# Project 2 Documentation:

# Polynomial Strong and Weak Order

Vincent Testagrossa

University of Maryland Global Campus

CMSC 350: Data Structures and Analysis

Dr. Jim Huskins

June 12th, 2022

# Contents

## [Problem Statement](#_Problem_Statement_1)

## [UML Class Diagram](#_UML_Class_Diagram)

## [Testing](#_Testing)

## [Output](#_Output)

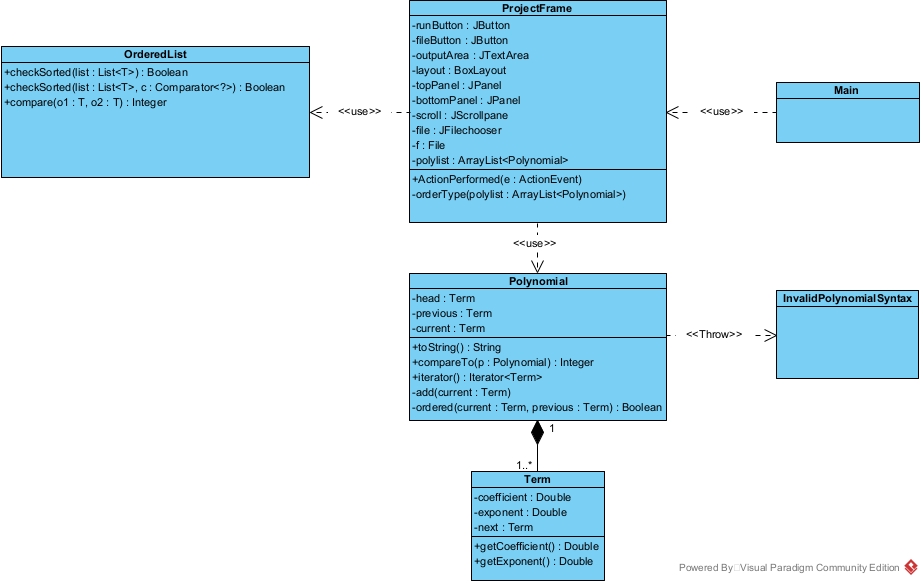
## [Reflection](#_Reflection)

# Problem Statement

Write 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. The program should consist of four outer classes:

1. A *Polynomial* class that defines an individual polynomial. *Polynomial* objects should be represented internally by a singly linked list. Each node contains one term consisting of the coefficient and the exponent. Implements *Iterable* and *Comparable* and has four public methods:
   1. *compareTo* which compares on *Strong Order*
   2. *iterator*
   3. *toString* that omits 0 coefficients, omits the variable *x* with an exponent of 1, omits the exponent with an exponent of 1, and builds the string as such: *5.6x^3 + 4x + 8.3* for the input *5.6 3 4 1 8.3 0*
   4. A constructor that accepts a string that defines one polynomial in the same format as provided in the input file.
2. An *InvalidPolynomialSyntax* unchecked exception that contains a constructor that allows a message to be supplied, thrown by the constructor of *Polynomial* if the exponents aren’t listed in descending order or coefficients/exponents are the improper type
3. An *OrderedList* utility class with an overloaded *checkSorted* method; the first of which takes a *comparable* list object and calls the second, which also takes a *comparator*.
4. The *Main* class with the *Main*  method. Allows the user to choose the file using *JFileChooser*, populates an *ArrayList* of *Polynomial* as it reads from the file, and displays them using the *toString* method. If *InvalidPolynomialSyntax* is thrown, it’s caught and displayed in a *JOptionPane*. The list is then checked using the *compareTo* method for Strong Order and then a named lambda that implements *Comparator* for Weak Order. The program should show whether the list is sorted by *Strong Order* or not and by *Weak Order* or not.

# UML Class Diagram

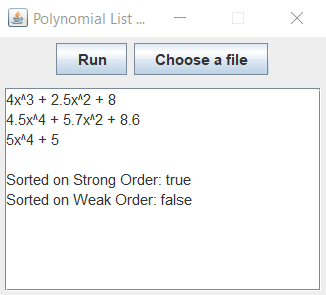


# Testing

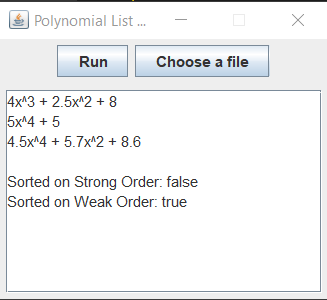
|  |  |  |  |
| --- | --- | --- | --- |
| Case number | Input | Expected Output | Actual Output |
| [Test 1](#_Test_Case_1:) | 4.0 3 2.5 2 8.0 0  4.5 4 5.7 2 8.6 0  5.0 4 5.0 0 | Is sorted by strong comparison: true  Is sorted by weak comparison: false | As expected |
| [Test 2](#_Test_Case_2:) | 4.0 3 2.5 2 8.0 0  5.0 4 5.0 0  4.5 4 5.7 2 8.6 0 | Is sorted by strong comparison: false  Is sorted by weak comparison: true | As expected |
| [Test 3](#_Test_Case_3:) | 5.0 4 5.0 0  4.5 4 5.7 2 8.6 0  4.0 3 2.5 2 8.0 0 | Is sorted by strong comparison: false  Is sorted by weak comparison: false | As expected |
| [Test 4](#_Test_Case_4:) | 5.0 x 5.0 0  4.5 4 5.7 2 8.6 0  4.0 3 2.5 8.0 0 | InvalidPolynomialSyntax | As expected |
| [Test 5](#_Test_Case_5:) | 5.0 4 5.0 5  4.5 4 5.7 2 8.6 0  4.0 3 2.5 2 8.0 0 | InvalidPolynomialSyntax | As expected |
| [Test 6](#_Test_Case_6:) | 4.0 3 2.5 2 8.0 1  4.5 4 5.7 0  5.0 4 5.0 2 | Is sorted by strong comparison: true  Is sorted by weak comparison: true | As expected |
| [Test 7](#_Test_Case_7:) | 5 5 4 4 3  5 5 4 4 3 0 | InvalidPolynomialSyntax | As expected |
| [Test 8](#_Test_Case_8:) | 0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0  0 0 0 0 0 0 | InvalidPolynomialSyntax | As expected |
| [Test 9](#_Test_Case_9:) | 5 5 0 4 0 3 0 2 1 1  5 5 4 4 3 3 2 2 1 1 | Is sorted by strong comparison: true  Is sorted by weak comparison: true | As expected |
| [Test 10](#_Test_Case_10:) | Improper filename | File not found error. | As expected |

# Output

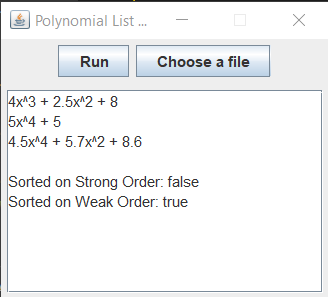
## Test Case 1:



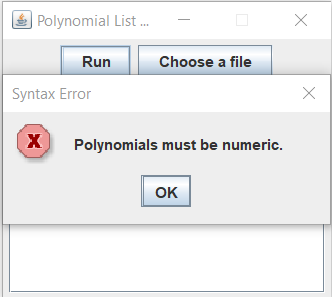
## Test Case 2:



## Test Case 3:



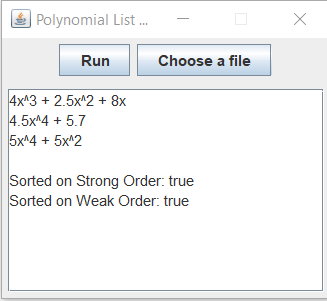
## Test Case 4:



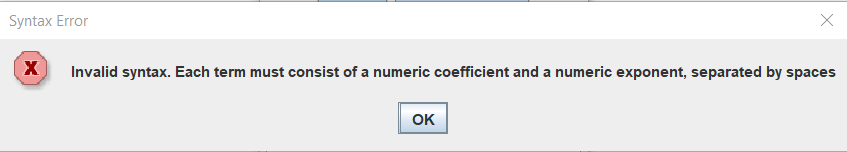
## Test Case 5:

## 

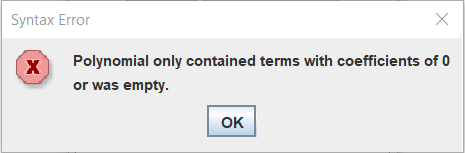
## Test Case 6:



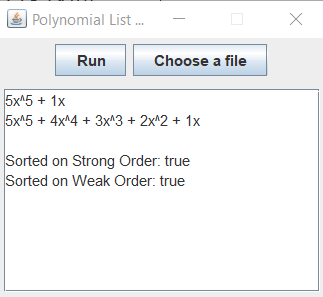
## Test Case 7:



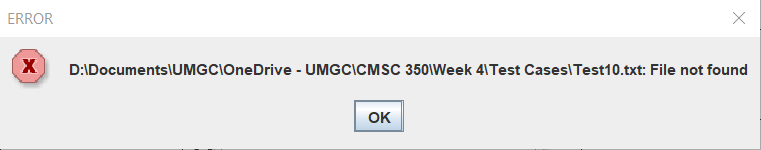
## Test Case 8:



## Test Case 9:



## Test Case 10:



# Reflection

This project has really pushed my knowledge in terms of Java’s abstraction. Having never used generic, I struggled with the *OrderedList* utility class, but I was determined to make it work with more than just the Polynomial type. One other issue I had was with the *Iterator*. I wanted to keep it generic, but kept getting type safety warnings when I was trying to use it to return an individual *Term*. Part of my issue was that I had initially made *Term* a *private* static class, and once I removed the private, I was able to make an *Iterator<Term>* which can return an individual *Term*. I didn’t go back and try and make the inner class more abstract because of time constraints, but once I got that working, my biggest struggle was creating the *Comparator*, which I didn’t think was terribly difficult after all the work done with *OrderedList*. I had dug through the source code for the *sort()* method in an older GNU project and found out how they had made passing a *null comparator* work using a ternary operator, which I now know how to use. I absolutely have gained a lot of value and knowledge from this project.