Polynomial processing system

Name: Zavaczki Péter-Tibor
Group: 30423
Coordinating professor: Cristina Bianca Pop

1. Objective of the project

The presented project processes polynomials, as in it implements several operations that I can perform on polynomials in real life mathematics, such as addition or differentiation.

2. Analysis of the problem

The processed elements being polynomials, they have to be composed of several monomials. The program operates with the characteristics of these monomials basically (coefficient, power). There is a need for a Graphical User Interface, also called GUI.

3. Modeling

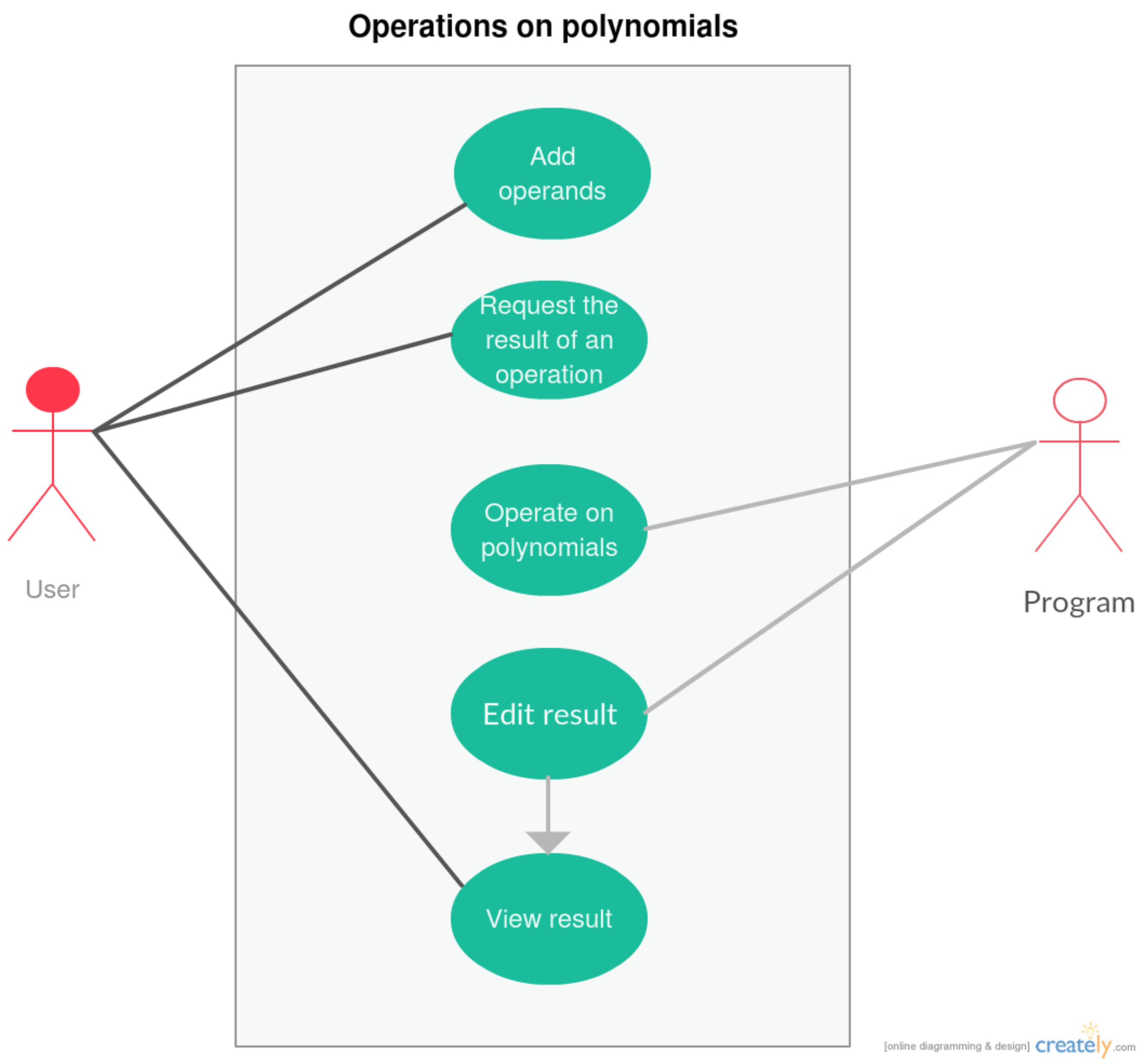
I have split the problem into several parts: creating the data structures, creating the operations that are to be used on the polynomials, creating the Graphical User Interface and processing the input string from the graphical user interface to be in the format with which I can work to create the polynomials, which is an array of integers for the coefficients and an integer which signals the highest power that we can find in the polynomial in discussion.

4. Scenarios

There are several cases of use of the given project. Since we have operations that work with two polynomials, this can be described as one case and for the operations that work with one polynomial being the other case, though we can input both polynomials if we work with one polynomial, but the program will only consider the first polynomial for the calculation of its result.

5. Use cases

In my point of view, there are two main components of the project's running mechanism: the user and the program itself, which access several functions of the project to reach a result from the user's input data.


Operations on polynomials
Add operands
Request the result of an operation
Operate on polynomials
Edit result
View result
User
Program
[online diagramming & design] creately.com

As we can see on the previous diagram, the user has access to several project elements, namely adding operands to be worked on, requesting the result of an operation with said operands and viewing the result of the requested operation, all possible via the Graphical User Interface. In contrast, the program has access to the elements which give us the result to the requested operation relative to the input operands. Besides the previously mentioned project element, the program can edit the result seen by the user through the Graphical User Interface.

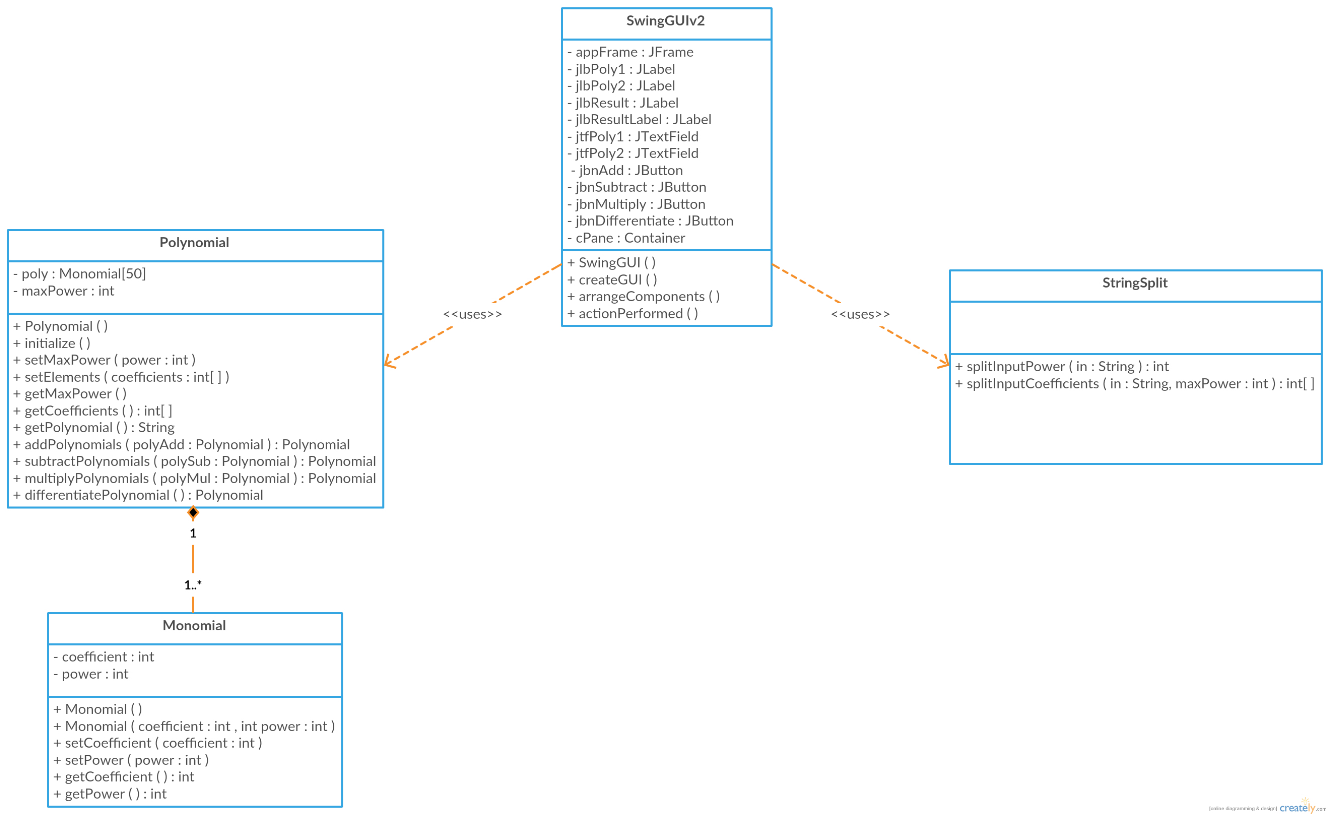
6. Program Design

6.1. Design Decisions

For my main structure I used objects of the class type Polynomial composed of several Monomial objects, among others (to be specified under the Data strucutres branch). The Monomial objects are composed of an integer coefficient field as well as an integer power field. Thus, we can say that a monomial, in string format would look some thing like the following structue: “<sign><integerCoefficient>x^<integerPower>”, ex.: “-10x^3” which would be $-10x^3$ in real life mathematics. Considering the format of the Monomial, we can affirm that a Polynomial in string format is a composition of its Monomial elements, going from its highest power Monomial to the Monomial with power 0, ex.: “-8x^4 +0x^3 +0x^2 +8x^1 -5x^0”.

The methods taking the tasks of operating on the Polynomial operands are somewhat reflective of real life operations with the structure of an operation being similar to the following: “<resultPolynomial> = <operandPolynomial1>.<operationName>(<operandPolynomial2>);”, ex.: “resultPolynomial = firstPolynomial.addPolynomials(secondPolynomial);”. As we can see, it resembles the real life style of “result = operand1 + operand2”, which in my opinion gives to the ease of use by somewhat advanced users who look through the code and eventually use it in developing further applications. To achieve this look for the methods, they had to be defined in the Polynomial class.

6.2. UML Class Diagram


SwingGUIv2
- appFrame : JFrame
- jlbPoly1 : JLabel
- jlbPoly2 : JLabel
- jlbResult : JLabel
- jlbResultLabel : JLabel
- jtfPoly1 : JTextField
- jtfPoly2 : JTextField
- jbnAdd : JButton
- jbnSubtract : JButton
- jbnMultiply : JButton
- jbnDifferentiate : JButton
- cPane : Container
+ SwingGUI ( )
+ createGUI ( )
+ arrangeComponents ( )
+ actionPerformed ( )
<<uses>>
<<uses>>
Polynomial
- poly : Monomial[50]
- maxPower : int
+ Polynomial ( )
+ initialize ( )
+ setMaxPower (power : int)
+ setElements (coefficients : int[ ])
+ getMaxPower ( )
+ getCoefficients ( ) : int[ ]
+ getPolynomial ( ) : String
+ addPolynomials (polyAdd : Polynomial) : Polynomial
+ subtractPolynomials (polySub : Polynomial) : Polynomial
+ multiplyPolynomials (polyMul : Polynomial) : Polynomial
+ differentiatePolynomial ( ) : Polynomial
StringSplit
+ splitInputPower (in : String) : int
+ splitInputCoefficients (in : String, maxPower : int) : int[ ]
1
1..*
Monomial
- coefficient : int
- power : int
+ Monomial ( )
+ Monomial (coefficient : int , int power : int)
+ setCoefficient (coefficient : int)
+ setPower (power : int)
+ getCoefficient ( ) : int
+ getPower ( ) : int
creately

6.3. Data Structures

The Monomial class models real life monomials, thus has an integer attribute called coefficient, having possible negative, positive and zero values and an attribute called power, which, similar to the coefficient attribute, has possible positive or zero values. Bear in mind that the program considers the power 0 as well as the power 1, they being inputted or outputted as “+0x^5” or “-10x^0” for example.

The Polynomial class has as an attribute an array of objects of the class type Monomial with the maximum number of elements of 50. That is, our first Polynomial operand as well as the second Polynomial operand and the result of the operations can be composed of at most 50 monomials. Bear in mind that with the multiplication of polynomials, the resulting Polynomial will have as maximum power, the sum of the maximum power of the first operand and of the maximum power of second operand minus one. The second attribute of the Polynomial class is an integer maxPower, which signals the highest power appearing among the array of Monomials, that is the maximum of the power fields in the Monomial array.

6.4. Class Design

- **Monomial**

The class Monomial stands at the base of the whole project, since the Polynomial type objects are composed of several Monomial type objects.

The class in discussion has two attributes: coefficient and power. The coefficient attribute stores the coefficient of said Monomial while the power attribute stores its power. The following example presents this: “<coefficient>x^<power>”, so if a Monomial mono1 has the attributes mono1.coefficient == 5 and mono1.power == 0, this would result in “+5x^0”.

In the class Monomial there are several methods implemented. The class in discussion has two constructors, one Monomial() without parameters, only to initialize the Monomial type objects and another Monomial(int coefficient, int power) which has two integer parameters: coefficient and power. The latter constructor initializes the object aswell as it sets the coefficient attribute of the Monomial object to the coefficient parameter’s value and power attribute of the Monomial object to the power parameter’s value. There are two methods to be used in case we chose to initialize the object with the constructor without parameters. The first method is setCoefficient(int coefficient) which sets the coefficient attribute of the Monomial type object to the value of the coefficient parameter. The second method is setPower(int power) which sets the power attribute of the Monomial type object to the value of the power parameter. There are two methods defined to retrieve the values in the coefficient and power attribute of the Monomial object. These are called getCoefficient() and getPower(). The getCoefficient method returns an integer value equal to the coefficient attribute of the Monomial object from which the method was called. The getPower method returns an integer value equal to the power attribute of the Monomial object from which the method was called.

- **Polynomial**

The class Polynomial contains most of the logic needed to power the project.

The class in discussion has two attributes: poly and maxPower. The poly attribute is an array composed of at most 50 Monomial type objects. This structure is necessary for the project and the class, since there is no polynomial without at least one monomial in the real world. On the account of the limit of 50 objects, we have to bear in mind that this applies to every such object, be it a result from a method or an input value. This is important to remember, since multiplication between polynomials results a polynomials with the maximum power being the sum of the two operand polynomials’ maximum powers minus one. The other attribute, called maxPower signals the highest value found between the power attributes of the Monomial type array. This attribute is important in the operations done with the polynomials, since the maximal power of the result is defined by the maximal powers of the operand polynomials.

In the class Polynomial there are a few methods for handling the future Polynomial type objects and also there are the methods which operate on the polynomials. There is only one constructor for this class called Polynomial(), its job being the initialization of the Polynomial type objects. There is a method called initialize() which instantiates the Monomial objects into the poly attribute's array.

The handling methods for the objects of type Polynomial are called setMaxPower(int power), setElements(int[] coefficients), getMaxPower(), getCoefficients() and getPolynomial(). The setMaxPower(int power) method takes the value from its parameter, power, and sets the maxPower attribute of the Polynomial object to the said parameter's value. The setElements(int[] coefficients) method takes the value from its parameter, the integer array coefficients, and sets the each of the poly Monomial array attribute's elements' coefficient attributeto the corresponding parameter's value, from the coefficients parameter's i-th position to the poly attribute's i-th position. The getMaxPower() method returns an integer value equal to the value found in the maxPower attribute of the Polynomial object at hand. The getCoefficients() method returns an integer array of all the coefficients ordered based on what power they are assigned to. For example if we have the Monomial "+5x^4", we would have in the array returned by the getCoefficients() method on the position 4 the value 5. The getPolynomial() method returns the whole polynomial as a String built by concatenating the following type of strings: " <coefficient sign><abstract value of coefficient>x^<power>".

The methods that describe operations on Polynomial type objects are called addPolynomials(Polynomial polyAdd), subtractPolynomials(Polynomial polySub), multiplyPolynomials(Polynomial polyMul), differentiatePolynomial(). The addPolynomials(Polynomial polyAdd) method returns a Polynomial type object as a result of the addition operation. The addition is effectuated between the current Polynomial object and the Polynomial object received through the polyAdd parameter as such: the coefficients are summed between each other if their corresponding powers are equal. Imposing this rule is possible via the array returned by the getCoefficients() method from the Polynomial objects, which returns an array with the appropriate coefficients appearing on the position that is equal to their corresponding power. If one polynomial has a higher power than the other, the addition is continued for the missing powers by adding a constant 0 to the coefficients of the remaining polynomial. For better visibility and closeness to reality, the method is written to ease its understanding by any user, reflecting real operations, so we would call it like polyResult = polyOperand1.addPolynomials(polyOperand2). In my opinion this eases the understanding because it looks like result = operand1 + operand2 which is the real life version of this operation. The subtractPolynomials(Polynomial polySub) method returns a Polynomial type object as a result of the subtraction operation. The subtraction is effectuated between the current Polynomial object and the Polynomial object received through the polySub parameter as such: the coefficients are subtracted between each other if their corresponding powers are equal. Imposing this rule is possible via the array returned by the getCoefficients() method from the Polynomial objects, which returns an array with the appropriate coefficients appearing on the position that is equal to their corresponding power. If one polynomial has a higher power than the other, the subtraction is continued for the missing powers by subtracting a constant 0 from the coefficients of the first Polynomial operand or, if the second operand is larger, subtracting the second Polynomial operand's remaining coefficients from 0. For better visibility and closeness to reality, the method is written to ease its understanding by any user, reflecting real operations, so we would call it like the following code part illustrates polyResult = polyOperand1.subtractPolynomials(polyOperand2). In my opinion this eases the understanding because it looks like result = operand1 - operand2 which is the real life version of this operation. The multiplyPolynomials(Polynomial polyMul) method returns a Polynomial type object as a result of the multiplication operation. The multiplication is effectuated between the current Polynomial object and the Polynomial object received through the polyMul parameter as such: the resulting Polynomial object is instantiated with the maxPower of the sum of the first Polynomial operand's maxPower and the second Polynomial operand's maxPower minus one. After this every coefficient from the first Polynomial operand is multiplied with every coefficient from the second Polynomial operand and the result of this multiplication is added to the current value of the coefficient at the power of the current power from the first operand plus the current power of the second operand. Imposing this rule is possible via the array returned by the getCoefficients() method from the Polynomial objects, which returns an array with the appropriate coefficients appearing on the position that is equal to their corresponding power. For better visibility and closeness to reality, the method is written to ease its understanding by any user, reflecting real operations, so we would call it like polyResult = polyOperand1.multiplyPolynomials(polyOperand2). In my opinion this eases the understanding because it looks like result = operand1 * operand2 which is the real life version of this operation. The differentiatePolynomial() method returns a Polynomial type object as a result of the differentiation operation. The differentiation is effectuated on the first Polynomial operand as such: passing though every element, we multiply the current coefficient with its element's power and put the resulting coefficient in the element with the power that is less with one that the current power, thus by the end of this pass through, by decreasing the maxPower of the current Polynomial type object by one, we get the differentiated Polynomial. Imposing this rule is possible via the array returned by the getCoefficients() method from the Polynomial objects, which returns an array with the appropriate coefficients appearing on the position that is equal to their corresponding power. For better visibility and closeness to reality, the method is written to ease its understanding by any user, reflecting real operations, so we would call it like polyResult = polyOperand1.differentiatePolynomial(). In my opinion this eases the understanding because it looks like result = operand1' which is the real life version of this operation.

- **StringSplit**

The class StringSplit contains the core of processing the input that the user gives to the program.

This class does not have any attributes, as it is not supposed to be instantiated in objects, its purpose is implementing the methods that process the user input.

The methods present in the class are splitInputPower(String in) and splitInputCoefficients(String in, int maxPower). The splitInputPower(String in) method returns an integer value equal to the highest power appearing in the parameter in of type String. Bear in mind that the in parameter should be composed of Monomial representations, such as "+5x^3 -2x^1 +7x^0" for example, in this case the splitInputPower method returns the value 3, as this is the highest power appearing in the String. The method does this job by splitting the String type parameter into a char type array and processing it character by character with respect to the characters before and after the current one. The splitInputCoefficients(String in, int maxPower) method returns an array of integers corresponding to the coefficients of the input string, with each coefficient being in its appropriate position, that is at the position of its power. In practice this would happen as follows: if we have the Monomial "+6x^9" among others, then in the output array we would have a 6 on the position 9. Also, if the input string does not specify every Monomial of the Polynomial, the program considers them as having coefficient 0, but they will still be put into the Polynomial object as valid Monomials. For example: if we have the input string "+4x^10 -3x^1 +0x^5 -0x^0" the returned array would have indices from 0 to 10 and the values would all be zeroes, except for the element at position 10, which gets the value 4 and the element at position 1, which gets the value -3.

- **SwingGUIv2**

Note: the class name has appended v2 to it because it is my second try to create the Graphical User Interface with the Swing package

The class SwingGUIv2 contains the main components to the Graphical User Interface, it creates the window, the text boxes and the buttons among others.

This class has the appFrame attribute of type JFrame, the jlbPoly1, jlbPoly2, jlbResult, jltResultLabel attributes of type JLabel, the jtfPoly1, jtfPoly2 attributes of type JTextField, the jbnAdd, jbnSubtract, jbnMultiply, jbnDifferentiate attributes of type JButton and the attribute cPane of type Container. The appFrame attribute represents the frame of the window we will receive when we create this Graphical User Interface, it is initialized with the name of the window passed as a String type parameter to the constructor. The jlbPoly1, jlbPoly2, jlbResult, jltResultLabel attributes represent non-modifiable text fields. The jlbPoly1 represents the label put next to the input box of the first operand. The jlbPoly2 represents the label put next to the input box of the second operand. The jlbResult represents the label put next to the output field of the result. The jlbResultLabel represents the label put into the field of the result to output the result into. The jtfPoly1 represents the text box in which the user inputs the first operand. The jtfPoly2 represents the text box in which the user inputs the second operand. The jbnAdd, jbnSubtract, jbnMultiply, jbnDifferetiate attributes represent the buttons of the operations. The jbnAdd represents the button of addition. If two valid polynomials are inputed into the text fields and this button is pressed, jlbResultLabel will output the resulting polynomial after the addition. The jbnSubtract represents the button of subtraction. If two valid polynomials are inputed into the text fields and this button is pressed, jlbResultLabel will output the resulting polynomial after the subtraction. The jbnMultiply represents the button of multiplication. If two valid polynomials are inputed into the text fields and this button is pressed, jlbResultLabel will output the resulting polynomial after the multiplication. The jbnDifferetiate represents the button of differentiation. If a valid polynomials are inputted into the first text field and this button is pressed, jlbResultLabel will output the resulting polynomial after the differentiation.

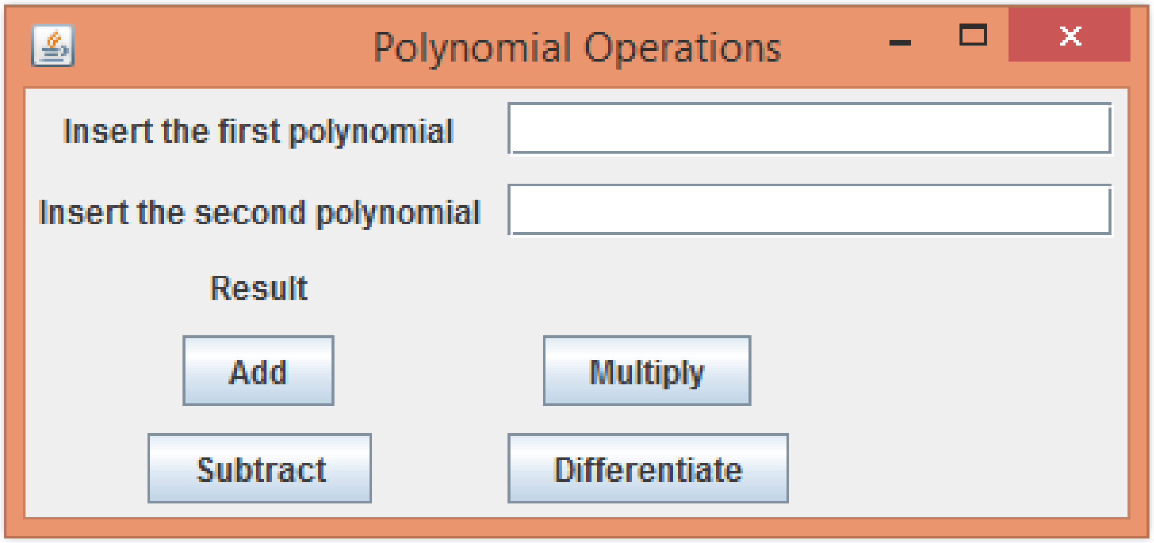
The methods in this class are its constructor, SwingGUIv2(), an initializer createGUI(), a layout modifier arrangeComponents(), and the method that makes the connection between elements from input to output, actionPerformed(ActionEvent e). The constructor SwingGUIv2() basically only helps at the instantiation of the Graphical User Interface. The createGUI() method creates the frame of the Graphical User Interface and sets the layout of the elements that will be in it. The arrangeComponents() method sets the position of the specific elements on the Graphical User Interface, sets some of the labels' values and adds action listeners to the buttons so that we know when one is pressed. The actionPerformed(ActionEvent e) method is creates three Polynomial type objects, to store the user input in and sets their value when a button is pressed, the result object is set accordingly to what button was pressed.

6.5. Relations

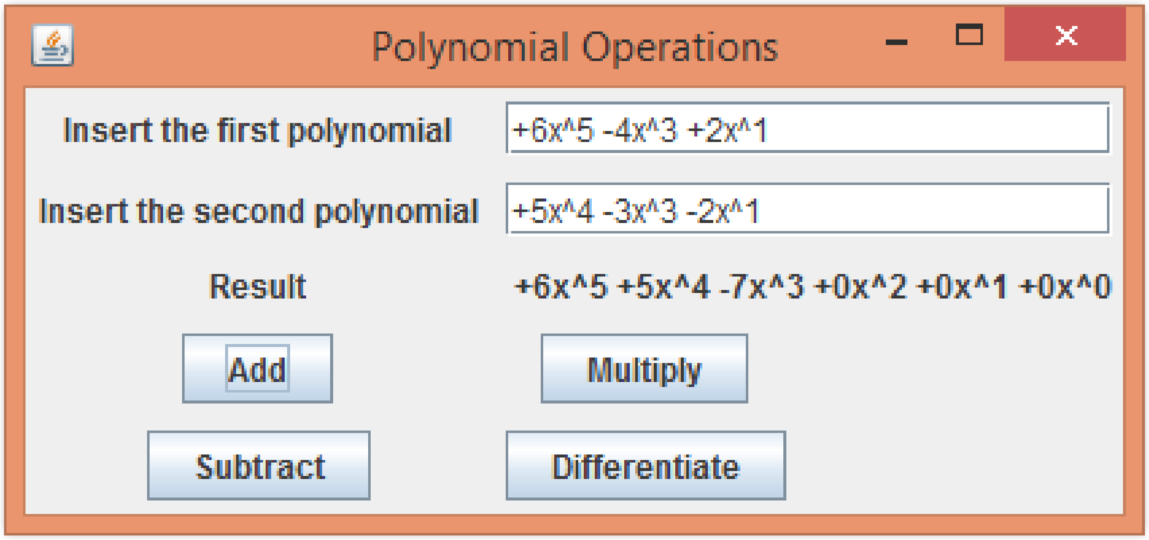
As we can see on the UML class diagram aswell, there is a composition relation between Monomial and Polynomial classes and a dependency relation between SwingGUIv2 and Polynomial classes, as well as between SwingGUIv2 and StringSplit classes.

6.6. Graphical User Interface

The Graphical User Interface is composed of labels, editable text boxes, and buttons.



As we can see, the result field is not initialized until we actually perform an operation on two valid polynomials.



```
+6x^5 -4x^3 +2x^1
+5x^4 -3x^3 -2x^1
+6x^5 +5x^4 -7x^3 +0x^2 +0x^1 +0x^0
```

Instead, as soon as we select an operation having valid parameters inputted, the result field appears, outputting all the terms in the polynomial for power 0 to the highest power appearing in the input string.

7. Implementation and testing

Almost every sepparate .java implementation file has another file associated to it, usually having the same name as the file it tests, with Test appended to its name. In these files every method is tested from the source file.

8. Results

Creating this project, among others, resulted for me in experience with working in Object Oriented Programming environments, and getting used to developing Object Oriented programs.

9. Conclusions

As a conclusion, I can say that there is still space for my personal improvement in the field, having poor preparation from the past.

10. Possible future development

Adding more functions to operate on polynomials, refining the output string, making the input string processing more flexible, better incapsulation can all be part of a future development.

11. Bibiliography

- https://creately.com/
- https://en.wikipedia.org/wiki/Class_diagram
- https://docs.oracle.com/javase/tutorial/uiswing/
- https://www.tutorialspoint.com/java/lang/index.htm