Project Documentation

Name: Dan Beck

Assignment: Project 2

Date: September 15th, 2020

Problem Statement: A program that examines a file of polynomials and determines whether the polynomials in that file are in ascending order. The program accepts text file and outputs to the console.

Analysis: Equations in raw form used:

Strong - 5.6 3 4 1 9 0

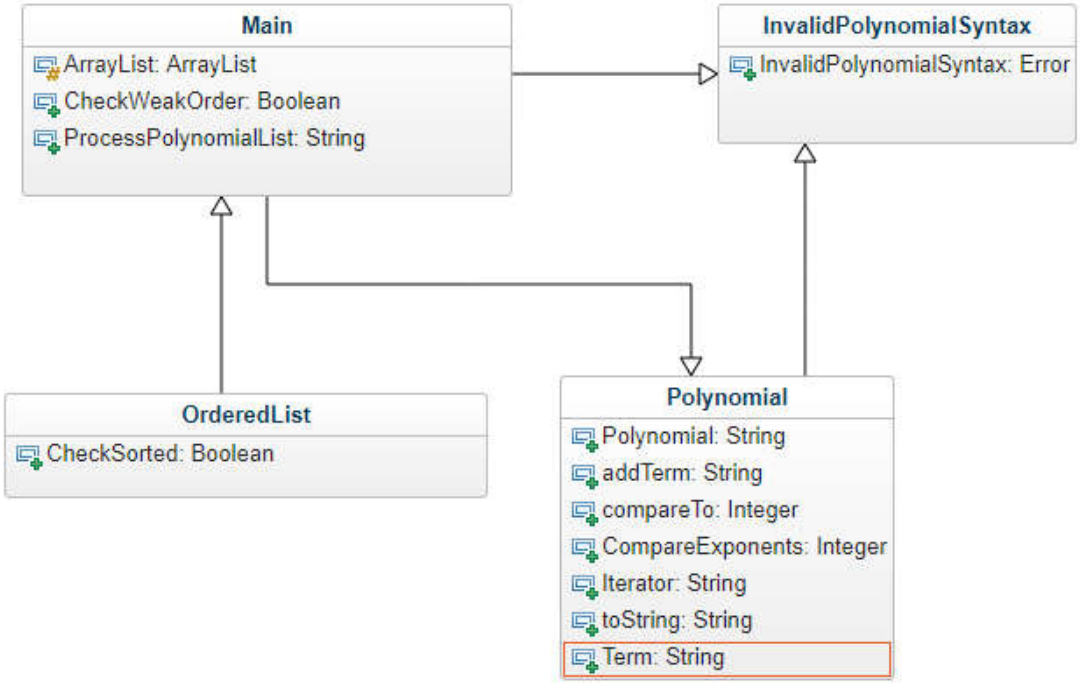
7.8 3 9 2 8.3 1 12 0

8 4 9 3 8.3 2 12 1 3 0

Weak - 7.1 3 3.6 2 8.3 1 6 0

5.6 2 4 1 9 0

Design (for project assignments only):



Code:

```
package BeckProj2;

/* File: Project 2 - Main Class
 * Author: Dan Beck
 * Date: September 15, 2020
 * Purpose: Executes the program. Creates an ArrayList from equations that are
extracted from the file.
 * Checks if the order of the equations are weak.
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Scanner;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class Main
{
    //constructor that creates the list for the equations
    private static List<Polynomial> polynomialList = new ArrayList<>();

    /*****************************************************************************
     * DESCRIPTION: Main
     * Executes the program
     *****************************************************************************/
    public static void main(String[] args)
    {
        processPolynomialList();
    }//end main

    /*****************************************************************************
     * DESCRIPTION: ArrayList<String> fromFile()
     * Allows user to select file
     * Expressions from each line are stored in an Arraylist
     * Returns listOfExpressions
     *****************************************************************************/
    public static ArrayList<String> fromFile()
    {
        // Create ArrayList for the list of expressions
        ArrayList<String> listOfExpressions = new ArrayList<>();

        //Allows user to select file and reads lines from the file
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
        fileChooser.setCurrentDirectory(new File(System.getProperty("user.dir")));
        int status = fileChooser.showOpenDialog(null);
        if (status == JFileChooser.APPROVE_OPTION)
        {
            File file = fileChooser.getSelectedFile();
            try
            {
             //scans each line. Creates one expression from each line
```

```
                Scanner scan = new Scanner(file);
                if (file.isFile())
                {
                    //loop to create the list
                    while (scan.hasNextLine())
                    {
                        String singleExpression = scan.nextLine();
                        listOfExpressions.add(singleExpression);
                    }//end while (scan.hasNextLine())
                }// if (file.isFile())
                scan.close();
            }//end try
            catch (NoSuchElementException nse)
            {
                JOptionPane.showMessageDialog(JOptionPane.getRootFrame(), "The
selected file is empty!");
            }//end catch (NoSuchElementException nse)
            catch (FileNotFoundException fnf)
            {
                JOptionPane.showMessageDialog(JOptionPane.getRootFrame(), "File can
not be found!");
            }//end catch (FileNotFoundException fnf)
        }//end if (status == JFileChooser.APPROVE_OPTION)
        return listOfExpressions;
    }//end public static ArrayList<String> fromFile()

    /*****************************************************************************
     * DESCRIPTION: checkWeakOrder(List<Polynomial> polynomialList)
     * Checks if the list is in weak order
     * Calls Polynomial class
     * Returns boolean
     *****************************************************************************/
    public static boolean checkWeakOrder(List<Polynomial> polynomialList)
    {
      //initially set the arg to true
        boolean isWeakOrder = true;
        Polynomial p = polynomialList.get(polynomialList.size() - 1);

        //loop to compare
        for (int i = polynomialList.size() - 2; i > 0; i--)
        {
            if (p.compareExponents(polynomialList.get(i)) < 0)
            {
                isWeakOrder = false;
            }//end if (p.compareExponents(polynomialList.get(i)) < 0)
        }//end for (int i = polynomialList.size() - 2; i > 0; i--)
        return isWeakOrder;
    }//end public static boolean checkWeakOrder(List<Polynomial> polynomialList)

    /*****************************************************************************
     * DESCRIPTION: processPolynomialList()
     * Generates an ArrayList from the user selected file
     * Calls InvalidPolynomialSyntax class
     * Calls OrderedList class
     * Output checks if the equations are strong ordered
```

```
     * Output checks if the equations are weak ordered
     ********************************************************************************/
    public static void processPolynomialList()
    {
        try
        {
             //Generates ArrayList from file
            ArrayList<String> a = fromFile();

            //Output start
            System.out.println("Equations:");

            //Loop that checks list order
            for (String element : a)
            {
                Polynomial p = new Polynomial(element);
                System.out.println(p);
                polynomialList.add(p);
            }//end for (String element : a)
        }//end try
        catch (InvalidPolynomialSyntax ex)
        {
            JOptionPane.showMessageDialog(JOptionPane.getRootFrame(),
ex.getMessage());
        }//end catch (InvalidPolynomialSyntax ex)

        if(OrderedList.checkSorted(polynomialList) == true)
        {
             //checks if the list is strong ordered
            System.out.println("\nList is Strong Ordered");
        }//end if(OrderedList.checkSorted(polynomialList) == true)
        else if(checkWeakOrder(polynomialList) == true)
        {
             //checks if the list is weak ordered
             System.out.println("\nList is Weak Ordered");
        }//end else if(checkWeakOrder(polynomialList) == true)
        else
        {
             //checks if the list is weak ordered
             System.out.println("\nList is Neither Weak or Strong Ordered");
        }//end else    }//end public static void processPolynomialList()
}//end Main class



package BeckProj2;

/* File: Project 2 - InvalidPolynomialSyntax Class
 * Author: Dan Beck
 * Date: September 15, 2020
 * Purpose:Class that creates InvalidPolynomialSyntax error to be caught in program
 */

public class InvalidPolynomialSyntax extends RuntimeException
{
```

```
    private static final long serialVersionUID = 1L;

    InvalidPolynomialSyntax(String msg)
    {
        super(msg);
    }//end InvalidPolynomialSyntax(String msg)
}//end class InvalidPolynomialSyntax



package BeckProj2;

/* File: Project 2 - OrderedList Class
 * Author: Dan Beck
 * Date: September 15, 2020
 * Purpose: Scans the ordered list and compares values to check how the list is
sorted
 */

import java.util.List;

/*****************************************************************************
 * DESCRIPTION: OrderedList
 * Scans the ordered list and compares values to check how the list is sorted
 *****************************************************************************/
public class OrderedList
{
    public static <T extends Comparable<? super T>> boolean checkSorted(List<T> list)
    {
        boolean listSorted = true;
        for (int i = list.size() - 1; i > 0; i--)
        {
            T current = list.get(i);
            if (!checkSorted(list, current))
            {
                listSorted = false;
            }//end if (!checkSorted(list, current))
        }//end for (int i = list.size() - 1; i > 0; i--)

        return listSorted;
    }//end <T extends Comparable<? super T>> boolean checkSorted(List<T> list)

    private static <T extends Comparable<? super T>> boolean checkSorted(List<T>
list, T current)
    {
        //set values to be compared
        T valueOne = list.get(list.indexOf(current));
        T valueTwo = list.get(list.indexOf(current) - 1);

        if (valueTwo != null)
        {
            return valueOne.compareTo(valueTwo) >= 0;
        }//end if (nextValue != null)
        return true;
```

```
    }//end <T extends Comparable<? super T>> boolean checkSorted(List<T> list, T
current)
}//end class OrderedList



package BeckProj2;

/* File: Project 2 - Polynomial Class
 * Author: Dan Beck
 * Date: September 15, 2020
 * Purpose: Compares the polynomials in the linked list. Iterates from highest to
lowest exponent.
 * Converts the polynomial to a string.
 */

import java.util.Comparator;
import java.util.Iterator;
import java.util.Scanner;

public class Polynomial implements Iterable<Polynomial.Term>, Comparable<Polynomial>
{

      //Sets the comparator
    Comparator<Polynomial> compare;
    private Term begin;

    /*****************************************************************************
     * DESCRIPTION: Polynomial(String fromFile)
     * Uses scanner to scan file from the original file that was selected. Splits
     * the String into individual term nodes and creates a linked list from the
     * extracted terms.
     *****************************************************************************/
    public Polynomial(String fromFile)
    {
        //set head Term to null
        begin = null;

        //Creates scanner to read polynomials from file
        Scanner scan = new Scanner(fromFile);

        try
        {
            while (scan.hasNext())
            {
                addTerm(scan.nextDouble(), scan.nextInt());
            }//end while (scan.hasNext())
            scan.close();
        } //end try
        catch (Exception e1)
        {
            System.out.println(e1.getLocalizedMessage());
            throw new InvalidPolynomialSyntax("Incorrect Syntax. Check inputs!");
        }//end catch (Exception e1)
    }
```

```
    /*****************************************************************************
     * DESCRIPTION: addTerm(double coef, int ex)
     * Checks for negative exponents
     * Sets the coefficient and exponent of the added term
     *****************************************************************************/
    public void addTerm(double coef, int ex)
    {
        if (ex < 0)
        {
            throw new InvalidPolynomialSyntax("Negative exponents are not allowed.
Check inputs!");
        }//end if (ex < 0)
        Term t = begin;
        if (t == null)
        {
             // then Polynomial is empty
            begin = new Term(coef, ex);
            begin.next = null;
        }//end if (t == null)
        else
        {
             //find end by looping to null next link
            while (t.next != null)
            {
                t = t.next;
            }//end while (t.next != null)
            t.next = new Term(coef, ex);
        }//end else
    }//end public void addTerm(double coef, int ex)

    /*****************************************************************************
     * DESCRIPTION: compareTo(Polynomial comparedPolynomial)
     * Compares exponents and coefficients
     *****************************************************************************/
    @Override
    public int compareTo(Polynomial comparedPolynomial)
    {
        Term currentTerm = this.begin;
        Term nextTerm = comparedPolynomial.begin;

        while (currentTerm != null && nextTerm != null)
        {
            // positive if this is larger, negative otherwise
            if (currentTerm.getExponent() != nextTerm.getExponent())
            {
                return currentTerm.getExponent() - nextTerm.getExponent();
            }//end if (currentTerm.getExponent() != nextTerm.getExponent())
            else if (currentTerm.getCoefficient() != nextTerm.getCoefficient())
            {
                if (nextTerm.getCoefficient() > currentTerm.getCoefficient())
                {
                    return -1;
                }//end if (nextTerm.getCoefficient() > currentTerm.getCoefficient())
                else if (nextTerm.getCoefficient() < currentTerm.getCoefficient())
```

```
                {
                    return +1;
                }//end else if (nextTerm.getCoefficient() <
currentTerm.getCoefficient())
            }//end else if (currentTerm.getCoefficient() !=
nextTerm.getCoefficient())

            //resets the values outside of the loop
            currentTerm = currentTerm.getNext();
            nextTerm = nextTerm.getNext();

        }//end while (currentTerm != null && nextTerm != null)

        //returns zero if both are null
        if (currentTerm == null && nextTerm == null)
        {
            return 0;
        }//end if (currentTerm == null && nextTerm == null)

        //if one with more terms than other
        if (currentTerm == null)
        {
            return -1;
        }//end if (currentTerm == null)
        else
        {
            return +1;
        }//end else
    }

    /**************************************************************************
     * DESCRIPTION: compareExponents(Polynomial comparedPolynomial2)
     * Compares 2nd set of exponents and coefficients
     *************************************************************************/
    public int compareExponents(Polynomial comparedPolynomial2)
    {
        Term currentTerm = this.begin;
        Term nextTerm = comparedPolynomial2.begin;

        while (currentTerm != null && nextTerm != null)
        {
            // positive if this is larger, negative otherwise
            if (currentTerm.getExponent() != nextTerm.getExponent())
            {
                return currentTerm.getExponent() - nextTerm.getExponent();
            }//end if (currentTerm.getExponent() != nextTerm.getExponent())
            else if (currentTerm.getCoefficient() != nextTerm.getCoefficient())
            {
                if (nextTerm.getCoefficient() > currentTerm.getCoefficient())
                {
                    return -1;
                }//end if (nextTerm.getCoefficient() > currentTerm.getCoefficient())
                else if (nextTerm.getCoefficient() < currentTerm.getCoefficient())
                {
                    return +1;
```

```
                }//end else if (nextTerm.getCoefficient() <
currentTerm.getCoefficient())
            }//end else if (currentTerm.getCoefficient() !=
nextTerm.getCoefficient())

            //resets the values outside of the loop
            currentTerm = currentTerm.getNext();
            nextTerm = nextTerm.getNext();

        }//end while (currentTerm != null && nextTerm != null)

        //returns zero if both are null
        if (currentTerm == null && nextTerm == null)
        {
            return 0;
        }//end if (currentTerm == null && nextTerm == null)

        //if one with more terms than other
        if (currentTerm == null)
        {
            return -1;
        }//end if (currentTerm == null)
        else
        {
            return +1;
        }//end else
    }

    /*************************************************************************
     * DESCRIPTION: Polynomial()
     * Calls Polynomial1 and Polynomial2 to be compared
     ************************************************************************/
    public Polynomial()
    {
        compare = (Polynomial polynomial1, Polynomial polynomial2) ->
polynomial1.compareExponents(polynomial2);
    }//end public Polynomial()

    /*************************************************************************
     * DESCRIPTION: Polynomial(Comparator<Polynomial> compare)
     * Constructor setting the compare variable
     ************************************************************************/
    public Polynomial(Comparator<Polynomial> compare)
    {
        this.compare = compare;
    }//end public Polynomial(Comparator<Polynomial> compare)

    /*************************************************************************
     * DESCRIPTION: Iterator<Term> iterator()
     * Generates an iterator that traverses the terms of a polynomial
     ************************************************************************/
    @SuppressWarnings({ "rawtypes", "unchecked" })
      @Override
    public Iterator<Term> iterator()
    {
```

```
    return new Iterator()
    {

        private Term cur = getHead();

        @Override
        public boolean hasNext()
        {
            return cur != null && cur.getNext() != null;
        }//end public boolean hasNext()

        @Override
        public Term next()
        {
            Term c = cur;
            cur = cur.next;
            return c;
        }//end public Term next()
    };//end new Iterator()
}//end public Iterator<Term> iterator()

/*****************************************************************************
 * DESCRIPTION: Polynomial
 * Uses scanner to scan file from the original file that was selected. Splits
 * the String into individual term nodes and creates a linked list from the
 * extracted terms
 *****************************************************************************/
@Override
public String toString()
{
    StringBuilder expressionBuild = new StringBuilder();

    //checks beginning to avoid adding symbol to begining
    if (begin.coefficient > 0)
    {
        expressionBuild.append(begin.toString());
    }//end if (begin.coefficient > 0)
    else
    {
        expressionBuild.append(" - ").append(begin.toString());
    }//end else

    //then check the other nodes if they are not null
    for (Term t = begin.next; t != null; t = t.next)
    {
        if (t.coefficient < 0)
        {
            expressionBuild.append(" - ").append(t.toString());
        }
        else
        {
            expressionBuild.append(" + ").append(t.toString());
        }
    }//end for (Term t = begin.next; t != null; t = t.next)
    return expressionBuild.toString();
```

```
}//end public String toString()

/*****************************************************************************
 * DESCRIPTION: Polynomial
 * Writes the term to a string
 *****************************************************************************/
static class Term
{
    private double coefficient;
    private int exponent;
    private Term next;

    private Term(double c, int e)
    {
        coefficient = c;
        exponent = e;
        next = null;
    }//end Term(double c, int e)

    private int getExponent()
    {
        return this.exponent;
    }//end int getExponent()

    private double getCoefficient()
    {
        return this.coefficient;
    }//end double getCoefficient()

    private Term getNext()
    {
        return next;
    }//end Term getNext()

    @Override
    public String toString()
    {
        String termString = String.format("%.1f", Math.abs(coefficient));
        if (exponent == 0)
        {
         //no variable
            return termString;
        }
        else if (exponent == 1)
        {
         //do not display exponent
            return termString + "x";
        }
        else
        {
         // display exponent after variable
            return termString + "x^" + exponent;
        }
    }//end public String toString()
}//end static class Term
```

```
    /*************************************************************************
     * DESCRIPTION: Term getHead()
     * Getter for the beginning of equation
     ************************************************************************/
    private Term getHead()
    {
        return begin;
    }//end private Term getHead()
}//end public class Polynomial implements Iterable<Polynomial.Term>,
Comparable<Polynomial>
```

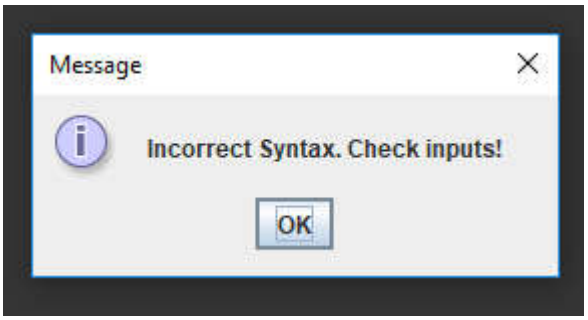
Testing: For each testing scenario you choose to test this program describe the following:

Test Cases: Strong, Weak, Invalid Syntax and StrongNorWeak test files.

Output:

```
Equations:
5.6x^3 + 4.0x + 9.0
7.8x^3 + 9.0x^2 + 8.3x + 12.0
8.0x^4 + 9.0x^3 + 8.3x^2 + 12.0x + 3.0
7.8x^2 + 8.3x + 12.0

List is Neither Weak or Strong Ordered
```




```
Equations:
5.6x^3 + 4.0x + 9.0
7.8x^3 + 9.0x^2 + 8.3x + 12.0
8.0x^4 + 9.0x^3 + 8.3x^2 + 12.0x + 3.0

List is Strong Ordered
```

```
Equations:
7.1x^3 + 3.6x^2 + 8.3x + 6.0
5.6x^2 + 4.0x + 9.0

List is Weak Ordered
```

Reflection: Comparing the polynomials proved to be tricky and took some extra research to get a better understanding. Converting the raw numbers to the polynomial form seemed to build on project 1 which I felt I had a better grasp of this time.