Programming Assignment 1

Big Integer

In this assignment you will use linked lists to store and manipulate integers with an unlimited number of digits

Worth 80 points (8% of course grade)

Posted Fri, Sep 14

Due Fri, Sep 28, 11:00 PM (WARNING!! NO GRACE PERIOD).

Extended deadline (with ONE time free extension pass): Mon, Oct 1, 11:00 PM (NO GRACE PERIOD)

You get ONE free extension pass for assignments during the semester, no questions asked. There will be a total of 5 assignments this semester, and you may use this one free extension pass for any of the 5 assignments.

A separate Sakai assignment will be opened for extensions AFTER the deadline for the regular submission has passed. The extension will be 3 days (72 hours). If/when you choose to use the one-time extension, you don't need to ask for permission - just drop your submission in the extension assignment. And you can do this even if you dropped something in the regular submission, and later on decided to use the extension - in this case, only the extension submission will be graded, the regular submission will be ignored.

- You will work **individually** on this assignment. Read the DCS Academic Integrity Policy for Programming Assignments - you are responsible for this. In particular, note that **"All Violations of the Academic Integrity Policy will be reported by the instructor to the appropriate Dean".**
- **IMPORTANT - READ THE FOLLOWING CAREFULLY!!!**

  Assignments emailed to the instructor or TAs will be ignored--they will NOT be accepted for grading.
  We will only grade submissions in Sakai.

  If your program does not compile, you will not get any credit.

  Most compilation errors occur for two reasons:

  1. You are programming outside Eclipse, and you delete the "package" statement at the top of the file. If you do this, you are changing the program structure, and it will not compile when we test it.
  2. You make some last minute changes, and submit without compiling.

  To avoid these issues, (a) START EARLY, and give yourself plenty of time to work through the assignment, and (b) Submit a version well before the deadline so there is at least something in Sakai for us to grade. And you can keep submitting later versions (up to 10) - we will accept the LATEST version.

Background

Integer.MAX_VALUE is the maximum value of a Java `int`: 2147483647. If you want to work with even bigger integers, you have the option of using the type `long`, which has the maximum value of `Long.MAX_VALUE` = 9223372036854775807.

But what if this is not enough? What if you are working on something like an astronomy application, and need to keep track of things such as number of stars in the universe? This is of the order of 10^{23}, larger than the maximum long value. For situations like this, you need to be able to work with an integer type that can hold arbitrarily large or small positive and negative values, with any number of digits. There is no built-in type in the language for this, so you need to craft your own. In this assignment, you will do exactly this, by implementing a class called `BigInteger`, with a representative small set of operations.

The trick is to store an integer as a linked list of digits. For example, the integer 754 will be stored as:

```
4 -> 5 -> 7
```

Why are the digits stored backward? It's because computations such as adding or multiplying big integers are easier to do if the linked list stores digits in ascending order of positional value. So the least significant digit is in the first node of the linked list, and the most significant digit is in the last node.

This is a simple linked list, NOT circular, with a front pointer. The sign (positive or negative), is stored separately in a boolean field in the

program.

Also, there can never be zeros at the end of the list. Such zeros will be insignificant. So, for instance, 00754 will be stored as:

```
4 -> 5 -> 7 (NOT 4 -> 5 -> 7 -> 0 -> 0)
```

Consequently, the value 0 will be stored as an EMPTY LINKED LIST. The number of digits is 0, and it is not negative.

Implementation and Point Assignment

Download the attached `biginteger_project.zip` file to your computer. DO NOT unzip it.

Instead, follow the instructions on the Eclipse page under the section "Importing a Zipped Project into Eclipse" to get the entire project into your Eclipse workspace.

You will see a project called `BigInteger` with classes `BigInteger`, `DigitNode`, and `BigTest` in package `bigint`. The `DigitNode` class implements the linked list node that will hold a digit of a big integer linked list.

You need to fill in the implementation of the following methods in the `BigInteger` class:

Method	Points
`parse`	15
`add`	35
`multiply`	30

Note: When parsing an input string as an integer, you can use the `Character.isDigit(char)` method to tell if a character is a digit.

Make sure to read the comments that precede classes, fields, and methods for code-specific details that do not appear here.

Observe the following rules while working on **BigInteger.java**:

- You may NOT add any `import` statements to the file.
- You may NOT add any new classes (you will only be submitting `BigInteger.java`).
- You may NOT add any fields to `BigInteger` class.
- You may NOT modify the headers of any of the given methods.
- You may NOT delete any methods.
- You MAY add helper methods if needed, as long as you make them `private`.

Also, you may NOT make any changes to the `DigitNode` class (you will only be submitting `BigInteger.java`). When we test your submission, we will use the exact same version of `DigitNode` that we shipped to you.

WARNING!!!

- If your linked list stores digits with most significant digit first, you will get ZERO for the corresponding test case, even if the result is mathematically correct. NO EXCEPTIONS.
- If your linked list stores insignificant zeros, you will get ZERO for the corresponding test case, even if the result is mathematically correct. NO EXCEPTIONS.
- If you use an array anywhere in your program, you will get ZERO. NO EXCEPTIONS.
- If you convert the input to a Java numeric type value (such as `int` or `long`) and then do arithmetic on it, instead of working with individual digits stored in a linked list, you will get ZERO. NO EXCEPTIONS.

Running/Testing

Use the class `BigTest` to test your implementation. Carefully read the code in the file to get a good idea of how the `BigInteger` methods are used.

Here's a sample run of `BigTest`:

```
(p)arse, (a)dd, (m)ultiply, or (q)uit? => p
Enter integer => 125
Value = 125

(p)arse, (a)dd, (m)ultiply, or (q)uit? => p
Enter integer => -126
Value = -126

(p)arse, (a)dd, (m)ultiply, or (q)uit? => p
Enter integer => +1
Value = 1
```

```
(p)arse, (a)dd, (m)ultiply, or (q)uit? => p
Enter integer => 005
Value = 5

(p)arse, (a)dd, (m)ultiply, or (q)uit? => p
Enter integer => 123xy56
Incorrect Format

(p)arse, (a)dd, (m)ultiply, or (q)uit? => a
Enter first integer => 12
Enter second integer => -13
Sum: -1

(p)arse, (a)dd, (m)ultiply, or (q)uit? => a
Enter first integer => 16756726
Enter second integer => 0
Sum: 16756726

(p)arse, (a)dd, (m)ultiply, or (q)uit? => m
Enter first integer => 12
Enter second integer => 200
Product: 2400

(p)arse, (a)dd, (m)ultiply, or (q)uit? => m
Enter first integer => 178
Enter second integer => -156
Product: -27768

(p)arse, (a)dd, (m)ultiply, or (q)uit? => m
Enter first integer => -16
Enter second integer => -05
Product: 80

(p)arse, (a)dd, (m)ultiply, or (q)uit? => q
```

NOTE: These are just sample tests, you will need to make several of your own for each method to make sure your implementation works correctly. Also, read the Grading Process section to understand what we look at to assess whether your methods run correctly or not. (It's NOT the printed output.)

Submitting

Submit your **BigInteger.java** source file (NOT BigInteger.class), in Sakai -> Assignments.

Refer to the instructions in the **Eclipse** page, under the section **The Eclipse Workspace** to know how to locate **BigInteger.java** on your computer for uploading.

Grading Process

Grading will be done after the submission deadline has passed. There is no real time feedback when you submit. You can submit to Sakai up to 10 times, and each submission will overwrite the previous. This means the auto-grader will only grade your last submission, since that will be the only submission in Sakai when the deadline has passed.

Your code will be auto-graded by a grading script that will run several test cases on each graded method.

For each test case, the result computed by your code will be compared with that computed by our correct code.

Note that the comparison is based on the state of the fields in the `BigInteger` object, and NOT on anything you might print in your program. (All printed output will be ignored. This also means if you threw in print statements for debugging and left them in your code, they will have no bearing on the grading.) What this means is we will compare the fields `negative`, `numDigits` and the linked list referenced via `front` in your `BigInteger` object, with those in the correct version in our test script.

For your benefit, we will ensure that the `add` and `multiply` methods are graded with the correct version of `parse`. In other words, even if your `parse` method does not work correctly, it will not impact the grade for your other methods. However, when testing `multiply` we will keep your `add` implementation since you may call it from `multiply`. This means if you do call `add` from implement `multiply`, any mistake in `add` will impact `multiply`.

When grading is done, your test report will be emailed, detailing the score on each test case. Test cases will be posted so you can run your program against them to verify the test report. Remember, verification means checking the state of the `BigInteger` instance fields, NOT what your program might print.