Single Variable Polynomial Processing

Fejér Alpár

Group 30424

Objective

The goal of this project is to develop a single variable polynomial processor . The operations included in this polynomial calculator are both binary and unary operations . The binary operations implemented are arithmetical , namely addition , subtraction , multiplication and division . The unary operations are derivation and integration of polynomials .

The project works with a graphical user interface , which allows the user to input polynomials as variables for the operations , choose operations to do on the variables , as well as viewing the results of these operations . Errors are written in the result box in case of incorrect input variables .

Problem analysis

The variables for this project are polynomials , so the first problem is modeling the instances of polynomials . These instances will be later used in the operations .

The graphical user interface is an important component of the project , as the user only cares of the simple usage and easy understanding of the program . The graphical user interface in this project is made to fit this description . With only the necessary fields and buttons present , the interface is as simple as it can be .

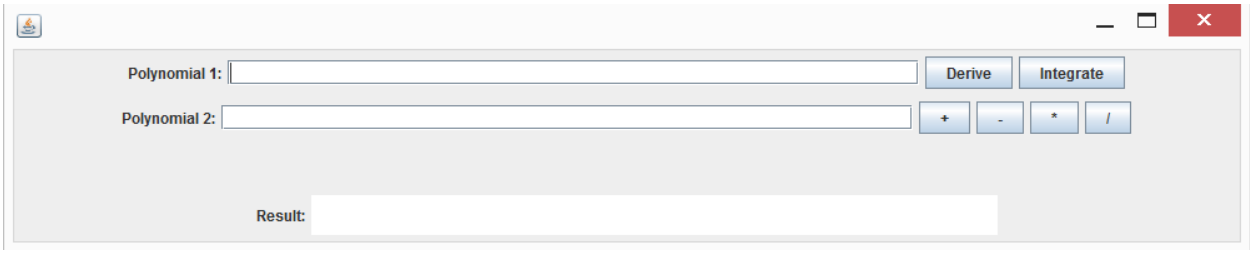
Modeling

In order to use the aspects of the object oriented programming , and also for eliminating redundancy by inheritance , the polynomials that are modeled are represented by a class that contains all the information about an object of polynomial type as well all the necessary methods that can be performed on an object of this type .

The most important building block of a polynomial is a monomial (for this project , a term) , after all , a polynomial is nothing more than the sum of monomials . A monomial is a single variable with an assigned coefficient and a degree . In order to make the model more structured , a new class was created , called Term , that defines the structure of monomials . This structure is used by the polynomial class .

The other important structural component of the application is the graphical user interface . It was designed to have a main frame that contains all the necessary buttons , and text fields . For the sake of simplicity , no additional frame is present in the project , the user can see all the necessary fields when opening the program .

User Interface



The graphical user interface is made to be simple , easy to use and to understand , and with as few distractions as possible . The first two text fields are for reading the two variables for the operations . For the unary operations , the first polynomial will be used .

The operations are done by clicking one of the buttons . No additional action is needed , the polynomials will be read automatically and the result will be written in the third text field .

Swing and Awt components are used in the GUI .

```
import java .awt .event .*;
import javax .swing .*;
```

Three JPanels are used for arranging the text fields and buttons .

```
private JPanel panel1 = new JPanel ( );
private JPanel panel2 = new JPanel ( );
private JPanel panel3 = new JPanel ( );
```

The operations are all launched by JButtons .

```
private JButton derive = new JButton ( "Derive" );
private JButton integrate = new JButton ( "Integrate" );
private JButton add = new JButton ( "+" );
private JButton subtract = new JButton ( "-" );
private JButton multiply = new JButton ( "*" );
private JButton divide = new JButton ( "/" );
```

These JButtons have a common ActionListener .

```
public void actionPerformed ( ActionEvent e ) {
        if ( e .getSource ( ) == add ){
                res .setText ( Operation .add ( new Polynomial ( pol1 .getText ( ) ) , new
Polynomial ( pol2 .getText ( ) ) ) .toString ( ) );
        }
        if ( e .getSource ( ) == integrate ){
```

```
                    res .setText ( Operation .integrate ( new Polynomial ( pol1 .getText ( ) ) )
.toString ( ) );
                }
                if ( e .getSource ( ) == divide ){
                    res .setText ( Operation .divide ( new Polynomial ( pol1 .getText ( ) ) ,
new Polynomial ( pol2 .getText ( ) ) ) .toString ( ) );
                }
                if ( e .getSource ( ) == derive ){
                    res .setText ( Operation .derive ( new Polynomial ( pol1 .getText ( ) ) )
.toString ( ) );
                }
                if ( e .getSource ( ) == subtract ){
                    res .setText ( Operation .sub ( new Polynomial ( pol1 .getText ( ) ) , new
Polynomial ( pol2 .getText ( ) ) ) .toString ( ) );
                }
                if ( e .getSource ( ) == multiply ){
                    res .setText ( Operation .multiply ( new Polynomial ( pol1 .getText ( ) ) ,
new Polynomial ( pol2 .getText ( ) ) ) .toString ( ) );
                }
        }
```

For input and output , JTextFields and JTextAreas are added to the graphical interface , and for better understanding JLabels mark them .

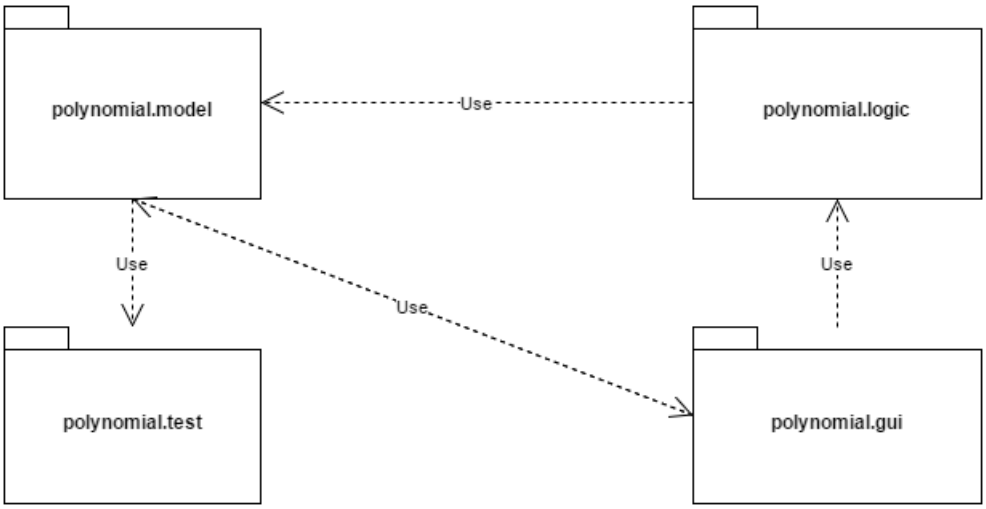
```
        private JTextField pol1 = new JTextField ( 50 );
        private JTextField pol2 = new JTextField ( 50 );
        private JTextArea res = new JTextArea ( 2 ,50 );

        private JLabel po1 = new JLabel ( "Polynomial 1:" );
        private JLabel po2 = new JLabel ( "Polynomial 2:" );
        private JLabel re = new JLabel ( "Result:" );
```

Classes and packages

This section contains the description of the used classes and packages in more detail .

Packages

There are a total of four packages in the java project , namely: polynomial .model , polynomial .gui , polynomial .logic and polynomial .test , which contains the unit tests for all the operations , the String constructors for user input , and the toString ( ) method for the output .

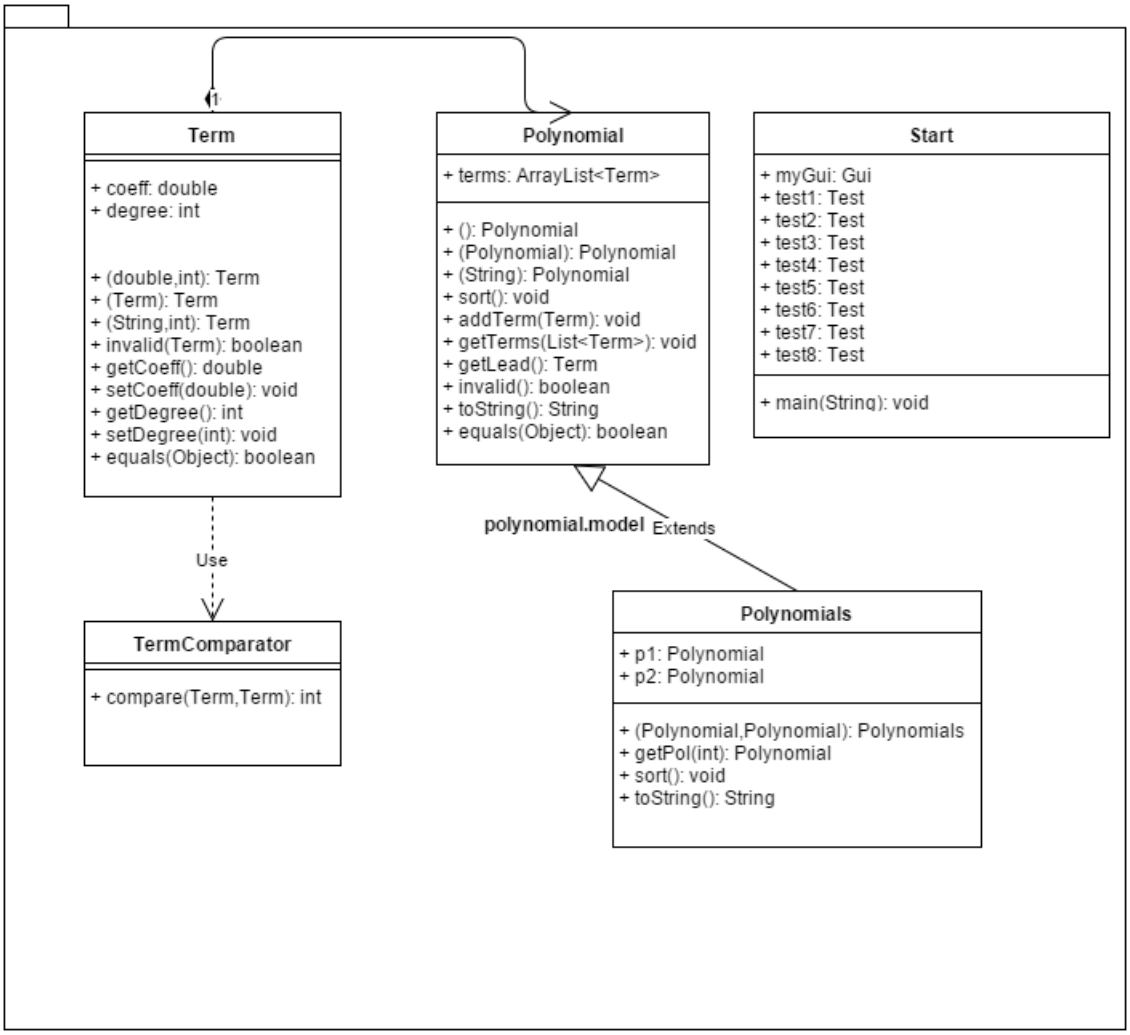


As it can be seen from the diagram , the model holds the main function of the program . This function first calls every JUnit test from the polynomial .test package sequentially to assure that the program will work correctly . Only when all the tests yield a positive result , the user interface is shown and the user can begin using the program .

Polynomial .model

This package contains everything used for modeling polynomials , from coefficients to splitting the String input into terms and creating polynomials from them .

The polynomial .model package holds five classes , namely: Term , TermComparator , Polynomial , Polynomials , and Start .



Term

The Term class is used to model monomials . The connection between this and our main modeling class , the Polynomial class is of composition , as the program simply does not use monomials separately . It has getter and setter methods , as well as overridden equals method .

The constructor for this class if overloaded , as it is used in many different scenarios . There is also an invalid method , returning a Boolean value , true if the term is correct and usable in operations and false otherwise .

TermComparator

The TermComparator class is for overriding the comparing of Terms , used for sorting the terms of Polynomials in decreasing order by degrees .

Polynomial

This is the main class for modeling polynomials , and the cornerstone of this project . It holds an ArrayList of monomials (Terms) . The class has overloaded constructors . The toString and equals methods are overridden as well . There are also methods for adding terms , getting the list of the terms in the current polynomial , getting the first term of the polynomial , and for sorting the polynomial by degrees of terms in descending order . This method is paramount for some of the operations , for a cleaner output , and for a flexible input .

Polynomials

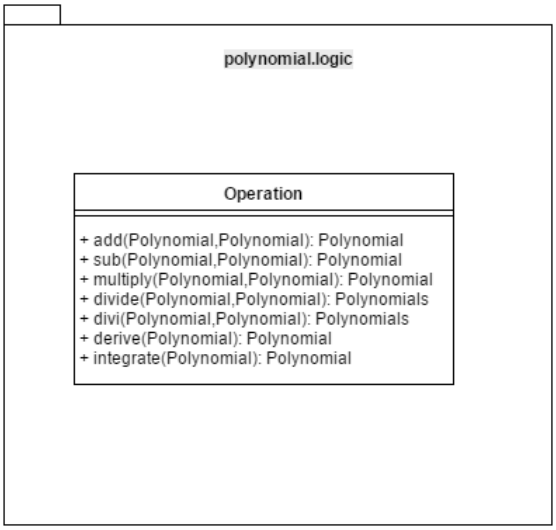
The Polynomials class is used for the result of a division . It holds two polynomials and inherits from the Polynomial class , with overridden sort and toString methods .

Start

The Start class holds the main method , it is the one that starts the program . It carries out the JUnit tests in the polynomial .test package before creating a user interface .

Polynomial .logic

The operations implemented in the project are contained in this package . It uses the modeled polynomials from the model package .



This class holds all the logic that goes into the project . All the operations implemented are methods in this class . Every methods first checks if its variables are not special polynomials , used for error omitting , with the invalid method . Also , every method sorts the result for a clear visualization of the result .

Addition

```
public static Polynomial add ( Polynomial p1 , Polynomial p2 ) {
                if  ( p1 .invalid (  ) || p2 .invalid (  ) )
                        return new Polynomial ( "Invalid" );
                Polynomial result = new Polynomial ( p1 );
                for  ( Term term : p2 .getTerms (  ) ) {
                        result .addTerm ( term );
                }
                result .sort (  );
                return result;
        }
```

The addition of two polynomials is straightforward . First , the result polynomial is initialized with the first operand , p1 . Then , every term from the second operand is added to the first polynomial .

Subtraction

```
public static Polynomial sub ( Polynomial p1 , Polynomial p2 ) {
                if  ( p1 .invalid (  ) || p2 .invalid (  ) )
```

```
                return new Polynomial ( "Invalid" );
        Polynomial result = new Polynomial ( p1 );
        for  ( Term term : p2 .getTerms ( ) ) {
                double coeff = term .getCoeff ( );
                term .setCoeff ( ( -1 )*coeff );
                if  ( term .getCoeff ( )!=0 )
                result .addTerm ( term );
        }
        result .sort ( );
        return result;
}
```

Subtracting one number from an other one is the same as adding the inverse of that number to the first one . The same can be said for polynomials . The subtraction method first negates every term from the second polynomial by multiplying its coefficients by -1 , and then it adds it to the first polynomial .

Multiplying

```
public static Polynomial multiply ( Polynomial p1 , Polynomial p2 ){
        if  ( p1 .invalid ( ) || p2 .invalid ( ) )
                return new Polynomial ( "Invalid" );
        Polynomial result = new Polynomial ( );
        for  ( Term term1 : p1 .getTerms ( ) )
                for  ( Term term2 : p2 .getTerms ( ) )
                {
                        int degree = term1 .getDegree ( ) + term2 .getDegree ( );
                        double coeff = term1 .getCoeff ( ) * term2 .getCoeff ( );
                        Term term = new Term ( coeff ,degree );
                        result .addTerm ( term );
                }
        result .sort ( );
        return result;
}
```

Multiplying polynomials is done by multiplying their terms one by one . The addTerm method makes sure that the terms with the same degree are not written twice or separately . Multiplying two terms is done by adding their degrees and multiplying their coefficients .

Dividing

```
public static Polynomials divide ( Polynomial p1 , Polynomial p2 ){
        if  ( p1 .invalid ( ) || p2 .invalid ( ) )
                return new Polynomials ( new Polynomial ( "Invalid" ) ,new Polynomial
( "Zero" ) );
        Polynomial p3 = new Polynomial ( p1 );
```

```
                Polynomial p5 = new Polynomial (  );
                Polynomials result;
                if  (  ( p1 .getLead (  ) .getCoeff (  ) == -Integer .MAX_VALUE && p1 .getLead (
) .getDegree (  ) == Integer .MAX_VALUE ) ||
                                        ( p2 .getLead (  ) .getCoeff (  ) == -Integer .MAX_VALUE && p2
.getLead (  ) .getDegree (  ) == Integer .MAX_VALUE ) )
                        return new Polynomials ( new Polynomial ( "dasd" ) ,new Polynomial (
"Zero" ) );
                if  ( p2 .equals ( new Polynomial ( "" ) ) ){
                        return new Polynomials ( new Polynomial ( "Div0" ) ,new Polynomial (
"Zero" ) );
                }
                while  ( p3 .getLead (  ) .getDegree (  ) >= p2 .getLead (  ) .getDegree (  ) ){
                        result = Operation .divi ( p3 , p2 );
                        p3 = result .getPol ( 1 );
                        p5 .addTerm ( result .getPol ( 2 ) .getLead (  ) );
                }
                result = new Polynomials ( p5 ,p3 );
                result .sort (  );
                return result;
        }

        private static Polynomials divi ( Polynomial p6 , Polynomial p7 ){
                int degree = p6 .getLead (  ) .getDegree (  )-p7 .getLead (  ) .getDegree (  );
                double coeff = p6 .getLead (  ) .getCoeff (  ) / p7 .getLead (  ) .getCoeff (  );
                Term term = new Term ( coeff ,degree );
                Polynomial fac = new Polynomial (  );
                fac .addTerm ( term );
                Polynomial fac2 = Operation .multiply ( fac , p7 );
                Polynomial rem = Operation .sub ( p6 ,fac2 );
                Polynomials result = new Polynomials ( rem ,fac );
                return result;
        }
```

Two methods are needed for dividing polynomials , as recursive calling is not an option for static functions . The second function is a helper function , and it returns a set of polynomials , the second one being the highest term that can divide the current polynomial , and the second one being the current remainder , computed by multiplying the term just computed with the divisor , and subtracting the product from the previous remainder .

The first method calls the helper function in a while loop , until the degree of the result is smaller than the degree of the divisor . At that point , an object of type Polynomials is made with the result and the remainder .

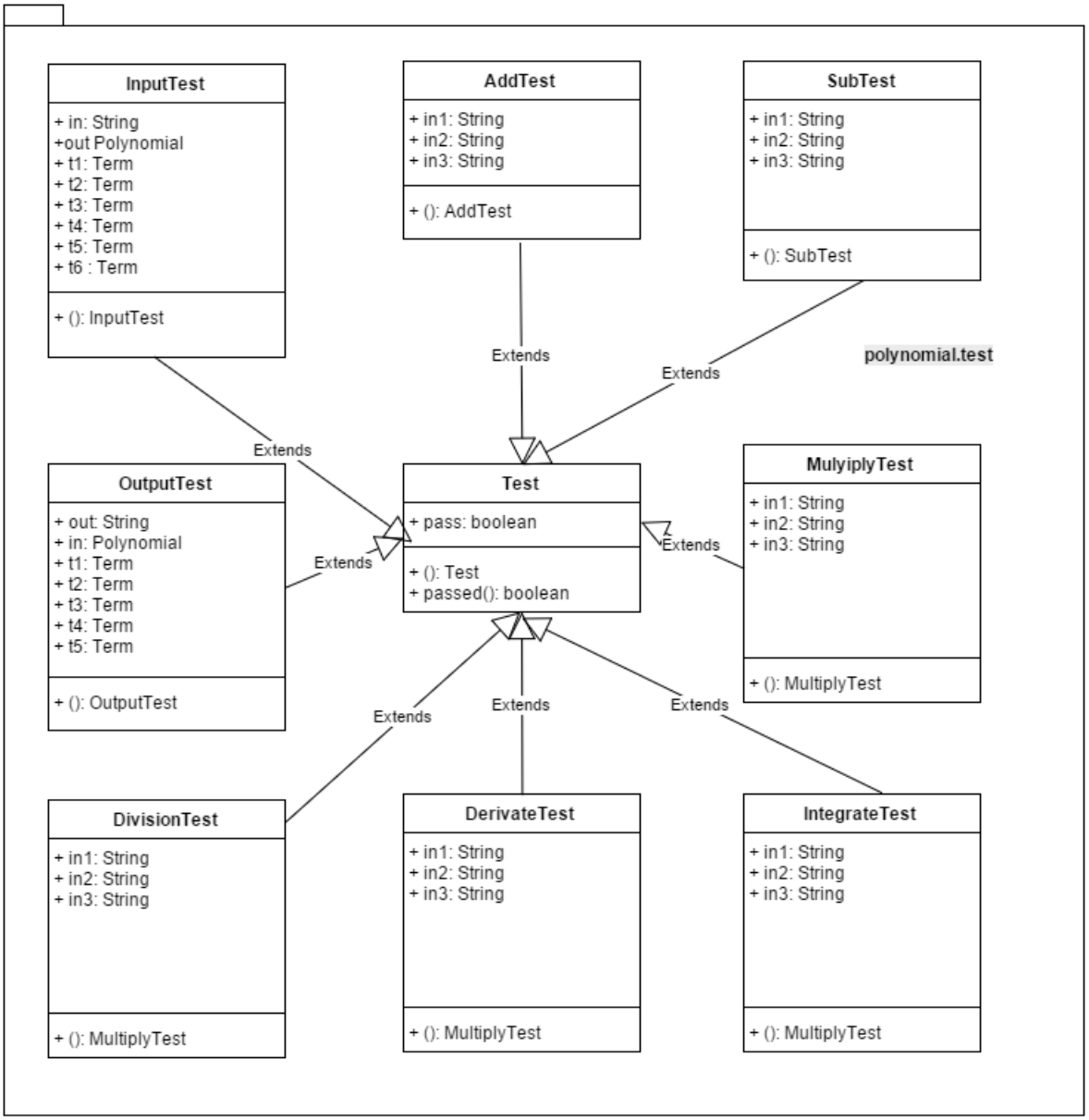
Deriving and Integrating

```
public static Polynomial derive ( Polynomial p ){
        if  ( p .invalid (  ) )
            return new Polynomial ( "Invalid" );
        Polynomial result = new Polynomial (  );
        for  ( Term term : p .getTerms (  ) ){
            if  ( term .getDegree (  ) != 0 ) {
                Term myTerm = new Term ( term .getCoeff (  )*term .getDegree (
) ,term .getDegree (  )-1 );
                result .addTerm  ( myTerm ) ;
            }
        }
        result .sort (  );
        return result;
    }

    public static Polynomial integrate  ( Polynomial p ) {
        if  ( p .invalid (  ) )
            return new Polynomial ( "Invalid" );
        Polynomial result = new Polynomial (  );
        for  ( Term term : p .getTerms (  ) ){
            Term myTerm = new                Term ( term .getCoeff (  )/ ( term
.getDegree (  ) + 1 ) ,term .getDegree (  )  + 1 );
            result .addTerm ( myTerm );
        }
        Term k = new Term ( - Integer .MAX_VALUE , - Integer .MAX_VALUE );
        result .addTerm ( k );
        result .sort (  );
        return result;
    }
```

The derivation and integration of polynomials is quite simple . The terms are derived or integrated separately . Integrating a polynomial yields a + C at the end of the result , this is achived with a special polynomial . This polynomial has as coefficient and degree the minimum value of the integer type , and it is interpreted uniquely by the toString method .

Polynomial .test

The JUnit tests are contained in this package . It assures the correctness of the program , as it would be useless if it wouldn't give correct answers .



The test classes in the test package all inherit from the main Test class. They test different methods, like the polynomial constructor and toString method, and all the polynomial operations.

The correct outputs are also given and compared to the outputs of the tested methods. If one of the test yields a negative result, the others are not conducted. That is why the operation tests use the already tested constructor and toString methods.

Polynomial .gui

This holds the user interface . It helps the user use the program without understanding it . It is a crucial part of the project .

The user interface is extremely easy to use , with no unnecessary buttons or complications. The input polynomials must be written in the following way:

Sign + coefficient + x^ + degree , with a few exceptions , to make it easier to write:

- The first term's sign is only needed if it is negative
- 1's don't have to be written as coefficients
- X^1 is written like x, no power sign or degree is written
- X^0 is omitted

Capital X is not recognized, and will result in "Incorrect input" error, like any other letter combination .

Some examples for correct inputs are:

"-x^7+5x-7"

"x^2-2x+3"

Further Development

The project meets the criteria for the given task , but some improvements can still be made . The two problems are with the Object Oriented aspect of the program , and the simplicity of the user interface .

The project can also be developed beyond what the task was , like computing the value of the polynomial in a specific point x , or representing the polynomials graphically . A memory can also be added to the calculator .

Bibliography

http://www.purplemath.com/modules/polydiv2.htm

http://stackoverflow.com/