**Polynomial Calculator**

made by Onet George, group 30421

**Objective**

The objective of this application is to perform several operations on two polynomials introduced in an interface.

**The analysis of the problem**

A polynomial is an expression consisting of terms, also called monomials or just elements. These terms contain a power and a coefficient, both integer values, and are associated to a variable or indeterminant. In our case, the indeterminant is ‘x’. Every single-variable polynomial can be written is this form:



or, in a more general manner, in this way:



where is the power and is the ’th coefficient. So, in a more programming point of view, a polynomial is just a list of monomials, where every element has a power and a coefficient. In this way, the operation will be performed in a simpler way.

**Modeling the problem and the packages**

The user is able to introduce two polynomials from the keyboard or by using buttons and perform several operations:

-the addition of two polynomials

- the difference of two polynomials

- the division of two polynomials

- the multiplication of two polynomials

- the derivative and integration of one polynomial

The result of the chosen operation will be displayed in a pop-up window.

**Different scenarios and use case**

The application lets the user to introduce in a TextField the polynomail in this form 5x^4-6x^13 and so on. The application will not work if the user does not introduce a polynomial or will just enter 0 in one polynomial, throwing an error message on the screen. If, for some reason, the user wants to introduce a monomial which has 0 as coefficient, the application will let this action flow normally, but that monomial will literally not exist. Another way to add the elements of the polynomials is by adding using the add button on the right of the screen, but the user will enter the monomials in power-descending. For example, if the user adds the first element which has the power 5, then he/she will be able to introduce only the powers from 0 to 4, selecting it from a ComboBox or a drop-down.

**Design**

The problem is separated in two packages: the first one, called PolynomialPackage, cosnsist of the classes used for building polynomials:

-Element class is used for creating elements or monomials;

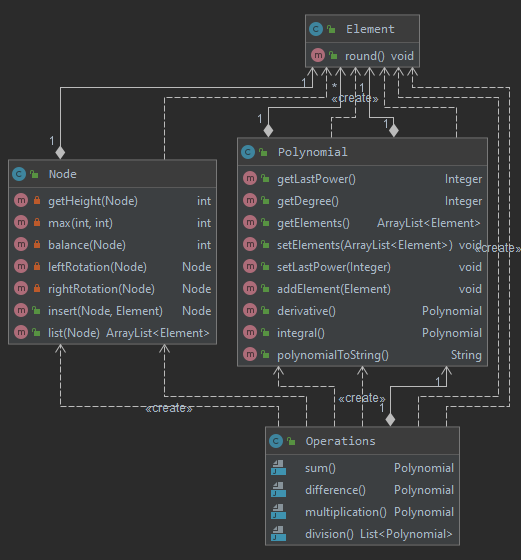
-Polynomail class is used for creating polynomials and for the derivative and anti-derivative operations;

-Operation class is used for the operations of the polynomials;

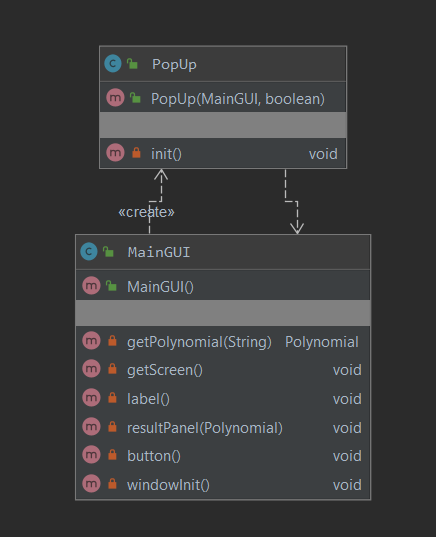
-Node is a class used for an AVL, which will come in handy for the multiplication operation

The other package is used for the GUI, or graphical user interface of the application. This package has two classes, one is used for the main interface, also called MainGUI, and the other one used for Pop-ups, which is called PopUp. The main class of the program is separated from the other packages.

This is the UML diagram of the PolynomialPackage’s classes:



And here is the diagram of the InterfacePackage’s classes:



**Data Structures**

The data structures that are used in this application are either primitive data types, such as int or double, or more complex ones, for example ArrayList. I have created an AVL in the Node class, where the program stores the elements in an AVL tree, making the complexity of the multiplication operation much more efficient than using a List or an ArrayList because of the insertion and the search aspect. The traversal of this tree structure is an iterative way, rather than a recursive one.

**Class Design**

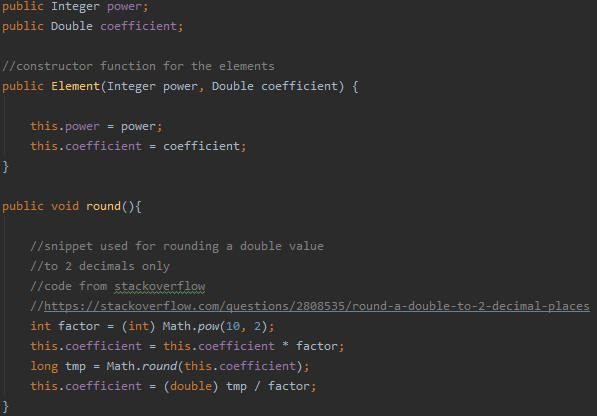
Each class consist of attributes, constructors, getters, setters and methods. There are some classes that omitted some of the functions enumerated above. Let’s start with the Polynomial package.

***Element***

The first class is the Element Class which has two attributes: the power and the coefficient. There is a constructor which has two parameters: the newPower and the newCoefficient and will give the new element created that values. For example,

newElement = new Element(5, 0.0);

Here the coefficient is a double value because of the integration and division operations of the polynomials, but the user will not be able to insert a ℝ/ℤ value. This class also has a method which rounds the double value and shows only two digits of the number (7.533333 will be 7.53).



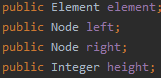
***Node***

Node class is a class used for creating an AVL-tree for the multiplication operations of two polynomials. The attributes are:

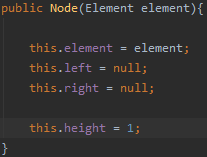
-the actual element which is inside the node

-the left and right parameters which get the left and right node of the current one

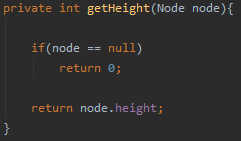
-the height which will help for the AVL part of the problem.



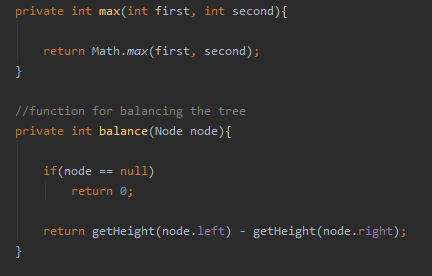
The ***constructor*** of the class will have a parameter which will be an Element variable. The method will create a new Node variable which will have the height equal with 1, the element will be the element from the parameter. The left and right node will be initially null.



There is a getter, which will be ***getHeight.*** The method will return 0 if the node itself it is null, otherwise will return the height of the node.



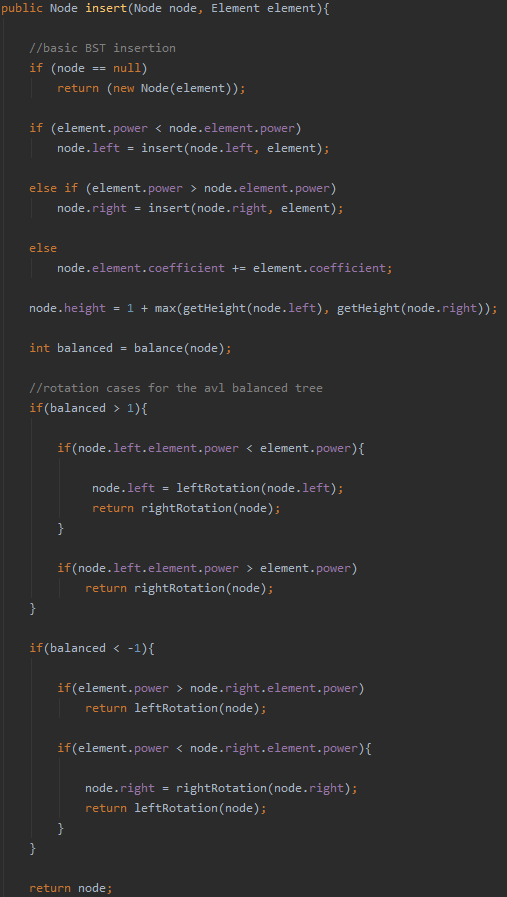
***Max*** and ***balance*** are another two methods used for creation of the AVL. Max has two parameters and will return the bigger one, and the balance will have only one parameter and will return the getHeight(left) – getHeight(right) if the node is not null, 0 otherwise.



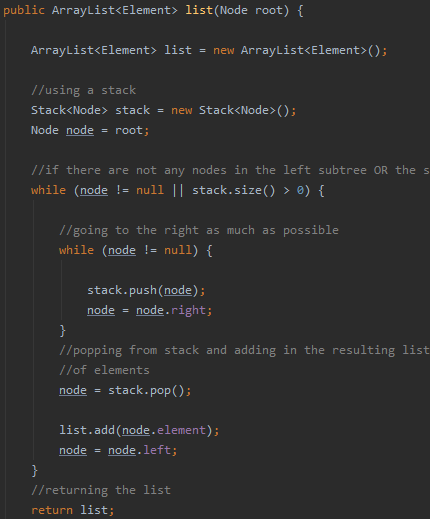
The rotation methods have the role of rotating the tree in order to be balanced all the time. There are two possible rotations: left or right, and the functions are called: ***leftRotation*** and ***rightRotation.***



The addition in the tree will be done by ***insert*** method. The insert is perhaps the most complex function in the program because there are 4 cases for the rotations: right rotation, left rotation, left-right rotation and right-left rotation.



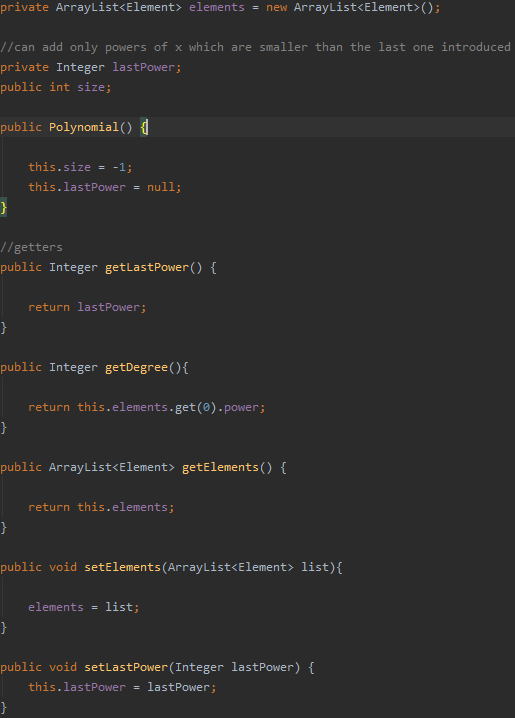
And the last function in the Node class is the ***list***, where the method will return an ArrayList which will contain all the elements of the tree in power-descending order, so the function performs an inorder traversal of the tree, but there will be a twist. Instead of going as far as possible in the left, the algorithm will go as much as possible to the right. The traversal will be done ITERATIVLY, using stack.



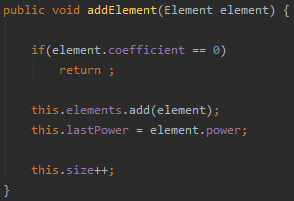
***Polynomial***

This class will have the role of adding new elements in the polynomial itself and doing two operations: derivative and anti-derivative or the integral. The class has three attributes: the array-list of elements, the last power introduced which will come in handy in the GUI and the actual size of the polynomial.

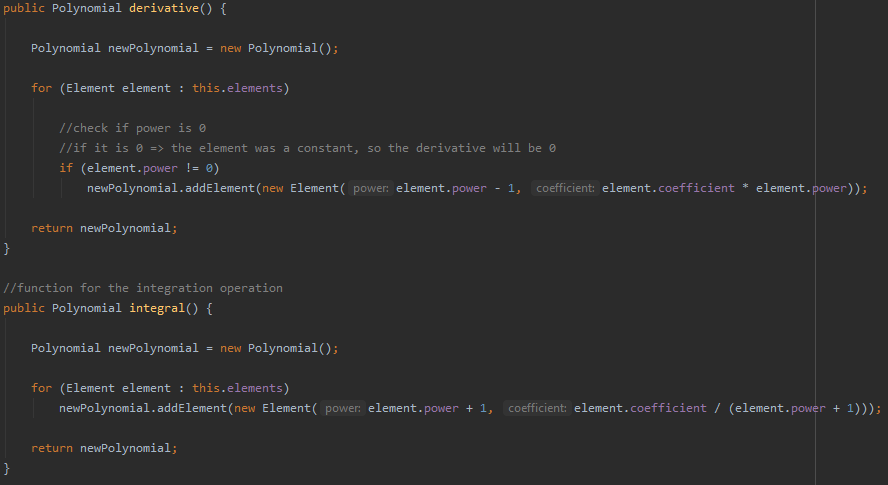
The ***constructor*** will give the size the value -1, meaning that the polynomial is not initialized yet and the last power a null value. There are getters and setters for every attribute which will return or set whatever is in their name.



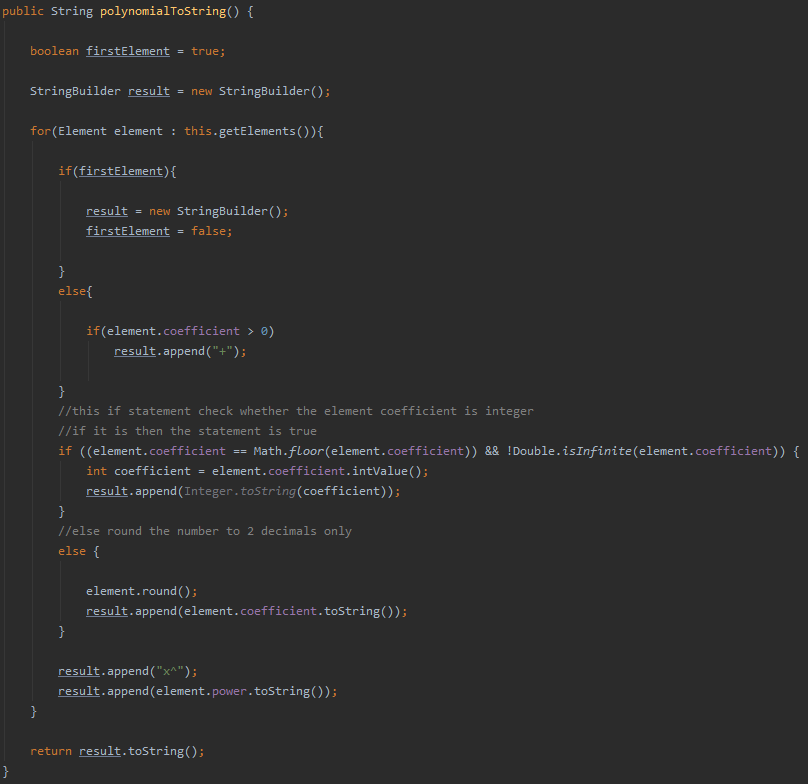
***addElement*** is a function responsible for the adding elements int the list of elements. This will have a parameter, an Element value.



The two operations performed by this class are, as I mentioned earlier, the ***integral*** and ***derivative***. Both of them will return a new polynomial. The integral will increase the power of each element and will divide the coefficient by the new power. The derivative will do the opposite: will take each element and decrement the power by one and will multiply every element’s coefficient by the last power.



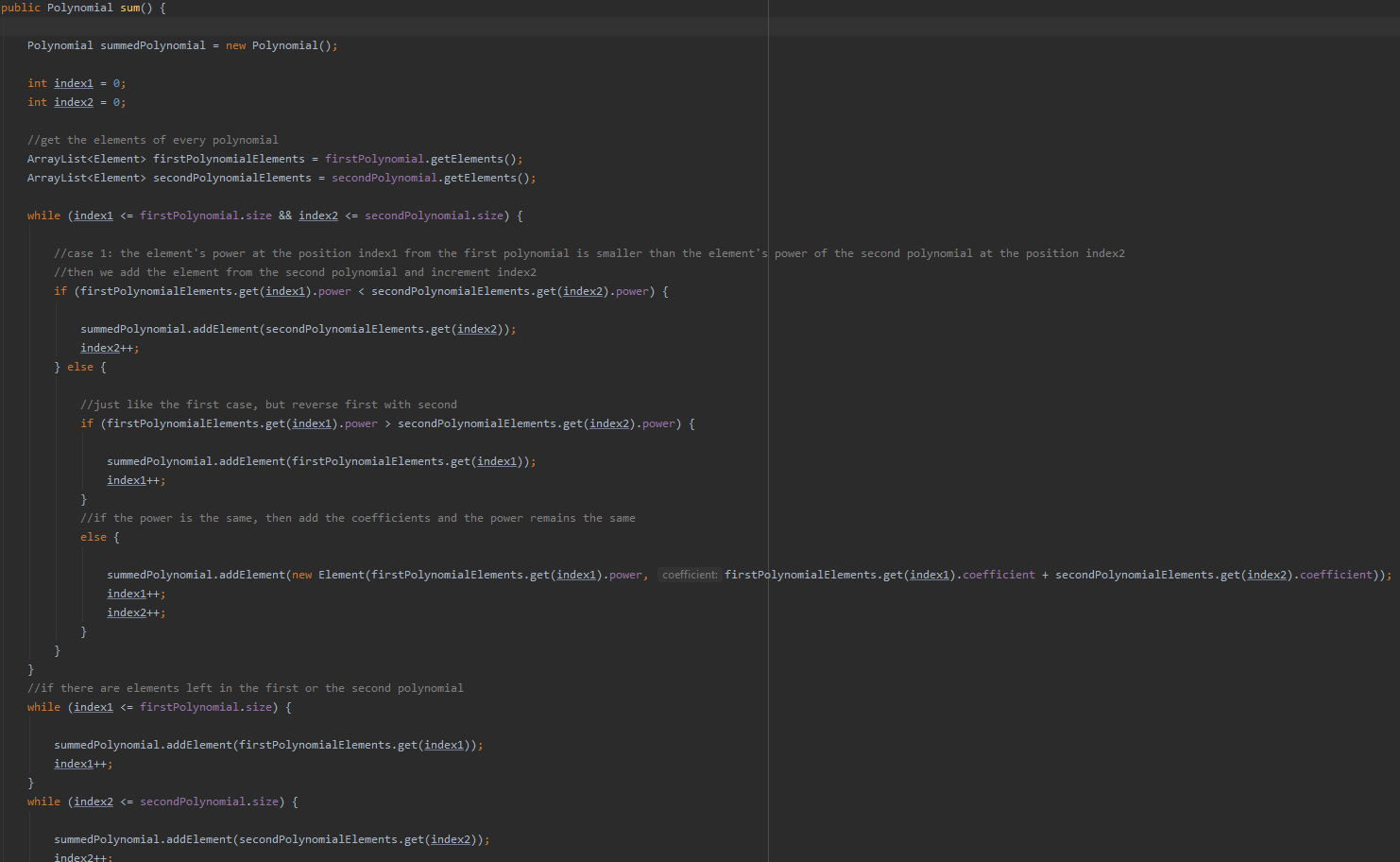
The last function is the ***polynomialToString***. This function will return a string which will be the polynomial, but in the string form (2x^3-5x^1).



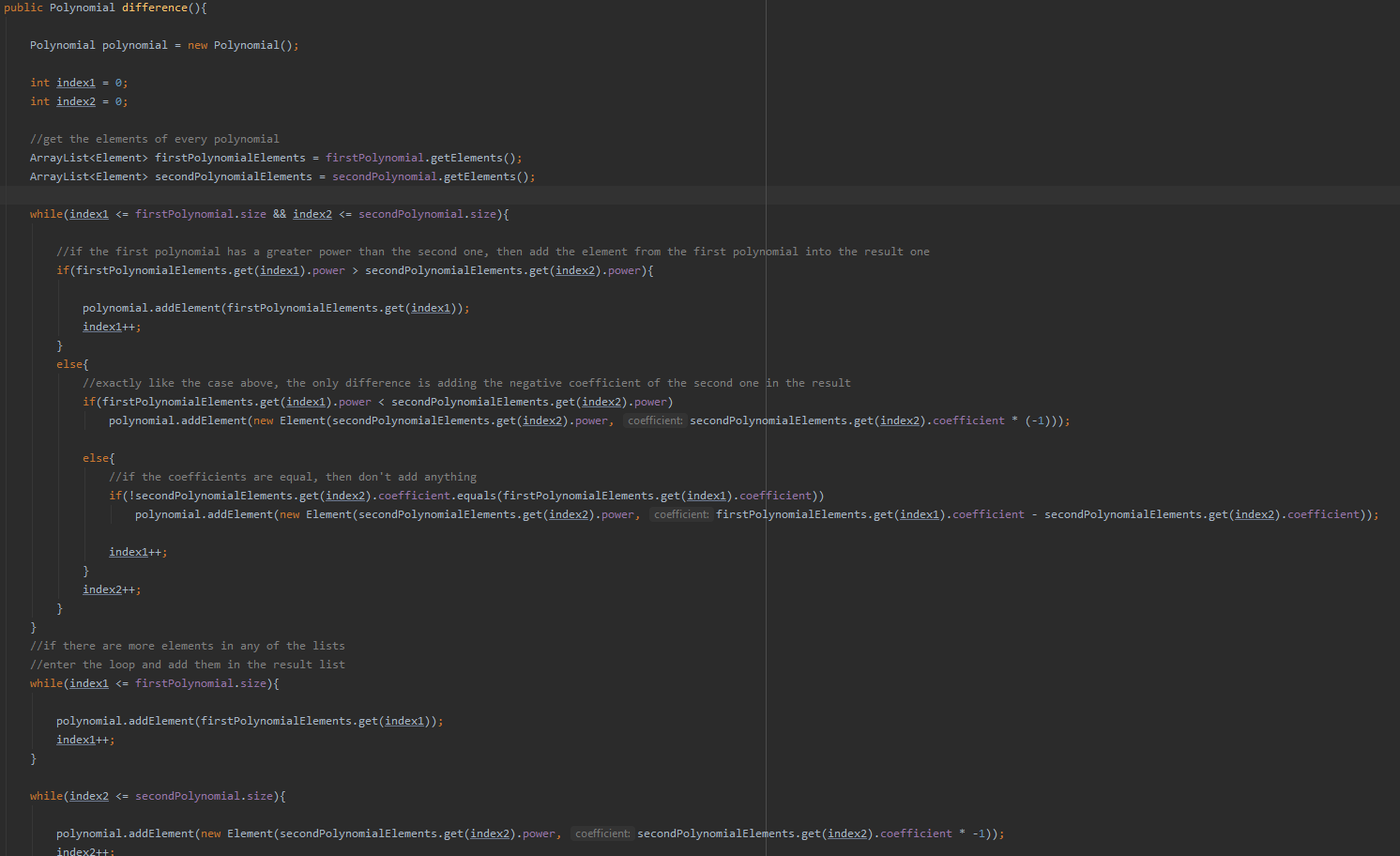
***Operations***

This class is pretty straight forward. This has 2 polynomials as attributes, and a constructor which takes two polynomials and setting the attributes that values. Let’s call the first polynomial “first” and the second one “second”. The operation will be performed in this manner: first (operation selected) second.

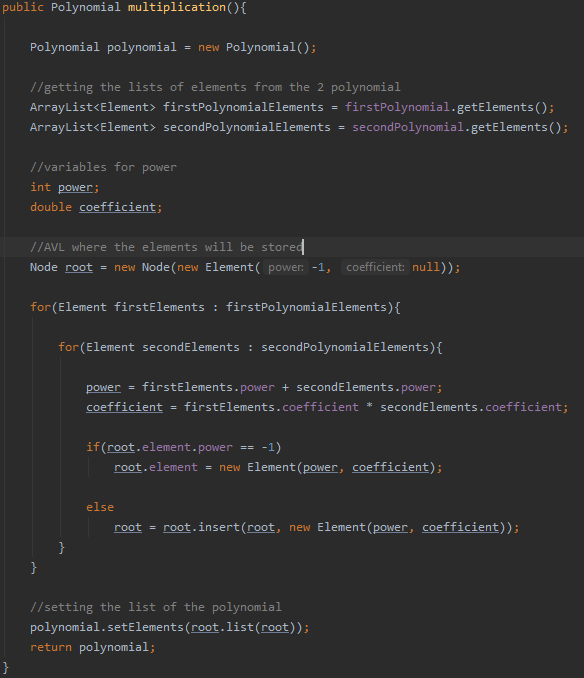
***Sum*** or addition is the function which performs the addition of the polynomials. It tries to go as much as possible in both polynomials: if the elements have the same power, then the program adds the coefficients, if the powers are different, then the new polynomial will insert the bigger power in it and move on. If any o the polynomials reached the maximum, then another while will be there to be sure that there are no more elements left. After all of this, the method will return the new polynomial created.



***Difference*** is exactly like an addition, but the coefficient will not be added, but will be subtracted (first - second).

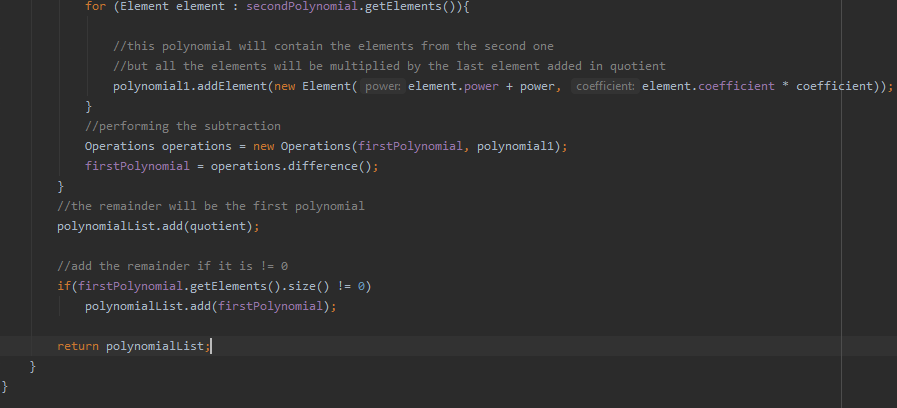


The ***multiplication*** is a little different, because the elements will be transferred into a Node class variable which will store them in an AVL, making the search more efficient. The algorithm has two for loops: the first one loops for all the elements in the first polynomial, and the second one in the second polynomial. Let us denote the actual element from the first with firstElement and secondElement for the second. The power will be the addition of the power: firstElement . power + secondElement . power and the coefficient will be the multiplication of the firstElement and the secondElement coefficients. The resulted polynomial will have as a list of elements the node . list().



The ***division*** has a lot in it. This function returns a list of polynomials: the first element of the list will be the remainder and the second one will be the quotient. The first if statement will possibly create an instant response for the output. If the first polynomial has a degree smaller than the second polynomial’s one, then the quotient will be null and the remainder will be the first polynomial. Otherwise, the calculations finally begin. While the first’s size is greater than 0 and the degree of the first is greater or equal with the second one then the power will be first . Index . power – second’s biggest power, and the coefficient is the division. The quotient will store this new element created using the power and the coefficient which were calculated earlier. The first polynomial will store now the difference between the first and the polynomial which has all the elements from the second polynomial, the only difference is that all the elements are multiplied by the new element. In the end, we add the quotient and the first polynomial in the list, first being the reminder.

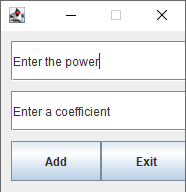


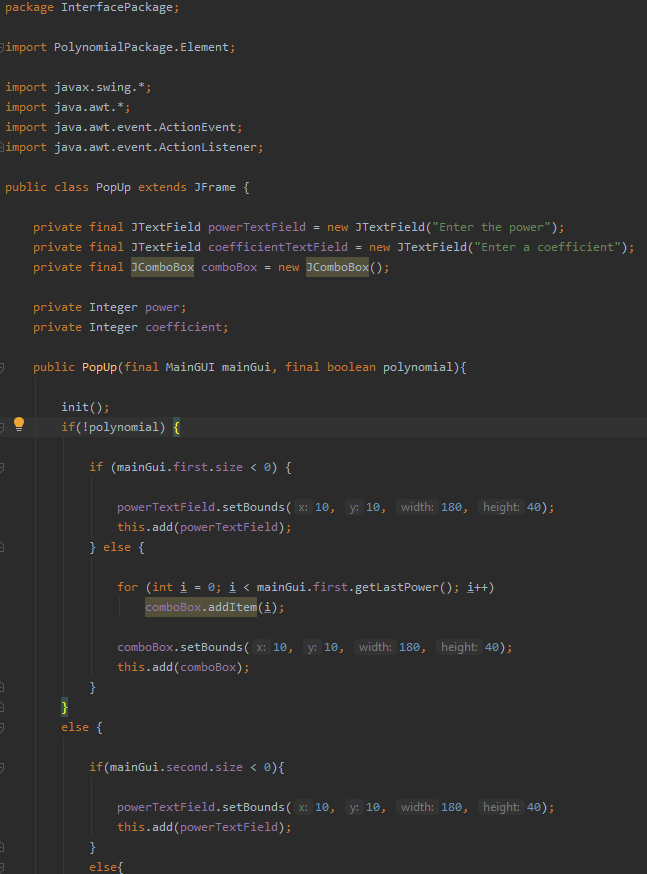


Interface Package contains two classes, one used for pop-ups and the other one used for GUI.

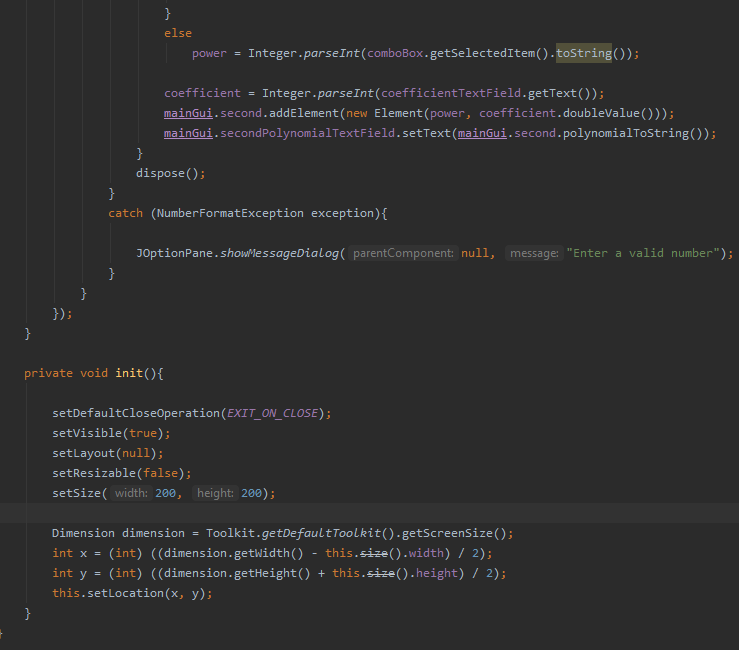
***PopUp*** class

Has 3 elements: combo box and 2 text fields. The box is used for all the elements added, except the first one. The class has a constructor which has as parameters a final MainGUI variable and a polynomial, where the element will be added. If this element is the first one, then the combo box does not show up, only a text field where the user introduced the degree of the polynomial. If not, the combo box will appear, having all the possible numbers from the degree to 0. There is a button for adding and one for exiting the pop up.







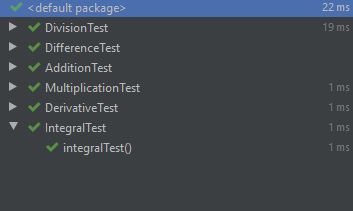


***MainGUI*** class

This class contains all the elements of the main interface of the program. There are 2 buttons for adding elements using pop up described above. The user has another option of adding the elements, writing them in a format (for example: 2x^3-5x^7+312x^4). There are two labels, which correspond to every polynomial. There are 8 buttons under the text fields, used for operations. The result will be shown in a new JOptionPane. There is a button in the left-down corner where you can reset the polynomials.

**Testing**

There are six tests created for every operation: addition, difference, multiplication, division, derivative and integral.



**Conclusion and Future improvements**

The making of this project made me remember basic OOP notions and let my mind think for making it efficient. For the improvements part, I think that the graphical interface can be better, but I really do not have that vision and creativity to create something spectacular 😊.

**Bibliography**

Most of the algorithms were created by me, there is an exception and I put it in the code as well as a comment. I used <https://www.geeksforgeeks.org/avl-tree-set-1-insertion/> for the AVL tree for a remainder and I also used stackoverflow in general for some questions.