**Homework 1**

1. **Objective**

The main objective of this homework is to propose, design and implement a system for polynomial processing. The polynomials have one variable and integer coefficients.

The secondary objectives are:

1. **Designing a structure that can store a simple monomial.** Because a polynomial consists of one or more monomials we need such a structure. This is detailed in cap 2 and the implementation can be found in cap 4;
2. **Designing a structure that can store a polynomial.** We need this to be able to process the polynomials. This is detailed in cap 2 and the implementation can be found in cap 4.
3. **Designing a way bring the polynomial to a canonical form.** This is done by the collapse method which can be found in cap 3 and the implementation in cap 4;
4. **Implement the addition operation.** One of the required operations. The implementation can be found in cap 4.
5. **Implementing the subtraction operation.** One of the required operations. The implementation can be found in cap 4;
6. **Implementing the product operation.** One of the required operations. The implementation can be found in cap 4;
7. **Implementing the division operation.** One of the required operations. The implementation can be found in cap 4;
8. **Implementing the differentiation operation.** One of the required operations. The implementation can be found in cap 4;
9. **Implementing the integration operation.** One of the required operations. The implementation can be found in cap 4;
10. **Designing a way to read the input from the user and to output the results.** The polynomials and a way to choose the desired operation are needed in the GUI; This discussed in cap 3 and cap 4.
11. **Implementing the Graphic User Interface (GUI).** Final step where we make a simple GUI for the user to interact with. Can be found in cap 3.
12. **Problem analysis, modeling, scenarios, use cases**

There are several use cases for this problem. The “actor” is the user who wishes to find the result of a certain operation on polynomials. The system in this case is the Polynomial Processing System (PPS). Let us see the use cases:

1. Use case (addition)

-The user introduces the first polynomial;

-The user introduces the second polynomial;

-The user presses the “+” button;

-The PPS displays the result on the first line of **Results**.

1. Use case (subtraction)

-The user introduces the first polynomial;

-The user introduces the second polynomial;

-The user presses the “-” button;

-The PPS displays the result on the first line of **Results**.

1. Use case (product)

-The user introduces the first polynomial;

-The user introduces the second polynomial;

-The user presses the “\*” button;

-The PPS displays the result on the first line of **Results**.

1. Use case (division)

-The user introduces the first polynomial;

-The user introduces the second polynomial;

-The user presses the “/” button;

-The PPS displays the result at **Results**. The first line is the quotient and the second is the remainder;

1. Use case (differentiation)

-The user introduces a polynomial on the first or second line or both;

-The user presses the “d” button;

-The PPS displays the result on the first or second line of **Results**.

1. Use case (integration)

-The user introduces a polynomial on the first or second line or both;

-The user presses the “I” button;

-The PPS displays the result on the first or second line of **Results**.

Of course, there can be several cases where the user tries to do an illegal action or an illogical one. One example is trying to divide by 0 or to integrate a monomial with the power =-1 which would output ln. For such cases exceptions are thrown, a message is displayed, and the PPS requests another input. If the user presses on the buttons of the operation before introducing any polynomials, both operands will be considered 0.

1. **Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)**

The system required is perfect for the OO approach. Firstly, we need a structure to store the Monomials. A Monomial is characterized by the coefficient and the power of the variable. Although the coefficients in the problem are integers, we will use the float type because of the integration of the polynomials.

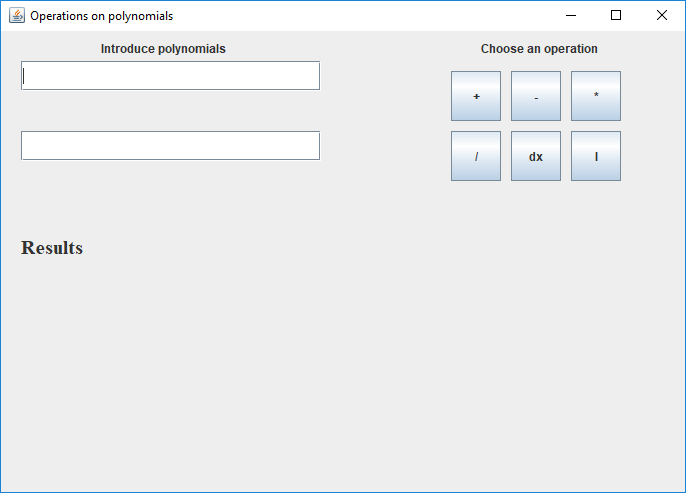
Secondly, we need a way to store all the Monomials as a Polynomial, so we use an Array List of Monomials to handle them easier. In the Polynomial class we implement all of the required operations as static methods, so they can be accessed by other classes like the Controller.

We need a comparator in the Comp class to sort in ascending order the Monomials in Polynomial in order to get the degree of the highest Monomial in the Polynomial and to get the coefficient of the highest Monomial.

In Controller we take the input coming from the Gui to analyze and compute the required operations and to output back to the Gui the results which will be displayed.

There are 3 packages: polynomial which contains the classes Polynomial, Monomial and Comp; gui which contains the Gui (and TheHandler); controller which contains the Main and Controller.

The user interface is a simple input and output GUI. Here we can introduce the polynomials in the text fields and by pressing enter save them. The polynomials must respect the input format (which can be seen by hovering the mouse over the text fields) otherwise a message is displayed with “Incorrect input”. After this, the operation should be chosen, and the result will be displayed below Results with floating point numbers for coefficients. If the result is 0, there will be nothing displayed.



1. **Implementation**

The most important classes used were:

1. Monomial

Here we have a coefficient of type float, the power of type int and getters and setters that we need later. The method toString has been overridden to better display the monomial.

1. Polynomial

Here we can find a list of Monomials and several methods to add Monomials to this list. Another method used is collapse.

**public** **void** collapse() {

**for** (**int** i = 0; i < **this**.pol.size() - 1; i++ ) {

**for** (**int** j = i + 1; j < **this**.pol.size(); j++ ) {

**if** (**this**.pol.get(i).getPower() == **this**.pol.get(j).getPower()) {

**this**.pol.get(i).setCoeff( **this**.pol.get(i).getCoeff() + **this**.pol.get(j).getCoeff());

**this**.pol.remove(j);

}

}

}

**for** (**int** i = 0; i < **this**.pol.size(); i++ ) {

**if** ( **this**.pol.get(i).getCoeff() == 0 )

**this**.pol.remove(i);

}

Collections.*sort*(pol, **new** Comp());

}

Here all the Monomials with the same power get summed and the Polynomial is sorted with the help of Comp.

The addition is done by parsing each input Polynomial and collapsing the resulted Polynomial.

**public** **static** Polynomial addition( Polynomial a, Polynomial b ) {

Polynomial c = **new** Polynomial();

**for** ( Monomial Ita : a.pol )

c.addMonomial( Ita.getCoeff(), Ita.getPower() );

**for** ( Monomial Itb : b.pol )

c.addMonomial( Itb.getCoeff(), Itb.getPower() );

c.collapse();

**return** c;

}

The subtraction is done in the same fashion.

**public** **static** Polynomial subtraction( Polynomial a, Polynomial b ) {

Polynomial c = **new** Polynomial();

**for** ( Monomial Ita : a.pol )

c.addMonomial( Ita.getCoeff(), Ita.getPower() );

**for** ( Monomial Itb : b.pol )

c.addMonomial( (-1) \* Itb.getCoeff(), Itb.getPower() );

c.collapse();

**return** c;

}

For the product, we take 2 Polynomials and create a new Monomial for each Monomial in a and b, summing the powers and multiplying the coefficients.

**public** **static** Polynomial product( Polynomial a, Polynomial b ) {

Polynomial c = **new** Polynomial();

**for** ( Monomial Ita : a.pol ) {

**for** ( Monomial Itb : b.pol ) {

c.addMonomial( Ita.getCoeff() \* Itb.getCoeff(), Ita.getPower() + Itb.getPower() );

}

}

c.collapse();

**return** c;

}

The Division presents a special case. Here we have c as an array of size 2 of Polynomials: c[0] will represent the quotient of the division and c[1] will be the remainder. Here we use the classical algorithm of Polynomial division: c[1] will take the value of a, and as long as the degree of c[1] is greater than the degree of b we add a monomial to the intermediary quotient d and c[0] becomes the product of b and the monomial and c[1] becomes the subtraction of c[1] and c[0]. At the end, c [0] takes the values from d and c is returned. This method uses another 3 private methods : MonomialPolynomialProduct which takes a Monomial and a Polynomial and return their product in a polynomial, degree and coeff which return the power or the coefficient of the greatest variable in the input polynomial.

**public** **static** Polynomial[] division( Polynomial a, Polynomial b ) {

Polynomial[] c = **new** Polynomial[2];// 1st quotient 2nd the remainder

c[0] = **new** Polynomial();

c[1] = **new** Polynomial();

Polynomial d = **new** Polynomial();

**for**( Monomial Ita : a.pol )

c[1].addMonomial( Ita.getCoeff(), Ita.getPower() );

**while** ( !c[1].pol.isEmpty() && *degree*( c[1] ) >= *degree*( b ) ) {

Monomial m = **new** Monomial( *coeff*( c[1] ) / *coeff*( b ), *degree*( c[1] ) - *degree*( b ) );

d.addMonomial(m);

c[0] = *MonomialPolynomialProduct*( m, b );

c[1] = *subtraction*( c[1], c[0] );

}

c[0] = d;

**return** c;

}

Differentiation is done by multiplying the coefficient with the power (for each Monomial) IF the power is not 0 (in which case the Monomial is not taken in consideration). The power is subtracted by 1.

**public** **static** Polynomial differentiation( Polynomial a ) {

Polynomial c = **new** Polynomial();

**for** ( Monomial Ita : a.pol) {

**if** ( Ita.getPower() != 0 )

c.addMonomial( Ita.getCoeff() \* Ita.getPower(), Ita.getPower() – 1 );

}

**return** c;

}

Integration has a special case. That case occurs when the power of one of the Monomials from the input Polynomial is -1. This means that the result will be a ln of x. For this case an exception is thrown.

**public** **static** Polynomial integration( Polynomial a ) {

Polynomial c = **new** Polynomial();

**for** ( Monomial Ita : a.pol ) {

**if** ( Ita.getPower() != -1 )

c.addMonomial(Ita.getCoeff() / ( Ita.getPower() + 1 ), Ita.getPower() + 1 );

**else**

{

**throw** **new** IllegalArgumentException( "Monomials with the power = -1 will output ln" );

}

}

**return** c;

}

The toString method has been overridden to better suit the output of the polynomial.

1. Controller

The controller takes the input and does the necessary operations. An important method from this class is StringToPoly and checkString. These are used to check if the input is under the correct form and to make the conversion from a string to an Object of type Polynomial.

**public** **boolean** StringToPoly( String s, **int** pol ) {

StringTokenizer multiTokenizer = **new** StringTokenizer( s, " " );

String a = **new** String();

String nr = **new** String();

**int** coeff = 0;

**int** power = 0;

**while** ( multiTokenizer.hasMoreTokens() ) {

a = multiTokenizer.nextToken();

**if** (checkString( a )) {

StringTokenizer Token2 = **new** StringTokenizer( a, "x^" );

**while** (Token2.hasMoreTokens()) {

nr = Token2.nextToken();

coeff = power;

power = Integer.*parseInt*( nr );

}

polynomials[pol].addMonomial( coeff, power );

}

**else** {

polynomials[pol].getPol().clear();

**return** **false**;

}

}

polynomials[pol].collapse();

**return** **true**;

}

**public** **boolean** checkString( String s) {

Matcher matcher = pattern.matcher(s);

**boolean** matches = matcher.matches();

**return** matches;

}

1. The Gui

The Gui is used to display the information and to take the input from the user.

1. **Results**

Testing was done with JUnit and here the operations were tested with 2 polynomials as input ( done in initialize). The polynomials are:

a = x^3 -4x^2 +2x – 3

b = x + 2

There were 6 Tests done, one for each operation: addition, subtraction, division, product, integration and differentiation. The test works by checking if the output of toString from each operation is equal to the correct answer (computed by hand). As can be seen, all the tests are positive.

1. **Conclusions**

This homework was a nice introduction to the more complex projects yet to come. I believe, after this homework, that I understand better the Object Oriented Design and how to work with bitbucket and github.

1. **Biography**

For the division of polynomials:

http://www.mesacc.edu/~scotz47781/mat120/notes/divide\_poly/long\_division/long\_division.html