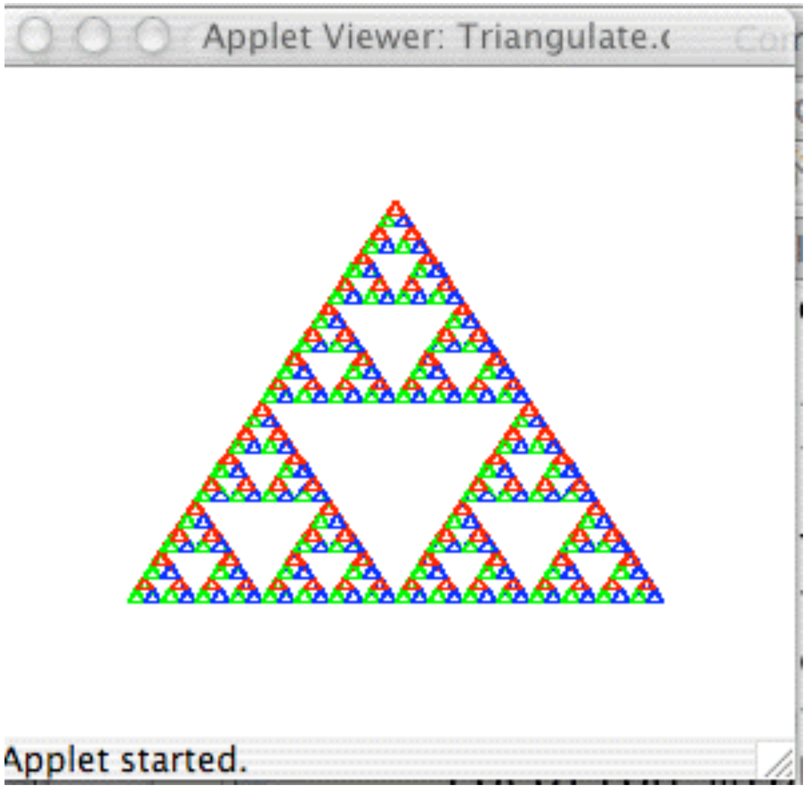
CS 51 Laboratory # 7
Recursion Practice

Due: Monday 11PM, but hopefully will be turned in by the end of the lab period.

Objective: To gain experience using recursion.

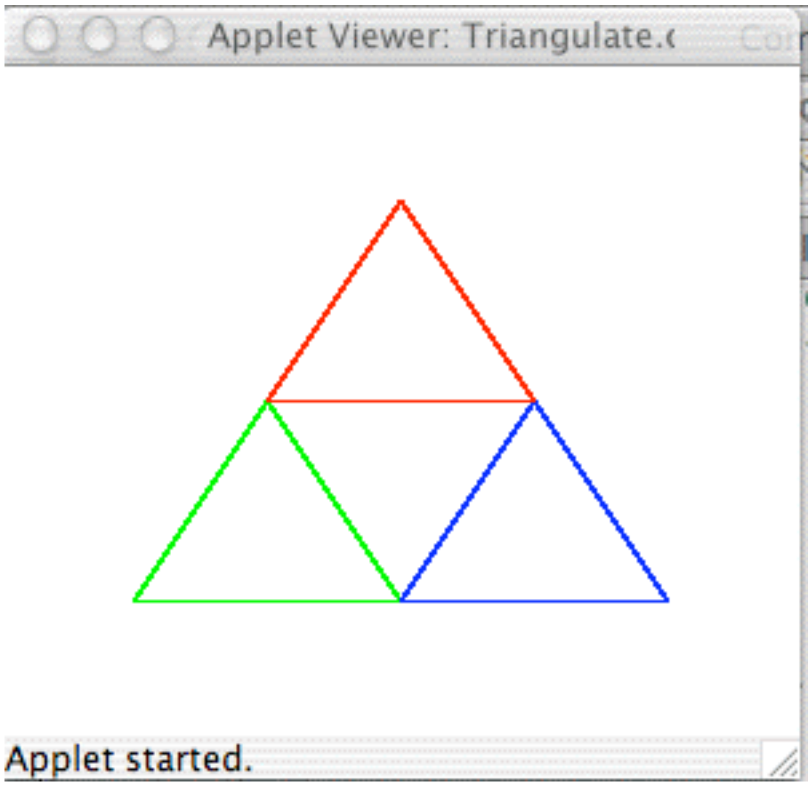
Recursive doodling. This week's assignment has two parts, each of which involves writing a recursive program.



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 or napkins. 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 above. The online version has a working version that you can drag.

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 all of the edges is less than 12 pixels long (by the way, this shape is called Sierpinski's Gasket).

How to Proceed Drag the starter folder to your directory as usual. Inside the starter, you will find two subfolders: `Triangulate` is the starter for Part 1, `Stairs` is the starter for Part 2.

Because this triangle doodle is going to be 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. Like the empty scribble, 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 (in the order left, right, and top), and the drawing canvas. It should then draw lines between the vertices to form a triangle. It will also 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 12 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 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.

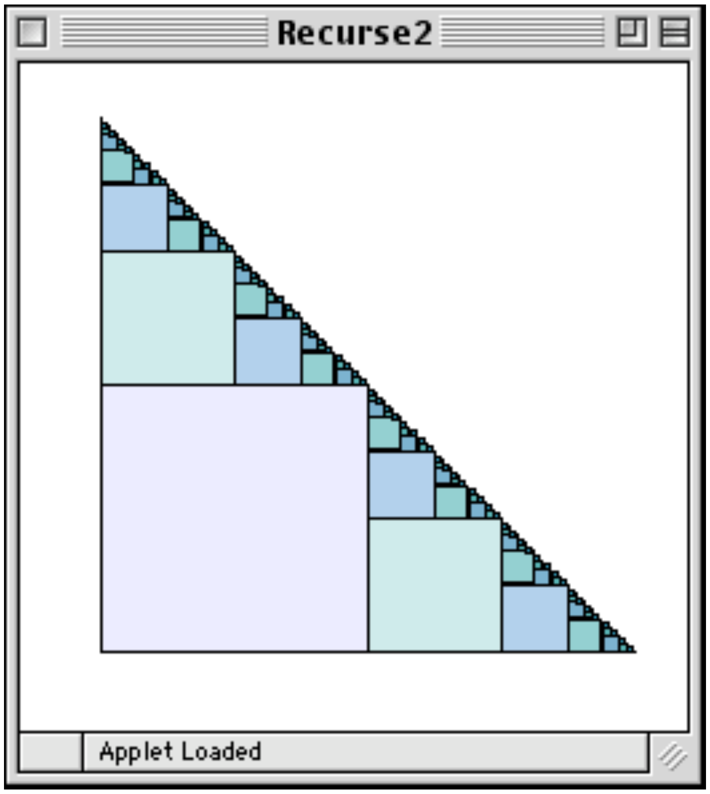
The window should be set to be 400 pixels square for this problem. You will also find the `distanceTo` method defined in the `Location` class useful to solve this problem:

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

Once this is working, 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 `Triangulate` class).

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.

Part 2: Your second task is to draw the picture given below:



We call the above drawing variegated stairs. To construct variegated stairs, first draw a large square. Then draw variegated stairs half as large on top-left of the square, and variegated stairs, also half as large, immediately to the lower-right of the square. The size of the initial square is normally 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 non-empty stairs when the squares are less than 3 pixels on a side.

As above, you will need to create an interface, `StairsInterface`, and two classes, `EmptyStairs` and `VariegatedStairs`. We have provided the class `StairController`, which extends `WindowController`. This time that class will only drag the stairs around if the user presses the mouse down on the stairs. Thus you will need to create methods `contains` and `move`. The constructor call in the `begin` method is:

```
stairs = new VariegatedStairs(lowerLeft, initialSize, initialColor, canvas);
```

Notice that the constructor passes the lower left corner.

In the online version of this handout, you can see that we made it look nicer by drawing the first square in a color created by `new Color(225,225,255)`. At each level of recursion, the color of the first two components (the red and green components) are decreased by 30. That results in nice shading.

You can obtain the red, green, and blue components from a color with the methods:

```
int getRed();
int getGreen();
int getBlue();
```

The window for this problem is 450 pixels wide and 400 pixels tall.

Notice that for this recursive drawing you can only drag it around by actually clicking on the object (because you implemented the `contains` method).

Submitting Your Work Before submitting your work, make sure that each of the .java files includes a comment containing your name. Also, 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. Use the Format command in the Source menu to make sure your indentation is consistent. Refer to the lab style sheet for more information about style.

Turn in your project the same as in past weeks; though make sure that the folder name begins with your last name (and includes the lab number).

This lab is due Monday at 11 p.m., though like the last lab I wouldn't be surprised if most of you complete it during the lab period.

Table 1: Grading Guidelines

Value	Feature
	Syntax Style (3 pts total)
1 pt.	Descriptive comments
1 pt.	Good names, constants
	Semantic style (3 pts total)
1 pt.	conditionals and loops
1 pt.	Parameters, variables, and scoping
1 pt.	Interface
1 pt.	Base class
	Correctness (4 pts total)
	Triangles
1/2 pt.	Draws initial triangle correctly
1/2 pt.	Calculates midpoints properly
1/2 pt.	3 recursive calls
1/2 pt.	Stopping condition correct
	Stairs
1/2 pt.	Draws initial rectangle correctly
1/2 pt.	Calculates position/dimensions of new rectangles properly
1/2 pt.	2 recursive calls and color scheme correct
1/2 pt.	Stopping condition correct
	Extra credit (1 pt max)
1 pt.	Third recursive program that draws something interesting