COMP 2080

Assignment 1 (10%)

Groups of 3 maximum

Due: Friday 16th February 2024 11:30PM

Submission Instructions:

1) **Fill in the full name and student number of each group member in the spaces provided further down this page.**
2) **Paste the code for each class and main program into this document after your chosen question.**
3) **One member of the group must upload this document to blackboard.**

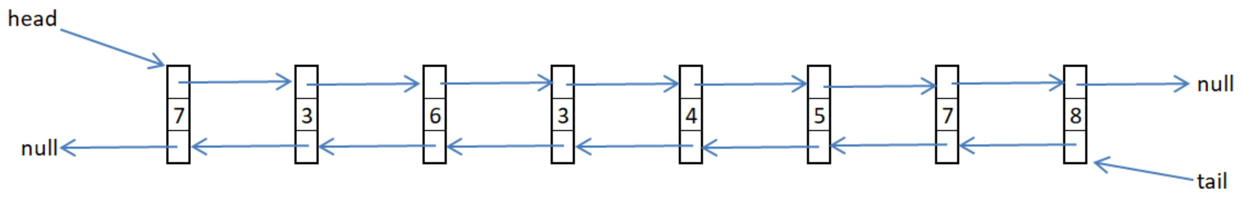
- You **must** have your names and student id numbers commented at the top of all code submitted.
- All submissions should at least compile.
- **Non-compiling assignments will not be marked and be given a grade of 0**.
- All your submissions should be suitably documented.

Name:	Student Id
Edwin Chung	101422064
Rafael Cas	101104938

Question:

Your task is to create a class called "***HugeInteger***" that stores and allows operations on huge integers. The class ***HugeInteger*** stores the digits of the number in a doubly linked list **in reverse order** (i.e the least significant digit is the first item, most significant digit is the last).

For example the number "87543637" would be stored in reverse order as shown below.



This class must have three (3) state values:

State variable name	Information stored
isPositive	Stores the sign of the number (positive or negative) as a boolean state value
head	Stores the first Node of the linked list (It is null if the list is empty).
tail	Stores or keeps track of the last item in the list (It is null if the list is empty).
length	Stores the number of digits in the number (excluding the sign)

Constructors

Constructors	
Public HugeInteger ()	An empty linked list is created. That is: ***isPositive*** must be set to true by default. **Head** and **tail** are set to null. If this variable is to be displayed, a "0" should be printed (the zero is not stored). **Length** must be set to 0;
Public HugeInteger (String number)	Creates the number from the string with all leading zeros (0) removed.

Additional Behavior

Method Prototype	
HugeInteger addPositive(HugeInteger num2)	Returns a new ***HugeInteger*** containing the result of adding num2 to the stored number**. You MUST assume *num2* and the number being added to are BOTH positive.**
int compareTo (HugeInteger num2)	Returns -1 if the number stored is less than num2 Returns 0 if the number stored is equal to num2 Returns 1 if the number stored is greater than num2
String toString()	Returns a string representation of the number
void concatenateDigit(int digit)	Adds a digit to the **end** of the number (at the front of the list). **Note: if the list is empty leading zeros should not be added.**
void addLast(int digit)	Adds a digit to the **front** of the number (at the end of the list). This can be used in the ***addPositive*** method

You must create a ***HugeInteger*** class based on the specifications outlined above and create a main program that tests it with the following code (on the next page):

```
  public static void main(String[] args) {
    HugeInteger hi = new HugeInteger();
    System.out.println(hi);
    // testing sign
    HugeInteger hi1 = new HugeInteger("34545234");
    System.out.println(hi1);
    HugeInteger hi2 = new HugeInteger("-2455434324344");
    System.out.println(hi2);
    //testing leading zeros
    HugeInteger hi3 = new HugeInteger("000034545234");
    System.out.println(hi3);
    HugeInteger hi4 = new HugeInteger("-00000002455434324344");
    System.out.println(hi4);
    // testing concatenate with a single digit
    HugeInteger hi5 = new HugeInteger();
    System.out.println(hi5);
    hi5.concatenateDigit(3);
    System.out.println(hi5);
    // testing add with two positive numbers
    HugeInteger hi6 = new HugeInteger("9");
    HugeInteger hi7 = new HugeInteger("6");
    HugeInteger hi8 = hi6.addPositive(hi7);
    System.out.println(hi6+" + "+hi7+" = "+hi8);
    HugeInteger hi9 = new HugeInteger("9996354");
    HugeInteger hi10 = new HugeInteger("4656");
    HugeInteger hi11 = hi9.addPositive(hi10);
    System.out.println(hi9+" + "+hi10+" = "+hi11);
    System.out.println(hi5.compareTo(hi4));
    System.out.println(hi2.compareTo(hi1));
  }
```

Sample output:

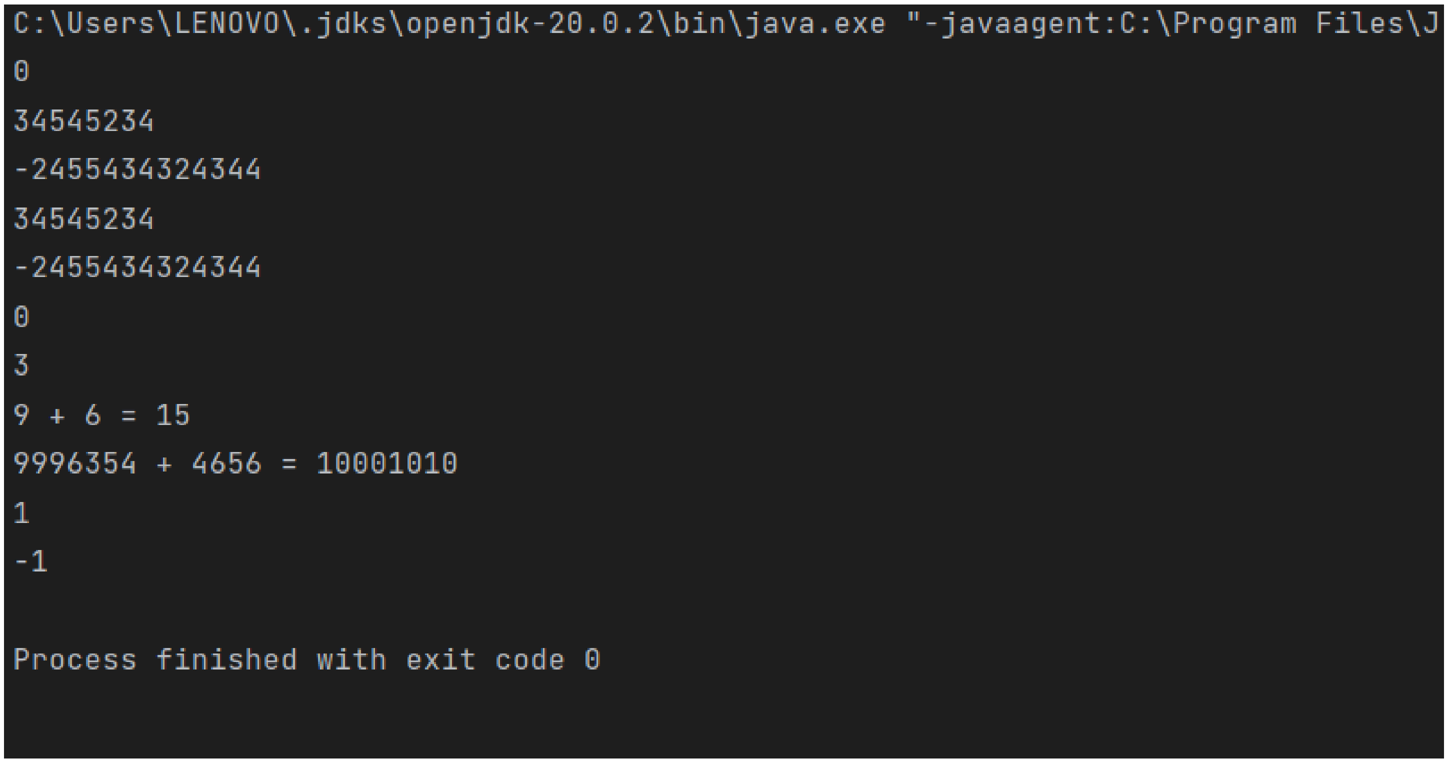
```
0
34545234
-2455434324344
34545234
-2455434324344
0
3
9 + 6 = 15
9996354 + 4656 = 10001010
1
-1
```

PASTE YOUR HUGEINTEGER CLASS HERE:

```java
public class HugeInteger {
   // four state variable names
   private boolean isPositive; // whether the number is positive or negative
   private Node head; // the most significant digit
   private Node tail; // the least significant digit
   private int length; // length of the number

   // Internal class Node for the assignment
   public class Node {
       int data;
       Node prev;
       Node next;
       Node(int data) {
           this.data = data;
           this.prev = null;
           this.next = null;
       }
   }

   // constructor to create the empty linked list
   public HugeInteger() {
       isPositive = true;
       head = tail = null;
       length = 0;
   }

   public HugeInteger(String number) {
       this(); // calls the default constructor
       if (number.startsWith("-")) {
           isPositive = false; // is negative based on the presence of -
           number = number.substring(1);
       }
       number = number.replaceFirst("^0+(?!$)", ""); // regex to remove leading zeros
       // Iterate through the characters of the string
       for (int i = 0; i < number.length(); i++) {
           int digit = number.charAt(i) - '0'; // Convert character to int
           addLast(digit); // ad the digit to the end of the linked list
       }
   }

   // Method to concatenate a single digit to the end of the linked list
   public void concatenateDigit(int digit) {
       Node newNode = new Node(digit);
       if (tail == null) {
           head = tail = newNode; // If the list is empty, set head & tail to the new node
       } else {
           // Else, add the new node after the tail and update it
           newNode.prev = tail;
           tail.next = newNode;
           tail = newNode;
       }
       length++;
   }

   // Add a single digit to the beginning
   public void addLast(int digit) {
       Node newNode = new Node(digit);
       if (head == null) {
           head = tail = newNode;
       } else {
           // Add the new node before the head and update it.
           newNode.next = head;
           head.prev = newNode;
           head = newNode;
       }
       length++;
   }

   public HugeInteger addPositive(HugeInteger num2) {
       HugeInteger result = new HugeInteger();
       Node current = this.head;
       Node Num2 = num2.head;
       int carry = 0;

       // Iterate through the digits of both numbers to perform addition
       while (current != null || Num2 != null || carry != 0) {
           int sum = carry;
           if (current != null) {
               sum += current.data;
               current = current.next;
           }
           if (Num2 != null) {
               sum += Num2.data;
               Num2 = Num2.next;
           }
           result.concatenateDigit(sum % 10); // Add the result digit to the result number
           carry = sum / 10; // Update carry for the next iteration
       }

       return result; // Return result of addition
   }

   // Method to compare two HugeInteger numbers
   public int compareTo(HugeInteger num2) {
       // Comparing with positive and negative
       if (this.isPositive && !num2.isPositive)
           return 1;
       if (!this.isPositive && num2.isPositive)
           return -1;
       // Comparing the length of the node
       if (this.length > num2.length)
           return this.isPositive ? 1 : -1;
       if (this.length < num2.length)
           return this.isPositive ? -1 : 1;

       // Comparing digits from most significant to least
       Node current = this.tail;
       Node Num2 = num2.tail;

       while (current != null && Num2 != null) {
           if (current.data > Num2.data)
               return this.isPositive ? 1 : -1;
           if (current.data < Num2.data)
               return this.isPositive ? -1 : 1;
           current = current.prev;
           Num2 = Num2.prev;
       }
       return 0;
   }

   // Convert the HugeInteger to a string representation
   @Override
   public String toString() {
```

```java
        if (head == null)
            return "0";
        StringBuilder sb = new StringBuilder(isPositive ? "" : "-");
        Node current = tail;
        while (current != null) {
            sb.append(current.data); // Append each digit to the StringBuilder
            current = current.prev; // Move to the next digit
        }
        return sb.toString(); // Return the string representation of the number
    }
}
```

```java
// Edwin Chung - 101422064
// Rafael Cas - 101104938

public class Main {
    public static void main(String[] args) {
        HugeInteger hi = new HugeInteger();
        System.out.println(hi);
        // testing sign
        HugeInteger hi1 = new HugeInteger("34545234");
        System.out.println(hi1);
        HugeInteger hi2 = new HugeInteger("-2455434324344");
        System.out.println(hi2);
        //testing leading zeros
        HugeInteger hi3 = new HugeInteger("000034545234");
        System.out.println(hi3);
        HugeInteger hi4 = new HugeInteger("-00000002455434324344");
        System.out.println(hi4);
        // testing concatenate with a single digit
        HugeInteger hi5 = new HugeInteger();
        System.out.println(hi5);
        hi5.concatenateDigit (3);
        System.out.println(hi5);
        // testing add with two positive numbers
        HugeInteger hi6 = new HugeInteger("9");
        HugeInteger hi7 = new HugeInteger("6");
        HugeInteger hi8 = hi6.addPositive(hi7);
        System.out.println(hi6+" + "+hi7+" = "+hi8);
        HugeInteger hi9 = new HugeInteger("9996354");
        HugeInteger hi10 = new HugeInteger("4656");
        HugeInteger hi11 = hi9.addPositive(hi10);
        System.out.println(hi9+" + "+hi10+" = "+hi11);
        System.out.println(hi5.compareTo(hi4));
        System.out.println(hi2.compareTo(hi1));
    }

}
```



```
C:\Users\LENOVO\.jdks\openjdk-20.0.2\bin\java.exe "-javaagent:C:\Program Files\J
0
34545234
-2455434324344
34545234
-2455434324344
0
3
9 + 6 = 15
9996354 + 4656 = 10001010
1
-1

Process finished with exit code 0
```