**POLYNOMIAL CALCULATOR**

**Neag Dragoș-Ion**

1. **Assignment objective**

The main objective for this assignment was implementing a polynomial calculator, that can perform the following operations: addition, subtraction, multiplication, division, derivative and integration, and then display the result.

In order to build such a calculator, I’ve divided the process into a series of steps.

* Choose an appropriate way of representing the polynomial, by dividing it into multiple monomials
* Implement the methods used for performing the necessary operations
* Design an application that resembles an architectural pattern
* Build a graphical user interface, that can be easily manipulated by anybody
* Implement a polynomial validator, that extracts the text provided by the user and, after checking if it matches the demanded pattern, it converts it into certain objects
* Finally, create tests in order to check the functionality of the operations applied on different polynomials.

1. **Problem analysis**
2. Modeling

While building the calculator, a series of requirements had to be met. These classify into two categories:

* Functional requirements
* Non-Functional requirements
* Functional requirements
* The calculator should allow users to insert polynomials
* The calculator should allow users to select the mathematical operation
* The calculator should be able to add two polynomials
* The calculator should be able to subtract two polynomials
* The calculator should be able to multiply two polynomials
* The calculator should be able to divide two polynomials
* The calculator should be able to derive a polynomial
* The calculator should be able to integrate a polynomial
* The calculator should be able to notify the user when the insert polynomial doesn’t have the required format
* Non-Functional requirements
* The calculator should be intuitive and easy to use by the user
* The calculator should have a minimalist design, not to overcomplicate things
* The calculator should indicate the user what format to use when typing it in

1. Scenarios

The possible scenarios can be represented as use-cases.

Take for example the subtraction of two polynomials:

**Use case:** subtract polynomials

**Primary Actor:** user

**Main Success Scenario:**

Step1. The user inserts two polynomials in the GUI

Step2. The user clicks the “SUBTRACT” button

Step3. The polynomial calculator displays in the result field the result of the subtraction

**Alternative Sequence:** Incorrect polynomials

* The user inserts polynomials that don’t match the required format
* The scenario returns to step 1

Though, the case presented above can be generalized for each operation performed by the calculator.

I’ve represented all the use-cases in the diagram below.

Diagram

Description automatically generated

1. **Design**

The application is pretty straight forward in terms of overall design. It expects the user to provide polynomials and select the operation he wants it to perform by pressing the respective button. Lastly, the result is displayed.

* Packages

In order to create a good work environment, I decided to make use of the architectural patterns, specifically the Model View Controller. The usage of this patterns brings various advantages to coding, as it allows the programmer to outline the design based on his ideas more clearly.

The model is responsible with encapsulating the application state and responding to state queries.

The controller is the actuator, that defines the actual behaviour of the calculator and is in charge with handling the response.

The view is basically the graphical part of the application, as it renders the models and requests updates from models.

As I said, my code is divided into four packages: main, model, view and controller, each one containing different classes related to their name, which I’ll present below.

* Classes

Diagram

Description automatically generated All the classes can be seen in the UML diagram:

* Main package

The main package contains one class: Main. This class is responsible with initializing the swing interface.

* Model package

In this package I’ve stored two classes, which are: Polynomial, Monomial. These classes encapsulate the actual objects of the problem.

* View package

Inside this package, we can find the class Dashboard, the user interface class, which holds the pieces of code related to the graphical interface. Besides it I’ve also created an abstract class named AppFrame, which inherits the JFrame class, and an exception that gets thrown when the polynomials typed in by the user do not respect the required format.

* Controller package

Lastly, in the controller package there are two classes implemented: Conversion and Operation. These two are responsible for the decoding of the text provided by the user, as well as for the necessary calculus.

1. **Implementation**

* Main

The Main class is only responsible for initializing the dashboard (the GUI).

* Monomial

The public class Monomial stores two attributes: coefficient and power. This class individually stores the characteristics of each monomial of a polynomial.

Besides the getters and setters, inside this class there are two more methods: deriveMonomial and integrateMonomial. As the name suggests, because the derivative and the integration of a polynomial are calculated in such a way that the monomials are independent of one another, I’ve decided to calculate and return the value of the derivative/integration for each monomial.

* Polynomial

The polynomial class stores a single attribute: a list of monomials. As when it comes to methods, there are some extremely important ones in this class.

* orderPowers – it sorts the monomials in descending order of the powers, in case the user provided a list of monomials that were randomly sorted
* compare – is an overridden method from the Comparator class, that is used for sorting the monomials by their power
* getThisPolynomial – is the method used to display a certain polynomial. The method acts similarly to a getter, just that instead of returning a list of monomials, it returns a string that can be displayed into the GUI. As a parameter it has the variable “withDecimals” which is used in order to know whether the function should display the coefficients as integers or as floats (for the division/integration)
* coefficientToDisplay – this method is used by the previous one, that explaining why it is private. Its role is judging whether a coefficient should or should not be displayed, as there are multiple cases. For example, if the coefficient is 1 or -1, then the value should not be displayed, but in the case of -1, the sign must be stored. That changes when the power of x is 0, so now even 1 or -1 must be displayed.
* polynomialDegree – the method is used for determining the degree of the polynomial and is helpful in situations where it matters, such as in the division operation, where the algorithm stops when the first polynomial has the degree lower than the second one.
* Conversion

The conversion class is responsible for validating the format of the text that was inserted by user, as well as for the decoding of that respective text (converting the string into a list of monomials).

The attributes of the class are represented by two string inputs, firstPolynomial and secondPolynomial, and the two polynomial outputs, polynomial1 and polynomial2.

* checkFormat – this method uses pattern matching and regular expression in order to validate the string inserted by the user. If the text matches the pattern, the method returns “true”, and if it doesn’t, the method returns “false”.
* decodify – the method is described by its name. It is used to convert a string into a polynomial. It begins by separating the text such that we only have monomials left, which are stored into a list of strings. For each polynomial, the method iterates through the list of strings and at each step determines the coefficient and the power of the monomial and initializes a new one, that is added to the list, list that will be nested inside the returned polynomial
* initializeMonomial – method that gets as parameters a coefficient and a power, and sets them into a new monomial, that will be returned
* calculateCoefficient – this method extracts the coefficient from the string provided as parameter. First, it iterates until the appearance of the character ‘x’, such that we know that until that point is the coefficient we are looking for, and it stores each character step by step into a new string. Next, it checks if the newly constructed string is empty, and if so, it returns 1. If the string is not empty, then it is necessary to check if the first character is either + or -, and if not, then the whole string is converted and returned. Otherwise, if the first character is + or -, we remove that first character from the string, and convert it into numerical velue. Lastly, if the first character proved to be a -, then the numerical value is also multiplied by -1, and is returned by the function.
* calculatePower – this method continues from where the last one left off. It is also provided with the same monomial string as the parameter. First thing that it does is to iterate to the first appearance of ‘^’, and store each character up to that point. If the string does not posses a ‘^’ character, then the algorithm checks whether the iterator reached the end of the array. If it did, then the next test is made to determine if there is a ‘x’ character in the string, which would mean the monomial is of power 1, and otherwise, of power 0. However, if the ‘^’ was met, then it means that from that point on to the end of the string, is the value of the power, which is converted into numerical value and then returned.
* Operation

The operation class is the class that holds the application together, as it is the binding between the models and the GUI.

The attributes of the class are four polynomials, polynomial1, polynomial2, result and remainderOfDivision.

The constructor of the class is creating the connection between the class Operation and Conversion, as it has two string parameters, which are then converted into polynomial1 and polynomial2.

* polynomialSumSubtraction – this method can solve addition, as well as subtraction of two polynomials. First it determines the maximum power between the two polynomial, in order to iterate through all the powers available. It then calculates the coefficient of the result at the power the iterator reached, and if the coefficient is not 0, then it creates a new monomial with the respective coefficient and power
* searchPower – this private method is used by the method above to calculate the coefficient at a specific power. The method iterates through each of the two polynomial and if the power is found, it adds the coefficient of that monomial to the variable “coeff”. The parameter “isSum” is used while iterating through the second polynomial, if the searched power is found, there will be two cases: if isSum = 1, then it will add that coefficient to the variable coeff, and if isSum = 0, then it will subtract that coefficient from the variable coeff. At the end, variable coeff is returned.
* polynomialGeneralMultiplication – this method can solve multiplication. It iterates through the first polynomial, and inside that iteration it also iterates through the second polynomial. What it basically does, it multiplies each monomial of the first polynomial with every monomial of the second polynomial, and the result of each such multiplication is added to the final result
* polynomialGeneralDivision – the method soves division based on the algorithm described in the laboratory. First it orders the two polynomials by their rank. Next, it divides the first monomial of the first polynomial with the first monomial of the second one, in order to obtain the quotient. The second polynomial gets multiplied by the obtained quotient and the result is subtracted from the first polynomial. The steps are repeated until the degree of the remainder is lower than the degree of the second polynomial.
* polynomialGeneralDerivative – the method calculates the derivative of the polynomial given as parameter. The function iterates through the polynomial and calculates the derivative of each monomial and if the power of the result is greater or equal than 0, then the monomial is added to the result
* polynomialGeneralIntegration – the method is similar to the derivative one, the only difference being that after it calculates the integration of each monomial, it checks if the coefficient is 0. At the end, if the polynomial contained only one monomial of power 0, it adds 0 to the result.

**User interface**

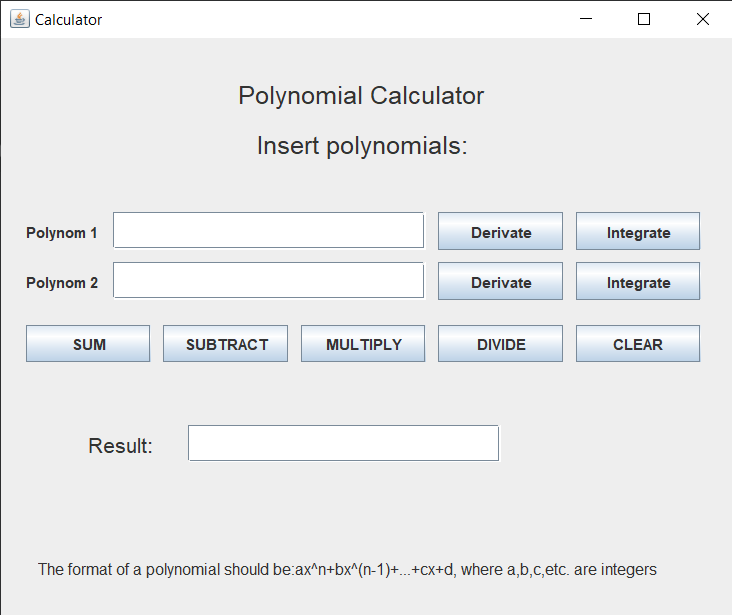
* Dashboard

The class Dashboard is the class that implements the graphical user interface using Java Swing.

The design I went for is presented in the picture below:

The attributes for the Dashboard class consists of three JTextFields: polynom1Field, polynom2Field and resultField, and nine JButtons: derivate1Button, integrate1Button, derivate2Button, integrate1Button, sumButton, subtractionButton, multiplicationButton, divisionButton and clearButton.

* initialize – this method creates the JFrame and sets its size and initializes the panel
* initializeCalculator – the method that stays behind the design of the calculator. It uses two text fields for the polynomials inserted by the user, and one text field to display the result. Below the text fields there are five buttons, each describing one operation. I’ve also added the button ‘CLEAR’, that resets the calculator. At the right side of the two text fields there are other four buttons, that act individually for each text field.

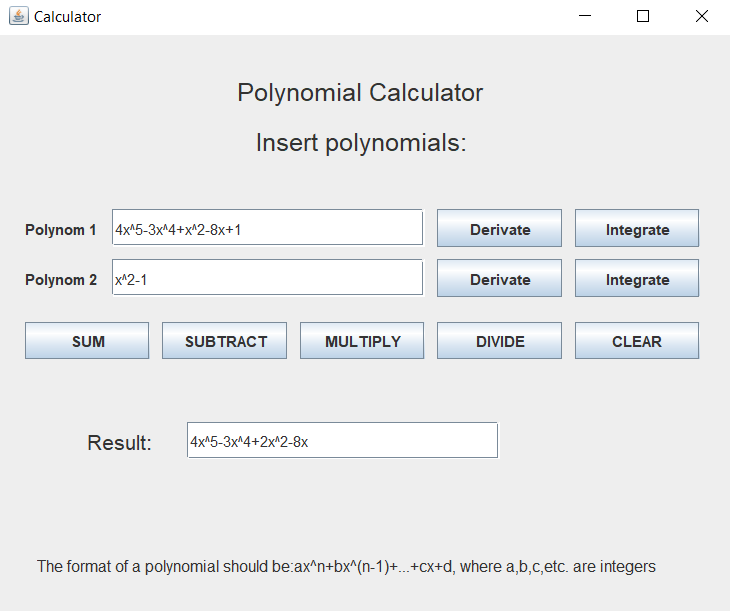


* initializeCalculatorListeners – this method is responsible for the actions that occur when a button is pressed. The process works similarly for each button. First, we try initializing the operation with the text inserted by the user and display an error message if it doesn’t match the pattern. Next, the operation is executed. The final tests are to check whether the user left a text field empty where he shouldn’t, and if so, again, an error message is displayed.

1. **Results**

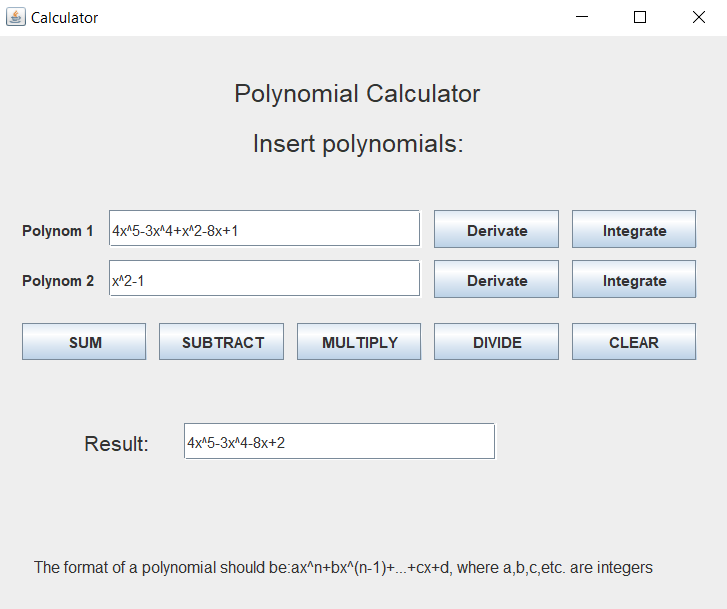
In order to prove the correctness of the calculator, a series of tests were done, using Junit.

The polynomials used for testing were: 4x^5-3x^4+x^2-8x+1 and x^2-1.

1. Addition

Expected result: 4x^5-3x^4+2x^2-8x

Actual result: 4x^5-3x^4+2x^2-8x

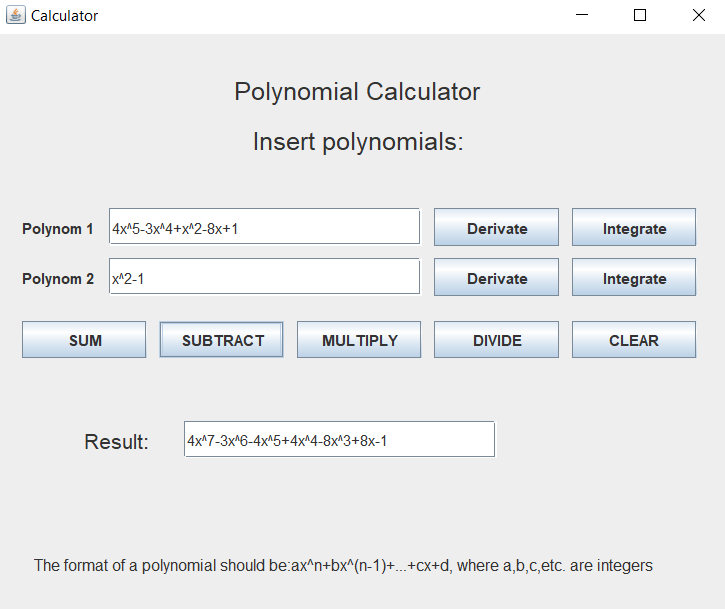
1. Subtraction

Expected result: 4x^5-3x^4-8x+2

Actual result: 4x^5-3x^4-8x+2

1. Multiplication

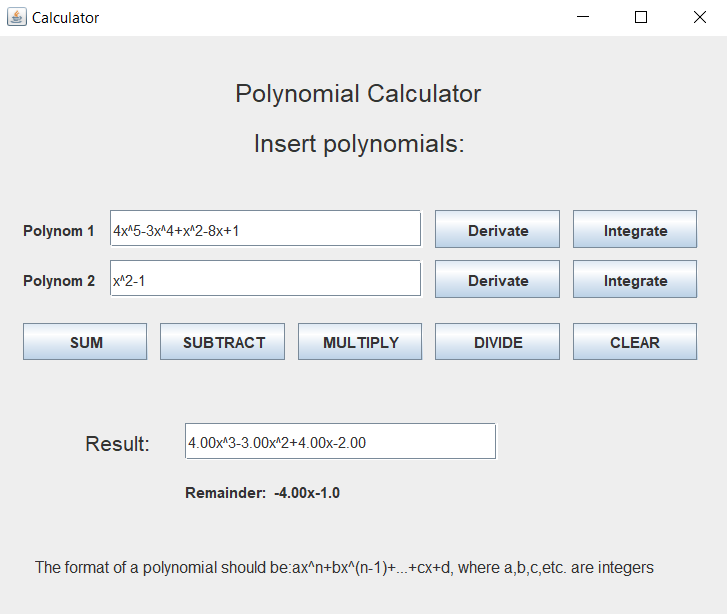
Expected result: 4x^7-3x^6-4x^5+4x^4-8x^3+8x-1

 Actual result: 4x^7-3x^6-4x^5+4x^4-8x^3+8x-1

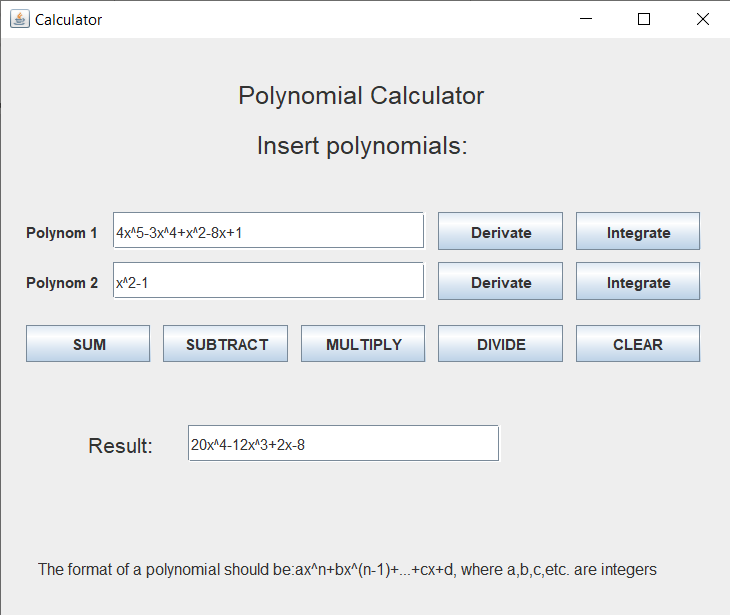
1. Division

Expected result: 4x^3-3x^2+4x-2

Actual result: 4x^3-3x^2+4x-2

1. Derivative

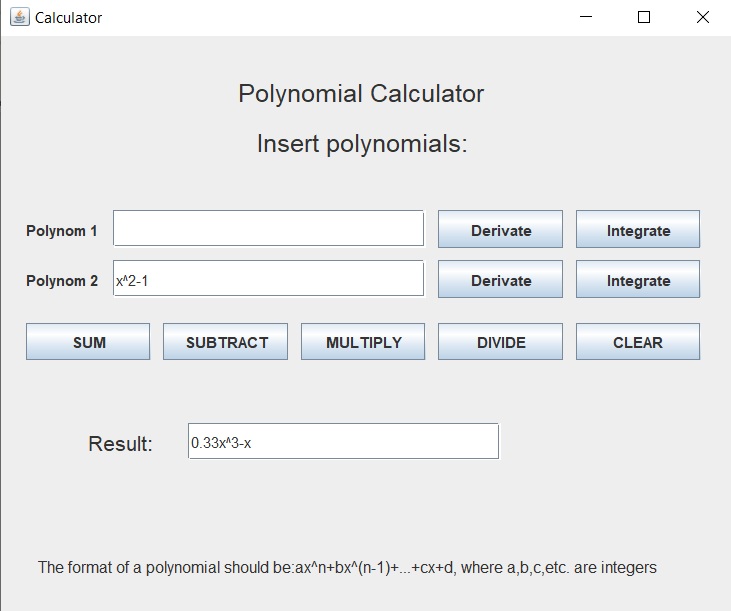
Expected result: 20x^4-12x^3+2x-8

 Actual result: 20x^4-12x^3+2x-8

1. Integration

Expected result: 0.33x^3-x

Actual result: 0.33x^3-x

1. **Conclusions**

This homework truly gave me a better understanding of java application designing. Although I was brought out of my comfort zone, having to adapt to a new style of coding, I think it was not done for nothing, as I’m sure it will help me out soon.

As for additional development, the sky is the limit when it comes to calculators, whether we’re talking about design or functionality. Perhaps some additional operations could be implemented, or maybe combining the ones we have now.

1. **Bibliography**

<https://online.visual-paradigm.com>

<https://stackoverflow.com>

https://google.github.io