**DOCUMENTATION**

**ASSIGNMENT 1**

NAME: Muntean Cristian

GROUP: 30424

Contents

[1. Objective 3](#_Toc97722306)

[2. Analysis, modelling, scenarios and use cases 3](#_Toc97722307)

[3. Design 6](#_Toc97722308)

[4. Implementation 7](#_Toc97722309)

[5. Results 10](#_Toc97722310)

[6. Conclusions 11](#_Toc97722311)

[7. Bibliography 12](#_Toc97722312)

# Objective

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation (i.e., addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result.

# Analysis, modelling, scenarios and use cases

1. Analysis

A polynomial is an expression that can be built from constants called coefficients and symbols called variables. Each variable has a degree associated to it. A polynomial with a single variable can be written in the form:

Where are the coefficients and x is the variable. Furthermore, we can say that a polynomial consists of a list of monomials, where a monomial is in the form of , where a is the coefficient and x is the variable. So, the polynomial has three monomials: the first one with the degree two, the second one with the degree one and the third one with the degree 0. We can deduce the following representation of the polynomial as a sequence of monomials:

Where is the coefficient of the monomial and n is the degree of the monomial. This way of representing polynomials can be used to perform operations on polynomials such as: addition, subtraction, multiplication, division, derivation and integration.

1. Modelling

The user will be able to perform operations on polynomials by introducing in the interface two polynomials and choose the desired operation. This operation can be one of the following:

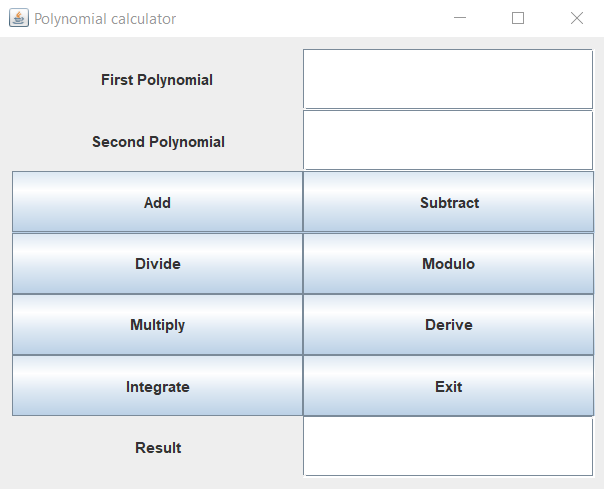
* Addition of two polynomials
* Subtraction of two polynomials
* Multiplication of two polynomials
* Division of two polynomials
* Modulo of two polynomials (shows the remainder resulting the division of two polynomials)
* Derivation of a polynomial
* Integration of a polynomial

The result of the chosen operation will be displayed on the interface, in the “Result” text field.

1. Scenarios and use cases

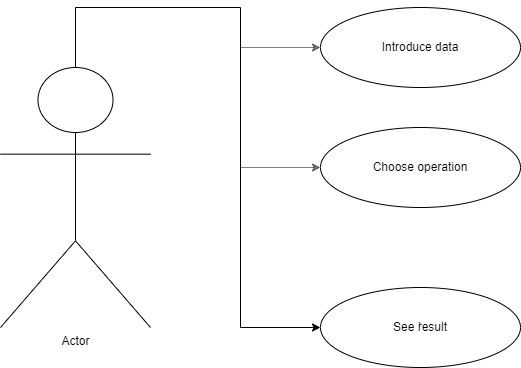
A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

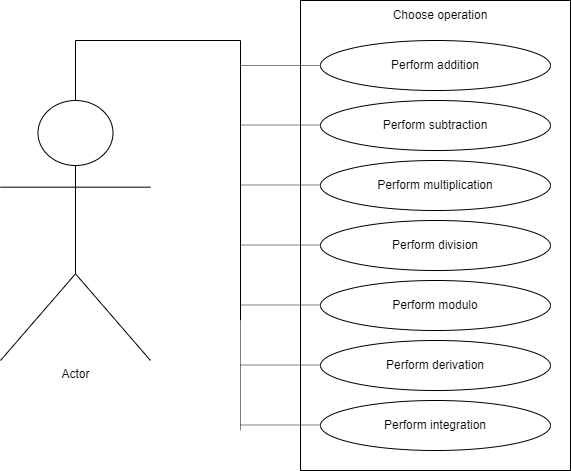
The use cases are strongly connected with the steps the user has to make, so I created the user interface with this in mind, resulting in a straight forward design.



The user will introduce two polynomials in the first two text fields, then he/she will have to click on the button corresponding the operation he/she wants to perform. The result will appear in the text field near the “Result” label. For some of the operations such as integration or derivation, the user has to add a single polynomial in the first polynomial text field.

Use case diagram:



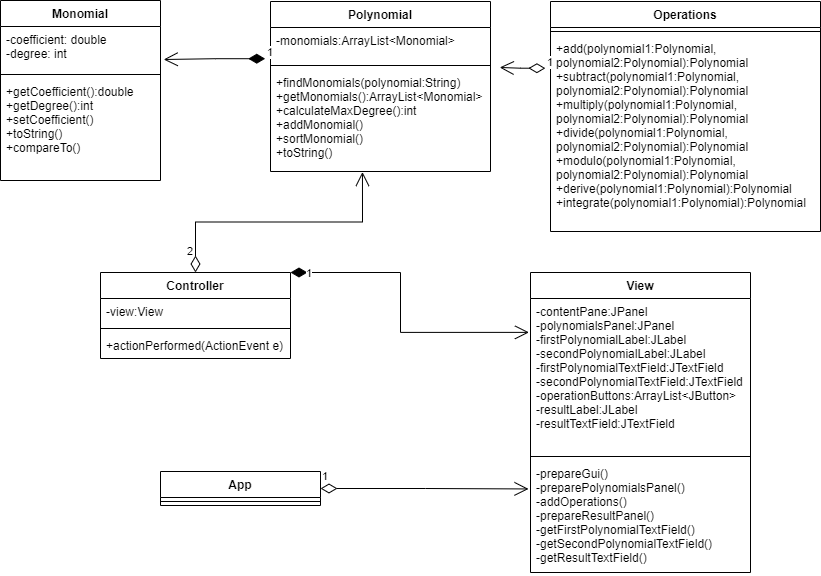


# Design

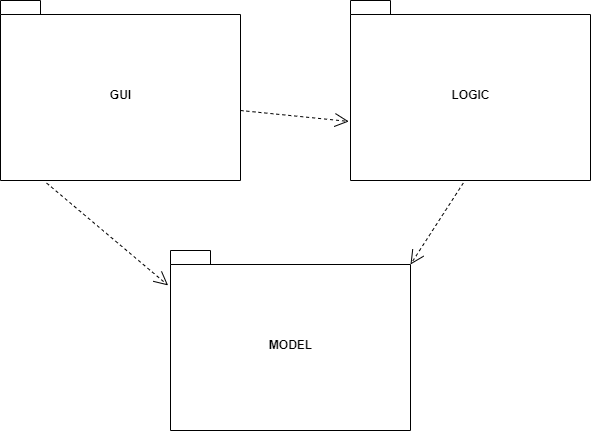
In the design of the application I the model-view-controller (MVC) architectural pattern, which separates the logic package (that contains the classes which implement the mathematical operations) from the model (that contains the classes which model the application data) and from the view (that contains the classes that are related to the graphical user interface). The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components to be used in the development process of the application:

* The *logic* package contains only the “Operations” class, which consists of static methods that have one or two parameters and return a Polynomial object.
* The *model* package contains the “Monomial” and “Polynomial” classes. The “Monomial” class has two fields: coefficient of type double and degree of type int. The “Polynomial“ class has an ArrayList of “Monomial“ objects as its only field.
* The *view* package contains the “View” and “Controller” classes. The “Controller” class is responsible of connecting the logic and model packages with the user interface. The “View” class implements the user interface.
* UML Diagram



* Package diagram



* Data structures

The data structures that I have used in the implementation of this project are either primitive types (such as int or double) or more complex structures such as ArrayList and the new classes “Polynomial” and “Monomial”.

I used the ArrayList over the classic Array because the size had to be dynamic, not knowing how many monomials a polynomial contains. In this way, I removed the possibility of exceeding the maximum declared size of a classic Array.

# Implementation

Because I used the MVC design architecture, my application has 4 main parts:

1. The Model part

This part contains the classes which model the application data. The model has 2 classes:

1. The “Monomial” class

Contains 2 fields: coefficient – double; degree – int

The most important methods of this class are:

* The Constructor. Simple contstructor that accepts two parameters(coefficient and degree) and sets the values of the created object to the two parameter values.
* Overriden “compareTo”. Used in sorting the list of monomials of a polynomial by their degree(descending).
* Overriden “toString”. Used when the result is converted from Polynomial type to String for printing the result in the user interface.

1. The “Polynomial” class

Contains a single field: ArrayList of Monomial objects.

The most important methods of this class are:

* The Constructors. There are 3 types of constructors in this class. The first type takes as a parameter a polynomial in the type of a String and converts it to a list of monomials using the “findMonomials” method. The second type of constructor is the default constructor and only creates an empty ArrayList of Monomials. The third type of constructor takes a Polynomial object as a parameter and adds the parameter’s monomials to its own list of monomials.
* The “findMonomials” method. Takes as a parameter a polynomial in the type of a String, finds and adds to the object’s monomial list all the monoials found in the parameter. It uses regular expressions to find all possible combinations that can be a monomial (.
* The “calculateMaxDegree” method finds the maximum degree of the polynomial and returns it.
* The “addMonomial” method takes as a parameter a Monomial object and adds it to the current Polynomial object’s Monomial list. If the Monomial list already has a Monomial with the parameter’s degree, their coefficients will be added. If not, a new Monomial object is added.
* The “sortMonomials” method sorts the Monomial list decreasing in relation with each Monomial’s degree.
* Overriden “toString” method is used to convert the Monomial list to a String in order to print it in the user interface.

1. The Logic part

This part contains only the “Operations” class. Every method in this class is static. By making the methods static, I removed the need of instantiating the class in order to use the methods.

The methods that were implemented in this class are;

* The “*add*” method. Takes as parameters two Polynomial objects and returns a result polynomial. We iterate through the first polynomial’s Monomial list and takes a decision of what the result monomial will be. The result polynomial adds the coefficients of the two monomials and adds to the result Monomial list a new Monomial with the sum as coefficient and the same degree if the same degree is found in both polynomials. If the degree is not found in both polynomials, a new Monomial is added with the given coefficient and the given degree.
* The “*subtract*” method. Takes as parameters two Polynomial objects and returns a result polynomial. The result polynomial subtracts the coefficients of the two monomials in the same way as the addition method.
* The “*multiplication*” method. Takes as parameters two Polynomial objects and returns a result polynomial. We multiply each pair of monomials from the two parameter polynomials and add a new Monomial object to the result polynomial’s Monomial list which has the coefficient equal to the multiplication of the pair’s coefficients and the degree equal to the sum of the pair’s coefficients.
* The “*division*” method. Takes as parameters two Polynomial objects and returns a result polynomial. The polynomial division algorithm is used that gives us the quotient and the remainder resulting from the division. The division method returns only the quotient.
* The “*modulo*” method. Takes as parameters two Polynomial objects and returns a result polynomial. Like in the polynomial division method, the same algorithm is used and the result Polynomial that is returned is the remainder of the division.
* The “*derivation*” method. Takes as parameter a single Polynomial object and returns a result polynomial. The result polynomial’s monomials have the coefficients equal to the coefficient of the parameter multiplied with the degree and the degree is equal with the parameter’s degree-1.
* The “*integration*” method. Takes as parameter a single Polynomial object and returns a result polynomial. The result polynomial’s monomials have the coefficients equal to the coefficient of the parameter divided by the degree+1 of the parameter and the degree is equal with the paramete’s degree+1.

1. The View part

Contains 2 classes: “Controller” and “View”.

1. The “Controller” class

Has as its only field a View object.

The main methods of this class are:

* The Constructor. Takes as a parameter a View object and assigns it to the current Controller object’s view field.
* The “actionPerformed” method. Takes as a parameter an ActionEvent object that corresponds to the command that has been passed from the user interface (by pressing a operation button). It takes the two polynomial Strings from the two TextFields’s texts and creates two Polynomial Objects. Depending on which command has been passed to the Controller, a operation method from the “Operation” class will be called and performed. After the Operation is complete the result will be put in the “resultTextField” in the user interface. Furthermore, the two input polynomial Strings will go through a verification step to assure that the given polynomial is valid or not.

1. The “View” class

Is the class that implements the graphical user interface. It’s fields are: contentPane, polynomialsPane of type JPanel; firstPolynomialLabel, secondPolynomialLablel, resultLabel of type JLabel; firstPolynomialTextField, secondPolynomialTextField and resultTextField of type JTextField and operationsButtons of type ArrayList<JButton>. It also contains a Controller object.

The main methos of this class are:

* The Constructor. Takes as parameter the title and calls the “prepareGui” method.
* The “prepareGui” method sets the dimension of the window of the user interface, sets the default close operation as the “EXIT\_ON\_CLOSE”, which means that when the “X” button is clicked, the application will stop completely. It creates the contentPane panel and gives it a padding and calls the “preparePolynomialsPanel” and “prepareResultsPanel” methods and sets as content pane the contentPane field.
* The “preparePolynomialsPanel” creates all the components that compose the input section of the user interface(first/secondPolynomialLabel, first/secondPolynomialTextField and all the operations buttons), calls the “addOperations” method and adds them to the polynomialsPanel field.
* The “addOperations” method creates all the buttons for the operations, adds them to the operationsButtons List, sets action commands and listeners on each one and adds each button to the polynomialsPanel.
* The “prepareResultsPanel” method creates all the objects that relate to the results part of the user interface(resultLabel and resultTextField), adds them to the polynomialPanel and adds the polynomialPanel to the contentPane.

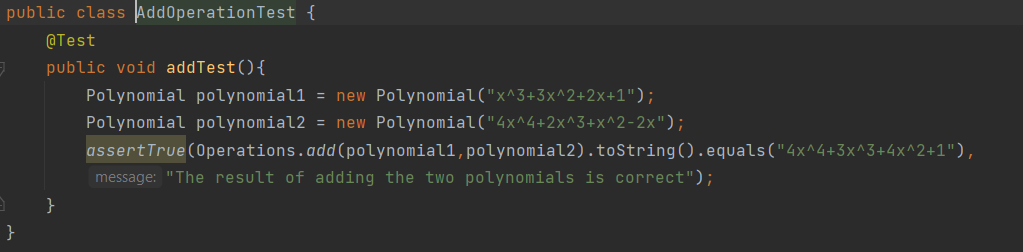
1. The App part

The App part contains only the “App” class which creates a new JFrame of type “View”, sets the frame visibility to true and sets the default close operation of the frame to exit the whole application on the click of the “X”

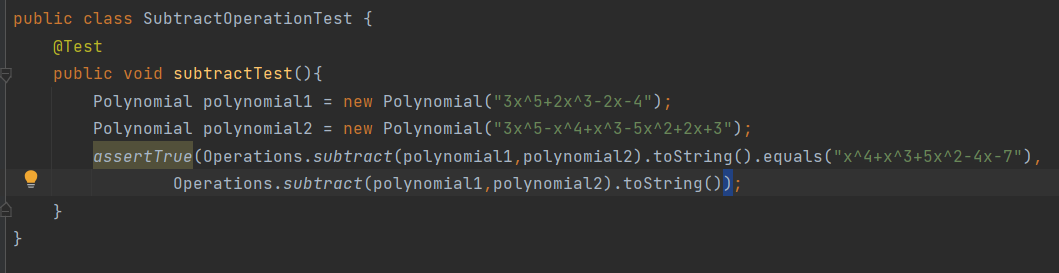
# Results

For the testing part of the application I used JUnit in order to create unit tests. These unit test are called by a suite test class. I created unit tests for all the operations in the following scenarios:

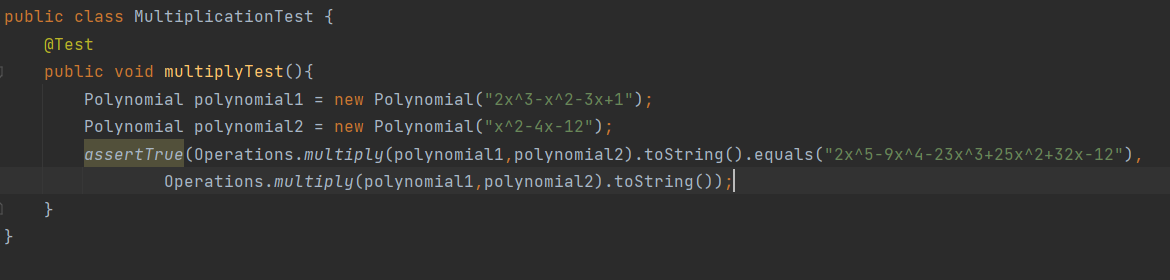
* *Addition*



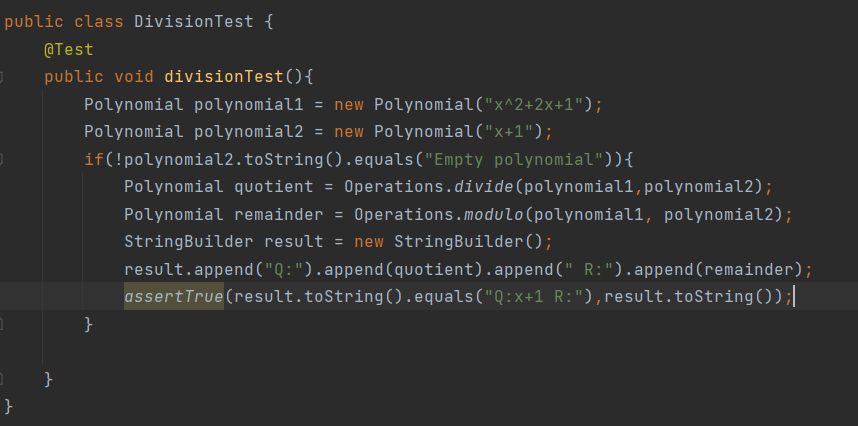
* *Subtraction*



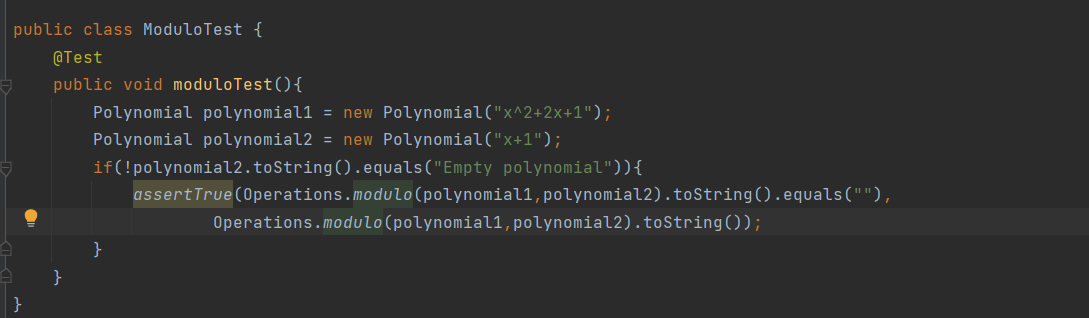
* *Multiplication*



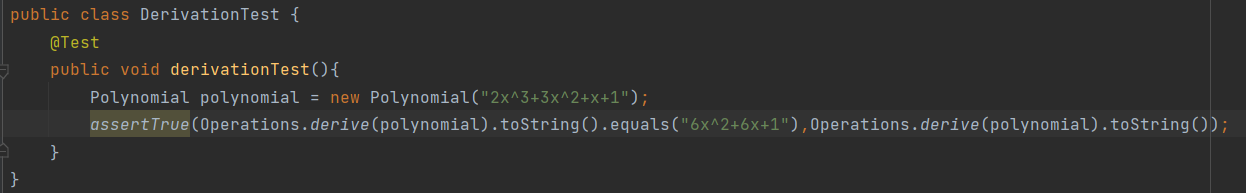
* *Division*



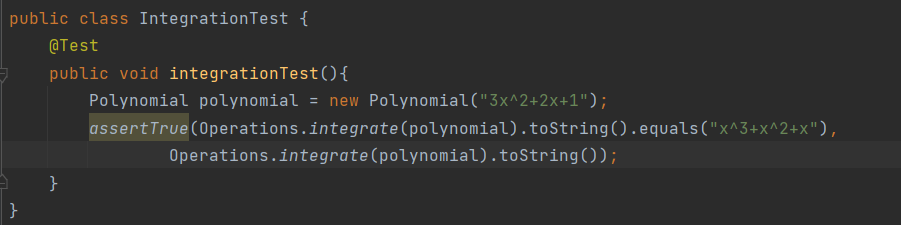
* *Modulo*



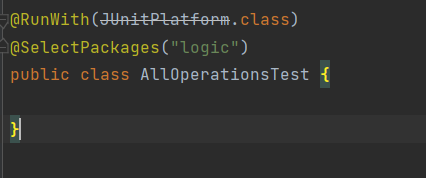
* *Derivation*



* *Integration*



These tests were placed in the package “logic” and the suite test class selects all the unit test classes from said package.



# Conclusions

This project was a good opportunity to solidify my OOP knowledge that I learned in the first semester. I also learned new things along the development of this project.

First of all, I became more familiar with the MVC pattern, which I have not used so much until now. Also, I learned that a good design architecture makes the implementation of the project much easier.

Secondly, I learned how to work with regular expressions in Java and how to create my own regular expression which helps me take the parts of a string I need and how to split the matchings of the regular expressions into groups in order to take more detailed information about that matching. For a concrete example, I used regular expressions to find each monomial in the input string and then get the coefficient and degree of said monomial by using groups.

# Bibliography

1. <https://draw.io/> - for diagrams
2. <https://stackoverflow.com/> - for random bugs I encountered during the development
3. <https://regex101.com/> - for testing the regular expressions and seeing the groups of the mathings
4. Programming Techniques in Java – Lectures of prof Cristina Bianca Pop