**Snowflake**

Many of the next-level ADTs are recursive by nature. In this lab, you'll get some more practice writing basic recursive methods, as well as an introduction to the interesting world of recursive graphics.

**All the method below must be completed without using loops (they should be recursive).**

1. Complete the method double sumReciprocals(int n) that returns the sum of the first n reciprocals (1 + 1/2 + 1/3 + 1/4 + ... + 1/n).

sumReciprocals(10) >>> 2.9289682539682538

1. Complete the method int productOfEvens(int n) that will return the product of the first n even numbers.

productOfEvens(4) >>> 384 //2 \* 4 \* 6 \* 8 == 384

1. (Riddle) What is it that no man ever yet did see, which never was, but always is to be?
2. Complete the method void doubleUp(Stack<Integer> nums) that replaces every element in the stack with two of that element.

(3, 7, 12, 9) >>> (3, 3, 7, 7, 12, 12, 9, 9)

1. Complete the method void countToBy(int n, int m) that prints how to count to n in increments of m separated by commas. You can assume it will always be possible.

countToBy(34, 5) >>> 4, 9, 14, 19, 24, 29, 34

countToBy(25, 4) >>> 1, 5, 9, 13, 17, 21, 25

1. Complete the method int matchingDigits(int a, int b) that returns the number of digits that "match" in the two (non-negative) integer parameters. Digits match if they have the same value at the same (relative) index.

matchingDigits(1000, 0) >>> 1

matchingDigits(298892, 7892) >>> 3

1. Complete the method void printThis(int n) that will print the pattern seen below. The middle character should always be an asterisk ("\*"); however, if n is even there should be two ("\*\*").

printThis(1): \*

printThis(2): \*\*

printThis(3): <\*>

printThis(4): <\*\*>

printThis(5): <<\*>>

printThis(6): <<\*\*>>

printThis(7): <<<\*>>>

printThis(8): <<<\*\*>>>

1. Complete the method void printNums2(int n) that will the pattern seen below. Note that odd values of n will have a single 1, while even values of n will print two 1s.

printNums2(1): 1

printNums2(2): 1 1

printNums2(3): 2 1 2

printNums2(4): 2 1 1 2

printNums2(5): 3 2 1 2 3

printNums2(6): 3 2 1 1 2 3

printNums2(7): 4 3 2 1 2 3 4

printNums2(8): 4 3 2 1 1 2 3 4

printNums2(9): 5 4 3 2 1 2 3 4 5

printNums2(10): 5 4 3 2 1 1 2 3 4 5

**Snowflake**

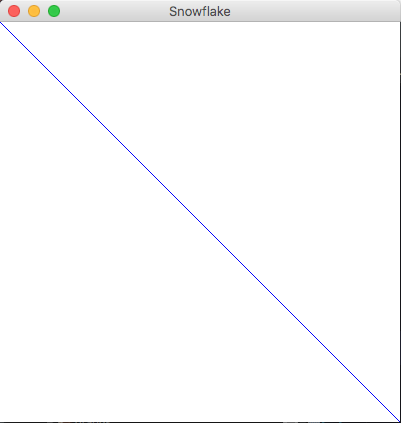
Relatively simple recursive code can lead to some surprisingly complex and interesting graphics. In this lab you will draw recursive graphics using standard Java library classes, so first a little background:

*A JPanel is a container object that can be drawn on and can contain other components (windows). Its paintComponent(Graphics g) method is called when it needs to be drawn. The Graphics object g contains methods to draw on that part of the screen; think of it like the paintbrush. Think of the panel as a rectangle of graph paper. Each small square is a pixel.*

*A JPanel is a rectangle width pixels across and height pixels tall. A location in the rectangle is described as (X, Y) where X and Y are integers. The (0, 0) location is at the top left.*

*Columns are numbered 0 to width-1 starting at the left. Rows are numbered 0 to height-1 starting at the top. X grows to the right, Y grows going down.*

Open the provided starter code, which gives you a simple framework for drawing (if you'd like to learn more about Java graphics, see [here](https://chortle.ccsu.edu/Java5/Notes/chap55/ch55_1.html)). Compile and run, you should see the below:

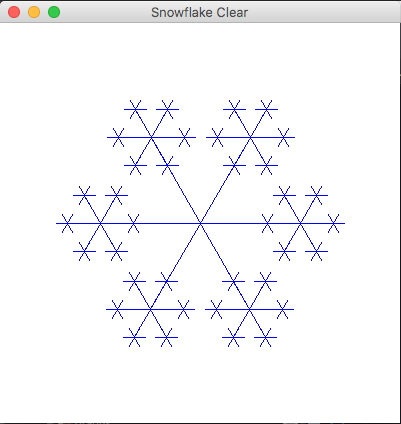


Start by writing a method drawStar that draws a 6-pointed star. This will get some of the graphics details out of the way and will explain some of the geometry. The drawStar method should be called from the paintComponent method (this method is called when the panel needs to be drawn), and should be supplied with the Graphics g parameter (the "paintbrush"). Think of this figure as six lines that radiate from a center (not three crossed lines). Draw the star based on the size of the window (not a hard-coded constant); make sure to use smaller of the height and width dimensions so it will always fit.



If you *truly* can't geometry, there is help in the lab folder. However, you should be able to figure out the math for this yourself.

Once you have the star drawn, it's time to add recursion and produce a snowflake. To draw a snowflake, draw smaller stars at the ends of the six lines that make up the big star - don't you already know where these locations are? Also, every recursive method should have a base case; in this case, that would be when the line length of a potential snowflake is sufficiently small.



Finally, add the code to make a nice-looking blizzard with random colors:



Using what you learned, create a new class **Sierpinski.java** that will produce the following image:



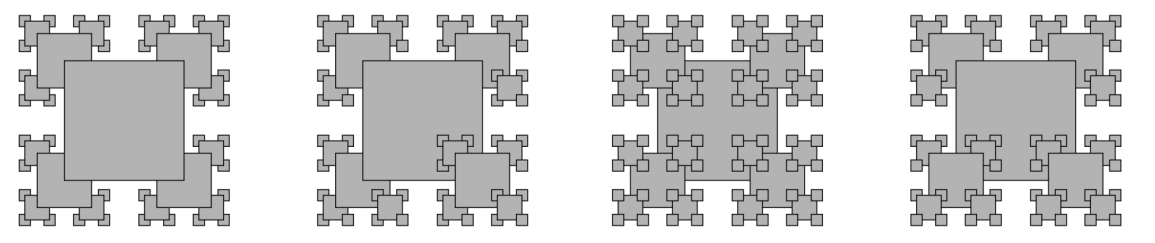
The only drawing (Graphics) method this figure uses is drawLine. The basic figure is a triangle. But rather than draw the full-sized triangle, draw three small triangles:

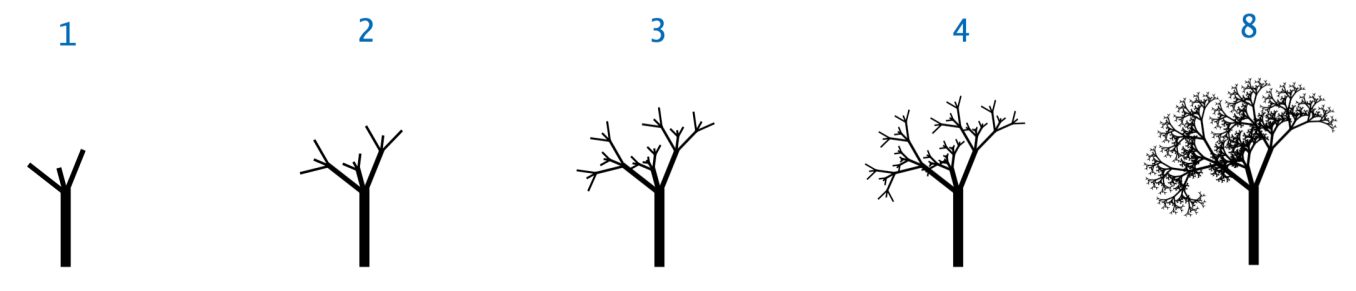
* The left vertex of the lower left triangle is the same as the left vertex of the big triangle. Its other two vertices are halfway along the bottom and left sides of the big triangle.
* The right vertex of the lower right triangle is the same as the right vertex of the big triangle. Its other two vertices are halfway along the bottom and right sides of the big triangle.
* The top vertex of the top triangle is the same as the top vertex of the big triangle. Its other two vertices are halfway along the right and left sides of the big triangle.

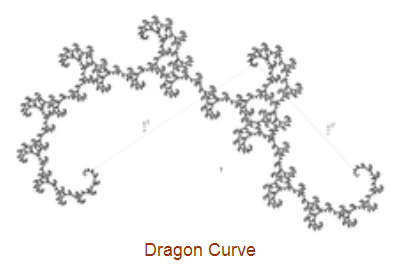
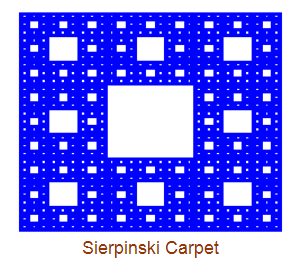
Write a method drawTriangle(Graphics g, int lowerLeftX, int lowerLeftY, int side) that draws a triangle with a lower left corner at (lowerLeftX, lowerLeftY) if side is small. Otherwise, it draws three triangles with side side/2 nested inside the big triangle. You may find it useful to remember that the height of an equilateral triangle is side \* Math.sqrt(3) / 2.

**(Optional) Other recursive graphics**

Simple recursive drawing code can lead to amazingly intricate graphics. Try the following challenges, and feel free to experiment with your own! If you come up with something cool, let me know!

Types of recursive squares:

Recursive trees:



Mondrian art



Koch snowflake

