CSC 151 Assignment #6

1. Honor Code

A. For individual assignments: Jane Doe and John Doe will be replaced by your full name(s)
I affirm that I have carried out my academic endeavors with full academic honesty.
[Signed, Manav Bilakhia]
B. Resources/References
Lecture notes
Geeksforgeeks for syntax help

2. Java files and outputs

A. Java files
Class: Huge.java

```java
/*
 * I affirm that I have carried out the attached academic endeavors with full academic
honesty.
 * Manav Bilakhia (MB)
 */
package assignment;
import java.util.ArrayDeque;
import java.util.Deque;
/**
 * This class represents large nonnegative integers having up to 50 digits.
 *
 * @author Jesse Grabowski
 * @author Charles Hoot
 * @author Frank M. Carrano
 * @author Joseph Erickson
 * @author Zeynep Orhan modified
 * @version 5.0
 */
public class Huge {
    private Deque<Integer> hugeNumber;
    private final static int MAX_SIZE = 50;
    /**
     * Constructor: No parameters. Creates an ArrayDeque for hugeNumber and add 0
     */
    public Huge()
    {
        hugeNumber = new ArrayDeque<>();
        hugeNumber.add(0);
    }
    /**
     * Constructor with an Integer array parameter: Creates an ArrayDeque for
     * hugeNumber and sets the value of the Huge number based on a given array of
     * Integers
     *
     * @param digits Integer[] digits
     */
    public Huge(Integer digits[])
    {
        hugeNumber = new ArrayDeque<>();
        // calling sethuge so that zeros are checked and every element of the array is
added
        setHuge(digits);
    }
    /**
     * Constructor with a String parameter: Creates an ArrayDeque for hugeNumber and
     * sets the value of the Huge number based on a given String
     *
     * @param hugeString a String of digits
     */
    public Huge(String hugeString)
    {
        hugeNumber = new ArrayDeque<>();
        // calling sethuge so that zeros are checked and every element of the string is
added
        setHuge(hugeString);
    }
    /**
     * Constructor with another Huge number parameter: Creates an ArrayDeque for
     * hugeNumber and sets the value of the Huge number based on a given Huge number
     *
     * @param huge
     */
    public Huge(Huge huge)
    {
        hugeNumber = new ArrayDeque<>();
        // calling sethuge so that zeros are checked and every element of the huge is
added
        setHuge(huge.toArray());
    }
    /**
     * setHuge: Sets the value of the Huge number based on a given array of Integers.
May
     * throw an Error if the number is too large. throw new Error("Overflow") when
     * the number of digits is greater than the MAX_SIZE Ignore zeros at the
     * beginning and add the digits starting from the 0th index to the end of the
     * deque
     *
     * @param digits An array of Integers that represents the Huge's digits.
     *
     */
    public void setHuge(Integer digits []) {
        if (digits.length > MAX_SIZE) {
            throw new Error("Overflow");//error overflow
        }
        //clearing so that everytime setter is called, it has a different value
        hugeNumber.clear();
        boolean startWithZero = true;
        for (int index = 0; index < digits.length; index++) {
            if (!digits[index].equals(0) || !startWithZero) {
                hugeNumber.add(digits[index]);
                if (startWithZero)
                    startWithZero = true;
            }

        }
        if (hugeNumber.size() == 0) {
            hugeNumber.add(0);

        }
    }

    /**
     * setHuge: Sets the value of the Huge number based on a string of numbers. May
throw an
     * Error if the number is too large, or a NumberFormatException if the string is
     * formatted incorrectly.
     *
     * throw new Error("Overflow") when the number of digits is greater than the
     * MAX_SIZE
```

```java
     *
     * throw new NumberFormatException("Non-hugeNumber in huge int"); when a digit
     * is not a character in [0-9]
     *
     * Ignore zeros at the beginning and add the digits starting from the char at
     * 0th index to the end of the deque
     *
     * @param hugeString The string to convert into a Huge.
     * @throws NumberFormatException if the string is formatted incorrectly.
     */
    public void setHuge(String hugeString) throws NumberFormatException {
        if (hugeString.length() > MAX_SIZE) {
            throw new Error("Overflow");//error overflow
        }
        //clearing so that everytime setter is called, it has a different value
        hugeNumber.clear();
        boolean startingWithZero = true;
        for (int index = 0; index < hugeString.length(); index++) {
            Character character = hugeString.charAt(index);
            if (!Character.isDigit(character)) {
                throw new NumberFormatException("Bad Character");
            }

            Integer digit = Character.getNumericValue(character);
            if (digit != 0 || !startingWithZero) {
                hugeNumber.add(digit);
                if (startingWithZero) {
                    startingWithZero = false;
                }
            }
        }
        if (hugeNumber.size() == 0) {
            hugeNumber.add(0);
        }
    }
    /**
     * Override toString: Print the digits next to each other without a space
     */
    @Override
    public String toString() {
        String str = "";
        for (Object o : hugeNumber.toArray()) {
            Integer digit = (Integer) o;

            str += digit.toString();
        }

        return str;
    }
    /**
     * toArray: Converts the Huge into an array of Integers.
     *
     * @return An array representation of the Huge.
     */
    public Integer[] toArray() {
        Integer[] arr = new Integer[hugeNumber.size()];

        Object[] objectArr = hugeNumber.toArray();
        for (int i = 0; i < hugeNumber.size(); i++) {
            arr[i] = (Integer) objectArr[i];
        }

        return arr;
    }

    /**
     * addHuge: Adds another Huge to this Huge without changing either one.
     *
     * @param h The Huge to add to this Huge.
     * @return A Huge which is the sum of both Huges.
     */
    public Huge addHuge(Huge h)
    {
        String operand1 = this.toString();
        String operand2 = h.toString();
        if (operand1.length() > operand2.length()){
            String temp = operand1;
            operand1 = operand2;
            operand2 = temp;
        }
        String result = "";
        int length1 = operand1.length();
        int length2 = operand2.length();
        operand1=new StringBuilder(operand1).reverse().toString();
        operand2=new StringBuilder(operand2).reverse().toString();
        int carry = 0;
        for (int index = 0; index < length1; index++)
        {
            int sum = ((operand1.charAt(index) - '0') +
                    (operand2.charAt(index) - '0') + carry);
            result += (char)(sum % 10 + '0');
            carry = sum / 10;
        }
        for (int index = length1; index < length2; index++)
        {
            int sum = ((operand2.charAt(index) - '0') + carry);
            result += (char)(sum % 10 + '0');
            carry = sum / 10;
        }
        if (carry > 0)
            result += (char)(carry + '0');

        result = new StringBuilder(result).reverse().toString();
        Huge sum = new Huge(result);
        return sum;
    }
    /**
     * multiplyHuge: Multiplies another Huge to this Huge without changing either one.
     *
     * @param h The Huge to multiply to this Huge.
     * @return A Huge which is the product of both Huges.
     */
    public Huge multiplyHuge(Huge h)
    {
        String operand1 = this.toString();
        String operand2 = h.toString();
        int length1 = operand1.length();
        int length2 = operand2.length();
        if (length1 == 0 || length2 == 0) {
            Huge product = new Huge();
            return product;
        }

        // storing result in a vector
        // in reverse order
        int result[] = new int[length1 + length2];
        // Below two indexes are used to
        // find positions in result.
        int position1 = 0;
        int position2 = 0;

        // Go from right to left in num1
        for (int index1 = length1 - 1; index1 >= 0; index1--)
        {
            int carry = 0;
            int num1 = operand1.charAt(index1) - '0';

            // To shift position to left after every
            position2 = 0;

            // Go from right to left in num2
            for (int index2 = length2 - 1; index2 >= 0; index2--)
            {
                // Take current digit of second number
                int num2 = operand2.charAt(index2) - '0';

                // Multiply with current digit of first number
                // and add result to previously stored result
                // charAt current position.
                int sum = num1 * num2 + result[position1 + position2] + carry;

                // Carry for next itercharAtion
                carry = sum / 10;

                // Store result
                result[position1 + position2] = sum % 10;

                position2++;
            }

            // store carry in next cell
            if (carry > 0)
                result[position1 + position2] += carry;
            position1++;
        }

        // ignore '0's from the right
        int index = result.length - 1;
        while (index >= 0 && result[index] == 0)
            index--;
        if (index == -1)
        {
            Huge product = new Huge();
            return product;
        }

        // generate the result String
        String s = "";

        while (index >= 0)
            s =s+ (result[index--]);
        if (s.length()> MAX_SIZE)
            throw new Error("Overflow");
        Huge product = new Huge(s);

        return product;

    }
    /**
     * getHuge: Returns a duplicate of the given Huge representation of a String.
     *
     * @param s The String to convert into a Huge.
     * @return A duplicate of the Huge version of the String.
     */
    public static  Huge getHuge(String s)
    {
        Huge huge = new Huge(s);
        return huge;
    }
    /**
     * isZero: Determines if the Huge is = 0.
     *
     * @return true if the Huge is 0, otherwise false
     */
    public boolean isZero()
    {
        /*
        String str = this.toString();
        int n = str.length();
        for (int index = 1; index < n; index++)
            if (str.charAt(index) != '0')
                return false;
        return true;
         */
        if(hugeNumber.getFirst() == 0)
            if (hugeNumber.size() == 1)
                return true;
            else
                return false;
        else
            return false;
    }
} // end Huge
```

Class: Driver.java

```
package assignment;
/**
 * This class to demonstrate the class Huge.
 *
 * @author Charles Hoot
 * @author Frank M. Carrano
 * @author Zeynep Orhan modified
 * @version 5.0
 */
public class Driver {
    public static void main(String[] args) {
        Integer h1[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        Integer h2[] = { 0, 0, 0, 0, 0, 0, 0, 0, 1 };
        Integer h3[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        Integer h4[] = { 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 };
        Huge huge1 = new Huge(h1);
        Huge huge2 = new Huge(h2);
        Huge huge3 = new Huge(h3);
        Huge huge4 = new Huge(h4);
        Huge addHuge, multHuge, rHuge;
        if (huge1.isZero()) System.out.println(huge1 + " is zero.");
        else System.out.println(huge1 + " is not zero.");
        if (huge2.isZero()) System.out.println(huge2 + " is zero.");
        else System.out.println(huge2 + " is not zero.");
        if (huge3.isZero()) System.out.println(huge3 + " is zero.");
        else System.out.println(huge3 + " is not zero.");
        if (huge4.isZero()) System.out.println(huge4 + " is zero.");
        else System.out.println(huge4 + " is not zero.");
        rHuge = new Huge();
        addHuge = huge3.addHuge(huge4);
        multHuge = huge3.multiplyHuge(huge4);
        System.out.println(huge3 + " + " + huge4 + " = " + addHuge);
        System.out.println(huge3 + " * " + huge4 + " = " + multHuge);
        System.out.println("huge1 is " + huge1);
        System.out.println("huge2 is " + huge2);
        System.out.println("huge3 is " + huge3);
        System.out.println("huge4 is " + huge4);
        String goodString = "1234567890123456789012345678901234567890";
        String badString = "12345678901234567890x12345678901234567890";
        try {
            System.out.println("Converting a string to a huge integer:");
            rHuge = Huge.getHuge(goodString);
            System.out.println(rHuge.toString());
            System.out.println("Converting a string to a huge integer that is too 
large:");
            rHuge = Huge.getHuge(badString);
            System.out.println(rHuge.toString());
        } // end try
        catch (NumberFormatException e) {
            System.out.println("Error converting a string to a huge integer.");
        } // end catch
        System.out.println("Zeros at the beginning should be removed");
        Integer h7[] = { 0, 0, 2, 2, 2 };
        Huge huge7 = new Huge(h7);
        System.out.println(huge7);
        System.out.println("Set to zero");
        huge7.setHuge("0");
        System.out.println(huge7);
        System.out.println("Trying a multiplication that will result in overflow.");
        Huge huge5 = Huge.getHuge(goodString);
        Huge huge6 = huge5.multiplyHuge(huge5);
    } // end main
} // end Driver
```

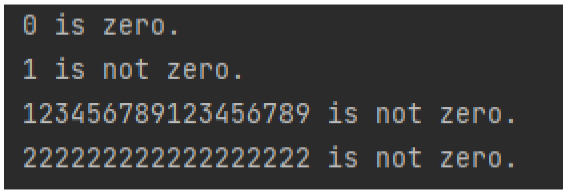
B. Sample output 1

I. Describe your test 1: checking is zero method

II. Text output 1:

0 is zero.
1 is not zero.
123456789123456789 is not zero.
222222222222222222 is not zero.

III. Screenshot 1:



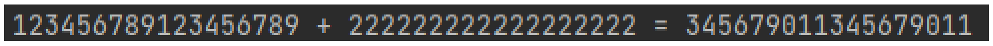
```
0 is zero.
1 is not zero.
123456789123456789 is not zero.
222222222222222222 is not zero.
```

C. Sample output 2

I. Describe your test 2: checking the add huge method

II. Text output 2:
123456789123456789 + 222222222222222222 = 345679011345679011

III. Screenshot 2:



```
123456789123456789 + 222222222222222222 = 345679011345679011
```

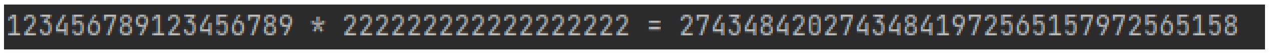
D. Sample output 3

I. Describe your test 3: checking the multiply huge method

II. Text output 3:

123456789123456789 * 222222222222222222 = 27434842027434841972565157972565158

III. Screenshot 3:



```
123456789123456789 * 222222222222222222 = 27434842027434841972565157972565158
```