Lab 7: Sierpinski's Gasket[1]

CSCI 1101B – Fall 2014

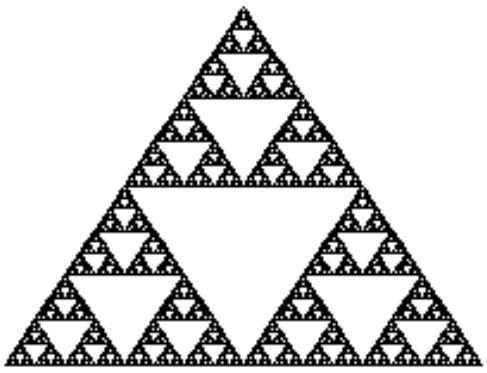
Due:

November 30 @ 10 pm (Monday lab)

December 1 @ 10 pm (Tuesday lab)

Objective: To gain experience using recursion.

Interesting geometric patterns can be produced by drawing smaller and smaller copies of a shape inside an initial copy of that shape. One such pattern is called Sierpinski's Gasket, which you will draw in this week's lab:



Because this pattern is going to be constructed recursively, you will need to design a class with a recursive constructor (and a recursive `move` method). We will call this the `ComplexTriangle` class. One way to think about constructing this recursively is to draw the outermost triangle (given its vertices) and then construct three smaller `ComplexTriangle`s inside that triangle: one at the top, one at the lower left, and one at the lower right. Thus, the `ComplexTriangle` class will need instance variables for the lines composing the outer triangle as well as instance variables for the three `ComplexTriangle`s inside. Besides the constructor, the only other methods that will be needed are a `move` method and a helper method `getMidpoint` (described below).

The constructor will take parameters for the vertices of the triangle (in the order `left`, `right`, and `top`) and the drawing canvas. It should first draw lines between the vertices to form a triangle. Then, if the bottom edge of this triangle is long enough (say, with length strictly greater than 5 pixels), it should construct three more `ComplexTriangle`s inside this triangle. To create the smaller triangles, you should find the midpoints of each of the sides of the triangle just drawn and then create a `ComplexTriangle` in the top, lower-left, and lower-right portions of the triangle (leaving the middle blank). If the edge of the outer triangle is not long enough, then the constructor should not make the three `ComplexTriangle`s.

[1] Adapted from a lab provided with *Java: An Eventful Approach*, K. Bruce, A. Danyluk, and T. Murtagh

To avoid writing duplicate code to find a midpoint three times (for the three edges of the outer triangle), you should write a helper method called `getMidpoint` that is sent a `Line` and returns the midpoint of that line (i.e., a `Location`). To help in writing this method, you can use the `getStart` and `getEnd` methods of the `Line` class to get the start and end `Locations` of the line, respectively.

You should also write a `move` method that moves the lines forming the triangle as well as the contained `ComplexTriangle`s.

I have provided you with an `Events` class and the bare skeleton of a `ComplexTriangle` class in the starter project. The `begin` method constructs a new `ComplexTriangle` object. The `onMousePress` and `onMouseDrag` methods use the `move` method that you will write in your `ComplexTriangle` class to drag the `ComplexTriangle` around, even when the mouse is not inside the `ComplexTriangle` (because there's no straightforward way to check whether the user clicked inside the `ComplexTriangle`). You should not need to modify the `Events` class at all (though feel free to change the constants defining the triangle while experimenting with your program).

Tips

- To start out, it is often a good idea to do a simpler version of the problem. For example, first have the constructor of `ComplexTriangle` just draw a simple triangle out of lines. Then try drawing only the `ComplexTriangle` in the top part of the triangle. Then add the lower right `ComplexTriangle`, then finally add the triangle in the lower left portion.
- You will find the `distanceTo` method defined in the `Location` class useful in determining whether to draw the inner triangles. If you have a `Location` object, you can call this method along with a `Location` object argument, and the result will be a double indicating the distance between the two locations; for example:

```
Location x = new Location(20, 50);
Location y = new Location(65, 150);
double distanceXY = x.distanceTo(y);
```

- If you are not careful in writing your program, you may get a `StackOverflowError` when you run the program. This will occur if your program continues to construct `ComplexTriangle`s without ever terminating. You should avoid this by making sure your `ComplexTriangle` constructor stops creating `ComplexTriangle`s at some point.

Submitting Your Work

Submit your compressed lab folder (named using your user ID) in the usual way on Blackboard. Remember to put your name in your `ComplexTriangle` class file and fill in all missing documentation (e.g., method and program summaries).