Assignment 08

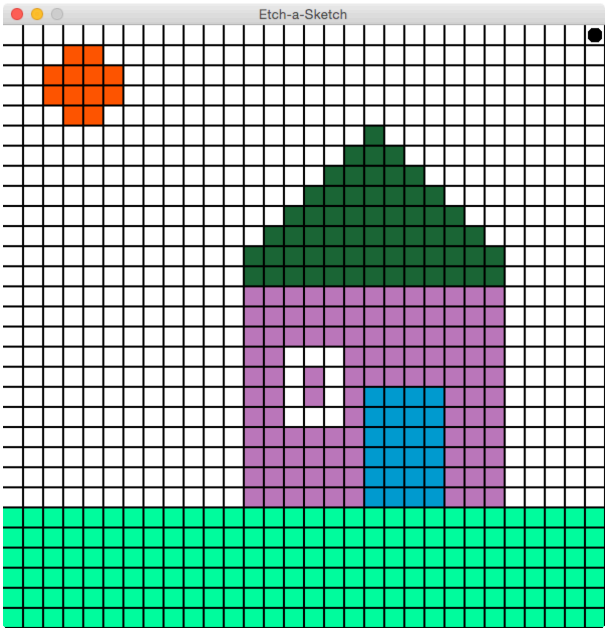
CS 120: Software Design I

Program 1: Etch-A-Sketch 2.0 – `Driver`

Write a program that allows you to recolor tiles in a 30x30 grid. You will use the keyboard to navigate a marker around the screen, change the color of the marker, lift the marker up, set the marker down, and clear the tiles. There is no other interaction with the user except through the keyboard events in the window. Meaning that there should be no `Scanner` or `System.out` calls in the final version of your program.

You will be writing this program in the class named `Driver` from the project template. All of your code will go in the `Driver` class (please examine the file to see where code should be added), and you must **not modify** any of the following supporting classes:

- `InstructionsWindow` : A Popup window with instructions.
- `KeyboardListener` : Listens for keyboard key presses and tells the `Driver` about them.
- `Line` : A line object.
- `Oval` : An oval object.
- `Rectangle` : A rectangle object.
- `Window` : The window object that we draw in.



Step 1: Laying out the grid

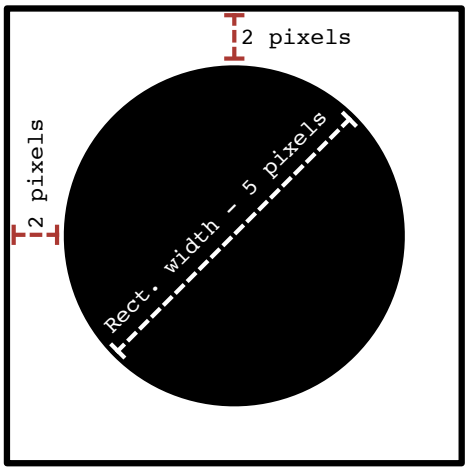
First, start by laying out a 30x30 grid of white rectangles to fill the window. You must keep these `Rectangle` objects in a two-dimensional array so you can easily change their colors later. You will need to declare the array in the class scope, but instantiate it in the `Driver` constructor. This way you can access it from the `handleKeyboardEvent` method (and other methods) later in the program.

Once you have `Rectangle` objects in place then place black lines between the rectangles to cover up the pink window color showing up between them. The lines must be black and have a thickness of `2` to properly cover the pink background. Since we do not need to modify these lines after placement you can declare and instantiate them in the `Driver` constructor.

Step 2: Displaying the marker

Next, create an `Oval` object to represent the marker. As with the `Rectangle` array, you will want to declare this object in the class scope, but instantiate it in the `Driver` constructor. The marker is initially black and positioned at the upper-left hand corner (0,0) in the grid.

The marker will be slightly smaller than the `Rectangle` objects. The `Oval` object will be 5 pixels smaller in both width and height. The `Oval` object will be positioned (roughly) centered on top of the `Rectangle` object, shifted by 2 pixels to the right and 2 pixels down from the upper left hand corner of the rectangle (this will give the appearance of centering the marker in the rectangle). See the diagram below:



Step 3: Moving the marker around the grid

Next, you will be moving the marker around the grid in response to the arrow keys being pressed by the user. The `handleKeyboardEvent` method is called whenever the user pressed a keyboard key. The template will print out a debugging message (which must be removed before the final submission) whenever one of the keys we care about are pressed.

In this step you will be changing the location of the marker in response to the up/down/left/right arrow keys. You will need to keep track of which grid cell (by it's row and column) that the marker is currently occupying, and move that cell reference by one row or column, depending upon the arrow key pressed. **Be careful** to not walk past the edge of the array - otherwise you will get an `ArrayIndexOutOfBoundsException`. Your code must prevent those exceptions from occurring by checking the direction of movement and making sure it is a valid move.

Once you have the new, valid location in the grid then you need to move the marker (using that object's `setLocation` method) to sit on top of the `Rectangle` object at that grid location. **Hint:** You may want to ask the destination `Rectangle` object what its X and Y coordinates are to help you position the marker correctly.

Once you have completed this step then you should be able to move the marker around the grid using the arrow keys and the marker should stop (and not issue any error messages) when it encounters an edge.

Step 4: Coloring the grid cells black

Next, as you move the marker in the grid you will need to change the color of the grid cell that it is leaving to the color of the marker. Initially, the marker is black so it will leave a black line behind it as it moves.

When the user presses the spacebar it will toggle the pen up and down. Initially the pen is down, so it will be drawing as it moves. If the user presses the spacebar then the pen will be up, so as it moves it will **not** recolor the cells it passes. The user can continue to press the space bar to toggle the pen up and down. You will likely need a separate variable to keep track of whether or not the pen is down or up.

Step 5: Custom Colors

Now you are ready to add a splash of color to your palette. The user may choose a color in a variety of ways using the keyboard keys.

- **R** : Increase the "Red" value by 5 (initially 0, max 255)
- **G** : Increase the "Green" value by 5 (initially 0, max 255)
- **B** : Increase the "Blue" value by 5 (initially 0, max 255)
- **K** : Reset the color to "Black" ("Red" = 0, "Blue" = 0, "Green" = 0)
- **X** : Pick a random color (see the `getRandomColor` method)

As the user changes the color the color of the marker `Oval` will change to the current color blend. As the marker moves the new color is left behind.

Note that the **R**, **G**, and **B** values must be `int` variables and stay in the range of 0-255, inclusive. If you try to pass a value greater than 255 into the `Color` constructor then an error will be issued. Your program must prevent such an error from being issued by checking the value and making sure that it never exceeds 255. If the user tries to exceed 255 then your program should leave it at 255.

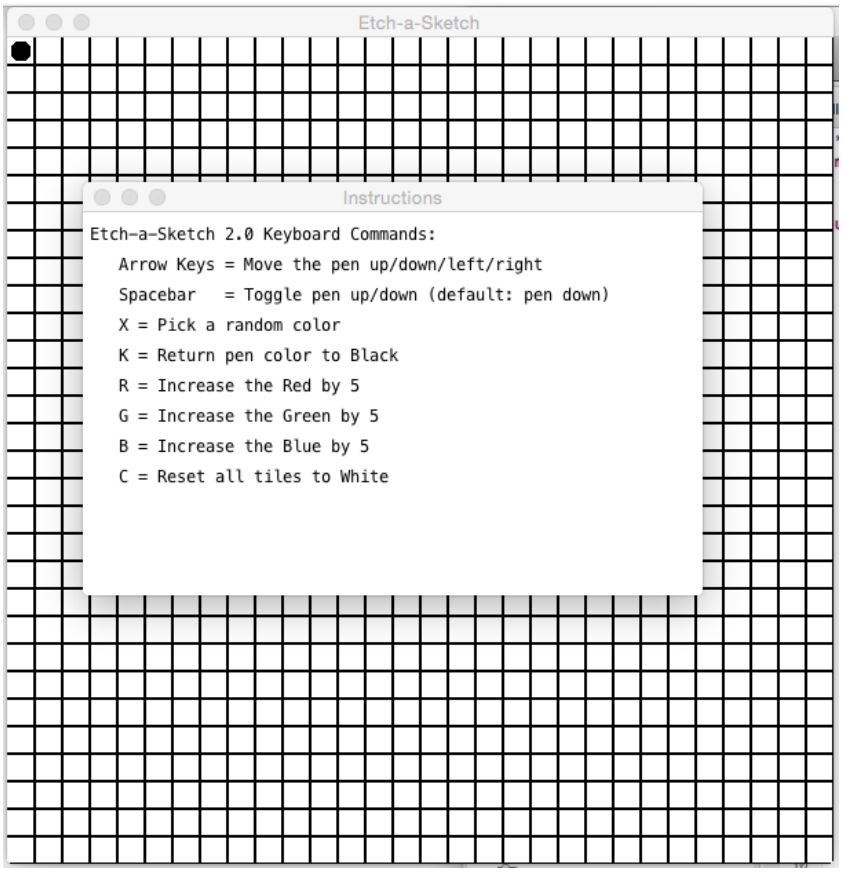
Step 6: Clearing the cells

Finally, when the user presses the **C** key then you will need to reset the color of all of the `Rectangle` objects to white. Note that the color and position of the marker remains the same, as does the current state of the pen (i.e., up or down).

Step 7: Testing

When you reach this step then you are ready to test and test some more. Make sure that there is absolutely no output in the console, and no errors are ever generated (e.g., from trying to exceed the bounds of the window, or trying to get a color greater than 255).

When the program starts up it should look like the image below.



Example Output
A runnable `jar` file has been provided that will allow you to run a solution for this assignment. This should allow you to better understand the expected behavior of your program.

Notes

Assignment Clarifications

- Make sure that you only add code to the `Driver` class, and have not modified any of the supporting classes (your changes will not be preserved in grading).
- You may add additional `private` methods to the `Driver` class.
- You will need to use class scoped variables in this assignment. You should only use them when absolutely necessary to carry data between methods in the `Driver` class.