Assignment 2: Welcome to Java!

Based on a handout by Eric Roberts and Mehran Sahami

Having helped Karel the Robot through the challenges of Assignment 1, you are now prepared to harness the power of the Java programming language to solve a variety of problems. In this assignment, you will work through seven small programs, each giving you a feel for how to use variables, methods, control structures, the **`acm.graphics`** package, and other features of Java not present in Karel's world. By the time you're done, you'll be ready to start building larger, more elaborate programs in Java.

The starter code for this assignment is available on the CS106A website under the “Assignments” tab.

Due Wednesday, January 30 at 3:15PM

Part One: The Pythagorean Theorem

One of the oldest results in mathematics is the *Pythagorean theorem*, which relates the lengths of the three sides of a right triangle by the simple equality

$$a^2 + b^2 = c^2$$

If a and b are known, then the value of c can be determined as

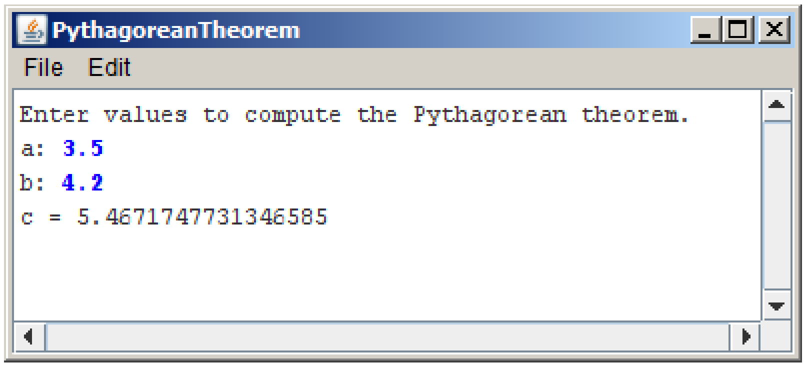
$$c = \sqrt{a^2 + b^2}$$

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`**.

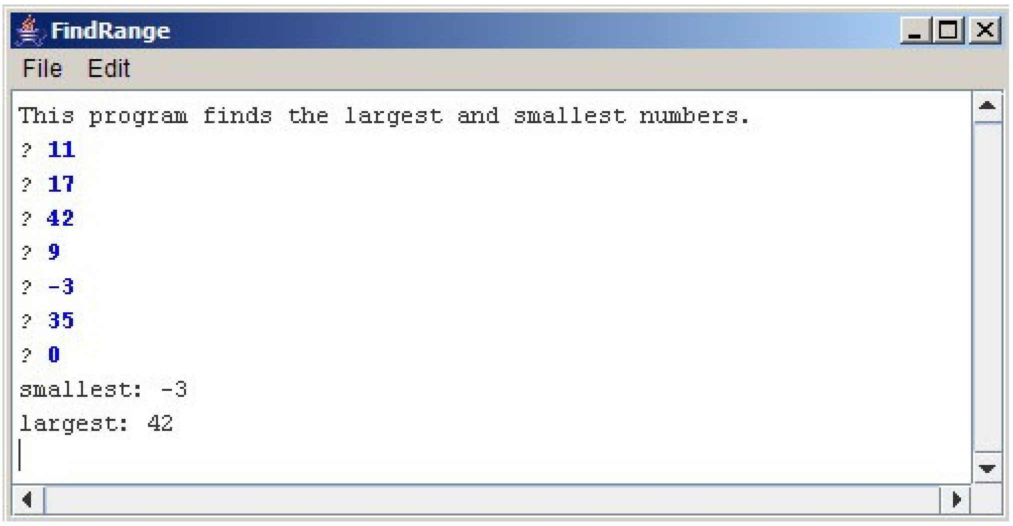
Write a **`ConsoleProgram`** that accepts values for **`a`** and **`b`** as **`double`**s (you can assume that **`a`** and **`b`** will be positive) and then calculates the solution of **`c`** as a **`double`**. Your program should be able to duplicate the following sample run, plus runs with other values:



```
Enter values to compute the Pythagorean theorem.
a: 3.5
b: 4.2
c = 5.4671747731346585
```

Part Two: Find Range

Write a **`ConsoleProgram`** that reads in a list of integers, one per line, until the user enters 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:



```
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.

Part Three: Hailstone Sequence

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:

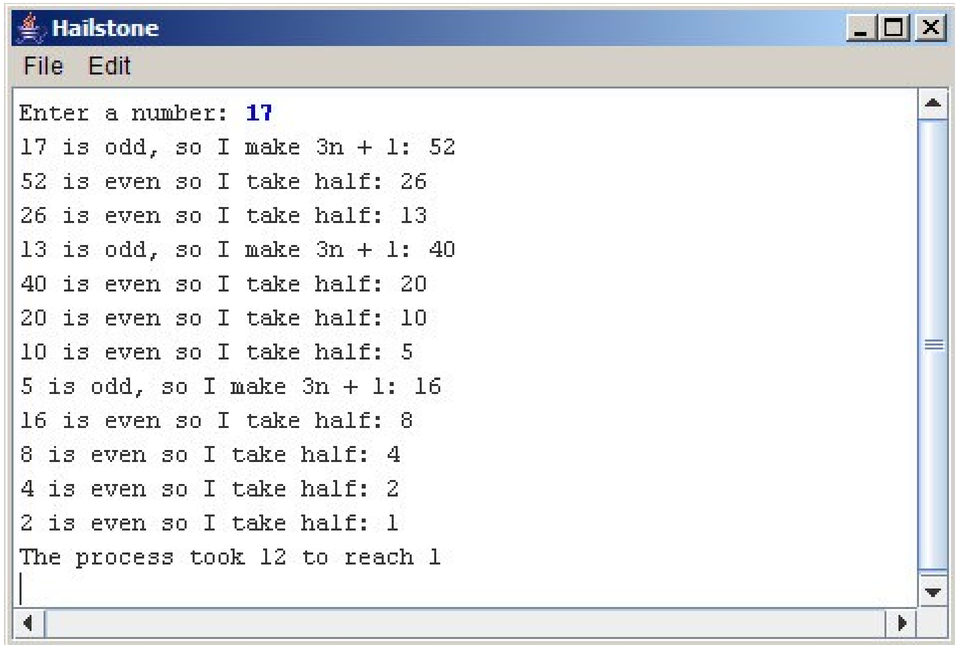
Pick some positive integer and call it *n*.
If *n* is even, divide it by two.
If *n* is odd, multiply it by three and add one.
Continue this process until *n* is equal to one.

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

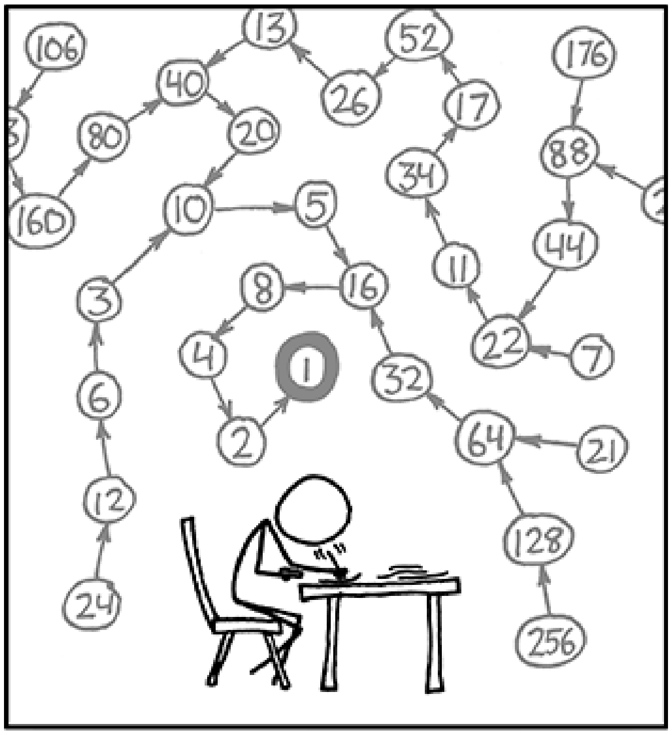
As you can see from this example, the number goes up and down, but eventually—at least for all numbers that have ever been tried—comes down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the winds over and over again before they finally descend to the ground. Because of this analogy, this sequence of numbers is sometimes 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:



```
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 *Collatz conjecture* is that this process always eventually reaches 1. Although the hailstone sequence terminates for all numbers anyone has even tried, no one has yet proven or disproven the Collatz conjecture. 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?



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

(xkcd)

Part Four: Artistry!

Now that you have the **`acm.graphics`** package available to you, you have the ability to turn the computer into a digital canvas. To warm up with the graphics package, why not take a few minutes to draw a pretty picture?

In this part of the assignment, your job is to draw a picture of your choice using the **`acm.graphics`** package. The requirements for this part of the assignment are fairly lax (they're described shortly), so feel free to use this as an opportunity to play around with Java and see what you can make with the tools that are now available to you!

For this assignment, you should draw a picture with the following properties:

1. Your picture must use <u>at least</u> three different types of **`GObject`**s – for example, you could use **`GLine`**, **`GRect`**, and **`GOval`**.

2. Your picture must have <u>at least</u> one filled object.

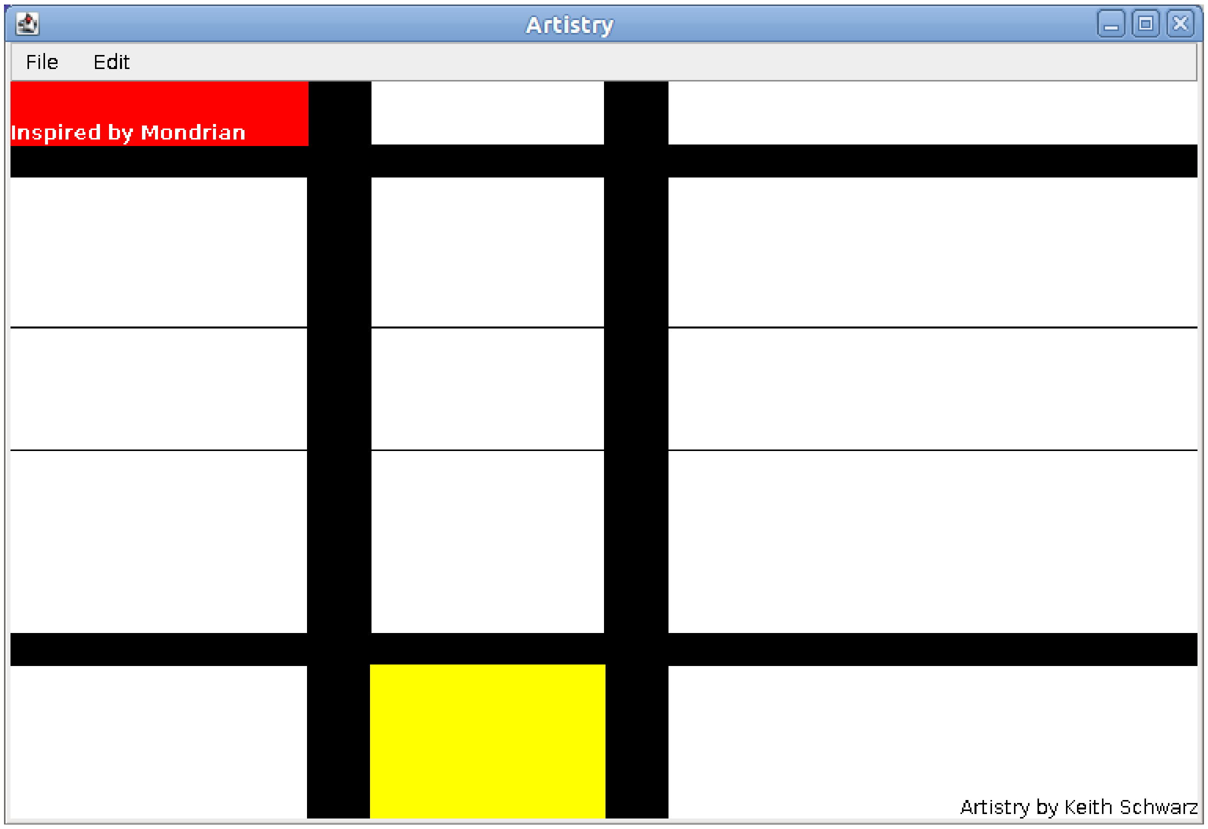
3. Your picture must have <u>at least</u> least two different colors of objects.

4. You must sign your name in the bottom-right corner. To do this, create a **`GLabel`** with the phrase "Artistry by ***name***," where ***name*** is your name, and align it so that it is flush up against the bottom-right corner of the window. Be sure that all the text is visible and that none of the letters in the **`GLabel`** are cut off. (This **`GLabel`** doesn't count as one of the three different types of **`GObject`**s that you're required to have. If you want to count **`GLabel`** as one of the **`GObject`** types you're using, you'll need to have a second **`GLabel`** in your picture).

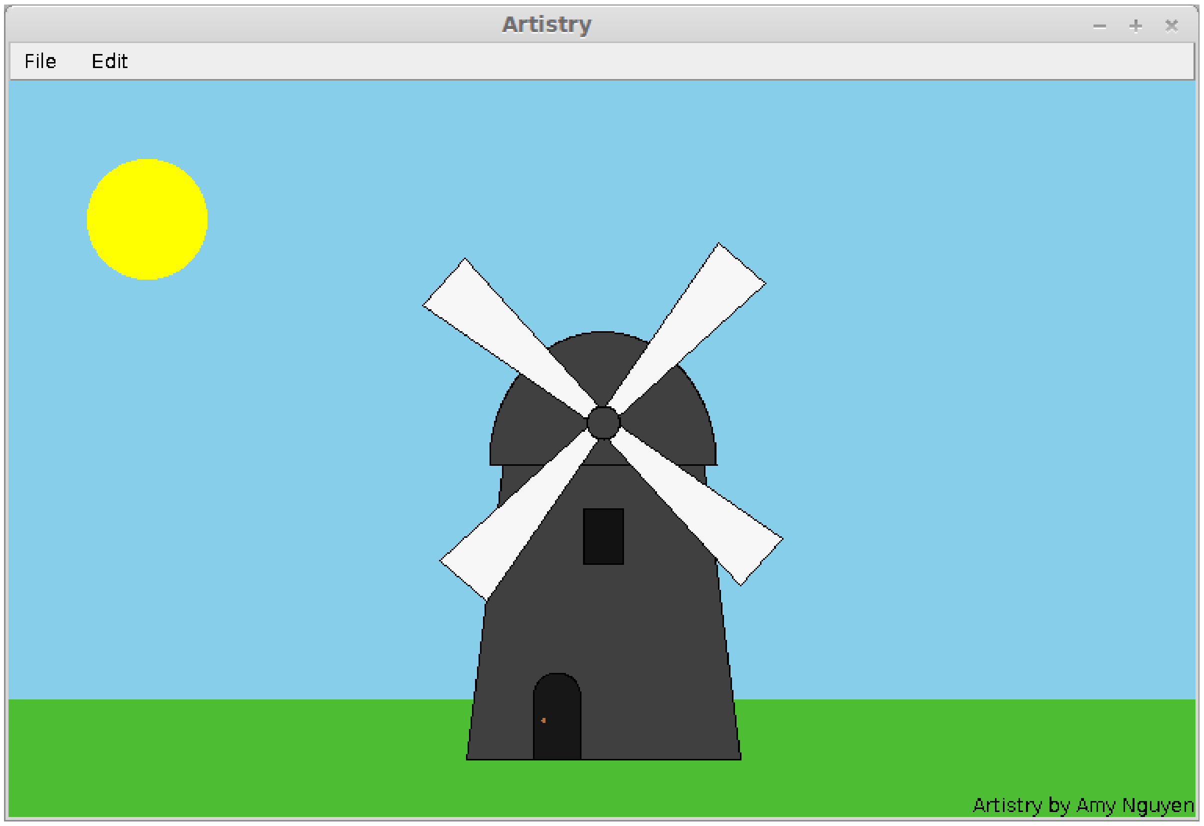
Although you're free to draw whatever you'd like, be sure to write clean and elegant code as you would in the other parts of this assignment. Have a nice decomposition, comment your code, use helper methods where appropriate, and prefer constants to magic numbers.

There are many other graphics objects besides **`GRect`**, **`GOval`**, **`GLabel`**, and **`GLine`** and you are free to use them in your program. Chapter 9 of the book gives a good overview of what else is out there. You may find the **`GArc`** and **`GPolygon`** classes particularly useful. Chapter 9 also describes the **`Color`** class in more detail, so if you're interested in colors beyond the standard set I would suggest giving it a read.

To get your imaginations going, here are some sample pictures from the course staff:



`GLabel`, **`GRect`**, **`GLine`**

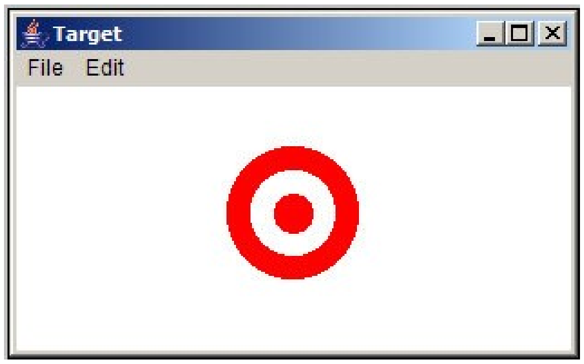


`GPolygon`, **`GRect`**, **`GOval`**, **`GPolygon`**, **`GArc`**, **`GLine`**

We recommend waiting until Wednesday's lecture to attempt these next problems. If you want to start earlier, we recommend reading about methods in Chapter Five of The Art and Science of Java.

Part Five: Target

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 **<u>centered</u>** in the window of a **`GraphicsProgram`** subclass.

Part Six: Fixing Broken Java

As you begin writing larger and larger programs, you will invariably end up making mistakes in your code. Don't worry – this is a normal part of programming! To help you navigate the error messages you'll get in Eclipse when your code isn't working correctly, as well as to give you practice debugging an incorrect program, this part of the assignment asks you to fix a buggy Java program so that it works correctly.

The **`FixingBrokenJava.java`** file contains a link to the course website, where you can download the source file for this part of the assignment.* This program has serious errors – it doesn't compile, and even if it did, it contains logic errors that prevent it from working correctly.

The program we've provided you reads in a positive integer from the user and then checks whether that number is *prime*; that is, whether it has any divisors other than 1 and itself. For example, 3 is prime and 31 is prime, but 54 is not (it's divisible by two) and 49 is not (it's divisible by seven). Unfortunately, our provided program has many problems: it doesn't compile due to syntax errors, and the logic itself is incorrect. Your task is to

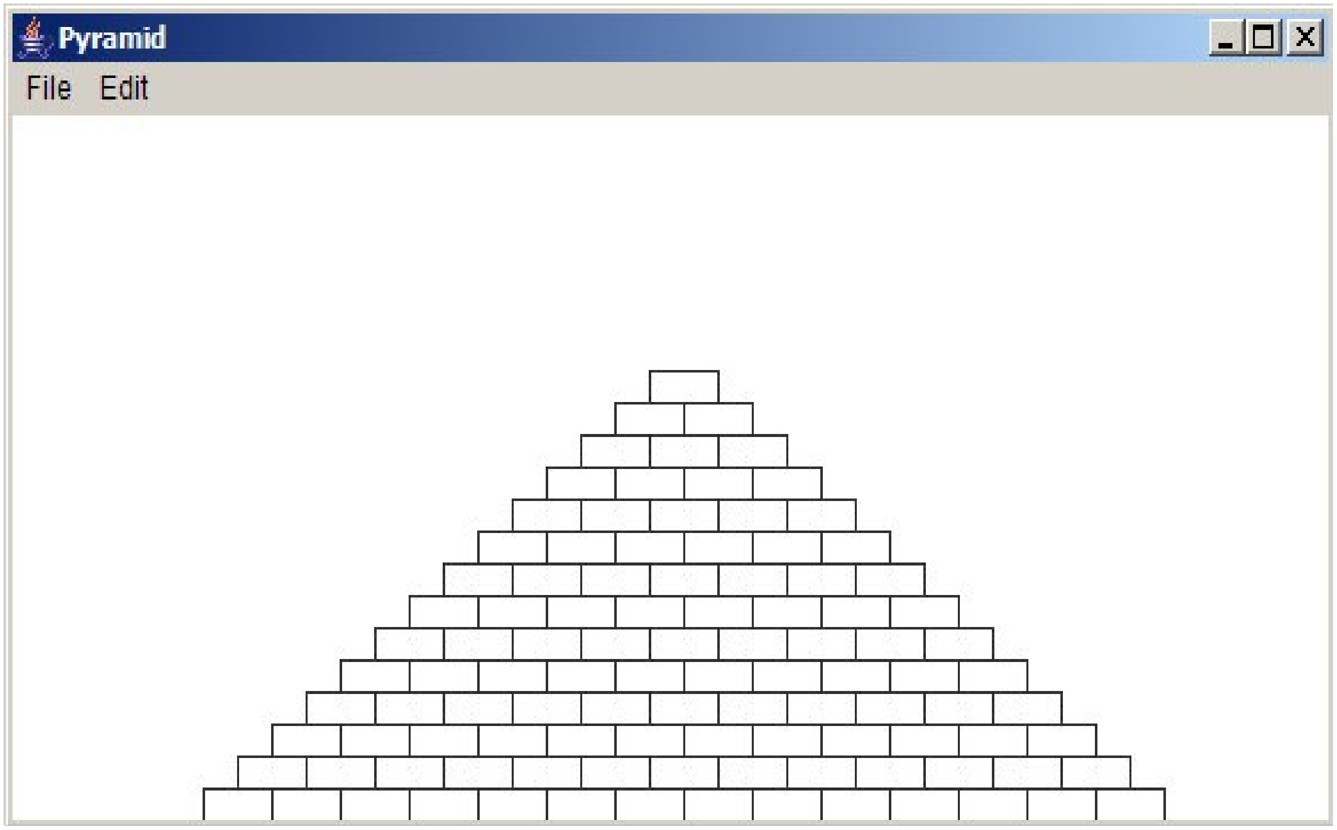
- Fix the program so that it compiles, and
- Correct the logic errors so that it works correctly.

To reiterate – <u>it is not enough to just get the program to compile</u>! You also need to find and correct any logic errors you encounter. Be sure to test your final program to make sure that it works correctly.

* We decided to put this file online rather than include it with the assignment, since the starter code has errors and we didn't want Eclipse to keep pestering you about our mistakes.

Part Seven: Pyramid

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 **centered** 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!