Assignment #2: Simple Java Programs
Due: 1:15pm on Monday, July 8th

Your job in this assignment is to write programs to solve each of these six problems. You should start by downloading the starter project for Assignment #2 from the CS106A assignment page (go to the CS106A web site and click the **Assignments** link). The starter project will provide java files for you to write your programs in.

1. In high-school geometry, you learned the Pythagorean theorem for the relationship of the lengths of the three sides of a right triangle:

    $$a^2 + b^2 = c^2$$
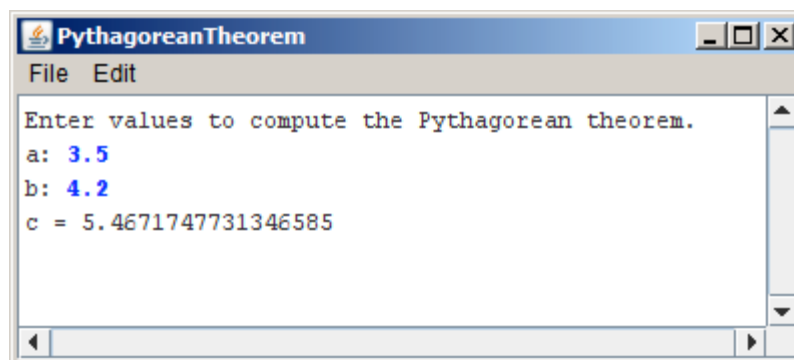
    which can alternatively be written as:

    $$c = \sqrt{a^2 + b^2}$$

    Most of this expression contains simple operators covered in Chapter 3. The one piece that's missing is taking square roots, which you can do by calling the standard function **Math.sqrt**. For example, the statement

    ```
    double y = Math.sqrt(x);
    ```

    sets **y** to the square root of **x**.

    Write a **ConsoleProgram** that accepts values for **a** and **b** as **double**s (you can assume that **a** and **b** will be positive) and then calculates the solution of **c** as a **double**. Your program should be able to duplicate the following sample run:

    ```
    PythagoreanTheorem                                    _ □ X
    File  Edit
    Enter values to compute the Pythagorean theorem.
    a: 3.5
    b: 4.2
    c = 5.4671747731346585
    ```

2. Douglas Hofstadter's Pulitzer-prize-winning book *Gödel, Escher, Bach* contains many interesting mathematical puzzles, many of which can be expressed in the form of computer programs. In Chapter XII, Hofstadter mentions a wonderful problem that is well within the scope of the control statements from Chapter 4. The problem can be expressed as follows:
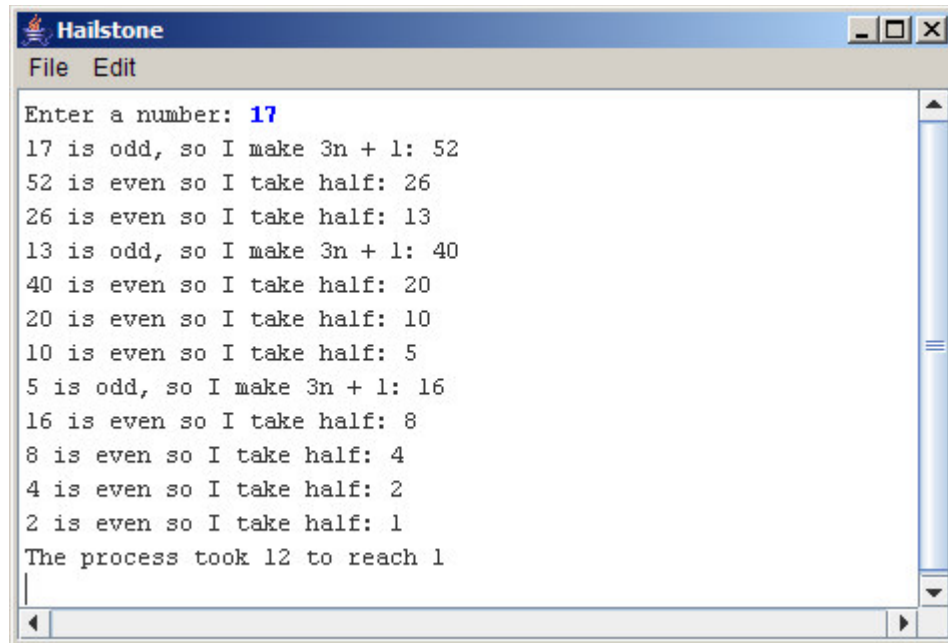
> Pick some positive integer and call it n.
> If n is even, divide it by two.
> If n is odd, multiply it by three and add one.
> Continue this process until n is equal to one.

On page 401 of the Vintage edition, Hofstadter illustrates this process with the following example, starting with the number 15:

15	is odd, so I make $3n+1$:	46
46	is even, so I take half:	23
23	is odd, so I make $3n+1$:	70
70	is even, so I take half:	35
35	is odd, so I make $3n+1$:	106
106	is even, so I take half:	53
53	is odd, so I make $3n+1$:	160
160	is even, so I take half:	80
80	is even, so I take half:	40
40	is even, so I take half:	20
20	is even, so I take half:	10
10	is even, so I take half:	5
5	is odd, so I make $3n+1$:	16
16	is even, so I take half:	8
8	is even, so I take half:	4
4	is even, so I take half:	2
2	is even, so I take half:	1

As you can see from this example, the numbers go up and down, but eventually—at least for all numbers that have ever been tried—comes down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the winds over and over again before they finally descend to the ground. Because of this analogy, this sequence of numbers is usually called the **Hailstone sequence,** although it goes by many other names as well.

Write a `ConsoleProgram` that reads in a number from the user and then displays the Hailstone sequence for that number, just as in Hofstadter's book, followed by a line showing the number of steps taken to reach 1. For example, your program should be able to produce a sample run that looks like this:
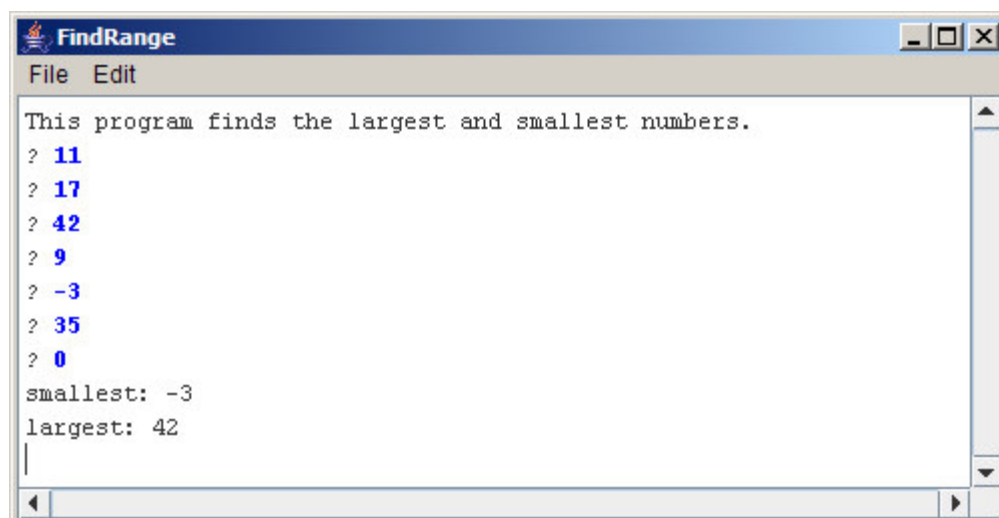
```
Hailstone                                        _ □ X
File  Edit
Enter a number: 17
17 is odd, so I make 3n + 1: 52
52 is even so I take half: 26
26 is even so I take half: 13
13 is odd, so I make 3n + 1: 40
40 is even so I take half: 20
20 is even so I take half: 10
10 is even so I take half: 5
5 is odd, so I make 3n + 1: 16
16 is even so I take half: 8
8 is even so I take half: 4
4 is even so I take half: 2
2 is even so I take half: 1
The process took 12 to reach 1
```

The fascinating thing about this problem is that no one has yet been able to prove that it always stops. The number of steps in the process can certainly get very large. How many steps, for example, does your program take when n is 27?

3. Write a **ConsoleProgram** that reads in a list of integers, one per line, until a sentinel value of 0 (which you should be able to change easily to some other value). When the sentinel is read, your program should display the smallest and largest values in the list, as illustrated in this sample run:

```
FindRange                                        _ □ X
File  Edit
This program finds the largest and smallest numbers.
? 11
? 17
? 42
? 9
? -3
? 35
? 0
smallest: -3
largest: 42
```

Your program should handle the following special cases:

- If the user enters only one value before the sentinel, the program should report that value as both the largest and smallest.

- If the user enters the sentinel on the very first input line, then no values have been entered, and your program should display a message to that effect.

4. Suppose that you've been hired to produce a program that draws an image of an archery target—or, if you prefer commercial applications, a logo for a national department store chain—that looks like this:
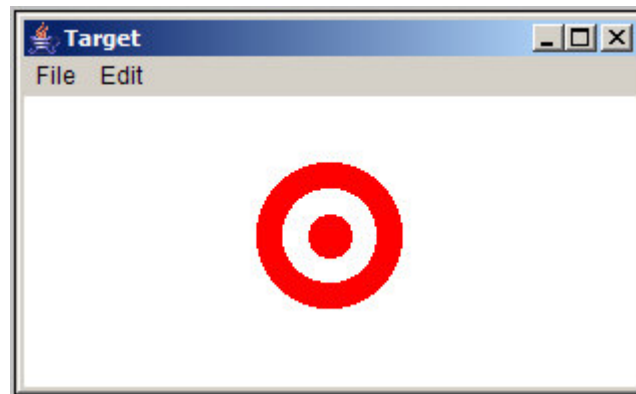

This figure is simply three **GOval** objects, two red and one white, drawn in the correct order. The outer circle should have a radius of one inch (72 pixels), the white circle has a radius of 0.65 inches, and the inner red circle has a radius of 0.3 inches. The figure should be **centered** in the window of a **GraphicsProgram** subclass.

5. Write a **GraphicsProgram** subclass that draws a partial diagram of the **acm.program** class hierarchy, as follows:
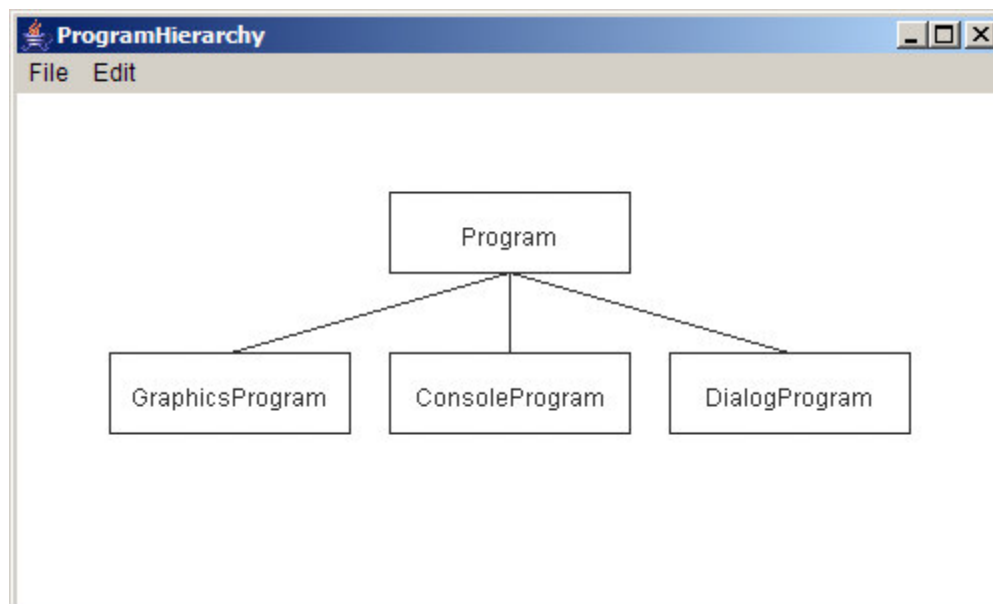
The only classes you need to create this picture are **GRect**, **GLabel**, and **GLine**. The major part of the problem is specifying the coordinates so that the different elements of the picture are aligned properly. The aspects of the alignment for which you are responsible are:

- The **width** and **height** of the class boxes should be specified as named constants so that they are easy to change. You should determine reasonable values for these constants to make your picture look similar (but, it need not be exact) to the figure above.

- The labels should be **centered** in their boxes. You can find the width of a label by calling **label.getWidth()** and the height it extends above the baseline by calling **label.getAscent()**. If you want to center a label, you need to shift its origin by half of these distances in each direction.

- The connecting lines should start and end at the center of the appropriate edge of the box.

- The entire figure should be **centered** in the window.

6. Write a **GraphicsProgram** subclass that draws a pyramid consisting of bricks arranged in horizontal rows, so that the number of bricks in each row decreases by one as you move up the pyramid, as shown in the following sample run:
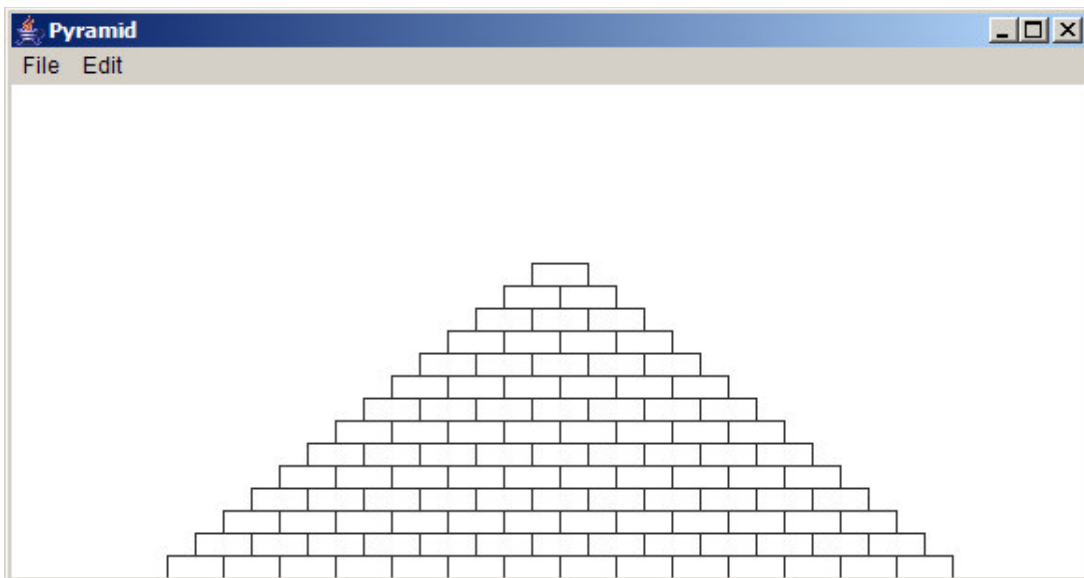
The pyramid should be **<u>centered</u>** at the bottom of the window and should use constants for the following parameters:

BRICK_WIDTH	The width of each brick (30 pixels)
BRICK_HEIGHT	The height of each brick (12 pixels)
BRICKS_IN_BASE	The number of bricks in the base (14)

The numbers in parentheses show the values for this diagram, but you must be able to change those values in your program.

Advice, Tips, and Tricks

Many of the programs you'll be writing need to work for a variety of user inputs. For example, the Pythagorean Theorem program should be able to handle any positive real numbers as inputs, and the Hailstone sequence should work for any positive integer the user enters. As with the Karel assignment, please be sure to test your programs extensively before submitting them. It would be a shame if your section leader dropped you from a ✓+ to a ✓ because you had forgotten to test your program on some particular input.

Some of the other programs that you'll be writing need to reference constants defined in your program. One of the major points of defining constants in a program is to make it easier to change your program's behavior simply by adjusting the value assigned to that constant. During testing, we will run your programs with a variety of different constants to check whether you have correctly and consistently used constants throughout your program. Before submitting, check whether or not your programs behave correctly when you vary the values of the constants in the program.

Style is just as important as ever in this assignment. Be sure to follow the style guidelines set out from the Karel assignment – use a top-down design, comment your code as you go, have intelligent method names, and indent your code properly. In addition to this, now that we've added variables, methods, and constants into the mix, you should check your code for the following stylistic issues:

• **Do you have clear names for your variables?** In Java, variables should be written in lowerCamelCase and should clearly describe what values they represent. Avoid single-letter variable names except in **for** loops or when the single letter really should be the name of the variable. Try to be descriptive about what sort of value will be stored in the variable.

• **Do you make appropriate use of methods?** In many of these programs you will end up writing similar code multiple times. Whenever possible, factor this similar code out into a method. This makes the code easier to read, easier to maintain, easier to test, and easier to debug.

• **Do you have appropriate inline comments?** Method comments are a great way to make your intentions easier to follow, but it is also important to comment the bodies of your methods as well. Use comments to indicate what task different pieces of the code are trying to perform, or to clarify logic that is not immediately obvious.

• **Do you make appropriate use of constants?** Several of the programs you'll write – especially the graphics programs – will require values that will not be immediately evident from context. When appropriate, introduce constants into your program to make the program more customizable and easier to read.

• **Did you update the file comments appropriately?** Java files should begin with a comment describing who wrote the file and what the file contains. Did you update the comments at the top of each Java file with information about your program?

This is not an exhaustive list of stylistic conventions, but it should help you get started. As always, if you have any questions on what constitutes good style, feel free to stop on by the Tresidder LaIR with questions, come visit us during office hours, or email your section leader with questions.

Good luck!