CMSC 350 Project 2

The second programming project involves writing a program that examines a file of polynomials and determines whether the polynomials in that file are in strictly ascending order using two different methods of comparison.

Each line of the input file will contain one polynomial. On each line will be the terms of the polynomial. Each term will be represented as a pair of values. The first element of that pair is a real value that represents the coefficient and the second an integer value, which is its corresponding exponent. For example, 5.6 3 4 1 8.3 0 represents the polynomial $5.6x^3 + 4x + 8.3$. They are intended to be written from the highest exponent to the lowest, but ensuring that is true is a program requirement. Exponents with zero coefficients will be omitted

The program for this project should consist of four outer classes. The `Polynomial` class is the first of those four. Instances of the `Polynomial` class should define an individual polynomial. `Polynomial` objects should be represented internally by a singly linked list. Each node of that linked list should contain one term of the polynomial consisting of its coefficient and exponent. You are not permitted to use the predefined Java `LinkedList` class, but instead must create the nodes of the linked list as instances of a static nested class inside the `Polynomial` class. The `Polynomial` class must implement both the `Iterable` and `Comparable` interfaces. It must have four public methods.

1. A constructor that accepts a string that defines one polynomial in the same format as provided in the input file.
2. A `compareTo` method that compares two polynomials. If the two polynomials have different highest order exponents, the one with the highest exponent is the greatest. If their highest exponents are the same, their coefficients are compared. If two polynomials have the same highest order exponent with the same coefficients the next highest exponent is examined, and so on.
3. An `iterator` method that produces an iterator the iterates across the terms of the polynomial from highest exponent to lowest and returns and an object that contains only the coefficient and exponent of the next term.
4. A `toString` method that converts a polynomial to a string. Terms with 0 coefficients should be omitted entirely. The exponent of the term with an exponent of 1 should omit the exponent and the term with exponent 0 should omit the variable *x* as well. As an example, the polynomial "5.6 3 4 1 8.3 0" should be converted to the following string `"5.6x^3 + 4x + 8.3"`.

The second class is `InvalidPolynomialSyntax`, which defines an unchecked exception that contains a constructor that allows a message to be supplied. It is thrown by the constructor of the `Polynomial` class should the supplied string contain coefficients or exponents of an improper type or should the exponents fail to be listed in strictly descending order.

The third class is `OrderedList`, which is a utility class that contains two overloaded implementations of a method named `checkSorted`, which determines whether a `List` object, supplied as a parameter, is in strictly ascending order. Both methods should be class (static) methods. The first of the overloaded methods should accept a list that contains elements that implement `Comparable`. The second should instead be supplied an additional parameter that is an object of a class that implements the `Comparator` interface. Refer to the signatures of the two `sort` methods in the predefined Java `Collections` class as a model for how these two methods should be defined. Do not duplicate code, but instead ensure that first overloaded method calls the second..

The fourth outer class is the class that contains the main method. The main method should allow the user to select the input file from the default directory by using an object of the `JFileChooser` class. It should then populate an `ArrayList` of objects of type `Polynomial` as it reads in the lines of the file. As the polynomials are read in, they should also be displayed in the format provided by the `toString` method. Should the `InvalidPolynomialSyntax` exception be thrown, it should be caught and a `JOptionPane` message should be displayed containing the reason for the invalid syntax. The list of polynomials should be then checked to see whether it is in sorted order according to two orderings. The first check is the one that uses the `compareTo` method of the `Polynomial` class. We refer to this first ordering as the strong order. The second results from the use of a comparator. We refer to it as the weak order. It should display whether it fails to be sorted by either of both of the two orderings or if it is sorted according to both.

Inside the main class, a named lambda expession should be defined that implements the `Comparator` interface. Its `compare` method should consider only the exponents and not the coefficients in comparing the polynomials, much like how functions are categorized by Big-O.

As an example, consider the following list of polynomials:

$4.0x^3 + 2.5x + 8.0$
$5.0x^4 + 5.0$
$4.5x^4 + 5.7x^2 + 8.6$

This list fails to be sorted by the strong order that considers both the coefficients and exponents because the third polynomial is less than the second because of the coefficients on the x^4 terms. The coefficient on that term in the second polynomial is 5.0, which is greater than 4.5, the coefficient on the corresponding term in the third polynomial.

They are sorted by the weak order that considers only the exponents, however. The third polynomial is greater than the second because the third has a x^2 term and the second one does not.

You are to submit two files.

1. The first is a `.zip` file that contains all the source code for the project. The `.zip` file should contain only source code and nothing else, which means only the `.java` files. If you elect to use a package the `.java` files should be in a folder whose name is the package name. Every outer class should be in a separate `.java` file with the same name as the class name. Each file should include a comment block at the top containing your name, the project name, the date, and a short description of the class contained in that file.
2. The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:
   a. A UML class diagram that includes all classes you wrote. Do not include predefined classes. You need only include the class name for each individual class, not the variables or methods
   b. A test plan that includes test cases that you have created indicating what aspects of the program each one is testing
   c. A short paragraph on lessons learned from the project

Grading Rubric

Criteria	Meets	Does Not Meet
Design	**20 points**	**0 points**
	Polynomial class implements Comparable and Iterable interfaces(6)	Polynomial class does not implement Comparable or Iterable interfaces (0)
	Polynomial object implemented with a singly linked list(6)	Polynomial object not implemented with a singly linked list (0)
	OrderedList class provides required methods and does not duplicate code (5)	OrderedList class not provide required methods or duplicates code(0)
	Unchecked exception class included(3)	Unchecked exception class not included (0)
Functionality	**60 points**	**0 points**
	Accepts input file as specified and populates array list (15)	Does not accept input file as specified or does not populate array list(0)
	Detects polynomials with invalid syntax and correctly displays error (10)	Does not detect polynomials with invalid syntax or does not correctly display error (0)
	Displays polynomials in the required format as they are read in (15)	Does not display polynomials in the required format as they are read in (0)
	Correctly determines whether a polynomial file is sorted by the weak order (10)	Does not correctly determine whether a polynomial file is sorted by the weak order (0)
	Correctly determines whether a polynomial file is sorted by the strong order (10)	Does not correctly determine whether a polynomial file is sorted by the strong order (0)
Test Plan	**10 points**	**0 points**
	Test cases include a file in both strong and weak sorted order (2)	Test cases do not include a file in both strong and weak sorted order (0)
	Test cases include a file in weak but not strong sorted order (2)	Test cases do not include a file in weak but not strong sorted order(0)
	Test cases include a file in neither strong nor weak sorted order (2)	Test cases do not include a file in neither strong nor weak sorted order (0)
	Test cases include files with both kinds of syntax errors (2)	Test cases do not include files with both kinds of syntax errors (0)
	Test cases include a polynomial with exponents of 0, 1 and 2 or more (2)	Test cases do not include a polynomial with exponents of 0, 1 and 2 or more (0)
Documentation	**10 points**	**0 points**
	Correct UML diagram included (3)	Correct UML diagram not included (0)

	Lessons learned included (4)	Lessons learned not included (0)
	Comment blocks included with each Java file (3)	Comment blocks not included with each Java file(0)