CS 100: Programming Assignment P2

Due: Thursday, February 11. (Either in lecture or in Carpenter by 10pm.)

You may work in pairs. Do not submit your assignment for grading unless you have read and understand the CS100 webpage on Academic Integrity. Follow the course rules for the submission of assignments.

Introduction

The purpose of the assignment is for you to deepen your understanding of (1) iteration (the for-loop and the while-loop), (2) various methods in the Math class (Math.sqrt, Math.cos, Math.sin, Math.exp, etc), (3) various graphics methods (drawOval, drawRect, drawLine, etc.), (4) elementary I/O, (5) types (double, int, long), and (6) the assignment statement.

Part A. Approximating n! (3 correctness + 3 style)

If n is a nonnegative integer, then n factorial is given by $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$. Thus,

$$\begin{aligned} 1! &= 1 \\ 2! &= 2 \\ 3! &= 6 \\ 4! &= 24 \\ 5! &= 120 \end{aligned}$$

We adopt the convention that 0! = 1. Note that n! is n times the "previous" factorial, i.e., $n! = n * (n-1)!$. Stirling's approximation to n! is given by

$$s_n = \sqrt{\frac{2\pi}{n}}\left(\frac{n}{e}\right)^n$$

where e = 2.718… is the natural logarithm base, i.e., Math.exp(1.0).

Write a program that prints a table with 20 lines. On the nth line display n, n!, the absolute error $|s_n - n!|$, and the relative error $|s_n - n!|/n!$. Follow these guidelines:

- Start by creating a project using the CUCSApplication stationary and CUCSApplication.java as a template.
- Use a for-loop to generate the table and exploit the update formula $n! = n * (n-1)!$.
- Compute s_n with Math.PI, Math.sqrt, Math.exp, and Math.pow. Refer to page 770 of the text.
- n! gets very big as n climbs to 20 and you should use a variable of type long to house its value.
- Math.abs can be used to get the errors.
- Use formatted print statements with 10.3e formats for both the absolute and relative errors.
- The table should have nicely aligned columns with appropriate headings.

Part B. Diamonds and Squares (4 correctness + 4 style)

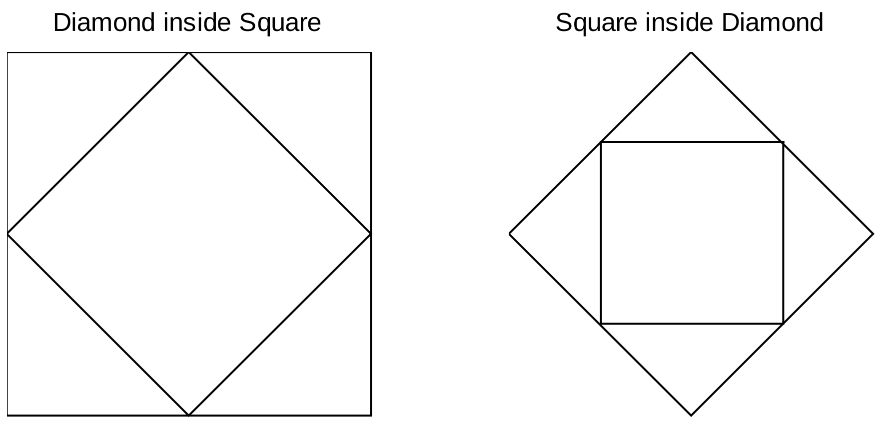
A *square* with radius r and center (x_c, y_c) has vertices

$$(x_c - r, y_c + r),\ (x_c + r, y_c + r),\ (x_c + r, y_c - r), \text{ and } (x_c - r, y_c - r).$$

A *diamond* with radius r and center $(x_c\,, y_c\,)$ has vertices

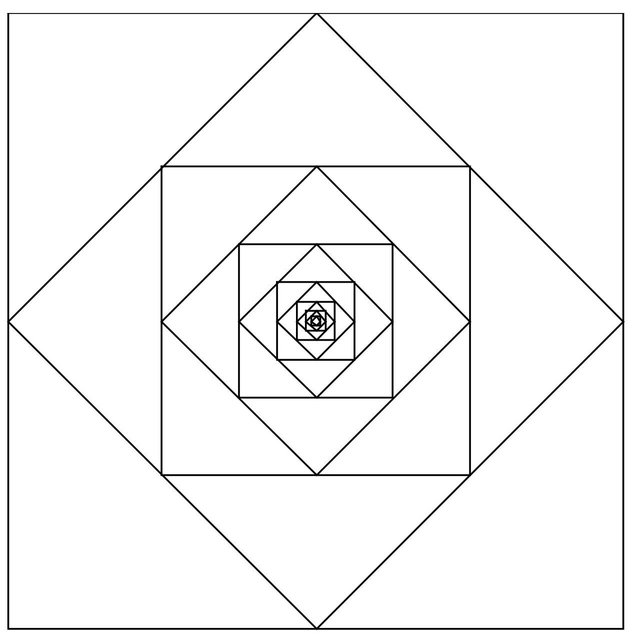
$$(x_c\,, y_c + r),\ (x_c + r\,, y_c\,),\ (x_c\,, y_c - r),\ \text{and}\ (x_c - r\,, y_c\,).$$

We can inscribe a diamond inside a square and vice versa:



Note that in the left situation the diamond and square have the same radius while in the right situation the square has half the radius of the diamond.

Imagine a process that draws an initial square, followed by an inscribed diamond, followed by an inscribed square, followed by an inscribed diamond, and so on *ad infinitum*:



Notice that the radius is halved after every diamond is drawn.

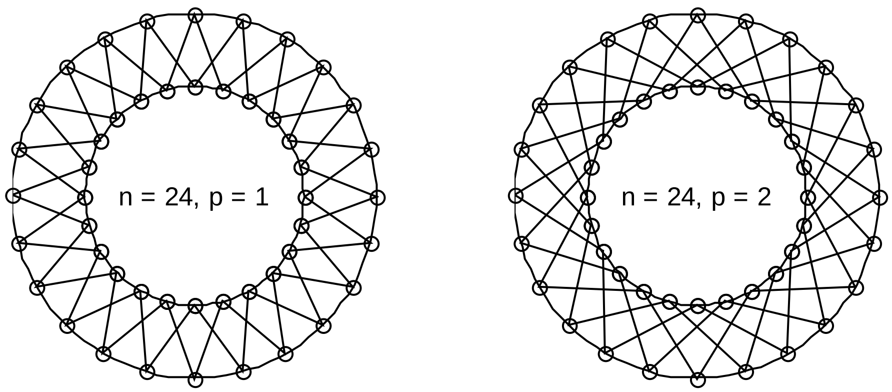
Write a program that does this repeated drawing except that it stops the process as soon as the radius drops below a given finite positive amount. Follow these guidelines:

- Work with the template provided by CUCSGraphicsApplication stationary. Resize and title the graph window as appropriate.
- The outer square should have radius 250 and center (500,300). Encapsulate these values as int constants.
- Use a while-loop to control the iteration.
- Each pass through the loop should draw either a square or a diamond. You'll have to use an if-else construct to decide which.
- Have an int variable r that is initialized to 250 and is halved every time a diamond is drawn. Use integer division for this.
- The loop should terminate after r is strictly less than the value of rMin where rMin is an int constant with value 5.

As a warm-up exercise, you might want to write a program that draws the Diamond-in-Square and Square-in-Diamond figures above.

Part C. Hubcaps (3 correctness + 3 style)

Here are some "hubcap" designs:



(The little circles are "rivet points" and are not part of the design.) The plotting of these designs involves six parameters:

- x_c is the x-coordinate of the center.
- y_c is the y-coordinate of the center.
- r_1 is the radius of the inner circle.
- r_2 is the radius of the outer circle.
- n is the number of spoke pairs.
- p is the spoke shift factor.

The kth spoke pair consists of a "left spoke" that connects

$$\left(x_c + r_2\cos(k\theta),\, y_c + r_2\sin(k\theta)\right) \text{ and } \left(x_c + r_1\cos((k-p)\theta),\, y_c + r_1\sin((k-p)\theta)\right)$$

and a "right spoke" that connects

$$\left(x_c + r_2\cos(k\theta),\, y_c + r_2\sin(k\theta)\right) \text{ and } \left(x_c + r_1\cos((k+p)\theta),\, y_c + r_1\sin((k+p)\theta)\right).$$

Write a program that draws a hubcap defined by $x_c = 500$, $y_c = 300$, $r_1 = 150$, $r_2 = 250$, $n = 60$, and $p = 2$. Follow these guidelines:

- Work with the template provided by the CUCSGraphicsApplication stationary. Resize and title the graph window as appropriate.
- Encapsulate the six design parameters as int constants.
- The drawing of the spokes is an iteration. Use a for-loop with the property that a spoke pair is drawn each pass through the loop.

Remember that screen coordinates are integers and that the values passed to the graph methods must have type int. Here is what should be produced: