Take-home Assessment 3: Café Wall

due April 20, 2021, 11:59pm

This assignment will assess your mastery of the following objectives:

- Write a functionally correct Java program to produce specified graphical output.
- Write `for` loops, including nested loops, to repeat code and manage control flow.
- Write and call methods that accept parameters to perform generalized tasks.
- Use class constants to represent and modify important values in a program.
- Follow prescribed conventions for spacing, indentation, naming methods, and header comments.

Program Behavior

Part A: Doodle

The first part of your assessment 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:

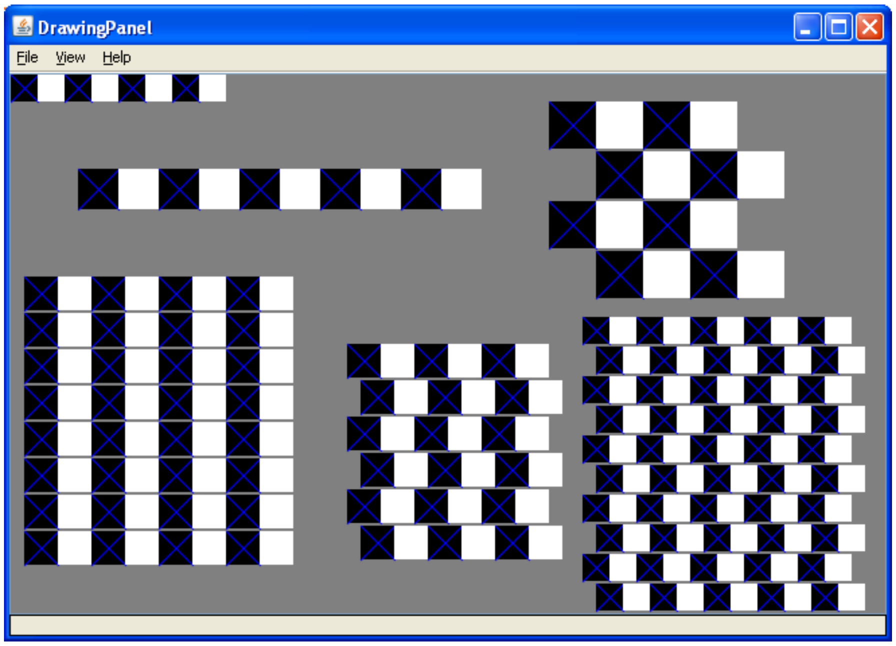
- The image should not include hateful, offensive, or otherwise inappropriate images.
- The image must be at least 100×100 pixels and must contain at least three distinct shapes and at least two distinct colors.
- The image must not be substantially similar to your solution for Part B, consist entirely of reused Part B code, or be substantially similar to a related CSE 142 assignments from a previous quarter or any examples from class.
- 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.

Shapes need not be unique. For example, three separate rectangles count as three distinct shapes.

This part of the assessment will only contribute to the Behavior dimension grade. It will not factor in to grading on the other dimensions.

Part B: Café Wall

The second part of your assessment 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. We will ask you to draw your box pairs in a specific order: Alternate between black boxes first, then blue Xs, then white boxes. In order to pass our tests 100%, you need to make sure that every shape in the rows is being drawn in the right order. 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). These components are labeled in the image below. The properties of each component are as follows:

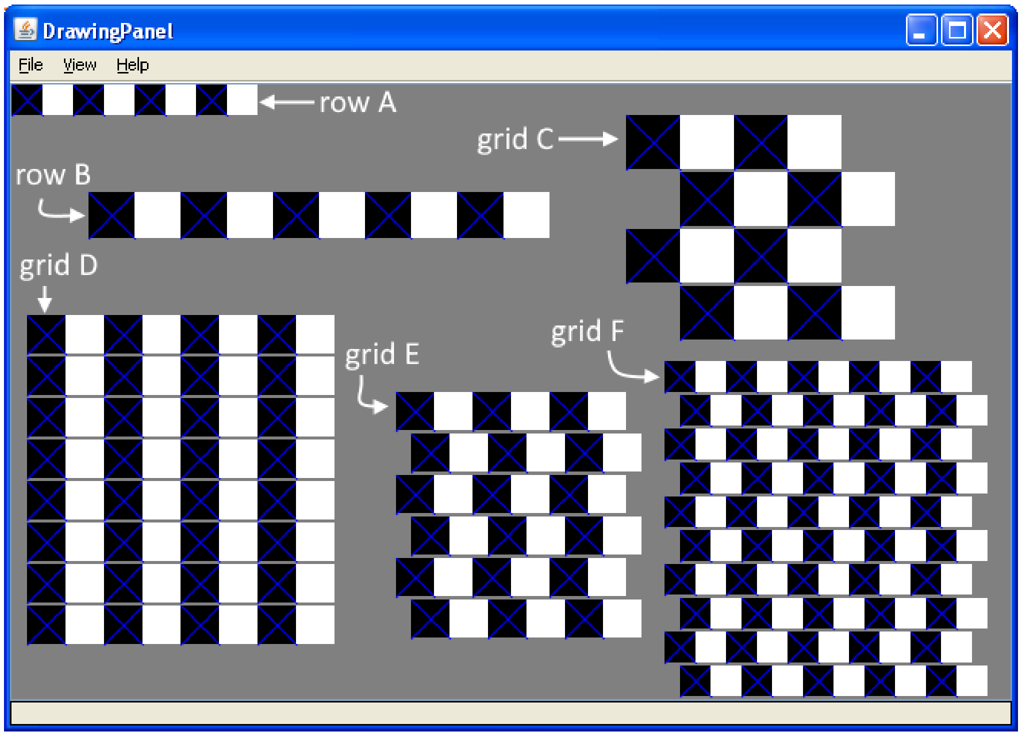
Rows

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

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

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



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.

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

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.

You can use the Check button in Ed to verify the correctness of your output for various values of the size constant. This feature will show you the number of pixels that are different between your output and the expected output, and will highlight the specific pixels that differ. If you are running your code in Ed, you should be able to get a perfect match.

If you have only a few pixels difference around the edges of figures, check the order in which you are drawing the parts of the image.

Development Strategy

As on Take-Home Assessment 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 at a specific position.

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

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

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

(5) **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.

(6) **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 assessments, Part B of this assessment will be graded on all four dimensions defined in the syllabus. Be sure to adhere to the following guidelines:

Required methods/structure

For this assessment, 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).

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

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

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**. *To ensure our testing and grading scripts work correctly, you* ***must*** *name your constant* `MORTAR`*.*

Make sure you declare your constant exactly right, using the keywords `public static final`.

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 Check button in Ed to test each value for the constant. Check will test based on which value you have in your code, whereas Mark will test both values.

Permitted Java Features

For this assessment, 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.

Code Quality Guidelines

In addition to producing the desired behavior, your code should be well-written and meet all expectations described in the grading guidelines and the Code Quality Guide. For this assessment, 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 and you should not have any trivial methods.

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 unnecessary parameters. For this assessment, 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 Code Quality Guide for more information.

Commenting

Your code should include a header comment at the start of your program, following the same format described in previous assessments. Your code should also include a comment at the beginning of each method that describes that methods behavior. Method comments should also explicitly name and describe all parameters to that method. 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 Code Quality Guide for examples and more information.

Running and Submitting

You can run your Café Wall program by clicking the "Run" button on the Café Wall slide in Ed. This will compile and execute your code and show you any errors, or the output of your program if it runs correctly. Clicking the "Check" button will run your Café Wall program and show you differences from the expected output. You can run your Doodle program by clicking "Run" button on the Doodle slide in Ed. If you believe your output is correct, you can submit your work by clicking the "Mark" button in the Ed assessment (you must do so for both the Café Wall and Doodle slides). You will see the results of some automated tests along with tentative grades. **These grades are not final until you have received feedback from your TA.**

You may submit your work as often as you like until the deadline; we will always grade your most recent submission. Note the due date and time carefully—**work submitted after the due time will not be accepted**.

Getting Help

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

- Reviewing relevant examples from lessons, section, and lab
- Reading the textbook
- Visiting support hours
- Posting a question on the message board

Collaboration Policy

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 assessment 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.

Reflection

In addition to your code, you must submit answers to short reflection questions. These questions will help you think about what you learned, what you struggled with, and how you can improve next time. The questions are given Café Wall Reflection slide in the Ed lesson; type your responses directly into those answer boxes.