Jasque Saydyk and Duxing Chen

Professor Elwakil, Melton, and McCarty

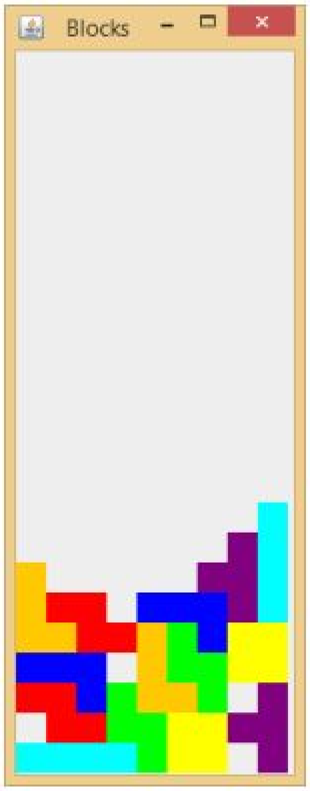
CS 136L Section 3801

27 January 2017

Lab 02 - Drawing Blocks

1. Problem Statement

The program needs to replicate the screenshot shown on the right. To do so, the program needs to draw each shape separately on the canvas, with each square in each block being 20 pixels by 20 pixels.



Requirements
- Frame for the blocks to be drawn in be 10*20 width, 28*20 height
- Each block being 20 by 20 pixels
- Each block be able to rotate
- Each block arranged in the fashion shown on the picture to the right

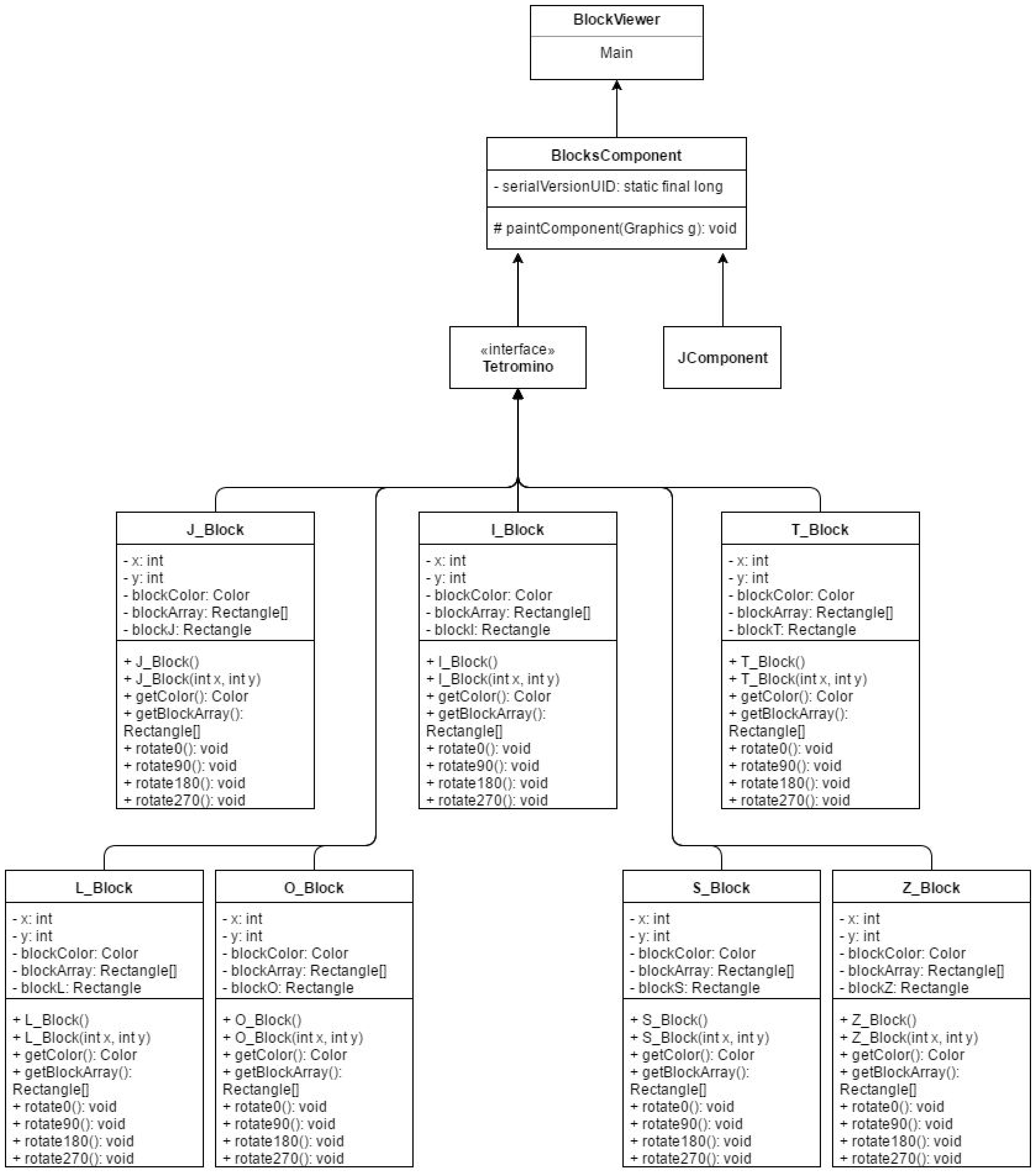
2. Planning

Looking at this problem and the starting code we received as a template to begin with, we originally planned to use methods in the BlocksComponent class to draw each of the Rectangles that make up the shapes. However, we ran into some issues with that method of solving the problem. First, the method used to render the shape on the screen, Graphics2D.fill() can only display one Shape at a time. Shape is an interface, so we could use any class that implements it, however, the Rectangle class best suited our needs for this project.

The question then became, how can we best display all of the rectangles needed to construct each of these shapes without using an extremely long line of Graphics2D.fill()? We then decided to use arrays to store the different Rectangles used to construct each of the different shapes.

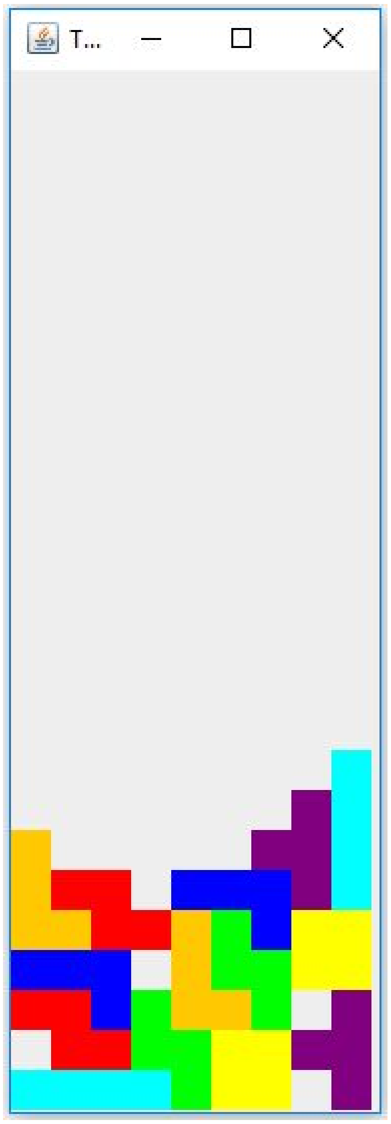
This in turn presented us with one last problem, how to implement the rotate method of the program. While we looked into a way we could do this with math, we could not find a satisfying way to do this, so we decided to just have multiple separate methods redraw the shape in the different positions it could be in each of the different rotations, 0 degrees, 90 degrees, 180 degrees, and 270 degrees.

It was at this point we realize that it would be impossible to implement this inside of the BlocksComponent class, so instead we created the Interface Tetromino that dictated the behavior that each Block class should exhibit, then had each of the block classes implement Tetromino. We had each Block class store the Color of the block, the blocks required to construct it, an array to hold those blocks, and the X and Y coordinates that the block will be drawn at. Below you will find a rough diagram showing the structure of the program.



3. Implementation and Testing

Figuring out how to implement this project took most of the time, along with testing out different ideas on how to do this project. Once we figured out a good solution, we then went about making the various classes and tying them into the BlockComponent class. To the right is the output of the project, correctly mimicking the snapshot given with the project.



4. Reflection

In retrospect, an abstract class may have been a better choice for inheritance rather than an interface, since each of child classes had almost exactly the same fields, which ended up with a lot of repeating code.

The only other thing is that it may have been useful to make a Square class to formalize that each shape is constructed from 20 by 20 pixel squares, rather than using multiple Rectangles knowing that they should be in units of 20. This would also allow for us to put an outline around each square, making the shapes look more aesthetically pleasing. The other big benefit this would bring is that in the game of Tetris, squares can be deleted, leaving only a part of the shape remaining. By making the square the most basic unit, we would have facilitated this future requirement if we were to expand this into a game.

Aside from that, this program is, in my opinion, solidly designed, allowing for more features to be added on as necessary.