Polynomial calculator

(Documentation)

1. Homework objective:

Main objective: The implementation of a polynomial calculator, which takes as input two polynomials and displays the result of basic mathematical operations.

Secondary objectives:

-modeling the project in a MVC manner, using different packages for specific classes ( )

-implementation of the Graphical User Interface ( )

-implementation if basic mathematical operations, applied on polynomials of one variable and integer coefficient & power

1. Problem analysis, modeling, scenarios, use cases

Problem analysis: It is required to correctly & efficiently implement the following operations: addition, subtraction, multiplication, division, derivation, integration, on polynomials of one variable and to display the result according to mathematical rules & structure.

Modeling: The project is modeled in a MVC manner, having 3 different packages, model, view and controller. The Monomial and Polynomial classes, in-between which exists an aggregation relationship (A list of monomials represents a polynomial), are both under the model package. The GUI class, in which are defined the graphical objects (buttons, text fields, labels) is under the view package. Finally, the Control class, in which graphical elements interact with data definitions and algorithms, is under the Controller package

Scenarios: A scenario represent a sequence of steps taken by the user to correctly interact with the application. The following are the possible scenarios:

Scenario 1: The user successfully makes the desired operation. The polynomials taken as input are valid, therefore any desired operation could be performed. The result of the chosen operation will be displayed.

Scenario 2: The user doesn’t succeed in making the desired operation. The failure is the result of an input error (user inserting invalid polynomials <other variables, characters in a wrong order> , or failing to insert the polynomial in the expected text field for the derivation and integration operations). The correct insertion structure is the following: “sign” + space + “monomial” + space “sign” + … . On the output text field will be displayed the specific error message (Input error, too many arguments).

Use cases:

A screenshot of a cell phone

Description automatically generated

1. Design

The project is designed under a MVC pattern, having 3 separate components: Model, View and Controller. Model holds the Monomial and Polynomial classes, View holds the GUI class and Controller holds the Control class. The Control class manipulates data stored under the Model package and interacts with the desired components of the graphical user interface, defined in the GUI class.

The GUI class contains buttons for each operation (Addition, Subtraction, Multiplication, Division, Derivation, Integration), three text fields for the first polynomial, the second polynomial and the result of the chosen operation, three labels and a clear button. Each button triggers a listener and calls different methods, in order to obtain the desired output. The two strings taken as input are saved in Array Lists, converting the coefficients and the powers of the monomials in integer values. The said Array Lists are then taken as method parameters, leading to the expected result. The Monomial class contains information about the power & the coefficient of the element, while the Polynomial class contains an Array List in which the monomials will be stored, and the implementation of the requested operations.

The addition operation adds monomials of same power. If one is null and the other isn’t, the existent monomial will be added. The result is obtained by adding the coefficients of the monomials.

The subtraction operation subtracts a monomial from another. It performs the subtraction operation on monomials of same power.

The derivation operation computes the derivative of each monomial. The derivative form is obtained by multiplying the coefficient of a monomial with its power, then decreasing the power.

Operation: (axn)’ = naxn-1

The integration operation computes the integrated of each monomial. The integrated form is obtained by dividing the coefficient with the successor of the power, then increasing the power.

Operation: I(axn) = a/(n+1)xn+1

The multiplication operation multiplies two polynomials by adding the powers & multiplying the coefficients of each of their monomials with another. The resulting monomials are stored in a Hash Map. If two monomials are to be stored on the same key (they have the same power), the value of the stored monomial will be overwritten by the sum of the two monomials of same power.

The division operation does not work as intended (mathematically correct), due to the fact that the remainder was not implemented. Therefore, computes the result by dividing each polynomial’s monomials one with another, subtracting the power and dividing the coefficients.

The algorithm of saving the monomials consists of splitting the arrays taken as input by spaces, and saving each element on a position of a string[]. If the elements is an operand (+ / -), we append it to the next item, which would be the monomial. If the first inserted monomial does not have a sign by default, we assume it is positive and append the “+” character to the monomial. We then create a monomial with the inserted values (coefficient & power), and add it into the Polynomial Array List. The creation of a monomial consists of taking the input string(format: “operand” + “coefficient” + x^ + “power”) and splitting it by x. We will then have a string[] of potentially two positions, one storing the coefficient, and one storing the power and a “^” character (ignored). We then convert the strings into the corresponding integers, which will be saved. Exceptions will be thrown if the input string has an invalid format, and an error message will be displayed.

Each button will display data regarding the input values. If the input string is invalid or there are too many arguments (derivation & integration operations), there will be an error displayed. If the input values are valid, the result of the chosen operation will be displayed.

Relations:![A screenshot of a cell phone

Description automatically generated]()

The data structure used is the list, using the Array List implementation.

The user interface represents a sequence of components, each labeled by its own purpose: inputs, output or operations.

1. Implementation

Classes:

Monomial:

Has two attributes, coefficient and power and two constructors, one which takes integers as parameters and sets the attributes, and one which takes a string as parameters, and converts it into the corresponding coefficient & power integers. It also contains a method toString(), which converts the attributes into strings, and getPower() & getCoefficient() methods, which return the particular attributes.

Polynomial:

Has two attributes, public Integer max\_power which stores the highest power of a monomial, and private ArrayList<monomial>, in which we store each monomial. The constructor sorts the list with respect to the power, and computes the maximum power. The toString() method converts each monomial into an array (format: “coefficient” + “x^” “power”), considering some rules. By checking the values of the coefficient & power fields, we will convert them into specific strings (when the coefficient is +/- 1, the coefficient ill not be displayed <only the operand>, when the power is 1, it will not be displayed, etc). The getCoefficient(int pow) method takes as parameter an integer, and returns the coefficient of the monomial whose power field equals the parameter. Polynomial add(Polynomial x) performs the addition operation. It creates an empty Array List in which the resulting monomials will be stored. The resulting coefficient of the monomial will be stored in the coefficient variable, and is computed by summing the coefficients of monomials of same power. The method returns a new Polynomial created with the resulting list. Polynomial sub(Polynomial x) performs the subtraction operation. It creates an empty Array List, and stores the resulting monomials. The monomials will be created after computing the difference of the coefficients of monomials of same power. The method returns a new Polynomial created with the list. The Polynomial mul(Polynomial x) method performs the multiplication operation. It creates an empty Array List and an empty Hash Map. It computes the resulting polynomial by multiplying the coefficients & adding the powers of each monomial with another(monomials belonging to different polynomials). The monomials are stored in the map on the key equal to their power. If two monomials are to be stored on the same key (they are monomials of same power), we compute the sum of their coefficients, and save the new monomial on that position. We then create the Array List by adding all the elements found in the Hash Map (we’re searching the set of keys). The Polynomial div(Polynomial x) method computes the result of the division operation. It creates an empty Array List and an empty Hash Map, and, similarly, saves the monomials in the map with respect to their power. The polynomial is computed by dividing the coefficients & subtracting the powers of each monomial with another. We then search the map’s key set, compute the array the create and return the new polynomial. The remainder of the division operation was not implemented, therefore the operation does not compute the correct polynomial. The public Polynomial der() computes the result of the derivation operation. It computes the new polynomial by deriving each monomial. One monomial is derived by multiplying the coefficient with the power, and decrementing the power. Each derived monomial is saved in an Array List, then the final polynomial is created. The public Polynomial integ() method computes the result of the integration operation. It computes the new polynomial by integrating each monomial. One monomial is integrated by dividing the power by the power’s successor, then incrementing the power. Each integrated monomial is saved in an Array List, then the final polynomial is created.

GUI:

This class contains the definition of each graphical object: 3 JTextFields for inserting two inputs and displaying the output, and 7 buttons: addition, subtraction, multiplication, division, derivation, integration and clear. The constructor of this class sets the editing property of the output textfield to false, defines a new panel of type JPanel, sets the layout to FlowLayour, then adds all the components to the panel. The class also contains methods for adding action listeners to the buttons, and definitions for buttons used with Junit.

Control:

This class has as attributes a variable of type GUI and two polynomials. The constructor creates the listeners for each button. The ArrayList<Monomial>conversion(String ex) method computes all the monomials of each polynomial. It splits the string taken as input by space, and saved all the resulting substrings into a string[]. Then, we check all the elements in the string[]; if the element is an operand (+/-), we append it to the next element, which would be a monomial. If the first element is a monomial and not an operand, the coefficient is positive and a “+” operator is automatically appended to it. We then store the resulting monomials in an Array List. The method also counts the number of apparitions of ‘x’. If x appears more than once in a substring, an error message will be displayed. The public void method() creates the polynomials by applying the conversion method on the two strings given as input. The int testFields(String s1, String s2) method verifies the contents of the text fields. There are 4 cases: when both of them are empty, when the first one is empty, when the second one is empty, and when neither of them is empty. The inner class operationListener which implements ActionListener is responsible with detecting action events. Mainly, we use it to detect when a button is pressed. Firstly, we try to call method() in order to create the polynomials. If the format of the polynomials is wrong (illegal characters are detected), the NumberFormatException is thrown, and an error message is displayed. We then verify whether any button was pressed or not in a try block. If the NullPointerException is thrown, an error message will be displayed. When a button is pressed, the actionevent detects which one, displays the result of the desired operation. Using a switch-case, we know whether the operation took place or there was an input error by checking the variable returned by testFields(String s1, String s2).

1. Results

Junit was used for testing the application. Each operation was tested once, excepting the division operation, whose remainder was not implemented.

Addition:

Input 1: 5x2 + 2x + 1

Input 2: x2 + 4

Expected output: 2+2x+6x2

Our output: 2+2x+6x2

The test was successful.

Subtraction:

Input 1: 5x2 + 2x + 1

Input 2: x2 + 4

Expected output: -3+2x+4x2

Our output: -3+2x+4x2

The test was successful.

Derivation:

Input : 5x2 + 2x + 1

Expected output: 2+10x

Our output: 2+10x

The test was successful.

Integration:

Input : 5x2 + 2x + 1

Expected output: x+x2+1.7x3

Our output: x+x2+1.7x3

The test was successful.

Multiplication:

Input 1: 5x2 + 2x + 1

Input 2: x2 + 4

Expected output: 4+8x+21x2+2x3+5x4

Our output: 4+8x+21x2+2x3+5x4

The test was successful.

1. Conclusions

This assignment has given me a better understanding of the model-view-controller pattern, and has made me look up different tricks, such as how to set the precision of a float variable using DecimalFormat, how to split an array with respect to a certain condition, and how to apply Junit tests to my application. In conclusion, this assignment has been a great opportunity to both learn new things and revise the information that I already had.

1. Bibliography

<https://coderanch.com/t/400972/java/setting-precision-float-values>

<https://javarevisited.blogspot.com/2016/03/how-to-remove-first-and-last-character-from-String-in-java-example.html>

<https://www.leepoint.net/GUI/components/30textfield/11textfield.html>

<https://www.leepoint.net/GUI/structure/40mvc.html>