Homework Project 2: Infinite Integer

CS 0445 (Spring 2015) — Data Structure

Due Sunday June 28, 2015 at 11:59pm

As you may know, there is a limit of how big the primitive type **int** can be. Generally around 10 digits. For this project, you will create a class named `LInfiniteInteger` (Linked List Infinite Integer) which can be used to represent a very large integer containing as many digits as users want. You infinite integer should be able to support both positive and negative numbers. Note that it is not really infinite since it is still limited by the amount of memory. What you need to do for this project is to modify the provided starter class file `LInfiniteInteger.java`.

The Class `LInfiniteInteger`

For this project, you **must** create the class `LInfiniteInteger`. For this project, you can imagine that an infinite integer is an object. It is an object containing a collection of data where data is a single digit. For example, an infinite integer 58392304 is simply an object containing 5, 8, 3, 9, 2, 3, 0, and 4. Obviously these data must be organized in order. To support negative value, this object should also contains the sign of the number. There will be a set of useful operations (methods) that we can do with this infinite integer as follows:

- Users can check how many digits are there in an infinite integer.
- Users can check whether the infinite integer is a negative number.
- Users can add two infinite integers and get a new infinite integer as a result.
- Users can subtract an infinite integer by another infinite integer and get a new infinite integer as a result.
- Users can multiply an infinite integer by another infinite integer and get a new infinite integer as a result.
- Users can compare two infinite integers using the method `CompareTo()`
- Users can obtain a string representing an infinite integer.

The above methods are defined in the interface named `InfiniteIntegerInterface.java`. Since the `LInfiniteInteger` will be implemented using linked list where each node contains only one digit, the performance may be slow for methods that require traversal. For better performance, you MUST use **doubly linked list** with **last node reference** just like one of your lab. Note that you are **NOT ALLOWED** to use any data structures (e.g., List, Bag, Stack, ArrayList, etc) to implement this infinite integer. Simply use Nodes just like the way we implement linked list implementation of ADT Bag and ADT List.

Constructors

Your class **must** contain the following constructors:

```
public class LInfiniteInteger implements InfiniteIntegerInterface
{
    private Node firstNode;
    private Node lastNode;
    private int numberOfDigits;
    private boolean isNegative;

    // extra instance variables

    public LInfiniteInteger(int x) {...}
    public LInfiniteInteger(String s) {...}

    // methods
}
```

The first constructor constructs an `LInfiniteInteger` object from an integer number. The second constructor constructs an `LInfiniteInteger` object from a string representing an integer. For example, if users want to construct some infinite integers, they can use the following statements:

```
InfiniteIntegerInterface a = new LInfiniteInteger(123);
InfiniteIntegerInterface b = new LInfiniteInteger(-1574);
InfiniteIntegerInterface x = new LInfiniteInteger("12345678901234567890");
InfiniteIntegerInterface y = new LInfiniteInteger("-23904857203449563489");
```

For simplicity, we will assume that users will always use a valid string representing a number. However, zeros padded as a prefix should be supported by your program. For example, user should be able to construct an infinite integers 1234567890 and -1234567890 using the following statement:

```
InfiniteIntegerInterface c = new LInfiniteInteger("00001234567890");
InfiniteIntegerInterface d = new LInfiniteInteger("-001234567890");
```

Note that sometimes you may want to construct an infinite integer object in a certain way that you may think it suits your implementation. If that is the case, you are allowed to have more constructors. But make sure you declare them `private` to ensure that users cannot use them.

Methods

For this class, you must implement the following methods:

1. `public int getNumberOfDigits()`: This method returns the number of digits of this infinite integer.
2. `public boolean isNegative()`: This method returns whether the infinite integer is a negative number.

3. `public String toString()`: This method returns a string representing this infinite integer. Note that if the number is 12345, this method should return `"12345"` (without quotation marks). If the number is -12345, this method should return `"-12345"` (without quotation marks). **Make sure** that you implement this method correctly since the provided test class (`LInfiniteIntegerTester.java`) will test your `LInfiniteInteger` class by checking the output string.

4. `public InfiniteIntegerInterface plus(final InfiniteIntegerInterface y)`:

   This method returns a **NEW** `LInfiniteInteger` object which represents the result of this infinite integer plus the infinite integer `y`.

5. `public InfiniteIntegerInterface minus(final InfiniteIntegerInterface y)`:

   This method returns a **NEW** `LInfiniteInteger` object which represents the result of this infinite integer subtracted by the infinite integer `y`.

6. `public InfiniteIntegerInterface multiply(final InfiniteIntegerInterface y)`:

   This method returns a **NEW** `LInfiniteInteger` object which represents the result of this infinite integer multiply by the infinite integer `y`. **NOTE** that for this method, you **must** implement it with performance in mind. If we implement $x \times y$ as $x + x + \cdots + x$, y times, it may be slow. Instead, we can use paper and pencil method for better performance. For example, when we learn how to multiply two numbers 123 and 321, we perform the following paper and pencil method:

```
  123 x
  321
  ---
  123     <- 123 x 1
 246  +   <- 123 x 20
369       <- 123 x 300
-----
39483
=====
```

7. `public int compareTo(final InfiniteIntegerInterface x)`: This method allows user to compare two infinite integer numbers. For simplicity, if the infinite integer `x` is less than the infinite integer `y`, `x.compareTo(y)` should return -1, `y.compareTo(x)` should return 1. If they are equal, simply returns 0.

Note that every method that takes an infinite integer as an argument, the type of that argument is declared `final`. By declaring the type of argument `final`, it prevents you from accidentally modify the reference variable to point to another infinite integer. However, it does not prevent you from modifying the content of the infinite integer. Make sure that we you implement a method such as `plus()`, you do not accidentally modify the content of operands. For example, if you have two infinite integer `x` and `y`. After you execute `x.plus(y)`, the content of `x` and `y` must remain unchanged.

Note that since one of our constructor takes `String` as the argument, you are **ALLOWED** to use the class `String` and its methods only in constructors. You are **NOT ALLOWED** to use the

class `String` in any other methods in your code except the method `toString()`. Note that use `toString()` to construct a new infinite integer are also allowed. For example,

```
InfiniteIntegerInterface x = new LInfiniteInteger("1234567890");
   :
InfiniteIntegerInterface y = new LInfiniteInteger(x.toString());
```

You are also allowed to use `Integer.parseInt()` to convert a string to an integer. **NO `import` of any kinds are allowed in your `LInfiniteInteger.java`.**

Hint

Note that when you have two numbers (e.g., 5 and 9) which can be either positive or negative numbers, the result of addition and subtraction can be either 14 or 4 (again can be either positive or negative). The following are all possible combinations and their answers:

- $5 + 9 = 5 - (-9) = 9 + 5 = 9 - (-5) = 14$,
- $5 - 9 = 5 + (-9) = (-9) + 5 = (-9) - (-5) = -4$,
- $9 - 5 = 9 + (-5) = (-5) + 9 = (-5) - (-9) = 4$, and
- $(-9) - 5 = (-9) + (-5) = (-5) - 9 = (-5) + (-9) = -14$.

You can see that no matter what combination, the answer will be either 14, -4, 4, or -14. The idea is, you only need to implement two algorithms; (1) adding two positive numbers, and (2) subtracting the larger positive number by the smaller positive number. Which one should be used depend on the operands and the operator.

Test Your Class

In the CourseWeb, you will find a program named `LInfiniteIntegerTester.java`. Note that this test class use the your method `toString()` to compare the result. Make sure that you implement the method `toString()` correctly. To test your program, simply execute this program. It will perform some tests on your infinite integer. If there are errors, it will tell you. Note that this program is not perfect. It cannot tell you what is wrong with your class `LInfiniteInteger`. You have to trace your code and `LInfiniteIntegerTester.java` and see why it says `FAIL`. If you think I need to add any additional tests, please let me know. I will add those into this test class and update it.

Another program named `LInfiniteIntegerCalculator` is also provided. This is a calculator program that uses your `LInfiniteInteger` class to perform calculation. You can enter huge integer numbers in arithmetic expression format (no parentheses supported) and get the result in the form of huge integer number. Unlike any other calculator programs which always show huge numbers in scientific format, this one will not.

Due Date and Submission

This assignment is due on Sunday June 28, 2015 at 11:59pm. No late submission will be accepted. Simply submit your `LInfiniteInteger.java` to the CourseWeb under Project 2.