CMSI 186
Programming Laboratory
Spring 2008

Program 4: Arithmetic First Principles

Unlike the previous (and some upcoming) programs, for this assignment, you are asked to do something we all know how to do — basic arithmetic. The wrinkle is, you are asked to do it from scratch, and to motivate the from-scratch characteristic, we'll do arithmetic for something that isn't available in a number of programming languages: the arbitrarily large — shall we say *ginormous* — integer.

Program to Write

Write a class called *math.GinormInt* that represents arbitrarily large integers, with up to *Integer.MAXINT* digits (once you look up *Integer.MAXINT*, I think you'll agree that this value is big enough for the label "arbitrarily large"). A JavaDoc description of this class can be found on the course Web site; most of the methods should be fairly self-explanatory:

http://myweb.lmu.edu/dondi/spring2008/cmsi186/program4-api

To demonstrate *math.GinormInt*, write a collection of simple applications using this class. Some ideas:

- An *Exponent* class that raises some integer (including negative numbers) to some non-negative integer power (including zero) — this is given as a free example.

  java Exponent –3428347589 1001

- A *Factorial* class that calculates the factorial of any non-negative integer.

  java Factorial 234918230

- A *Fibonacci* class that takes a natural number *n* and calculates the *n*th number in the Fibonacci sequence (0, 1, 1, 2, 3, 5, 8…).

  java Fibonacci 10000

- A *GCD* class that finds the greatest common divisor of two arbitrarily large positive integers.

  java GCD 12359485749568767666745857293509351 1234456797957795749373967798797979743

Note that, while you can write "normal" versions of these applications, those versions will not be capable of handling numbers as large as the values representable by *math.GinormInt*.

Design Notes

- You are free to decide on the internal representation of your arbitrarily large integers as well as your implementations for their arithmetic operations. The bottom line is that you produce a class that works correctly, and that you implement these operations *from scratch*.
- As always, write out the stubs for the code first; while they remain unimplemented, have the methods throw an *UnsupportedOperationException*.
- After laying out the stubs, write unit tests. Place the tests in a separate *math.GinormIntTest* class, called from its *main()* method. Invoking *java math.GinormIntTest* should run these tests.
- After completing *math.GinormIntTest*, proceed to implementation: start with correctly representing and displaying large integers first; then move into addition and subtraction. From here, you can do multiplication, integer division, and finally remainder (mod); the need for comparisons (equals, greater than, less than) will arise naturally as you implement these operations.

Gotchas

- While conceptually simple, this assignment is *much* harder than it looks. Fortunately, arithmetic is something we largely know how to do; the trick here is learning how to represent this knowledge in Java.
- Certain approaches to some operations are easier to program and run more efficiently than others — pay attention in class for discussion of your algorithm alternatives.
- While these integers are created and displayed as decimal strings, there's nothing stopping you from representing them internally with a different base (e.g., binary). Consider this alternative when deciding on a representation.