COP 3003 Object-Oriented Programming, Fall, 2016

Lab 1
Due: 11:59pm September 5, 2016
Important Notes:

1. Your programming format must be consistent with the conventions introduced in code_conventions.ppt. Failure to do that may cost you up 10% of your grade.
2. Please name your program where the main method is Lab1_Last_4_digits_UIN.java. Do not write your name anywhere in your program. Your assignment will NOT be grade if this requirement is not met.
3. When submitting, zip your source code and submit it on Canvas.

Assignment:

Data type int is a primitive type in Java. Each int variable is allocated 4 bytes or 32 bits in the memory. The range for a 32 bit integer is approximately -2*10^9 – 2*10^9. Then, int cannot represent integers, larger than 10 digits.

In this assignment, you are going to implement a class, called HugeInteger. This class can represent 40-digit integers. The idea is simple. In the HugeInteger class, we just use a 40-element int array. Each element stores a digit. In detail, class HugeInteger has the following three instance variables:

- private static final int NUM_DIGITS=40;
- private int digits[]=new int[NUM_DIGITS]; // by default, all array elements
  // are initialized zero
- private boolean positive;

The reasons why NUM_DIGITS is created are to 1) improve the readability of the program and 2) make the program easier to maintain.

Then we need to decide how to place digits in the array, called digits. Basically, you have the following choices:

1. the most significant digit (MSD) is at digits[0]
2. the most significant digit (MSD) is at digits[NUM_DIGITS-1]
3. the least significant digit (LSD) is at digits[0]
4. the least significant digit (LSD) is at digits[NUM_DIGITS-1]

It is up to you to select any of the above four. No matter which one you choose, please make it clear in comments. In this document, the discussion is based on the 4th choice. **Note your algorithm is not supposed to use data types with larger range, i.e. long or double or BigInteger. Your grade will be zero otherwise.**

Once we figured out the instance variables, it is time to outline the following two things: 1) HugeInteger's constructor(s) and 2) what member methods HugeInteger provides.

As to HugeInteger's constructor(s), your program needs to at least implement the following constructor:

```
/**
 * Converts an integer string to an array of digits.
 *
 * It does not check the validity of the string.
 *
 * @param num integer string
 */
public HugeInteger(String num)
```

As to HugeInteger's other methods/functions, there are two categories:

1. For comparison
    a. public boolean isEqualTo(HugeInteger hi)
    b. public boolean isGreaterThan(HugeInteger hi)
    c. public boolean isNotEqualTo(HugeInteger hi)
    d. public boolean isLessThan(HugeInteger hi)
    e. public boolean isGreaterThanOrEqualTo(HugeInteger hi)
    f. public boolean isLessThanOrEqualTo(HugeInteger hi)
2. For calculation
    a. public void add(HugeInteger hi) // adds the value in "hi" to the current object
    b. public void subtract(HugeInteger hi) // subtracts the value in "hi" from
    // the current object
    c. public void multiply(HugeInteger hi) // multiplies the current object by the value
    // in "hi"
    // (**<u>cannot call add or subtract</u>**)
    d. public void negate() // flip the sign
    e. public boolean isZero()
    f. public String toString(); // the returned string does not include the leading zeros

For the first category, think about if we have to implement each of them from scratch.

For the second category, the addition and subtraction should begin with the least significant digits in digits[NUM_DIGITS-1] and hi.digits[NUM_DIGITS-1]. Do not forget carries and borrows. As you may have already expected, the sign of integers may cause some complexity. You are encouraged to develop the following two methods only for internal uses:

```
/**
 * Is a private method and not for outside users to call. This method
 * interprets array1 and array2 as two integers whose LSB digits are
 * at array1[NUM_DIGITS-1] and array2[NUM_DIGITS-1]. Without sign
 * information, array1 and array2 can only represent positive integers.
 *
 * This method adds array1 and array2, and store the result in another
 * integer array for return.
 *
 * @param array1 first integer
 * @param array2 second integer
 * @return addition of array1 and array2
```

```
     */
    private static int[] addArrayDigits(int [] array1, int [] array2)



    /**
     * Is a private method and not for outside users to call. This method
     * interprets array1 and array2 as two integers whose LSB digits are
     * at array1[NUM_DIGITS-1] and array2[NUM_DIGITS-1]. Without sign
     * information, array1 and array2 can only represent positive integers.
     *
     * This method subtracts array2 from array1, and store the result in
     * another integer array for return.
     *
     * The assumption is the integer in array1 is greater than the one in
     * array2
     *
     * @param array1 first integer
     * @param array2 second integer
     * @return subtraction of array2 from array1
     */
    private static int[] subtractArrayDigits(int [] array1, int [] array2)
```

Lastly, *multiply* could be a little challenging. It is essentially implemented by a double-level for loop. Hint: you may need to count the number of digits in integers.

You are given a main method for testing purposes. Please use it without changing anything. The wanted output is attached.

At last, ignore number overflows!

Grading Policies:

- Good programming format: 10%
- Implement the constructor correctly: 20%
- Implement the comparison methods correctly: 30%
- Implement method *add* correctly: 17.5%
- Implement method *subtract* correctly: 17.5%
- Implement method multiply correctly: 5%