When you have done both parts of this program, you will have designed and implemented a Reverse Polish Calculator (RPC) for symbolic polynomials in one variable, x, with coefficients of type double. These operations are available in computer algebra programs such as Maple, although the I/O is fancier.

An example of a polynomial is

\[ 3.5x^4 + 2.3x^2 - x + 3.5 \]

In this polynomial, the coefficients are 3.5, 0, 2.3, -1.0 and 3.5 for exponents 4, 3, 2, 1, and 0, respectively. (The 0 is because there is no \( x^3 \) term.)

The first part of this program is an extension of Exercise 2 in Chapter 2. First, define a Term class that will contains the exponent and coefficient for one term (e.g. 2.3\( x^2 \)). You do not need to store the name of the variable (x in the example). You should also define a Polynomial class that implements a complete polynomial object. You should use an ordered linked list for your polynomial representation (as in Section 2.10). The list should be sparse, that is, it should only contain Terms for elements with non-zero coefficients.

Polynomials can be added, subtracted, multiplied and divided just like ordinary numbers can. To add or subtract polynomials add or subtract the coefficients of terms with the same exponent. Remember that a missing term in one polynomial should be treated as a 0 coefficient. To multiply two polynomials, you have to take each term of the first polynomial, multiply it term-by-term by each term of the second polynomial, and then add up any coefficients with the same exponents. As an example:

\[
(2.0x^2 + 1.0) * (3.5x^4 + 2.3x^2 - x + 3.5) = \\
7.0x^6 + 4.6x^4 - 2.0x^3 + 7.0x^2 + 3.5x^4 + 2.3x^2 - x + 3.5 = \\
7.0x^6 + 8.1x^4 - 2.0x^3 + 9.3x^2 - x + 3.5
\]

Polynomial division works backwards – you have to find both a quotient and a remainder, such that the quotient * the polynomial divided-by + the remainder = the polynomial divided. (Since division is extra credit, you will have to look up the exact algorithm).

RPN stands for Reverse Polish Notation, and is a method of representing mathematical expressions without using parentheses. Each operation is represented by the operands followed by the operator (+, -, *, or /). An operand can be a single value, or it can be a complete operation. For example (using integers instead of polynomials), 12+ refers to (1+2); 12+3* means ((1+2)*3), and 123+* means (1 * (2+3)). RPN is also called postfix, and is described more fully (including code for evaluating a postfix numerical expression) in the Case Study in Section 3.4 of your textbook.
Your main program will read expressions (with ? in the place of each polynomial) and compute the result, using a stack.

An example:
??+?*=q would…
  Input and push the first polynomial
  Input and push the second polynomial
  Pop the top 2 polynomials, add them and push them onto the stack
  Input and push the third polynomial
  Pop the top 2 polynomials, multiply them and push them onto the stack
  Print the polynomial on top of the stack
  Quit

The program is divided into two parts. Part A requires knowledge from chapters 1-2 of the textbook. Part B requires knowledge from chapter 3. To complete this program on time, you will need to work on Part A immediately so that you will be ready to work on Part B as we learn about stacks. (This assignment is good preparation for the IRC, where you will not know everything that you need to know at the outset of the project, but you will be applying new material as you learn it).

PART A: Implement a linked-list based sparse representation of a polynomial as a list of terms. The coefficients of your polynomial should be of type double. Include at least the following member functions:

• A default constructor
• A toString method (see below for format)
• A method to add a term to an existing polynomial
• A method to change a term in an existing polynomial (Changing the coefficient of a term to 0 should remove it from the polynomial's list)
• A method to read a polynomial (use JOptionPane or System.in).
• A method to add a polynomial to the current polynomial and return a new polynomial result
• A method to subtract a polynomial from the current polynomial and return a new polynomial result
• A method to multiply a polynomial times the current polynomial and return a new polynomial result
• (EXTRA CREDIT) A function to divide the current polynomial by a polynomial and return a new polynomial result.

You must use the LinkedList class, and you must use iterators. For full credit, you should traverse your list using iterators, not numeric indexes. You may use any code that you like from the textbook, but you must annotate it with the page number. You may not use code from any other source, including the Internet.
To build the calculator, you will use the algorithm for Evaluating Postfix Expressions that is in your textbook on pp. 171-176. This should be implemented using a stack of polynomials.

The input to your program would satisfy the following conditions:

- Input will come from a file whose name should be specified by the user.
- All characters other than ?, +, −, *, / (if you implement division), = and q should be ignored.
- ? causes a new polynomial to be prompted for and read, using console I/O (System.in). This polynomial is pushed to the stack.
- +, −, *, and / (if division is implemented) should be treated as arithmetic operators, causing the operation to be performed on the top two elements of the stack (with the top element as the second value in subtraction and division).
- = causes the polynomial on top of the stack to be printed.
- q ends the program.

The output to your program would satisfy the following conditions:

- Terms are printed in descending order of exponent
- The ^ character is used to indicate exponents, e.g. 3.5x^4 + 2.3x^2 − x+ 3.5
- Terms with 0.0 coefficients are not printed.
- If the exponent is 1, only x should be printed, not x^1.
- If the exponent is 0, only the coefficient should be printed (1.1, not 1.1x^0).
- If the coefficient is 1.0, the coefficient is not printed (x^2, not 1.0x^2)

PART B: Implement the RPN calculator using a stack of polynomials. You must use the java.util.Stack class in your solution. Code from your textbook may be used with proper attribution.

In addition to satisfying all of the requirements above, your program should also provide useful error messages when the input RPN expression is incorrect, e.g. not enough operands on the stack for an operation to complete, and should not throw any exceptions that are not caught.