**Polynomial Calculator**

1. Objectives

1.1 Main Objective

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

1.2 Secondary Objectives

* Define a representation for a monomial. (Chapter 3)
* Define a representation for a polynomial based on monomials previously defined. (Chapter 3)
* Implement all the required operations on monomials. (Chapter 4)
* Implement all the required operations on polynomials. (Chapter 4)
* Define the entity of a calculator that is able to perform specific operations on. polynomials previously defined. (Chapter 3)
* Implement the test environment and test the calculator using various test cases. (Chapter 4)
* Implement the validation method for the input. (Chapter 4)
* Implement the conversion method: raw input 🡪 polynomial object. (Chapter 4)
* Create the view for the application. (Chapter 4)
* Implement the control unit for the application. (Chapter 4)

1. Task analysis, Modeling, Scenarios, Uses Cases

The application should allow the user to insert polynomials in a string format through the text fields provided by the graphical interface and then choose from the supported operations the one to be performed. In case the inputs are valid polynomials the interface will display the result of the selected operation in a specific field, otherwise, the user must be warned that the input does not obey the expected format and thus the application could not carry on the given command.

* Use Case: Compute
* Primary Actor: Client
* Main Success Scenario:
  + The calculator waits for the user input.
  + The user introduces polynomials
  + The calculator waits for the command of a specific operation.
  + The user selects the operation.
  + The calculator validates the input.
  + The calculator interprets the input and computes the result.
  + The calculator displays the result.
* Alternative Sequences:
  + The user input is not valid
    - The calculator displays an error message specifying the incorrect input.
    - The command is not executed.
    - The calculator returns in its initial state (step 1).

1. Design

In order to achieve our goal, we need to approach the task in a bottom up way. In other words, the first things we should define are the tools that the calculator needs to perform its job. These are, of course, the polynomials. Taking a closer look at what a polynomial consists of, it is very easy to see that monomials are the first things we should define. That being said, the first check point is to implement the Monomial class and based on it the Polynomial class.

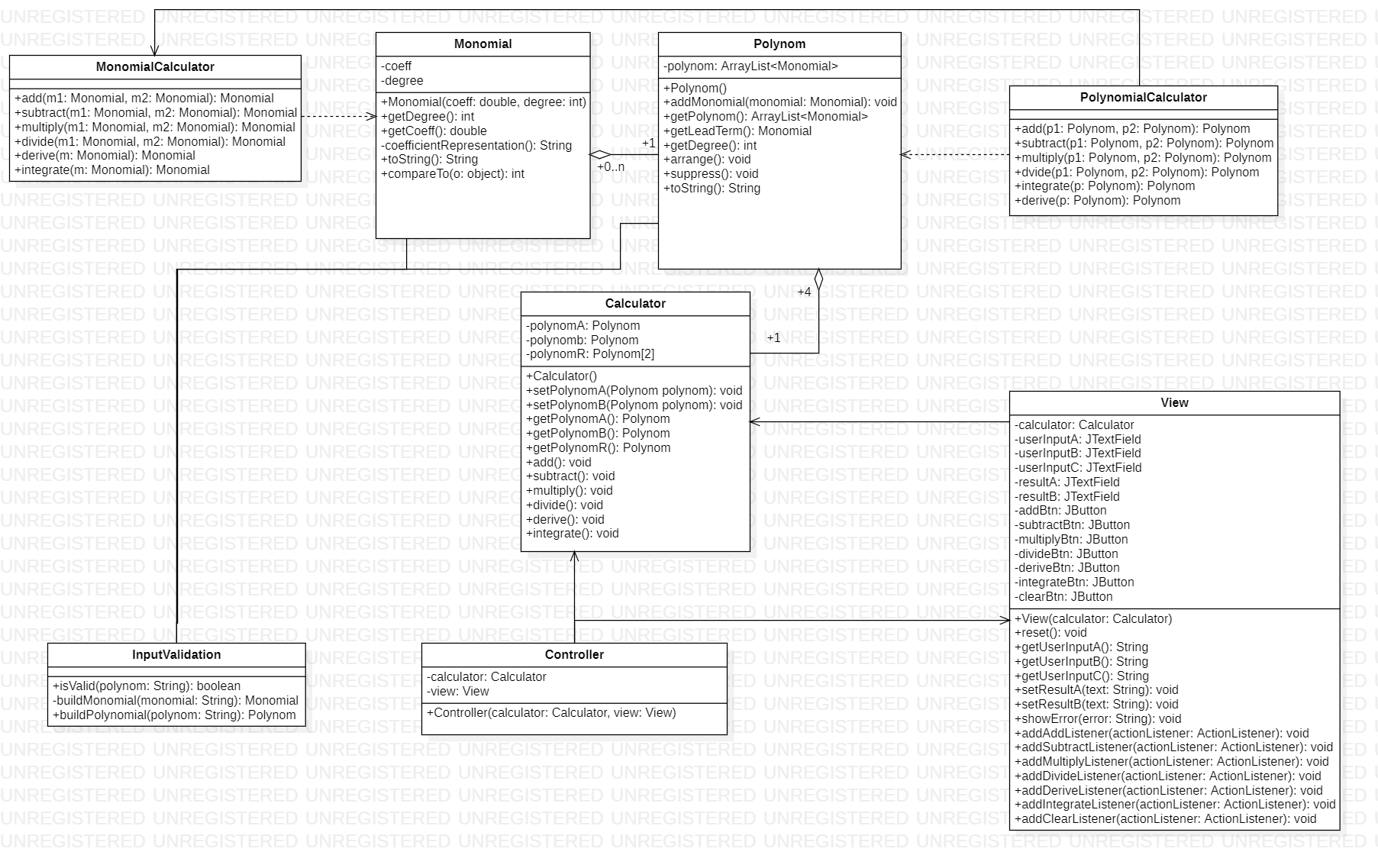
Having the polynomials to work with, we move our attention further to the functionalities of the calculator. It should be able to add, subtract, multiply, divide, derive and integrate polynomials but since each of these operations are done at a lower level on monomials, this is our starting point. The class MonomialCalculator is to be implemented to handle all the above-mentioned operations on monomials and then, based on it, the class PolynomialCalculator which implements the operations at a polynomial level.

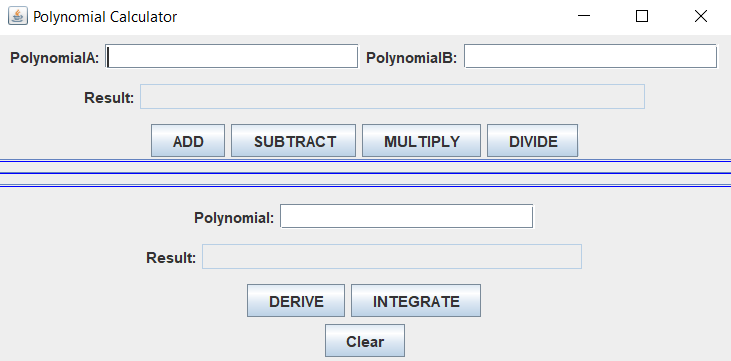
At this moment, we have created everything we need to bring the polynomial calculator to life. The class Calculator will be implemented to work with polynomial objects and use the PolynomailCalculator methods to perform the demanded operations.

At this point, we are done implementing the model of the application and move on to the next part of the design, the view. The view class should take care of all the graphical parts that are display on the screen and needed for the user to interact with the calculator. It mainly consists of text fields, buttons, labels and any other objects that could make the view more friendly.

Finally, a controller class is needed to make the link between the model and view and to deal with the user commands. It provides the functionalities for the buttons and directs the user input to the calculator in order for it to compute and return the result which is then displayed on the screen. Now the application seems complete, but there is one more thing the we must take care of. The user might not always input a valid polynomial and this will cause later on exceptions because the calculator supposes that the given polynomial is mathematically correct and obeys some rules in terms of format. In order to address this problem, the class InputValidation is created which offers the necessary method for verifying the validity of a polynomial as well as converting it into a Polynomial object.

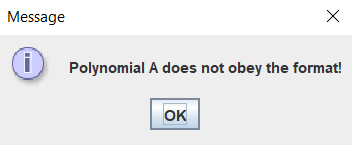
In order to make things clearer, the UML class diagram of the application is presented below.

 Next, the graphical user interface is presented:



At first, we observe that the panel is divided in two sections. This is because we separate the operations that take two inputs from the ones that take only one in order to remove ambiguity. The upper part is reserved for the two inputs operations: add, subtract, multiply, divide and is interpreted as PolynomialA (operator) PolynomialB. The lower part deals with the other two operations that take only one input (Polynomial). The Clear button is used to remove the text from the input fields and the results from their fields.

In case of an input that does not obey the form: Anx^x+An-1x^n-1+…+A1x+A where An can be any integer number and n natural number, a popup will appear specifying the error. An example where the PolynomialA input is not valid is shown below.



1. Implementation

* Monomial Class

It is characterized by its two private fields: coeff and degree which are initialized in the constructor. A specific method is coefficientRepresentation() which returns a string representation of the coefficient as a double decimal if it is a double number or as an int in case it can be casted as integer.

* Polynomial Class

The attribute of this class is an ArrayList of Monomial objects. The class provides methods for adding a monomial to the list and also to sort the list by the degree of monomials. It also has two methods for obtaining the lead term of the polynomial (getLeadTerm) and the degree of the polynomial (getDegree). One more method that is very usefull is the suppress() method which, in case that more monomials with the same degree exist, adds all of them into a single monomial object.

* MonomialCalculator Class

Consists only of public static methods that take as parameters Monomial objects and return a Monomial object as well. The class provides methods for the following operations: add, subtract, multiply, divide, derive and integrate. It is meant for making the polynomial operations easier.

* PolynomialCalculator Class

Consists only of public static methods that take as parameters Polynomial objects and return a Polynomial object as well. The class provides methods for the following operations: add, subtract, multiply, divide, derive and integrate. This is the class that provides the needed functionality of the polynomial calculator.

* Calculator Class

This is the class that implements the polynomial calculator. It has three attributes: polynomA, polynomB and polynomR. The polynomR attribute is a vector that holds at position 0 the result of every operation except for the division where the result is interpreted as follows: polynomR[0] 🡪 quotient and polynomR[1] 🡪 remainder. The class provides the needed mathods for getting and setting all iti attributes. Nonetheless, each operation: add, subtract, multiply, divide, derive and integrate is represented by a specific method that modifies the polynomR attribute accordingly.

* InputValidation Class

The role of this class is to provide the necessary methods for validating and converting the user input into specific class objects which are then passed to the calculator for computational work. The first method is isValid() which takes as parameter the string given by the user. It checks whether or not the string can be converted in an object of Polynom Class and returns a Boolean value. The second and third methods are meant to work together. More exactly, the buildPolynomial() method uses the buildMonomial() method in order to obtain the Monomial class objects which are added to the Polynom class object. They both take as parameter a string and return a Monomial object (buildMonomial()) or a Polynom object(build Polynomial()). All of this methods use regex expressions in order to achieve their goal.

* Controller and View Classes

These two classes are responsible with the graphical user interface. The View class creates the panel and its components such as JTextField and JButton objects. In order for the controller to be able to “see” what the user is doing and what is he asking, the View class has to implement a set of methods that return the state of its objects. These are the methods getUserInputA(), getUserInputB() and getUserInputC() for reading the user input and for writing out the results two more methods are implemented setResultA() and setResultB(). Another important method is showError() which displays the given string in a pop up window such that the user is informed of the error prompted by the controller. The rest of the methods implemented in the View class are dealing with adding the action listeners specific to each button. The Controller class, on the other hand, is handling all everything related to the logical functionality of the interface. Here is where all the listeners are created, defined and added to the buttons from the View class. The Controller class establishes the connection between the model and the view, interpreting the user’s commands and transposing them on the model such that the output is produced and displayed for the user to see it.

1. Results

The testing of the application is done using the Junit framework. This implies creating the testing class for the class that is to be tested as well as a test runner class. Our focus is on the calculator class which is the “brain” of the application. The application is all about yielding the correct result for the user’s input. We must verify that all the operations provided by the Calculator class: add, subtract, multiply, divide, derive and integrate work correctly for any possible valid input. The CalculatorTest class has one attribute which is a Calculator object based on which the test is performed. The class has 6 methods each corresponding to an operation and the constructor in which the calculator attributes are setted.

Some of the scenarios for which the calculator was tested are:

* The polynomials have degree 0.
* First polynomial has a higher degree than the second.
* Second polynomial has a higher degree than the first.
* One of the polynomials has degree 0 and the other greater than 0.
* The polynomials have negative coefficients.

1. Conclusions

Considering all the requirements presented in the first chapter, implementing this application provides a good way to exercise the main object oriented programming techniques. Moreover, a few new concepts were introduced such as: Model View Controller design and Junit testing which are good to be in a programmer requisite. To sum up all the things that I have learned during the implementations of this application, I’ll put them in a list below:

* Understanding the Model View Controller Design.
* Experience working with the Junit test framework.
* Improved knowledge of basic OOP concepts.

Considering the further improvements that can be done, we could think of enlarging the working field of the calculator. Whit this, I want to refer to the coefficients of the polynomials which could be real numbers not only integers. Same goes for the degrees of the monomials which could be negative thus making them integers. All the required operations can work very well considering these changes in the monomial class definition. Furthermore, a new operation could be implemented to increase the utility of the application. The operation I am referring to is the process of finding the roots of a polynomial. Limiting the area of work for this operation such that is works only polynomials with degree one, two, three or four makes the implementation quite accessible. Going a few steps further makes things way too complicated because polynomials with higher degree than 4 require special mathematics techniques in order to find all the roots.

1. Bibliography

* <https://www.vogella.com/tutorials/JUnit/article.html>
* <https://mkyong.com/tutorials/junit-tutorials/>
* <https://www.tutorialspoint.com/swing/index.htm>
* <https://docs.oracle.com/javase/8/docs/api/index.html?java/util/regex/package-summary.html>
* <https://www.tutorialspoint.com/java/java_regular_expressions.htm>
* In lab ppt presentation: PTA1-SUPPORT-PRESENTATION