Chapter 72 Programming Exercises

Exercise 1

A **prime number** is an integer that cannot be divided by any integer other than one and itself. For example, 7 is prime because its only divisors are 1 and 7. The integer 8 is not prime because its divisors are 1, 2, 4, and 8.

Another way to define prime is:

```
prime(N)    = prime(N, N-1)

prime(N, 1) = true

prime(N, D) = if D divides N, false
              else prime(N, D-1)
```

For example,

```
prime(4)   = prime(4,3)
prime(4,3) = prime(4,2)
prime(4,2) = false
```

Another example,

```
prime(7)   = prime(7,6)
prime(7,6) = prime(7,5)
prime(7,5) = prime(7,4)
prime(7,4) = prime(7,3)
prime(7,3) = prime(7,2)
prime(7,1) = true
```

Translate the math-like definition of prime into two Java methods that return `boolean`. Use the `%` operator to test divisibility. Put your method into a class, write a testing class, and test your program. (Look at `FactorialTester.java` in this chapter.)

If you run your program for integers larger than about 12,000 (on a Windows system) you will run out of memory. Your program will stop running and report a *StackOverflowError*. This is because each activation in the activation chain requires some memory, and 12,000 activations uses up all the memory that has been reserved for this use.

The method can be improved by a factor of two by noting that only integers less than or equal to N/2 could possibly divide N. Modify the definition of prime and see what difference it makes.

This is not a good method for finding primes. If you really want to compute primes, use the Sieve of Eratosthenes.

Click here to go back to the main menu.

Exercise 2

Assume that female rabbits live for only 4 months. Modify the math-like definition of the Fibonacci series to account for dying rabbits. Implement the new series as a Java method.

First draw a chart showing the population of rabbits by month. Then deduce the new rules for the series. You will have more base cases than in the original series.

Click here to go back to the main menu.

Exercise 3

The *greatest common divisor* GCD of two integers `a` and `b` is the largest integer `k` that divides both `a` and `b` evenly. (That is, `k` divides both `a` and `b` without leaving a remainder.)

For example, `GCD( 6, 9 ) == 3` because 3 evenly divides both 6 and 9 and no greater integer does so. Another example, `GCD( 25, 55 ) == 5`.

Here is a math-like definition of GCD:

```
GCD(0,N) = N

GCD(A,B) = GCD( B%A, A )     % is the remainder after integer division B/A
```

For example,

```
GCD( 6, 9 ) = GCD( 9%6, 6 ) = GCD( 3, 6 )
GCD( 3, 6 ) = GCD( 6%3, 3 ) = GCD( 0, 3 )
GCD( 0, 3 ) = 3
```

Translate the math-like definition of GCD into a Java method. Write a `main()` method to test it.

Click here to go back to the main menu.

End of Exercises