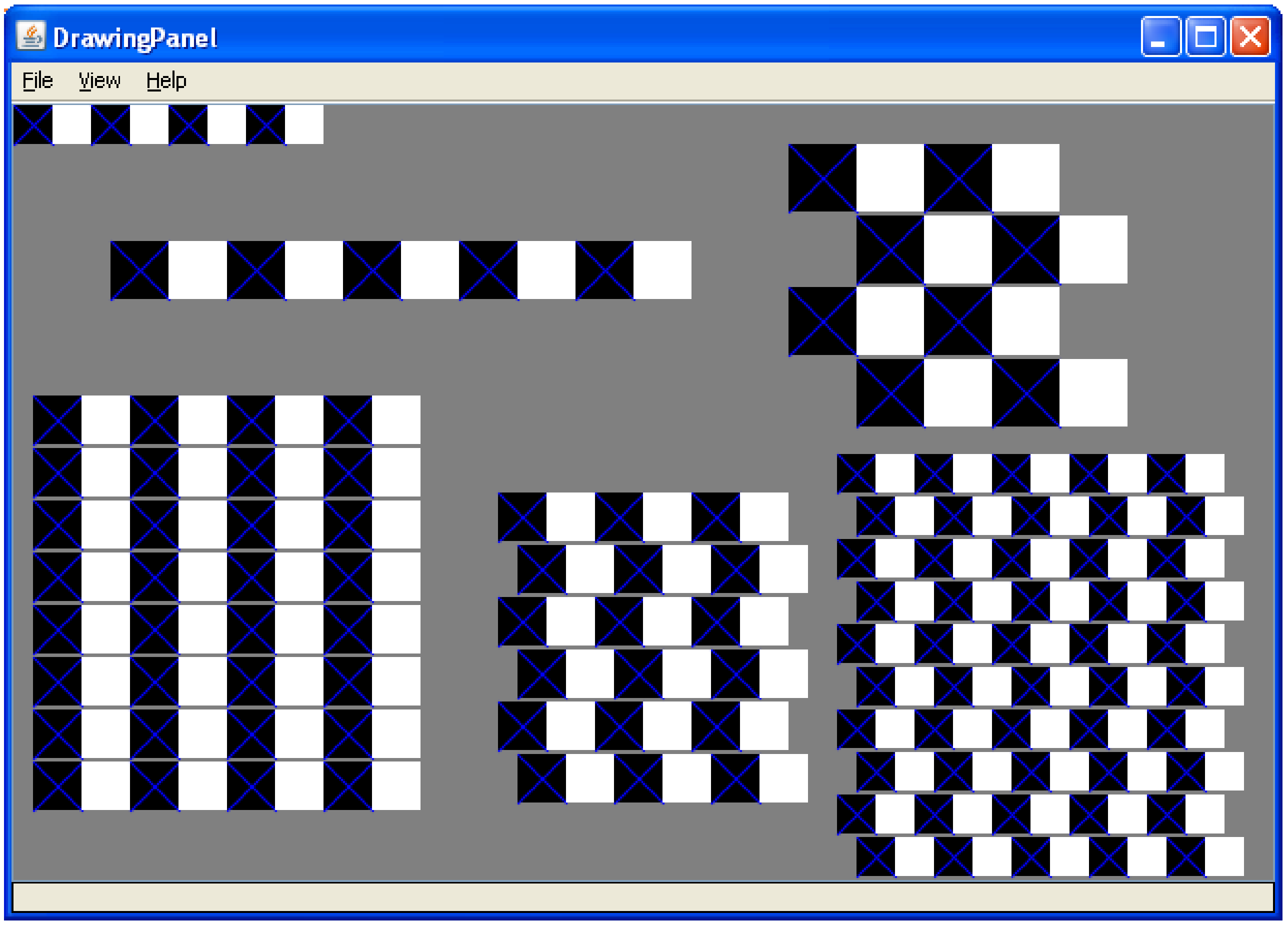
This assignment will give you practice with for loops, value parameters, and graphics. You will be using a special class called DrawingPanel written by Stuart Reges and his coauthor Marty Stepp.

Part A: Doodle.java (2 points)

For the first part of the assignment, turn in a file Doodle.java that draws a figure using the DrawingPanel 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.

Part B: CafeWall.java (18 points)

In the second part of the assignment we will be exploring something that is known as the Café Wall illusion (as described in http://en.wikipedia.org/wiki/Caf%C3%A9_wall_illusion). You are to produce the image below. It has a size of 650 pixels by 400 pixels and has a gray background.



This image has several levels of structure. Black and white squares are used to form rows and rows are combined to form grids. The output has two free-standing rows and four grids. You should first write a method to produce a single row. Each row is composed of a certain number of black/white pairs of boxes where each black box has a blue X in it.

The free standing rows in the output have the following properties:

Description	(x, y) position	Number of pairs	Size of each box
upper-left	(0, 0)	4	20
mid-left	(50, 70)	5	30

Notice that we specify each row in terms of how many pairs of black/white boxes it has. Your method doesn't have to be able to produce a row that has a different number of black versus white boxes. The boxes are specified using a single size parameter because each should be a square. You should develop a single method that can produce any of these rows by varying the position, the number of black/white pairs, and the box size. You will need to use one or more for loops to write this code so that it can produce any of the various rows.

Once you have completed this method, write a method that produces a grid of these rows by calling your row method appropriately (you will again use one or more for loops to solve this task). Grids are composed of a series of pairs of rows where the second row is offset a certain distance in the x direction relative to the first. The output has four grids with the following properties:

Description	(x, y) position	Number of pairs	Size of each box	2nd row offset
lower left	(10, 150)	4	25	0
lower middle	(250, 200)	3	25	10
lower right	(425, 180)	5	20	10
upper right	(400, 20)	2	35	35

Each grid is a square, which is why a single value (number of pairs) indicates the number of rows and columns. For example, as the table above indicates, the lower-left grid is composed of 4 pairs. That means each row has four pairs of black/white boxes (8 boxes in all) and the grid itself is composed of 4 pairs of rows (8 rows total). A single box size is again used because each box should be a square. The offset indicates how far the second row should be shifted to the right in each pair. The figure in thelower-left has no offset at all. The grid in the upper-right is offset by the size of one of the boxes, which is why it has a checkerboard appearance. The other two grids have an offset that is in between, which is why they produce the optical illusion (the rows appear not to be straight even though they are).

Each pair of lines in the grid should be separated by a certain distance, revealing the gray background underneath. This is referred to as the "mortar" that would appear between layers of brick if you were to build a wall with this pattern. The mortar is essential to the illusion. Your program should use 2 pixels of separation for the mortar, but you should introduce a program constant that would make it easy to change this value to something else.

You are required to have the two methods described above (one for a single row, one for a grid). We will expect you to use good style (indentation, commenting, good variable names, etc) and to include a comment for the class and for each method.

You should download DrawingPanel.java from the class web page and save it in the same folder as your programs. Include the following line of code at the beginning of your class file:

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

You can use the DrawingPanel's image comparison feature (File, Compare to Web File...) to check your output. Sample outputs are available for different mortar settings. Two versions of each output are provided because there are two slightly different approaches that are each correct. You must match one of each pair within 500 pixels for your output to be considered correct.

For this assignment you are limited to the language features in Chapters 1 through 3G of the textbook. In particular, you should not use if/else statements. Turn in your two program files using electronic turnin. You do not have to turn in DrawingPanel.java.

There is an additional option for debugging your program: You can request a special version of the Graphics object that keeps a count of how many times basic drawing commands are performed. The method getDebuggingGraphics should be used instead of getGraphics to obtain this version:

```
    Graphics g = panel.getDebuggingGraphics();
```

You can print the counts for the drawing methods by calling the getCounts method of the DrawingPanel class as the last line of main:

```
    System.out.println(panel.getCounts());
```

This will print something like:

```
  {drawLine=31, drawOval=4, drawRect=15, fillOval=9, fillRect=26}
```

There are different ways to correctly solve this problem, so your counts do not have to match if you manage to get 0 pixels of difference when your graphical output is compared to the expected output. But if you have a pixel difference, then the counts can provide you a sense of what you might be doing incorrectly (e.g., performing a drawing command too many times or too few times). Our sample solution used lines methods 234 times and used rectangles methods 234 times.