# CSE 142, Spring 2019
# Programming Assignment #3: Café Wall (20 points)
### Due Tuesday, April 23, 2019, 9:30 PM

This assignment will give you practice with for loops, value parameters, and graphics. You will be using a special class called `DrawingPanel` written the authors of our textbook. 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`. Turn in two Java files (`Doodle.java` and `CafeWall.java`) as described below.

## Part A: Doodle.java (2 points):

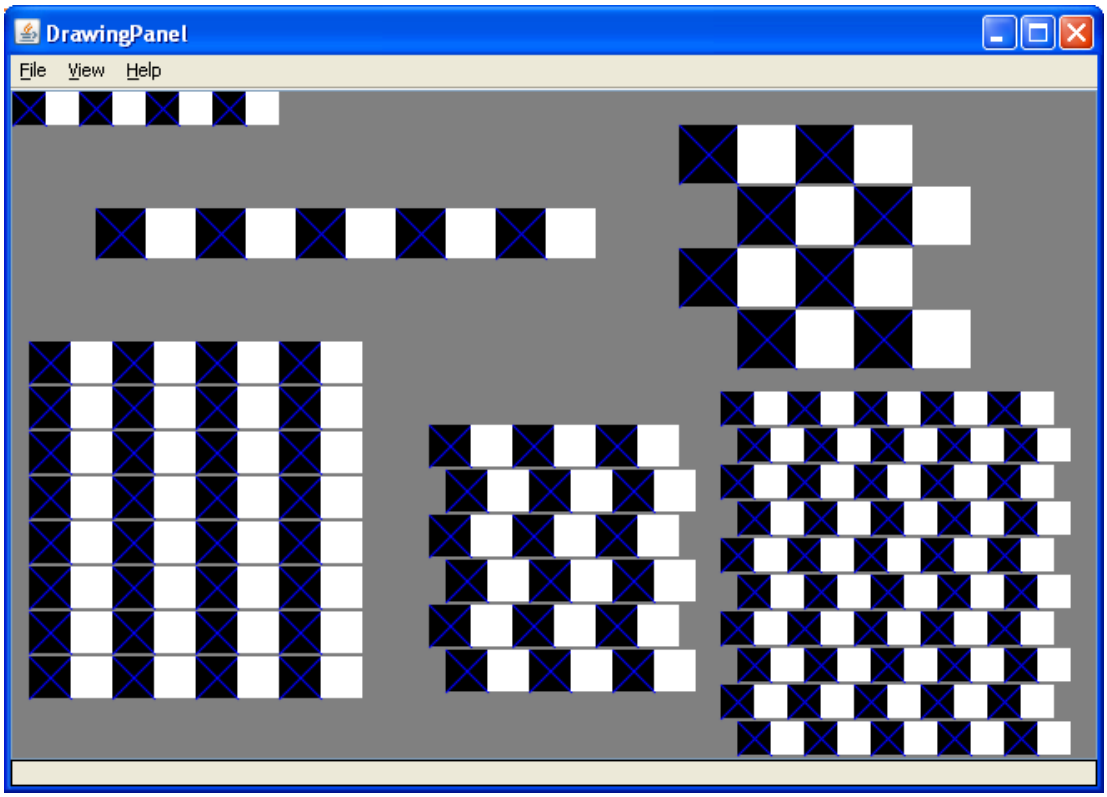
For the first part of the assignment, turn in a file `Doodle.java` that creates an image using the `DrawingPanel` class. You may draw any image you like with the following restrictions:

- The image must be at least 100 x 100 pixels.
- The image must contain at least three distinct shapes and use at least two distinct colors.
- The images must not include hateful, offensive, or otherwise inappropriate images.
- The image should be entirely your own work, and must not be substantially similar to your solution to Part B, consist entirely of reused Part B code, or be substantially similar to a related CSE142 assignment from a previous quarter.
- The code must not have any infinite loops and must not read any user input.

Your score for Part A will be based solely on the criteria defined above; 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 will turn in a file `CafeWall.java`. You are to produce the image below. The `DrawingPanel` 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. **You should then write another method that uses your row method to produce a single grid.**

## Development Strategy (How to get started):

### 1. Drawing a row:

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

| Description | (x, y) position | Number of box 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.

### 2. Drawing a grid:

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 row 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 the lower 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 must introduce a class constant that would make it easy to change this value to something else**.

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. 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. If there are no visible differences to the naked eye, your output will most likely be considered correct.

## Style Guidelines:

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.

**You are required to have the two methods described above (one for a single row, one for a grid).** You may use additional methods if you like. **You are expected to use parameters effectively, and none of your methods should take unnecessary parameters**. For this assignment, a parameter is considered unnecessary if its value is redundant with another parameter, if its value could be computed using other parameters' values, or if it is never used in the method.

We expect you to use good style throughout your code. Give meaningful names to your methods, variables, and parameters. Properly indent your code. Follow Java's naming conventions as specified in Chapter 1. Limit the lengths of your lines to fewer than 100 characters. Include meaningful header comments at the top of your program and at the start of each method.