Implementing Polynomials in Java

Introduction

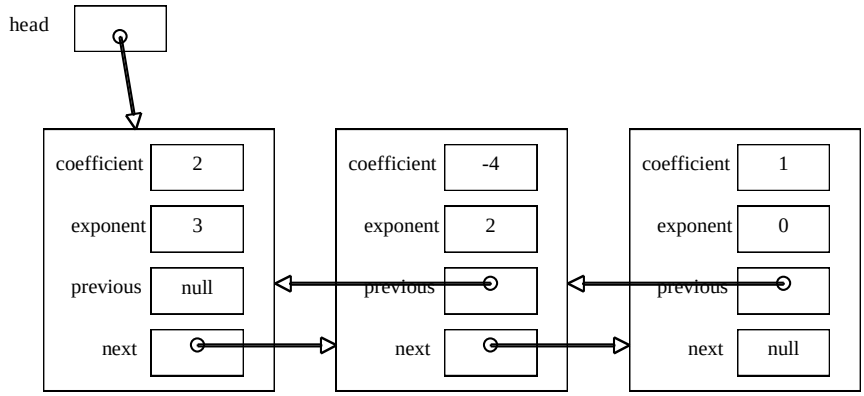
A polynomial, $P(x)$, is a mathematical function of the form:

$$P(x) = \sum_{i=0}^{d} a_i x^i$$

The degree, *d*, of the polynomial is the largest exponent in any of the terms of the polynomial. For example, the largest exponent in $P(x) = 2x^3 - 4x^2 + 1$ is 3, meaning that $P(x) = 2x^3 - 4x^2 + 1$ is a cubic polynomial. It is customary to write the term with the largest exponent first, followed by the term with the next largest exponent and so on, with the constant term (which you can think of as the term with exponent 0) appearing last. You may assume for the purposes of this assignment that all the exponents are positive valued or 0.

The constants $a_0, a_1, \ldots, a_d$ are the coefficients of the polynomial. The coefficients can be positive, negative or zero (except for the term with *d* as its exponent – the coefficient of that term cannot be zero). For example, the coefficients of $P(x) = 2x^3 - 4x^2 + 1$ are 2, -4 and 1. Note that the coefficient of the x term in $P(x) = 2x^3 - 4x^2 + 1$ is 0.

In the first part of this assignment, you are required to develop your own classes to represent polynomials in Java as double linked lists. The diagram below illustrates how the polynomial $P(x) = 2x^3 - 4x^2 + 1$ is implemented as a linked list.



Notice that in the linked list:

- The list has as many nodes as there are terms in the polynomial.
- Each node in the linked list contains a value for the coefficient and a value for the exponent.
- The nodes in the linked list are ordered. Nodes with a higher exponent precede nodes with a lower exponent (this is in keeping with the convention that terms with higher exponent are written first).
- The constant term has an exponent of 0.
- Other than the first and last nodes in the linked list, each node references the previous node and the next node in the list.

- The reference to the previous node in the head of the list is set to null.
- The reference to the next node in the tail of the list is set to null.

In the second part of this assignment you will also need to implement the methods necessary to add, subtract and multiply any 2 polynomials. Recall that only the 'like' terms (the terms with the same variables and exponents) of a polynomial are added or subtracted. We commonly refer to the process of adding or subtracting 'like' terms as 'collecting like terms'. For example, if $P_1(x) = 4x^2 + 3x + 2$ and $P_2(x) = x^2 + 2x + 1$ then

$$\begin{aligned} P_1(x) + P_2(x) &= 4x^2 + 3x + 2 + x^2 + 2x + 1 \\ &= 4x^2 + x^2 + 3x + 2x + 2 + 1 \\ &= 5x^2 + 5x + 3 \end{aligned}$$

To multiply 2 polynomials each term of the first polynomial is multiplied by each term in the second. You then need to collect like terms so that there is only one term for each exponent. For example, if $P_1(x) = x^2 + 3x$ and $P_2(x) = 2x + 1$ then

$$\begin{aligned} P_1(x) \times P_2(x) &= \left(x^2 + 3x\right) \times \left(2x + 1\right) \\ &= x^2 \times 2x + x^2 \times 1 + 3x \times 2x + 3x \times 1 \\ &= 2x^3 + x^2 + 6x^2 + 3x \\ &= 2x^3 + 7x^2 + 3x \end{aligned}$$

Three classes are provided for you: PolyList, PolyNode, and PolyOpps. PolyList contains the methods and fields for creating a double linked list of polynomial terms. PolyNode defines the fields of the nodes in the linked list. PolyOpps contains the methods for adding, subtracting and multiplying polynomials. You may add extra classes and you may add your own variables and methods to the classes supplied. You MUST NOT change the existing variables and method signatures. If you change them the automated marking program cannot check your submission and you will lose marks as a result.

Part A: Representing a polynomial as a linked list

1. Write code for the addNode method in the PolyList class that inserts the new node in its correct position in linked list (that is, all nodes before the new node in the list have larger exponents and all nodes after the new node have smaller exponents). For example, addNode should add a node for the x^2 term after the node for the x^3 term in the list but before the node for the x term. If the list is empty then addNode should set the node as the head of the list. addNode should return true if it successfully added the new node to the linked list. It should return false and leave the linked list unchanged if the list already contains a node with the same exponent. (15 marks)

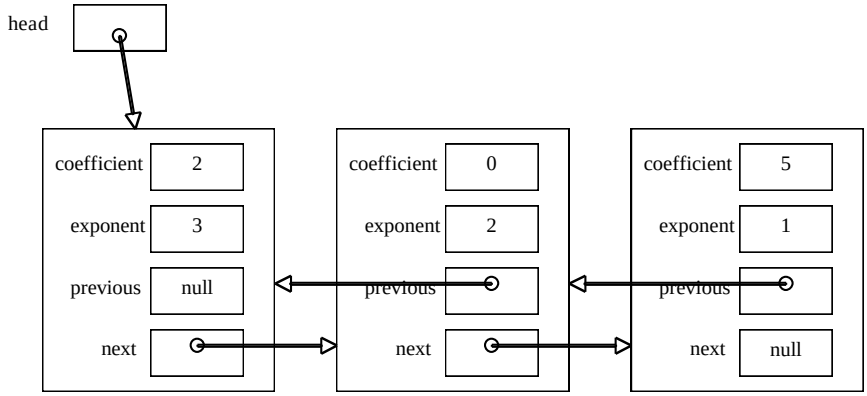
2. Add code to the getDegree method that returns the degree of the polynomial. If the linked list is empty then getDegree should return -1. (5 marks)

3. Implement the getNode method in the PolyList class. getNode should return a reference to this node in the linked list that contains the specified exponent. getNode should return null if no node in the list has the specified exponent. (10 marks)

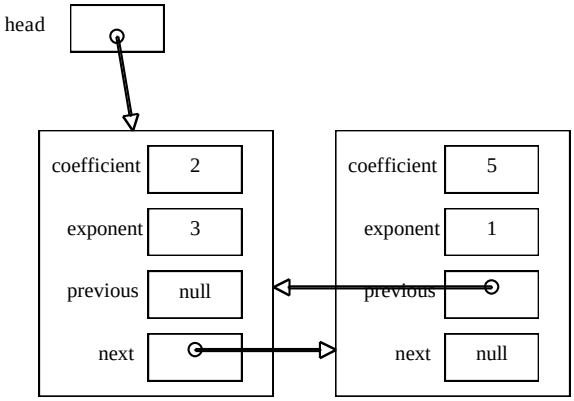
4. Write code for the reduce method in the PolyList class that removes nodes from the linked list with a coefficient of 0. The reduce method should leave the linked list unmodified if there are

no nodes with a coefficient of 0. For example, the reduce method removes the middle node that has a coefficient of 0 from the linked list below. (14 marks)

Linked list before reduce method:



Linked list after reduce method:



5. Implement the toString method so that it returns a string representation of the polynomial. **This must be completed for the testing program to correctly display**. For a polynomial of degree *d*, toString should return a string in the form:

$$a_d x \wedge d + a_{d-1} x \wedge (d-1) + \ldots + a_0 x \wedge 0$$

For example, your toString method should return the string: "$2x \wedge 3 + 1x \wedge 2 + 6x \wedge 1 + 3x \wedge 0$" to represent the polynomial $P(x) = 2x^3 + x^2 + 6x + 3$. (10 marks)

Part B: Adding, subtracting and multiplying polynomials

1. Implement the addPolys method of PolyOpps to add any 2 polynomials. Note that the polynomials may not have the same degree. If either method argument is null, or if either input polynomial contains one or more negative exponents then addPolys should throw an IllegalArgumentException. If the method arguments are valid then addPolys should add the 2 polynomials and return a reference to a new linked list containing the resultant polynomial. (14 marks)

2. Implement the subtractPolys method of PolyOpps to subtract any 2 polynomials. If either method argument is null, or if either input polynomial contains one or more negative exponents then subtractPolys should throw an IllegalArgumentException. If the method arguments are valid then subtractPolys should subtract the 2 polynomials and return a reference to a new linked list containing the resultant polynomial. (14 marks)

3. Write code in the multiplyPolys method of PolyOpps to multiply any 2 polynomials. If either method argument is null, or if either input polynomial contains one or more negative exponents then multiplyPolys should throw an IllegalArgumentException. If the method arguments are valid then multiplyPolys should multiply the 2 polynomials and return a reference to a new linked list containing the resultant polynomial. (18 marks)

   Hint: To multiply 2 polynomials multiply each term in the first polynomial by each term in the second polynomial. Remember that you will need to collect like terms so that there is at most one node for each exponent in the resultant polynomial.

Total Marks: 100