COP 3337 **Assignment 6: Customized Linked List** **Fall 2017**

Pestaina **Due: Sunday, December 10**

Overview

A polynomial is a sequence of polynomial terms in descending exponent order.

Example 1: 4

Example 2: $2x^3 - x^2 + 5x - 2$

Example 3: $2x^7 - 5x^5 + 3x^4 - 15x^2 + 19x - 6$

The Example 2 polynomial terms may be represented as: (3, 2), (2, -1), (1, 5), (0, -2)

Specific Requirements

1. Write a class *PolynomialTerm* to represent one term of a Polynomial, each with a non-negative integer *exponent* and a non-zero integer *coefficient*:
   - Instance variables of type **int** for the *exponent* and *coefficient*, and accessors.
   - A parameterized constructor that enforces the constraints described above
   - Method *value(..)* to evaluate a *PolynomialTerm* for a given **int** value (of *x*).
   - Method *plus(..)* to return the sum of two *PolynomialTerms*.
   - Method *times(..)* to return the product of two *PolynomialTerms.*
   - Implements *Comparable* based on *exponent*s only.
   - Override *toString()*.

2. Write a class *Polynomial* to represent a Polynomial:
   - A customized linked list to store *PolynomialTerm*s in descending order. The only instance variable provides a reference to the node storing first term.
   - A parameter-less constructor that creates the 0-polynomial (no terms).
   - A constructor ***public*** *Polynomial**(int**[] data)*.The *data* parameter is an array of alternating *exponents* and *coefficients*; each pair of consecutive **ints** defines one *PolynomialTerm*. E.g. [1, 5, 3, 2, 0, -2, 2, -1] for Example 2 above. The terms may be in any order, but always with *exponent* first then *coefficient*.
   - Helper *insert(PolynomialTerm term)* to insert a new *PolynomialTerm*; throw an exception if the exponent of the new term matches an existing one.
   - Method *isZero()* to return **true** iff a *Polynomial* is the zero-polynomial.
   - Method *value(* **..** *)* to evaluate a *Polynomial* for a given **int** value (of x).
   - Method *plus(..)* to return the sum of a pair of *Polynomials*.
   - Method *times(..)* to return the product of a pair of *Polynomials*.
   - Override *toString()*.

Algorithm Notes

1. Adding *PolynomialTerm*s: $(e, c_1) + (e, c_2) = (e, c_1 + c_2)$. The exponents must be the same. If the coefficient sum $c_1 + c_2 = 0$, the sum of the terms is **null**.
2. Multiplying *PolynomialTerm*s: $(e_1, c_1) * (e_2, c_2) = (e_1 + e_2, c_1 * c_2)$.
3. Adding *Polynomials*: Combines like terms – add terms with the same exponent.
4. Multiplying *Polynomials*: Let $P(x) = p_1(x) + P'(x)$, $p_1(x)$ the 1st term, $P'(x)$ the rest.
   Then, $P(x) * Q(x) = p_1(x) * Q(x) + P'(x) * Q(x)$.
5. **Consider providing recursive implementations of the *Polynomial* methods**.