CSE 142: Computer Programming I — Winter 2020

Assignment 3: Café Wall (20 points) *due January 28, 2020, 11:59pm*

This assignment focuses on `for` loops, parameters, and graphics. Turn in the following **TWO** Java files using the link on the course website:

- `Doodle.java` – A program that produces your custom image
- `CafeWall.java` – A program that produces the Café Wall illusion described below

Program Behavior

Part A: Doodle (2 points)

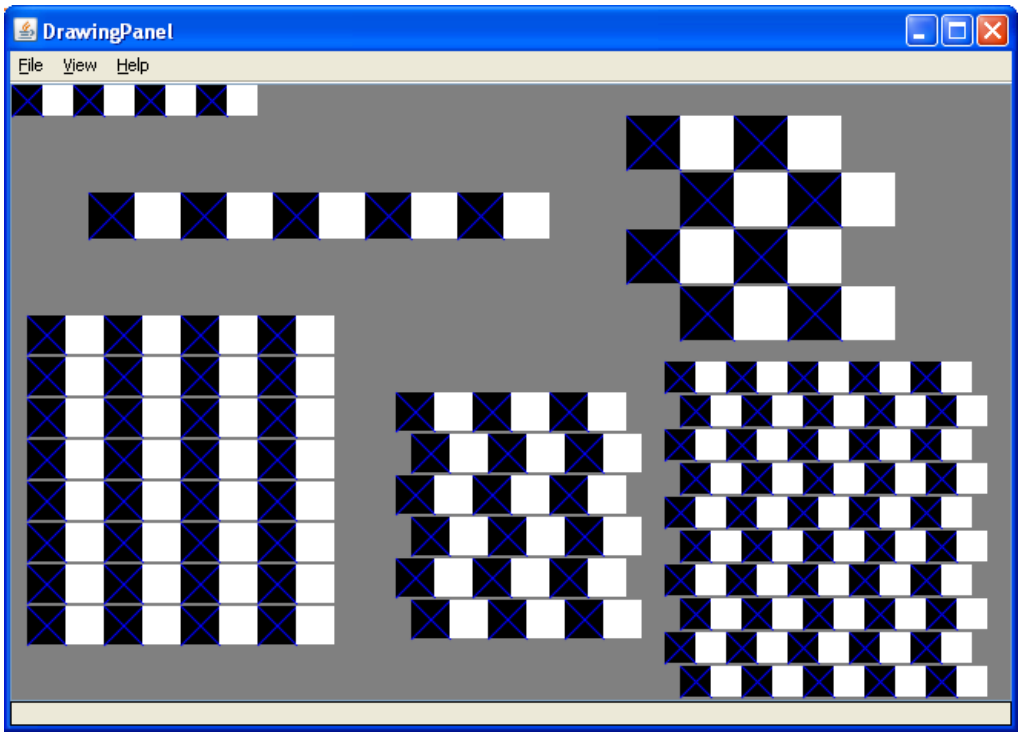
The first part of your assignment is to write a program that uses `DrawingPanel` to produce an image of your choice. Your program can produce any image you like, with the following restrictions:

- The image should not include hateful, offensive, or otherwise inappropriate images.
- The image must be at least 100×100 pixels.
- The image must contain at least three distinct shapes and at least two distinct colors. (Note that shapes need not be unique; for example, three separate rectangles count as three distinct shapes.)
- The image must not be substantially similar to your solution for Part B, consist entirely of reused Part B code, or be substantially similar to a related CSE 142 assignments from a previous quarter.
- The code must successfully compile and run, and must not enter an infinite loop.
- The code must not use material beyond Supplement 3G of the textbook.

If your program compiles, runs, and meets all of the above requirements, you will receive the full 2 points.

Part B: Café Wall (18 points)

The second part of your assignment is to produce an image that demonstrates what is known as the Café Wall illusion. Your program should produce the following image:



This image is drawn on a 650×400 pixel `DrawingPanel` with a `Color.GRAY` background. The image consists of four grids made up of rows, along with two stand-alone rows. Each row consists of pairs of black and white boxes, with a blue 'X' drawn over each black box. Each grid consists of pairs of rows, with the second row in each pair offset to the right by a certain number of pixels (potentially zero). Each grid is also a square; that is, the number of row pairs in the grid is equal to the number of box pairs in each row of that grid.

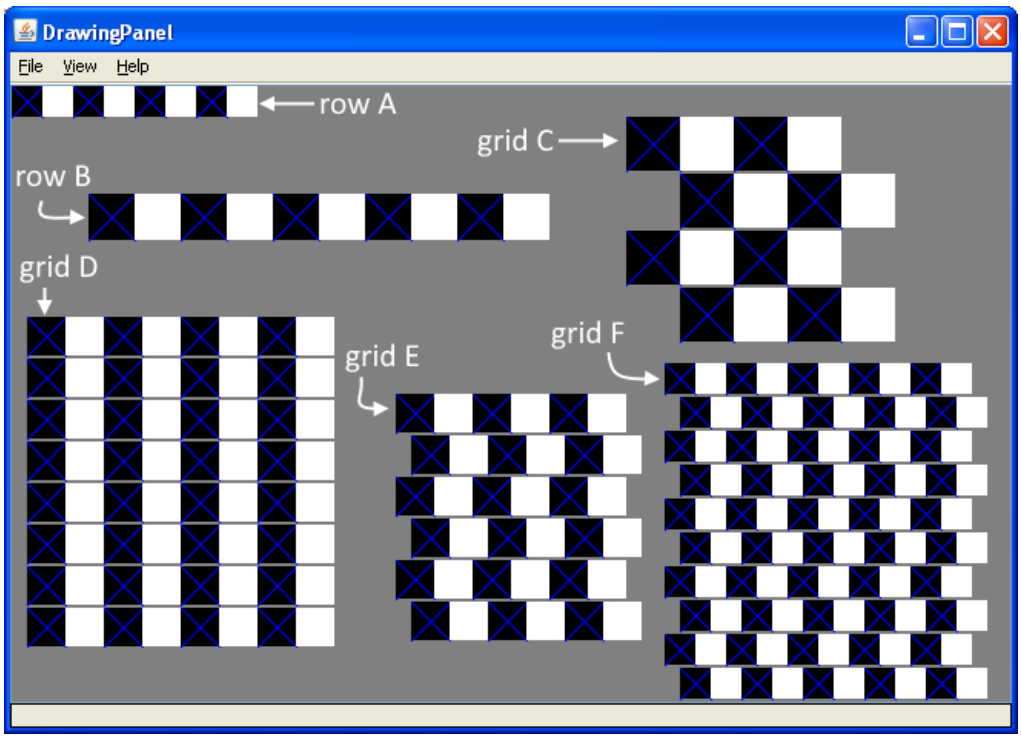
The overall image consists of six components (two stand-alone rows and four grids). Thse components are labeled in the image below. The properties of each component are as follows:

Rows

Label	Position	Box Pairs	Box Size
A (upper left)	$(0, 0)$	4	20
B (middle left)	$(50, 70)$	5	30

Grids

Label	Position	Row Pairs	Box Size	Offset
C (upper right)	$(400, 20)$	2	35	35
D (lower left)	$(10, 150)$	4	25	0
E (lower middle)	$(250, 200)$	3	25	10
F (lower right)	$(425, 180)$	5	20	10



Since each grid is a square, the number of row pairs is the same as the number of box pairs.

The second row in each pair within a grid is offset to the right by a specified amount. For example, in grid D (the lower left), the offset is zero, so the rows are perfectly aligned. In grid C (the upper right), the offset is the same as the size of each box, giving a "checkerboard"-like appearance.

In each grid, the rows are separated vertically by a small amount, allowing the gray background to show. (This separation is what triggers the illusion.) We will refer to this separation as "mortar." By default,

your image should use 2 pixels of mortar. However, we should be able to change the size of the mortar by changing a single value and recompiling your program. See below for more details.

The mortar is a space between the rows, not a separate component you need to draw.

There will not be an Output Comparison Tool for this assignment. Instead, you can use the "Compare to Web File" feature of `DrawingPanel` (found in the File menu) to check your output. This feature will show you the number of pixels that are different between your output and the expected output, and will allow you to highlight the specific pixels that differ. Different operating systems draw shapes in slightly different ways, so it is possible to have some pixels different between your output and the expected output even if your code is correct. However, there is no specific minimum or maximum number of pixels of difference that will be considered acceptable. If there are no visible differences to the naked eye, your output will most likely be considered correct.

Look carefully for differences, and utilize the "Highlight diffs" feature to make sure there are no major differences.

Development Strategy

To complete this assignment, you will need to download the file `DrawingPanel.java` from the course website. Save this file in the same folder as your programs. You will also need to include the following line of code at the beginning of each of your programs (before your `public class` declaration):

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

As on Assignment 2, we recommend you approach the program in stages, as follows:

(1) **Single row:** Write a method to produce a single row of black and white boxes.

(2) **Parameterized rows:** Modify your row method to be able to produce a row with any number of box pairs of any size.

(3) **Single grid:** Once your row method completely works, write a method that produces a single grid by calling your row method.

(4) **Parameterized grids:** Modify your grid method to be able to produce a grid with any number of row pairs of any size boxes and with any offset.

(5) **Mortar:** Once you are able to produce any grid, add your constant (see below) so the size of the mortar can be changed.

You do not need to worry about producing rows with an odd number of boxes, grids with an odd number of rows, or rectangular grids.

Implementation Guidelines

Like all CSE 142 assignments, Part B of this assignment will be graded both on "external correctness" (whether the program compiles and produces exactly the expected output) and "internal correctness" (whether your source code follows the implementation and style guidelines in this document). You will need to adhere to the following guidelines to receive full credit on part B. (Part A of this assignment is graded only on the requirements stated above; it is not graded on internal correctness.)

Required methods/structure

For this assignment, your program is required to utilize the following structure:

(1) Your program must include a method to draw a **single row** at a time. This method should use parameters so that it can produce any of the required rows in the final image.

(2) Your program must also include a single method to draw a **grid**, which must be acheived by calling the row method described in (1).

Your program must have *at least* these two methods, though you may have more. Your grid method must call your row method.

Both your row method and your grid method should make use of `for` loops to produce each grid/row from its component rows/boxes.

Changing mortar with a class constant

As described above, your program should be able to be easily changed to alter the mortar size. To achieve this, your program should include one (and only one) class constant that represents the size of the mortar. (The default mortar size is 2.) The size of the mortar should be able to be altered by changing **only the value of the constant**.

The course web site will contain files that show you the expected output if your mortar constant is changed to 1 instead of 2. You can use the "Compare to Web File" feature of `DrawingPanel` to check your output.

Permitted Java Features

For this assignment, you are restricted to Java concepts covered in chapters 1 through 3 and supplement 3G of the textbook. In particular, you MUST use `for` loops and parameters, and you **may not** use `if` or `if-else` statements.

Style Guidelines

You should follow all guidelines in the Style Guide and on the General Style Deductions page of the course website. Pay particular attention to the following elements:

Capturing Structure

Your program *must* use the method structure described above, though you may include additional methods if you like. As always, your `main` method should be a concise summary of the program.

Using Parameters

Your program should utilize parameters to define generalized methods that can be used to create various similar results. In addition, your methods should not accept any unncessary parameters. For this assignment, a parameter is considered unncessary if its value is unused, is always the same as the value as another parameter, or can be directly computed from the values of other parameters.

Code Aesthetics

Your code should be properly indented, make good use of blank lines and other whitespace, and include no lines longer than 100 characters. Your class, methods, variables, and constant should all have meaningful and descriptive names and follow the standard Java naming conventions. (e.g. `ClassName`, `methodOrVariableName`, `CONSTANT_NAME`) See the Style Guide for more information.

Commenting

Your code should include a header comment at the start of your program, following the same format described in Assignments 1 and 2. Your code should also include a comment at the beginning of each method that describes that method's behavior. Comments should be written in your own words (i.e. not copied and pasted from this spec) and should not include implementation details (such as describing loops or expressions included in the code). See the Style Guide or lecture code for examples.

Getting Help

If you find you are struggling with this assignment, make use of all the course resources that are available to you, such as:

- Reviewing relevant lecture examples
- Reviewing this week's section handouts
- Reading the textbook

- Visiting the IPL
- Posting a question on the message board

Academic Integrity

Remember that, while you are encouraged to use all resources at your disposal, including your classmates, **all work you submit must be entirely your own**. In particular, you should **NEVER** look at a solution to this assignment from another source (a classmate, a former student, an online repository, etc.). Please review the full policy in the syllabus for more details, and ask the course staff if you are unclear on whether or not a resource is OK to use.