Assignment 0414
We switch from arithmetic to geometry as we finish our tour of mathematical computations with a non-deterministic technique that you might not have seen before, and an extra-credit exercise that explores a more conventional, deterministic equivalent.

Outcomes
This assignment will affect your proficiency measures for outcomes 1a–1c, 2a–2c, and 3a–3f.

Not for Submission
You may want to read up on $\pi$, integration, and Monte Carlo techniques to gain additional insight into this assignment’s concepts.

For Submission 1: $\pi$ Estimator
Write a Java program, PiEstimator, for estimating the value of $\pi$ using the basic approach and control structure discussed in class. Specifically, your program should: (1) “throw” lots of random darts at a unit circle inscribed within a square, centered on the origin; then (2) determine the percentage of darts that fell within the circle; finally, (3) calculate an estimate for $\pi$ using the ratio of the unit circle’s area to that of the inscribing square (whose area is 4). If present, $\text{args}[0]$ will specify the number of darts to be thrown, defaulting to 10,000 if the argument is not there.

PiEstimator’s output should begin with the single word start on its own line (you’ll see why later). Then, as the program starts to throw darts at the circle/square, it should print the $x$ and $y$ coordinates of each dart, along with in or out to indicate whether the dart landed within the unit circle.

Once the requisite number of darts has been thrown, the program should print the single word end on its own line. Finally, it should print the number of darts thrown, the number that landed within the circle (“hits”), and the program’s estimate for $\pi$.

The following example shows the beginning and end this output (with the intervening 9,996 darts elided for brevity!):

```
$ java PiEstimator
start
-0.0992410926682088 -0.8366737391455505 in
-0.7906731244069747 -0.897140635926081 out

...9,996 more darts...

-0.9191750657381172 -0.1582362506844901 in
-0.28920571668046136 -0.9655439024563417 out
end
Darts: 10000 Hits: 7786 Pi estimate: 3.1144
```

A visualizer program is available so that you can see the darts as they are “thrown.”

For Submission 2: Monte Carlo Integrator
Write a Java program, MonteCarloIntegrator, that estimates the area under a curve using the randomized technique discussed in class. At a minimum, the program must be able to handle polynomials of arbitrary degree. If the program is invoked without arguments, a (by now) familiar “usage” message should be displayed which details how to specify the function to integrate and over what range. The format should be as follows:

- The first argument is a name that uniquely identifies the type of function to use.
- Zero or more arguments after the function name shall supply any additional information needed by the function (and they thus depend on the function’s type).
- The two arguments after the function information indicate the range over which the area under the function’s curve is to be estimated. The range may be given with either the upper or lower bound first.
- An optional final argument, which must start with total=, specifies the number of darts to be thrown, defaulting to 10,000 if the argument is not there.

Note that the processing of the arguments alone can be somewhat tricky. The recommended ap-
approach is to start with the last argument. If it indicates a custom number of darts, process that then assume that the preceding two arguments represent the function bounds. Then, return to the first argument to determine the function type, and process the arguments in between as supplementary information for that function.

For the single required function type of polynomials of arbitrary degree, use the name poly as the first argument. Succeeding arguments, up to the upper and lower bound of the definite integral, shall provide the coefficient for each term in the polynomial in descending power. Thus, the last argument before the integral bounds is the constant term in the polynomial.

**Polynomial Argument Examples**

The following examples illustrate specific instances of the given argument format:

```
poly 2 0 -1 9 5.2 1 3
```

...estimates the area of \( f(x) = 2x^4 - x^2 + 9x + 5.2 \) for \( x = 1 \) to 3 using the default 10,000 darts.

```
poly 100 30.75 -20.5 total=100
```

...estimates the area of \( f(x) = 100 \) (yes, that is just a horizontal line) for \( x = -20.5 \) to 30.75 using only 100 darts.

```
poly 9 8 7 6 0.5 1.0 total=20000
```

...estimates the area of \( f(x) = 9x^3 + 8x^2 + 7x + 6 \) for \( x = 0.5 \) to 1 using 20,000 darts.

**Other Functions**

If you have additional time, you may implement other types of functions. Some ideas:

```
sin 4 2 1 5 1 -2
```

...estimates the area of \( f(x) = 4 \sin (2x + 1) + 5 \) for \( x = -2 \) to 1 using the default 10,000 darts. Note how, in this format, the first argument after sin is a coefficient to attach; the last argument before the bounds is a constant to add; and the arguments in between are interpreted as polynomial coefficients, similar to poly.

```
ln 1 1 0 0.5 1 10 total=5000
```

...estimates the area of \( f(x) = \ln x + 0.5 \) for \( x = 1 \) to 10 using 5,000 darts. Note the similar pattern of use for the arguments—this consistency helps not only the user but you the developer, allowing you to leverage similar or shared code for handling them.

**Monte Carlo Integrator Output**

The Monte Carlo integrator shall start with an initial loop that estimates the minimum and maximum \( y \) values of the given function in the given domain. To aid in debugging, the program should display these values as well as the final range (a little margin, say 10%, helps when visualizing the thrown “darts”). Then, it should display start just like the \( \pi \) estimator, subsequently showing dart positions and statuses until the requested number of darts has been thrown. The format for this portion matches the one for the \( \pi \) estimator.

When all of the darts have been thrown, the program should print end on its own line, concluding with the estimated area under the curve.

Here is some sample elided output for estimating the area under \( f(x) = x^2 + x + 1 \) for the domain from \(-5\) to 5:

```
$ java MonteCarloIntegrator poly 1 1 1 -5 5
Estimating maximum and minimum...
Minimum in range: 0.75
Maximum in range: 30.999999999557353
Final lower bound: -3.0999999999557355
Final upper bound: 34.09999999951309
start
3.268551266538889 11.685628627658286 in
...9,996 more darts...
-2.307828133029284 30.65425361841106 out
-0.5641696231092643 17.37460157903887 out
end
Estimate: 92.18159999868374
```

**Using the Dartboard Program**

For “visual” debugging and a touch of fun, a Dartboard program has been provided to work alongside these random estimation programs. This program accepts the precise output of both PiEstimator and MonteCarloIntegrator to display your “darts” in a window.

By default, Dartboard displays a 512×512 window for the area bounded by \((-1, 1)\) and \((1, -1)\). These parameters can be customized with six arguments, corresponding to the window width and height, upper-left corner, and lower-right corners of the “dart board,” respectively.

For example, this command will display a 1024×1024 window, and the coordinates will en-
close the $12 \times 12$ square whose upper left corner is 
$(-6, 6)$ and lower right corner is $(6, -6)$:

```java
    java Dartboard 1024 1024 -6 6 6 -6
```

When Dartboard is given input whose format exactly matches the prescribed output of either the $\pi$ estimator or the Monte Carlo integrator, it plots the provided hits in blue and misses in red. If all goes well, the hits will resemble the shape or function that you are counting, and the missed will resemble the rest of the target range.

You can test the Dartboard manually by typing in entries that match the output format of the preceding estimators. For example, typing:

```java
    start
    0 0 in
    -1 1 out
    2 -2 out
    end
```

...will display three $\times$’s along a downward diagonal, with the upper-left and lower-right ones in red (because they are “out”) and the one in between in blue (because it is “in”).

**Implementation Notes**

- Due to the nature of these programs, unit tests are more difficult to design for them and therefore they are not required for this assignment. However, if you can implement some tests, don’t stop yourself.
- Yes, for this assignment, you are writing code from scratch. :) In addition, you are free to create additional classes or methods to help with your implementation(s). These choices are all available to you now.

**Extra Credit: Riemann Integrator**

In our grading system, extra credit as an additional column in your standards development report where proficiencies do not count against you if they lower your overall outcomes.

Write a Java program, RiemannIntegrator, that uses the classic adjacent rectangles technique to estimate the area under a given curve. For this implementation, start with a single rectangle then increase the number of rectangles per iteration until the difference between previous and succeeding estimates falls within $\pm 0.01\%$ of each other. At this point, the program should stop and provide its final estimated area.

Have your Riemann integrator accept the exact argument format as the Monte Carlo integrator *(Pro-tip: Having the same arguments format implies that you can use the same argument processing code!*), with the exception, of course, of the number of darts to throw (because there are no darts here!).

Because this technique is entirely different from the randomized approach, its output will not be the same; in fact, as an extra credit assignment, we will leave the output choice to you. It should be enough to show clearly how the integrator iterates through rectangle sizes until it converges on that $\pm 0.01\%$.

**How to Turn It In**

Submit your code to your GitHub repository under the folder estimators. As always, don’t forget to commit as you go.