**Technical University of Cluj-Napoca  
Faculty of Automation and Computer Science  
Computer Science Department**

***Drimbarean Maria Cristina***

***Group:30425***



1.Assignment objective

1.1 General view

*Task was described with the following statement: “Propose, design and implement a system for*

*polynomial processing. Consider the polynomials of one variable and integer coefficients.” So we are able to analyze and design what we want the application to do, as we do not have a lot of specfications from the user. Polynomials appear in a wide variety of areas of mathematics and science. For example, they are used to form*[*polynomial equations*](https://en.wikipedia.org/wiki/Polynomial_equation)*, which encode a wide range of problems, from elementary*[*word problems*](https://en.wikipedia.org/wiki/Word_problem_(mathematics_education))*to complicated problems in the sciences; they are used to define polynomial functions, which appear in settings ranging from basic*[*chemistry*](https://en.wikipedia.org/wiki/Chemistry)*and*[*physics*](https://en.wikipedia.org/wiki/Physics)*to*[*economics*](https://en.wikipedia.org/wiki/Economics)*and*[*social science*](https://en.wikipedia.org/wiki/Social_science)*; they are used in*[*calculus*](https://en.wikipedia.org/wiki/Calculus)*and*[*numerical analysis*](https://en.wikipedia.org/wiki/Numerical_analysis)*to approximate other functions. In advanced mathematics, polynomials are used to construct*[*polynomial rings*](https://en.wikipedia.org/wiki/Polynomial_ring)*and*[*algebraic varieties*](https://en.wikipedia.org/wiki/Algebraic_variety)*, central concepts in*[*algebra*](https://en.wikipedia.org/wiki/Algebra)*and*[*algebraic geometry*](https://en.wikipedia.org/wiki/Algebraic_geometry)*.*

*1.2 Personal approach*

*Therefore it is common sense that we have to provide to the user basic operations on polynomials. They are as follows:add, substract,multiply two polynoms, multiply a polynom with a scalar, devide, itegrate, differentiate, evaluate the polynomial. Also a graphical user interface will make the application more usable as it will facilitate the input and the output .*

*This operations were chosen as they are the most used by users and also i consider them very useful for further processing of data aquired from modeling other real world problems.*

*2.The analysis*

*2.1Mathematical approach*

*This is a mathematical topic that interested people from the begining of modern science, that is why a lot of descriptions of the problem were done. This makes it very easy for us because we do not have to think of the algorithms any more we just have to choose the best one for our implementations in orther to be also corect and effective. In the following i describe mathematically the algorithms chosen.*

*In*[*mathematics*](https://en.wikipedia.org/wiki/Mathematics)*, a****polynomial****is an*[*expression*](https://en.wikipedia.org/wiki/Expression_(mathematics))*consisting of*[*variables*](https://en.wikipedia.org/wiki/Variable_(mathematics))*and*[*coefficients*](https://en.wikipedia.org/wiki/Coefficient)*which only employs the operations of*[*addition*](https://en.wikipedia.org/wiki/Addition) *,* [*subtraction*](https://en.wikipedia.org/wiki/Subtraction)*,*[*multiplication*](https://en.wikipedia.org/wiki/Multiplication)*, and non-negative*[*integer*](https://en.wikipedia.org/wiki/Integer)[*exponents*](https://en.wikipedia.org/wiki/Exponentiation)*. An example of a polynomial of a single variable x is x2 − 4x + 7.*

Polynomials can be added using the [associative law](https://en.wikipedia.org/wiki/Associative_law) of addition (grouping all their terms together into a single sum), possibly followed by reordering, and combining of like terms. For example, if

\begin{align}
 P &= 3x^2 - 2x + 5xy - 2 \\
 Q &= -3x^2 + 3x + 4y^2 + 8
\end{align}

then

P + Q = 3x^2 - 2x + 5xy - 2 - 3x^2 + 3x + 4y^2 + 8 

which can be simplified to

P + Q = x + 5xy + 4y^2 + 6 

*This is a well known Horizontal approach, but it is not relevant for computer wich is able to interpret them only as a list of objects. So the algorithm should simply fit.*

To work out the product of two polynomials into a sum of terms, the distributive law is repeatedly applied, which results in each term of one polynomial being multiplied by every term of the other.[[8]](https://en.wikipedia.org/wiki/Polynomial#cite_note-Edwards-1995-p47-8) For example, if

\begin{align}
 \color{Brown} P &\color{Brown}{= 2x + 3y + 5} \\
 \color{RoyalBlue} Q &\color{RoyalBlue}{= 2x + 5y + xy + 1}
\end{align}

then

\begin{array}{rccrcrcrcr}
{\color{Brown}{P}} {\color{RoyalBlue}{Q}} & {{=}}&&({\color{Brown}{2x}}\cdot{\color{RoyalBlue}{2x}})
&+&({\color{Brown}{2x}}\cdot{\color{RoyalBlue}{5y}})&+&({\color{Brown}{2x}}\cdot {\color{RoyalBlue}{xy}})&+&({\color{Brown}{2x}}\cdot{\color{RoyalBlue}{1}})
\\&&+&({\color{Brown}{3y}}\cdot{\color{RoyalBlue}{2x}})&+&({\color{Brown}{3y}}\cdot{\color{RoyalBlue}{5y}})&+&({\color{Brown}{3y}}\cdot {\color{RoyalBlue}{xy}})&+&
({\color{Brown}{3y}}\cdot{\color{RoyalBlue}{1}})
\\&&+&({\color{Brown}{5}}\cdot{\color{RoyalBlue}{2x}})&+&({\color{Brown}{5}}\cdot{\color{RoyalBlue}{5y}})&+&
({\color{Brown}{5}}\cdot {\color{RoyalBlue}{xy}})&+&({\color{Brown}{5}}\cdot{\color{RoyalBlue}{1}})
\end{array}

which can be simplified to

PQ = 4x^2 + 21xy + 2x^2y + 12x + 15y^2 + 3xy^2 + 28y + 5

* The [derivative](https://en.wikipedia.org/wiki/Derivative) of the polynomial *a*n*x*n + *a*n−1*x*n−1 + ... + *a*2*x*2 + *a*1*x* + *a*0 is the polynomial n*a*n*x*n−1 + (n−1)*a*n−1*x*n−2 + ... + 2*a*2*x* + *a*1. If the set of the coefficients does not contain the integers (for example if the coefficients are integers [modulo](https://en.wikipedia.org/wiki/Modular_arithmetic) some [prime number](https://en.wikipedia.org/wiki/Prime_number) *p*), then k*a*k should be interpreted as the sum of *a*k with itself, k times. For example, over the integers modulo *p*, the derivative of the polynomial *xp* + 1 is the polynomial 0.[[11]](https://en.wikipedia.org/wiki/Polynomial#cite_note-Barbeau-2003-pp64-65-11)
* A primitive integral or [antiderivative](https://en.wikipedia.org/wiki/Antiderivative) of the polynomial *a*n*x*n + *a*n−1*x*n−1 + ... + *a*2*x*2 + *a*1*x* + *a*0 is the polynomial *a*n*x*n+1/(n+1) + *a*n−1*x*n/n + ... + *a*2*x*3/3 + *a*1*x*2/2 + *a*0*x* + *c*, where *c* is an arbitrary constant. For instance, the antiderivatives of *x*2 + 1 have the form 1/3*x*3 + *x* + *c*.
* The algorithm for dividing 2 polynomials is more complex and will be prezented in the implementation section
* Also evaluation of the polynomial and also the scalar multiplycation should not rise any questions about any algorithm

2.2Modeling

The first description of the object came to me simply as an array of coefficients, and the indexes wich describe the power of the term. This will be very simply to implement and also will require few other helper algorithms. But if i have to think in Object Oriented Programming i have to think of a single term as the basic object, and the hole polynom as a Collection of it. This will be more close to reality as if we had a power of 44 but no smaller ones we will need an array with a lot of zeros and if we use an ArrayList we will have only one term wich is very close to reality. it might look very hard to implement at first, but let’s not forget that a lot of operations are already implemented for the Collections like, comparable interface, add and remove terms and sorting. I belive this is the most efficient approach.

2.3Scenarios

As far as the scenarios are concerned we needed a way to perform the operations and the answer occured after i made the basic model classes: IntegerMonom, Polynomial wich contained the list of terms. Then i figured that i will need a new class for implementing the operations and also a separate class for performing divizion as it is a more complex one and also needs to give the result with real coefficients. I also choose the model, control, view architecture because it fits here. The terms and the polynom are models, classes that implement operations are controllers and also the GUI is a view.

2.4Use cases

The application will be used for processing the polynomials, so the use cases will appear as the user enters data and chooses wath to do with it. Here is a presentation of the Graphic Interface:

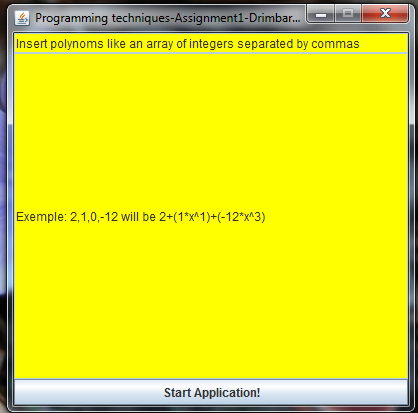


Figure1. The first menu with some speciffications.

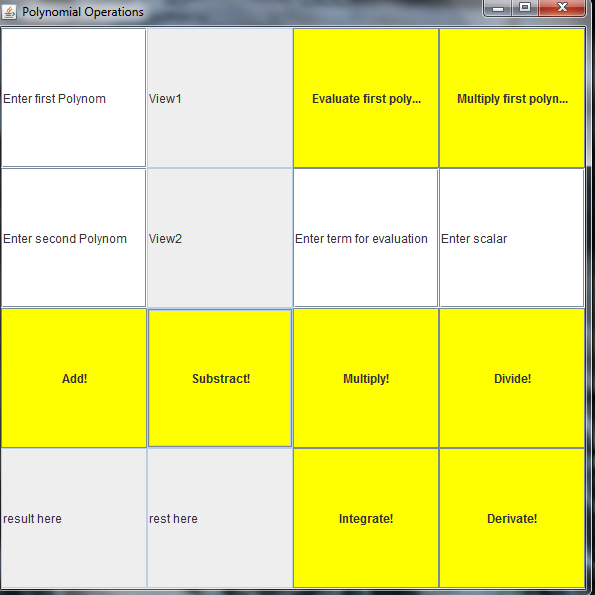
**

Figure2. Buttons and fields of the application

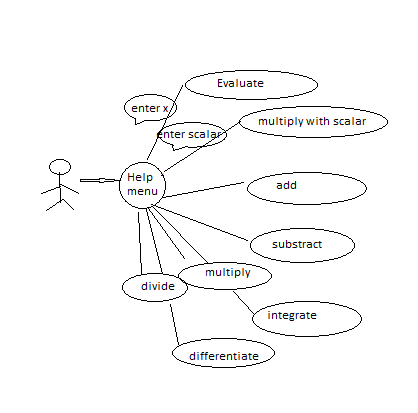
Utilizing the gui should generate the use cases as you choose what operations to do by pressing the yellow buttons. You should put the data in the specified fields.

3.Projection

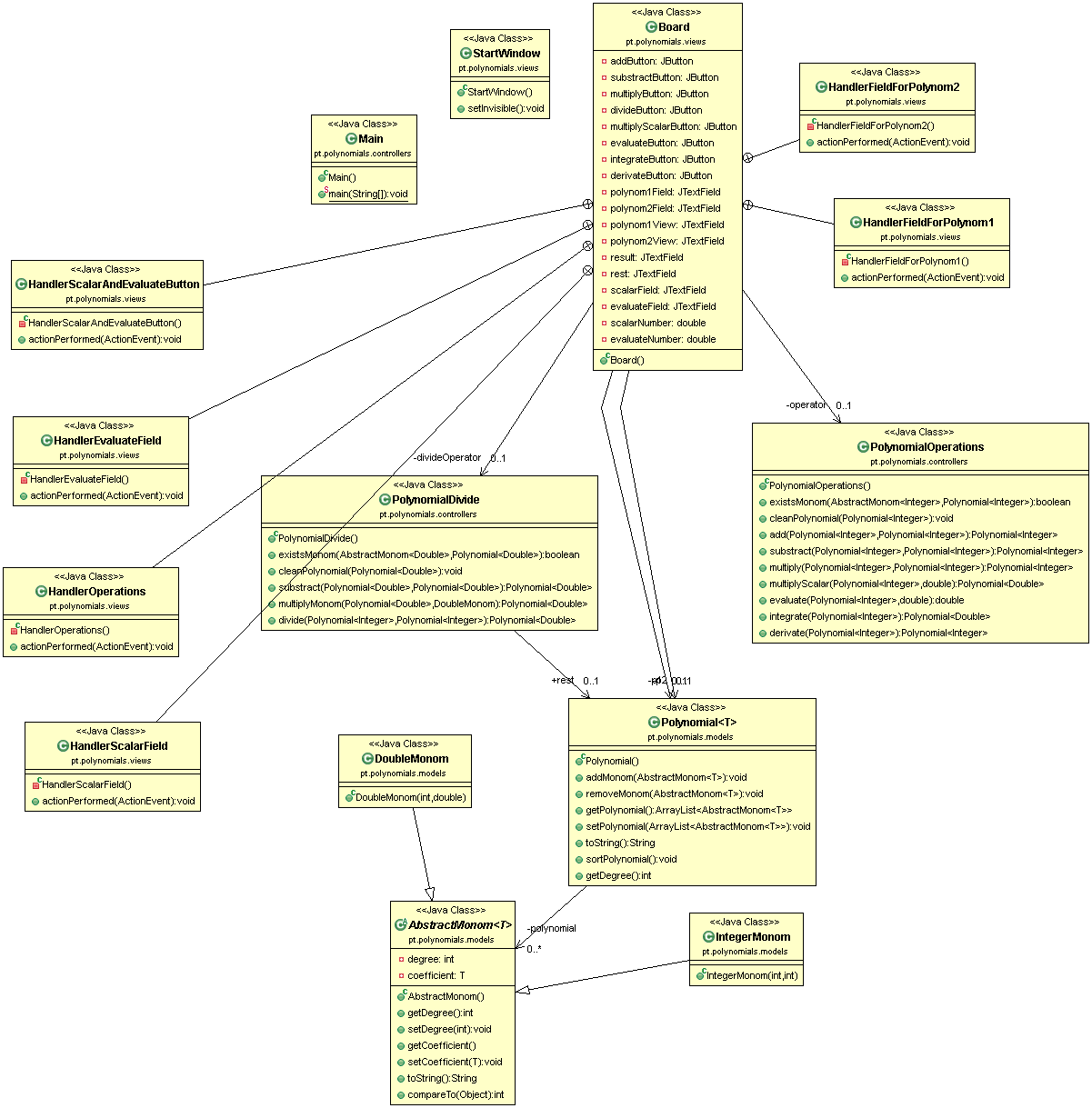
3.1 UML Diagrams

a)Use case

the user is selecting the behaviour of the app



b)Object diagram

All the relationships are presented here. The inheritance between AbstractMonom and double and integer monoms. The composition between polynomial and abstract monom and all the agregations. 

c)Sequence diagram

3.2Data 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

AbstractMonom, IntegerMonom, DoubleMonom. This terms were added in a list of type

ArrayList<AbstractMonom<T>>,Term 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 objects are PolynomialDivide, PolynomialOperations intended to

store any kind of operation implemented. I also used Generics for implementing the abstract monom and the polynomial as i needed different coefficients and i didn’t want to do norrowing with down cast only for reprezenting the polynoms.

3.3 Class projection

I used the three packages: models, controllers and views. And I will begin to explain all the classes starting with:

package pt.polynomials.models;

a)AbstractMonom

**public** **abstract** **class** AbstractMonom<T> **implements** Comparable{

**private** **int** degree;

**private** T coefficient;

has the degree and coefficient as an instance variable, and also implements the interface that enables the comparation based on the degree, (overrides the compareTo from class object):

@Override

**public** **int** compareTo(Object o) {

**int** comparedegree=((AbstractMonom)o).getDegree();

**return** **this**.degree-comparedegree;

}

The concrete implementations of this Class are used for performing the operations and to be added in the result:

**public** **class** IntegerMonom **extends** AbstractMonom<Integer>{}

**public** **class** DoubleMonom **extends** AbstractMonom<Double>{}

This are the heders of the classes to exemplify the inheritence and the implementation used. They contain specific constructors with specific integer and double parameeters.

b)Polynomial

**public** **class** Polynomial<T> {

**private** ArrayList<AbstractMonom<T>> polynomial = **new** ArrayList<AbstractMonom<T>>();

This is also a generic class and contains the list of monoms, it has methods that use the Collection methods for adding,removing elements also sorting the list and getting the reprezentation. The specific method is

**public** **int** getDegree() {

Iterator i = polynomial.iterator();

**int** degree = 0;

**while** (i.hasNext()) {

degree = ((AbstractMonom) i.next()).getDegree();

}

**return** degree;

}

that gets the degree of the polynom and taht is reprezented by the degree of the last term(they are constructed as sorted elements and also they are sorted after each operation).

package pt.polynomials.controllers;

C)PolynomialOperations

Contains all the operations performed for integer coefficients that do not need processing with double coefficients. Two helper methods are:

**public** **boolean** existsMonom(AbstractMonom<Integer> monom, Polynomial<Integer> p)

**public** **void** cleanPolynomial(Polynomial<Integer> p)

They are used to see if the monom exists in a polynom or to remove the term with coefficient 0.

• Addition

The mathematical model for the addition operation is defined as follows:

-we have two polynomials to add, meaning P(x) and Q(x)

-the first polynomial is expressed by the expression:

P( x)=an xn+an−1 xn−1+...+a1 x1+a0 , an≠0, n∈ℕ

-the second polynomial is expressed by the expression:

Q( x)=bn xn+bn−1 xn−1+...+b1 x1+b0 ,bn≠0,n∈ℕ

-the resulting polynomial is expressed by the expression:

R( x)=(an+bn) xn+(an−1+bn−1) xn−1+....+(a1+b1) x1+(a0+b0)

**public** Polynomial<Integer> add(Polynomial<Integer> p1, Polynomial<Integer> p2) {

Polynomial<Integer> resultI = **new** Polynomial<Integer>();

**for** (AbstractMonom<Integer> m1 : p1.getPolynomial()) {

**for** (AbstractMonom<Integer> m2 : p2.getPolynomial())

**if** (m1.getDegree() == m2.getDegree()) {

resultI.addMonom(**new** IntegerMonom(m1.getDegree(), m1.getCoefficient() + m2.getCoefficient()));

**break**;

}

**if** (!existsMonom(m1, resultI))

resultI.addMonom(**new** IntegerMonom(m1.getDegree(), m1.getCoefficient()));

}

**for** (AbstractMonom<Integer> m2 : p2.getPolynomial())

**if** (!existsMonom(m2, resultI))

resultI.addMonom(**new** IntegerMonom(m2.getDegree(), m2.getCoefficient()));

cleanPolynomial(resultI);

resultI.sortPolynomial();

**return** resultI;

}

Iterates through the first polynom and for each monom of the second one asks if the degree is equal,if true it adds the coefficients and adds the element in result and does not search anymore (they are constructed as unique list of monoms) then after the first for asks is the monom already exists in result, if not it adds it because it means the for has not add it because it has no term with the same degree. Then it takes again the second polynom and asks if it exists in result is not then it adds it. This is the exact implementation of the algorithm. At the and it clears the polynom from terms with 0 coefficient and also sorts it based on the degree.

• Subtraction

The mathematical model for the subtraction operation is defined as follows:

-we have two polynomials to subtract, meaning P(x) and Q(x)

-the first polynomial is expressed by the expression:

Q( x)=bn xn+bn−1 xn−1+...+b1 x1+b0 ,bn≠0,n∈ℕ

-the second polynomial is expressed by the expression:

P( x)=an xn+an−1 xn−1+...+a1 x1+a0 ,an≠0,n∈ℕ

-the resulting polynomial is expressed by the expression:

R( x)=(an−bn) xn+(an−1−bn−1) xn−1+...+(a1−b1) x1+a0−b0

**public** Polynomial<Integer> substract(Polynomial<Integer> p1, Polynomial<Integer> p2) {

Polynomial<Integer> resultI = **new** Polynomial<Integer>();

**for** (AbstractMonom<Integer> m1 : p1.getPolynomial()) {

**for** (AbstractMonom<Integer> m2 : p2.getPolynomial())

**if** (m1.getDegree() == m2.getDegree()) {

resultI.addMonom(**new** IntegerMonom(m1.getDegree(), m1.getCoefficient() - m2.getCoefficient()));

**break**;

}

**if** (!existsMonom(m1, resultI))

resultI.addMonom(**new** IntegerMonom(m1.getDegree(), m1.getCoefficient()));

}

**for** (AbstractMonom<Integer> m2 : p2.getPolynomial())

**if** (!existsMonom(m2, resultI))

resultI.addMonom(**new** IntegerMonom(m2.getDegree(), -m2.getCoefficient()));

cleanPolynomial(resultI);

resultI.sortPolynomial();

**return** resultI;

}

the first for Iterates through the first polynom and for each monom of the second one (enhanced inner for) asks if the degree is equal,if true it substracts the coefficients and adds the element in result and does not search anymore (they are constructed as unique list of monoms) then after the first for asks is the monom already exists in result, if not it adds it because it means the first for has not add it because it has no term with the same degree. Then it takes again the second polynom and asks if it exists in result is not then it adds it with the reversed coefficient. This is the exact implementation of the algorithm. At the and it clears the polynom from terms with 0 coefficient and also sorts it based on the degree.

• ***Multiplication***

*For the multiplication operation it is better to provide an example as the*

*mathematical model is rather difficult to follow*

*Example:* ***P(x)****=2x^2 + 3x^1 +1*

***Q(x)****=6x^1 + 2*

*We have to multiply each term from the first polynomial with all the terms from the*

*second polynomial. One would obtain:*

*2x^2 \* 6x^1 + 2x^2 \* 2 = 12x^3 + 4x^2*

*3x^1 \* 6x^1 + 3x^1 \* 2 = 18x^2 + 6x^1*

*1\*6x^1 + 1\*2 = 6x^1 + 2 +*

***R(x)****=12x^3 + 22x^2 + 12x^1 + 2`*

**public** Polynomial<Integer> multiply(Polynomial<Integer> p1, Polynomial<Integer> p2) {

Polynomial<Integer> resultI = **new** Polynomial<Integer>();

**for** (AbstractMonom<Integer> m1 : p1.getPolynomial()) {

Polynomial<Integer> result = **new** Polynomial<Integer>();

**for** (AbstractMonom<Integer> m2 : p2.getPolynomial()) {

IntegerMonom monom = **new** IntegerMonom(m1.getDegree() + m2.getDegree(),

m1.getCoefficient() \* m2.getCoefficient());

result.addMonom(monom);

}

resultI = add(resultI, result);

}

cleanPolynomial(resultI);

resultI.sortPolynomial();

**return** resultI;

}

The first for iterates through the first polynom and multiplies the current coefficent with the coefficient of every monom of the second polynom and adds every element in the partial result “result”. After the inner for it adds the partial result in the final result “resultI” using the previous implemented method. This is the exact implementation of the algorithm. At the and it clears the polynom from terms with 0 coefficient and also sorts it based on the degree.

• ***Differentiation***

*The mathematical model for the differentiation operation is defined as follows:*

*-we have one polynomials to differentiate, meaning P(x) or Q(x)*

*-the polynomial is expressed by the expression:*

*-therefore the differentiated polynomial will have the following form:*

*P '* ( *x*)=*n\*a(n)\* x^n*−1+(*n*−1)\**a(n*−1) \**x^n*−2+...+*a*1

**public** Polynomial<Integer> derivate(Polynomial<Integer> p1) {

Polynomial<Integer> resultI = **new** Polynomial<Integer>();

**for** (AbstractMonom<Integer> m1 : p1.getPolynomial()) {

**if** (m1.getDegree() != 0) {

resultI.addMonom(**new** IntegerMonom(m1.getDegree() - 1, m1.getCoefficient() \* m1.getDegree()));

}

}

resultI.sortPolynomial();

**return** resultI;

}

It simply iterates with an enchanced for trough the polynomial and multiplies for each monom the coefficient with the degree and then it decrements the degree but only if the degree is different then 0.Adds all the monom in the resultI and then it sorts it(although it is not necesary) and returns the result.

• ***Integration***

*The mathematical model for the differentiation operation is defined as follows:*

*-we have one polynomials to differentiate, meaning P(x) or Q(x)*

*-the polynomial is expressed by the expression:*

*P*( *x*)=*an\* x^n*+*an*−1 \**x^n*−1+...+*a*1\* *x*1+*a*0 *, an*≠0, *n*∈ℕ

***-****therefore the integrated polynomial will have the following form*

∫*P*( *x*)=*an* /(*n*+1)\* *x^*(*n*+1)+*an*−1 /*n\* x^n*+....+*a*1 /2 *x^*2+*a*0 /1 *x*1

**public** Polynomial<Double> integrate(Polynomial<Integer> p1) {

Polynomial<Double> resultI = **new** Polynomial<Double>();

**for** (AbstractMonom<Integer> m1 : p1.getPolynomial()) {

**if** (m1.getDegree() != 0)

resultI.addMonom(**new** DoubleMonom(m1.getDegree() + 1, m1.getCoefficient() / m1.getDegree()));

**else**

resultI.addMonom(**new** DoubleMonom(m1.getDegree() + 1, m1.getCoefficient()));

}

resultI.sortPolynomial();

**return** resultI;

}

Method iterates through the polynomial and for each monom it divide the coefficent with the degree and increments the degree, but only for degrees higher then 0 because it cannot divide by 0. Treats the constant separately in the else brunch were it only increments is degree.

D) PolynomialDivide

The separate class is needed for the divizion because it has helper functions defined with double coefficients but that resamble exactly the alghorithms for integer coefficients:

**public** Polynomial<Double> rest = **new** Polynomial<Double>()

**public** **boolean** existsMonom(AbstractMonom<Double> monom, Polynomial<Double> p)

**public** **void** cleanPolynomial(Polynomial<Double> p)

**public** Polynomial<Double> substract(Polynomial<Double> p1, Polynomial<Double> p2)

**public** Polynomial<Double> multiplyMonom(Polynomial<Double> p1, DoubleMonom monom)

This all methods make the divizion easier to be understand:

• ***Division***

*For the division I applied the classical algorithm of division presented below with an*

*example. For the division the following mathematical relations take place:*

*P(x) = D(x) \* Q(x) +R(x) where*

*P(x)- dividend, D(x)- divisor, Q(x)- quotient, R(x)- remainder*

*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)*

*Example: 2x^3 + 3x^2 – x^ 1 + 5 | x^2 – x ^1 +1*

*-----------------------------------------------------------------*

*- 2x^3 + 2x^2 – 2x^1 | 2x^1 + 5*

*/ 5x^2 – 3x^1 + 5*

*-5x^2 + 5x^1 – 5*

*------------------------------------------------------------------*

*/ 2x^1 /*

*P(x)= 2x^3 + 3x^2 – x^ 1 + 5, D(x)= x^2 – x ^1 +1*

*Therefore the remainder is R(x)= 2x^1 and the quotient is Q(x)= 2x^1 +5;*

**public** Polynomial<Double> divide(Polynomial<Integer> p1, Polynomial<Integer> p2) {

Polynomial<Double> middle = **new** Polynomial<Double>();

Polynomial<Double> bottom = **new** Polynomial<Double>();

Polynomial<Double> divizor = **new** Polynomial<Double>();

DoubleMonom lastDivizorMonom = **new** DoubleMonom(0, 1);

DoubleMonom lastMiddleMonom = **new** DoubleMonom(0, 1);

;

**for** (AbstractMonom<Integer> m : p1.getPolynomial()) {

middle.addMonom(**new** DoubleMonom(m.getDegree(), m.getCoefficient()));

}

**for** (AbstractMonom<Integer> m : p2.getPolynomial()) {

divizor.addMonom(**new** DoubleMonom(m.getDegree(), m.getCoefficient()));

lastDivizorMonom = **new** DoubleMonom(m.getDegree(), m.getCoefficient());

}

**int** n = p1.getDegree() - p2.getDegree();

Polynomial<Double> result = **new** Polynomial<Double>();

rest= **new** Polynomial<Double>();

**do** {

**for** (AbstractMonom<Double> m : middle.getPolynomial()) {

lastMiddleMonom = (DoubleMonom) m;

}

**if** (lastMiddleMonom.getDegree() >= lastDivizorMonom.getDegree()) {

DoubleMonom monom = **new** DoubleMonom(n,

lastMiddleMonom.getCoefficient() / lastDivizorMonom.getCoefficient());

result.addMonom(monom);

bottom = multiplyMonom(divizor, monom);

middle = substract(middle, bottom);

}

n--;

} **while** (middle.getPolynomial().size()!=0 && n >= 0);

result.sortPolynomial();

rest = middle;

rest.sortPolynomial();

**return** result;

}

I put the first polynomial in the middle variable and the second polynomial in the divizor variable. Then i always need the higher degree monom of the divizor wich i put in the lastDivizorMonom. The degree of the result will be the difference of the 2 polynomial degree.

So i initialize variable n with the result degree and as long as n>0 and the middle has elements i do the previous described algorithm: I always take the higher degree of middle and divide it by lastDivizorMonom. the result gets this monom and then i multiply the divizor with this obtained monom and substract it from middle. I also check the condition that middle needs to have >= degree then the divizor. If not the algorithm stops and middle will contain the rest. Then i clean the result i sort it and i return it. the alghorithms works very well but it is logical even when p1 has lower degree then p2 when the quotient will be 0 and the rest will be p1.

**package** pt.polynomials.views;

Has two classes for displayng the user interface. They extend javax.swing.JFrame and use JButtons and JTextFields. All elements are added in the constructor of each class. They also have inner classes that implement java.awt.event.ActionListener to handle the buttons pressed or the text inserted in the texts fields.

**public** **class** Board **extends** JFrame

**public** **class** StartWindow **extends** JFrame

4.Implementation an testing

I implemented all the classes in Eclipse. I tested them as i created them in Main() class because i had to be sure the methods work before i put them in the GUI. There were not very hard to be put in practice as we learned the methods to process polynomials in highschool and the use of ArrayList made some of the additional operation very easy, adding elements, sorting , iteration and i had only to concentrate on the alghorithm. Also a problem that was risen was the reading of the polynom. To avoid any confusion I let the user introduce only the coefficents as an array separeted by commas, as it is easier then typing all the power and also makes the construction in a sorted order. I found that this was the most efficient way to make the input.

Insert polynoms like an array of integers separated by commas

"Exemple: 2,1,0,-12 will be 2+(1\*x^1)+(-12\*x^3)”

5.Results

The application is easy to use, you only insert the terms and press the buttons. It is also user friendly. But as a big observation i need to say that i should have add an exception handler at the input in case someone introduces an incorect input.

6.Conclusions

As things that i learned during this assignment i need to point out a few things:

The modelling took the most of the time because i was not sure what way to approach(vector vs ArrayList) .I choose to do the one that in theory would be more efficient and found out that it was a good choice and a good start makes the job very easy for you.

Second i was very pessimistic about the operations as they were very easy for human hand and seemd farfetched for the computer, but again i learned that a good logic and a transition from mathematical approach-> pseudocode-> java syntax can really work and the computer can be smarter then you think.

Last but not least is the time organisation, as implementation can generate big errors you should try to do things before the immediate comming of the due date as you do not really know how much time yu will need.

7. *Bibliography*

<https://www.mathsisfun.com/-for> polynomial algorithms

<http://www.tutorialspoint.com/-> for using collections, how to make umls