# **Project #1: Algebra and Calculus on Polynomials**

**Due Date: 11:59 PM-March 4, 2019[[1]](#footnote-0)**

**Objectives**

1. Understand the design, implementation and use of the ArrayList container class.
2. Gain experience implementing abstract data types using already developed data structures.
3. Gain experience with object-oriented programming concepts.

## **Overview**

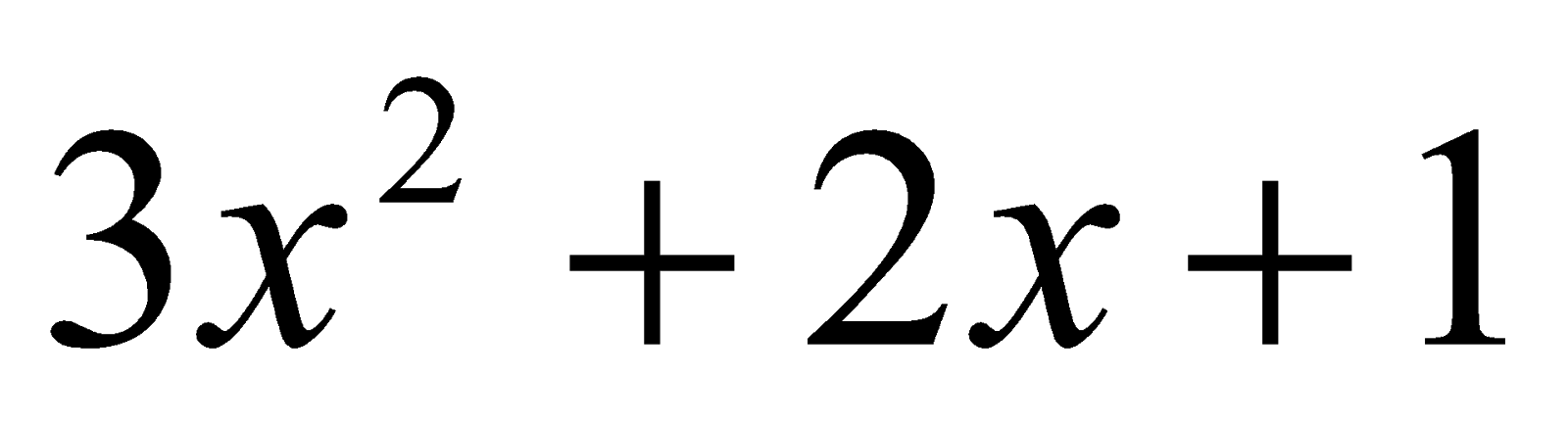
You will implement and test a polynomial class, using the List ADT as the container to store the terms in a polynomial. The List is a generalization and combination of the IndexList and the SimpleList that were discussed in class. You will implement two versions: a) with dynamic arrays, and 2) with a singly-linked list that has a dummy header.

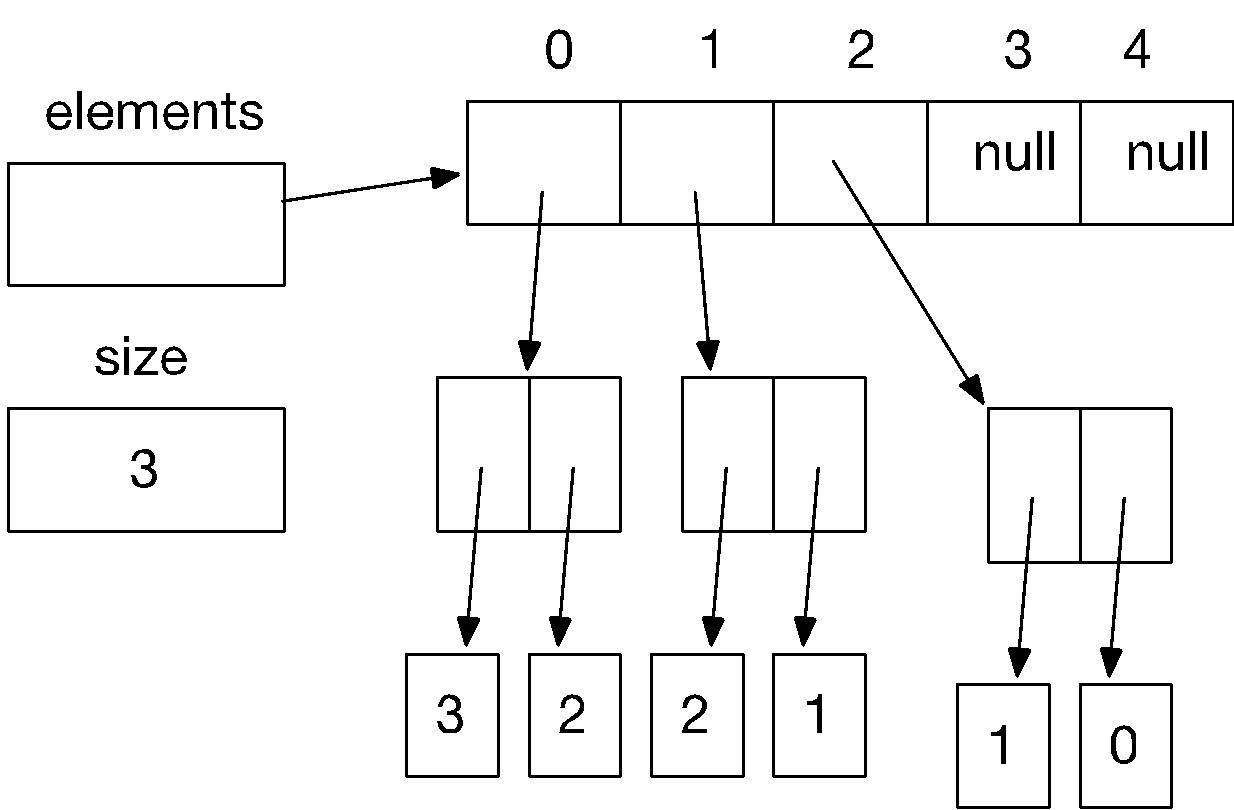
As mentioned before, the list will be used to store the terms in a polynomial. The general form of a polynomial is as follows: **P(x) = anxn+an-1xn-1+...+a1x+a0**. Here, each term has a coefficient, denoted as ai, and a exponent *i*, which represent the power of the variable x. Your polynomial class must implement the following operations:

1. Addition – given two polynomials P1 and P2, compute the polynomial P3=P1+P2.
2. Subtraction – given two polynomials P1 and P2, compute the polynomial P3=P1-P2.
3. Multiplication – given two polynomials P1 and P2, compute the polynomial P3=P1\*P2.
4. Scalar Multiplication - given a polynomial P, multiply it by a constant *c*, and return it as a new polynomial.
5. Derivative – given a polynomial P, finds its derivative.
6. Indefinite integral – given a polynomial P, finds its indefinite integral (anti-derivative).
7. Definite integral – given a polynomial P, evaluate its definite integral over an interval [a,b].
8. Degree – given a polynomial P, find its degree (the largest exponent in any term).
9. Evaluate – given a polynomial P,evaluate it at value *x,* to compute y=P(x).
10. String Converter – given a polynomial *P*, return its string representation as suggested in the following examples:

|  |  |
| --- | --- |
| Polynomial | Corresponding String |
| 3x5+4x2+(-3)x+6 | 3x^5+4x^2-3x+6 |
| -5x100+4x5+(-1) | -5x^100+4x^5-1 |

**Implementation with you own ArrayList**

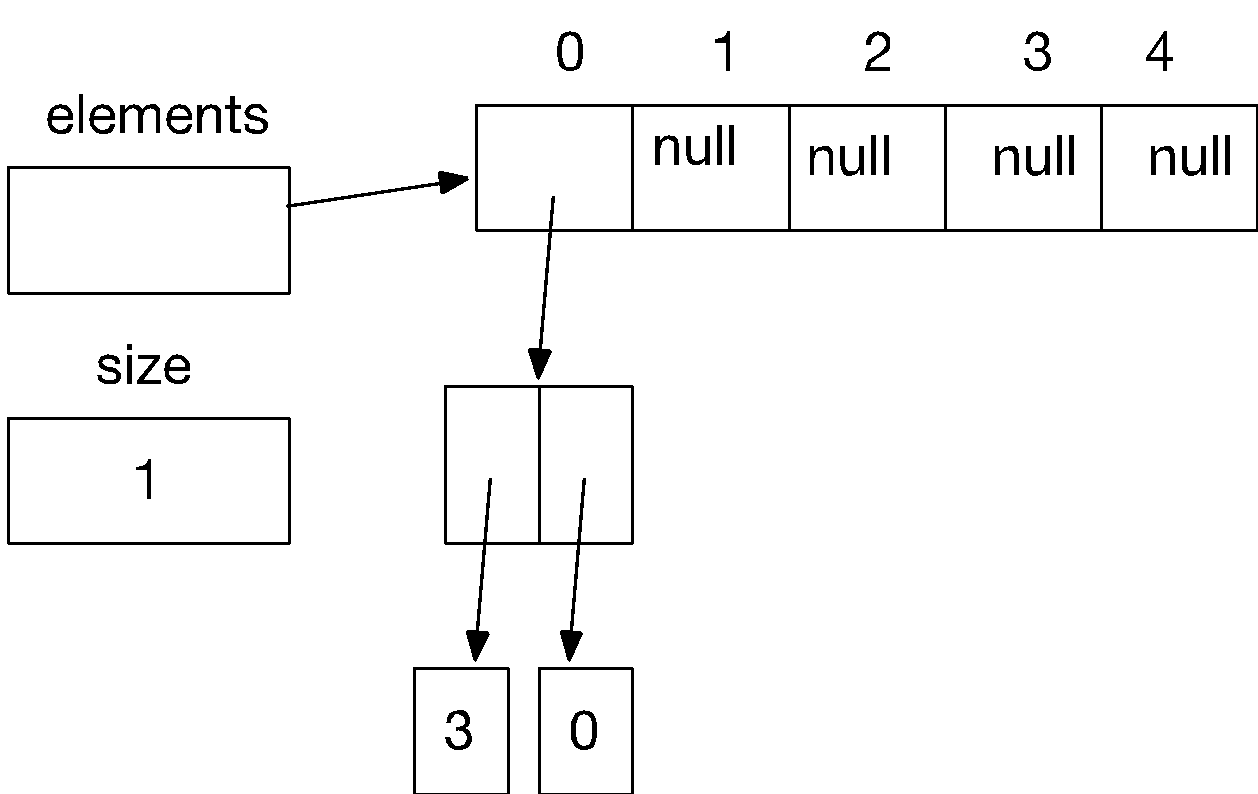
You must implement an ArrayList container class, which implements the List ADT. This List ADT will be provided in a GitLab repo has that starter and Junit testing files. The ArrayList container will store the terms in the polynomial, in decreasing order of exponent. Thus, each element in the ArrayList represents a term in the polynomial. For example, if we need to represent the following polynomial: , then the organization of the ArrayList container associated with the polynomial class should look like this (assuming initial capacity of 5):

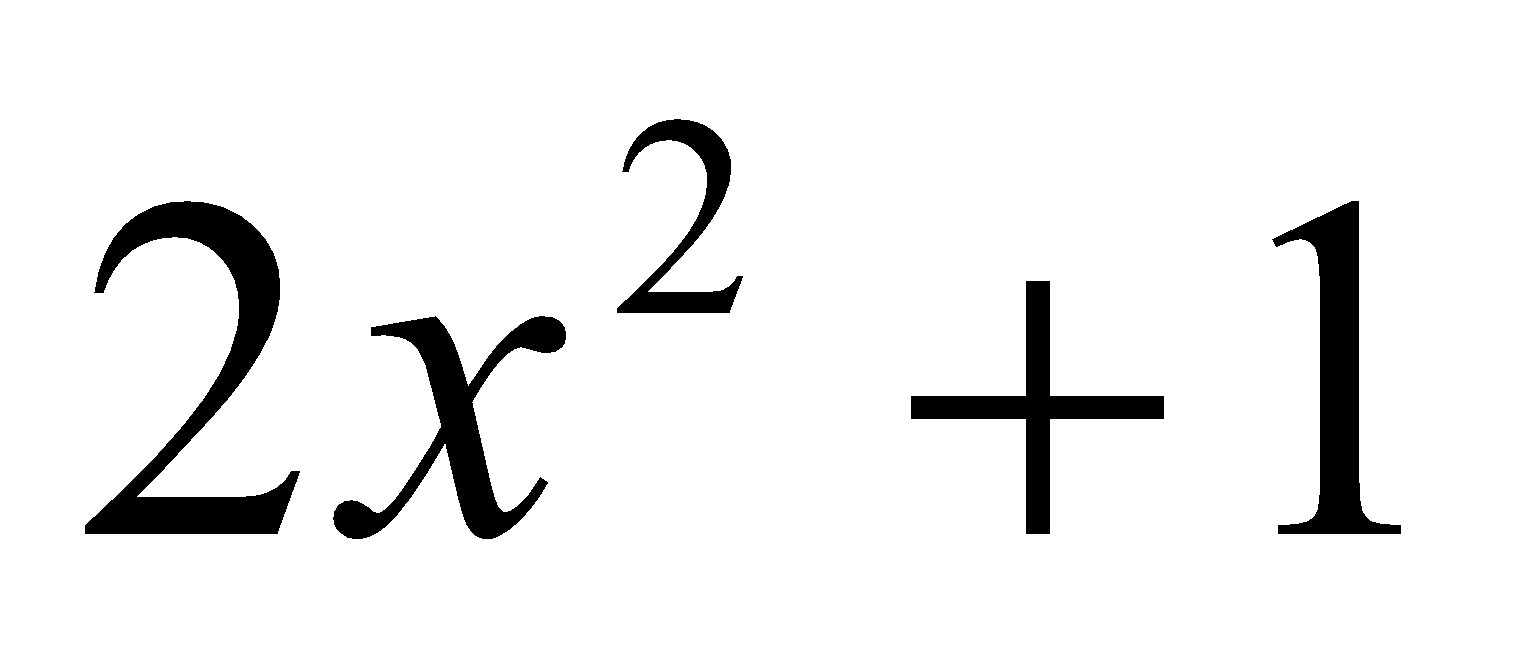


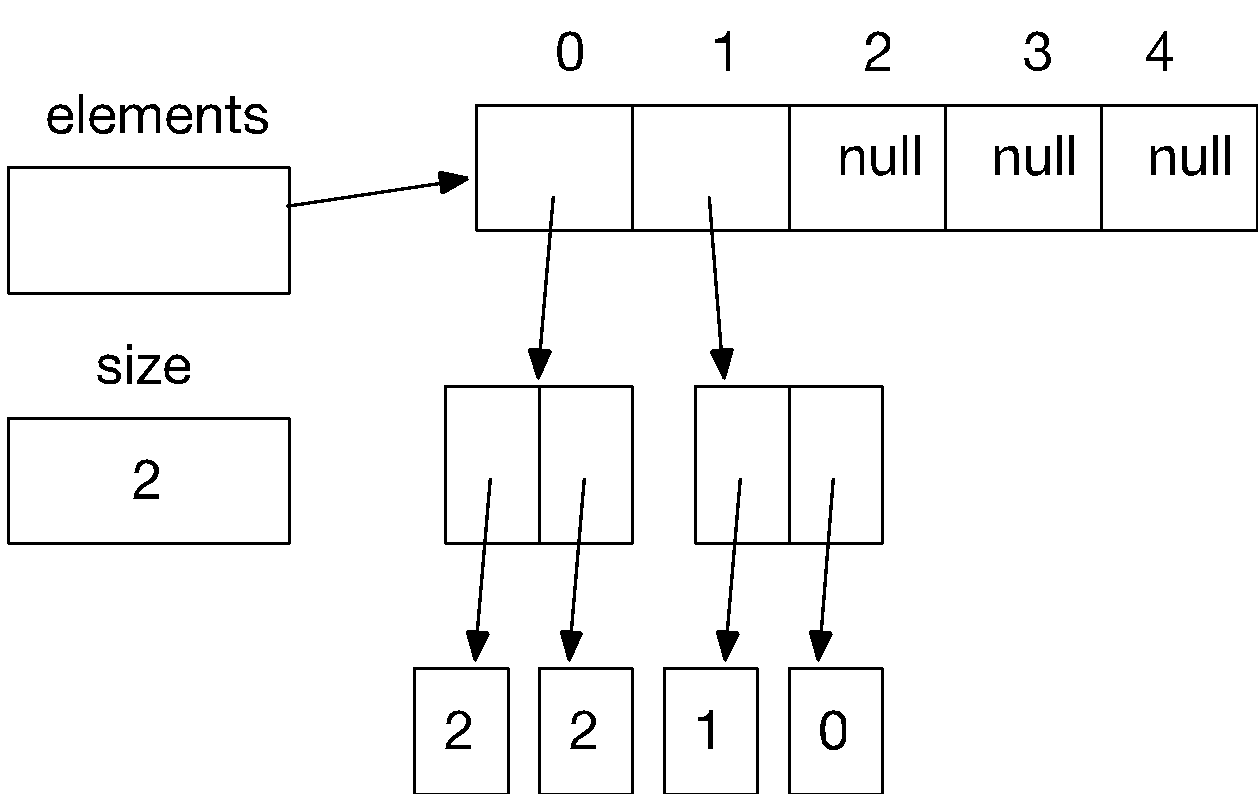
Notice that each element array is a reference to a term object. Each terms holds two values: the coefficient of the term, and the exponent. The project repo has an interface named Term, and you will write a term class that implements this interface.

**In your implementation, you cannot store terms containing a coefficient equal to zero.** The only exception is the case in which the polynomial correspond to P(x) = 0, meaning that the polynomial is just the number 0. When you implement your mathematical operations you must make sure you do not add terms to the polynomial that are zero. Again, the only exception is when the resulting polynomial is the value 0.

To clarify, this point consider the following expression: (2x + 1) – (2x – 2). In this case, the resulting polynomial will be 3, and the representation will be (assuming initial capacity of 5):



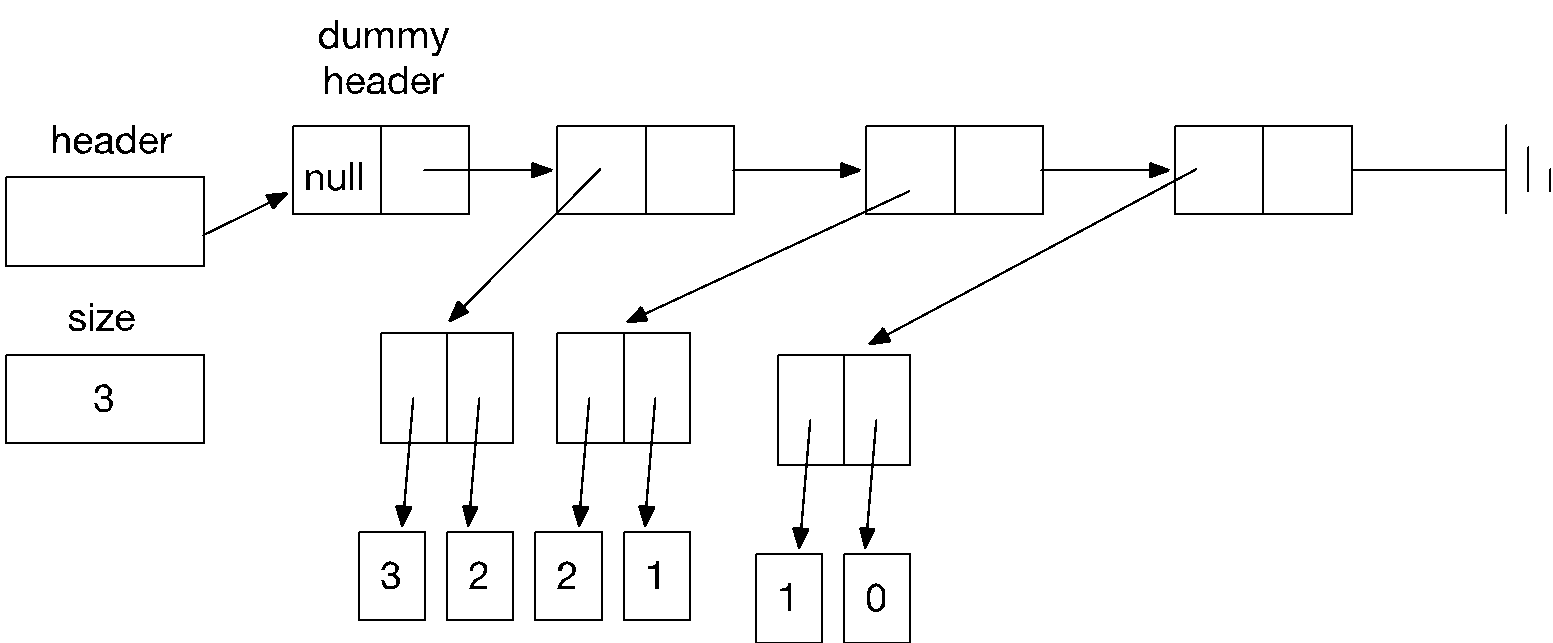
As you can see, the terms with variable x cancel out, and there is no need to represent 0x in the polynomial. Likewise, there is no need to represent the term corresponding to x raised to the first power in this polynomial: 

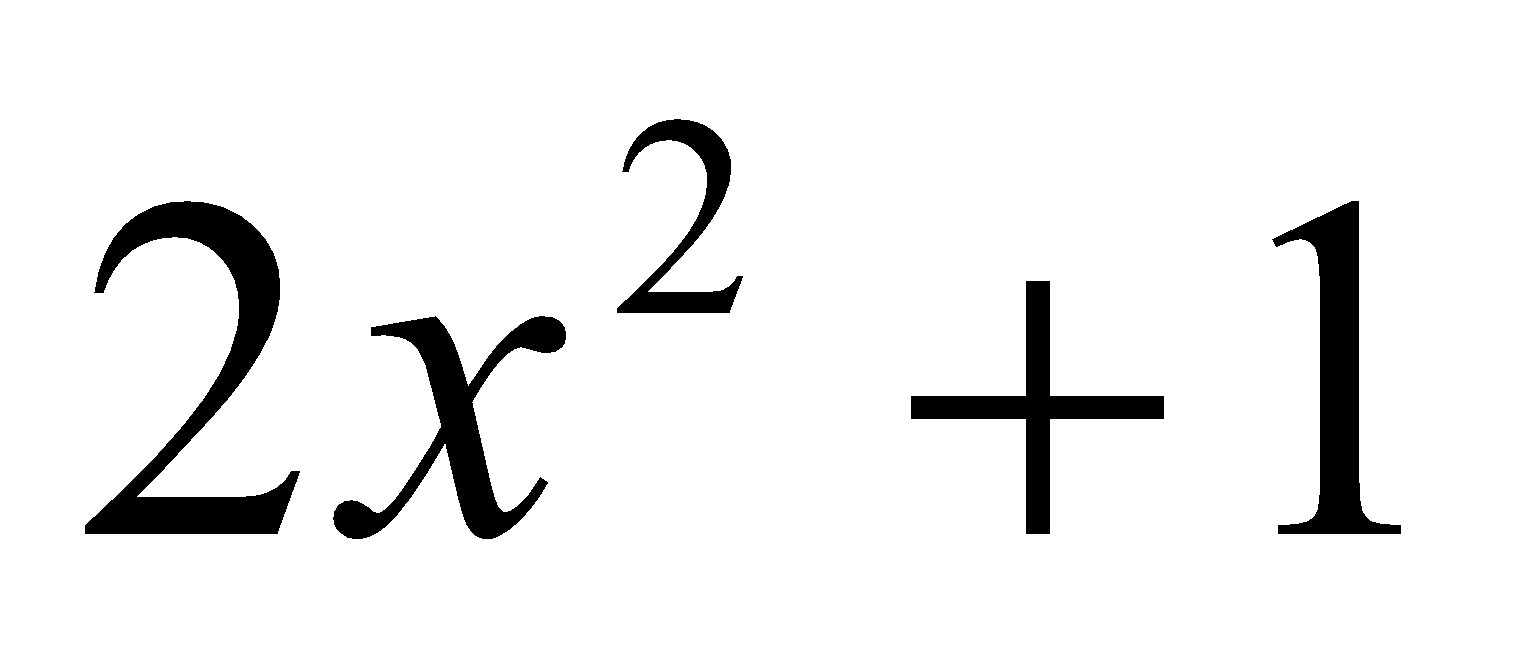


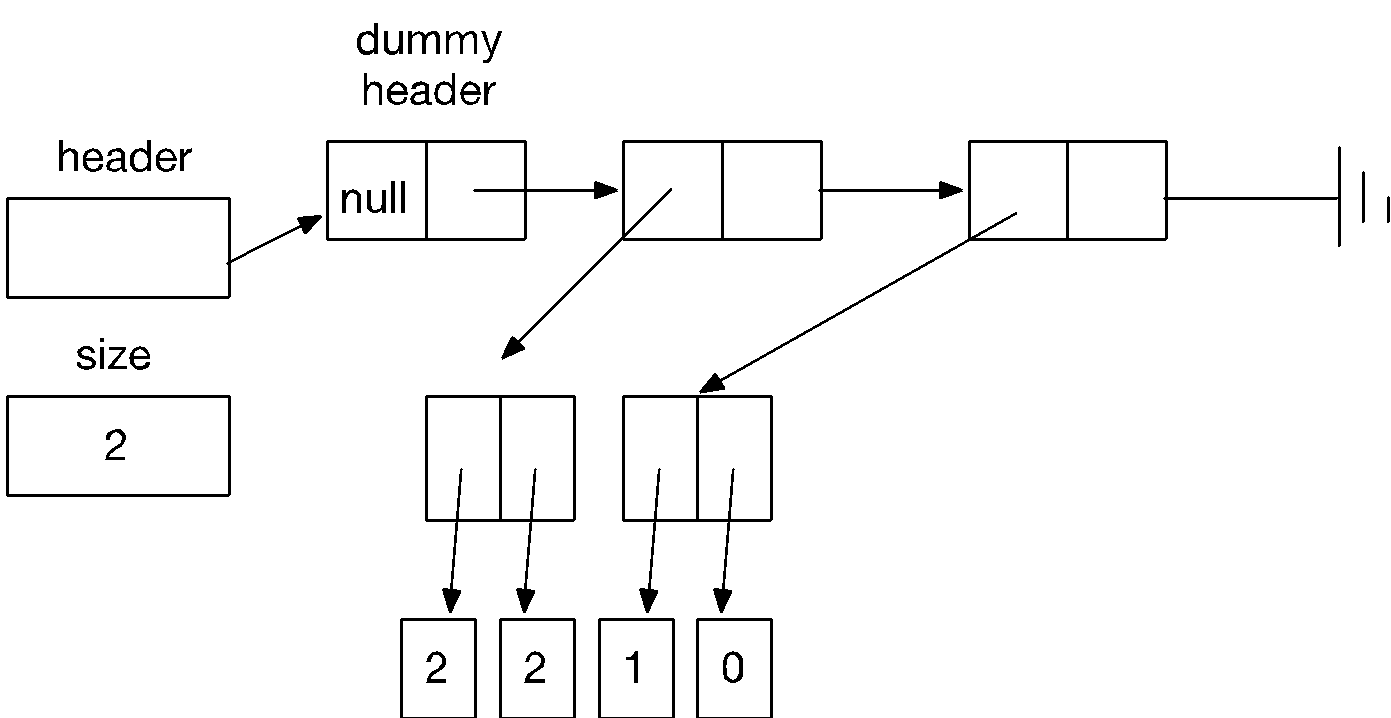
In this case, the term corresponding to a*x* is not represented since the coefficient is zero.

**Implementation with you own SinglyLinkedList**

You must implement an **SinglyLinkedList** container class, which implements the List ADT. This List ADT will be provided in a GitHub repo has that starter and Junit testing files. The **SinglyLinkedList** container will store the terms in the polynomial, in decreasing order of exponent. Thus, each element in the **SinglyLinkedList** represents a term in the polynomial. For example, if we need to represent the following polynomial: *3x2+2x+1*, then the organization of the **SinglyLinkedList** container associated with the polynomial class should look like this (assuming initial capacity of 5):



The same rules for 0 terms apply to the implementation with **SinglyLinkedList.** There is no need to represent the term corresponding to x raised to the first power in this polynomial: :



**What to turn in**

The following repo has the starter project:

<https://gitlab.com/manuelr417/p1starter.git>

**List**

You will be given a List interface and a ListFactory interface. The List inferface specifies the operations in a List. You will implement two concrete clases: ArrayList which implements the list with a dynamic array, and SinglyLinkedList which implements the list with a singly-linked list with a dummy header.

The ListFactory provides a method to create instances of a List. You will implement an ArrayListFactory and a SinglyLinkedListFactory. The idea is that the polynomial is written in terms of the interface List, and you use the proper factory to create either an ArrayList or a SinglyLinkedList.

**Polynomials**

You will be given two interfaces that specify what a polynomial and a term must do. These interfaces are called Polynomial.java and Term.java respectively. You implementation will consist of adding Java code to implement two modules: PolynomialImp.java (which implements interface Polynomial) and TermImp.java (which implements Term.java). There is also a TermListFactory class **that you cannot modify.** Your PolynomialImp keeps track of a list of terms and a ListFactory. It uses the factory create the list of terms. TermListFactory tells PolynomialImp which factory to use.

**YOU CANNOT BIND PolynomialImp to either ArrayList or SinglyLinkedList. The test cases will test if the implementation can be changed or not.**

**JUnit Files**

In addition you will receive six JUnit files that contain the test case for the project. **You project must pass Test1 and Test 4 to be considered a running program.** Otherwise you will get a score of at most 30 points.

**NOTE: Projects that do not use List as implemented with your ArrayList and SinglyLinkedList will be considered not running and you will get a score of 0 in the project.**

**List of Files to submit**

From the starter project you will create a new GitLab repo for your project. The project must be organized as the starter project.

The list of files is a follows:

1. List.java – List interface
2. ArrayList – YOUR implementation of the List with a dynamic array
3. SinglyLinkedList.java – YOUR implementation of the List with a singly linked list with dummy header.
4. ParameterCheck.java - utility class to check bounds and null arguments.
5. ListFactory – interface for the class that creates lists
6. ArrayListFactory – YOUR implementation of the factory for ArrayList
7. SinglyLinkedListFactory - YOUR implementation of the factory for SinglyLinkedList
8. Term.java – interface that defines operations in a polynomial term.
9. Polynomial.java – interface that defines operations in a polynomial.
10. TermImp.java – YOUR implementation of Term.java
11. PolynomialImp.java – YOUR implementation of Polynomial.java
12. TermListFactory.java – class used to create the list terms
13. Test1.java – JUnit Test Case 1 - ArrayList: **Your implementation must pass this test case without errors to be considered a running program.**
14. Test2.java – JUnit Test Case 2 - ArrayList: Second test case
15. Test3.java – JUnit Test Case 3 - ArrayList: Third test case
16. Test4.java – JUnit Test Case 4 - ArrayList: **Your implementation must pass this test case without errors to be considered a running program.**
17. Test5.java – JUnit Test Case 5 - ArrayList: Second test case
18. Test6.java – JUnit Test Case 6 - ArrayList: Third test case

By passing test cases 1-6 you will earn 60 pts in the project. The rest of the points will be awarded upon passing other test cases that the professor will use, plus verifying the documentation of the code.

You can go to the class web page and download a zip file containing all these files. Just access the link named Projects, and download the sources files associated with the link: *Project #1 – Polynomials*.

|  |
| --- |
| PROJECT DUE DATE: **11:59 PM-March 4, 2019** |

1. Specifications written by Dr. Manuel Rodríguez Martínez. [↑](#footnote-ref-0)