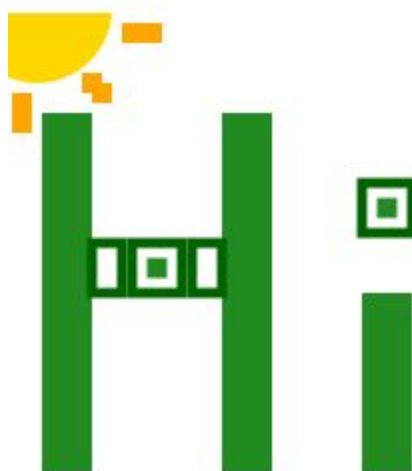


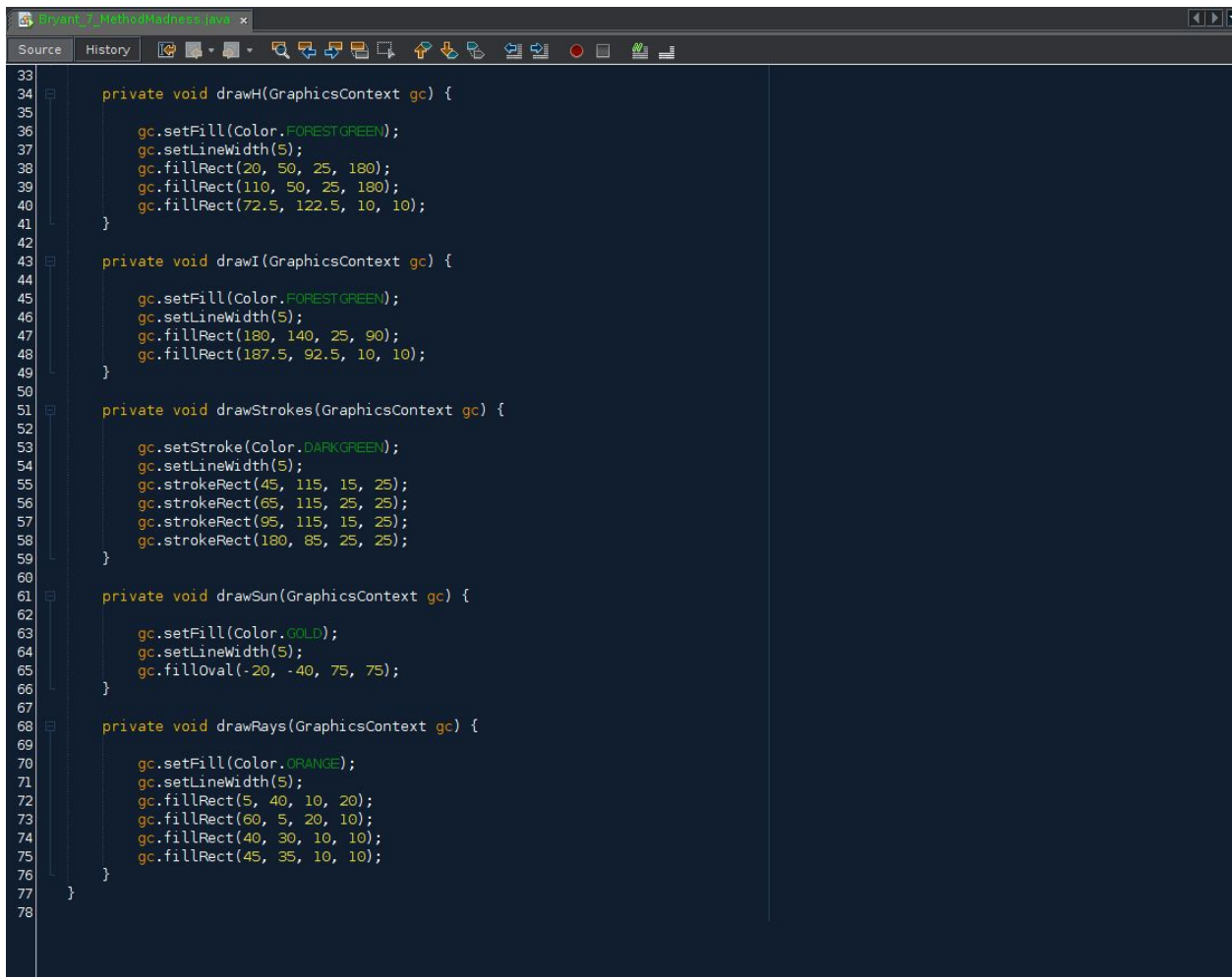
Method Madness: Essay

My method program generates a picture of the word “Hi” along with a little sun in the corner to top it all off. This program is very unique because of the fact I can conveniently avoid conversation by running the program and displaying my initial greeting. They then all walk away because I’m such a nerd as to making a program just to greet someone without use of speech. My design can be seen through the code from the coordinates and size after the main specifications. Example, `gc.fillRect(20, 40, 25, 25)`; the underlined portion contain the coordinates as well as the size of the object. Finally, we had used encapsulation all along with methods. All of my structure is grouped within a specific method in the private void.

The structure of my program consists of five methods, all separated from their function. I had an “H” method, an “i” method, a stroke rectangle method, a “sun” method, and a “rays” method that ended with the creation of this:



The different classes used in my program is all about the same. The main class dished out all of the information it needed to create my picture; it's where the magic happens. Other than this, I primarily used Graphics Context (gc) for my methods to use to draw this thing. I had minimal errors in my program besides the fact that I had only one method starting out, so it took a while to load and display the picture. I handled the errors with care, I wanted to make my program efficient and easy to read. This is the written code that I used:

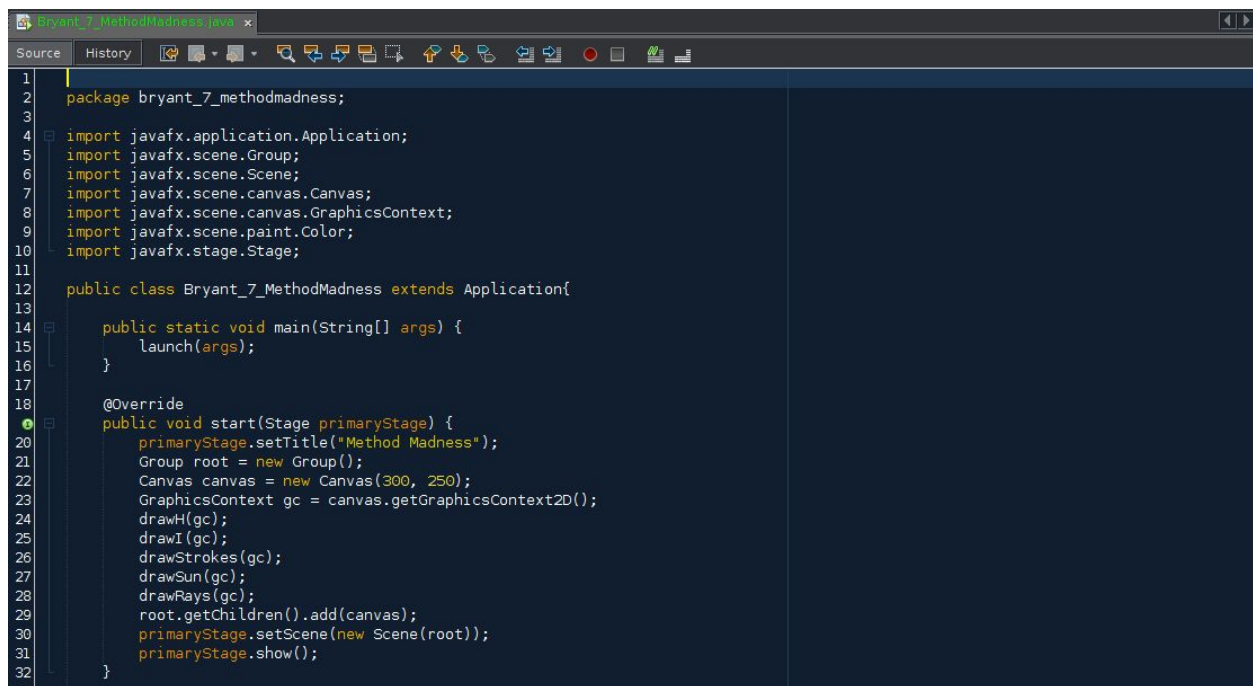


```
33
34 private void drawH(GraphicsContext gc) {
35     gc.setFill(Color.FORESTGREEN);
36     gc.setLineWidth(5);
37     gc.fillRect(20, 50, 25, 180);
38     gc.fillRect(110, 50, 25, 180);
39     gc.fillRect(72.5, 122.5, 10, 10);
40 }
41
42
43 private void drawI(GraphicsContext gc) {
44     gc.setFill(Color.FORESTGREEN);
45     gc.setLineWidth(5);
46     gc.fillRect(180, 140, 25, 90);
47     gc.fillRect(187.5, 92.5, 10, 10);
48 }
49
50
51 private void drawStrokes(GraphicsContext gc) {
52     gc.setStroke(Color.DARKGREEN);
53     gc.setLineWidth(5);
54     gc.strokeRect(45, 115, 15, 25);
55     gc.strokeRect(65, 115, 25, 25);
56     gc.strokeRect(95, 115, 15, 25);
57     gc.strokeRect(180, 85, 25, 25);
58 }
59
60
61 private void drawSun(GraphicsContext gc) {
62     gc.setFill(Color.GOLD);
63     gc.setLineWidth(5);
64     gc.fillOval(-20, -40, 75, 75);
65 }
66
67
68 private void drawRays(GraphicsContext gc) {
69     gc.setFill(Color.ORANGE);
70     gc.setLineWidth(5);
71     gc.fillRect(5, 40, 10, 20);
72     gc.fillRect(60, 5, 20, 10);
73     gc.fillRect(40, 30, 10, 10);
74     gc.fillRect(45, 35, 10, 10);
75 }
76
77 }
78
```

Like I explained earlier, the methods that I created include an “H” and an “I” method along with strokes, a sun, and rays for the sun.

The values passed to my methods, as described in the picture, are primarily the fill and stroke colors, the line width, and the objects with the coordinates that are listed. The values returned by the methods however, are slightly different. The private void set the line width, set the colors, and interpreted the objects sent to it and placed them on a canvas sent to them from the main class. This is the computer's form of communication with itself, it just needs some human help.

Fortunately, all of my methods are being called to by the main class for information. The main class that I wrote looks like this:

A screenshot of an IDE window titled 'Bryant_7_MethodMadness.java'. The code is written in Java and uses a dark theme. It shows a package declaration, several imports for JavaFX classes, and a public class 'Bryant_7_MethodMadness' that extends 'Application'. The class contains a 'main' method and an '@Override' 'start' method. The 'start' method sets the stage title, creates a root group, a canvas, and a graphics context, then calls several drawing methods (drawH, drawI, drawStrokes, drawSun, drawRays) and finally sets the scene and shows the stage.

```
1 package bryant_7_methodmadness;
2
3
4 import javafx.application.Application;
5 import javafx.scene.Group;
6 import javafx.scene.Scene;
7 import javafx.scene.canvas.Canvas;
8 import javafx.scene.canvas.GraphicsContext;
9 import javafx.scene.paint.Color;
10 import javafx.stage.Stage;
11
12 public class Bryant_7_MethodMadness extends Application{
13
14     public static void main(String[] args) {
15         launch(args);
16     }
17
18     @Override
19     public void start(Stage primaryStage) {
20         primaryStage.setTitle("Method Madness");
21         Group root = new Group();
22         Canvas canvas = new Canvas(300, 250);
23         GraphicsContext gc = canvas.getGraphicsContext2D();
24         drawH(gc);
25         drawI(gc);
26         drawStrokes(gc);
27         drawSun(gc);
28         drawRays(gc);
29         root.getChildren().add(canvas);
30         primaryStage.setScene(new Scene(root));
31         primaryStage.show();
32     }
33 }
```

The main class sends the canvas, the title of the canvas, and all of the method names to the individual methods to input what they have and send back with a completed canvas. If you look at the very top of the latest image, you will see that to create this masterpiece, I had to access some special packages so I could make the image. I often used the fillRect modifier in my code as you can see in the second image with my methods. Finally, the constructors I called upon

were all in the basis of Graphics Context. This allowed me to create my images and set my colors along with my line widths.

All in all, I have learned a lot about Computer Science through this project. I hope we will be able to do something more complex later on because it was so cool to learn JavaFX. I am proud of all the work I have put into this wonderful, exciting project I have partaken in. I simply love to imagine things in my head and make them a reality, and now I can with JavaFX!

~~~~