Assignment 0326

This assignment asks you to do something that we all know how to do: arithmetic. The wrinkle is, you are asked to implement arithmetic from scratch (“first principles”), and to motivate this, we’ll do arithmetic for something that isn’t available in some programming languages: arbitrarily large integers.

Outcomes

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

Not for Submission

You may refer to the java.math.BigInteger class as an example of an arbitrarily large integer implementation, but for reference, experimentation, and testing only. All submitted code must be written by you.

For Submission

Write a class that represents arbitrarily large integers. The sample test harness and API documentation use the name Intzilla, but just for fun you are encouraged to give it any appropriately imposing name, updating the test harness code as necessary. Instances of this class may have up to Integer.MAXINT digits (once you look it up, you’ll agree that this value is big enough for the label “arbitrarily large”).

Specifications

To model real-world Java class documentation, an API description of this class can be found on the course website. The required methods should be fairly self-explanatory.

Demonstration Programs

To demonstrate the power of your large integer class, write two or more simple applications using this class. A free Exponent example is provided to get you started; this program raises an integer (including negative numbers) to some non-negative integer power (including zero):

$ java Exponent -3428347589 1001
-3428347589 raised to the +1001-th power is -4168041497(…unbelievably humongous number of digits…)2887589.

Some ideas for demonstration programs include:

- A Factorial class that calculates the factorial of any non-negative integer.
- A Fibonacci class that takes a natural number \( n \) and calculates the \( n \)th number in the Fibonacci sequence \((0, 1, 2, 3, 5, 8, \ldots)\). (recommended for those who saw the Fibonacci problem in CMSI 185)
- A DataStorage class that takes a number and a data unit (e.g., kilobyte, megabyte, gigabyte, terabyte, petabyte, yottabyte, etc.) then displays the number of bytes represented by that value. To keep it interesting, use binary rather than decimal multiples; prefixes and corresponding powers of 2 can be found on the web.
- A MarketCapitalization class that takes a stock price and number of shares for some company then displays the market capitalization of that company. (look up the stock information of a favorite company to see some examples—for simplicity, assume that amounts are given in cents [e.g., a company with a stock price of $122.34 would be entered as 12234])

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. What matters is that you produce a class that works correctly, and that you implement these operations from scratch.
- As always, write out compilable stubs for the code first.
- A starter test suite has been provided, but is by no means complete. However, to get started, you can implement the methods for which tests have already been written.
- Beyond the starter test suite, write new tests first and then implement the tested methods.
• For methods that are not finished by the due date, throw an `UnsupportedOperationException`.

• Note from the above bullet that this assignment is known to be highly challenging. Fortunately, arithmetic is something that we largely know how to do by hand; the trick here is figuring out how to express these activities in Java.

• Certain approaches to arithmetic operations are easier to program and run more efficiently than others—pay attention in class for some possible algorithm alternatives.

• Although these integers are created and displayed as decimal numbers, there is nothing stopping you from representing them internally with a different base (e.g., binary). Consider this alternative when deciding on a representation. We are striving solely for correctness here, and are not worried about performance or efficiency.

**How to Turn It In**

Upload your code to your GitHub repository. Don’t forget to commit as you go.

Due to the number of source files involved in this assignment, you should place your code in a separate folder.