March, 2016

Laboratory Work – Assignment 1

Polynomial Processing

Name : Faur Andrada

Group : 30421

1. Introduction
   1. Task objectives

The task of the problem is defines as follows : “ Propose , design and implement a system for polynomial processing . Consider the polynomials of one variable and integer coefficients . ” In other words we are asked to build a system for polynomial processing independent of the operations involved or the complexity of the GU I. ( console input / output vs proper Graphical User Interface ) .

* 1. Personal approach

Accordingly to the task , the aim of this project is to solve a problem of polynomial processing by implementing a set of operations that are specific for polynomials . In other words , we implement a sort of polynomial calculator capable to solve operations like : addition , subtraction , multiplication , division, integration and differentiation. This type of operations were chosen due to the fact that are the most used and the most important operations regarding polynomials.

1. Problem description
2. Problem analysis

Some problems are short term , some are long term . Some involve decisions . Some involve a whole range of problems from which priorities must be chosen . Some may not be completely soluble and may have to be coped with . There is no one way that will solve all problems . There are various approaches , or ” tools ” , which will help to solve certain types of problems . Most of them are only ordered common sense , but this is precisely what is lacking in many intuitive attempts to tackle problems . Analysts must be comfortable with a number of tools and should not be afraid of trying out several on any given problem . They are all methods that help us to think our way through the issue . Many problems are complex , involving a whole range of causes . Hence we identify our problem domain and we try to decompose it in modules easy to implement . We should always keep in mind that if we do not have a good model we have to do more complex programs . As already being said we have our problem domain and the solution domain ; the solution domain is closely related to the person that interacts with our program , or the user , so it is very dependent of the user interface , or as we also like to call it GUI . Our problem domain that is defined by the mathematical definition of a polynomial :

P ( x ) = a n  x n + a n−1 x n−1 + . . . + a 1 x + a 0  , where a n ≠ 0 , n ∈ N

Of course that the first coefficient should be different from 0 , in order for the polynomial of degree n to exist . Also we can observe that the degree of the polynomial is a natural number . Hence we can see the polynomial as a sum of several terms , each having the form ai \* xi . As the task requires , the coefficient ai is an integer coefficient, but we need to keep in mind that after performing the division , for example , the resulting polynomial will actually have real coefficients . The operations we aim to implement work as follows : addition implies adding the coefficients of the terms with the same degree from the two polynomials P(x) and Q(x) , subtraction implies subtracting the coefficients of the terms with the same degree from the two polynomials , multiplication implies multiplication of each term from the first polynomial with each term from the second polynomial , meaning the multiplication of the coefficients and the addition of the degrees and so on and so forth.

1. Modeling

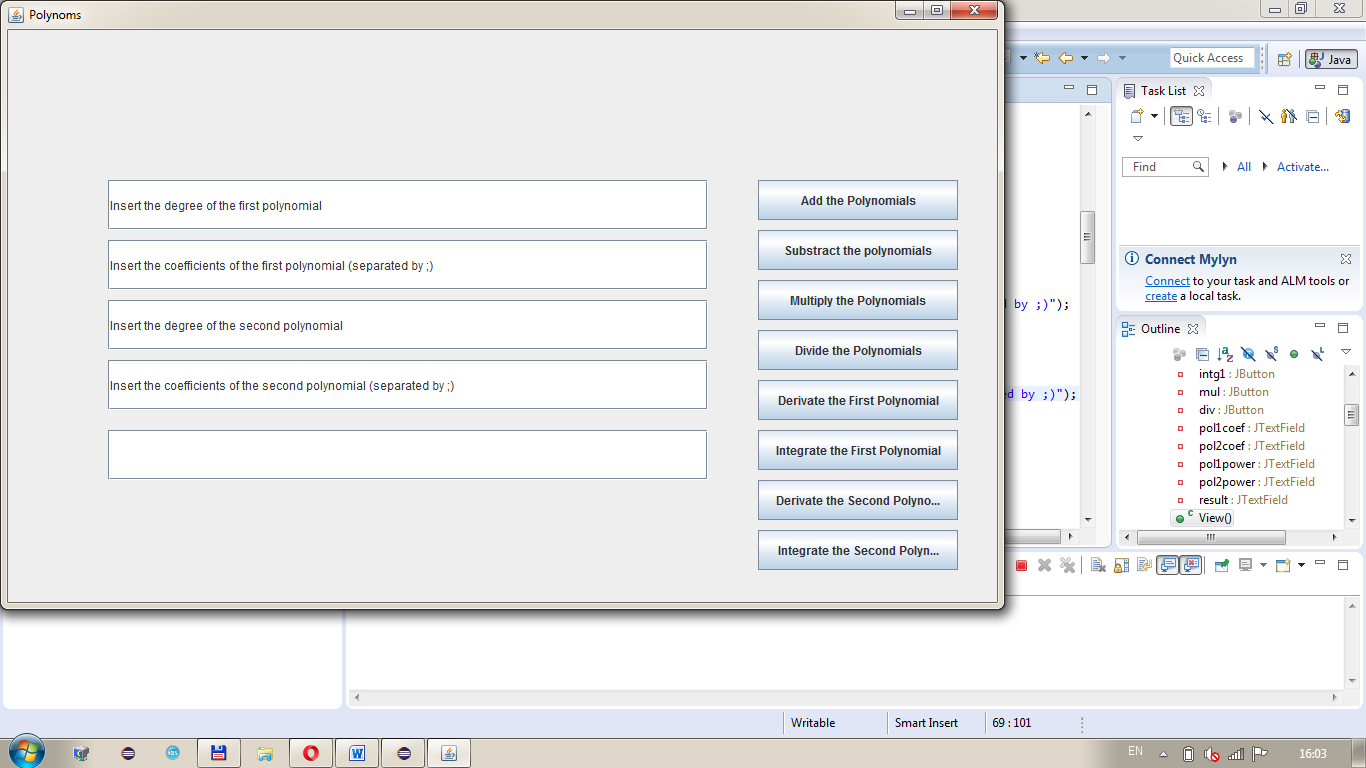
I have questioned myself how to structure my model . . . how many classes do I need , how many packages ? Should I use arrays in order to keep the coefficients of the polynomials that I am working with or it would be better to look for another approach ? I was trying to find a way of modeling my idea that would be efficient , but not only efficient . . . there were many details to think about , like performance and memory and so on . Shortly I decided upon using ArrayList instead , even though the first option , with arrays , seemed to be easier to implement.

1. Scenarios

I have already specified that , at first , I was very optimistic about implementing my problem with arrays to keep the coefficients of my terms and start doing operations such as addition , subtraction , multiplication based on this aspect . But as my problem seemed to get more complex ( I decide upon other operations such as division or integration or differentiation ) , I started to have more classes , so I decided that using ArrayList was a better option. I divided the problem into 4 packages : ” gui ” , ” monom ” , ” polinom ” and ”test ” . The first package , ” gui ” , corresponds to the user-interface part , while the others are as it follows : ” monom ” represents a general term of the polynomial ; ” polinom ” builds the polynomyals and defines and implements operations that run on polynomials ; ” test ” verifies the operations .

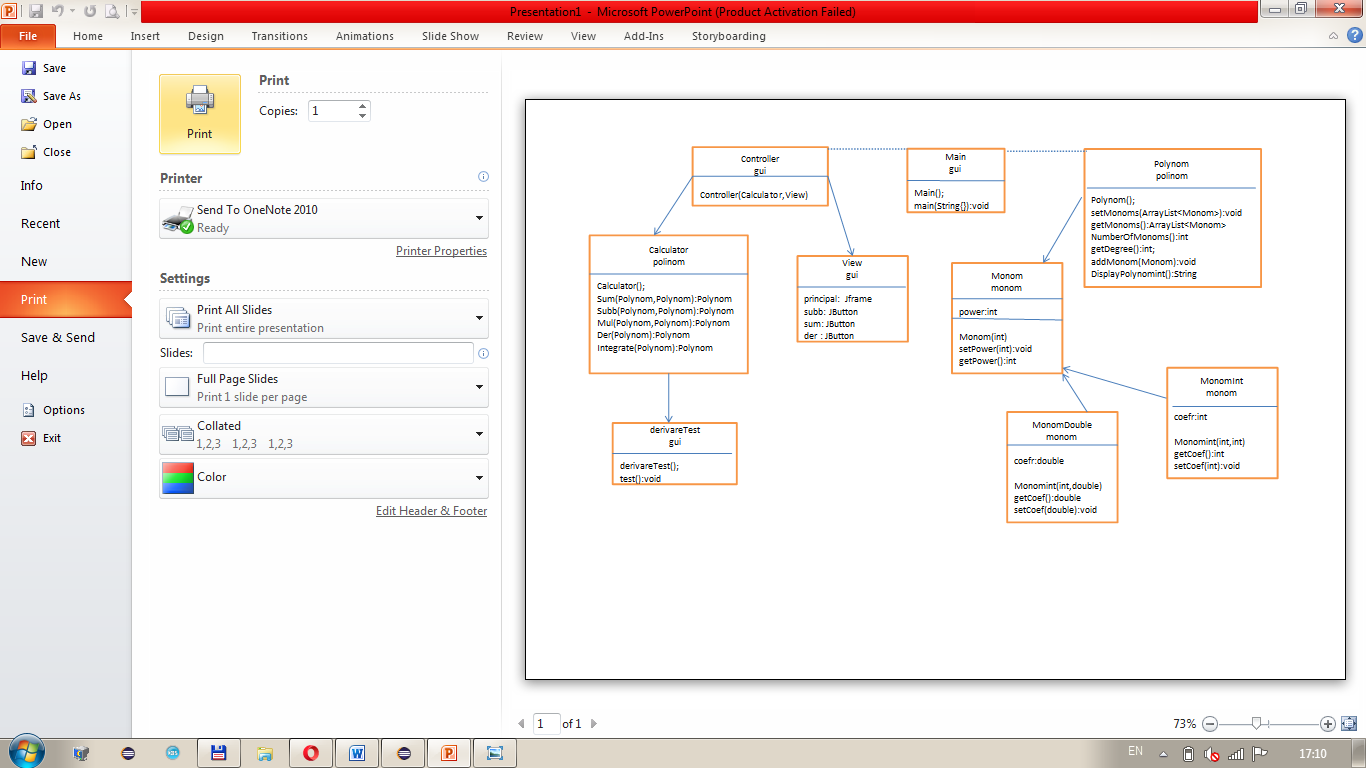
1. Use cases

I have decided upon the following model regarding the interface :

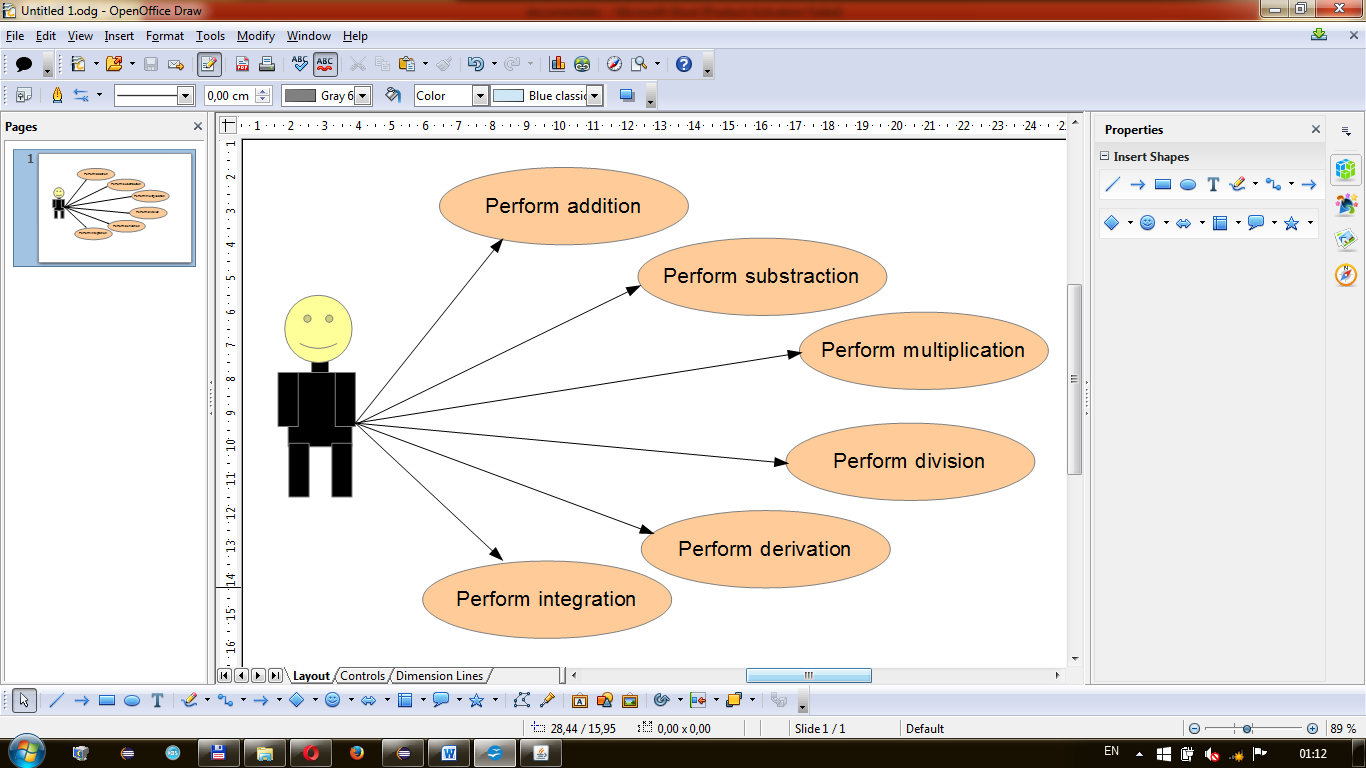


The user gives the degree of the first polynomial and then inserts the coefficients ( first the coefficient of an , then the coefficient of an-1 and so on ) . The same operation must be done for the second polynomial if the user wants to proceed an operation that requires both polynomials . After that , the user will select one of the operations available and the answer will appear in the last textField that initially was empty , or if an error occurs, there will be an error message displayed in the same spot.

1. Projection
2. UML diagram



1. Use Case Diagram



The use case diagram present the actor , ( which is the user ) that interacts with the application . He can perform several actions such as adding , subtracting , multiplying, dividing , integrating , differentiating 2 polynomials .

1. Data structures

The data structures used at this problem are either primitive data types such as integers or floats or more complex objects such as ArrayList type objects or new created objects such as Monom, Polynomial or Calculator . The object Monom was introduced for representing a term in the sequence of elements that form a polynomial . Those monoms were added in a list of type ArrayList<Monom> , Monom being the base-type . Again the object Polynomial has been introduced in order to obtain a list with such ” terms ” that form together a polynomial ; adding several such terms we obtain the desired polynomial . Last but not least, the object Calculator is intended to store the operations that can be implemented : addition , subtraction , multiplication , division, integration and differentiation.

1. Class projections

Class projection refers mainly to how the model was thought , how the problem was divided in sub-problems , each sub - problem representing more or less the introduction of a new class . I divided the problem into four packages : the first one is the ” gui ” package and it handles the interface part ; the second one is the ” monom ” package and it deals with the terms of each polynomial ; the third one is the package ” polinom ” which conceal the polynomials being created from monoms , but also the operations that can be implemented using those polynomials ; and the last one is the package ” test ” that I used to verify the operations .

1. package gui – it handles the interface part , the part that deals with the

user and it contains three classes : ” Controller ” , ” Main ” and ” View ” .

Controller.java - this class contains the action that each button does

**public** **class** Controller {

**private** Calculator calculator;

**private** View view;

**public** Controller(Calculator calculator, View view) {

**this**.view = view;

**this**.calculator = calculator;

**this**.view.addSumListener(**new** sumListener());

**this**.view.addSubbListener(**new** subbListener());

**this**.view.addMulListener(**new** mulListener());

**this**.view.addIntegrateListener(**new** integListener());

**this**.view.addDerListener(**new** derListener());

**this**.view.addIntegrate1Listener(**new** integ1Listener());

**this**.view.addDer1Listener(**new** der1Listener());

}

**class** sumListener **implements** ActionListener {}

class subbListener implements ActionListener {}

class derListener implements ActionListener {}

class der1Listener implements ActionListener {}

class mulListener implements ActionListener {}

class integListener implements ActionListener {}

class integ1Listener implements ActionListener {}

}

Main.java – it joins graphical interface and methods that are implemented

**public** **class** Main {

**public** **static** **void** main(String args[]) {

View v = **new** View();

Calculator cl = **new** Calculator();

Controller c = **new** Controller(cl, v);

}

}

View.java - this class actually implements the graphical interface

**public** **class** View {

**private** JFrame principal;

**private** JButton sum;

**private** JButton subb;

**private** JButton der;

**private** JButton intg;

**private** JButton der1;

**private** JButton intg1;

**private** JButton mul;

**private** JButton div;

**private** JTextField pol1coef;

**private** JTextField pol2coef;

**private** JTextField pol1power;

**private** JTextField pol2power;

**private** JTextField result;

**public** View() {

principal = **new** JFrame();

principal.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

principal.setTitle("Polynoms");

principal.setSize(1020, 720);

JPanel panel = **new** JPanel();

panel.setLayout(**null**);

**this**.principal.add(panel);

principal.setVisible(**true**);

sum = **new** JButton("Add the Polynomials");

subb = **new** JButton("Substract the polynomials");

der = **new** JButton("Derivate the First Polynomial");

intg = **new** JButton("Integrate the First Polynomial");

der1 = **new** JButton("Derivate the Second Polynomial");

intg1 = **new** JButton("Integrate the Second Polynomial");

mul = **new** JButton("Multiply the Polynomials");

div = **new** JButton("Divide the Polynomials");

//set each button’s boundaries and the text for the text fields...

}

1. package monom

Monom.java - this class represents a monom , i.e. the power and the coefficient corresponding to it ; the ” Monom ” class also contains the constructor along with the getters and setters .

**public** **class** Monom {

**private** **int** power;

**public** Monom(**int** power) {

**this**.power = power;

}

**public** **void** setPower(**int** power) {

**this**.power = power;

}

**public** **int** getPower() {

**return** **this**.power;

}

MonomDouble.java – this class represents a monom whose coefficients are real numbers

MonomInt.java – this class represents a monom whose coefficients are integer numbers

1. package polinom

Calculator.java - is the most complex class . This class defines and implements operations that run on polynomials .

public class Calculator {

public Polynom Sum(Polynom A, Polynom B) {}

public Polynom Subb(Polynom A, Polynom B) {}

public Polynom Mul(Polynom A, Polynom B) {}

public Polynom Der(Polynom A) {}

public Polynom Integrate(Polynom A) {}

}

Polynom.java - unites all the monoms forming an polynomial using an arrayList of monoms ( ArrayList<Monom> ) .

1. package test
2. Interface

As it was shown in the Classes projection section, the interface consists of a frame that incorporates eight buttons ( sum, subb, der, intg, der1, intg1, mul, div ) and five text fields (pol1coef, pol2coef, pol1power, pol2power, result ). Each button has the bounds setted on ( 750 , 200 , 200 , 40 ) , and each represents as it follows :

sum – addition of the two polynomials ;

subb – substraction betwen the two polynomials ;

der – derivation of the first polynomial ;

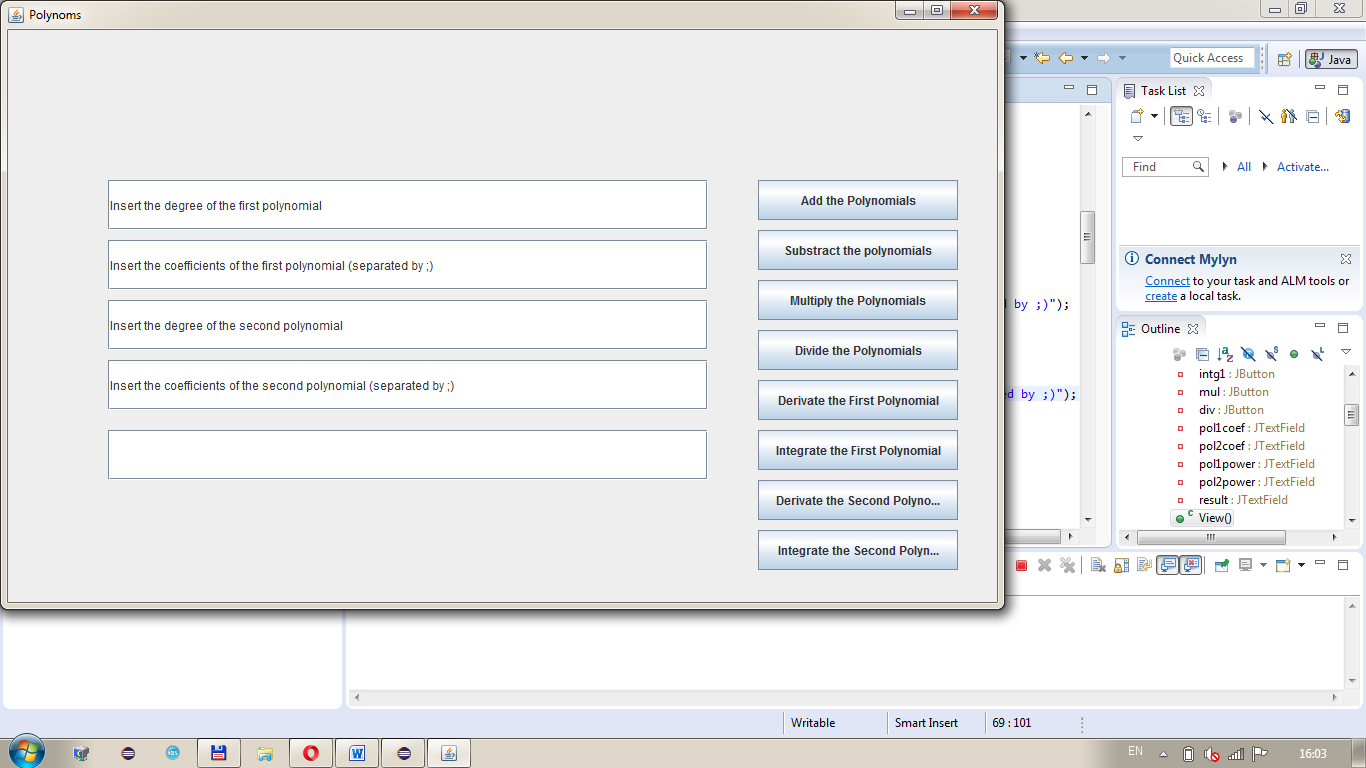
intg – integration of the first polynomial ;

der1 – derivation of the second polynomial ;

intg1 – integration of the second polynomial ;

mul – multiplication of the polynomials ;

div – division of thw two polynomials .



The text first and the third text fields ( pol1power and pol2power ) are used for getting the degree of each polynomial , while the second and the fourth ( pol1coef and pol2coef ) aim at getting the coefficients , in order to create the polynomials . The last one ( result ) will the answer will be filled with the result of the operation that the user had had select by pressing one of the buttons in the right side of the frame , or , if an error occurs, there will be an error message displayed in the same spot .

1. Packages

My program contains 4 packages : ” gui ” , ” monom ” , ” polinom ” and ” test ”.

The ” gui ” contains the classes : Controller.java , Main.java and View.java .

The ” monom ” contains the classes : Monom.java , MonomDouble.java and MonomInt.java .

The ” polinom ” contains the classes : Calculator.java and Polynom.java

1. Algorithms

In this section I will describe mainly the method that are implementing the operations:

* Adition

additionAdding polynomials is just a matter of combining like terms , with some order of operations considerations thrown in.

example :

Applicable method:

public Polynom Sum(Polynom A, Polynom B)

- in this method we construct a new object of type Polynomial in which we keep the resulting polynomial

- declares two lists to extract the elements from both Polynomials given at parameters

- defines two iterators to cross this lists

- crosses the first list and adds the elements of the first polynomial to the resulting list

- crosses the second list and adds the elements of the second polynomial to the resulting list

- when we cross the second list we are careful about the terms with the same degree; for them the coefficients are added

* Substraction

change signssubtractionSubtracting polynomials is quite similar to adding polynomials, but you have that pesky minus sign to deal with.

example :

Applicable method:

public Polynom Subb(Polynom A, Polynom B)

- in this method we construct a new object of type Polynomial in which we keep the resulting polynomial

- declares two lists to extract the elements from both Polynomials given at parameters

- defines two iterators to cross this lists

- crosses the first list and adds the elements of the first polynomial to the resulting list

- crosses the second list and subtracts the elements of the second polynomial to the resulting

- when we cross the second list we are careful about the terms with the same degree ; for them the coefficients are subtracted

* Multiplication

To multiply two polynomials:

* multiply each term in one polynomial by each term in the other polynomial
* add those answers together, and simplify if needed

To multiply one term by another term, we first multiply the **constants**, then multiply **each variable** together and combine the result.

Example: (x + 2y)(3x − 4y + 5)

(x + 2y)(3x − 4y + 5)

= 3x2 − 4xy + 5x + 6xy − 8y2 + 10y

= 3x2 + 2xy + 5x − 8y2 + 10y

Note: **−4xy** and **6xy** are added because they are Like Terms.

Applicable method:

public Polynom Mul(Polynom A, Polynom B)

- in this method we construct a new object of type Polynomial in which we keep the resulting polynomial

- declares two lists to extract the elements from both Polynomials given at parameters

- defines two iterators to cross this lists

- while crossing the first list it crosses also the second lists and it multiplies each term from the first list with each term from the second list

-each new term is added to the resulting list

* Division

public Polynom Der(Polynom A)

The classical algorithm for division is described as follows:

-we have two polynomials P(x) the dividend and Q(x) the divisor

-we divide the most significant term of the dividend to the MST of the divisor

- this way we are obtaining the first term of the quotient

- we multiply the result with the divisor and we subtract this result from the dividend

- this computation gives us the first remainder of the division

- we repeat this procedure taking now the obtained remainder as the dividend

-the algorithm ends when the deg (remainder) is less than the deg (quotient)

* Integration

The integral of any polynomial is the sum of the integrals of its terms.

Applicable method:

public Polynom Integrate(Polynom A)

- in this method we construct a new object of type Polynomial in which we keep the resulting polynomial

- declares a list to extract the elements from the Polynomial given at parameters

- defines an iterators to cross this list

- while crossing the list it changes each element's coefficient is divided by its degree incremented with 1 and each element's degree is incremented with 1

* Derivation

In order to find the derivative of a polynomial, we have to differentiate each term of the polynomial and then to add up the resultant terms.

Applicable method:

public Polynom Der(Polynom A)

-differentiate each term of the polynomial

- add up the resultant terms.

1. Implementation and testing

This project was developed in Eclipse and it was only tested in this environment, however the program should maintain its portability. I have tried to implement my problem in a way that appears to me as being the most efficient one, that is why I have changed my model at first. On the subject of this I have decided to use ArrayList instead of arrays because I believe that to performance and memory management.

Testing implies checking for any errors in the program or limitations of this program. I assumed that the worst thing that could happen was that the operations implemented not to work correctly, which is the main reason for creating the package ” test ” where to verify the correctness of the implemented operations.

1. Results

The application is user friendly and it is also very useful for performing operations on polynomials, such as addition, substraction, derivation, integration and division. As the application is developed on a Java platform, it is highly portable and allows it to run on several operating systems (as long as they have the Java SDK installed). I strongly believe that the interface provides the right hints so that any user that has basic knowledge about polynomials can easily use it.

1. Conclusions

All in all, this project was quite helpful because it made me think not only how to solve the task, but also how to design it in order to be accesible and easy to use. Even though it was a bit difficult until I got the final shape of the application, I have learned that a good modularisation with provide both a good implementation and a better view of the whole image. All these things being said, I have learned a lot from this project and I am very anxious to apply what I have learned in the future projects.

1. Bibliography

* <https://www.miun.se/siteassets/fakulteter/nmt/summer-university/problemanalysispdf>

Problem Analysis Techniques

(Operations on polynomials: )

* <http://www.mathportal.org/calculators/polynomials-solvers/polynomial-factoring-calculator.php>
* <http://www.webmath.com>
* <http://www.purplemath.com>
* <https://www.mathsisfun.com/algebra/polynomials.html>
* <http://hyperphysics.phy-astr.gsu.edu/hbase/intpol.html>
* <http://www.math.brown.edu/UTRA/derivtips.html>