Lab I: Fourth Arduino Lab! Drawing on the OLED and multi-dimensional arrays.

Due: March 24, 9 p.m.

1 Introduction

You've now seen some Java Abstact Data Types (ADTs) to manage collections of objects, and you also have some experience with Java's standard arrays. An array in Java is a collection of values of the same type. For example, when you type:

var x = new int[]1, 2, 3;

you create an array of three integers.

Technically speaking, an array is a **contiguous block of memory**. That means all the values in the array are stored in registers in your computer that are adjacent to one another. Arrays can either be created with fixed values like the previous example, or an array can be created with a size and then filled later. You can create an array of integers with space for three integers as follows:

var x = new int[3];

To then assign a value (42 for example) to the element of the array (a for example) at some index i, you would write:

a[i] = 42;

Arrays can be multidimensional. For example, the statement:

var x = new int[2][3];

creates a two-dimensional array of three integers (aka a 2x3 matrix).

Note that unlike some modern OOP languages such as Kotlin or Swift, arrays are not objects in Java. An ArrayList, by contrast, is an object. But a regular array is just a collection of values.

2 Programming Task

In this lab we will be mapping locations on the OLED display to locations in a two dimensional array on your Arduino board. By doing so we will be able to draw some nice looking shapes on our OLED, like ovals and rectangles.

To begin, you will be altering the file called *ShapeRenderer.java* and completing a constructor, methods to generate shapes and methods to render them on your OLED display.

More specifically, in the *ShapeRenderer* class, you will implement:

1. The constructor for the ShapeRenderer class, i.e. *ShapeRenderer(OLED oled, int width, int height)*. Both width and height must greater that 1; if they are not, you should throw an $InvalidDimensionException$. In addition, if either width or height is greater than 1 and even, subtract one from its value in order to make it odd. Given valid inputs, you can initialize the oledDisplay attribute of the ShapeRenderer class such that it references the input oled. Also initialize the screen attribute such that it holds a two dimensional array of boolean values with two odd dimensions (width, height). Initialize each element in the boolean array to the value false.

2. The method *toString()*. This method will override the toString method in the Object class and should print the contents of the screen attribute to the console. If a value in the screen attribute is true, print a "1" and if it's false print a "0". Place a newline after each row in the screen attribute.

3. The method *createOval(int a, int b)*. This method should accept two EVEN integers as input, a and b. It will then set every element in the screen attribute to TRUE if and only if the element is within a an oval centered in the middle of the array (i.e. at the coordinates floor(width/2) + 1 and floor(height/2) + 1), with a width of (2a + 1) and a height of (2b + 1). Every other element in the screen attribute should be set to false. Remember that the equation of an oval that is centered at (x1,y1), with a width of 2w and a height of 2h is defined by the following equation:

$$(x - x1)^2/w^2 + (y - y1)^2/h^2 = 1$$

   Note however that if $2a + 1$ or $2b + 1$ exceeds the dimensions of the screen attribute or if either a or b is less than 0, you should throw an $InvalidDimensionException$. Also throw an $InvalidDimensionException$ if a or b are ODD!

4. The method *createRectangle(int a, int b)*. This method should accept two EVEN integers as input, a and b. It will then set every element in the screen attribute to TRUE if and only if the element is within a rectangle that is centered in the middle of an array (i.e. at floor(width/2) + 1, floor(height/2) + 1) and that has a width of 2a and height of 2b. Every other element in the screen attribute should be set to false.

   Note however that if $2a$ or $2b$ exceeds the dimensions of the screen attribute

or if either value is less than 0, you should throw an $InvalidDimensionException$. Also throw an $InvalidDimensionException$ if a or b are ODD!

5. The method *writeToOLED()*. This method should draw all of the values in the screen attribute to the OLED display that is reference by the attribute o. There will be a single element in the screen attribute for every pixel on the OLED! To make a pixel at a position (i,j) on the OLED bright you can use the following command:

   $this.oledDisplay.getCanvas().setPixel(i, j, MonochromeCanvas.Color.BRIGHT)$

   To make that same pixel dark you can use the following command:

   $this.oledDisplay.getCanvas().setPixel(i, j, MonochromeCanvas.Color.DARK)$

3 Testing your Classes

As always, a small number of test cases have been provided to help you test your implementations. Make sure you pass these tests! And, as ever, consider writing your own tests, too.

4 What to Submit

With this lab ask ghag you submit your code as well as two submit two images of shapes on your board's OLED. Code to generate shapes can be found between lines 114 and 130 of OLEDDisplay.java. We ask that each image you submit also include your name. An example can be found in Figure 1.

You should therefore submit:

1. ShapeRenderer.java
2. Oval.jpg (or .png)
3. Rectangle.jpg (or .png)

HAVE FUN AND GOOD LUCK!

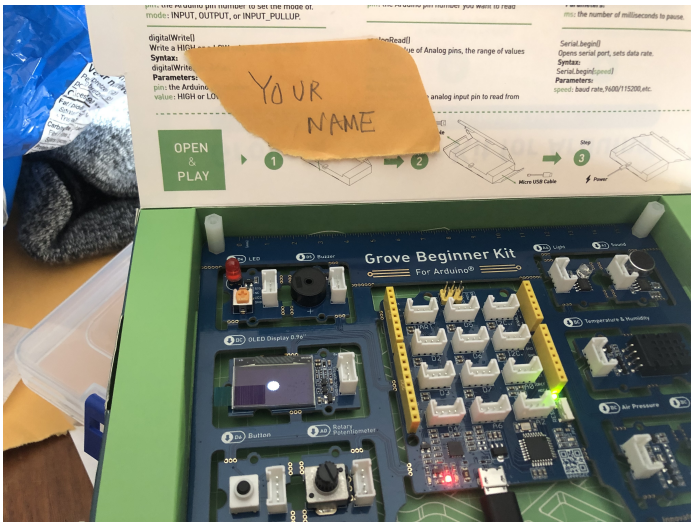

Figure 1: Image of the Grove Board with an OVAL on the display and with YOUR NAME visible in the image. Please also generate an image like this with a RECTANGLE!