ECE 242: Data Structures and Algorithms- Fall 2015

Project 4: FunWithBigNumbers

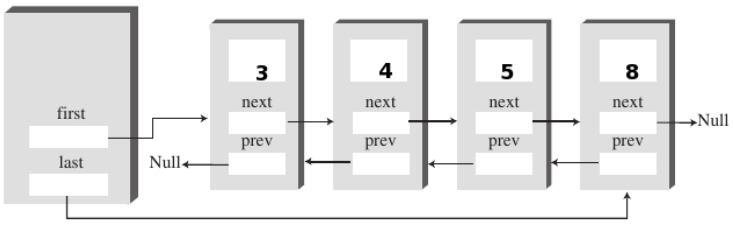
Due Date: **window of submission up to 11:50pm November 11, 2015**

Description

In many applications, such as numerical approximations, or the computation of large prime numbers for cryptographic applications, the standard Java types such as int and double are limited in size. For example, an int variable in Java has a maximum value of 2,147,483,647=$2^{31}-1$ (largest known prime until 1855), and primes generated for cryptographic purposes can easily be several hundred digits. The situation is similar for IEEE floating point numbers (stored as double). To overcome these limitations, a common approach is to implement a custom number class, that supports addition, subtraction, multiplication and division. In fact, Java provides arbitrary precision arithmetic in the BigDecimal and BigInteger classes; for this project, we will develop our own number class, called BigNumberLL using a Doubly-Linked-List and for integer only, that can be used for arbitrary precision addition, subtraction and multiplication (we will not deal with division for this project). In other words, the only limitation on the size and precision of a BigNumber will be available memory and our class will easily be able to handle numbers that are hundreds of digits long. Notice that storing a real number of arbitrary precision and size means that we must have a way to store an arbitrary number of digits associated with the number; in other words, we need a list that can have arbitrary length to store all the digits of a given number. We will use a linked list to store our arbitrary length integer, where each item in the linked list will contain a digit (between 0 and 9). In order to implement addition, subtraction, and multiplication, we will obviously need to traverse and manipulate the linked list associated with an arbitrary precision number. Although there is more than one right answer, you will follow the skeleton of the provided code and fulfill are the specifications that are presented below.

How to represent BigNumbers?

You will be using a doubly linked list (using two ends). For example: the number **3,458** can be implemented as follows:



The `Link` class can be implemented using:

```
class Link {
    public int val = 0;
    public Link prev = null;
    public Link next = null;

    public Link(int val) {
        this.val = val;
        next = null;
        prev = null;
    }
}
```

and the class `BigNumberLL` class is defined as:

```
class BigNumberLL {
    public Link first;             // reference to first Link
    public Link last;              // reference to last Link
    private int size=0;            // size of the number (#Links)
    private boolean positive=true; // sign of the bignumber

    // simple constructor
    BigNumberLL() {
        first=null;
        last=null;
    }

    // a constructor that takes a string and converts it into a BigNumber
    BigNumberLL(String s) {
    // to complete
    }

///// All Methods to follow..............

}
```

Note that we have added the variable `positive` that stores the sign of the big number (Remark: the correct terminology should be "non-negative" rather than positive, since the zero is included in our positive set). For the constructor, we note that the input String may contain a minus sign (first character), zeros on the left, comas or spaces. This String will have to be somehow filtered in order to be converted into into a BigNumberLL format. Mode details and specifics are provided below. In addition, we note here that one can use the method `Character.getNumericValue(c)` to convert a character to an integer.

Methods to implement

Basic methods

- `insertLast`: If you read a String, character by character from left to right, you need to be able to insert each integer (converted from the character) inside the linked-list while keeping the ordering of a queue. For example: 3,458, do `insertLast(3)`, `insertLast(4)`, etc.
- `insertFirst`: reverse the order of insertion. While doing an operation (for example addition), you often perform it step by step from the right to the left. The resulting integer needs to be inserted in the list using `insertFirst` to make sure that the ordering of the whole number is correct.
- `deleteFirst`: it is used by the `trimFront` method.
- `trimFront`: if the BigNumber list contains a lot of zeros on the left, we can trim this number (remove the zeros) and reduce its size.
- `isEmpty`: self-explanatory
- `getSize`: self-explanatory
- `setSign`: change the sign of a given big number
- `getSign`: self-explanatory
- `toString`: Convert the big number list into a string that can be used to print. Here, we want to include a comma "," to separate all the thousands. For example: 3458 is "3,458"; 345 is "345", 12349875 is "12,349,875". Also, a negative sign must be added at the beginning of the String if the number is negative.

Static operations methods

Note that the methods such as: add, subtract, and multiply are declared as being static. This is so that the method can be used generically, on any two instances of the `BigNumberLL` class. So if we had BigNumberLL variables X and Y, then to add them we could simply use the statement `BigNumberLL.add(X, Y)`. Here is the list of the methods to implement:

- `compare(X,Y)`: Compare two `BigNumberLL` and return the sign of (X-Y). You would return 1 is $X-Y > 0$; 0 if $X = Y$ or -1 if $X - Y < 0$.
- `add(X,Y)`: Add two `BigNumberLL` and return a new one.
- `subtract(X,Y)`: Subtract two `BigNumberLL` and return a new one.
- `multiply(X,Y)`: multiply two `BigNumberLL` and return a new one.
- `multiplyBase(X,i)`: multiply one `BigNumberLL` X with an integer i, and return a new `BigNumberLL`. This kernel (base) can be used by the `multiply` method to perform the multiplication step by step.

Important In this project, while using the operations add, subtract, or multiply we will consider only positive number! (to simplify the coding). We note two exceptions: (i) the subtraction of two positive input numbers can return a negative number; (ii) the compare method must be general while comparing numbers, you can input negative or positive numbers to compare them.

Specification

In order to test your `BigNumberLL` class, we are providing a class that performs arbitrary arithmetic operation from user input, called `BigNumberCalculator`. The code is asking to choose between 5 main operations: (p,=,+,-,*,X).

If 'p' is chosen the user will be asked to enter a number (or arbitrary size) that will be converted from String to BigNumberLL format and converted back again by the code to String format to be printed on screen. For example if the input number is "3458" it will return "3,458"; if it is "00, 003 458" it will also returned "3,458"; if it is "-00003,458" it becomes "-3,458".

If '=' is chosen the user will be asked to enter two numbers (of arbitrary size) that will be compared.

If '+' is chosen the user will be asked to enter two numbers (of arbitrary size) that will be added.

If '-' is chosen the user will be asked to enter two numbers (of arbitrary size) that will be subtracted.

If '*' is chosen the user will be asked to enter two numbers (of arbitrary size) that will be multiplied.

If 'X' is chosen, the code exit.

Examples

Execution of various matrix application examples to illustrate how the code should work.

Option 'p'

```
Welcome to BigNumber Calculator
Choose Operation (p,=,+,-,*) or X for exit
p
Enter first number:
-00001 345,785
Number is -1,345,785
  }
}
```

Option '='

```
Choose Operation (p,=,+,-,*) or X for exit
=
Enter first number:
12346
Enter second number:
-13498
sign(number1-number2) is 1
```

Option '+'

```
Choose Operation (p,=,+,-,*) or X for exit
+
Enter first number:
123
Enter second number:
3
answer is 126

Choose Operation (p,=,+,-,*) or X for exit
+
Enter first number:
3
Enter second number:
123
answer is 126

Choose Operation (p,=,+,-,*) or X for exit
+
Enter first number:
123456789123456789
Enter second number:
987654321987654321
answer is 1,111,111,111,111,111,110
```

Option '-'

```
Choose Operation (p,=,+,-,*) or X for exit
-
Enter first number:
46
Enter second number:
1045
answer is -999

Choose Operation (p,=,+,-,*) or X for exit
-
Enter first number:
1045
Enter second number:
```

```
46
answer is 999

Choose Operation (p,=,+,-,*) or X for exit
-
Enter first number:
555666777888999222
Enter second number:
222333666999111333
answer is 333,333,110,889,887,889
```

Option '*'

```
Choose Operation (p,=,+,-,*) or X for exit
*
Enter first number:
1234
Enter second number:
0
answer is 0

Choose Operation (p,=,+,-,*) or X for exit
*
Enter first number:
467
Enter second number:
9
answer is 4,203

Choose Operation (p,=,+,-,*) or X for exit
*
Enter first number:
12567
Enter second number:
98767
answer is 1,241,204,889

Choose Operation (p,=,+,-,*) or X for exit
*
Enter first number:
01234567891234567891234567 89
Enter second number:
98765432198765432198765432 1
answer is 121,932,631,356,500,531,591,068,431,581,771,069,347,203,169,112,635,269
```

REQUIREMENT, HINTS, SUBMISSIONS

1. You should implement your code step-by-step. Option 'p' first, then option '=', follows by "+", "-" and finally "*". Reminder: use only input positive numbers with "+", "-", "*" that help reducing the difficulty of the problem.

2. You will work alone or in groups of two maximum. **Each group should submit one copy of the solution. Please do not forget to include the name of both partners!** Include the **name**, **student ID**, and **email address** of both partners. This information should be included in a README file. Submit a single zip file composed of: All your source files (*.java), your README file.

Grading Policy

This project will be graded out of 100 points:

1. Your program should compile successfully. (20 points)
2. Your program should implement basic functionality and run correctly. (60 points)
3. Overall programming style: source code should have proper identification, and comments. (20 points)

Extra Credit

You can earn 10 additional points by extending the functionality of your code by addressing operations "+,-,*" using negative numbers as well. You can earn another 10 additional points by performing the division. Indicate what you have done in the README file.