Technical University of Cluj-Napoca

Polynomial Calculator

-project documentation-

Pop Andrei

Group 30424

1. Objective

We all know how difficult it is to work with large polynomials, so main objective of the calculator is to ease dealing with polynomial operations like addition, subtraction, multiplication, division, integration and differentiation. Each of these operations can be performed in a very short period of time (instantly) after pressing a button representing the operation that we want to perform. Because the calculator will be used generally by students, which some of them are not specialized in the math field, another objective was to create a user friendly interface that would facilitate the use of the calculator.

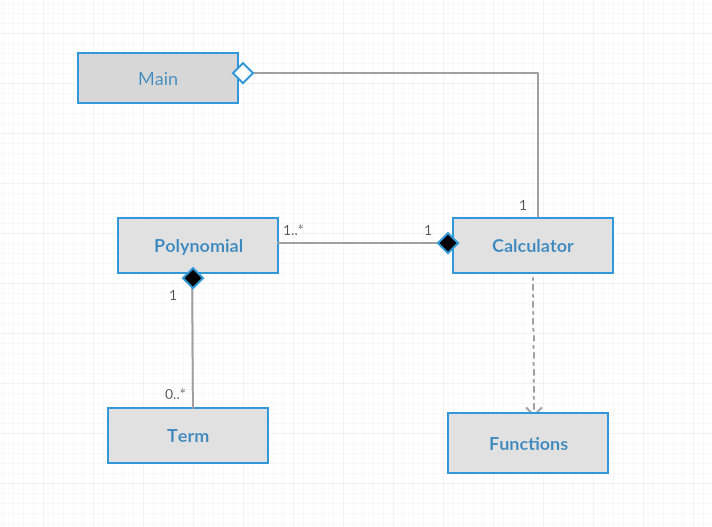
1. Problem analysis, modeling, scenarios, use cases

The problem requires to create a polynomial processing system. The polynomials have integer coefficients and they are one variable polynomials. In order to work with polynomials, before anything else we have to think about structures to store the coefficients which are directly related to the degree. In order to do this, a specific class was created. The next problem when dealing with polynomials is storing all the coefficients – degree pairs which completely characterize the polynomial. Also, in order to ease the operations later, they need to be stored in a predefined order, preferably ascending. In order to this we need another special structure to hold all the pairs. After dealing with all the storing requirements, we move further, to the tasks that need to be performed. Even if they aren’t fully specified, the most used operations on polynomials were implemented. These operations are addition, multiplication, subtraction, division, integration and differentiation. Each of them needs to be implemented separately by specialized algorithms. Sometimes an operation will require computations that need to be performed by other operations. For example when we divide two polynomials we’ll use multiplication and subtractions as well along the process. These operations need to be performed in “reasonable” time so we need to choose/create efficient algorithms that would work on the structures defined. Performing the operations in good time and having all the storing structures defined would not solve our problem, because all the effort of creating a good program would be useless if it can’t be used by a user who doesn’t know all the design details. In order to make the usage of the calculator much easy, the application needs to perform input processing in order to get the needed data for creating the polynomial. The application will be used only for the operations specified so the using scenarios are clear. For addition, subtraction, multiplication and division we’ll always need two polynomials in order to obtain a correct result, because all of them are binary operations. For integration and differentiation only one polynomial is used because they’re unary operations. In order to test properties like correctness and robustness of the app, various test cases will be used, each of them meant to test some features and properties of the application.

The use cases are sequences of actions that can take many forms and which can be performed by the system when interacting with the users. In our case the computer can perform the following operations: addition, subtraction, multiplication, division, integration and differentiation. The user is a primary actor who can enter two polynomials and then select an operation that will be performed on the polynomial(s) by the calculator. In case of differentiation and integration, only the first polynomial will be used.

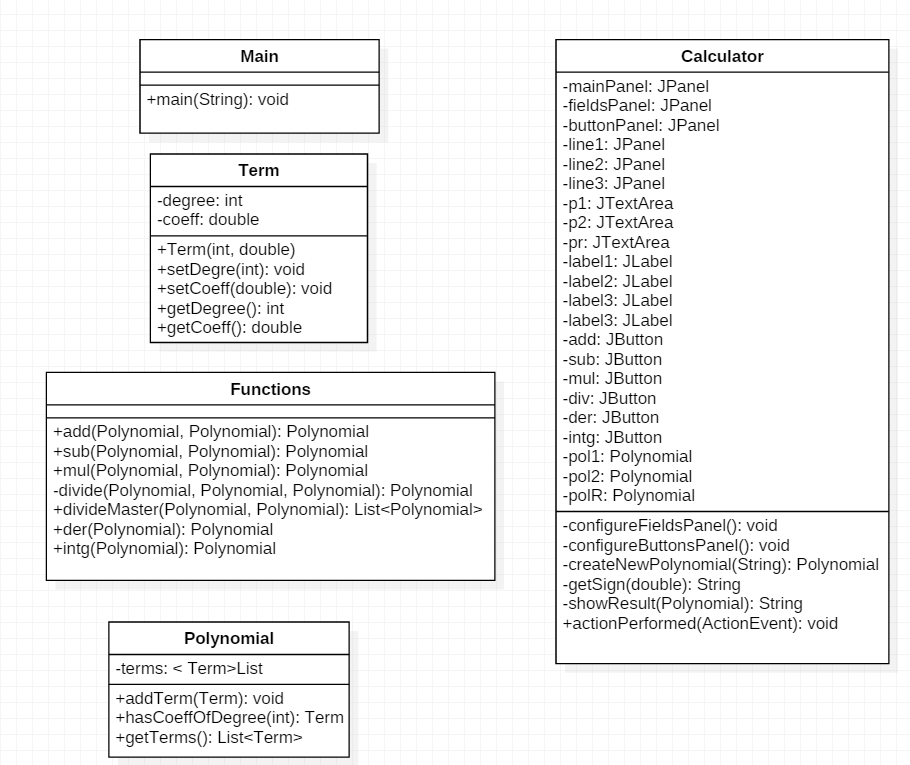
1. Design

3.1 UML diagram



The above diagram shows the classes and the relationships between them:

* between Term and Polynomial -> composition
* between Polynomial and Calculator -> composition
* between Calculator and Functions -> dependency
* between Calculator and Main -> aggregation

3.2 Class Design

3.3 Class Implementation

* Main Class
* Contains a new instance of the Calculator Class which represents the controller and the view of the application.
* Calculator Class
* Represents the controller and the view part of the application and we can call it “the brain” of the calculator. It takes the user input and the user commands, executes the selected operation and then displays the result of the operation.
* It is composed by the following methods and constructors:
  + - configureFieldsPanel(): this method will configure the panel containing the fields in which the user will provide the input data
    - configureButtonsPanel(): this method will configure the panel containing the specific buttons for each operation.
    - createNewPolynomial(String): this method will take a String parameter which represents the polynomial. It will process the string in such a way to get the coefficient-degree pairs and store them into a list. At the end, the method will return an instance of a new polynomial.
    - getSign(double): this method will return a string containing the sign of the parameter passed to the method.
    - showResult(Polynomial): this method takes polynomial as parameter and displays the characteristic string of the polynomial into the result field.
    - actionPerformed(ActionEvent): this is the action listener which decides what operation is to be performed, according to the user’s input.
* Functions Class

- Contains all the methods that are specific for each operation.

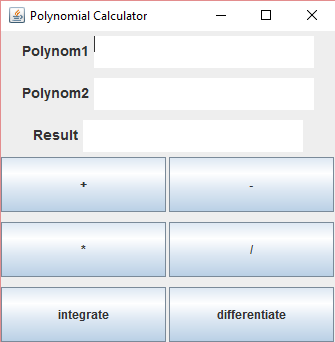
- The methods are:

* + - add(Polynomial, Polynomial): this methods takes 2 polynomials as parameters and returns a polynomial representing the result of addition.
    - sub(Polynomial, Polynomial): this method takes 2 polynomials as parameters and returns a polynomial representing the result of subtraction
    - mul(Polynomial, Polynomial): this method takes 2 polynomials as parameters and returns a polynomial representing the result of multiplication.
    - divideMaster(Polynomial, Polynomial): this method takes 2 polynomials as parameters and returns a list of 2 polynomials representing the result and the remainder of the division.
    - der(Polynomial): this method takes one polynomial as parameter and returns a new polynomial representing the result of differentiation of the first polynomial.
    - Intg(Polynomial): this method takes one polynomial as parameter and returns a new polynomial representing the result of integration of the initial polynomial.
* Polynomial Class
* This class contains all the pairs coefficient-degree specific to a certain polynomial. These pairs are stored into an array list.
* The methods contained by this class are:
* addTerm(Term): this method adds a new term into the list and then sorts the list again in order to keep all the element organized.
* hasCoefficientOfDegree(int): this method returns the term from the list which has the same degree as the given parameter.
* getTerms(): this method will return a list of terms that are specific for that polynomial.
* Term Class
* This class stores the pairs coefficient-degree so it has two attributes: coefficient and degree which can be accessed or modified through getters and setters.
* The class contains the following methods:
* setDegree(int): sets the degree of a term according to the parameter
* setCoeff(double): sets the coefficient of a term
* getDegree(): returns the degree of a term
* getCoeff(): returns the coefficient of a term

3.4 Data structures

The basic data structure that was used is implemented by the class Term and it stores pairs of the form coefficient – degree. Each of these pairs need to be stored in other structures. The appropriate structures used to keep all the pairs are lists. Each of the lists will store the corresponding data for polynomials and, in order to ease the computations, they will be kept ordered.

3.5 Graphical user interface

The graphical user interface is very simple. It was designed this way such that a non-specialized user can use the calculator without having to read the documentation about how to use it. It contains 3 text fields. The first two text fields: Polynom1 and Polynom2 are filled by the user. The text typed in the first two fields represent the polynomials and they should be provided before each operation. The 3rd text field shows the result of the operation. Under the fields some buttons are displayed. Each of the buttons represent an operation. When they’re pressed, an operation will be performed and the result of the operation will be showed in the 3rd text field called “Result”’.

1. Testing and implementation

I have chosen some representative tests in order to prove that the program runs correctly. Each of the methods implemented in the Functions Class will be tested because these methods are the ones that generate the result.

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Aim | Description | Result |
| 1 | Check if the addition method works properly | Two polynomials will be given into the text field and then the ‘+’ button will be pressed in order to generate a result. | The result was the expected one so the method works properly. |
| 2 | Check if the subtraction method works properly | Two polynomials will be given into the text field and then the ‘-’ button will be pressed in order to generate a result. | The result was the expected one so the method works properly. |
| 3 | Check if the multiplication method works properly | Two polynomials will be given into the text field and then the ‘\*’ button will be pressed in order to generate a result. | The result was the expected one so the method works properly. |
| 4 | Check if the division method works properly | Two polynomials will be given into the text field and then the ‘/’ button will be pressed in order to generate a result. | The result was the expected one so the method works properly. |
| 5 | Check if the integration method works properly | A polynomial is provided in the first text field (Polynom 1) and then the integrate button is pressed. A result will be shown in the Result text field. | The result was the expected one so the method works properly. |
| 6 | Check if the differentiation method works properly | A polynomial is provided in the first text field (Polynom 1) and then the differentiate button is pressed. A result will be shown in the Result text field. | The result was the expected one so the method works properly. |

1. Results

As it was shown above, the program works properly. Each of the operations provided for the user by the calculator is correctly computed, so the result would be as expected. Some problems can occur when the user tries to divide a polynomial that has a degree which is less than the degree of the second polynomial. In this case, the operation isn’t performed and a message is display in order to warn the user about the fact. Only when a good input is provided the operation can be performed.

1. Future improvements

In order to improve the polynomial calculator, more operations can be added, such as root finding, multiplication by scalar or computing the value of the polynomial at a certain point. In order to implement the root finding operation some algorithms such as Bisection Method or Newton’s method should be implemented.

1. Conclusions – what I’ve learned

During the implementation of the polynomial calculator I’ve learned how to work better with Collections and Generics and how to take advantage of the features provided by Java. I have also learned how to override the method equals when needed, in order to see if two elements are equal. After implementing the functions which perform operations on the polynomials I have gained some experience in working with polynomials and how the operations should be performed efficiently. As a conclusion, I think that building this small java application I have also improved my programming skills and I have learned some important OOP principles.