Recursion and Fractals
Due: Sunday, Oct 23 at 11:59PM

1 Recursion and Fractals Specification

1.1 Lab Instructions

- This is an *individual* lab.
- Make sure to read through all of the specifications so your submission is complete.
- Follow all the submission steps in the Setup document by the lab deadline.

1.2 Lab Link

The skeleton code for the lab is available at `https://classroom.github.com/a/jDwcxpk7`.

1.3 Introduction

For this lab, you will implement a Java class that draws a fractal shape. A fractal is an image that looks the same when scaled/rotated different amounts. Fractals appear in nature quite frequently because their creation process is very simple but can lead to complicated results.

Fractal shapes are also concise to describe in code, so we can draw a complicated image with few lines of code.

Two classes are included in the project: **DrawableFrame** is a JFrame that you can draw lines on. **FractalDrawing** is an abstract class that you should use as the basis for your own class. This class provides code to initialize the frame, and provides you with a Graphics object that you can draw lines on. **Do not** modify either of these classes!

The **FractalDrawing** class also has a couple of helpful methods. Calling `show()` will make your current drawing appear in the frame. Calling `wait(long)` will make your program pause for some milliseconds. You can use this to slow down the drawing operations so that you can see more granular changes in your drawing.

1.4 Class Construction

Create a new class in the `edu.wit.cs.comp2000` package named **Sierpinski.java**. The class should extend `FractalDrawing`, which has a useful constructor and other important methods.

1.5 main()

In a `main()` method, create a new `Sierpinski` object. The constructor of the superclass should handle all of the drawing, so all you need to do is create the object.

1.6 drawFractal()

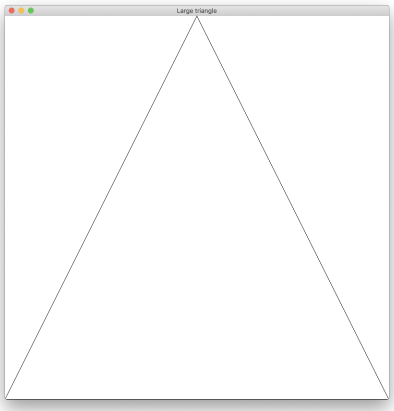
Implement this wrapper method to invoke a recursive drawing method with its starting parameters. In the case of Sierpinski's triangle, the parameters should be the x/y coordinates of the three vertices of the largest triangle left to draw. A signature might look like:

```
private void drawSierpinski(int x1, int y1, int x2, int y2, int x3, int y3)
```

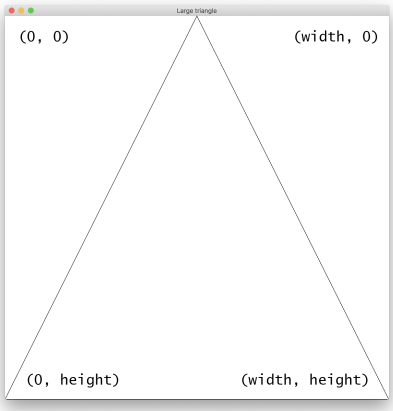
with vertices `(x1, y1), (x2, y2), (x3, y3)`. The starting vertices should fill up the viewing window – you can refer to the `width` and `height` of the window for the initial arguments. How can you calculate the vertices based on the window size?

1.7 drawSierpinski()

Begin by drawing a large triangle. This can be achieved by calling `graphics.drawLine()` three times, once for each pair of vertices. The `graphics` object is defined in the superclass.



First large triangle – three lines

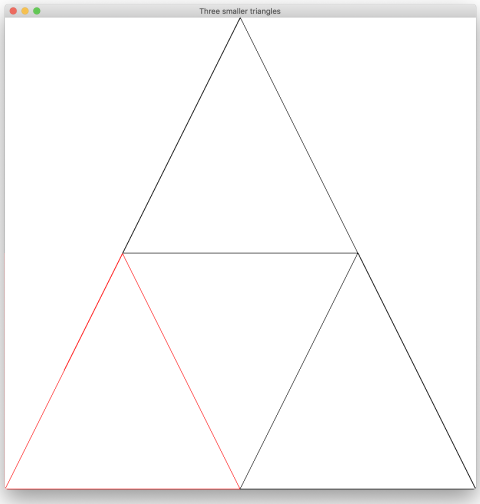


Coordinates of drawing corners

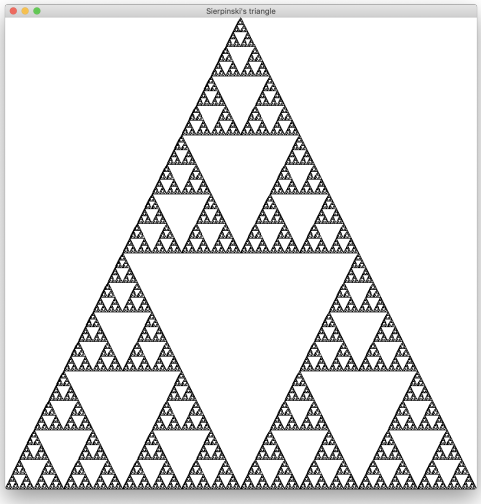
I would recommend printing debug information about the arguments each time you call `drawSierpinski` to better understand what work your code is doing. (Comment it out for your final submission.)

Once you can draw a single triangle, it's time to start thinking about recursive calls to draw smaller triangles, and a base case – when to stop drawing. After drawing the largest triangle, you should draw three smaller triangles as the next step. Where will these vertices be in relation to the original triangle? They should each use two midpoints and one corner of the current triangle. See the image below for the second level of triangles to draw, including red highlighting of one of those triangles.

Before testing a recursive method, we need a base case! How small can triangles get? At some point, a small enough triangle won't be visible at all. As a base case, stop when a triangle's line segments have a length of 3 or less.



Three smaller triangles with one highlighted



The full results

1.8 Testing

There are not separate JUnit tests for this lab. The visual results are the tests. If you see a drawing of Sierpinski's triangle, your code ran successfully. If you don't, you still have work to do.

If you want to slow down the drawing so it doesn't appear all at once, you can put a `wait` call inside `drawSierpinski`.

If you are curious, play around with the initial arguments to your recursive method. You can skew the entire fractal shape with a single small change!

1.9 Available Resources

- Lecture slides
- Other sections of the provided code
- me
- The textbook
- **DO NOT** refer to or use online implementations

2 Double Check:

- Have you created and implemented the `Sierpinski` class?
- Have you added the file to your repository and committed/pushed your code?

3 Grading

Grades and any comments for the lab will be posted to your project on github. Grades will also be posted to Brightspace, eventually.