DOCUMENTATION

ASSIGNMENT 1

NUME STUDENT: Utiu Monica-Iulia
GRUPA: 30423

CONTENTS

1. Assignment's objectives

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation to be performed and view the result.

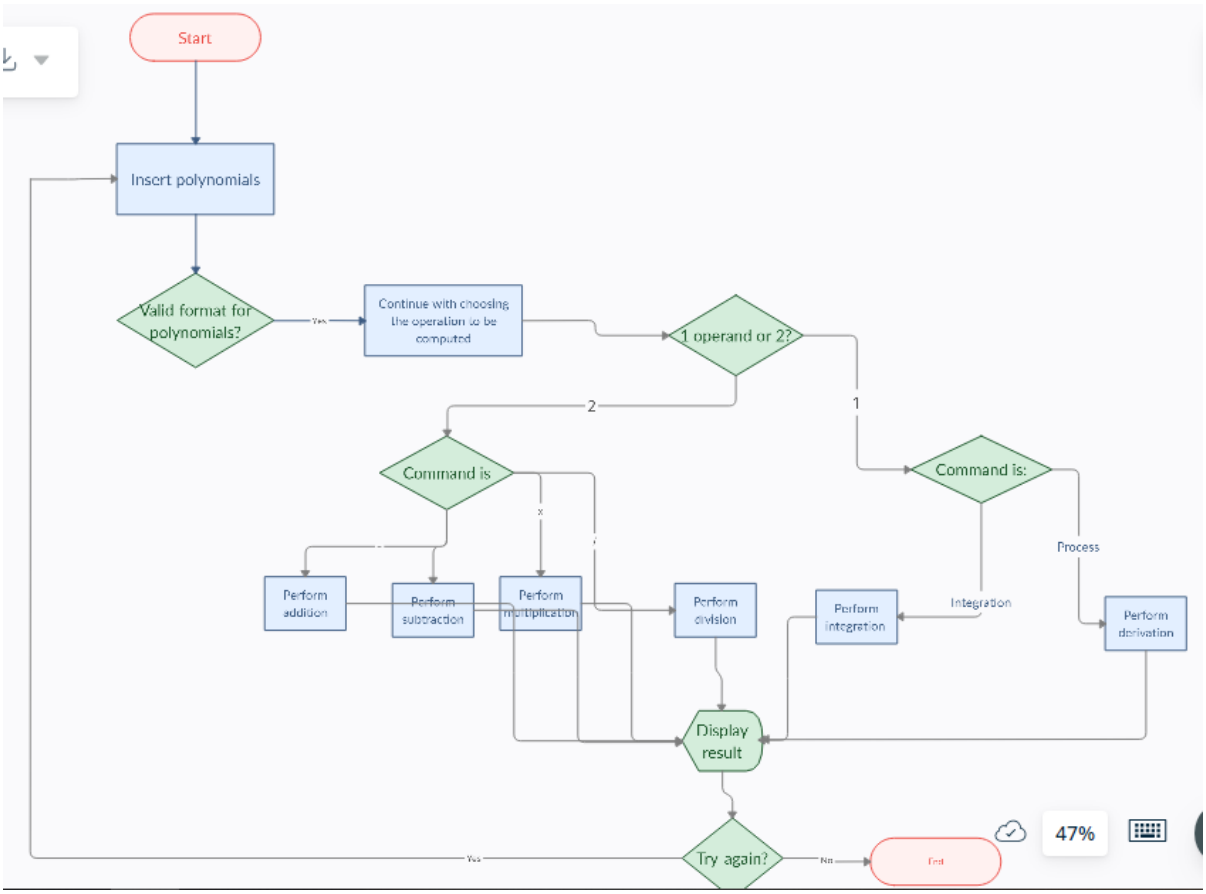
Sub-objectives:

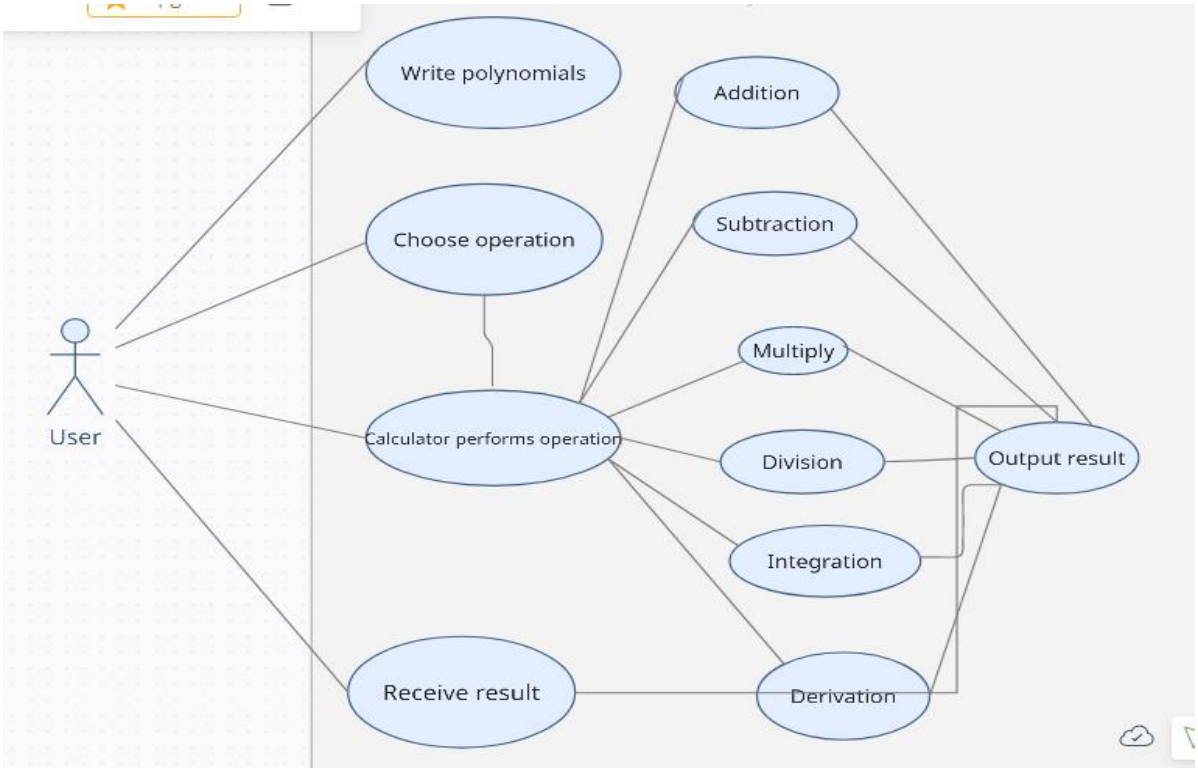
- Analyze the problem and identify requirements
- Design the polynomial calculator
- Implement the polynomial calculator
- Test the polynomial calculator

2. Problem analysis, modelling, scenarios, use-cases

The user should be able to insert polynomials, for this:
    - The application should provide text fields
    - Information about polynomials have to be extracted from those fields
    - Parsed in a form the calculator could use or signalize that they violate the format of polynomial that could be computed
- The user should be able to select the operation to be computed, for this:
    - The app should provide a way in which to choose the operation, for example, buttons with proper descriptors and should be able to distinguish which operation is chosen
    - The calculator should know how to compute that operation, so the calculator must implement: addition, subtraction, multiplication, division, derivation and integration
    - The calculator should output a correct result to the interface that the user interacts with, so the app should be able to represent it's internal computed data for the result, into a String or Object that could be represented on the screen in a n intuitive, known to the user way (a.k.a with the same format the user inputs)
- The user should be able to view the correct result, as presented above
- The user should be able to perform operations until it closes the program (that means adding a closing functionality and terminating the program when the user hits a button)
- All this should be easy for the user and intuitive to utilize

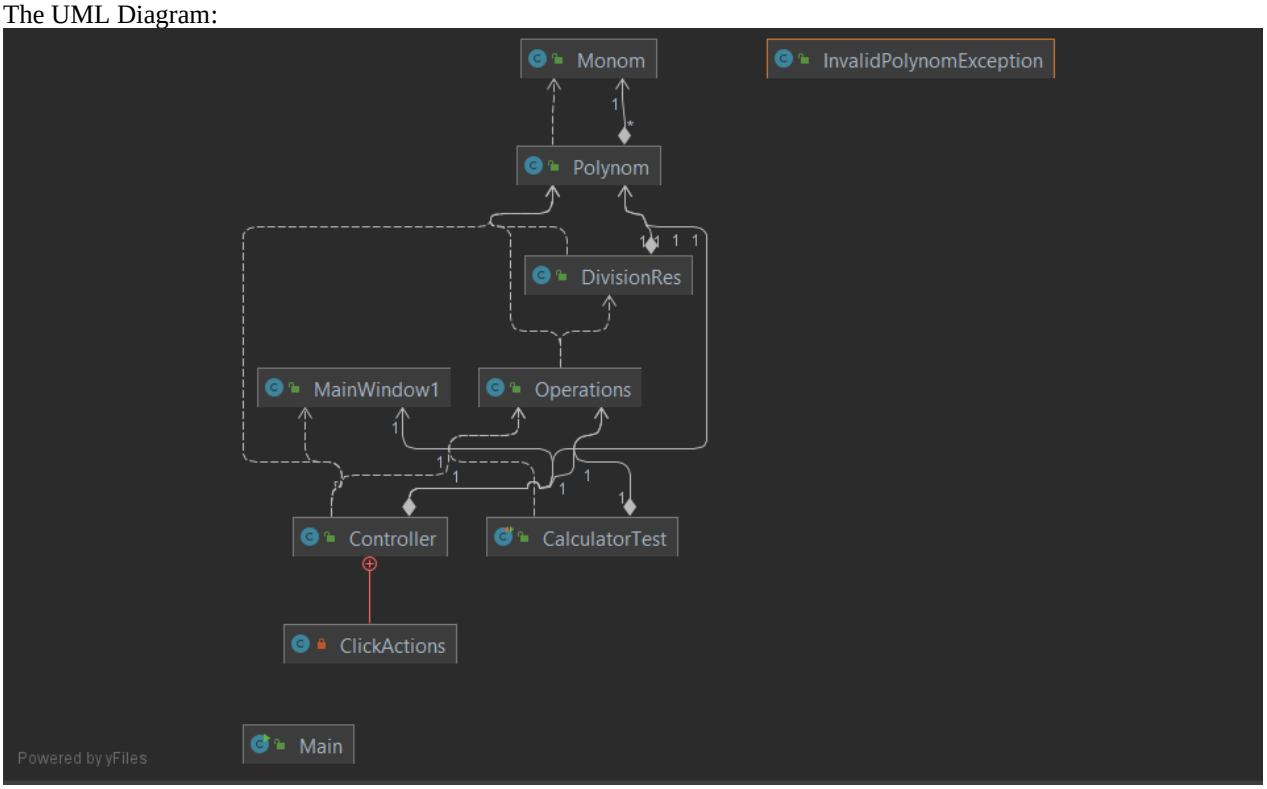
A flow chart diagram to showcase the use of the calculator and a use-case diagram:


Start
Insert polynomials
Valid format for polynomials?
Yes
Continue with choosing the operation to be computed
1 operand or 2?
2
1
Command is
Command is:
Process
Integration
Perform addition
Perform subtraction
Perform multiplication
Perform division
Perform integration
Perform derivation
Display result
Try again?
Yes
No
End
47%


Write polynomials
Addition
Choose operation
Subtraction
Multiply
User
Calculator performs operation
Division
Output result
Integration
Receive result
Derivation

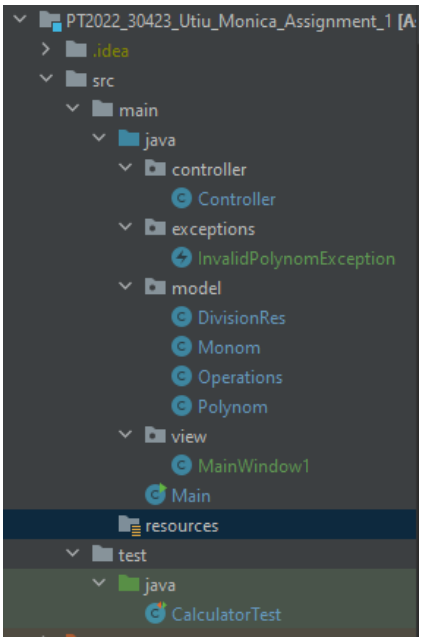
3. Design

The UML Diagram:



The packages and classes:

Respecting the Model-View-Controller structure, I have a controller package in which we have the class Controller, with a inner class, ClickActions, being in charge with interpreting the user input, computing what it's requested and then output it to the user, the model package where all our data modelled for our implementation is present, the classes Monom, Polynom, DivisionRes which is specifically designed for the output of the division operation and Operations which has the implementation of the operations and the view package, containing the UI implementation in the class MainWindow1.



I also have an exceptions package which contains the InvalidPolynomException thrown when an input from the user is invalid.

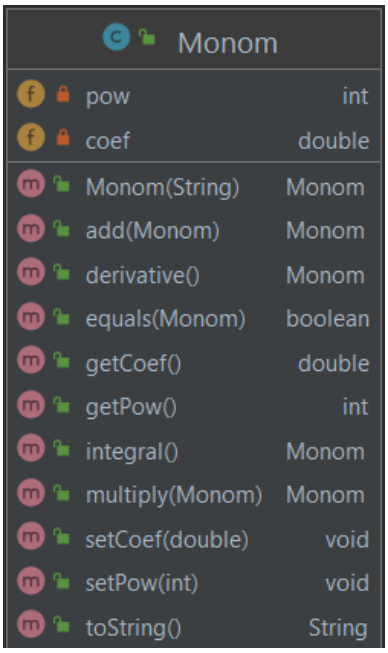
The Main class is in the basic java package, separated from the MVC.

Another important package is the test one, which contains the CalculatorTest class where all the operations are tested using JUnit.

- Data structures
  The Monom class contains two fields, one for the coefficient and one for the power, whilst the Polynom class has as it's only fields an ArrayList of monoms. The DivisionRes class also has two fields, one for the quotient polynom and one for the remainder polynom.
  The data structures are basic and simple and any complex structure needed can be builded with those for a cleaner and more persistent implementation.
- Algorithms used
  There aren't any special fundamental algorithms used, but, of course, mathematical derived algorithms for the implementation of the operations.

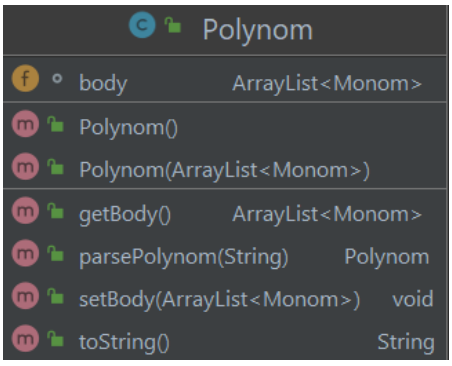
4. Implementation

- The Model package
  - Monom Class



The two private fields are "pow" and "coef", "pow" is meant to store the integer positive power of the monomial and "coef" is a double data type which stores the coefficient of the monomial that the user should input as an integer, but because of the division operation and integration operation, the double data type is more appropriate.
The class has a single constructor which receives the power and the coefficient to build the new monomial.
Getters and setters are present because of the access nature of the fields.
I also have some initial mathematical operation classes, first implemented for the monomial calculator from which I had progressed and are not later used in the program for the polynomial operations, but I kept them, maybe for further implementation uses or tweaks.
An important method to note is the Overridden one, toString(), useful for the output result and the equals() method.

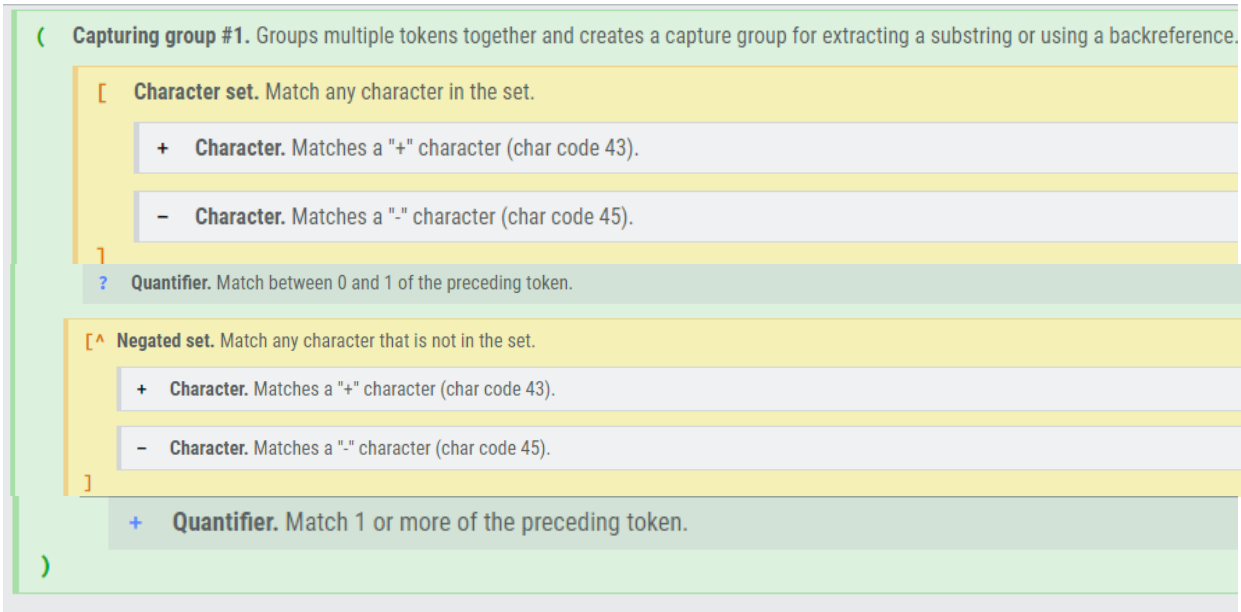
  - Polynom Class



This class has only one, private field, called "body" which is an ArrayList of Monomials, establishing an aggregation relation between the two classes (or a composition one since the polynomial can't exist without the monomials comprising it, at least in my implementation).
The basic constructor creates a 0 polynomial and the other constructor gives the polynomial a list of monomials that comprise it.
The methods in the class are the getter and setter of the fields, the "parsePolynom" method which receives a string (from the input text field that the user writes in) and parses it with the help of regex and pattern matching.

This parsing is done in two steps: the first one is the pattern matching of a monomial as a structure :

```
Pattern patternp = Pattern.compile("([+-]?[^+-]+)");
Matcher matcherp = patternp.matcher(polynom);
```

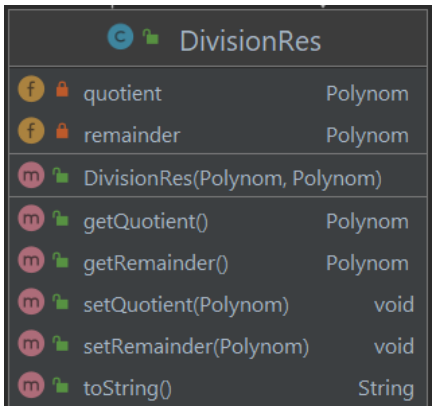


The second step being, matching the actual coeficient and power for every monomial:

```
while(matcherp.find()) {
    Pattern pattern = Pattern.compile("^([+-])?(([0-9]+)?([0-9]+\\.[0-
9]+)?)(X(\\^([0-9]+))?)?$");
    Matcher matcher = pattern.matcher(matcherp.group(1));
    int parsedCoef=1,parsedPow=1;
    if(!matcher.matches()) continue;
    String sign = matcher.group(1);
    String coef = matcher.group(2);
    String pow = matcher.group(7);
```

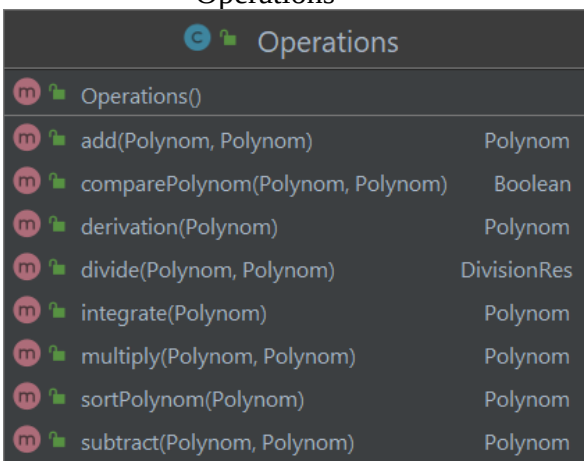
The Regex matching is longer for this one, but the outline of it is that in the first group ([+-]) we'll have the sign, that could be present or not, the second group (([0-9]+)?([0-9]+\.[0-9]+)?) will contain the coefficient of the monomial and the seventh group ([0-9]+) will contain the power. Also to be noted that in the 6-th group it is checked if 'X' appears.

- DivisionRes Class



This is a specific class, designed for the output of a division. The two private fields are polynomials representing the quotient and remainder of a division.
It has a constructor, getters and setters and a 'toString' method to output the two polynomials together in a string.

- Operations



This class does not have fields, but instead has methods that use one or two polynomials to perform, from a mathematical point of view, the computations of the operations needed for the calculator: add – for addition, subtract – for subtraction and so on.
One of the most mentionable methods could be the divide one because it's a bit more complex, but also the sortPolynom is important for better computation and display of polynomials.
There is also a compare method which compares the polynomials, checking only for the same power.

```
public DivisionRes divide(Polynom p1, Polynom p2) {
    Polynom quotient = new Polynom();
    if(p1.getBody().get(0).getPow() < p2.getBody().get(0).getPow()) {
        return null;
    }
    while(!p1.getBody().isEmpty() && (p1.getBody().get(0).getPow() >=
p2.getBody().get(0).getPow()) ) {
        int pow = p1.getBody().get(0).getPow() -
p2.getBody().get(0).getPow();
        double coef = p1.getBody().get(0).getCoef() /
p2.getBody().get(0).getCoef();
        Monom monom = new Monom(coef,pow);
        Polynom dev = new Polynom();
        quotient.getBody().add(monom);
        dev.getBody().add(monom);    // the temp result of division
        dev = multiply(dev,p2);      // multiply it with divisor
        p1 = subtract(p1,dev);       // subtract the result
    }
    quotient = sortPolynom(quotient);
    p1 = sortPolynom(p1);
    return new DivisionRes(quotient,p1);
}
```

Above is the implementation of division which mimics the on-paper way of dividing polynomials, resulting in quotient and remainder.

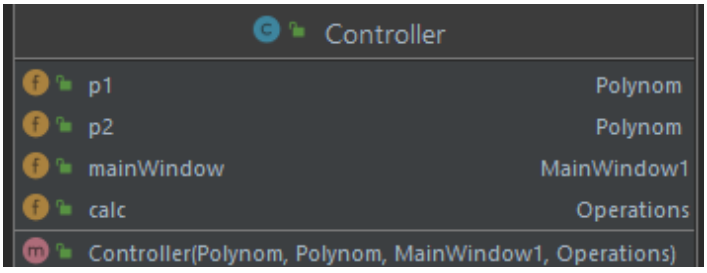
- The exceptions package
  - InvalidPolynomException Class

This Exception extends the Throwable Class and is used when the parsing is not successful to alert the user that the format of the polynomial inputed is invalid.

- The view package
  - MainWindow1 Class

This class is the View part of our MVC and contains multiple fields visible to the user: 6 buttons, 1 for each operation, 2 text fields for the input polynomials and a text area of the 2 rows for the output designed to also be able to output the division result, 3 labels for each text and the actual frame.
The class extends JPanel, opting for Java Swing for the UI. This implementation is a basic one, with a simple design. The panel has been set to a GridLayout for the positioning of the components.

- The controller package
  - Controller Class



The third part of our mechanism, the MVC, is the Controller which also has an inner class ClickAction in its constructor which implements the ActionListener interface. The class contains 4 fields, 2 polynomials also known as the operands, the mainWindow which is the View part and the calc from the Model. The constructor makes sure that everything is instantiated and adds action listeners to the buttons.
In the inner class, the actionPerformed is overridden and in it, the source of the event is checked for every button and when determined, the operation corresponding to the button source is called after the parsing of the polynomials.

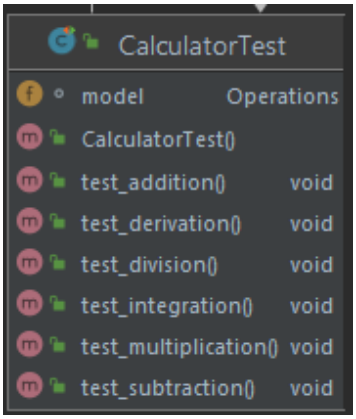
```
@Override
public void actionPerformed(ActionEvent e) {
```

```
    if(e.getSource() == mainWindow.ba ) {
        try {
            p1 = p1.parsePolynom(mainWindow.t1.getText());
            p1 = calc.sortPolynom(p1);
            p2 = p2.parsePolynom(mainWindow.t2.getText());
            p2 = calc.sortPolynom(p2);
            mainWindow.t3.setText(calc.add(p1,p2).toString());
        } catch (InvalidPolynomException ex) {
            mainWindow.showDialog(ex.getMessage(),
JOptionPane.ERROR_MESSAGE);
            ex.printStackTrace();
        }

    }
```

An example, detailing the steps if the button for the addition is the source of the event. The polynomials are parsed, sorted and then the text field should deliver the result of the operation implemented in the Model, called from the Controller, but triggered from the View by the user.

- Test package
  - CalculatorTest Class



This class, intended for the Junit has a field which references the Operations class that implemented the operations.
It contains a method for each operation to be tested.

The Main class is very restricted, only with a few declarations and a call to start the applications user interface.

```
Polynom p1 = new Polynom();
Polynom p2 = new Polynom();
MainWindow1 mainWindow = new MainWindow1();
Operations calculator = new Operations();
mainWindow.setVisible(true);
Controller controller = new Controller(p1,p2,mainWindow,calculator);
```

5. Results

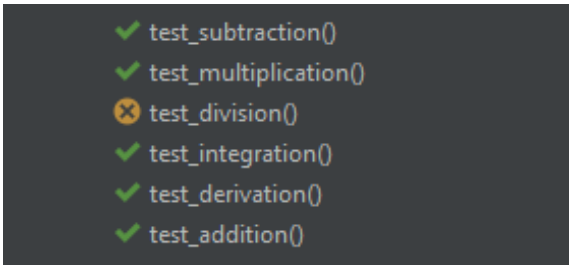
Se vor prezenta scenariile pentru testare cu Junit sau alt framework de testare.

Most of my tests use the same polynomials :
P1: X^2+7X
P2: 5X^3+3X+23

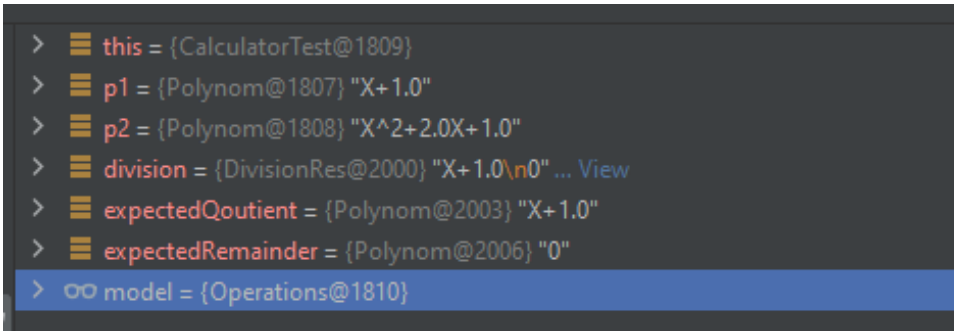
When using assertEquals(), the test seems to fail because of minor difference in how I declare a polynomial and instantiate it and how I output it as a string.

Expected :-5.0X^3+X^2+4.0X-23.0-23.0
Actual :-5.0X^3+X^2+4.0X-23.0-23.0

Results when using assertTrue() :



The division seems to not be done correctly, but when checking the variables when debugging this can be observed:



The division variable contains before the \n the quotient of the division and after, the remainder which are the same as the expectedQuotient and expectedRemainder, but the failure comes from the different format of the strings, having only little glitches.

Example of test:

```
@Test
public void test_addition() {
    Polynom p1 = new Polynom();
    p1.getBody().add(new Monom(1,2));
    p1.getBody().add(new Monom(7,1));

    Polynom p2 = new Polynom();
    p2.getBody().add(new Monom(3,1));
    p2.getBody().add(new Monom(5,3));
```

```
    p2.getBody().add(new Monom(23,0));

    Polynom sum = model.add(p1,p2);
    Polynom expected = new Polynom();

    expected.getBody().add(new Monom(5, 3));
    expected.getBody().add(new Monom(1, 2));
    expected.getBody().add(new Monom(10, 1));
    expected.getBody().add(new Monom(23, 0));
    assertTrue(model.comparePolynom(sum, expected));
   // assertEquals(sum,expected);
}
```

6. Conclusions

The calculator is a (almost fully) functionally polynomial calculator which can successfully perform addition, subtraction, multiplication, division, integration and derivation of one variable polynomials having integer coefficients and powers.

I have learned from this the basic MVC approach of implementing a GUI, the basis of GUI and a very simple implementation of a user interface and application that the user interacts with through events.

Further development should start with solving all the mini-inconsistencies in the program, then adapting the calculator to many-variable polynomials and other mathematical operations containing polynomials related to geometry for example. Computing points on a line represented by a 2-variable polynomial equation, relative-positioning to a line, to a plane and other special uses. I would've also liked a more creative, yet still clean aspect of the calculator, so an improvement should be a more attractive design for the calculator, making it even more intuitive for the user and I would have also loved to force the user to not be able to input something that would violate the format of the polynomial.

7. Bibliography

https://regexr.com/

https://www.codeproject.com/Articles/5256006/A-class-for-manipulating-Polynomials-and-Monomials

https://docs.oracle.com/javase/tutorial/uiswing/

https://www.geeksforgeeks.org/java-swing-simple-calculator/

https://www.mathway.com/Calculator/polynomial-division-calculator

https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

https://javamana.com/2022/02/202202131726089508.html

- My own code from my OOP classes