*Technical University of Cluj-Napoca*

*Faculty of Automation and Computer Science*

*2nd Semester 2019-2020*

**Programming Techniques**

**Homework 1**

**Polynomial Calculator**

Student: Osváth Tamás

Group: 30421

Teacher: Pop Cristina

Contents

[1. Objectives 3](#_Toc35278895)

[2. Problem Analysis, Modelling, Scenarios, Use Cases 3](#_Toc35278896)

[**2.1.** **Problem Analysis** 3](#_Toc35278897)

[**2.2.** **Modelling** 4](#_Toc35278898)

[**2.3.** **Scenarios, Use Cases** 4](#_Toc35278899)

[3. Design 7](#_Toc35278900)

[**3.1.** **Class Design and UML Diagram** 7](#_Toc35278901)

[**3.2.** **Packages and Relationships** 8](#_Toc35278902)

[**3.3.** **Algorithms and Data Structures** 8](#_Toc35278903)

[**3.4.** **Graphical User Interface** 9](#_Toc35278904)

[4. Implementation 9](#_Toc35278905)

[**4.1.** **Class Monomial** 9](#_Toc35278906)

[**4.2.** **Class Polynomial** 10](#_Toc35278907)

[**4.3.** **Class Operations** 11](#_Toc35278908)

[**4.4.** **Class View** 12](#_Toc35278909)

[**4.5.** **Class Model** 12](#_Toc35278910)

[**4.6.** **Class Controller** 13](#_Toc35278911)

[**4.7.** **Class Main** 13](#_Toc35278912)

[5. Usage and Testing 14](#_Toc35278913)

[6. Conclusions 15](#_Toc35278914)

[7. Bibliography 15](#_Toc35278915)

# Objectives

The main objective is to design and implement a system for polynomial processing and calculations. Consider the polynomials of one variable and integer coefficients.

In order to achieve the main objective, we have to take in consideration the following secondary objectives, implementation of:

* Addition of polynomials (chapters: 3.3, 4.3)
* Subtraction of polynomials (chapters: 3.3, 4.3)
* Multiplication of polynomials (chapters: 3.3, 4.3)
* Division of polynomials (chapters: 3.3, 4.3)
* Derivation of polynomials (chapters: 3.3, 4.3)
* Integration of polynomials (chapters: 3.3, 4.3)
* Graphical user interface (chapter: 3.4, 4.4)

# Problem Analysis, Modelling, Scenarios, Use Cases

## **Problem Analysis**

A system for processing polynomials may be useful in almost any environment that uses mathematics. Polynomials and corresponding operations are frequently used in engineering, Artificial Intelligence and many other fields. Also, such a system can be used by non-professionals too, such as high school students to verify their solutions.

The system can be used in many fields because it provides the most basic and mainly used operations on polynomials, such as addition, subtraction, multiplication, division, derivative and integration.

Also, the using of the application is easy, because of the user friendly Graphical User Interface, with text boxes and buttons, no console-based operations needed.

## **Modelling**

For a system of processing polynomials the main objectives in terms of modelling is the modelling of a polynomial. A polynomial consists of a list of monomials, so the first thing is to model the Monomial entity. Monomials have two important attributes: coefficient and exponent. A polynomial in fact means a list of monomials, and the operations on polynomials are in fact operations on the monomials.

## **Scenarios, Use Cases**

The user introduces one or two polynomials (notation: P and Q), after that he chooses the type of operation he wants to perform and the result is printed. In case the operation is derivation or integration, the first input (P) will be considered as the input for the operation.

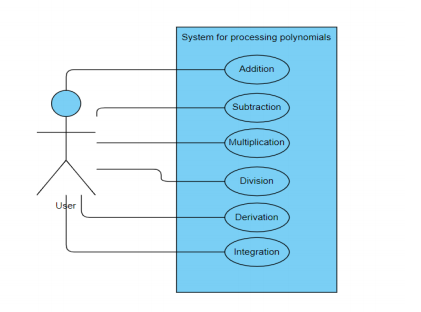
The correct format for the introduction of the data is: coefficient x ^ exponent, exponent should be written even if it is 1 or 0.

An example would be:

P = 1x^2 – 1x^0; Q = 1x^1 – 1x^0; P / Q = 1x^1 + 1x^0;

After inputs it is selected the operation “Division”, and the result will be prompted to the “Result” text box, which is not editable.

The use-case diagram is the following:



**Use case title: Addition**

Actor: User

Main success scenario:

1. User introduces the polynomials (input1 and input2)
2. User chooses addition by pressing the ‘Addition’ button
3. System takes information from the input
4. The operation is executed
5. System displays the result

**Use case title: Subtraction**

Actor: User

Main success scenario:

1. User introduces the polynomials (input1 and input2)
2. User chooses addition by pressing the ‘Subtraction’ button
3. System takes information from the input
4. The operation is executed
5. System displays the result

**Use case title: Multiplication**

Actor: User

Main success scenario:

1. User introduces the polynomials (input1 and input2)
2. User chooses addition by pressing the ‘Multiplication’ button
3. System takes information from the input
4. The operation is executed
5. System displays the result

**Use case title: Division**

Actor: User

Main success scenario:

1. User introduces the polynomials (input1 and input2)
2. User chooses addition by pressing the ‘Division’ button
3. System takes information from the input
4. The operation is executed
5. System displays the result

**Use case title: Division**

Actor: User

Main failure scenario:

1. User introduces the first polynomial, and 0 for the second
2. User chooses addition by pressing the ‘Division’ button
3. System takes information from the input
4. System displays an error message since the division with 0 is not possible

**Use case title: Derivative**

Actor: User

Main success scenario:

1. User introduces the polynomial (input1)
2. User chooses addition by pressing the ‘Derivative’ button
3. System takes information from the input
4. The operation is executed
5. System displays the result

**Use case title: Integration**

Actor: User

Main success scenario:

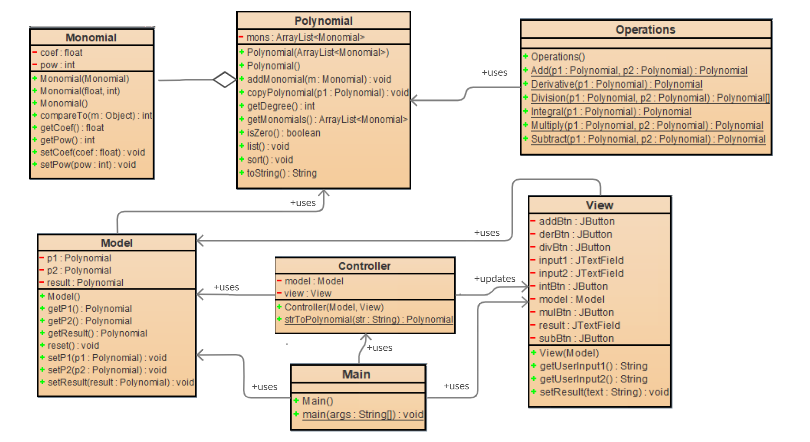
1. User introduces the polynomial (input1)
2. User chooses addition by pressing the ‘Integrate’ button
3. System takes information from the input
4. The operation is executed
5. System displays the result

# Design

## **Class Design and UML Diagram**

The design of the assignment is based on the **M**odel **V**iew **C**ontroller architectural pattern, because the interaction between the user and the system is done through a **G**raphical **U**ser **I**nterface. From this pattern, the Model is the one in which we are mostly interested.

A polynomial is represented as a list of monomials, thus three important classes stand at the base of the Model. The Monomial class represents a monomial having 2 attributes: the coefficient and the exponent. The Polynomial class is in fact a list of monomials. The Operations class implements static methods for all the operations on polynomials.



UML Diagram for the Polynomial Calculator

The class View is responsible for the graphical user interface, providing text fields and buttons, and it uses the model. A GUI increases productivity and makes more easier to handle the application rather than having a console-based one.

The class Controller connects the elements from model and view, by taking data from the user (input) and controls the data flow to the model, retrieving the corresponding results.

## **Packages and Relationships**

According to the MVC pattern, the package is structured in 3 packages:

* Model – contains the Model, Monomial, Polynomial and Operation classes, these being part of the Model entity from the pattern.
* Gui – contains the View and Controller classes. As discussed, they implement the graphical user interface of the application and therefore can be in the same package.
* Polynomial – contains the Main class, which runs the application

The relationship between Monomial and Polynomial can be defined as aggregation, since it is a ‘Has-a’ relationship: a Polynomial can have multiple Monomials.

The other relationships are mainly association, for example Controller class updates field from View class, Main class uses information from Controller, Main and View classes, and so on, as seen on the UML Diagram.

## **Algorithms and Data Structures**

We can talk about 6 operations that are used in this implementation, namely the addition, subtraction, multiplication, division, integration and differentiation. These operations appear in the Model package, Operations class.

The most simple and trivial algorithm is the addition and subtraction of monomials with the same exponent, the only thing to do is to change the coefficient of the regarding monomial.

The algorithms used for the operations are trivial, they can be implemented using high school methods. The addition and subtraction of the polynomials is trivial, they only take the monomials term-wisely and do the corresponding operation. The multiplication is a bit more complex, it basically multiplies the polynomials term-wisely (coefficient \* coefficient, power + power), adding the result to the new polynomial. The division was the most challenging, but it is also based on high school methods, repeating simple divisions on the quotient. Differentiation and integration are also simple algorithms, term-wisely we apply the corresponding formula (incrementing/ decrementing the power, multiplying/ dividing the coefficient).

After any operation, a sorting function is used for normalization, in order to ensure that terms with the same power appear only once and the terms are sorted decreasingly regarding their exponent.

The data structures used are:

* ArrayList for the monomials of a polynomial
* Array of dimension 2 for the Division operation: on the first position is the quotient, on the second one the remainder

## **Graphical User Interface**

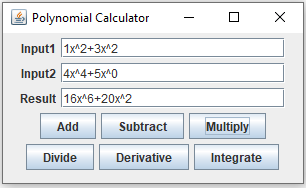
The graphical user interface provides an easy handling of the system for the user. It has some text fields (for introducing data and output of the result) and buttons.

The buttons are representative since they have a label describing shortly the operation it performs.

The GUI is simple, but meaningful, since it displays all the relevant information about the application. To be mentioned that it was designed exclusively manually, no helper applications.

If the input format for the monomial is incorrect, it is simply not considered in the input.

If the Input2 is equal to 0, we get an error message since the division by 0 is not possible.



# Implementation

## **Class Monomial**

The monomial class represents a monomial in order to be added to a polynomial.

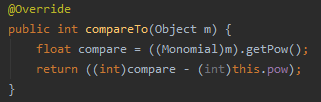
The fields of the class are:

* Private float coeff – the coefficient of the monomial
* Private int pow – the power of the monomial

The class has 3 constructors:

* Monomial() – sets both attributes to 0
* Monomial(coef, pow) – creates a monomial with the given attributes
* Monomial(mono) – creates a copy of the mono Monomial (Copy constructor)

Besides the getters and setters, there is an overridden method named compareTo(), which compares a monomial to another monomial by their power and coefficient, in order to be able to sort the monomials of a polynomial. For this, it is also needed that the Monomial class to implement the Comparable class.



## **Class Polynomial**

The Polynomial class represents a polynomial and implements

methods for representing it.

The field of the class is an ArrayList of monomial objects,

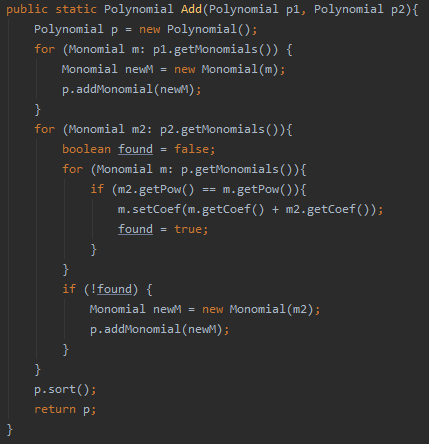
which compose the polynomial itself.

It also contains a method named sort(), which sorts the monomials

of the polynomial in order to have de powers in descending order.

## **Class Operations**

The operations class implements methods for all the operations.

* **Public static Polynomial Add(Polynomial p1, Polynomial p2)** – Addition of polynomials
* The function assigns to a new Polynomial p the

value of p1. Then, for every element from p2 it

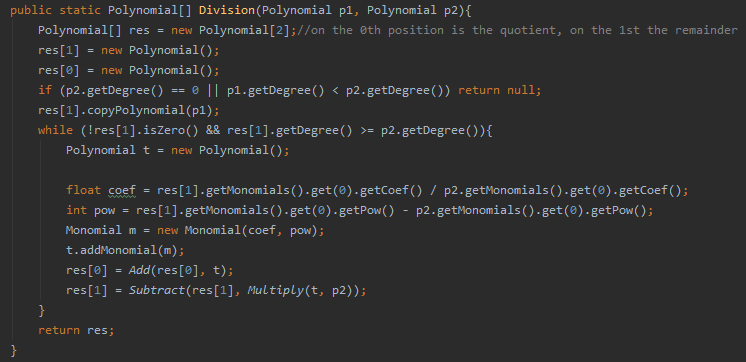
is checked if there is a term in p with the same

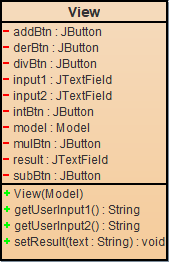
power. If yes, they are added together. If there is

no match, a flag remains unchanged, and the

element is added to the result as a monomial.

* The result is sorted and returned.
* **Public static Polynomial Subtract(Polynomial p1, Polynomial p2)**– Subtraction of polynomials
* The same algorithm as for addition
* **Public static Polynomial Multiply(Polynomial p1, Polynomial p2)** – Multiplication of polynomials
  + The algorithm is simple, every term from p1 is being multiplied with every term from p2 (in fact, it is an addition of the coefficients).
  + If resulting the multiplication there are 2 terms in the polynomial with the same power, their coefficients are added.
* **Public static Polynomial[] Divide(Polynomial p1, Polynomial p2)** – Division of polynomials
  + The algorithm is the same as in mathematics with repeated divisions of the remainder
  + On the first position of the returned array is the quotient, on the second the remainder.



* **Public static Polynomial Derivative(Polynomial p1)** – Derivative of a polynomial
  + Every monomial from the polynomial is differentiated separately and added to a new polynomial which is returned
* **Public static Polynomial Integral(Polynomial p1)** – Integral of a polynomial
  + Every monomial from the polynomial is integrated separately and added to a new polynomial which is returned

## **Class View**

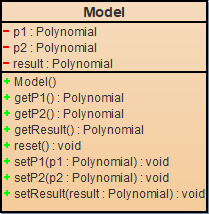
The View class implements the view of the application. It is

implemented using the Swing library, having elements like

buttons, text fields, labels, panels.

In the constructor, every component is prepared and added to the panel. There are 3 TextFields and 6 JButtons, one for each operation.

It also contains methods for getting user input (getUserInput1() and getUserInput2()), setting the result, and methods for adding listener for each button (Addition button, Subtraction button, etc.)



## **Class Model**

The Model class implements the structure of the model

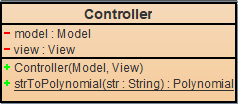
from the MVC design pattern. It creates polynomials in

order the Controller to be able to manipulate them through

setters. It also has a reset() method which resets the polynomials.

## **Class Controller**

Controller class creates a view and a model instance, connecting them together.

This class has a function, strToPolynomial(String str),

which using a Regex function extracts all the monomials

from a polynomial, and returnes a Polynomial type object:

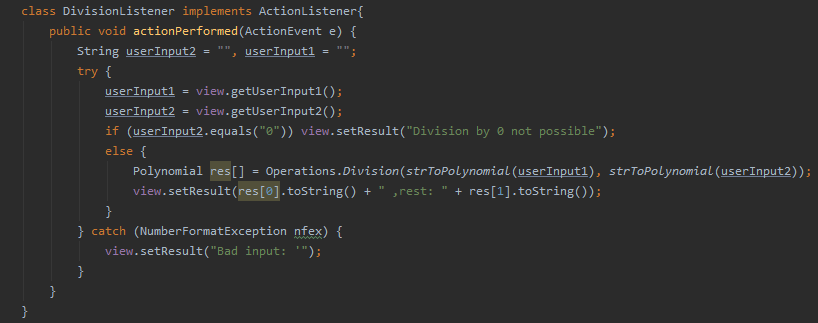
the resulting polynomial.

The most important functions of the controller are the Listeners,

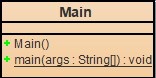
for every operation, each one having a separate class,

which implements the behavior for each of the buttons. For each listener the behavior is similar, takes the input(s) from the view, calls the corresponding operation from the model, and the view is updated with the corresponding result (or error eventually).

Example: The DivisionListener Class



## **Class Main**



The Main class is for testing the application, where model, view and

controller are created and instantiated. It contains only a main() method,

which runs the application.

# Usage and Testing

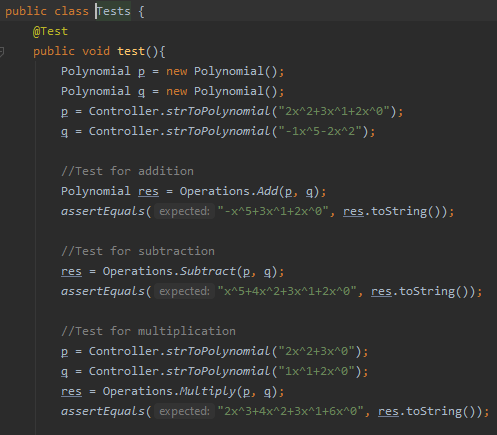
The user has to introduce the polynomials in the text boxes, and then to press to button corresponding to the desired operation.

The valid input format is: coefficient x ^ power

Examples of valid inputs:

* 2x^3+3x^2
* -5x^5+1x^0
* -2x^1-2x^0

The testing was done by using the JUnit framework, and using assertions, in particular the assertEquals() function.



# Conclusions

During the development of the assignment, some research was needed. By completing this assign, I can affirm that I am more familiar with some aspects regarding the development of an application.

I understood better the meaning of the MVC pattern and how the classes should be implemented and used. Also, by creating the view manually, I understood exactly how a GUI works, before I’ve done this only by using some simple drag and drop tools.

Also, I learned to work with JUnit, testing the application, Regular expression matching as well as sorting some classes written by me, using compareTo() and implementing the Comparable class.

Future developments:

* More work on the graphical interface
* Handling polynomials with float/ double coefficients
* Handling polynomials with negative coefficients
* Optimization of the algorithms (Most of the algorithms run on O(n^2))
* Making the input be more user-friendly (not forcing to introduce 1 coefficient, 1 and 0 as powers of x)
* Handling polynomials with more than one variable
* More operations (Power series, surface integral, etc.)

# Bibliography

* <https://www.mathsisfun.com/algebra/polynomials-division-long.html>
* <https://www.vogella.com/tutorials/JavaRegularExpressions/article.html>
* <https://junit.org/junit5/docs/current/user-guide/>
* <https://www.javatpoint.com/java-swing>
* <https://regex101.com/>
* <https://javapapers.com/oops/association-aggregation-composition-abstraction-generalization-realization-dependency/>
* <https://www.leepoint.net/GUI/structure/40mvc.html>