COMP 250 Assignment 1

Prepared by Prof. Michael Langer

Posted: Fri. Sept. 16, 2016
Due: Sun. Oct. 2 at 23:59 PM.

General Instructions

- The T.A.s handing this assignment are Stephanie Laflamme, Victor Chenal, Tzu-Yang (Ben) Yu, Pierre Thodoroff, and Rohit Verma. Their office hours and location and email contacts are posted on mycourses (see Announcements). Stephanie is the team leader, so if you have general issues about the assignment then please contact her or me.

- Use the mycourses discussion boards for clarification questions only. Stephanie will monitor the discussion board.

- The starter code is given on the public web page. Do not change any of the starter code. Add code only where instructed. [Added Sept 24: You can write your helper methods, as long as they are called only the "// ADD CODE HERE" block. The main point here is that we don't want you changing the methods that were given to you.]

- Do not change the package name **(a1posted)**, as this slows down the grading. If you are unsure what a package is, see the Java tutorials .

- The starter code includes a tester class that you can run to test if your methods are correct. Your code will be tested using a more extensive set of test cases. We encourage you modify this tester code and to share your tester code with other students on the discussion board. Try to identifiy tricky cases. Do *not* hand in your tester code. The TAs have their own.

- **Submission Instructions:** Submit a single zipped file **A1.zip,** which contains the modified NaturalNumber.java file, to the myCourses assignment A1 dropbox. Include your name and student ID number in the comment at the top of the NaturalNumber.java file. If you accidently submit the starter code, you will be penalized, so doublecheck that you have submitted the correct file. If you have any issues that you wish the TAs (graders) to be aware of, then include them as a comment at the top of your file.

- **Late assignment policy:** Late assignments will be accepted up to only 3 days late and will be penalized by 20 points per day. If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc. So make sure you are nowhere near that threshold when you submit.

Introduction

Computers represent integers as binary numbers (base 2), typically using a relatively small number of bits e.g. 16, 32, or 64. In Java, integer primitive types *short, int* and *long* use these fixed number of bits, respectively. Using a fixed number of bits for integers limits the range of values that one can work with, however. This is a significant problem, for example, in cryptography and data security where one needs to use very large integers to transform files so that they are encrypted.

For any base, one can represent any positive integer *p* uniquely as a polynomial, namely the sum of powers of the base:

$$p = \sum_{i=0}^{n-1} a_i \ base^i = a_0 + a_1 \ base + a_2 \, base^2 + \ldots + a_{n-1} base^{n-1}$$

where the coefficients a_i satisfy

$$0 \le a_i < base$$

and $a_{n-1} > 0$. The last condition is important for uniqueness and comes up below in the Tips.

The positive integer p can be represented as a list of coefficients $(a_0, a_1, a_2, \ldots \ a_{n-1})$. Notice that the ordering of the coefficients is opposite to the usual ordering that we use to represent numbers, e.g. integer 35461 would be represented as a list (1,6,4,5,3).

In this assignment, you will implement basic arithmetic operations on large positive integers. Java has class for doing so, called **BigInteger**. You will implement your own version of that class. In particular, you will implement basic arithmetic algorithms that you learned in grade school. Your representation will allow you to perform these operations in any base from 2 to 10. The methods could be extended to larger bases but we will not bother since it would require symbols for the numbers {10, 11, ..} and otherwise the solution would be the same.

You are given a partially implemented **NaturalNumber** class. The class has two private fields: **base** and **coefficients.** The coefficients a_i of the number are represented using the **LinkedList<Integer>** class, with coefficients ordered as described above. The starter code for the class also contains:

- code stubs of the methods that you are required to implement.
- helper methods that are implemented for you and that you are free to use, namely **clone(), timesBaseToThePower()**, **compareTo()**, **toString()**. You are not allowed to modify these helper methods.
- a **Tester** class with a simple example of how the methods that you will implement could be tested.

Your Task

Implement the following methods. The signatures of the methods are given in the starter code. You are *not* allowed to change the signatures.

1) plus (30 points)

Implement the grade school addition operation.

We call it **plus** rather than **add** to avoid confusing it with the **add** method for lists. Also note that this is the easiest of the four methods to implement.

2) times (30 points)

Implement the grade school multiplication method. Do not build up rows in a 2D table in your solution. Instead accumulate the sum of the rows. Here you will use the **plus()** method, so note that you need to complete the previous part.

We suggest that you write a helper method that computes the product of the multiplicand (a) times a single digit in the multiplier (b), i.e. what you essentially do in grade school in each row of your 2D table.

The starter code provides a slow multiplication method which uses repeated addition. You can use this method to verify the correctness of your result. You should also use it to appreciate how slow the slow method is relative to grade school multiplication. Examine for yourself how big the integers can be before the slow multiplication method becomes intolerably slow.

3) minus (25 points)

Implement the grade school subtraction method. The starter code verifies **a.minus(b) > 0**, so you can assume this in your tests.

Although this question is worth fewer points, it is perhaps more challenging than the first two, because it can be tricky to handle the borrowing properly.

4) divide (15 points)

Implement the grade school long division algorithm. This is the most challenging question. We have provided you with a slow division method which is based on repeated subtraction.

Other Requirements

Add comments to the beginning of each method to describe your solution. Pseudocode in particular would be helpful so that the TA can see quickly what your approach is. Also intersperse comments throughout the method to clarify more detailed steps that might not be obvious from the code itself.

Use [Java naming conventions] for variable names e.g. variables and method names should be mixed case with a lower case first letter.

Points will be removed for poor style e.g. non-existing or unhelpful comments, or improper indentation. Eclipse does proper indentation automatically, as do other excellent IDEs. (It also has a powerful debugger, and provides suggestions to correct your compile time errors.)

Tips

We suggest that you begin by testing your code on numbers that are written in base *10*. Once that is working, test it on bases 2 to 9. Use an online converter to verify your answers e.g.

http://www.cleavebooks.co.uk/scol/calnumba.htm

You may write your own helper methods, but if you do then you must be sure to document them, so that the TA grading it can easily follow what you are doing.

The methods should not change the numbers. This is the reason why the starter code for the methods “clones” the number when necessary before it operates on the number. (Note that all Java classes have a **clone()** method which that a copy of the object. I have written a helper clone method for the NaturalNumber class that overrides the default one.) There are several reasons why your operations might change the numbers. First, some methods are just easier to implement if the two numbers have the same number of digits. So, if in some instance the two numbers don’t have the same number of digits, then we clone and modify the smaller number to add higher order digits with value 0. Second, s method might change the digits of one of the numbers. For example, the subtraction operation **a.minus(b)** may require “borrowing” from higher to lower powers for the number **a**.

The code uses a **LinkedList** rather than an **ArrayList**. We are not suggesting that a linked list is a better data structure for this problem from an efficiency point of view, and indeed one could argue that an array list would be better since there are many times when you will want to access an element at an arbittary position in the list and array lists are typically faster for that. The main reason we use a **LinkedList** is that this class has more methods, for example, **addFirst()**, **addLast()** which makes coding easier to read,. [ADDED Sept 22: Also, feel free to iterate through the list using the get(index) method, even though as I’ve argued in the lecture that this is relatively inefficient. As long as the number of digits in your number isn’t too big (in the thousands or worse), then this particular inefficiency isn’t a problem.]

Get started early! Have fun! Good luck!