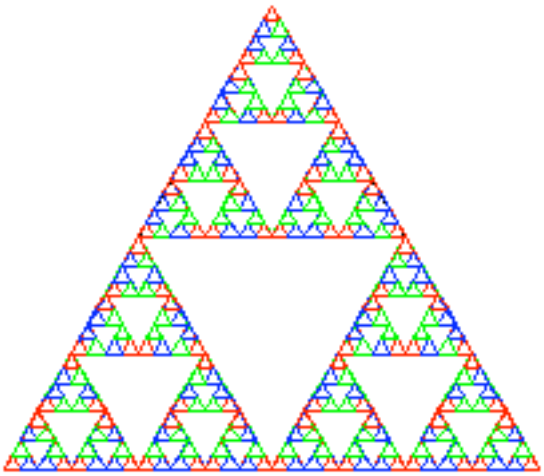
Recursion Lab
Doodler

Objective: To gain experience using recursion.

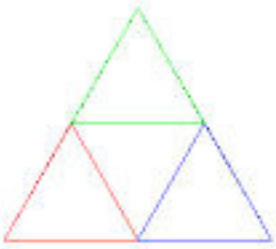
Recursive doodling. This assignment has two parts, each of which involves writing a recursive program.

When we doodle, those of us who are less than gifted artistically often resort to making simple geometric patterns on the backs of our notebooks, envelopes, napkins or whatever. One of the all-time favorites is to draw smaller and smaller copies of a single shape inside of another. For example, one can take a shape like a triangle or rectangle and then divide it up into smaller shapes (perhaps by drawing lines between midpoints of adjacent sides) and then divide each of the new regions and so on until all you have is a big blob of ink on the paper.

Fortunately, Java's drawing resolution is good enough that if we get it to do this kind of doodling, we can produce things quite a bit more interesting than a big blob. For example, repeatedly dividing triangles into smaller triangles can lead to a picture like that shown below.



This image was formed by first drawing a triangle and connecting the midpoints. This breaks the triangle into four smaller triangles (at the top, center, lower-left, and lower-right).



The center triangle is then left untouched, but the top, lower-left, and lower-right triangles are further triangulated in exactly the same way until the length of the edges is smaller than 6 pixels long. (In case you are interested, this shape is called Sierpinski's Gasket.)

How to Proceed

Part 1: Your first task is this lab is to use recursion to create a rendition of the Sierpinski Gasket that you can drag around the canvas.

Because this triangle doodle is going to be represented as a recursive structure, you will need to design an interface for the doodle (called `TriangleDoodle`), a base class (called `EmptyTriangleDoodle`), and a recursive class (called `ComplexTriangleDoodle`). The only method that will be needed for triangle doodles is a `move` method, so design the interface appropriately.

The `EmptyTriangleDoodle` class represents an empty doodle. When you create an empty doodle, nothing is drawn on the screen, so the constructor and move method are both pretty trivial.

The `ComplexTriangleDoodle` is more complicated. It will need instance variables for the lines composing the outer triangle as well as instance variables for the three triangle doodles inside.

Its constructor will take parameters for the vertices of the triangle 3 (in the order left, right, and top), and the drawing canvas. It should first draw lines between the vertices to form a triangle. It will then construct three more triangle doodles inside this triangle. If any of the edges of the new triangle are long enough (say, with length greater than or equal to 6 pixels), then the constructor should find the midpoints of each of the sides and then create complex triangle doodles in the top, lower-left, and lower-right portions of the triangle (leaving the middle blank). If none of the edges are long enough, then the constructor should just create three empty triangle doodles for the instance variables (as we did with scribbles). The move method should move the lines forming the triangle as well as the contained triangle doodles.

We have provided you with a main program that extends `WindowController`. The `begin` method constructs a new object from class `ComplexTriangleDoodle`. The `onMousePress` and `onMouseDrag` methods use the `move` method to drag the triangle doodle around (even when the mouse is not in the triangle doodle).

To start out, it often a good idea to do a simpler version of the problem. For example, first have the constructor of `ComplexTriangleDoodle` just draw a simple triangle out of lines. Then try drawing only the triangle doodles in the top part of the triangle. Then add the lower right, then finally adding those in the lower left portion.

You will also find the `distanceTo` method defined in the `Location` class useful to solve this problem:

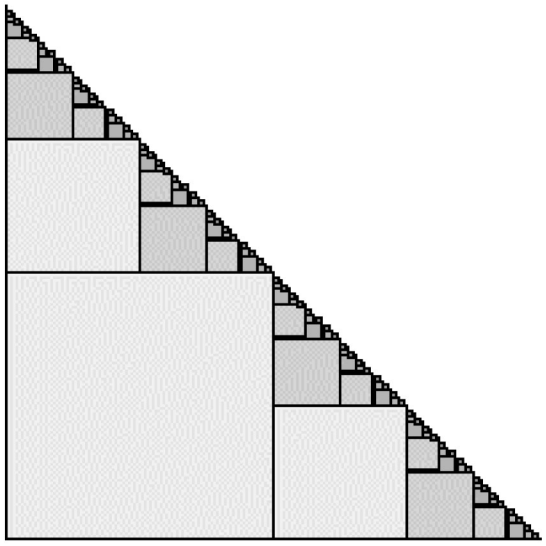
```
public double distanceTo( Location point )
```

if you are not careful in writing this program (and the next one), you may get a StackOverflowError. This will occur if your program continues to construct new triangle doodles without ever terminating. You can avoid this if you make sure you have written the base class correctly and that the complex doodle invokes its constructor only to create simpler doodles, eventually creating only empty doodles.

While the only requirement is that your program be able to draw the gasket using black lines, you can experiment with a color scheme to achieve a picture like that shown in the online version of this handout. This extension will require the addition of a color parameter to the triangle doodle constructors (and the addition of such a parameter to the construction in the `begin` method of the main class).

You might also want to get extra practice with recursive data structures by adding methods to let you draw multiple gaskets and manipulate them with the mouse.

Part 2: Your second task is to draw a movable version of the picture given below:



Let's call this variegated stairs (got to love those big words!). A variegated stairs drawing is constructed by first drawing a large square. Then draw variegated stairs half as large on top of the square, and variegated stairs, also half as large, immediately to the right of the square. The size of your initial square should be a power of 2 (128 is a good choice) so that you don't end up with gaps between the rectangles when you divide the size in half. Stop creating stairs when the squares are less than 3 pixels on a side.

Again, the only requirement is that your program be able to draw the stairs using black lines, but you may wish to add extra functionality for extra practice. You can make the picture produced look nicer by filling the interiors of the squared drawn at each level of recursion with a different shade. Fill the first square with the color created by `new Color(225,225,255)`. Then, at each level of recursion, use the color obtained by decreasing the first two components (the red and green components) by 30. That results in nice shading.

Submitting Your Work Before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations. As ever, please submit your work both on GitHub and on paper.