Assignment 1: Large Numbers

Goal

Implementing Arithmetic Operations on Large Numbers Using a Doubly Linked List

Motivation

Handling large numbers efficiently is critical in various fields such as cryptography, high-precision scientific calculations, and financial transactions. In many cases, the primitive data types in Java (such as int or long) cannot store such large values, leading to overflow issues. Therefore, this assignment introduces you to representing large **non-negative** integers using a Doubly Linked List where each digit is stored in a node.
In this assignment, you will implement addition and multiplication of large numbers stored as doubly linked lists. By doing so, you will understand how to work with linked lists and how to handle big numbers manually, without relying on built-in data types for large integers.

Assumptions

All numbers in this assignment are non-negative integers.

Example of Big Number Representation Using a Linked List

Each large number will be stored as a doubly linked list, where each node contains a single digit. For example, the number 123456789 will be stored in the following structure:
Head -> [1] <-> [2] <-> [3] <-> [4] <-> [5] <-> [6] <-> [7] <-> [8] <-> [9] <- Tail
This structure allows you to traverse the number from either the most significant digit (the head) or the least significant digit (the tail), which is useful for performing arithmetic operations such as addition and multiplication.

Examples

Addition:
Input: num1 = 123456789, num2 = 987654321

Representation:
num1: Head -> [1] <-> [2] <-> [3] <-> [4] <-> [5] <-> [6] <-> [7] <-> [8] <-> [9]
+
num2: Head -> [9] <-> [8] <-> [7] <-> [6] <-> [5] <-> [4] <-> [3] <-> [2] <-> [1]

Result:
Head -> [1] <-> [1] <-> [1] <-> [1] <-> [1] <-> [1] <-> [1] <-> [1] <-> [1] <-> [0]

Multiplication:
Input: num1 = 123, num2 = 456

Representation:
num1: Head -> [1] <-> [2] <-> [3]
*
num2: Head -> [4] <-> [5] <-> [6]

Result:
Head -> [5] <-> [6] <-> [0] <-> [8] <-> [8]

Provided Skeleton Files

You are provided with the skeleton files **Link.java**, **BigNum.java**, and **Main.java**.

```
public class Link {
    int digit;
    Link prev;
    Link next;
   // Constructor for the Link class
    public Link(int digit) {
        this.digit = digit;
        this.prev = null;
        this.next = null;
    }
}
```

```
public class BigNum {
    private Link head;
    private Link tail;


    public BigNum(String numberStr) {
        // TODO: Implement the string to linked list conversion
    }

    public BigNum() {}

    public BigNum add(BigNum other) {
        // TODO: Implement addition of two BigNum objects
        return null;
    }

    public BigNum multiply(BigNum other) {
        // TODO: Implement multiplication of two BigNum objects
        return null;
    }

    public String toString() {
        StringBuilder result = new StringBuilder();
        Link current = head;
        while (current != null) {
            result.append(current.digit);
            current = current.next;
        }
        return result.toString();
    }
}
```

In the **BigNum** class, you will implement the constructor and the `add` and `multiply` methods.

Explanation of Methods to Implement

Constructor: public BigNum(String numberStr)

This constructor is responsible for initializing a BigNum object using a string representation of a large non-negative integer. The input to this constructor is a string (e.g., "123456789"). The constructor should convert this string into a doubly linked list where each node represents a single digit of the number. The most significant digit should be stored in the head node, and the least significant digit should be stored in the tail node.

Implementation:

1. Iterate over each character in the input string `numberStr`.
2. Convert each character to its numeric value (using `Character.getNumericValue()`).
3. Append each digit to the linked list in the order they appear in the string. The first digit should go in the head of the list, and the last digit should be at the tail. For example, if the input is "123", the resulting linked list should look:
   Head -> [1] <-> [2] <-> [3] <- Tail.

Method: add(BigNum other)

Input: The add method takes one input parameter `other` - a BigNum object representing another large non-negative integer to be added to the current BigNum object.

Output: The method should return a new BigNum object representing the sum of the current BigNum and other.

Implementation:

- Traverse both BigNum linked lists from tail to head (i.e., from the least significant digit to the most significant digit).
- For each pair of corresponding digits, calculate their sum. If the sum exceeds 9, handle the carry by adding it to the next digit sum.
- If one number is shorter than the other, assume the missing digits as 0 and continue adding until both lists are exhausted.
- If there is still a carry left after processing all digits, append it as a new digit to the result (at the head).
- Return the result as a new BigNum object.

Method: multiply(BigNum other)

Input: The multiply method takes one input parameter – `other` -a BigNum object representing another large non-negative integer to be multiplied with the current BigNum object.

Output: The method should return a new BigNum object representing the product of the current BigNum and other.

Implementation:

- Use the long multiplication technique, which involves multiplying each digit of the current BigNum by each digit of the other number.
- Handle carry for each multiplication step.
- After multiplying one digit of the other number with all digits of the current number, shift the result by appending the appropriate number of zeros (this is equivalent to shifting the partial results).
- Sum the partial products to get the final result.
- Return the final result as a new BigNum object.

For convince, we attached the Main.java file with the running examples.

Enjoy!