Technical University of Cluj-Napoca

Student: Natalia Carina Sîrbu

Computer Science, 2nd year, group 30425

March 13, 2017

**Processing operations on polynomials**

Homework 1

Table of Contents

[1. Introduction to polynomials 3](#_Toc477293643)

[2. Objectives 3](#_Toc477293644)

[2.1. Main objective 3](#_Toc477293645)

[2.2. Problem analysis and approach 3](#_Toc477293646)

[3. Scenarios 4](#_Toc477293647)

[4. Design 4](#_Toc477293648)

[4.1. Class diagram 4](#_Toc477293649)

[4.2. Data structures 6](#_Toc477293650)

[4.3. Class details 6](#_Toc477293651)

[4.3.1. Term 6](#_Toc477293652)

[4.3.2. Polynomial 6](#_Toc477293653)

[4.3.3. Operation 7](#_Toc477293654)

[4.3.4. View 9](#_Toc477293655)

[4.3.5. Controller 9](#_Toc477293656)

[5. Testing 10](#_Toc477293657)

[6. Usage and further development 10](#_Toc477293658)

[7. Lessons learned 11](#_Toc477293659)

[8. Bibliography 11](#_Toc477293660)

# **Introduction to polynomials**

In order to get a better understanding of the project, the notion of polynomial should be clear. A polynomial has a simple and accessible form that is widely use in areas of mathematics and science. In fact, it is an expression that consists in constants, variables and exponents.

Basic mathematical operations can be performed on polynomials, such as addition, subtraction, multiplication, division, differentiation and integration.

# **Objectives**

## Main objective

The purpose of this project is to design and implement a system for polynomial processing using the object-oriented programming. The polynomial should have only one variable.

The approach used for this project was the bottom-up method. This means that we begin with the base elements of the problem which are presented with a great amount of details. After that, these elements will form a larger system when linked together. We can look at this approach as a “seed” model, when “beginnings are small but eventually grow in complexity and completeness”.

One starting point for this project is to look for different nouns related to the subject, that when chosen correctly, become the name of classes. Inside of each class, the methods will be named according to different verbs chosen from the field of the subject.

After the algorithm is established, the graphical user interface should be developed. The graphical user interface is a visual way in which one may interact with the computer using graphical icons, windows and menus. Considering the fact that the user is not necessarily a programmer or a person with basic programming knowledge, the interface should be user friendly, displaying messages that can guide the user.

## Problem analysis and approach

As the method used points, the first thing done was to create the base element of the polynomial: the term. It is also known as the monomial, meaning a polynomial with only one term. Moreover, the class of polynomial was created and its methods implement the project requests.

The final form of the graphical user interface is accessible to a wide range of people. It is plain and simple and does not have any parts that can confuse the user in one way or another.

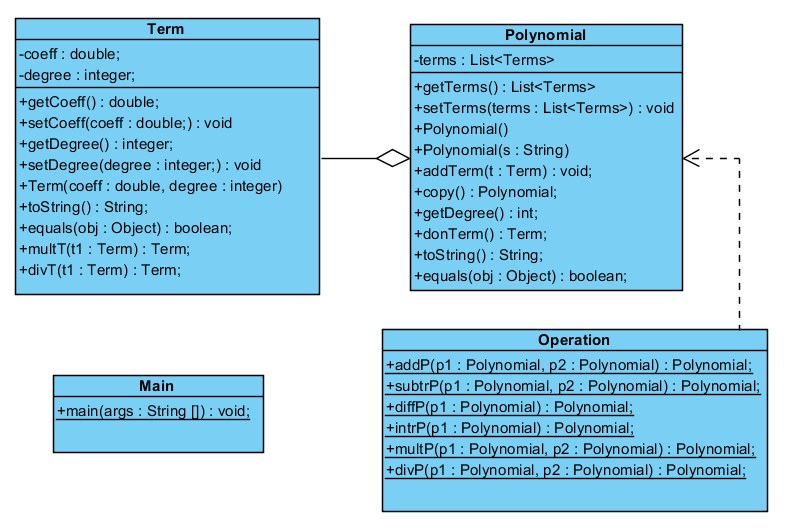
# **Scenarios**

This project is specifically intended for polynomials with integer coefficients and with only one variable. However, one may enter some wrong data unaware of the specifications. In this case, the application will fail. Such scenarios are listed below:

* the input consists only in letters (a – z, A - Z), except for ‘x’, which written like that is a valid input;
* the user inputs real numbers for the degrees (e.g. 2x **1/2**). Allowed inputs for the degree are in “int” range, where “int” is the data type for the degree;
* the user inputs negative numbers for the degree (e.g. x-9);
* the user inputs real numbers for the coefficient (e.g. -2x, 0.5x5 etc.);
* the user inputs some special characters. The only allowed special character is ‘^’ when used in the form ‘x^’;
* the input has one or many whitespaces between the terms of the polynomial.

# **Design**

## Class diagram

This class diagram was designed using UML, i.e. Unified Modeling Language.

The **Term** class will store a certain term of the polynomial that has a coefficient “coeff” of type double and a degree “degree” of type integer. The getters and setters for these attributes are in this class, along with the methods that apply to the instances of this class.

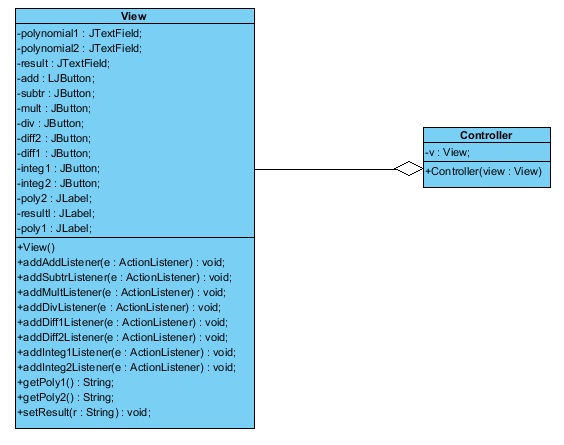
The relation between the **Term** class and the **Polynomial** class is an aggregation relationship which can be looked at as a “has a” relationship: a polynomial “has a” term.

The **Polynomial** class will construct the polynomial based on the terms from the class **Term**. Also, other methods that operate with polynomials are listed here and they will be explained further.

The **Operation** class contains all the operations that can be performed on the given polynomials. Every method from this class is static, therefore it should be called accordingly. The relation between the **Operation** class and the **Polynomial** class is a dependency relationship based on the fact that a change in one class will produce a change in the other class.

There is also the **Main** class with the main function. The only purpose of this class is to instantiate a view and a controller which uses the view, thus connecting them together.

The **View** and the **Controller** class diagram:



The **View** class is a part of the graphical user interface where all the visual elements are developed. Text fields, buttons and labels are declared here. Some methods here get the input from the text field, while other methods add action listeners to the button chosen.

The **Controller** class has only one purpose: to control the action performed by each button depending on the choice of the user.

## Data structures

In this project, two types of data structure were used:

1. In the **Term** class, the private attribute *terms* is declared as a list of terms, i.e. *List<Term>*, and used to store an *ArrayList<Term>*;
2. In the **Operation** class, we have the division, where in order to display the quotient and the remainder, an array will hold on the first position the value of the quotient and on the second position the value of the remainder.

## Class details

### Term

This class has two private attributes: *coeff* of type double and *degree* of type int (i.e. integer) that represent the coefficient and the degree of the object created by this class which is the term of the polynomial. Also, the getter and setter for each attribute are declared in this class.

* *public Term (double coeff, int degree)* – this is the constructor of the class which has two parameters for the object to be created: a degree (*int degree*) and a coefficient (*double coeff*).
* *public String toString()* -represents the overriding of the *toString()* method. This method will return a string that has the following form *coeff*x^*degree*. For the coefficient, in case of displaying decimals, only the first 3 will be shown. That is the reason why each term displays 3 ‘0’s when shown on the screen.
* *public Boolean equals (Object obj)* – this method has an object as parameters and it is necessary to be implemented because when we try to compare two terms in order to see if they are equal, the program will compare the address in the memory of the two polynomials. There are three *if* ‘s that verify whether the *obj* has the same address with *this*, if it is null or if is not of type *Term*. If none of these *if* ‘s will be entered, the *obj* will be cast to Term. The method returns a Boolean: true or false, depending on the degree and the coefficient of the two terms to be compared: *(this.degree == that.degree) && (this.coeff == that.coeff)*, where *that* is *Term that = (Term) obj*.
* *public Term multT (Term t1)* – this method returns the multiplication of two terms that is done in following way: the coefficients are multiplied and the degrees are added.
* *public Term divT (Term t1)* -this method returns the division of two terms performed in the following way: the coefficients are divided and the degrees are subtracted.

### Polynomial

The purpose of this class is to assemble the “polynomial” using the terms from the **Term** class. It has a private attribute called *terms* of type *List<Term>.* The getter and the setter for this attribute are also listed in this class.

* *public Polynomial ()* – this is the empty constructor of the class
* *public Polynomial (String s)* – this is also a constructor of the class, but this has a String as a parameter. This constructor will turn the input string into a term that can be used in the following operations (adding, subtraction, multiplication, division, differentiation and integration), a term with integer coefficient and degree. In order to do that, regular expressions (regex) were used. For this particular case, the pattern is: “([\\+|\\-)?((\\d\*x\\^?\\d\*)|(\\d+))](file:///\\+|\\-)%3f((\\d*x\\%5e%3f\\d*)|(\\d+)))”. For each term we have 7 cases:
* if the term matches “^([\\+|\\-)?(\\d+x\\^\\d+)$](file:///\\+|\\-)%3f(\\d+x\\%5e\\d+)$)”, it means we have a term of the form: ax^b;
* if the term matches “([\\+|\\-)?(\\d+x)](file:///\\+|\\-)%3f(\\d+x)) ”, it means we have a term of the form: ax;
* if the term matches “(\\+)?(x)”, it means we have a term of the form: x;
* if the term matches “([\\-)(x)](file:///\\-)(x))”, it means we have a term of the form: -x;
* if the term matches “(\\+)?(x\\^\\d+)”, it means we have a term of the form: x^b;
* if the term matches “([\\-)(x\\^\\d+)](file:///\\-)(x\\%5e\\d+))”, it means we have a term of the form: -x^b;
* if the term matches “^(([\\+|\\-)?(\\d+))$](file:///\\+|\\-)%3f(\\d+))$)”, it means we have a term of the form: a.
* *public void addTerm (Term t)* – this is the method that verifies if a certain term “t” is in the polynomial and if it is (i.e. if they have the same degree) their coefficients will be added, otherwise the new term will be added to the polynomial.
* *public Polynomial copy ()* – this is a very useful method when the terms of a polynomial should be preserved. Therefore, a copy of the polynomial is created.
* *public int getDegree()* – this is a method that returns the degree of a polynomial. The degree of the polynomial is the highest degree of its terms.
* *public Term domTerm()* – this method will return the dominant term, i.e. the term with the highest degree.
* *public String toString()* – this method returns the string representation of the terms of the polynomial.

### Operation

This class has only static methods and each method implements the basic mathematical operations requested: addition, subtraction, multiplication, division, differentiation and integration.

Addition

* *public static Polynomial addP (Polynomial p1, Polynomial p2)* – this is the method that performs the addition of the two polynomials given as parameters: p1 and p2.

Subtraction

* *public static Polynomial subtrP (Polynomial p1, Polynomial p2)­* – the subtraction method, as the name denotes, will return the difference between two terms. This action was done as an addition, but instead, the signs of the second polynomial were reversed. After that, they were added to the first polynomial using the method *addTerm(Term t)* from the **Polynomial** class that was imported in this class.

Differentiation

* *public static Polynomial diffP (Polynomial p1)* – this method will perform the first derivative of one of the input polynomials, by choice. The algorithm is basic: the coefficient will be multiplied with the degree and ‘1’ will be subtracted from the degree. The exception is when the degree of the polynomial is 0. Therefore, the index of that term is stored in an auxiliary variable and then is removed from the polynomial.

Integration

* *public static intrP (Polynomial p1)* – the integration method will be performed on one polynomial at the time and it gives a different result depending on the user’s input polynomial. The integration is done in the following way: the coefficient of the term is divided by the degree of the term and the degree will be incremented with ‘1’. There are no exceptions here, because when we have to integrate a constant (e.g. 2) the application will consider the polynomial to have the degree equal to ‘0’.

Multiplication

* *public static Polynomial multP (Polynomial p1, Polynomial p2)* – the multiplication method uses the input polynomials as parameters and perform the multiplication termwise. Each polynomial will be covered step by step and the terms will be multiplied using the *multT(Term t1)* method from the **Term** class. Therefore, each coefficient will be multiplied according to the method and the degree of the terms will be added.

Division

* *public static Polynomial divP (Polynomial p1, Polynomial p2)* – the division of the polynomials chosen by the user will be performed using the long polynomial division. The algorithm starts by initializing the quotient with an empty array of terms and the remainder with the dividend (i.e. p2). The “while” loop will be executed as long as the degree of the dividend is greater or equal to the degree of the divisor and as long as the dividend has terms. Moreover, the dominant term of each polynomial will be divided (using *divT* method from the **Term** class) and stored in the quotient. The remainder will hold the difference between the current remainder and the product between p2 and the terms from the quotient (*remainder = subtrP(remainder, multP(p2, term)*, where *term* is an auxiliary polynomial which holds the terms from quotient. Then the dominant term from the remainder will be removed. There are some exceptions that were not treated in this project such as: dividing two polynomials that have the same degree, dividing a constant to a polynomial. This method will return an array which has on the first position the value of the quotient and on the second position, the value of the remainder.

### View

The **View** class includes the declarations of the attributes needed to build the graphical user interface. This graphical user interface requests:

- 3 text fields: two for the input polynomials and one for displaying the result (the last one is read only, i.e. the user cannot insert anything here);

- 8 buttons, each for every operation to be performed: addition, subtraction, multiplication, division, differentiation and integration. For the differentiation and integration two buttons are available: the user can choose which one of the input polynomial is to be differentiated or integrated;

- 3 labels that display a text for the user to be guided in the application: “Polynomial1”, “Polynomial2” and “Result”.

In the constructor of the class, the bounds for each attribute is set and then added to the frame that will be displayed on the screen.

For each button from the graphical user interface, an action listener is needed in order to respond to the user request. This kind of implementation is necessary because the action of each button should be defined.

### Controller

The **Controller** class will take care of the mouse events. This will occur when the user opens the application and after he or she inputs the two polynomials and a button will be pressed.

An instance of the class **View** is created: *v*. Moreover, the class constructor has a parameter *view* of type *View*.

For each operation handled by this project a button was designed to implement it. In this class, for each one of the following: addition, subtraction, multiplication, division, differentiation and integration was created a private class to describe the action performed.

The classes are quite similar. In each class, one, two or three instances of the **Polynomial** class are created. The *setResult(String s)* method from the **View** class will be used to display the result as a String in the text field for the result. In the case of the division, in the *Result*’s test field will be displayed both the quotient and the remainder.

# **Testing**

The testing part was done using a unit testing framework. Junit was used to verify in the expected result of one of the basic mathematical operation: addition, subtraction, multiplication, division, differentiation and integration, is equal to the actual result. This was done by creating two instances of the class **Polynomial**, *p1* and p2, with terms, on which the operations will be performed. Furthermore, for each operation a method was created. For example, in the method for addition, the *actualResult* will be the polynomial resulted after adding *p1* and *p2* with the method *addP* from the **Operation** class. The *expectedResult* was created after the computation was done “on the paper”. This is a custom polynomial for each operation. Then, the *assertEquals (expectedResult, actualResult)* will assert if that the two polynomials are equal. The same procedure is done for every other operation: subtraction, multiplication, division, differentiation and integration.

# **Usage and further development**

One may use this application when it is the case to perform the following basic mathematical operations: addition, subtraction, multiplication, division, differentiation (of first degree) and integration.

Two polynomials will be used for the addition, subtraction, multiplication and division. The first derivative and the integration will use only one polynomial chosen by the user.

This is a pretty simplistic concept of the application. In the future, more features can be implemented using this skeleton.

For example, one may want to calculate the value of the polynomial in a certain point and such a function can be implemented (e.g. the value of the polynomial 3x4+6x when x = 3).

Another interesting function to be implemented will be to raise the polynomial to a certain power (e.g. (2x4+4)2).

The graphical user interface can be improved and the user would be able to keep track of the past operations. Also, checking for garbage input function was not implemented in this project.

Since the project is design to support only one variable, a great improvement would be to allow the usage of more variables.

# **Lessons learned**

During the development of this project, I came to a better understanding of the object-oriented programming. The division was quite challenging, but I learned how to implement the algorithm for the long division of the polynomials. The regular expressions (regex) represented the main point when I were to separate Strings in different ways and the *split()* method was not that useful. Also, I have learned how to use and take advantage of the great amount of functionalities provided by *ArrayList<>* class. My coding skills were improved and I have become more organized in my code.

# **Bibliography**

<https://www.mathsisfun.com/definitions/polynomial.html>

<https://en.wikipedia.org/wiki/Polynomial>

<http://junit.org/junit4/>

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

<https://www.tutorialspoint.com/java/java_regular_expressions.htm>