2018

Polynomial Processing

GROUP 30423
ADRIAN - RADU MACOCIAN

FACULTY OF AUTOMATION AND COMPUTER SCIENCE | TEHNICAL UNIVERSITY – CLUJ NAPOCA

Assignment objective

Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

Secondary Objectives:

- Design a monomial class that consists of a grade and coefficient (Projection)
- Create the polynomial class, as a list of more monomials (Projection)
- Divide the operations into mono and binary operations and create interfaces for each type (Projection)
- Create a class that implements said operations for addition, subtraction, multiplication, integration and derivation (Projection)
- Create the division operation without implementing the interface since its output is the only one that is divided into 2 polynomials, the quotient and the reminder (Projection)
- Create the graphical user interface (Projection)
- Use the regex library to deconstruct the string input into polynomials (Projection)
- Override the toString method for the polynomials in order to make the representation readable (Projection)

Assignment analysis, assumptions, use-cases, errors

The polynomial processor must be capable to extract the monomials from a string and then execute any of the six operations (addition, multiplication, subtraction, division, derivation, integration) to one or both depending on the chosen operation. Errors may occur if the given polynomials don't respect the structure that is given by the regular expression. For that same reason we will assume that all the given polynomials respect the following pattern:

$$\ldots + q_1x\text{^}g_1 + q_2x\text{^}g_2 + \ldots$$

where q is the coefficient and g is the grade. Even though we are used, in mathematics, to ignore the coefficient if it's 1 and the grade if it's 0, in order for the application to work, the user has to write them anyway so that the regex can recognize the pattern.

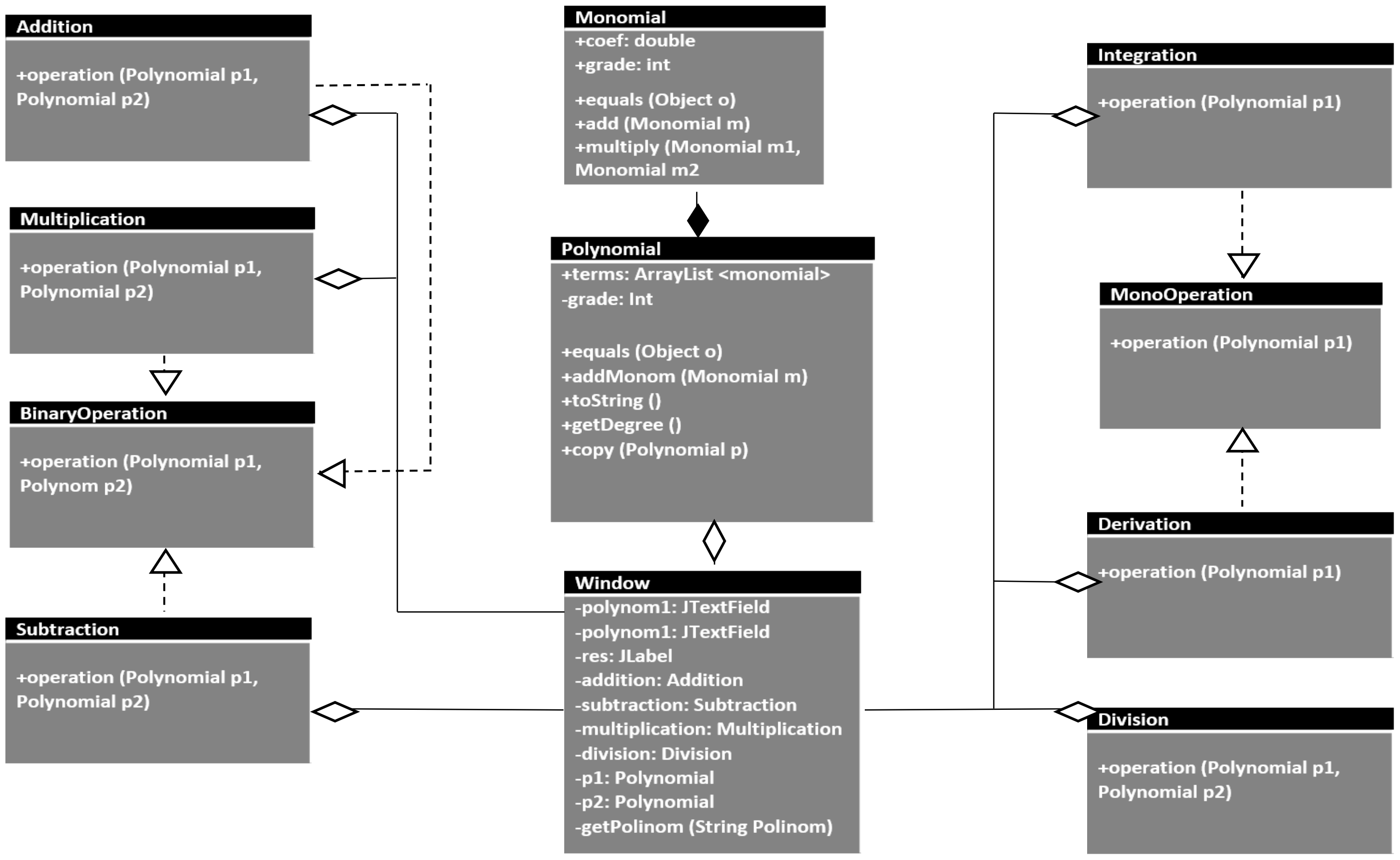
The use-cases are easy to follow: once we run the application, we just have to write the polynomials in the two text boxes assigned for them and then chose the operation we want to execute by clicking on the button titled by the operation. Since they can only be executed on one polynomial, the derivation and the integration operations both use just the polynomial in the left text box ("polynomial 1"). Pressing the exit button will, naturally, close the application. One of the exceptional use cases is the attempt to divide by zero, but fear not, the processor will take care of that problem by avoiding the user to not divide by zero in the result field, and of course by stopping the division right there

Projection

As described above, in order to maintain an object orientated programming structure, I decided to create a class that describes the monomial, having a coefficient and a grade, and the addition and multiplication of monomials as methods. I also described an "equals" method in order to use the junit package testing. Then I created a polynomial class that stores a list of monomials and the maximum its grade, which is used for the division of polynomials. It has a method that sets the grade of the polynomials and then returns it, a method to add monomials into the polynomial, which checks not to add the same monomial twice, in which case it will just add the coefficients. It also has an equals method used for junit testing, a static cloning method that crates another polynomial identical to the one in the argument of the method. This is also used for the division and is useful since it will create a completely separate polynomial instead of referencing the same monomials. And it overrides the toString method used to display the polynomials.

The graphical user interface is created with a the awt and swing packages. It uses a grid bag layout for the panel and it has a separate class that is used just to set the layout for visibility reasons only. Inside the window class, everything is done with action listener for the buttons and the only extra method it uses is the one that transforms the string given in the text boxes into polynomials.

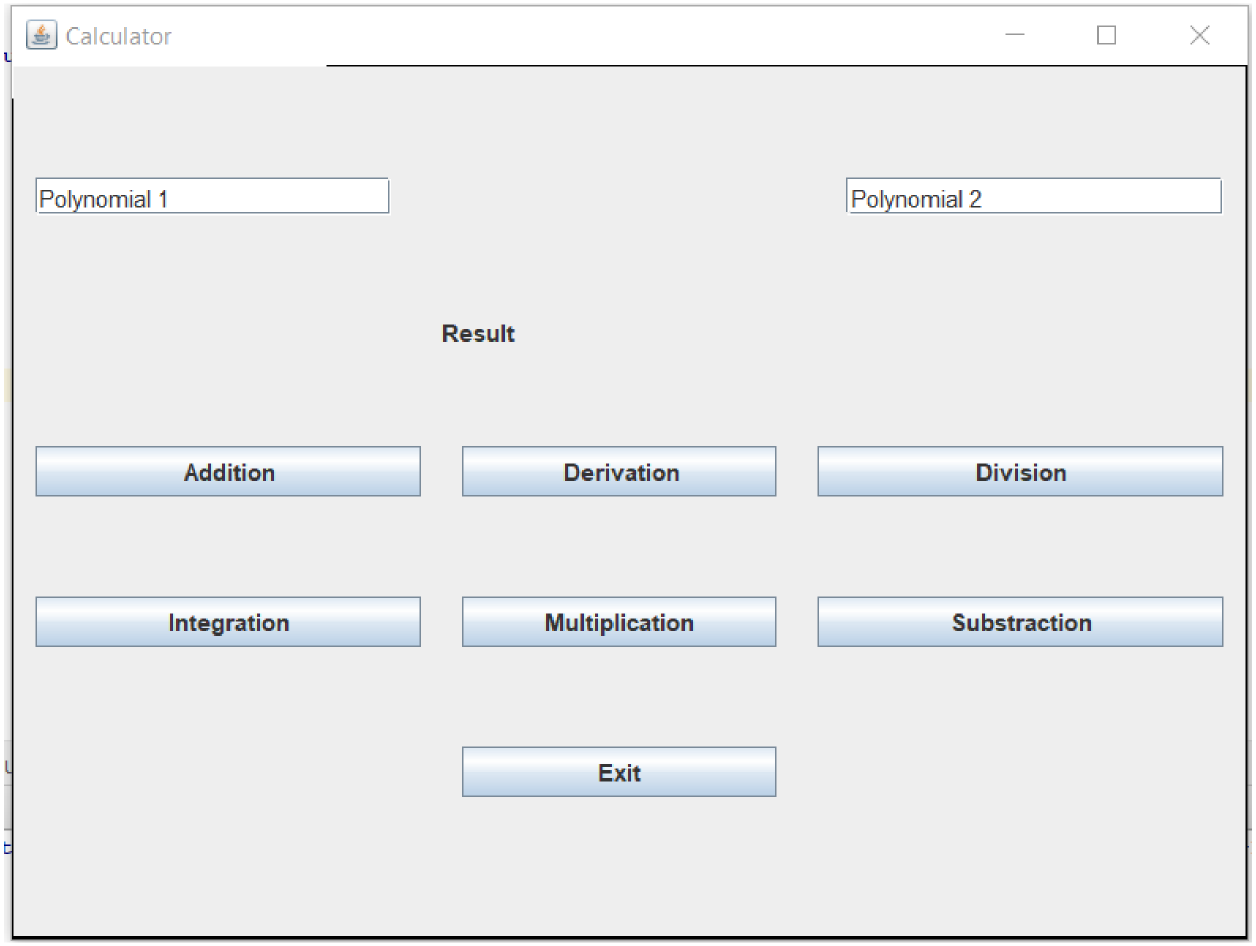
The logic package contains the two interfaces: mono operation and binary operation. Those two host a method which returns a polynomial and receive one polynomial or two respectively as arguments. The derivation and integration classes implement mono operation interface since they work on only one polynomial at a time, while the addition, subtraction and multiplication classes implement the binary operation interface. All of the classes do exactly what their name suggest returning one polynomial as a result. The only exception being the division class which has to return two polynomials, namely the quotient and the rest of the division. For this reason, it doesn't implement any of the interfaces and it has a solitary method which is also the only one to use the grade of the polynomial the cloning of the polynomials.


Addition
+operation (Polynomial p1, Polynomial p2)
Multiplication
+operation (Polynomial p1, Polynomial p2)
BinaryOperation
+operation (Polynomial p1, Polynom p2)
Subtraction
+operation (Polynomial p1, Polynomial p2)
Monomial
+coef: double
+grade: int
+equals (Object o)
+add (Monomial m)
+multiply (Monomial m1, Monomial m2
Polynomial
+terms: ArrayList <monomial>
-grade: Int
+equals (Object o)
+addMonom (Monomial m)
+toString ()
+getDegree ()
+copy (Polynomial p)
Window
-polynom1: JTextField
-polynom1: JTextField
-res: JLabel
-addition: Addition
-subtraction: Subtraction
-multiplication: Multiplication
-division: Division
-p1: Polynomial
-p2: Polynomial
-getPolinom (String Polinom)
Integration
+operation (Polynomial p1)
MonoOperation
+operation (Polynomial p1)
Derivation
+operation (Polynomial p1)
Division
+operation (Polynomial p1, Polynomial p2)

The only algorithm used is the Polynomial long division which was adapted since it's not possible to access the highest order term inside the polynomial but the barebones of the algorithm remain

Implementation

The graphical user interface uses a grid bag layout for the only panel it uses. Its resolution is set with an absolute value on 640 x 480 pixels and the position is set for the center of the screen.



Each individual button has its own action listener that executes the operation. Result is a label which shows the string conversion of the result of the given operation. The text is taken from the text boxes and converted into a string, then the regex converts said string into monomials and adds that monomials directly into polynomials that are passed as arguments for the operation called with the action listener of the buttons. All the operations return a polynomial which then has to be converted into a string with the overridden toString method of the Polynomial class and is passed to the label and shown on screen. The exit button is not necessary since we still have the utility buttons of the window but I added it for commodity reasons.

In order to avoid a huge wall of code created by the combination of the 7 buttons action listener I've created an extra setWindow class which receives the buttons, labels, text boxes, window and panel as arguments and then sets the whole interface up. This way the code looks cleaner and the Window class can be used only for the action listeners.

Testing

I have created 3 test classes, one for testing the monomial class and its methods, one for testing the polynomial class and its methods and one for testing each operation separately. Since the polynomial class does not have a constructor creating the tests can take quite some time so that's why all the tests are simple cases.

The monomial test class has two tests: one for testing the addMonom method and one for testing the monomials multiplication method

The polynomial test class tests crucial method of adding a monomial to the polynomial

The operation test class has 6 tests, representing the 6 available operations in the processor: addition, multiplication, subtraction, division, integration and derivation. All the tests are done on a one monomial polynomial since it's easier to check whether the tests are accurate this way.

Results

Following the test results, I found some problem cases, since the application uses double coefficients. For the division of the polynomials we have to divide the coefficients as well and sometimes, even if the input is a integer (stored in a double) the result can be erroneous and leading to even more errors down the way, thus the operation giving a false result.

Conclusions

The polynomial processor works as a polynomial calculator that is capable of doing basic operations on any polynomial. As long as the input syntax is respected the processor can take care of the rest. It's structured in an object oriented programming style extending the monomials into a list that becomes it's one entity: a polynomial. The operations are also designed to implement interfaces so that I can practice the OOP paradigms. The whole graphical interface is done manually through trial and error, but since no helping software was used, I'm capable of finding any bugs and fixing them a lot easier since I know exactly why I did something or why something is in a specific place.

The only exception to the object oriented programming style structure is the division operation since it was "special" in the way that it didn't just return one polynomial but two. Because of that I had to create a special class that is not extending or implementing since adding such a thing would have been useless. The only solution to this problem I could think of was changing the operation interfaces so that they return a string instead of a polynomial. That way it wouldn't matter how many polynomials an operation has to return since all of them would return the final string. The problem with this approach is that we don't have access to the result of the operation. It would be fine in this specific context, for a polynomial processor, but in any other case in which we would also want to access or modify the result we would need to convert the string back into a polynomial which wouldn't work in the case of a quotient and rest, thus the problem would persist.

I feel like I've learned a lot about object oriented programming while working on this assignment. I knew vaguely how OOP should work but in a more barbaric way. This assignment thought me a lot about interfaces since I didn't understand them before and I was avoiding them until I had to really use them just now. Besides the assignment I've also learned how to use a repository with git to back up my work and to be able to share it freely. I'm sure it is a huge help for working in a team since this gives the possibility to access each other's work in a matter of seconds.

One of the other big improvements in my coding abilities following this assignment is my ability to use awt and swing. I've created some user interfaces before but mostly using a specific software design especially for that. Being forced to write all the code by myself I've learned how the layouts, panels and windows work. And to my surprise this was not even that hard to learn once you understand the basic principles of panels and the specifics of the layouts.

Possible improvements:

As stated before the problem of the coefficients being integers and still being treated as real numbers can be solved by creating separate cases for real and integer coefficients. This would solve some of the inconsistencies created by working with double numbers. This would also solve a visual problem where the resulting polynomial always has the coefficients with at least a decimal, even when that decimal is 0. This is not necessarily interfering with the way the program works but it is slightly inconvenient and disturbing. The graphical interface could be made more colorful and it could be created only using one of the awt or swing libraries. It is advised not to combine them, since they could have some unforeseen interactions between each other that could mess with the interface.

One of the biggest possible improvements is working with the regex so that the input can be made more flexible and natural. Right now, the input has to follow that strict structure in order for the application to interpret correctly the polynomials but I could try to make it recognize the polynomials and interpret the lack of a coefficient as the coefficient being 1 or the lack of an x being the grade = 0.

Another improvement could be made by finding a way to implement the division operation using the interfaces so that it's not just a solitary class, the purpose of the assignment being using the object oriented style of programming.

Bibliography:

Most of the project was done by myself with the help of the laboratory professor, who gave us some ideas on how to make the implementation as object orientated as possible. The only two external information sources were used for the creation of the graphical user interface and for creating the division of polynomials:

https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

https://en.wikipedia.org/wiki/Polynomial_long_division