CSCI 1933 Project 1

Due Date: February 16th, 2024 (11:55PM)

1 Introduction

Welcome to the first CSCI 1933 project! All students are expected to understand the rules listed below. While some rules may seem unforgiving, all guidelines are made to help the TAs grade efficiently and fairly. As a result, we will generally not make exceptions. Rules listed in the syllabus also apply but may not be listed here.

1.1 Submission

Project 1 is due on **Friday, February 16, 2024** at **11:55pm CST** on **Canvas**.

Submit a zip or tar file on Canvas containing **all your .java files and your README.** Note that the provided Canvas.java class is **not** to be modified, and should not be included in your submission. You are allowed to change or modify your submission, so submit early and often, and verify that all your .java files are in the submission. Failure to submit the correct files will result in a score of zero for all missing parts. Late submissions will be penalized in accordance with the syllabus.

Note: Make sure to include a README.txt in your submission that contains the following information (10 point deduction if missing):

- Group members' names and x500s
- Contributions of each partner (if working with a partner)
- How to compile and run your program
- Any assumptions
- Additional features that you implemented (if applicable)
- Any known bugs or defects in the program
- Any outside sources (aside from course resources) consulted for ideas used in the project, in the format:
  - idea1: source
  - idea2: source
  - idea3: source
- Include the statement: **"I certify that the information contained in this README file is complete and accurate. I have both read and followed the course policies in the 'Academic Integrity - Course Policy' section of the course syllabus."** and type your name(s) underneath.

You may be penalized for a missing or incomplete README.

1.2 Working with a partner

As discussed in lecture, you may work with **one** partner to complete this assignment. If you choose to work as a team, ***please only turn in one copy of your assignment***. Include both of your names and x500s in a comment at the top of each file you submit. In doing so, you are attesting to the fact that both of you have contributed substantially to completion of the project and that both of you understand all code that has been implemented.

1.3 Identification

Include your name and x500 in a comment in all files you submit, even when not working with a partner. Example: // Written by Christopher Dovolis, dovol002

1.4 Questions

Questions related to the project can be discussed with peers in lecture/lab/Discord only in abstract. Such questions might relate to programming in Java, understanding the writeup, or topics covered in lectures and labs. Do not post any code or solutions publicly. Questions that require explaining your solution or posting your code can be asked to TAs in office hours or through help tickets in Discord.

1.5 Grading

Grading will be done by the TAs, so please address grading problems to the TA who graded your project privately.

1.6 Code Style

Part of your grade will be decided based on the "code style" demonstrated by your programming. In general, all projects will involve a style component. This should not be intimidating, but it is fundamentally important. The following items represent "good" coding style:

- **Use effective comments** to document what important variables, functions, and sections of the code are for. In general, the TA should be able to understand your logic through the comments left in the code.

  Try to leave comments as you program, rather than adding them all in at the end. Comments should not feel like arbitrary busy work - they should be written assuming the reader is fluent in Java, yet has no idea how your program works or why you chose certain solutions.
- **Use effective and standard indentation.**
- **Use descriptive names for variables.** Use standard Java style for your names: `ClassName`, `functionName`, `variableName` for structures in your code, and `ClassName.java` for the file names.

Try to avoid the following stylistic problems:

- Missing or highly redundant, useless comments. `int a = 5; //Set a to be 5` is not helpful.
- Disorganized and messy files. Poor indentation of braces (`{` and `}`).
- Incoherent variable names. Names such as `m` and `numberOfIndicesToCount` are not useful. The former is too short to be descriptive, while the latter is much too descriptive and redundant.

The programming exercises detailed in the following pages will both be evaluated for code style. This will not be strict – for example, one bad indent or one subjective variable name are hardly a problem. However, if your code seems careless or confusing, or if no significant effort was made to document the code, then points will be deducted.

1.7 Project Overview

Project 1 will involve implementing several shape classes in Java and using a drawing class we have implemented for you to create a fractal based on your shapes **using recursion**.

Classes you will need to create (more information in Section 2):

- Circle.java
- Rectangle.java
- Triangle.java

Classes you will need to edit (more information in Section 2):

- FractalDrawer.java

Classes given to you (no need to edit or include in your submission):

- Canvas.java

2 Shape Classes (57 points)

You should design classes for three different shapes (triangle, square, and circle) with the specifications listed below. These classes are to be used by the main program (the class with the "main" method), and passed to the Canvas class for drawing. Thus, they must use standard class names and method names in order to work correctly. We will leave class attributes up to you, looking at the values that can be set is a good place to start.

The TAs will be using automated tests to grade your submissions so make sure to use the given method signatures i.e. the function names, order of the input parameters and the output type should match the ones given.

Note: In java, both integer and floating point division are performed using the `/` operator. Integer division floors the result of the division- for example, `1/2` will produce `0`, rather than the (expected) `0.5`. Make sure that if you use any division in your methods for the shape classes, at least one of the operands is of type double. For example, `1.0/2`, `1/2.0`, and `1.0/2.0` will all produce `0.5`.

Note: In order to use `Color` in this project, you will need to import `java.awt.Color` at the top of each shape class.

2.1 Circle Class Methods

- Constructor (Function name should be: Circle)
  Input: x position (double), y position (double), radius (double)

- calculatePerimeter
  Input: none
  Output: perimeter of the circle (type double)

- calculateArea
  Input: none
  Output: area of the circle (type double)

- setColor
  Input: color of the shape (type Color)
  Output: none (void)

- setPos
  Input: x, y position of the center (both doubles)
  Output: none (void)

- setRadius
  Input: radius (double)
  Output: none (void)

- getColor
  Input: none
  Output: color of the shape (type Color)

- getXPos
  Input: none
  Output: x position of the center (double)

- getYPos
  Input: none
  Output: y position of the center (double)

- getRadius
  Input: none
  Output: radius (double)

2.2 Triangle Class Methods

Hint: You can assume the triangles we want to draw are isosceles triangles.

- Constructor (Function name should be: Triangle)
  Input: x position of bottom left corner (double), y position of bottom left corner (double), width (double), height (double)
- calculatePerimeter
  Input: none
  Output: perimeter of the triangle (type double)
- calculateArea
  Input: none
  Output: area of the triangle (double)
- setColor
  Input: color of the shape (type Color)
  Output: none
- setPos
  Input: x, y position of bottom left corner (both doubles)
  Output: none
- setHeight

Input: height (double)
Output: none

- setWidth
  Input: width (double)
  Output: none
- getColor
  Input: none
  Output: color of the shape (type Color)
- getXPos
  Input: none
  Output: x position of the bottom left corner (double)
- getYPos
  Input: none
  Output: y position of the bottom left corner (double)
- getHeight
  Input: none
  Output: height (double)
- getWidth
  Input: none
  Output: width (double)

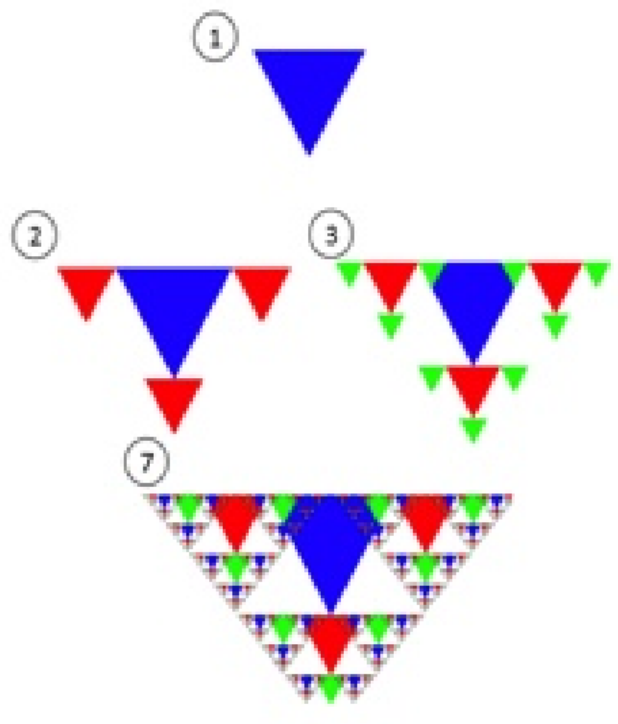
2.3 Rectangle Class Methods

- Constructor (Function name should be: Rectangle)
  Input: x position of upper left corner (double), y position of upper left corner (double), width (double), height (double)
- calculatePerimeter
  Input: none
  Output: perimeter of the rectangle (type double)
- calculateArea
  Input: none
  Output: area of the rectangle (double)
- setColor
  Input: color of the shape (type Color)
  Output: none
- setPos
  Input: x, y position of upper left corner (both doubles)
  Output: none
- setHeight
  Input: height (double)
  Output: none

- setWidth
  Input: width (double)
  Output: none
- getColor
  Input: none
  Output: color of the shape (type Color)
- getXPos
  Input: none
  Output: x position of the upper left corner (double)
- getYPos
  Input: none
  Output: y position of the upper left corner (double)
- getHeight
  Input: none
  Output: height (double)
- getWidth
  Input: none
  Output: width (double)

3 Fractal Drawer Class (38 points)

For this part of the project, you will write a Java program that uses your shape classes to draw a fractal. Fractals are geometric patterns that repeat on themselves at smaller and smaller scales. They have been studied for centuries because of their interesting mathematical properties and often appear in natural objects (e.g. snow flakes, plants). Fractals are also a classic application of **recursion**; you can read more about fractals and their history here. Consider the example below, which illustrates the process of constructing a fractal composed of triangles at several steps in the process. Notice that at each step, triangles of increasingly smaller sizes are drawn at the three points of each existing triangle. *More examples of fractals can be found in Section 5.*



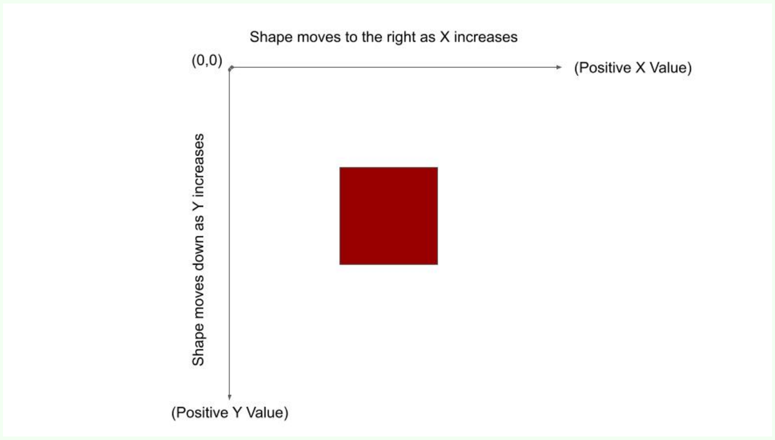
To help you with the drawing, we've already implemented a Canvas class that supports all of the drawing capability you need. You should look at the code if you're interested, but all you'll need to know is how you can interact with Canvas objects.
Here are the method specifications of the Canvas class:

- Canvas() (default constructor): creates drawing of default size
- Canvas(int height, int width): creates drawing of specified width and height
- void drawShape(Circle circleObj)
- void drawShape(Rectangle rectangleObj)
- void drawShape(Triangle triangleObj)

Each of the drawing methods will draw the shape you pass it in the specified location and in the specified color. Here's an example of how you might use the Canvas class to draw a single blue circle:

```
Canvas drawing = new Canvas(800,800);
Circle myCircle = new Circle(0,0,100);
myCircle.setColor(Color.BLUE);
drawing.drawShape(myCircle);
```

Note: Java(JFrame) uses a coordinate system where the origin is in the upper left-hand corner. As either the x and/or increases, the shape will move right and/or down.


Shape moves to the right as X increases
(0,0)
(Positive X Value)
Shape moves down as Y increases
(Positive Y Value)

Note: When you compile the `Canvas.java` class, you may have noticed that you receive the following warning (or something similar):

```
Note: Canvas.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

The provided Canvas class uses the deprecated `JApplet` class. You can ignore this warning, and it does not indicate that anything is wrong with your code.

You will be responsible for filling in the FractalDrawer class. For full credit, your program should have the following features:

- Ask the user for input (choices: "circle", "triangle", or "rectangle") and use the corresponding shape as the base of your fractal.
- Draw a pattern that repeats on itself **at least 7 times** and uses a new color in each repetition (feel free to be creative - you can cycle colors or choose colors at random, as long as overlapping shapes are distinct from one another).

  IMPORTANT: The pattern must repeat on itself **using recursion** for full credit. Changing a shape's size or position should be involved in recursive calls.

- Draw **at least four shapes per layer for rectangles and circles** and **at least three per layer for triangles** (e.g. draw the new shapes on all three points or sides of a triangle, or on four opposite sides of a circle - again, feel free to be creative).
- Compute the **total area of all shapes that form your fractal** and print the result to the screen after your program is finished drawing. You do not need to take into account any

overlap, simply sum the area of all the shapes that were drawn, even if they overlap each other.

To receive full credit, your program will need to be able to draw a fractal **using recursion** for all three possible inputs (circle, triangle, or rectangle). We suggest that the most convenient implementation is to simply implement three different methods, one that draws a circle fractal, one that draws a triangle fractal, and one that draws a rectangle fractal.

Note: If nothing appears on the canvas when you run your program, or of only a partial fractal appears, try resizing the canvas window. If you need to do this to display the entire fractal, you will not be penalized. The TAs are aware that many fractals will only appear after resizing the window.

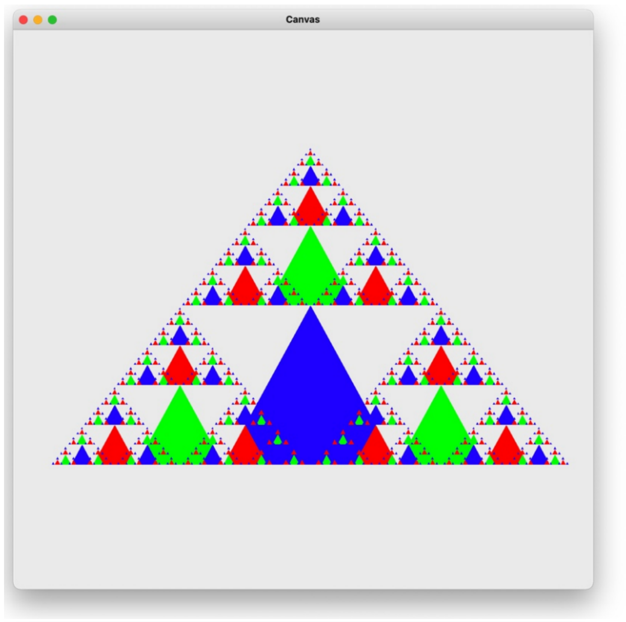
4 Grading Information

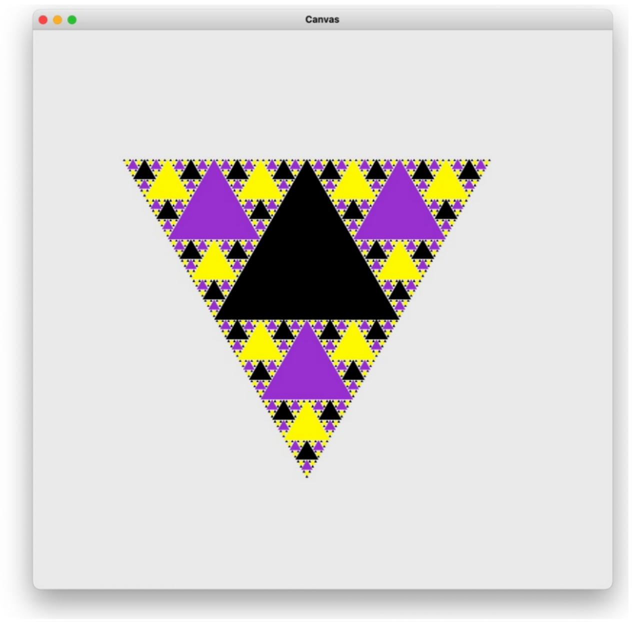
Your program will be graded according to the following score breakdown:

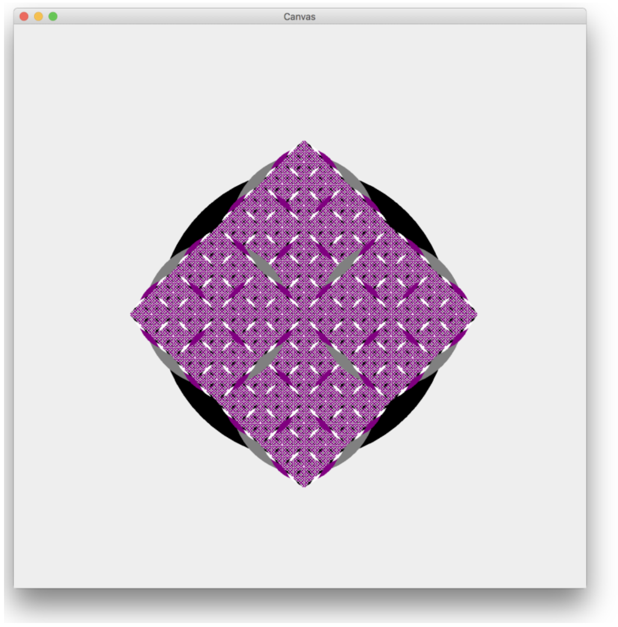
- **Style:** 5 points
- **Shape Classes:** 57 points
- **Fractal Drawer Class:** 38 points

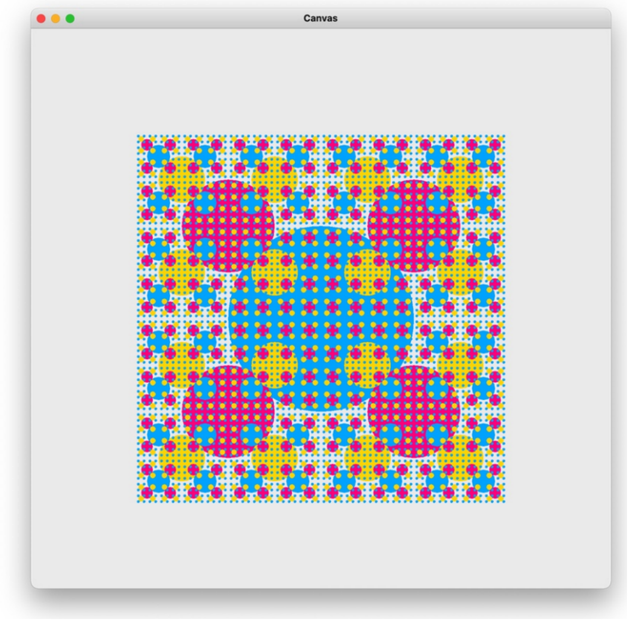
5 Fractal Examples

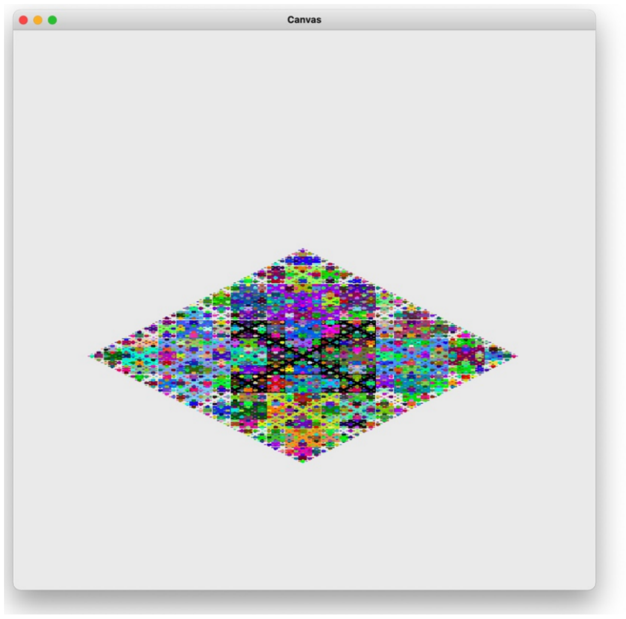
Here are some examples of fractals for all three shapes, if you want some inspiration. Your fractal patterns do not have to match these; feel free to come up with your own patterns, as long as they meet the basic requirements listed in section 3!

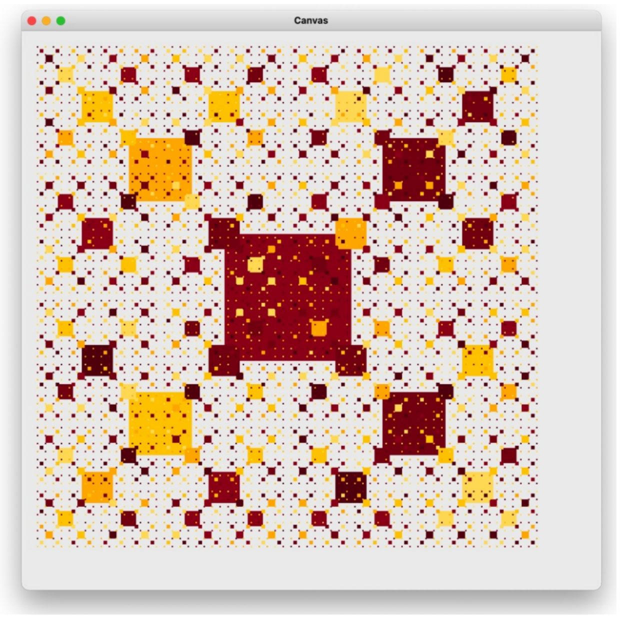

Canvas


Canvas


Canvas


Canvas


Canvas


Canvas