Polynomial calculator

Haș Darius

30423/1

1. The assignment's objective

The main objective of the assignment is to perform basic operations with polynomials with positive powers: addition, subtraction, multiplication, division, derivation and integration.These operations are carried out by following a particular pattern, pattern that ensures that certain intermediate steps (secondary objectives) are performed: polynomials are introduced as strings, so it is absolutely necessary that their shape be verified and transformed into an OOP model that facilitates their good management (validation of polynomials is done in Controller class, being detailed in chapter 4, and the OOP model is given by the Polynomial class containing a list monomials, having two attributes: power and coefficient, being also detailed in chapter 4). Another secondary objective is the design of the logic that form the basis of each operation, as well as developing an UI (User Interface) through which the user can enter the polynomials and select the operation that he wishes to perform.

2. Problem analysis, modelling, scenarios, use cases

In what follows, in order to do a problem analysis, a use case with different scenarios will be presented below:

Use Case: Polynomial Calculator

Primary Actor: The user who has access to the application

Success scenario:

- The application waits for the user to introduce two strings which actually represent two polynomials in the user interface that is provided.
- The user enters two valid polynomials.
- He chooses the operation he wants to perform.
- The validity of polynomials is verified.
- The strings entered are converted into lists of monomials, each list actually forming object of the polynomial class.
- The operation that was chosen is performed on the two objects of the polynomial class.
- The result (which at this stage is an object of the polynomial class) is converted into a string.
- This string is displayed through the UI the user.

Alt scenariu:

- The application waits for the user to introduce two strings which actually represent two polynomials
  - The user enters two non-valid polynomials.
  - He chooses the operation he wants to perform.
  - The validity of polynomials is verified.
  - A message is displayed informing the user that the polynomial entered are invalid.
  - The user must press ok, then the application is right back in the state where it awaits for the user to enter two strings/polynomials.

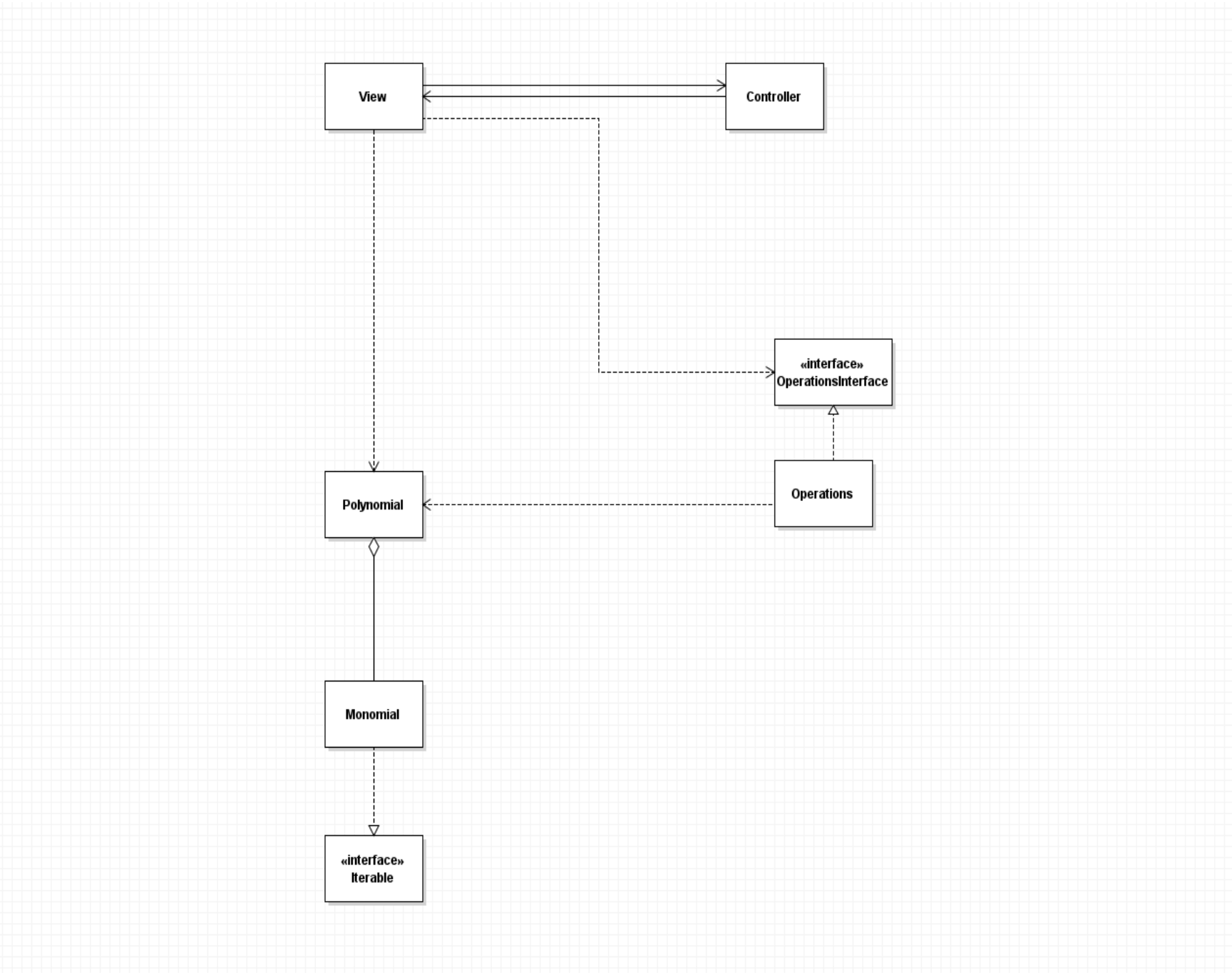
It can be notable that the introduction of polynomial as strings brings after itself the need to validate them (the user has total freedom over the input, so it is essential that it is verified). An OOP model is also required to ensure the proper functioning and performance of operations (Monomial and Polynomial classes, as detailed in Chapter 4).

3. Design (design decisions, UML diagrams, data structures, design classes, interfaces, relationships, packages, algorithms, user interface)

Talking about the decisions in the OOP design, the classes that help replicate the polynomial's behavior and characteristics are the Monomial class and the Polynomial class that, actually, contains a linked list of Monomial objects. Other classes are: View, Controller, that are associated one with the other and help develop a User Interface. Also, we have the Operations class that implement the methods provided in the OperationsInterface interface, actually holding the logic that is behind each of the operations that this application is able to perform. These classes will be analyzed more in depth in the next chapter. Talking about the interfaces used, the Monomial class implements the Comparable interface, in that way providing the logic necessary (by overriding the method compareTo()) to keep the list of monomials in the polynomial class sorted. Also, the interface that the Operations class implements, OperationsInterface, provide the definition of each of the arithmetic operations that can be performed.

As it was mentioned above, the data structure used in this assignment is a LinkedList, that is sorted, when it is formed, after all the elements are inserted into it, the order of them being maintained throughout the performing of any operation.

In terms of packages, the application contains Packages that mimic the MVC architectural pattern (we have the package Model, that contains the Monomial and Polynomial classes, the package View and the package Controller). Also, we have the Operations package, that contains the OperationInterface and the Operation class, but also a package for testing, in which we have two classes : OperationsTest that contains methods that test each of the arithmetic operations, and ControllerTest that contains a method that tests the validity of a polynomial (Checks if the string entered is a valid polynomial). The UML diagram of the assignment is:


View
Controller
«interface»
OperationsInterface
Polynomial
Operations
Monomial
«interface»
Iterable

The relationships between the classes are as follows:

Between the View and Controller classes there is an association. The Operations class implements OperationsInterface and Monomial class implements Comparable. The Operations and Polynomial classes have a relationship of dependency, as do the Controller and Polynomial and the Controller and Operations classes, as it is shown in the above UML diagram. The Polynomial class is an aggregate of the Monomial one, because it contains a List of Monomials.

The user interface is developed using java Swing and it presents two text fields where the user can enter the polynomials on which he wants to perform an operation, a label that at first shows the message "The result will be displayed here" and later will show the result and buttons for each application:

- "+" for addition
- "-" for subtraction
- "x" for multiplication
- "/" for division
- "∫" for integration
- "∂" for derivation
- A button "Another application" that acts like a resitter: it blanks the text fields and the label will once more show "The result will be displayed here" allowing the user to perform another operation.

If the provided strings are not valid polynomials, it will be displayed a message on the screen informing that the polynomials are not valid(a pop-up message).Also, if a non-valid operation is being tried, another pop-up message will be shown(division by 0).

4. Implementation

4.1 The Monomial class- contains two fields: coefficient and power which are both of Double types. It contains getters and setters for these and it also overrides the method compareTo() of the implemented interface Comparable, comparing two monomials first by their power and then, if their power is equal by their coefficient (this was done because the user has the freedom to add a polynomial that is =3x in that way "1x+2x", so it was mandatory in order to keep the list of the Monomials contained in the Polynomial class sorted).

4.2 The polynomial class- contains a list of Monomials. As constructors, it has two, one that has no parameter that just instantiates the list as a new LinkedList, and another one with a parameter of type string.In this constructor the conversion of the string into the polynomial Class is done in the following way: the string is split into a list of strings of monomials of

form(<sign><coefficient>x^<power>) by the “+” character that separates them. If between the monomials there was a “-“ character, it is replaced before the split with “+-“. Then, after the split, from each monomial there are extracted the power and the coefficient by paying attention to certain cases such as the polynomial with the power 0(x^<power> is omitted , the polynomial with the coefficient 1 or -1(<coefficient> might be omitted). At each step a new monomial is created, it is being given the power and coefficient and then it is added into the Polynomial. After all of the strings of monomials were converted into actual monomials and inserted into the polynomial, the polynomial is being sorted. After the polynomial is sorted, a method of this class is called: avoidDuplicates. This method makes sure that even if the user enters a string polynomial for example “1x+x+3x”, the actual Polynomial will contain only a monomial with the coefficient 3 and power 1, equivalent to 3x.So, it makes sure that each monomial that is in the list of the Polynomial is of a different power. It is always called after the sorting of the polynomial, in that way making sure that by iterating through the list, no two neighbor monomials have the same power, and if they do, they are grouped together by adding their coefficient into one of the monomials and removing the other from the list.

Another method of the polynomial class is the toString() method that is in fact the exact opposite of the constructor with the string parameter. It transforms each Monomial object into the list into a string and adds it to the final string that will be printed for the user, also paying attention to certain special situations.

The getListOfMonomials() method acts like a getter for the list and addElement() method adds a new Monomial element to the Polynomial.

4.3 The Operations class – implements the method that were declared in OperationsInterface.

Because of the list being ordered, addition and subtraction methods are performed sort of like the merge of two ordered lists. There is an iterator through each list and in a while loop, each time it is added in the result the monomial with the greater power (in addition both with + sign, in subtraction the monomial belonging to the second polynomial with – sign and the other one also with +). In case that both monomials have the same power, in the result it is added a monomial with the same power and with the coefficient equal to the sum or difference of the coefficients, depending on the operation.

The multiplication method simply iterates through both lists multiplying monomial by monomial and adding the result to the final polynomial. The final polynomial is sorted and then it is applied the avoid duplicatesMethod to add the monomials with the same power.

The division method is based on the long division method. Actually, in a while loop, while the first polynomial’s grade(the greatest power, the power of its first element, because it is ordered this is the greatest power) is greater or equal to the grade of the second there is computed a partialQuotent having the power the difference between the first and second polynomials’ powers and coefficient the division of the same polynomials’ coefficients and the remainder(not the final reminder, just one that is temporary) as

the multiplication between the second polynomial and the partialQuotient. After that, the partialQuotient is added to the final Quotient and the first polynomial gets the difference between him and the remainder.

After the while loop, the quotient will be ready and the final remainder will be found in the first polynomial. The quotient and the remainder will be returned.

The integration method is done simply by iterating the list, each monomial's power gets an increment and its coefficient gets the division between him and the newly incremented power.

The derivation method is done similarly, but the coefficient gets coefficient multiplied with the power and after that the power is decremented by 1 for each monomial.

While the first four operations can be performed only if provided both polynomials, the integration and derivation methods can be performed only by providing the first polynomial.

All these methods return a result of type Polynomial, except from division, which returns a List<Polynomial> that contains two polynomials (Quotient, remainder).

4.4 Controller class – it contains a field of class View by which it is associated with the class View. It also contains methods for validation() which checks if the two strings entered are valid as polynomials in the following manner:

There are formed two lists of strings of monomials by splitting the initial string after the "+" character. After that, by iterating, each monomial string is verified if it matches a certain desired form with regEx:

"-?[1-9]\d*[x](\^[1-9]\d*)?" – standing for a string that can or not begin with "-" that has after it a digit different from 0 (023x is not permitted) followed by any number of digits, followed by "x^" a digit different from 0 (3x^021 is not permitted) followed by any number of digits ; it basically checks if the monomial has the desired form : <coeff>x^<power> (coefficient can also be negative, that's why we have "-?"), also the part that corresponds to x^<power> can miss in this case.

Or

"0" – the user can also add 0 which is a special monomial

Or

"-?[1-9]\\d*" – standing for the monomial that has the power 1 and is essentially a number

Or

"-?[x](\\^[1-9]\\d*)?" – standing for the polynomial that has the coefficient + or – 1 (such as x^7 or -x^2)

If the string of any monomial doesn't match any of these valid forms it means that it is not correct and the method will return false. Otherwise, true.

The Controller class also contains openPopUp() method which displays a message on the string in case an operation is asked to be performed on a/an invalid string/s. Also, the Controller implements ActionListener and overrides the method actionPerformed() that checks that if a button is pressed, its corresponding operation is performed if the strings introduced are valid or contains a call to the openPopUp() method otherwise. For the integration and derivation methods, there is enough to provide just one polynomial(the first one), but for the other ones the user has to specify them both.Even if there are two polynomials provided, only the first one will be integrated,derivated.

4.5 View class – contains a Controller field and a panel field, being instantiated as GridBagLayout().It contains buttons for all the operations and getters and setters for them, textFields for the strings to be introduced and a label where the result will be displayed. Its most important feature is its Constructor that is responsible for the layout of the user interface. Using GridbagLayout, we have two parameters => gridx, gridy which help arrange everything as a matrix (by column and line). So, as an example, on the first line we have gridy=0 and at first column(gridx=0) we put the textField for the first polynomial, at second column(gridx=1) the button for addition and at the third one the button for the subtraction.

5. Results

There are two kinds of tests: OperationsTest and ControllerTest.

In OperationsTest, there were given two pairs of polynomials:

```
Polynomial polynomial1=new Polynomial("-7x^5+4x^8-x^3-x-1");

Polynomial polynomial2=new Polynomial("3x^4+3x^6-x^7+x+12");

Polynomial polynomial3=new Polynomial("-x^2-x-1");

Polynomial polynomial4=new Polynomial("-2x^2+x-2");
```

The addition, subtraction, multiplication and division were performed on them.

For integration and derivation there were given three possible polynomials:

```
Polynomial polynomial1=new Polynomial("-7x^5+4x^8-x^3-x-1");

Polynomial polynomial2=new Polynomial("3x^4+3x^6-x^7+x+12");

Polynomial polynomial3=new Polynomial("1");
```

On the operations testing there were given only valid polynomials because this is the only scenario in which the actual operations can be performed(If polynomials are not valid, then there is generated an error message for the user indicating that he didn't introduce proper polynomials and no operation is performed).

In ControllerTest, there were given strings for testing if they are valid or not, using the validation() method of the controller.

The method validation() is implemented with two strings as parameters because most operations make use of two polynomials

to demonstrate functionality, so to demonstrate functionality there was used the same string as both parameters. Some examples of strings tested are: "1x","0x","012","x^02","01x", "-7x^5+4x^8-x^3-x-1","3x^4+3x^6-x^7+x+12".

In both classes of tests, tests were performed using Assert.assertEquals(result,expectedValue) instruction.

6. Conclusions

The polynomial calculator performs six basic arithmetic operations between two polynomials with positive powers. As a later improvement, it could perform operations on polynomials with negative powers and also, it could perform other, more complex operations that makes use of these ones already implemented.

7. Bibliography