Infinite Precision Arithmetic (Project 2)

In most programming languages, including Java, the integer datatype can only hold numbers up to a maximum value. For example, the maximum value an `int` in Java can hold is $2^{31} - 1$. However, there are many scientific fields which make use of numbers far larger than these, and require the ability to make computations upon them. For this project you will implement addition, multiplication, and exponentiation by representing positive integers as linked lists.

Objectives

- To learn how to build a project from the ground-up.
- To review Linked Lists and build one in Java.
- To practice writing your own tests to ensure software correctness.
- To practice reading files in Java.

Requirements

- Define a new class which can represent arbitrarily large numbers.
- Read a file which contains various arithmetic expressions as strings.
- Solve each arithmetic expression, and print out the result.
- Create multiple tests to ensure your functions work as expected.

Specifications

Your program must accept one command-line argument. It represents the name of the input file.

The input file will contain an arithmetic expression on each line. For example, a possible input file might look like this:

```
1234567890 + 987654321

111111111   * 122333444455555

10  ^  50
```

Your program should be able to read each line properly regardless of extraneous or missing spacing. The arithmetic operators you must account for are `+` (Addition), `*` (Multiplication), and `^` (Exponentiation). Your program must compute each expression and print the result to standard output (i.e., using `System.out.println`).

The corresponding printed output for the example input would be:

```
1234567890 + 987654321 = 2222222211

111111111 * 122333444455555 = 13592604925913506171605

10 ^ 50 = 100000000000000000000000000000000000000000000000000
```

Your output formatting must match this exactly.

- Exactly one space between an operator and a number.
- Lines and expressions in exactly the same order as they were given in the input file.
- An equals sign, followed by the correct answer as computed by your program.

Remove leading zeroes from your outputs. For example you may end up with a number like `000409`. When you print this out as a result, be sure to remove leading zeroes so that it's printed as `409`.

Implementation

The main problem in this project is representation of the big integers and calculation of arithmetic operations `+`, `*`, and `^`.

Integers are to be represented in your program as Linked Lists of digits, one digit per node in the list. Each digit may be represented as a `char` or as an `int`, as you prefer. You will find that the operations are easier to implement if you store the list backward (low order to high order).

Important: You may only use code you have written. You must implement your own Linked List class, and not use the `LinkedList` class that is available in the Java standard library.

Example: The integer 436 (four hundred and thirty-six) may be represented in a linked list as: $6 \rightarrow 3 \rightarrow 4$. Note that results should be returned with digits in the correct order, so be sure to convert between this representation and a "normal" representation as appropriate.

Addition is easily performed by starting at the lowest order digit, then adding pairs of digits until you reach the end of both numbers. Don't forget to carry!

Multiplication can be performed the same way as you would do it on paper.
(Seriously, try a 2 or 3 digit multiplication problem on paper to remind yourself of how it works!)
For each nth digit, make a copy of the right operand. Multiply the nth digit of the left operand by each digit in the copy while remembering to carry, then put n zeros on the end of the copy. Add all of these copies together to get the result.

Exponentiation is a bit more complicated. The trivial implementation would be to use repeated multiplication, i.e. `12^4 = 12*12*12*12`. However, with large numbers, this becomes untenably slow for even moderately large exponents. Instead, you must implement the *exponentiation by squaring* algorithm, which takes advantage of the observation that, for a positive integer n,

$$x^n = \begin{cases} x(x^2)^{\frac{n-1}{2}} & \text{if } n \text{ is odd} \\ (x^2)^{\frac{n}{2}} & \text{if } n \text{ is even} \end{cases}$$

This drastically reduces the number of multiplications that need to be performed, and can be used to deal with larger values of x or n. You can assume that the exponent (i.e., n in the equations above) will fit in a regular `int` variable.

Addition, multiplication, and exponentiation may be implemented iteratively (using loops) or recursively, as you prefer.

Submission and Grading

You find the starter code for this project on Canvas. The starter code just sets up the project for you to ensure that it can be autograded correctly. Most of the design is left up to you.

You should write tests for your program from the beginning. You are required to submit your own tests for your program. You are encouraged to seek help during office hours, though if your question is about a bug in your program, I will first ask to see your testing efforts so far.

Your grade will be determined based on three pieces:

- The thoroughness of your tests (measured by branch coverage, which you can enable in IntelliJ IDEA).
- The percentage of instructor tests that your submission passes.
- The design of your project.

When structuring the source files of your project, use a flat structure; that is, your source files will all be contained in the `src` directory.

You will submit your files through Canvas.