COS125 - Precept 4 (Loops)

1 Tracing Loops

Write the largest value that variable `counter` takes on in each of the code snippets below.

Code	$n = 2$	$n = 8$	$n = 128$

`int counter = 0;`
`for (int i = 0; i < n; i++)`
`   counter++;`
	2	8	128

`for (int i = 0, counter = 1; i < n; i++)` `   counter *= 2;`			

`int counter = 0;`
`while (counter < n)`
`   counter++;`			

`int counter = 0;`
`for (int i = 1; i <= n; i *= 2)`
`   counter++;`			

`for (int i = 0, counter = 0; i < n; i++)`
`   for (int j = 0; j < n; j++)`
`      counter++;`			

`int counter = 0;`
`for (int i = 1; i <= n; i *= 2)`
`   for (int j = 0; j < n; j++)`
`      counter++;`			

2 Computing π

Download `precept4.zip` from the precepts webpage. Unzip and open the project folder.

Write a program `MonteCarloPi.java` that computes an approximation to $\pi \approx 3.1416\ldots$ with the following Monte Carlo simulation: reading a number of iterations n from the command line, initialize a variable `hits = 0` and repeat the following n times:

1. Sample a (uniformly) random point (x, y) from the square $[-1, 1] \times [-1, 1]$.

2. Check if the point lies inside the unit disc: if $x^2 + y^2 \leq 1$, increment `hits` by 1.

Finally, multiply `hits` by $\frac{4}{n}$ and print the result.[1] (*Hint: you may find a particular lecture slide useful.*)

3 Greatest Common Divisor

Euclid's Algorithm

Create a program named `Euclid.java` that implements Euclid's algorithm for finding the greatest common divisor (GCD) between two numbers. Your algorithm should take two positive `long` command-line arguments.

Euclid's algorithm proceeds as follows: set r_1 to be the larger and r_2 the smaller between a pair (a, b) of positive integers, and generate a sequence by setting r_n to be the remainder of the division of r_{n-2} by r_{n-1} (recall that `%` is the Java operation for the remainder).

The algorithm terminates when $r_n = 0$, and the GCD is r_{n-1}.

Gaussian Integers (Bonus)

The GCD algorithm also works for other objects, not just integers! One example are the Gaussian integers: complex numbers $ai + b$ such that both a and b are integers.[2]

Euclid's algorithm works exactly the same way once we figure out what "quotient" and "remainder" mean with respect to two Gaussian integers x and y:

1. Let $e + fi = \frac{x}{y}$ (a complex number that is not necessarily a Gaussian integer). Then the quotient is $q = m + ni$, where m is the closest integer to e and n is the closest integer to f, tiebreaking up. Formally, e and f are the (unique) integers that satisfy $-\frac{1}{2} < e - m \leq \frac{1}{2}$ and $-\frac{1}{2} < f - n \leq \frac{1}{2}$.

2. The remainder is $r = x - qy$.

Implement Euclid's algorithm for Gaussian integers in `EuclidGaussian.java`. Your algorithm should take four positive `long` command-line arguments, and interpret them as the pair of complex numbers $(ai + b, ci + d)$.

[1] The probability of a random point in a 2×2 square belonging to the inscribed circle is the ratio between the area of the circle ($\pi \cdot 1^2 = \pi$) and the area of the square (4), so π is 4 times this probability.

[2] The formal version of this statement is that Gaussian integers form an Euclidean domain; this yields a very elegant proof of Fermat's theorem, a characterization of the integers that can be written as a sum of squares.