Stanford University, CS 106A, Homework Assignment 2
Simple Java (Graphics and Console Programs)

Based on past assignments by Keith Schwarz, Eric Roberts, Mehran Sahami, Marty Stepp, and others. Modified by Alisha Adam and Rohit Talreja.

Due Date: Monday, July 4th at 5:00pm

This assignment practices standard Java programming such as console interaction, variables, constants, control statements, parameters, and graphics. As with all assignments, there is a **starter project ZIP archive** including all of these problems on our web site in the area for Homework 2. You will **turn in** the following files:

QuadraticEquation.java, **ExamScores.java**, **Hailstone.java**, **Pyramid.java**, and **Target.java**

The ZIP archive contains other files and libraries; you should not modify these. When grading/testing your code, we will run your **.java** code with our own original versions of the support files, so your code must work with them.

Problem 1: `QuadraticEquation`

Write an interactive `ConsoleProgram` named `QuadraticEquation` that finds real roots of a quadratic equation. A *quadratic equation* is a mathematical equation of the form $a\,x^2 + b\,x + c = 0$, where a is nonzero. Given the values of a, b, and c, the quadratic formula says that the roots of the quadratic equation are given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a}$$

The quantity (b^2 - 4 a c) is called the *discriminant*. If it's greater than zero, there are two different real roots of the quadratic equation, which are given by the above formula. If it's exactly zero, there's just one root, given in two different ways by the quadratic formula. If it's negative, there are no real roots to the equation.

Your job is to write a program that prompts the user for the values of a, b, and c, then prints out the roots of the corresponding quadratic equation. Your program must *exactly* duplicate the output of the following sample runs below, plus runs with other values. *(User input as read from a call to* `readInt` *is shown in blue bold font below.)*

```
CS 106A Quadratic Solver!
Enter a: 1
Enter b: -3
Enter c: -4
Two roots: 4.0 and -1.0
```

```
CS 106A Quadratic Solver!
Enter a: 1
Enter b: 6
Enter c: 9
One root: -3.0
```

```
CS 106A Quadratic Solver!
Enter a: 2
Enter b: 4
Enter c: 6
No real roots
```

expected output from three separate runs of the program (user input in blue)

For reference, to compute the square root of some number x, you can use the `Math.sqrt` method. For example, the following code sets y to the square root of x:

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

You may assume that the user doesn't enter 0 as their value for a. Aside from the above restriction, the values of a, b, and c can be any integers. You should not do any rounding of real numbers in your output.

Problem 2: ExamScores

Write an interactive `ConsoleProgram` named `ExamScores` that displays information about exam scores. The program should repeatedly prompt the user to enter exam scores until a particular "sentinel" value is entered. By default this sentinel value is `-1`. You may assume that the user will type positive integers other than the sentinel itself. Once the sentinel value is entered, the program should display the maximum and minimum exam score typed, as well as the average score and the number of scores that failed the exam. "Failing" the exam in this context is receiving a score of 59 or lower. Your program should *exactly* duplicate the output of the following sample run (user input is shown in blue), plus be able to run properly with other values:

```
CS 106A Exam Master
Next exam score (or -1 to quit)? 68
Next exam score (or -1 to quit)? 94
Next exam score (or -1 to quit)? 76
Next exam score (or -1 to quit)? 45
Next exam score (or -1 to quit)? 89
Next exam score (or -1 to quit)? 54
Next exam score (or -1 to quit)? 73
Next exam score (or -1 to quit)? -1
Highest score = 94
Lowest score = 45
Average = 71.28571428571429
2 student(s) failed the exam.
```

This program should be written to use a **constant** to represent the sentinel value of -1. If we change only that constant's value, your program should now use the new sentinel value throughout the code and output. For example, if the constant's value is changed to -42, the program's output would look like:

```
CS 106A Exam Master
Next exam score (or -42 to quit)? 76
Next exam score (or -42 to quit)? 89
Next exam score (or -42 to quit)? 83
Next exam score (or -42 to quit)? -42
Highest score = 89
Lowest score = 76
Average = 82.66666666666667
0 student(s) failed the exam.
```

If only one score is entered, that score is the maximum, minimum, and average. For example:

```
CS 106A Exam Master
Next exam score (or -1 to quit)? 83
Next exam score (or -1 to quit)? -1
Highest score = 83
Lowest score = 83
Average = 83.0
0 student(s) failed the exam.
```

If no scores are entered, the program should instead print the following message saying that no scores were entered:

```
CS 106A Exam Master
Next exam score (or -1 to quit)? -1
No scores were entered.
```

Problem 3: 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 of *The Art and Science of Java*. The problem can be expressed as follows:

> Pick some positive integer and call it *n*.
> If *n* is even, divide it by 2.
> If *n* is odd, multiply it by three and add one.
> Continue this process until *n* is equal to one.

Hofstadter illustrates this process in his book 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
```

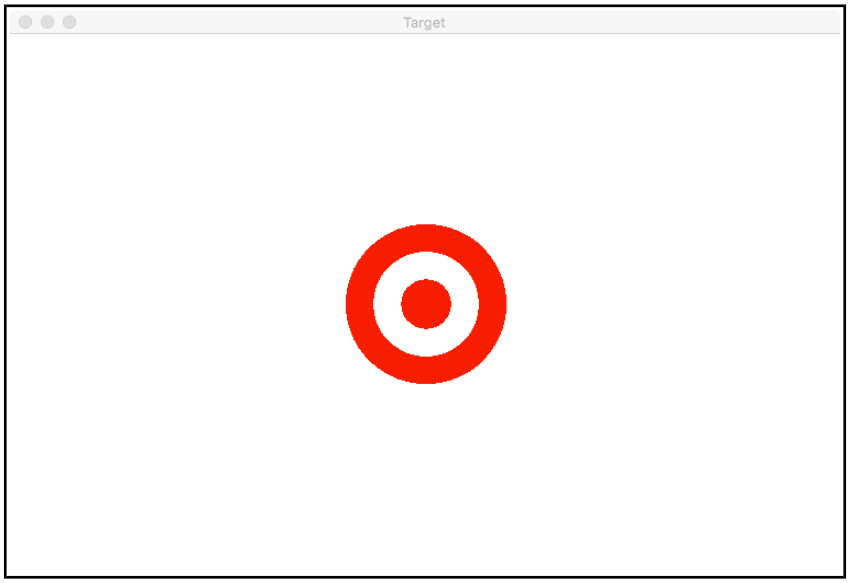
As you can see from this example, the numbers go up and down, but eventually – at least for all numbers that have ever been tried – come down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the winds over and over again before they 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. The fascinating thing about this problem is that no one has yet been able to prove that it always stops. The number of steps can certainly get very large.

Write an interactive `ConsoleProgram` named `Hailstone` 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 the number of steps taken to reach 1. A sample run of the program may look 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 steps to reach 1
```

Problem 4: Target

Write a `GraphicsProgram` named `Target` that produces an image of an archery target – or, if you prefer commercial applications, a logo for a national department store – that looks like this:

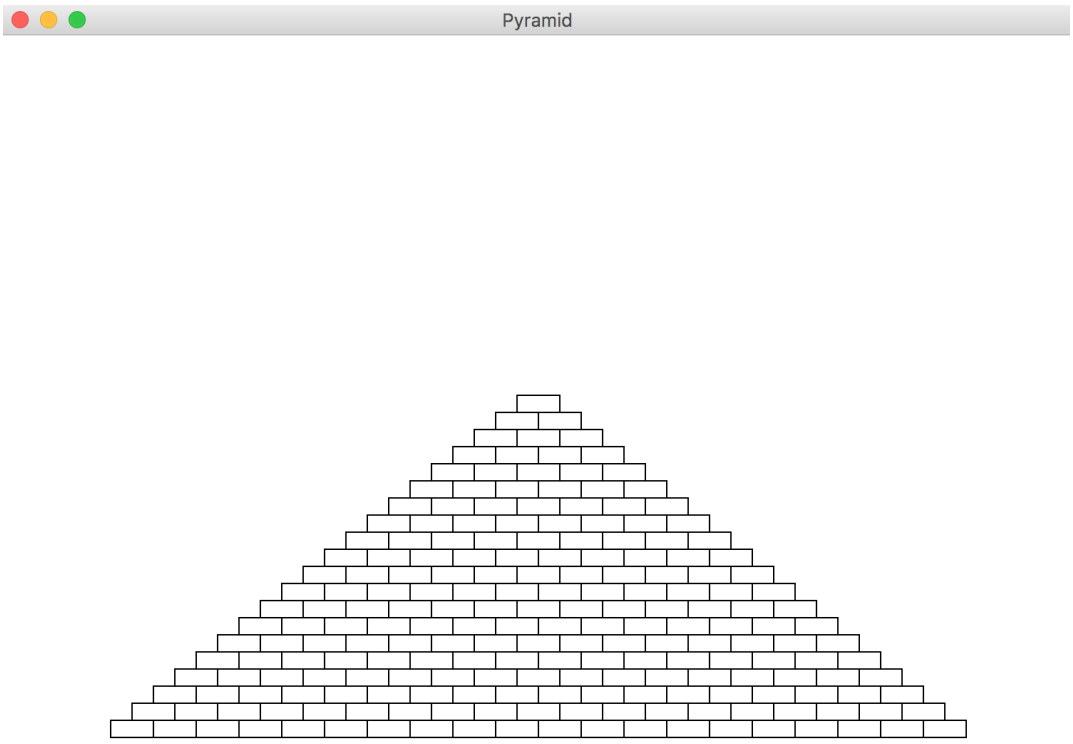


This figure is simply three `GOval` objects, two red and one white, drawn in the correct order. By default, the outer circle has a radius of one inch (72 pixels), the white circle has a radius of 0.65 inches, and the inner circle has a radius of 0.3 inches. The figure should be **centered** in the window.

We should be able to create a target of any size by changing a single **constant** in your code. When we change this value, all three circles should be scaled proportionately, and the figure should still be centered in the window.

Problem 5: Pyramid

Write a `GraphicsProgram` that draws a pyramid consisting of bricks arranged in horizontal rows, so 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 of the pyramid (14)

The numbers in parentheses show the values for this diagram, but you must be able to change the values of these constants in your program and get the correct result.